

Tobias Lars Hansen

# A NSGA-II Based Optimal Path Planner for Autonomous Docking of ROV Using Waypoint Guidance

Master's thesis in Marine Technology

Supervisor: Professor Martin Ludvigsen

June 2019



Tobias Lars Hansen

# **A NSGA-II Based Optimal Path Planner for Autonomous Docking of ROV Using Waypoint Guidance**

Master's thesis in Marine Technology  
Supervisor: Professor Martin Ludvigsen  
June 2019

Norwegian University of Science and Technology  
Faculty of Engineering  
Department of Marine Technology









## MASTER THESIS IN MARINE CYBERNETICS

SPRING 2019

FOR

**Tobias Lars Hansen,**  
**Student number: 759257**

A NSGA-II Based Optimal Path Planner for Autonomous Docking of ROV Using Waypoint Guidance

### **Work description (short description)**

Remotely operated vehicles (ROVs) are used in a wide range of underwater operations, such as intervention, maintenance, and repair (IMR) due to their large payload capabilities. Underwater operations can be streamlined in terms of saving cost, increasing efficiency, and maintaining safety by increasing the level of autonomy. Furthermore, the research area of path planning is essential for achieving persistent autonomy, and the non-dominated sorting genetic algorithm, NSGA-II likely will render a high-performance optimal path planner for autonomous docking missions of ROVs using waypoint guidance.

### **Scope of work**

1. Review necessary literature within the field of underwater operations utilizing ROVs, and relevant literature from similar fields, such as AUV.
2. Propose a mathematical model of the ROV and present relevant theory concerning guidance, navigation, and control.
3. Define the optimal path planning problem for autonomous docking missions.
4. Propose optimization theory behind the non-dominated sorting genetic algorithm, NSGA-II and develop an optimal path planner able to take on autonomous docking missions for the ROV Minerva 2 based on this algorithm.
5. Examine the performance of the NSGA-II based optimal path planner as a stand-alone application.
6. Implement the NSGA-II based optimal path planner into the control system of the ROV Minerva 2 to obtain an integrated system with the capabilities to take on autonomous docking missions.
7. Examine the performance of the integrated system performing autonomous docking missions in simulations.

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

The master thesis should be submitted within 11<sup>th</sup> June

Supervisor: Martin Ludvisen  
Co-Supervisor: Stein M. Nornes



# Summary

The contribution from this master's thesis is an optimal path planner for Remotely Operated Vehicles (ROVs) based on the Non-dominated Sorting Genetic Algorithm (NSGA-II). This contribution is a result of the exploration of a research hypothesis stating that path planning is an essential research area in terms for achieving persistent autonomy, and thus, persistent autonomy could be achieved by implementation of a high-performance optimal path planner based on NSGA-II in the control system. Beforehand to the establishment of the research hypothesis, a comprehensive literature review is carried out intended to provide an overview of different optimization techniques applied in path planning. The overall concluding remark from this literature review, is that NSGA-II is contained within the category of evolutionary algorithms that is well suited for path planning both in terms of computational complexity and optimization performance.

The overall motivation behind the work is to increase the level of autonomy in ROV operations. Likely, a higher level of autonomy streamlines such operations, that is, save costs, increase efficiency and accuracy, while maintaining safety. Less dependence on human intervention and the mother-ship causes cost savings. An advanced control architecture, including the path planner, increase efficiency and accuracy. Lastly, obtaining satisfactory performance of the integrated system, and indirectly, the reduced dependence of human intervention, maintain safety.

The path planner is developed and implemented into the control system of the ROV Minerva 2, constituting an integrated system enabling Minerva 2 to perform autonomous docking missions by trajectory of paths consisting of waypoints interconnected by straight line segments. The ROV Minerva 2 is an asset of the Applied Underwater Robotics Laboratory (AUR-lab) at the Norwegian University of Science and Technology (NTNU).

Furthermore, the thesis proposes a definition of the optimal path planning problem for autonomous docking missions. The problem definition takes into account parameters, for instance, the initial position of the ROV, the position of the docking station, obstacle positions of static spherically modelled obstacles and depth in the environment. The optimal path planning problem is defined as a multi-objective optimization problem, taking into account four objective functions responsible for assigning the generated paths four desired properties. The objective functions are to be minimized by applying NSGA-II, and thus, obtaining optimal paths. The desirable properties are short path length, sufficient safety margin to obstacles, absence of sharp turns in the horizontal plane, and sufficient depth, ensuring reliable altitude measurements by the Doppler Velocity Log (DVL).

The optimal path planner is developed by applying NSGA-II to the optimal path planning problem, generating a population of paths based on the objective functions. The optimization performance of the path planner is thoroughly verified in simulations of the path planner solving the optimal path planning problem as a stand-alone application. Due to the multi-objective nature of the path planning problem, optimization yields a set of Pareto-optimal paths. That is paths that are equally optimal and indistinguishable from each in terms of objective function values. Post-processing of the Pareto-optimal set of paths selects one path for the ROV trajectory in autonomous docking missions, done by a path selector algorithm. The path selector is based on additional user-defined preferences on the objectives.

The NSGA-II based optimal path planner is efficient in terms of computational complexity. By the big-O notation, NSGA-II has the complexity  $\mathcal{O}(MN^2)$ , where  $M$  is the number of objectives, and  $N$  is the number of paths in a population. Furthermore, the dimension of  $N$  depends on the number of waypoints in each path. Moreover, Minerva 2 utilizes constant jerk guidance for reference position, which is a waypoint based guidance scheme. Such waypoint based guidance schemes eliminate the need for further path refinement, and thus, save computational resources. Therefore, constant jerk guidance is directly compatible with the NSGA-II based optimal path planner.

The performance of the integrated system is verified in Hardware-In-the-Loop (HIL) simulations. The integrated system is obtained by implementing the NSGA-II based optimal path planner into the advanced control system of Minerva 2. In the integrated system, autonomous docking missions can be performed in two different modes, with different control strategies. The former mode, called auto-depth mode, is straight forward. It merely follows the path as it is, generated by the NSGA-II based optimal path planner. The latter mode, called auto-altitude mode, performs an online update of the depth-coordinate of the waypoints causing the ROV to follow the seafloor at the desired altitude by applying altitude control.

Both modes for autonomous docking missions render trajectories avoiding static obstacles, while also fulfilling the other objectives. However, the auto-altitude mode differs from auto-depth mode, as the former initially in the mission dives down to the desired altitude. Overall, both modes show satisfactory performance in simulations, even though the auto-altitude mode renders a slightly more oscillatory behaviour. This challenge considering the auto-altitude mode is assumed to be solvable by tuning of the altitude control scheme in the control system. The benefit of the auto-altitude mode is that, in this mode, the ROV takes on a fixed desired altitude such that reliable DVL measurements are available. This result eliminates the risk of bottom collision and ensures reliable state estimation. State estimation performed by the vehicle observer is known to deteriorate with the absence of reliable DVL measurements. Thus, the auto-altitude mode is recommended due to its benefits concerning safety when performing autonomous docking missions in complex scenarios where the seafloor is complex or data on the sea depth is partly lacking.

# Sammendrag

I denne avhandlingen presenteres en optimal baneplanlegger for fjernstyrte undervannsfartøy basert på den genetiske algoritmen NSGA-II. Avhandlingen er gjort på bakgrunn av en forskningshypotese som sier at baneplanlegging er et viktig forskningsområde med tanke på å oppnå vedvarende autonomi i undervannsoperasjoner, og videre at det vil være fordelaktig å implementere en baneplanlegger basert på NSGA-II i kontrollsystemet til fjernstyrte undervannsfartøy. Defineringsen av denne forskningshypotesen er basert på en omfattende litteraturstudie som er ment å gi en oversikt over ulike optimeringsteknikker brukt innenfor baneplanlegging.

Den overordnede motivasjonen bak denne oppgaven er å øke graden av autonomi i operasjoner med fjernstyrte undervannsfartøy. Dette er på bakgrunn av at høyere grad av autonomi vil føre til lavere kostnader, økt effektivitet og nøyaktighet, samt opprettholde sikkerheten i slike operasjoner. Mindre avhengighet av menneskelig innblanding vil føre til reduserte kostnader. En avansert kontrollarkitektur, som inkluderer en baneplanlegger, vil føre til økt effektivitet og nøyaktighet. Og ikke minst, tilfredsstillende ytelse av det integrete systemet vil ivareta sikkerheten.

Baneplanleggeren er utviklet og implementert i kontrollsystemet til det fjernstyrte undervannsfartøyet Minerva 2, slik at fartøyet er i stand til å utføre autonome dokkingoperasjoner ved å følge baner som består av banepunkter satt sammen av rette linjer. Minerva 2 er et fartøy som eies av Applied Underwater Robotics Laboratory (AUR-lab), et forskningssenter for undervannsrobotikk ved Norges Tekniske-Naturvitenskapelige Universitet (NTNU).

Masteroppgaven tilbyr en definisjon av det optimale baneplanleggingsproblemet for autonome dokkingoperasjoner. Problemdefinisjonen benytter inputtparametre som startposisjon av det fjernstyrte fartøyet, posisjonen til dokkingstasjonen, posisjoner til statiske hindringer som er modellert som kuler og sjødybde i operasjonsmiljøet. Det optimale baneplanleggingsproblemet er definert som et optimeringsproblem basert på fire kostfunksjoner. Kostfunksjonene er ansvarlige for å tilegne banene som genereres av baneplanleggeren ønskede egenskaper, og minimeres ved bruk av NSGA-II. De ønskede egenskapene er kort banelengde, tilstrekkelig sikkerhetsmargin til hindringer, fravær av skare svingninger i det horisontale plan, samt tilstrekkelig dybde for å måle høyden over havbunnen ved hjelp av en Doppler Velcity Log (DVL).

Baneplanleggeren er utviklet ved å løse baneplanleggingsproblemet ved hjelp av NSGA-II, som genererer en populasjon av optimale baner basert på kostfunksjonene. Ytelsen til baneplanleggeren som en frittstående applikasjon utenfor det integrerte systemet er verdifisert gjennom simuleringer. Siden baneplanleggingsproblemet har flere kostfunksjoner, gir optimeringen et sett med Pareto-optimale baner. Det betyr baner som er like optimale og ikke kan skilles fra hver andre med tanke på kostfunksjonsverdier. Etter optimeringen velges en av banene fra det Pareto-optimale settet med baner av en banevelger-algoritme. Banevelgeren er basert på ytterligere brukerdefinerte preferanser på kostfunksjonsverdier.

Den NSGA-II-baserte optimale baneplanleggeren er effektiv med tanke på beregningsmessig kompleksitet. NSGA-II har en beregningsmessig kompleksitet på  $\mathcal{O}(MN^2)$ , der  $M$  er antall kostfunksjoner og  $N$  er antall baner i populasjonen. Dessuten avenger dimensjonen til  $N$  av antall banepunkter i hver bane.

Ytelsen til det integrerte systemet er verdifisert gjennom Hardware-In-The-Loop (HIL) simuleringer. Det integrerte systemet består av implementeringen av baneplanleggeren i det avanserte kontrollsystemet til Minerva 2. I det integrerte systemet, kan autonome dokkingoperasjoner gjennomføres i to forskjellige moduser med forskjellige styringsstrategier. Det første moduset kalles auto-depth mode, og er ganske rett fram. Minerva 2 følger ganske enkelt den banen som genereres av baneplanleggeren slik den er. I det andre moduset, som kalles auto-altitude mode, gjøres en online oppdatering av dybdekoordinatene til den genererte banen slik at ROVen følger havbunnen på en konstant ønsket høyde over havbunnen ved hjelp av høydekontroll.

Begge modusene for autonome dokkingoperasjoner gir banefølgning som unngår kollisjoner med statiske hingringer, og samtidig oppfyller de andre ønskede egenskapene til en optimal bane. Det er imidlertid forskjell på auto-depth mode og auto-altitude mode, ettersom i sistnevnte dykker Minerva 2 ned til ønsket høyde over havbunnen før forflytningen i det horisontale plan starter. Stort sett er ytelsene god i begge moduser, men det er tendenser til mer oscillatorisk oppførsel i auto-altitude mode. Dette er en utfordring som antas å være løselig ved å endre på parameterinstillingene for høydekontroll i kontrollsystemet. Fordelen med auto-altitude mode er imidlertid at på grunn av den konstante høyden over havbunnen, garanteres pålitelige DVL-målinger. Dette fører til at risikoen for kollisjon med havbunnen elimineres og fører samtidig til pålitelig ytelse av observeren. Ytelsen til observerern viser seg å forfalle dersom DVL-målingene er fraværende eller dårlige. På bakgrunn av disse fordelene er auto-altitude mode anbefalt for autonome dokkingoperasjoner der havbunnen er kompleks eller kunnskap om havdybden er delvis fraværende.

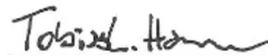
# Preface

This report is written based on the work of my master's thesis marine cybernetics during the spring of 2019 at the Norwegian University of Science and Technology. The master thesis is based on the ROV Minerva 2 and is a continuation from the project thesis performed during the fall of 2018, giving insight into the control system of Minerva 2.

The main topics of the master's thesis are the design and development of an optimal path planner for autonomous docking missions for ROVs using waypoint guidance and the implementation of the path planner into the control system of Minerva 2, achieving an integrated system able to take on autonomous docking missions.

It is assumed that the reader of this thesis retains basic knowledge within marine engineering and control systems.

Trondheim, June 2019.



---

Tobias Lars Hansen





# Acknowledgements

I have put great effort behind this master's thesis and also gained a lot of knowledge along the way. In order to achieve this, I have received great help and assistance. Firstly, I would like to thank my supervisor, professor Martin Ludvigsen. I am thankful for his efforts, as he has provided me with feedback on my progress, given me ideas for improvement and motivating challenges. Furthermore, I would like to thank postdoctoral researcher Stein Nornes, for his efforts on providing me answers on technical challenges I have had during the work.



# Contents

<b>Summary</b>	<b>v</b>
<b>Sammendrag</b>	<b>vii</b>
<b>Preface</b>	<b>viii</b>
<b>Acknowledgements</b>	<b>x</b>
<b>Abbreviations</b>	<b>xx</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.1.1 Remotely Operated Vehicles (ROVs) . . . . .	1
1.1.2 Autonomy . . . . .	2
1.2 Motivation . . . . .	3
1.3 Research Topic and Objectives . . . . .	4
1.4 Scope and Limitations . . . . .	4
1.5 Literature Review . . . . .	5
1.5.1 ROV Control System Development . . . . .	5
1.5.2 Strategies for Homing/Docking Applied for AUVs . . . . .	9
1.5.3 Optimization Techniques for Path Planning . . . . .	10
1.6 Structure of Thesis . . . . .	14
1.6.1 Appendices . . . . .	14
1.7 Thesis Contribution . . . . .	15
<b>2 Mathematical Modelling of ROVs</b>	<b>17</b>
2.1 Notation and Reference Frames . . . . .	17
2.2 Kinematics . . . . .	18
2.3 Kinetics . . . . .	19
2.3.1 Rigid-Body Kinetics . . . . .	20
2.3.2 Hydrostatic Terms . . . . .	21
2.3.3 Hydrodynamic Terms . . . . .	22
<b>3 ROV Control, Guidance and Navigation</b>	<b>23</b>
3.1 ROV Minerva 2 . . . . .	23
3.1.1 Acoustic Navigation Specifications . . . . .	23
3.2 Control System Overview . . . . .	25
3.3 Controller . . . . .	26
3.4 Observer . . . . .	26
3.5 Guidance . . . . .	27
3.5.1 Constant Jerk Guidance for Position Reference . . . . .	28
3.6 Altitude Control . . . . .	29
3.7 Optimal Path Planning . . . . .	31
3.7.1 Objective Function - Path Length . . . . .	32

3.7.2	Objective Function - Safety Margin . . . . .	32
3.7.3	Objective Function - Avoid Sharp Turns in the xy-plane . . . . .	33
3.7.4	Objective Function - Ensuring Reliable DVL Measurements . . . . .	34
3.7.5	Constraint - Ensuring Safe Docking . . . . .	35
3.7.6	Path Selector . . . . .	36
3.7.7	The Path Planning Problem . . . . .	37
<b>4</b>	<b>Optimization Theory</b>	<b>39</b>
4.1	Non-Dominated Sorting Genetic Algorithm (NSGA-II) . . . . .	39
4.1.1	Fast Non-Dominated Sorting Approach . . . . .	39
4.1.2	Crowding Distance Assignment . . . . .	41
4.1.3	Selection . . . . .	43
4.1.4	Simulated Binary Crossover (SBX) . . . . .	44
4.1.5	Polynomial Mutation . . . . .	45
4.1.6	Computational Complexity . . . . .	46
4.1.7	NSGA-II Parameters . . . . .	46
<b>5</b>	<b>Method</b>	<b>47</b>
5.1	Optimal NSGA-II Based Path Planner . . . . .	47
5.2	LabVIEW based ROV Motion Control System . . . . .	49
5.2.1	Software and Hardware Platforms . . . . .	50
5.2.2	Hardware-in-the-Loop (HIL) Simulations . . . . .	50
5.3	Frigg VI - Graphical User Interface . . . . .	50
5.4	Njord VI - ROV Motion Control System . . . . .	51
5.5	Verdandi VI - Hardware-In-The Loop (HIL) Simulation . . . . .	52
5.6	Integrated Autonomous System . . . . .	52
5.6.1	Auto-Depth Mode . . . . .	54
5.6.2	Auto-Altitude Mode . . . . .	54
<b>6</b>	<b>Results</b>	<b>57</b>
6.1	Optimization . . . . .	57
6.1.1	Path Planner with Three Objectives, $O_1$ , $O_2$ and $O_3$ . . . . .	58
6.1.2	Path Planner with Four Objectives, $O_1$ , $O_2$ , $O_3$ and $O_4$ . . . . .	63
6.2	Integrated System . . . . .	68
6.2.1	Auto-Depth Mode . . . . .	71
6.2.2	Auto-Altitude Mode . . . . .	75
6.3	Extended Case: Curved Seafloor . . . . .	78
<b>7</b>	<b>Discussion</b>	<b>83</b>
7.1	Optimization . . . . .	83
7.1.1	Path Planner with Three Objectives, $O_1$ , $O_2$ and $O_3$ . . . . .	84
7.1.2	Path Planner with Four Objectives, $O_1$ , $O_2$ , $O_3$ and $O_4$ . . . . .	85
7.1.3	In General on the NSGA-II Based Optimal Path Planner . . . . .	87
7.2	Integrated System . . . . .	87
7.2.1	Path Generation in the Integrated System . . . . .	87
7.2.2	Comparison of Auto-Depth Mode and Auto-Altitude Mode . . . . .	88
7.2.3	Extended Case . . . . .	89
<b>8</b>	<b>Conclusions</b>	<b>91</b>
8.1	Conclusions . . . . .	91
8.2	Further work . . . . .	92
	<b>Bibliography</b>	<b>93</b>
	<b>Appendices</b>	<b>A1</b>
A	Matlab-code . . . . .	A1
A1	Main.m . . . . .	A1

A2	parameters.m . . . . .	A4
A3	OFE_multiple_obstacles.m . . . . .	A6
A4	initialize_population.m . . . . .	A9
A5	NDS_CDA.m . . . . .	A11
A6	BTS.m . . . . .	A16
A7	make_new_pop.m . . . . .	A18
A8	selection.m . . . . .	A20
A9	path_selector.m . . . . .	A21
A10	plot_tool.m . . . . .	A22



# List of Figures

1.1	Illustration of the ROV Minerva 2 . . . . .	2
1.2	Control Architecture Layers [1] . . . . .	7
1.3	Generalization of an Autonomous Subsea Operation [2] . . . . .	8
1.4	Case study results in static current environment with obstacles, illustrations by [3] . . . . .	12
2.1	The NED- and BODY-frame relative to the Earth-centered Earth-fixed (ECEF) frame[4] . . . . .	18
3.1	SSBL Underwater Positioning[5] . . . . .	24
3.2	ROV Motion Control System Architecture [6] . . . . .	25
3.3	Correlation between jerk, acceleration, velocity and position [7] . . . . .	29
3.4	Linear seafloor approximation using DVL [7] . . . . .	30
3.5	Illustration of the closes distance between an obstacle and any path segment . . . . .	33
3.6	Illustration of the Change of Heading in the xy-plane . . . . .	34
3.7	Illustration of Objective Function Evaluating Placement of Waypoint z-coordinates . . . . .	35
3.8	Illustration of Terminal Phase of the Docking Operation . . . . .	36
4.1	Graphical explanation of non-dominated sorting . . . . .	40
4.2	Crowding distance calculation with two objective functions, $f_1$ and $f_2$ . . . . .	42
4.3	Selection in the $i$ -th iteration of NSGA-II[8] . . . . .	43
4.4	SBX . . . . .	44
5.1	Flowchart of the NSGA-II based path planner . . . . .	49
5.2	Frigg VI . . . . .	51
5.3	Njord VI . . . . .	51
5.4	Verdandi VI . . . . .	52
5.5	Autonomy window with optimal path planning capabilities . . . . .	53
5.6	Zoomed path planning module graphical user interface . . . . .	53
6.1	Selected path depicted in 3D . . . . .	59
6.2	Selected path in the NE-plane . . . . .	59
6.3	Selected path in the ED-plane . . . . .	60
6.4	3D plot of objective function values in the final population . . . . .	61
6.5	2D plot of objective function values $O_1$ versus $O_2$ in the final population . . . . .	61
6.6	2D plot of objective function values $O_1$ versus $O_3$ in the final population . . . . .	62
6.7	2D plot of objective function values $O_2$ versus $O_3$ in the final population . . . . .	62
6.8	Selected path from four-objective optimal path planner . . . . .	64
6.9	Selected path in the NE-plane . . . . .	64
6.10	Selected path in the ED-plane . . . . .	65
6.11	3D plot of objective function values $O_1, O_2, O_3$ in the final population . . . . .	66
6.12	3D plot of objective function values $O_1, O_2, O_4$ in the final population . . . . .	66
6.13	3D plot of objective function values $O_1, O_3, O_4$ in the final population . . . . .	67
6.14	3D plot of objective function values $O_2, O_3, O_4$ in the final population . . . . .	67
6.15	3D plot of the selectedpath for simulations in the integrated system . . . . .	69
6.16	Selected path projected in the NE-plane . . . . .	69

6.17	Selected path projected in the ED-plane . . . . .	70
6.18	3D ROV trajectory in auto-depth mode . . . . .	71
6.19	2D ROV trajectory in the NE-plane in auto-depth mode . . . . .	72
6.20	Time evolution of NED-coordinates in auto-depth mode . . . . .	73
6.21	Time evolution of thruster %-rpm in auto-depth mode . . . . .	74
6.22	Time evolution of ROV altitude in auto-depth mode . . . . .	74
6.23	3D ROV trajectory in auto-altitude mode . . . . .	75
6.24	2D ROV trajectory in the NE-plane in auto-altitude mode . . . . .	76
6.25	Time evolution of NED-coordinates in auto-altitude mode . . . . .	77
6.26	Time evolution of thruster %-rpm in auto-altitude mode . . . . .	78
6.27	Time evolution of ROV altitude in auto-altitude mode . . . . .	78
6.28	3D plot of generated path for extended case . . . . .	80
6.29	3D ROV trajectory in the extended case . . . . .	81
6.30	3D ROV trajectory in the extended case . . . . .	81
6.31	Time-evolution of ROV altitude in the extended case . . . . .	82
6.32	DVL measurements and estimates from extended case simulation . . . . .	82



# List of Tables

1.1	Optimal Travel Time Comparison [3] . . . . .	12
1.2	Comparison of Optimization Algorithms for Path Planning [9], [3] . . . . .	13
2.1	SNAME 1950 Notation . . . . .	17
3.1	Available measurements based on sensors . . . . .	25
3.2	User defined path planning parameters . . . . .	37
4.1	NSGA-II user defined parameters . . . . .	46
6.1	NSGA-II parameters . . . . .	57
6.2	General path planning problem parameters . . . . .	58
6.3	Additional path planning problem parameters . . . . .	58
6.4	Path selector parameters . . . . .	58
6.5	Properties of the selected path ( $\mathcal{P}_{\text{selected}}$ ), $M = 3$ . . . . .	60
6.6	Final population, $P$ , numeric properties, $M = 3$ . . . . .	63
6.7	Additional path planning problem parameters . . . . .	63
6.8	Path selector parameters . . . . .	63
6.9	Properties of the selected path, $\mathcal{P}_{\text{selected}}$ , $M = 4$ . . . . .	65
6.10	Final population, $P$ , numeric properties, $M = 4$ . . . . .	68
6.11	General path planning problem parameters for the integrated system . . . . .	68
6.12	Properties of the selected path, $\mathcal{P}_{\text{selected}}$ , auto-altitude mode . . . . .	70
6.13	Waypoints of $\mathcal{P}_{\text{selected}}$ for the integrated system . . . . .	70
6.14	Additional auto-altitude path planning problem parameters . . . . .	75
6.15	Verdandi seafloorcoefficients . . . . .	79
6.16	Path planning problem parameters for extended case . . . . .	79
6.17	Properties of the selected path, $\mathcal{P}_{\text{selected}}$ , $M = 4$ . . . . .	80
6.18	Waypoints of $\mathcal{P}_{\text{selected}}$ for the extended case . . . . .	80
7.1	Comparison path planners in sections 6.1.1 and 6.1.2 . . . . .	86



# Abbreviations

AOI	Area of Interest
APF	Artificial Potential Field
APS	Acoustic Positioning System
AROV	Autonomous Remotely Operated Vehicle
AUR-lab	Applied Underwater Robotics Laboratory
AUV	Autonomous Underwater Vehicle
CNN	Convolutional Neural Network
COB	Center of Buoyancy
COG	Center of Gravity
cRIO	Compact Reconfigurable Inputs and Outputs
DOF	Degree of Freedom
DP	Dynamic Positioning
DVL	Doppler Velocity Logger
EA	Evolutionary Algorithm
EKF	Extended Kalman Filter
FM	Fast Marching
FPGA	Field-Programmable Gate Array
GA	Genetic Algorithm
GUI	Graphical User Interface
HIL	Hardware-in-the-Loop
ILOS	Integral Line-of-Sight
IMR	Inspection, Maintenance and Repair
IMU	Inertial Measurement Unit
IO	Input-Output

LSM Level Set Methods

MOGA Multi-Objective Genetic Algorithm

NED North-East-Down

NP-hard Non-Deterministic Polynomial-Time Hard

NSGA-II Non-Dominated Sorting Genetic Algorithm

NTNU Norwegian University of Science and Technology

PAES Pareto-Achieved Evolution Strategy

PID Proportional-Integral-Derivative

PNO Passive Nonlinear Observer

PSO Particle Swarm Optimization

QPSO Quantum-behaved Particle Swarm Optimization

ROV Remotely Operated Vehicle

RRT Rapidly-exploring Random Trees

SBX Simulated Binary Crossover

SLAM Simultaneous Localization and Mapping

SNAME Society of Naval Architecture and Marine Engineering

SPEA Strength-Pareto Evolutionary Algorithm

SRG Speed-Regulated Guidance

SSBL Super Short Base Line

TCP Transmission Control Protocol

UDP User Datagram Protocol

UHI Underwater Hyperspectral Imaging

UUV Unmanned Underwater Vehicles

VI Virtual Instrument

# Chapter 1

## Introduction

This thesis covers enabling the remotely operated vehicle (ROV) Minerva 2 to perform autonomous docking missions. Considering the research area of path planning key to obtain persistent autonomy [9], the research hypothesis upon which this thesis is based concerns the path planner. Furthermore, the research hypothesis states that the non-dominated sorting genetic algorithm, NSGA-II, will render a high-performance optimal path planner enabling Minerva 2 to take on autonomous docking missions. The work includes the development of an optimal path planner based on NSGA-II, which is a multi-objective evolutionary algorithm. The path planner is applied to the optimal path planning problem for autonomous underwater docking missions. Additionally, an integrated system is obtained implementing the path planner in the control system of Minerva 2, enabling the ROV to perform autonomous underwater docking missions.

### 1.1 Background

#### 1.1.1 Remotely Operated Vehicles (ROVs)

In the industry, some underwater operations that utilize remotely operated vehicles (ROVs) are intervention, maintenance, and repair (IMR) operations, seabed mapping, inspection, military, and research-related tasks, etc. ROV operations today are highly dependent on a mother-ship with dynamic positioning (DP) capabilities. The ROV is tethered to the mother-ship, and a skilled human operator is required to achieve accuracy and efficiency during operation.

ROVs are a subclass of unmanned underwater vehicles (UUVs). [10] gives a review of the ROV subclass of UUVs and distinguishes between the different types of ROVs by the task at hand for the ROV to perform. ROVs are tethered vehicles, with a physical link. i.e. the umbilical, supporting the vehicle from the surface with unlimited power, thus the possibility to carry out operations of infinitely long duration, as well as online communication. The tether reduces the ROVs level of autonomy and is of course subject to forces perturbing the vehicle. Nevertheless, ROVs have great capabilities in underwater operations due to their access to power, robotic manipulator arms, high payload capacity in terms of equipment and DP capabilities. According to [11] ROVs are further divided into the following three subclasses depending on their purpose:

1. Pure Observation Class
2. Observation Class
3. Work Class

## ROV Minerva 2

This thesis is based on the ROV Minerva 2, illustrated in Figure 1.1. Minerva 2 is the latest asset of the NTNU research center for underwater robotics, Applied Underwater Robotics Laboratory (AUR-lab) [12] .



Figure 1.1: Illustration of the ROV Minerva 2

The AUR-lab has previously owned/owns ROVs such as the ROV Minerva and the ROV 30k, thus some contributions covering ROV control systems are already available. These contributions apply to the ROV Minerva 2, and thus, relevant contributions are addressed in section 1.5.1 in a literature review manner.

### 1.1.2 Autonomy

This section aims to clarify the meaning of the expression autonomy in underwater operations. The term autonomy says something about to what extent an intelligent system can deliberate and manage unexpected events in complex environments with the use of integrated mathematical models, data from sensors, instruments and optimization algorithms in real-time [1]. For such systems, there exist several definitions of autonomy levels, describing to what extent the system is autonomous. One of these definitions given in [13], suggests dividing autonomy into four levels:

1. Manual Operation
  - The vehicle is controlled directly by a human operator.
2. Management by Concept
  - The system automatically recommends actions for selected functions, prompted by the operator for decision-making.
3. Management by Exception
  - The system automatically executes mission-related functions.
  - The human operator may override or alter parameters and cancel or redirect actions, acting as a supervisor.
4. Fully Autonomous
  - The system automatically executes mission-related functions.

- The operator is only alerted to function progress.

Today's ROV operations are mostly in levels 1 and 2. However, future ROV operations will likely cover levels 3 and 4. From a development point of view, as suggested in [14], it is beneficial to keep a human supervisor in the loop, i.e., autonomy level 3.

## 1.2 Motivation

The motivation behind this thesis is to increase the level of autonomy for ROV underwater operations, focusing on underwater docking operations. It is believed that increasing the level of autonomy in such operations will be beneficial, in terms of making ROVs less dependent of human operators, and thus, saving cost, increase accuracy and efficiency while maintaining safety. Overall, it can be claimed that this is a step to make underwater operations more streamlined.

When ROVs first were commercially introduced, the ability to operate at depths outside of the range of human divers did arise. ROVs have unique capabilities when it comes to IMR-operations compared to other classes of UUVs, due to their high payload capacity and unlimited power access. Thus, ROVs are capable of performing some of the most complex underwater operations. In the future, by increasing the level of autonomy of ROV operations, the dependence of a mother-ship could be decreased. Additionally, a higher level of autonomy is likely to contribute to lower reliance on human intervention. Thus, increasing the level of autonomy in ROV operations could lead to significant cost reductions. With the vast amount of subsea wells on the Norwegian Continental Shelf, the need of IMR-operations in the oil and gas industry alone in the future can be assumed to be significant.

Furthermore, [14] examines the possibility of achieving commercially available autonomous underwater vehicles (AUVs) with intervention capabilities. This article suggests that it is beneficial to keep the human in the loop as a supervisor and that fully autonomous intervention vehicles are more applicable for inspection-related operations. Some research has been done in the field of docking of AUVs, for instance in [15]. Some of this research could apply to ROVs. Thus, autonomous underwater docking technology is not new, however, it is first now that the ability to utilize it commercially has arisen. In that sense, applying advanced control architectures to ROVs could render autonomous ROVs (AROV), with the capability of performing mission-related functions autonomously [2]. Likely, such advanced control architectures will render ROV operations even better accuracy and efficiency compared to what is achievable by skilled human operators.

By further development of ROVs in terms of autonomous capabilities, it is believable that such vehicles can play a role in integrated autonomous underwater operations [1]. The vision of Oceanering proposes an integrated solution, where the ROV is deployed by a ship with DP capabilities, in the standard way. Then, instead of the mother-ship doing station-keeping while the IMR-operation is performed, the ROV is tethered to a floating buoy providing communication. Due to this online communication, the human operator can sit elsewhere, e.g., onshore and supervise the operation [16]. On that note, the availability of 4G communication technology in the North Sea, which at the moment is a significant region in the oil and gas industry, is excellent as all the rigs are equipped with 4G antennas. Communication by fiber optics is another option. Of course, this would require equipping ROVs with batteries providing the necessary power rather than supplying the ROV with electricity through the umbilical. Also, installing battery recharging facilities on the underwater docking stations is likely to benefit the entire repertory of underwater vehicles.

Moreover, [17] reviews the role of the ROV in future underwater operations. This paper addresses several concepts and discusses the benefits and challenges. Through the paper, the authors conclude that an AUV carrying a ROV within its body is the ideal solution. The AUV will be able to dock at underwater docking stations to charge batteries and exchange data. The AUV could move to a site requiring IMR-operations where the ROV could be deployed and perform the operation while tethered to the AUV. This concept will, of course, require a higher level of autonomy than the

former.

Summing up, the applicability of ROVs to autonomous docking missions and IMR-operations are addressed in general for overview purposes. Thus, there seems to be a market for ROVs, making such vehicles a building block in the future repertory of underwater vehicles. Moreover, there are several angles of looking at the problem of increasing the level of autonomy. One of those is improving the path planner supported by [9], which claims that the performance of the path planner is a crucial factor for achieving persistent autonomy in AUVs. Since AUVs are underwater vehicles operating at a high level of autonomy in general, this is probably applicable to ROVs as well, assuming an increase in their level of autonomy. Overall, this has lead to the research hypothesis upon which this thesis is based, in short, the development of the path planner will lead to persistent autonomy in docking operations.

### 1.3 Research Topic and Objectives

Overall, this thesis is based on the research hypothesis, which states that the research area of path planning is key for achieving persistent autonomy. Furthermore, it states that solving the optimal path planning problem by the non-dominated sorting genetic algorithm, NSGA-II, will render a high-performance optimal path planner for autonomous docking missions of ROVs using waypoint guidance. The origin and further exploration of this research hypothesis are based on the following objectives:

- Conduct a literature study within the field of underwater operations, aiming to find an interesting angle in terms of research questions for this thesis.
- Propose a mathematical model of the ROV and present relevant theory concerning guidance, navigation, and control.
- Propose theory within mathematical modelling of ROVs, including theory within guidance, control and navigation.
- Define the optimal path planning problem for autonomous docking missions.
- Propose optimization theory behind the non-dominated sorting genetic algorithm, NSGA-II, and develop an optimal path planner able to take on autonomous docking missions for the ROV Minerva 2 based on this algorithm.
- Examine the performance of the path planner as a stand-alone application.
- Implement the NSGA-II based optimal path planner into the control system of the ROV Minerva 2 in order to obtain an integrated system with the capabilities to take on autonomous docking missions.
- Examine the performance of the integrated system performing autonomous docking missions in simulations.

### 1.4 Scope and Limitations

This scope of this thesis is examining the optimal path planning problem applied to autonomous docking missions of the ROV Minerva 2. A comprehensive literature survey is carried out, narrowing in on optimization techniques used in path planning. The thesis covers the development of an optimal path planning algorithm based on the non-dominated sorting genetic algorithm, NSGA-II as well as the implementation of this path planner into the control system of ROV Minerva 2. The



path planner is implemented as a part of the autonomy module in the control system, enabling Minerva 2 to take on autonomous docking missions. The NSGA-II path planner is developed using `Matlab` and later implemented into the control system. The performance of the path planner as a stand-alone application and the performance of the integrated system are examined. Due to technical difficulties with Minerva 2 and the mother-ship of Minerva 2, RV Gunnerus, the results obtained during the work with this thesis is solely based on simulation results. According to the plan outlined beforehand, experimental results should be obtained in terms of field trials. Unfortunately, field trials have not been possible to conduct.

This thesis presents a comprehensive overview of the different aspects of the control system of Minerva 2 relevant for autonomous docking missions, in terms of mathematical modelling and theory within guidance, navigation, and control. These are essential components when examining the overall performance during docking missions. However, settings concerning these components of the control system are addressed, but changes are not proposed during the work with this thesis, as it emphasizes the path planning module.

Preliminary to the master's thesis, the work with the project thesis during the fall of 2018 acts as a base. The project thesis focused on getting an overview of the Minerva 2 control system, in terms of literature study and performing simple simulations of the ROV in the control system without significant changes. During this literature study, the objective has been to obtain interesting research topics. Thus, the literature review in Section 1.5.1 and the theory presented in Chapter 2 and Sections 3.1.1, 3.3, 3.4 and 3.5 is quite similar to what is presented in the project thesis. Still, this master's thesis presents a more extensive work both in terms of literature review and presentation of theory compared to the project thesis.

## 1.5 Literature Review

### 1.5.1 ROV Control System Development

In cooperation with the AUR-lab several relevant publications originates from the Norwegian University of Science and Technology (NTNU), with the ability to have hands-on experience with ROVs, and thus, performing field trials in addition to simulations and theoretical work. Relevant literature will be addressed, where the goal is to give an overview of the development of the ROV motion control system. The doctoral thesis [7] presents the governing mathematical models for the vessel dynamics of the ROV for an automatic motion control system and focuses on underwater navigation and guidance. Challenges in navigation are solved by a control architecture combining model-based state estimation with sensor-based state estimation. That is, combining an extended Kalman filter with a sensor-based attitude-observer. The proposed sensor-based state estimation utilizes sensors such as inertial measurement unit (IMU) and Doppler velocity log (DVL). Within the guidance topic, automated tasks such as trajectory tracking and terrain following are examined. Constant jerk reference models are introduced for A to B moves, to reduce pilot-induced oscillations. Terrain following is enhanced by seafloor geometry approximation based on the DVL combined with a guidance law relating the desired depth to the corresponding reference altitude.

The motion control system from [7] is extended in the doctoral thesis [18] focusing on path generation. The implementation of reference models rendering smooth references in position, velocity, and acceleration for target tracking and DP capabilities is proven to enhance the motion accuracy of the control system. A trust allocation algorithm and alternatives to the controller and observer are provided. On that note, the development of ROV DP capabilities is addressed in many publications. In [1], the development of DP and tracking system is examined, in an experimental approach. In [19] field tests are carried out to compare different observers. Namely, the compared observers are linear Kalman filter, extended Kalman filter, adaptive Kalman filter, and passive nonlinear observer (PNO), based on performance in DP operations.

The doctoral thesis [10] is written based on the motivation of increasing the level of autonomy in marine robotics. Furthermore, [10] proposes safe path planning and re-planning systems using a Voronoi diagram for obstacle avoidance and Dubins path and Fermat spirals for path smoothing. A signal processing module is developed and implemented to increase the accuracy of ROV operations. The utilization of seabed mapping, the importance of Human-Machine Interface, and the usage of cameras in underwater operations are also pointed out.

Challenges in underwater operations with an increasing level of autonomy, such as improving underwater navigation, are crucial to achieving accuracy and efficiency in such operations. Moreover, [20] addresses challenges in underwater navigation, proposing a technique for increasing the accuracy of acoustic navigation. Furthermore, [21] offers a broader overview of challenges, not only concerning DP operations but also more complex autonomous underwater operations. The motivation behind this paper is the arguably increasing need for underwater IMR-operations and the possibilities of streamlining such operations by increasing the level of autonomy. It addresses research topics such as localization, guidance, and path planning.

Next, [22] is another publication addressing the DP capabilities for ROV based on experimental results. This article concludes that an underwater vehicle during transit should be oriented such that its sensors are located orthogonal to the seafloor to achieve reliable sensor measurements. This finding is a vital take-home message when it comes to ocean mapping and monitoring.

The ROV motion control system is further developed in the doctoral thesis [6] based on ocean mapping and monitoring, in terms of obtaining the desired quantity of data with desired quality. The thesis proposes a control strategy with capabilities of maintaining an ROV at a constant distance from an area of interest (AOI) , using a DVL. By pointing the cameras mounted on the ROV as well as the DVL directed towards the AOI, field tests showed that this strategy enhanced the ROV motion control system with the capability of obtaining high-quality datasets of the AOI. This enhancement is promising to retrieve data from distant and dangerous environments requiring significant human operator endurance. Significant enhancements to achieve a genuinely autonomous system are also addressed, enabling the system to deliberate and make decisions. The use of underwater hyperspectral imaging (UHI) in computer analysis and online classification of objects of interest is demonstrated to perform autonomous mapping operations from surface launch to AOI using autonomy control architecture.

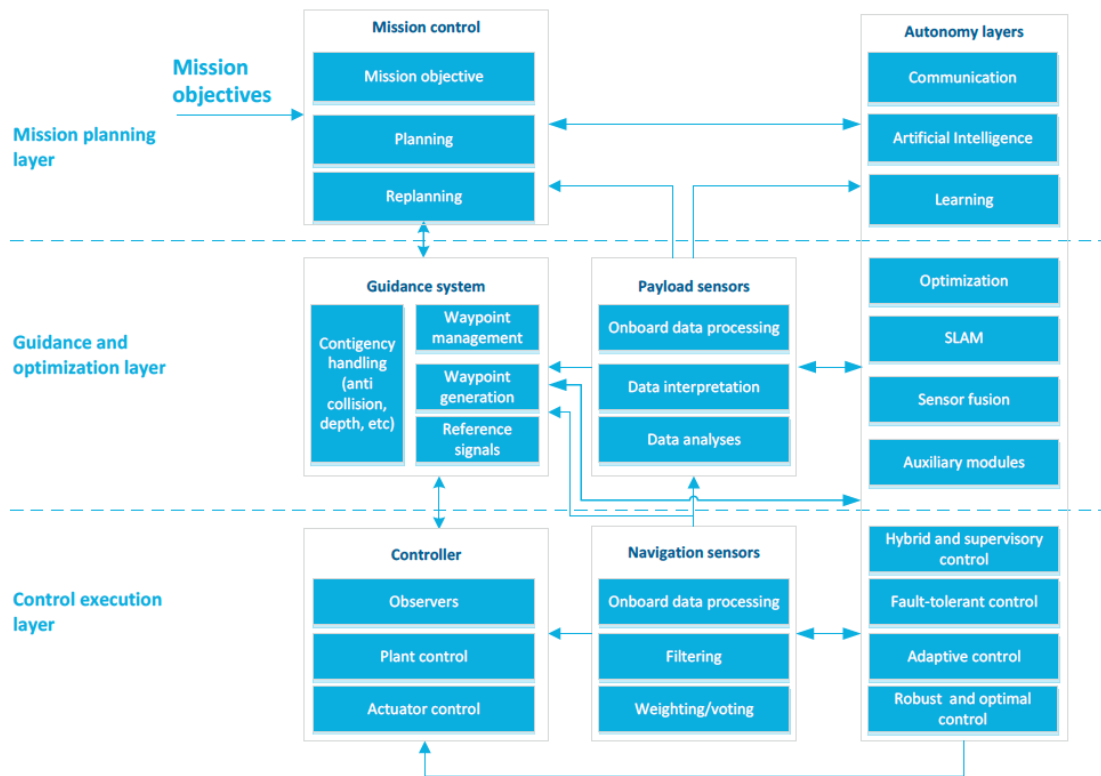


Figure 1.2: Control Architecture Layers [1]

Moreover, [1] proposes an advanced control architecture, divided into multiple layers. The control architecture depicted in Figure 1.2 is designed in a bottom-up manner, traditionally divided into three layers. The top layer, mission planning layer, takes the mission objectives as input and performs mission planning. The top layer provides the next layer, guidance, and optimization, with input enabling it to generate reference signals based on waypoint management and collision avoidance, etc. The reference signals are given as input to the bottom layer, the control execution layer, which computes the control signals. The control architecture is augmented by an autonomy layer, which can give input to all levels of control in the traditional architecture. The end goal is obtaining a system able to deliberate, learn, plan, and re-plan. This system can act autonomously, to streamline underwater operations in terms of saving costs and reducing the dependence of human operators and hence maintain safety.

Furthermore, [2] proposes such an autonomy layer. The goal is to obtain a semi-autonomous ROV with the ability to perform certain subsea operations autonomously, based on a hybrid control architecture combining a deliberative layer and a reactive layer. Subsea operations are generalized from the launch of the ROV at the surface to the approach of a SOI on the seabed.

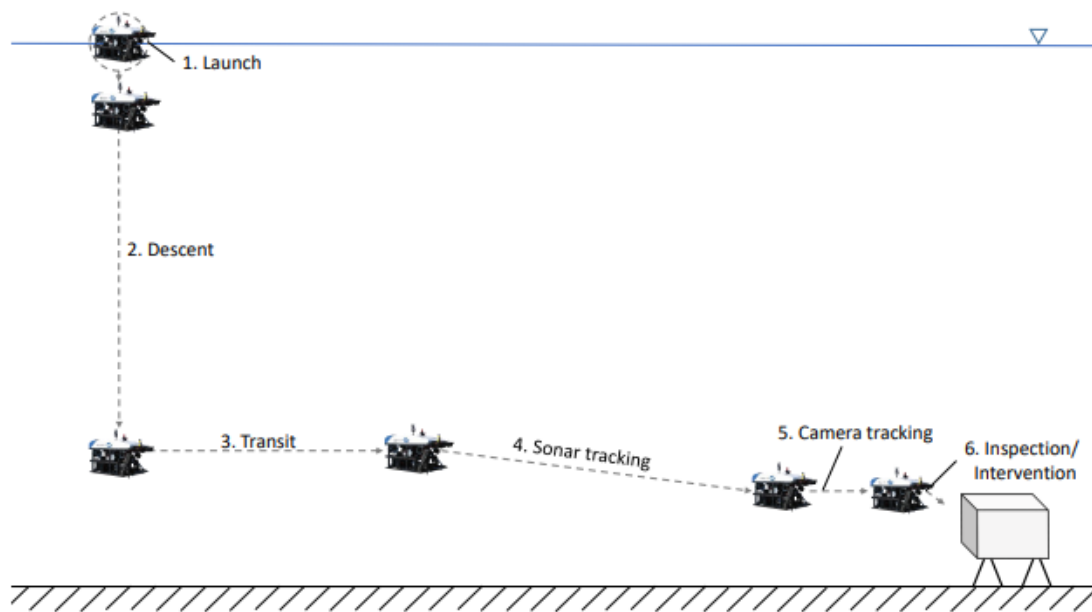


Figure 1.3: Generalization of an Autonomous Subsea Operation [2]

The deliberative layer handles the overall mission planning necessary to get the ROV from the surface to the SOI and is divided into six steps, namely launch, descent, transit, sonar tracking, camera tracking, and inspection/intervention, as shown in Figure 1.3. In general, the reactive layer concerns situations that occur when the vehicle is exposed to unexpected situations, in [2] focusing on collision/obstacle avoidance using computer vision. This increased level of autonomy in the control system enhanced the ROV with autonomous steering abilities, and promising results were obtained both from simulations and field trials.

In underwater robotics, computer vision typically utilizes sonars and camera to give the system understanding of the operating environment. In the master's thesis [23], computer vision based collision avoidance for ROV is addressed, utilizing cameras. The ROV control system is enhanced with the capability of identifying obstacles based on the analysis of input from cameras. Field trials are carried out with the presence of obstacles, and the ROV is able to exit path based on the processing of camera-data successfully.

Moreover, [24] addresses the application of computer vision in underwater robotics. The publication states how IMR-operations can benefit from the vast research within the field of computer vision. The objectives of [24] are to propose a system that allows an ROV to adjust its pose towards intervention objects and locate intervention targets. Solutions to these objectives are provided using camera-based computer vision. The former objective is solved by a close-range guidance algorithm, while the latter by a label detection algorithm. The algorithms are tested using recorded video. The information extraction seems satisfactory, however, there seem to be some drawbacks with visual navigation as the stand-alone navigation method. Marine snow is addressed in a challenge in underwater utilization of camera-based computer vision.

Furthermore, [25] proposes a visual motion estimation algorithm for Minerva 2. The stereo camera on the ROV renders 3D images, enabling visual motion estimation based on feature extraction from photography. Interconnecting the visual motion estimation capabilities with the already existing state estimation capabilities from a Kalman Filter, as given in [7], the goal is to improve the overall ROV motion control system. In [25], Simultaneous Localization and Mapping (SLAM) is addressed as an exciting research topic for further enhancement of the ROV motion control system.

Along with the introduction of the ROV Minerva 2, the motion control system needed further

development. The new ROV can exercise quite large amounts of propulsion power. The assumption that ROVs are considered passively stable in roll and pitch, such that these degrees of freedom (DOFs) could be disregarded in the control design does not longer hold for Minerva 2, in contrast to previous ROVs owned by the AUR-lab. Thus, the motion control system developed up to this point consider only 4 DOFs, namely surge, sway, heave, and yaw. However, in [26] the motion control system is augmented to consider all 6 DOFs, meaning that the controller is expanded to account for roll and pitch motion. This augmentation also required a new thrust allocation algorithm, which is provided in the thesis, solving the thrust allocation problem as an optimization problem. [26] also pays attention to system identification and determination of the hydrodynamic coefficients of the ROV Minerva 2 using both analytic and experimental methodology.

### 1.5.2 Strategies for Homing/Docking Applied for AUVs

Docking is the last sequence of an underwater operation and is a maneuver where an underwater vehicle docks into an underwater structure. Homing is a maneuver leading to the docking operation and can, therefore, be seen as a necessary step to consider when doing underwater docking as the overall underwater operation. When referring to underwater docking, the operation in mind is often the entire operation from the initial position of the vehicle until the vehicle is parked in an underwater structure. As this thesis focuses on enabling ROVs to perform autonomous docking and the previous work done within such docking operations for ROVs is limited, a literature study is carried out to establish knowledge on docking of AUVs. The reason is that some of the work within docking missions of AUVs probably can be applied to docking of ROVs.

Moreover, [15] addresses docking missions for AUV. It focuses on most aspects of a planned mission that is the underwater docking operation. A typical layout of an underwater docking system is given. The docking problem is defined for the appropriate application, with a known initial position of the AUV and a known position of the docking station. Path following utilizing lookahead based steering is chosen as the preferred guidance algorithm of getting to the docking station, and thus, a path is generated. Some attention is also paid to control design in terms of designing a suitable autopilot controller. Performance is verified by simulation, in two cases, one where the docking station is stationary, and one where the docking station is moved by a step after the mission is initiated such that the AUV has to re-plan its trajectory towards the docking station.

Furthermore, [27] is a NTNU doctoral thesis on path-planning, guidance, and navigation for docking of underactuated AUVs in the presence of current. In contrast to AUVs which are underactuated, as they cannot control sway velocity, ROVs are fully actuated and thus controlled in 4-6 DOFs. Due to this, AUVs have to transit with some compensation crab angle when operating in environments with cross-current. ROVs, on the other hand, are fully- or over-actuated. Therefore ROVs can control their sway velocity when subject to current. Thus, AUVs can mainly transit forward, while ROVs are more maneuverable, enabling more options in terms of guidance algorithms for the ROV. Next, [27] lists the benefits of developing underwater docking capabilities, such as charging of batteries, data exchange, development of a subsea structure residence increasing underwater operation efficiency. However, challenges within communication and navigation are listed. This can be generalized to concern ROV operations as well. A hybrid guidance law combining integral line-of-sight (ILOS) and speed-regulated guidance (SRG) laws are proposed enabling to dock with zero crab angle in environments with cross-current. A path-planning algorithm enabling an underactuated AUV to plan its transit with a limited field of view using logarithmic spirals. The thesis also proposes a convolutional neural network (CNN) based machine learning algorithm enabling the AUV to detect the docking station.

### 1.5.3 Optimization Techniques for Path Planning

There is not a vast amount of publications on optimal path planning for ROVs, however, there is significantly more on adjacent fields such as AUVs and UAVs. The reason being that vehicles with a higher level of autonomy, such as AUVs and UAVs, are more dependent on the performance of the path planner than ROVs, which commonly are controlled by a pilot. However, while increasing the level of autonomy for ROVs, the performance of the path planner becomes more crucial, making optimal path planning an exciting research topic. This Section contains reviews of some publications on optimal path planning.

Surveys on path planning for AUVs are carried out in [9] and [3], which states that path planning is a crucial component to improve persistent autonomy for AUVs. These articles present a review of research works focusing on technology areas such as path optimization for AUVs. Furthermore, brief walk-throughs of several optimization techniques are given and assumptions, benefits, and drawbacks are addressed.

Firstly, [9] and [3] address a category of graph search methods. This category concerns discrete optimal path planning algorithms, such as Dijkstra, A\*, Field D\* and Theta\*. These algorithms require knowledge about the search space, represented by a grid-shape graph wherein the edges are labeled with a cost of traveling from a vertex to one of its neighbors. Dijkstra's algorithm is able to obtain paths based on minimum cost, as it computes every possible path assuming a known starting point and destination point. A\* is proven to be a more effective algorithm within this category, due to its heuristic searching ability providing an estimate of the cost of the best path that passes through a particular node. Along with the cost leading up to a particular node, this enables the algorithm to determine which node it must visit next.

According to [9] and [3], graph search methods are in general criticized for unnaturally constraining the motion of the vehicle to limited directions due to their discrete state transitions. Modified variants of A\*, such as Field D\* and Theta\*, are developed to overcome this drawback. Furthermore, algorithms within this category are still not able to overcome the challenge that they are computationally expensive for high dimensional problems. As for the A\* algorithm, better performance in terms of path smoothness can be achieved with a higher resolution grid. However, this results in the A\* algorithm coming off significantly worse in terms of computational cost.

The second category of optimization techniques addressed in [9] and [3] consists of the Fast Marching (FM) algorithm and the Level Set Methods (LSM). The FM algorithm can be regarded as a continuous version of Dijkstra's algorithm and uses a first-order numerical approximation of the Eikonal equation. To overcome drawbacks concerning lack of efficiency with the FM algorithm there exists an expanded version, known as FM\*, which is heuristically guided to maintain the accuracy of FM and the efficiency of A\*. However, FM\* is limited as it uses a linear anisotropic cost function to improve computational efficiency. The LSM generates the time-optimal path by solving a particle tracking equation backward in time after it evolves a front from the vehicles start point until it reaches the destination point. Thus, the LSM is a more general algorithm than FM. Next, [9] and [3] addresses yet another improvement of the FM\* algorithm, namely an improvement using wavefront expansion to calculate the shortest time paths. Summing up this category, the articles state that the LSM can be used for more complex problems while the FM\* using wavefront expansion requires less computational time.

Moreover, [9] and [3] address the artificial potential field (APF) for global path planning, an algorithm used in robotics and underwater path planning. In this context, path planning is done by introducing an artificial potential field on the obstacles in the environment, preventing vehicles of getting close to them, and thus, generating safe paths. APF is a computationally inexpensive method, however, prone to producing locally optimal solutions.

Furthermore, [9] and [3] address Rapidly-exploring Random Trees (RRT) as an algorithm to solve the path planning problem. The key idea behind RRT is a tree that is incrementally growing to

explore the search space until the tree branches reach the destination point. As a result of this, RRT is more likely to find feasible paths that are not necessarily optimal. On the other hand, RRT is effective on high dimensional problems. The RRT\* algorithm is a modified version of the RRT algorithm, aiming to improve the algorithm even more by reducing the required memory for storing the tree. The idea is that the tree grows to a predefined number of nodes, then a weak node is removed whenever a high-performance node is added.

Lastly, [9] and [3] addresses the category of evolutionary algorithms (EAs). This category consists of algorithms such as different versions of genetic algorithms (GAs) and particle swarm optimization (PSO). EAs are population-based search schemes utilizing information sharing among the population members to enhance the search process following deterministic and/or probabilistic rules. However, there are some differences between GAs and PSO. For instance, GAs incorporate survival of the fittest, while in PSO all particles retain through the course of the run. Furthermore, PSO does not utilize the standard evolutionary operators, such as crossover and mutation, as GAs do. In the PSO algorithm, each particle adjusts its course through the search space based on its own searching experience as well as the searching experience of other particles. The quantum-behaved particle swarm optimization (QPSO) is considered an improved version of the conventional PSO. Whereas the conventional PSO algorithm uses position and velocity update for every particle, QPSO assumes that each particle has quantum behaviour. Summing up, EAs are considered well suited to large-scale optimization problems, which path planning problems often are.

Furthermore, the general path planning problem with bounded velocities and multiple obstacles is non-deterministic polynomial-time hard (NP-hard) [28], and thus, can be computationally expensive. EA based optimization techniques have been proven to be an effective way of dealing with such NP-hard problems. Moreover, due to their population-based characteristics, EAs can be implemented on parallel machines to achieve super-linear speed-up with the number of processors. On the other hand, a drawback with EAs is that they may converge to sub-optimal solutions within finite-time[9],[3].

In addition to the comprehensive survey on optimization techniques, [3] also provides case studies on path planners utilizing different optimization techniques and a novel QPSO based path planner. Performance evaluation of these path planners is carried out through simulations in two different mission scenarios. The simulation environment is a current field with an area of  $25^2 \text{ km}^2$  with a grid resolution of  $1 \text{ km}^2$ . The current field in question is two dimensional, thus constant in the z-direction. However, the path planning problem is still three dimensional. The first scenario has no obstacles, while the latter has static obstacles with fixed size and known positions. The objective of the path planning problem is to obtain the path with the least time usage, with a constant water-reference speed in an environment with current, while maintaining collision avoidance. The starting point and the destination point are known, and the path planners in question are based on the optimization techniques A\*, RRT and RRT\*, GA, PSO, and QPSO. Next, [3] provides comparison among the path planners. Also, they are compared by Monte-Carlo simulations on a 100-runs basis for both scenarios yielding the mean and the standard deviation of the cost value. Figure 1.4 presents graphical results from some of the case studies in [3] given from the scenario with obstacles. Figure 1.4a represent results obtained by an A\* based path planner, Figure 1.4b represents results obtained by a RRT\* based path planner and Figure 1.4c represents results obtained by path planners based on the EAs, namely PSO, QPSO, and GA.

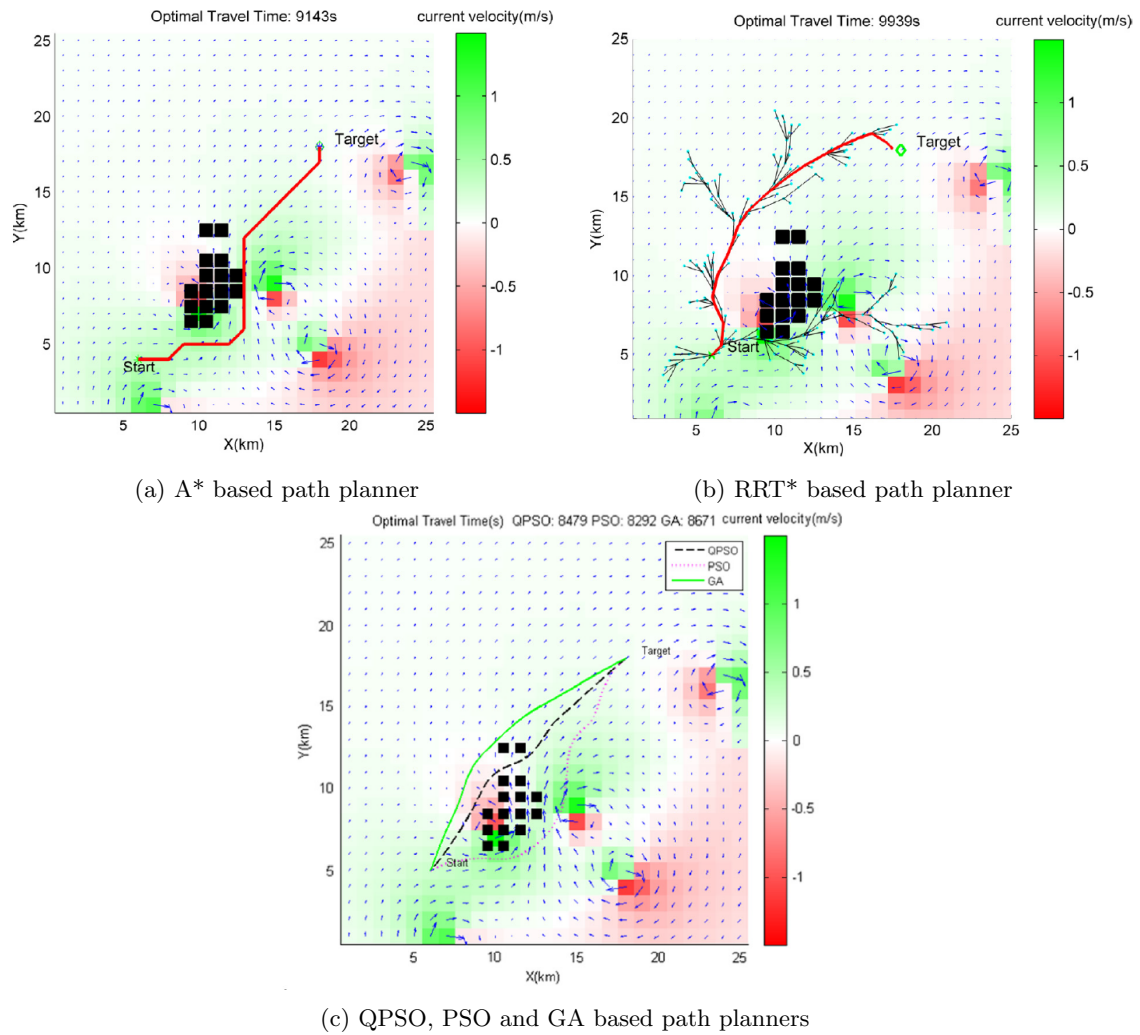


Figure 1.4: Case study results in static current environment with obstacles, illustrations by [3]

The common objective of the case studies is to obtain the time-optimal trajectory. Thus, Table 1.1 presents the optimal travel time obtained using the different optimization technique based path planners for the case studies in[3].

Table 1.1: Optimal Travel Time Comparison [3]

Path Planner Optimization Algorithm	Travel Time [s]	Travel Time [s]
	Without Obstacles	With obstacles
A*	8 343	9 143
RRT	8 976	10 9069
RRT*	7 461	9 939
GA	7 492	8 671
PSO	7 729	8 292
QPSO	7 519	8 479

Firstly, [9] and [3] comments that all path planners generate feasible path following what is expected in the survey as mentioned above. From the statistical results from the Monte Carlo simulations and the data in Table 1.1, it is concluded that the GA, PSO, and QPSO based path planners outperform the A\*, RRT and RRT\* based path planners in such a static current field environment with static obstacles.



Table 1.2: Comparison of Optimization Algorithms for Path Planning [9], [3]

Category	Algorithms	Comments
Graph search schemes	Dijkstra	Discrete state transitions
	A*	Computationally expensive for high-dimensional problems
	Field D*	Discrete state transitions limits directional changes
	Theta*	
	FM/FM*	Computationally expensive for high-dimensional problems
	LSM	
	APF	Computationally inexpensive
		Susceptible to locally optimal solutions
	RRT	Fast and effective
	RRT*	Generates feasible solutions that not necessarily are optimal
		Generated paths may require further refinement
EAs	GA	Suitable for handling NP-hard problems
	PSO	May converge to suboptimal solutions in finite time
	QPSO	

Overall, there are several concluding remarks made on path planning for AUV in [9] and [3]. Table 1.2 is intended to summarize some properties of the different path planners. Firstly, path planning problems are often large scale optimization problems. No matter the chosen algorithm, the computational requirement grows exponentially for solving high dimensional problems. Due to this fact, many path planners project the three-dimensional problem into two dimensions. It is also pointed out that such two-dimensional problems do not completely embody the three-dimensional information of the ocean environment, especially regarding currents which are important in AUV operations due to their maneuverability. Thus, the conclusion that EAs are beneficial for path planning problems. The different optimization algorithms have both benefits and drawbacks, and [9] and [3] underlines the difficulty of comparing and reviewing different optimal path planners. Also, it is mentioned that most of the research in path planning for AUVs is based on simulation and that experience based on real-life field trials is lacking. Lastly, it is suggested that path planning still is an incomplete research field and that the lack of performance metrics poses a challenge when it comes to comparing path planners based on different optimization techniques. It is also stated that there is a gap between online and offline path planning and path planners with online re-planning capabilities would benefit from an increase in available computational power. On that note, [29] proposes an interesting three-dimensional collision avoidance algorithm for AUVs, which poses as a component in path planners in complex environments.

The articles [9] and [3] treat the category of GAs altogether as a whole, yet, there exists a wide range of genetic algorithms. These algorithms are, as mentioned, population-based algorithms applying survival of the fittest and genetic operators such as crossover and mutation. Of course, the performance also tends to vary between the different GAs, also depending on the application. Moreover, [8] proposes a fast and elitist multi-objective genetic algorithm (MOGA), the non-dominated sorting genetic algorithm (NSGA-II). Multiple objectives in optimization problems give rise to a set of optimal solutions, called Pareto-optimal solutions. That is, solutions which according to evolutionary heuristic are equally good. It is desirable for a MOGA to generate as many Pareto-optimal solutions as possible to ensure diversity among the solutions. Furthermore, [30] suggests that the path to be followed by a vehicle using a MOEA based path planner should be selected, from the Pareto-optimal set, according to expert preference. Besides, [31] proposes an AUV NSGA-II based path planner showing promising results.

Moreover, [8] shows that that NSGA-II overcomes criticism towards the previous iteration of the algorithm, namely NSGA. The challenges that are solved in the proposed algorithm are lowering the computational complexity, ensuring elitism, and reducing the need for user input. In this article, case studies are carried out on difficult test problems in the literature, and NSGA-II is shown to outperform MOGAs such as the Pareto-achieved evolution strategy (PAES) and the

strength-Pareto Evolutionary Algorithm (SPEA). Thus, making NSGA-II a superior GA.

## 1.6 Structure of Thesis

This Section addresses the structure of this master's thesis. The structure is outlined according to the objectives given in Section 1.3:

**Chapter 1** presents a literature review in Section 1.5 aiming to provide an overview of the ROV Minerva 2 control system and at the same time obtaining exciting research questions.

**Chapter 2** presents the theory behind the mathematical modelling of ROVs.

**Chapter 3** aims to provide an overview of the control system for the ROV Minerva 2, giving some modules of the control system more attention than others. More explicitly, modules that are important when developing an optimal path planner for autonomous docking and simulating it in an integrated system are the acoustic navigation specifications, given in Section 3.1.1, the controller, given in Section 3.3, the observer, given in Section 3.4, the guidance module, given in Section 3.5 and the optimal path planning problem given in Section 3.7. A summary of the optimal path planning problem for autonomous docking missions follows in Section 3.7.7.

**Chapter 4** presents the theory required to apply NSGA-II to optimization problems, focusing on all operators necessary to solve such problems. The user-defined parameters of NSGA-II are summed up in Section 4.1.7.

**Chapter 5** presents the development of the path planner and the integration of the path planner into the control system. Section 5.1 covers the development of the NSGA-II based optimal path planner in accordance with Section 3.7.7 and Chapter 4. Background on the existing control system for Minerva 2 is given in Sections 5.2-5.5, based on the theory given throughout Chapter 2 and 3. Lastly, Section 5.6 covers the implementation of the NSGA-II based path planner into the control system, forming the integrated system.

**Chapter 6** presents the results obtained during the work with this thesis. Section 6.1 presents results based on the NSGA-II based optimal path planner as a stand-alone application. Sections 6.2 and 6.3 presents results from autonomous docking scenarios simulated by the integrated system.

**Chapter 7** presents the discussion of the results presented in Chapter 6, namely, Section 7.1 corresponds to the results presented in Section 6.1 and Section 7.2 corresponds to the results presented in Sections 6.2 and 6.3.

**Chapter 8** presents concluding remarks on the NSGA-II based optimal path planner and the integrated system and addresses topics for further work.

### 1.6.1 Appendices

The attached appendices in this thesis consist of the `Matlab`-coded NSGA-II based optimal path planner corresponding to the method given in Chapter 5.1. The attached `Matlab`-code includes a `parameters.m`-file, given in Appendix A2, where the parameters of the optimal path planning problem for autonomous docking of the ROV Minerva 2 are assigned numerical values, corresponding to summaries of the optimal path planning problem given in Section 3.7.7 and NSGA-II given in Section 4.1.7.

## 1.7 Thesis Contribution

This thesis proposes an optimal path planner for autonomous docking missions. The path planner is based on the multi-objective optimization algorithm, non-dominated genetic sorting algorithm NSGA-II, and is efficient in terms of computational complexity.

The focus of this thesis is two topics, namely, the examination of the performance of the NSGA-II based optimal path planner as a stand-alone application and examination of the performance of autonomous docking missions during simulations of the integrated control system of Minerva 2 including the path planner. The overall objective of this work is to increase the level of autonomy of ROV operations, as it is believed to streamline IMR-operations in terms of saving costs, increase efficiency while maintaining safety. Enabling ROVs to perform autonomous docking missions is regarded as a step towards a higher level of autonomy in underwater operations, as ROVs have unique abilities in terms of payload capacity and power access.



## Chapter 2

# Mathematical Modelling of ROVs

The mathematical modelling of the ROV is represented using Fossen's Robotic-like Vectorial model [4]. To fully state the mathematical model, a brief description of the applied kinematics and relevant notations are given. The theory reproduced in this Chapter is based on [4].

### 2.1 Notation and Reference Frames

In order to describe the ROV motion in 6 DOF, the SNAME 1950 notation is used. An overview of this notation, concerning motion due to forces/moments and resulting positions/orientations and velocities are given in Table 2.1. The orientation is given in Euler Angles.

Table 2.1: SNAME 1950 Notation

DOF		Forces/Moments	Velocities	Position/Euler Angles
1	translation in x-direction (surge)	X	u	x
2	translation in y-direction (sway)	Y	v	y
3	translation in z-direction (heave)	Z	w	z
4	rotation about the x-axis (roll)	K	p	$\phi$
5	rotation about the y-axis (pitch)	M	q	$\theta$
6	rotation about the z-axis (yaw)	N	r	$\psi$

Two reference frames are used to describe these motions:

- The North-East-Down (NED) coordinate system,  $\{n\} = (x_n, y_n, z_n)$ , where the axes respectively are aligned with the north -, east - and down direction.
- The body-fixed (BODY) reference frame,  $\{b\} = (x_b, y_b, z_b)$ , where the axes respectively point in the forward -, port-starboard and top-bottom direction of the vessel.

The NED-frame is a fixed reference frame located on the surface of the earth, rotating relative to the center of the earth. The BODY-frame is a moving reference frame attached to the vehicle that in question, as shown in Figure 2.1.

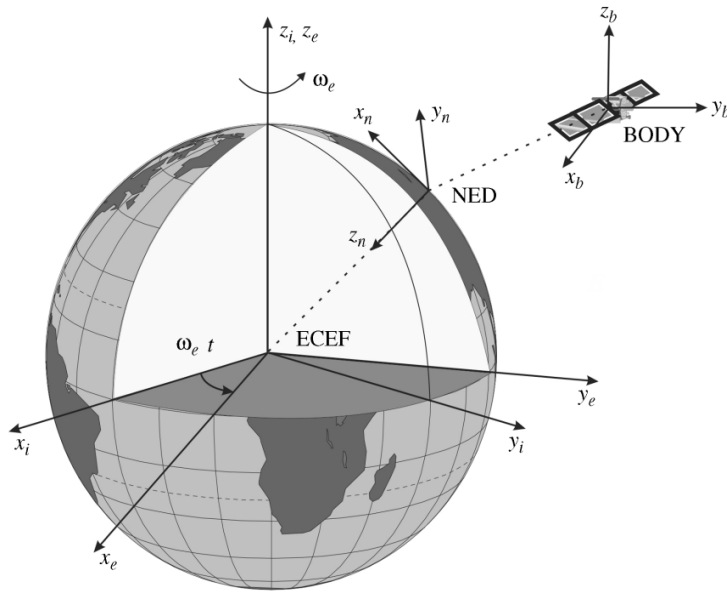


Figure 2.1: The NED- and BODY-frame relative to the Earth-centered Earth-fixed (ECEF) frame[4]

Furthermore, the notation given in (2.1a)-(2.1d) is used for the NED position and - orientation and the BODY linear and angular velocities, respectively.

$$\mathbf{p}^n = [x \ y \ z]^T \quad (2.1a)$$

$$\Theta_{nb} = [\phi \ \theta \ \psi]^T \quad (2.1b)$$

$$\mathbf{v}^b = [u \ v \ w]^T \quad (2.1c)$$

$$\boldsymbol{\omega}^b = [p \ q \ r]^T \quad (2.1d)$$

The body-fixed forces,  $\mathbf{f}^b$ , and - moments,  $\mathbf{m}^b$ , are given in the force vector,  $\boldsymbol{\tau}$  given in (2.2).

$$\boldsymbol{\tau} = [\mathbf{f}^b \ \mathbf{m}^b]^T = [X \ Y \ Z \ K \ N \ M]^T \quad (2.2)$$

By implementation of this notation, the next step is to state the necessary kinematics for modelling of an ROV operating in 6 DOF.

## 2.2 Kinematics

The relation between position and orientation in the NED-frame and the linear and angular velocities in the BODY-frame is given in (2.3).

$$\dot{\boldsymbol{\eta}} = \begin{bmatrix} \dot{\mathbf{p}}^n \\ \dot{\Theta}_{nb} \end{bmatrix} = \begin{bmatrix} \mathbf{R}_b^n(\Theta_{nb}) & \mathbf{0}_{3 \times 3} \\ \mathbf{0}_{3 \times 3} & \mathbf{T}_{\Theta}(\Theta_{nb}) \end{bmatrix} \cdot \begin{bmatrix} \mathbf{v}^b \\ \boldsymbol{\omega}^b \end{bmatrix} = \mathbf{J}(\boldsymbol{\eta})\boldsymbol{\nu}, \quad (2.3)$$

where  $\boldsymbol{\eta}$  and  $\boldsymbol{\nu}$  are the generalized position and velocity coordinates given in (2.4a)-(2.4b).

$$\boldsymbol{\eta} = [x \ y \ z \ \phi \ \theta \ \psi]^T \quad (2.4a)$$

$$\boldsymbol{\nu} = [u \ v \ w \ p \ q \ r]^T \quad (2.4b)$$

Moreover, the rotation matrix  $\mathbf{R}_b^n(\boldsymbol{\Theta}_{nb}) \in \mathbb{R}^{3 \times 3}$  represents the relationship between linear velocities in NED and BODY. This matrix is given in (2.5).

$$\mathbf{R}_b^n(\boldsymbol{\Theta}_{nb}) = \mathbf{R}_{x,\phi} \cdot \mathbf{R}_{y,\theta} \cdot \mathbf{R}_{z,\psi}, \quad (2.5)$$

where

$$\mathbf{R}_{x,\phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi \\ 0 & \sin \phi & \cos \phi \end{bmatrix}, \quad \mathbf{R}_{y,\theta} = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix}, \quad \mathbf{R}_{z,\psi} = \begin{bmatrix} \cos \psi & -\sin \psi & 0 \\ \sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.6)$$

The transformation matrix  $\mathbf{T}_\Theta(\boldsymbol{\Theta}_{nb}) \in \mathbb{R}^{3 \times 3}$  represent the relationship between the angular velocities in BODY and the orientation rates in NED. This matrix is given in (2.7).

$$\mathbf{T}_\Theta(\boldsymbol{\Theta}_{nb}) = \begin{bmatrix} 1 & \sin \phi \tan \theta & \cos \phi \tan \theta \\ 0 & \cos \phi & -\sin \phi \\ 0 & \sin \phi / \cos \theta & \cos \psi / \cos \theta \end{bmatrix} \quad (2.7)$$

## 2.3 Kinetics

The 6 DOF nonlinear equation of motion for a ROV is expressed in (2.8).

$$\underbrace{\mathbf{M}_{RB}\dot{\boldsymbol{\nu}} + \mathbf{C}_{RB}(\boldsymbol{\nu})\boldsymbol{\nu}}_{\text{rigid-body kinetics}} + \underbrace{\mathbf{M}_A\dot{\boldsymbol{\nu}}_r + \mathbf{C}_A(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r + \mathbf{D}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r}_{\text{hydrodynamic terms}} + \underbrace{\mathbf{g}(\boldsymbol{\eta})}_{\text{hydrostatic term}} = \boldsymbol{\tau} + \boldsymbol{\tau}_{\text{ext}}, \quad (2.8)$$

The equation contains terms concerning rigid-body kinetics, hydrodynamics forces and moments, and hydrostatic restoring forces and moments on the left-hand side. The right-hand side of the equation contains external forces and control inputs. This Section contains a more detailed description of all the terms.

In (2.8),  $\mathbf{M}_{RB} \in \mathbb{R}^{6 \times 6}$  is the rigid body inertia matrix and  $\mathbf{C}_{RB}(\boldsymbol{\nu}) \in \mathbb{R}^{6 \times 6}$  is the Coriolis-centripetal matrix.  $\mathbf{M}_A \in \mathbb{R}^{6 \times 6}$  is the added mass matrix,  $\mathbf{C}_A(\boldsymbol{\nu}_r) \in \mathbb{R}^{6 \times 6}$  is the added mass Coriolis-centripetal matrix and  $\mathbf{D}(\boldsymbol{\nu}_r) \in \mathbb{R}^{6 \times 6}$  is the damping matrix. The hydrodynamic terms takes into account the relative velocity,  $\boldsymbol{\nu}_r$ , which is given in (2.9).

$$\boldsymbol{\nu}_r = \boldsymbol{\nu} - \boldsymbol{\nu}_c, \quad (2.9)$$

where  $\boldsymbol{\nu}_c \in \mathbb{R}^{6 \times 1}$  is the velocity properties of the current acting on the ROV in the BODY-frame. This vector is expressed in (2.10), on a generalized velocity coordinate form.

$$\boldsymbol{\nu}_c = [u_c \quad v_c \quad w_c \quad 0 \quad 0 \quad 0]^T \quad (2.10)$$

Moreover, in (2.8),  $\mathbf{g}(\boldsymbol{\eta}) \in \mathbb{R}^{6 \times 1}$  is a vector of gravitational/buoyancy forces and moments. On the right-hand side of the equation,  $\boldsymbol{\tau} \in \mathbb{R}^{6 \times 1}$  is the vector of control inputs, in this case, propulsion forces and moments. Furthermore,  $\boldsymbol{\tau}_{ext} \in \mathbb{R}^{6 \times 1}$  is the vector of external forces acting on the ROV. In underwater operations using ROVs, relevant external forces are, for example, forces and moments due to the umbilical which the ROV is tethered to and forces and moments due to the manipulator arm perturbing the ROV as might be the case if the manipulator arm is not modelled exactly.

### 2.3.1 Rigid-Body Kinetics

Recall the rigid-body terms from (2.8),  $\mathbf{M}_{RB}\dot{\boldsymbol{\nu}} + \mathbf{C}_{RB}(\boldsymbol{\nu})\boldsymbol{\nu}$ , - in this subsection the rigid-body forces and moments will be revisited with a higher level of detail.

Naturally, in the mathematical modelling it is assumed that the ROV can be considered a rigid body, as it will not be deformed when forces and moments are acting on it. For underwater vehicles, it is often assumed homogeneous mass distribution and xz-plane symmetry, that is, starboard-port symmetry. Such an assumption simplifies the rigid-body inertia matrix,  $\mathbf{M}_{RB}$  since it implies that some moments of inertia are equal to zero, as expressed in (2.11).

$$I_{xy} = I_{yz} = 0 \quad (2.11)$$

By placing the origin of the BODY-frame, i.e. the body-fixed reference point CO, in the centre of gravity in the y-direction,  $y_g = 0$ , the vector  $\mathbf{r}_g^b$  describes the distance from CO to the centre of gravity (COG) in the BODY-frame as given in (2.12).

$$\mathbf{r}_g^b = [x_g \quad 0 \quad z_g]^T \quad (2.12)$$

Furthermore, the mass of the ROV Minerva 2 is denoted  $m$  and the inertia matrix is denoted  $\mathbf{I}_b$ . The skew-symmetric matrix notation,  $\mathbf{S}(\mathbf{r}_g^b)m$ , and the  $\mathbf{r}_g^b$ -vector are used when expressing the rigid-body inertia matrix, such that it takes the form given in (2.13) [4].

$$\mathbf{M}_{RB} = \begin{bmatrix} m\mathbf{I}_{3 \times 3} & -m\mathbf{S}(\mathbf{r}_g^b) \\ m\mathbf{S}(\mathbf{r}_g^b) & \mathbf{I}_b \end{bmatrix} = \begin{bmatrix} m & 0 & 0 & 0 & mz_g & -my_g \\ 0 & m & 0 & mz_g & 0 & mx_g \\ 0 & 0 & m & my_g & -mx_g & 0 \\ 0 & mz_g & -my_g & I_x & 0 & -I_{xz} \\ mz_g & 0 & mx_g & 0 & I_y & 0 \\ my_g & -mx_g & 0 & -I_{zx} & 0 & I_z \end{bmatrix}, \quad (2.13)$$

where  $I_{ij}$  is the different moments of inertia about the origin of the BODY-frame.

Moreover, when the BODY-frame is rotated relative to the NED-frame inertial forces and moments due to this rotation need to be taken into account in the rigid-body kinetics. Recall the rigid-body Coriolis-centripetal matrix from (2.8),  $\mathbf{C}_{RB}(\boldsymbol{\nu})$ , which represents these forces and moments. Note that the assumptions stated in (2.11) and (2.12) also applies here, thus  $\mathbf{C}_{RB}(\boldsymbol{\nu})$  given in [4] reduces to the matrix given in (2.14).



$$\mathbf{C}_{RB}(\boldsymbol{\nu}) = \begin{bmatrix} 0 & 0 & 0 & mz_g r & -m(x_g q - w) & -m(x_g r + v) \\ 0 & 0 & 0 & -mw & m(z_g r + x_g p) & mu \\ 0 & 0 & 0 & -m(z_g p - v) & -m(z_g q + u) & mx_g p \\ -mz_g r & mw & m(z_g p - v) & 0 & -I_{xz} p + I_z r & -I_y q \\ m(x_g q - w) & -m(z_g r + x_g p) & m(z_g q + u) & I_{xz} p - I_z r & 0 & -I_{xz} r + I_x p \\ m(x_g r + v) & -mu & -mx_g p & I_y q & I_{xz} r - I_x p & 0 \end{bmatrix} \quad (2.14)$$

### 2.3.2 Hydrostatic Terms

A rigid body submerged in water, such as the ROV Minerva 2, is influenced by a buoyancy force and a gravitational force - recall the term  $\mathbf{g}(\boldsymbol{\eta})$  from (2.8). The buoyancy force,  $B$ , is given by the volume of the displaced water,  $\nabla$ , the density of the water,  $\rho$ , and the gravitational acceleration,  $g$ , as shown in (2.15).

$$B = \rho g \nabla \quad (2.15)$$

The gravitational force,  $W$  is given as shown in (2.16).

$$W = mg \quad (2.16)$$

In the NED-frame, the forces  $B$  and  $W$  act in the vertical plane, respectively in negative and positive  $z_n$ -direction. Recall the relationship between the center of gravity (COG) and the origin of the BODY-frame (CO), given in (2.12). Similarly, the relation between the centre of buoyancy (COB) and CO is given in (2.17).

$$\mathbf{r}_{buo}^b = [x_{buo} \quad y_{buo} \quad z_{buo}]^T \quad (2.17)$$

Naturally, in the BODY-frame, change in attitude affects the direction of the forces  $B$  and  $W$ . The buoyancy force acts in the center of buoyancy (COB), while the gravitational force acts on the center of gravity (COG). If the COB and the COG do not coincide with each other, moments governs in the BODY-frame. Thus, the term  $\mathbf{g}(\boldsymbol{\eta})$  has to be given in accordance with the relation between the NED- and the BODY-frame given in Section 2.2, as shown in (2.18). Note that the assumptions stated in (2.11) and (2.12) also applies here.

$$\mathbf{g}(\boldsymbol{\eta}) = \begin{bmatrix} (W - B)\sin(\theta) \\ -(W - B)\cos(\theta)\sin(\phi) \\ -(W - B)\cos(\theta)\cos(\phi) \\ y_b B \cos(\theta)\cos(\phi) + (z_g W - z_b B)\cos(\theta)\sin(\phi) \\ (z_g W - z_b B)\sin(\theta) + (x_g W - x_b B)\cos(\theta)\cos(\phi) \\ -(x_g W - x_b B)\cos(\theta) + y_b B \sin(\theta) \end{bmatrix} \quad (2.18)$$

Underwater vehicles are often designed such that they are slightly positively buoyant, i.e. the magnitude of  $B$  is slightly larger than the magnitude of  $W$ , such that the vehicle surfaces automatically in the case of failure.

### 2.3.3 Hydrodynamic Terms

Recall the hydrodynamic terms from (2.8),  $\mathbf{M}_A \dot{\boldsymbol{\nu}}_r + \mathbf{C}_A(\boldsymbol{\nu}_r) \boldsymbol{\nu}_r + \mathbf{D}(\boldsymbol{\nu}_r) \boldsymbol{\nu}_r$  - in this subsection the hydrodynamic forces and moments will be revisited with a higher level of detail.

Firstly, the added mass matrix,  $\mathbf{M}_A$ , which is proportional to the relative acceleration,  $\boldsymbol{\nu}_r$ , can be seen as the additional mass of the fluid that put into motion when the vessel is moving. Assuming that the ROV will operate below the wave-affected zone such that it is not necessary to model wave-induced forces and moments. This assumption simplifies the mathematical model since it results in the ROV not having a time-varying added mass, as it will if there are significant wave-induced forces and moments acting on it. In other words, the added mass matrix  $\mathbf{M}_A$  is constant, stated mathematically in (2.19).

$$\mathbf{M}_A = \mathbf{A}(\omega) = \text{constant} \quad (2.19)$$

Similarly, the added mass Coriolis and centripetal matrix,  $\mathbf{C}_A(\boldsymbol{\nu}_r)$ , is proportional to the relative velocity,  $\boldsymbol{\nu}_r$ . The forces and moments in this matrix represent forces and moments due to the movement of the added mass of the vessel in the BODY-frame relative to the NED-frame.

Lastly, the damping matrix,  $\mathbf{D}(\boldsymbol{\nu}_r)$  is also proportional to the relative velocity,  $\boldsymbol{\nu}_r$ . This matrix represents forces and moments related to energy transported away from the vessel.

The matrices  $\mathbf{M}_A$ ,  $\mathbf{C}_A(\boldsymbol{\nu}_r)$  and  $\mathbf{D}(\boldsymbol{\nu}_r)$  are large matrices which are given in more detail in [4].

## Chapter 3

# ROV Control, Guidance and Navigation

### 3.1 ROV Minerva 2

As already mentioned, this thesis is based on the AUR-Lab property ROV Minerva 2. In this Chapter theory behind the development of the control system for ROV Minerva 2 is given. Sections 3.2-3.6 addresses essential topics for autonomous underwater docking that are implemented in the control system over the years in accordance with publications addressed in Section 1.5.1. Section 3.7 has a higher degree of novelty as it proposes the properties of the optimal path planning problem for autonomous docking missions, which are parts of the contributions from this thesis. Before defining the optimal path planning problem for autonomous docking applications, topics like underwater navigation and an overview of the control system focusing on control, state estimation, and guidance are addressed throughout this Chapter. Moreover, the path planner aiming to solve the optimal path planning problem for autonomous docking applications is considered a part of the autonomy module in the control system.

#### 3.1.1 Acoustic Navigation Specifications

The ROV Minerva 2 is equipped with sensors to keep track of navigation. Measurements from these sensors act as input into the observer in order to estimate the states of the ROV. This Section aims to give an overview of essential sensors which the ROV Minerva 2 is equipped with.

The acoustic positioning system (APS) is able to track Minerva 2 due to a transponder mounted on the ROV. The transponder exchanges signals with the APS, such that the global  $x$ ,  $y$  and  $z$  coordinates of the transponder are determined. Minerva 2 is deployed from RV Gunnerus, equipped with a Kongsberg HiPAP, which is a Super Short Baseline (SSBL) APS[7]. SSBL systems consist of a multi-element transducer, using range and angle measurements to determine the position of the transponder. Figure 3.1 illustrates the principle of acoustic SSBL navigation.

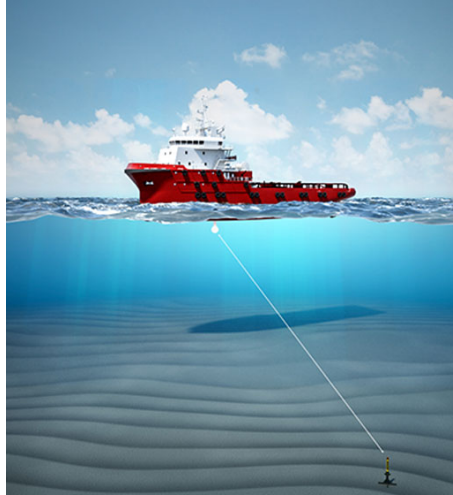


Figure 3.1: SSBL Underwater Positioning[5]

The DVL enables Minerva 2 to keep track of its linear velocities,  $u$ ,  $v$  and  $w$ , based on the Doppler shift in the echo of the acoustic signals sent from the DVL occurring when the signals are reflected from the seafloor or the water column [7]. Additionally, the DVL enables the ROV to keep track of the vertical distance down to the seafloor underneath locally, called the altitude. Minerva 2 is equipped a DVL providing four acoustic beams keeping track of its altitude and linear velocities.

Minerva 2 is equipped with an inertial measurement unit (IMU). The IMU is a sensor consisting of a 3-axis accelerometer, a 3-axis, and a 3-axis magnetometer, which respectively enables measuring linear accelerations, angular velocities and magnetic field components such as heading.

The depth of Minerva 2 is measured by a pressure gauge sensor, enabling the calculation of depth based on the measured pressure.

The common denominator in underwater navigation is that these sensors all are prone to noise and local variations in the environment in which they are operating in, and thus sensor fusion and proper state estimation are challenges. The update rates of the navigation sensors are also crucial in terms of navigation accuracy. Thus, the transit speed is constrained during operations; for instance, in constant altitude control, the DVL is only able to look so far ahead. If the transit speed is too high, there will be some time delay, and the risk for collision with the seabed can become critical. On that note, the transit speed should be chosen such that there is some safety margin on available thrust/thruster rpm in case of local variations or unexpected events in the environment. Also, Minerva 2 is equipped with cameras and sonars, which can play a role in visual navigation as well as mission-related tasks. The performance of the sonars and cameras are also factors that should be considered when determining the transit velocity. Previous simulations and field trials have shown that a transit speed of 0.2-0.3  $m/s$  is suitable for most ROV underwater operations.

The resulting measurements from the ROV Minerva 2 navigation sensors are summed up in Table 3.1, and these measurements act as input to state estimation.

Table 3.1: Available measurements based on sensors

Sensor		Available Measurement
Acoustic Positioning System (APS)	3-DOF positions	$x, y, z$
Doppler Velocity Logger (DVL)	3-DOF linear velocities	$u, v, w$
Accelerometer	3-DOF linear accelerations	$\dot{\mathbf{v}}^b$
Gyroscopes	3-DOF angular velocities	$p, q, r$
Magnetometer	Heading	$\psi$

## 3.2 Control System Overview

Figure 3.2 aims to provide an overview of the ROV motion control system. In the following Sections, some of the modules in the Figure are given further attention. In general, the user is controlling the ROV from the top-side of the architecture through a graphical user interface (GUI) which is giving the user relevant system parameters. The control system then takes input through the guidance module, either via joystick control, waypoint data, or truly autonomously. The behavior of the ROV on the bottom-side is then initiated, through control signals based on estimated states and measurements by sensors on-board of the ROV.

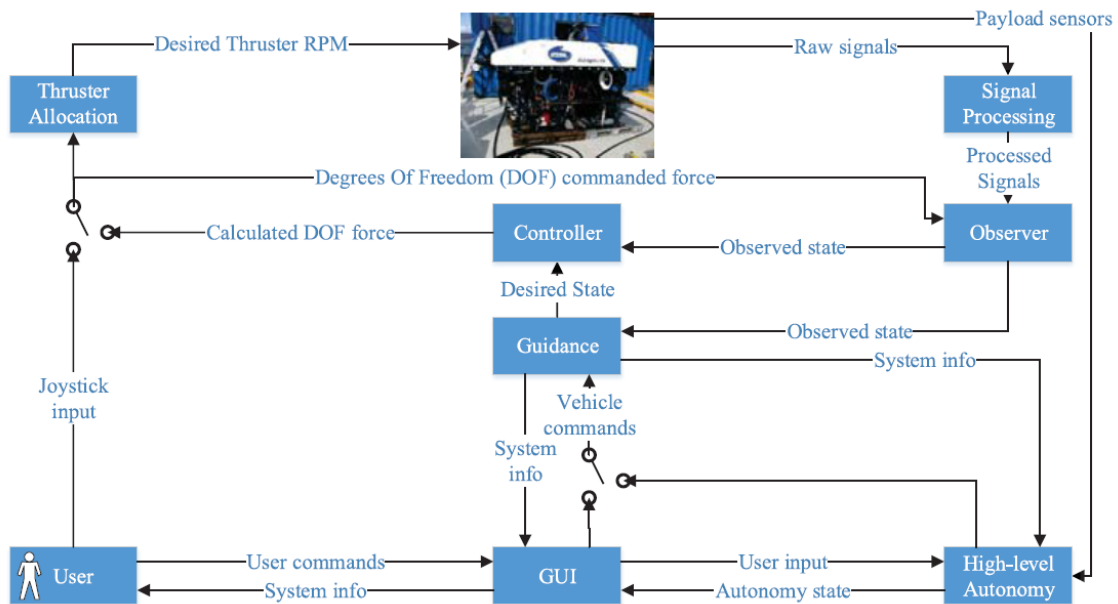


Figure 3.2: ROV Motion Control System Architecture [6]

The controller, observer and guidance module are considered key when components in autonomous docking missions, and thus, they are further addressed in Sections 3.3, 3.4 and 3.5. Furthermore, the path planner can be reviewed as a part of the content in the high-level autonomy module, and thus, the optimal path planning problem, which is a part of this thesis contribution, is proposed in Section 3.7.

### 3.3 Controller

The controller calculates control signals in order to obtain the desired states based on measurements and estimated states. As covered in Section 1.5.1, over the years different controllers have been implemented and tested in the ROV motion control system. However the default implement controller is a nonlinear proportional-integral-derivative (PID) combined with a feed-forward term [7]. A walk-through of the controller proposed in [7] is presented in the following paragraphs. The control vector is given in (3.1).

$$\boldsymbol{\tau} = \boldsymbol{\tau}_{PID} + \boldsymbol{\tau}_{FF}, \quad (3.1)$$

where the nonlinear PID control vector,  $\boldsymbol{\tau}_{PID}$ , is given according to the kinematics given in (2.3), as expressed in(3.2).

$$\boldsymbol{\tau}_{PID} = -\mathbf{J}^T(\boldsymbol{\eta}) \left( \mathbf{K}_p \tilde{\boldsymbol{\eta}} + \mathbf{K}_d \mathbf{J}(\boldsymbol{\eta}) \boldsymbol{\nu} + \mathbf{K}_i \int_0^t \tilde{\boldsymbol{\eta}}(t) d\tau \right), \quad (3.2)$$

where the matrices  $\mathbf{K}_p$ ,  $\mathbf{K}_d$  and  $\mathbf{K}_i \in \mathbb{R}^{6 \times 6}$  respectively are the proportional, integral and derivative gain matrices, assuming that the ROV is controlled in all 6 DOFs. Additionally, the integral term includes a saturation term to avoid integral wind-up. The tracking error  $\tilde{\boldsymbol{\eta}}$  is given as the deviation between the measured states,  $\boldsymbol{\eta}$  and the desired states  $\boldsymbol{\eta}_d$ , as expressed in (3.3).

$$\tilde{\boldsymbol{\eta}} = \boldsymbol{\eta} - \boldsymbol{\eta}_d \quad (3.3)$$

The feed-forward term is given in (3.4), based on copy-dynamics of (2.8) and the desired states  $\boldsymbol{\eta}_d$  and  $\boldsymbol{\nu}_d$ .

$$\boldsymbol{\tau}_{FF} = \mathbf{M} \dot{\boldsymbol{\nu}}_d + \mathbf{C}(\boldsymbol{\nu}_d) \boldsymbol{\nu}_d + \mathbf{D}(\boldsymbol{\nu}_d) \boldsymbol{\nu}_d, \quad (3.4)$$

where  $\mathbf{M} = \mathbf{M}_{RB} + \mathbf{M}_A$  and  $\mathbf{C}(\boldsymbol{\nu}_d) = \mathbf{C}_{RB}(\boldsymbol{\nu}_d) + \mathbf{C}_A(\boldsymbol{\nu}_d)$ .

### 3.4 Observer

An observer, or state estimator, reconstruct the states of the system and can even reconstruct unmeasured states. Similarly as for the controller, several implementations and tests have been carried out for different observers. However, the default implemented observer is an Extended Kalman Filter (EKF), which has the advantageous ability to take nonlinear dynamics into account. The discrete-time EKF is based on a copy of the discretization of the control plant model on the form given in (3.5)[4].

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{B}\mathbf{u} + \mathbf{E}\mathbf{w} \quad (3.5a)$$

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{v} \quad (3.5b)$$

The state vector  $\mathbf{x}$  represents the states which are to be reconstructed as estimates,  $\hat{\mathbf{x}}$  by the EKF, while  $\mathbf{y}$  is the measurement vector of these states. The control plant model is typically obtained

by writing the kinematics, expressed in (2.3), and the kinetics, expressed (2.8), in the form given in (3.5). The kinetics can, however, be simplified while still regarding all 6 DOF, a simplification that correspondingly can be accounted for by a bias-model [32]. The kinematics, kinetics, and the bias-model then all contribute to the dynamics  $\mathbf{f}(\mathbf{x})$ . The vectors  $\mathbf{w}$  and  $\mathbf{v}$  are white noise vectors, respectively representing process noise and measurement noise, according to assumptions for implementation of the EKF[4]. The matrix  $\mathbf{H}$  describes which measurements are available for the observer. Then, the state estimate update at time instant  $k$  is given in (3.6a), based on the current state estimation propagation,  $\bar{\mathbf{x}}(k)$ . During every recursion, the state estimate propagation,  $\bar{\mathbf{x}}(k+1)$ , is updated beforehand of the next iteration as shown in (3.6b)[4].

$$\hat{\mathbf{x}}(k) = \bar{\mathbf{x}}(k) + \mathbf{K}_{KF}(k) [\mathbf{y}(k) - \mathbf{H}(k)\bar{\mathbf{x}}] \quad (3.6a)$$

$$\bar{\mathbf{x}}(k+1) = \hat{\mathbf{x}}(k) + h [\mathbf{f}(\hat{\mathbf{x}}(k)) + \mathbf{B}\mathbf{u}(k)], \quad (3.6b)$$

where  $h > 0$  is the discretization sampling time and  $\mathbf{K}_{KF}(k)$ , is the Kalman gain matrix, given as expressed in (3.7)[4].

$$\mathbf{K}_{KF}(k) = \bar{\mathbf{P}}(k)\mathbf{H}^T(k) [\mathbf{H}(k)\bar{\mathbf{P}}(k)\mathbf{H}^T(k) + \mathbf{R}(k)]^{-1}, \quad (3.7)$$

Where  $\bar{\mathbf{P}}(k)$  is the error of the covariance propagation, which is calculated for each time instant  $k$  based on the error covariance update,  $\hat{\mathbf{P}}(k)$ , as shown in (3.8a)-(3.8b)[4].

$$\hat{\mathbf{P}}(k) = [\mathbf{I} - \mathbf{K}_{KF}(k)\mathbf{H}(k)] \bar{\mathbf{P}}(k) [\mathbf{I} - \mathbf{K}_{KF}(k)\mathbf{H}(k)]^T \quad (3.8a)$$

$$\bar{\mathbf{P}}(k+1) = \Phi(k)\hat{\mathbf{P}}(k)\Phi^T(k) + h^2\mathbf{E}(k)\mathbf{Q}(k)\mathbf{E}^T(k) \quad (3.8b)$$

The matrices  $\mathbf{Q}(k) = \mathbf{Q}^T(k) > 0$  and  $\mathbf{R}(k) = \mathbf{R}^T(k) > 0$  are the process noise and measurement noise covariance matrices, respectively. The matrix  $\Phi(k)$  is the discrete-time linearized system matrix obtained, for instance, as shown in (3.9)[4].

$$\Phi(k) = \mathbf{I} + h \frac{\partial \mathbf{f}(\hat{\mathbf{x}}(k))}{\partial \hat{\mathbf{x}}} \quad (3.9)$$

Summing up, the discrete-time EKF algorithm is a recursive process carried out for every time instant  $k = 0, 1, 2, \dots, N$ , initiated by the initial conditions  $\bar{\mathbf{x}}(0) = \bar{\mathbf{x}}_0$  and  $\bar{\mathbf{P}}(0) = \bar{\mathbf{P}}_0$ , followed by the calculations in equations (3.7), (3.6a), (3.8a), (3.6b) and (3.8b), in that order, for every iteration[4].

## 3.5 Guidance

Following Figure 3.2, the guidance module generates the desired states for the system based on trajectory waypoints generated by the optimal path planner integrated into the high-level autonomy module, described in detail in Section 3.7.

The guidance module should provide feasible trajectories for the vehicle. To ensure that the desired trajectory is obtained, guidance laws are defined according to the control objectives. Due to the over-actuated nature of Minerva 2, constant jerk guidance for position reference is a suitable guidance scheme. Constant jerk is a waypoint based guidance scheme, claimed to be beneficial as the generated waypoint based ROV trajectory does not require any further refinement, and thus, savings in terms of computational power which can be utilized at other stages of the control sequence[9].

### 3.5.1 Constant Jerk Guidance for Position Reference

Constant jerk guidance for position reference is an algorithm ensuring smooth transitions in position and velocity, while generating the fastest path between two waypoints,  $\mathbf{p}_k$  and  $\mathbf{p}_{k+1}$ . The algorithm takes into account physical constraints of the systems, that is, velocity,  $v$  [ $m/s$ ], acceleration,  $a$  [ $m/s^2$ ], and jerk  $j$  [ $m/s^3$ ], ensuring that the system does not exceed its capabilities in terms of maximum velocity, acceleration and jerk,  $v_{max}$ ,  $a_{max}$  and  $j_{max}$ [33].

**Assumption:** Assuming known constraints  $v_{max}$ ,  $a_{max}$  and  $j_{max}$  and the total distance  $d_{tot}$  between  $\mathbf{p}_k$  and  $\mathbf{p}_{k+1}$ , the following holds for the constant jerk reference model[33]:

$$|v| \leq v_{max} \quad (3.10a)$$

$$|a| \leq a_{max} \quad (3.10b)$$

$$|j| \leq j_{max} \quad (3.10c)$$

Further, the trajectory from  $\mathbf{p}_k$  to  $\mathbf{p}_{k+1}$  follows up to seven phases depending of the step in position between  $\mathbf{p}_k$  and  $\mathbf{p}_{k+1}$ . The seven phases are the following[33]:

- Phase 1: Start acceleration - constant jerk for maximum increase in acceleration.
- Phase 2: Constant acceleration - constant acceleration for maximum increase in velocity.
- Phase 3: End acceleration - constant jerk for maximum decrease in acceleration towards zero.
- Phase 4: Constant velocity - main transit phase at maximum velocity.
- Phase 5: Start deceleration - constant jerk for maximum increase in deceleration.
- Phase 6: Constant deceleration - constant acceleration for maximum decrease in velocity towards zero.
- Phase 7: End deceleration - constant jerk for maximum decrease in deceleration towards zero.

Figure 3.3[7] illustrates these seven phases, aiming to depict the correlation between jerk, acceleration, velocity, and position. Depending on the length of the step in position between  $\mathbf{p}_k$  and  $\mathbf{p}_{k+1}$ , some of the phases may be omitted, due to the dynamics of the physical system that is the vehicle.



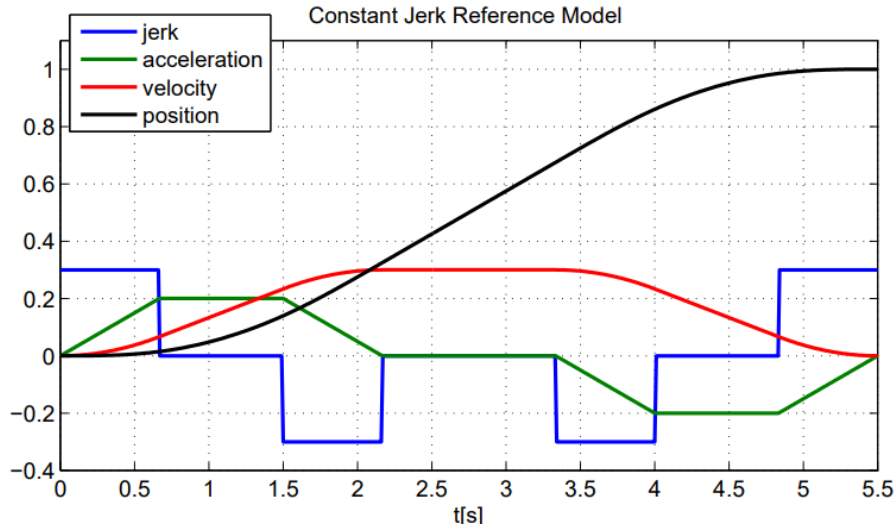


Figure 3.3: Correlation between jerk, acceleration, velocity and position [7]

Next, the algorithm provides several output parameters based on the known input parameters  $v_{max}$ ,  $a_{max}$ ,  $j_{max}$  and  $d_{tot}$ . These output parameters are describing distances,  $d_i$ , velocities,  $v_i$ , and time instants,  $t_i$ , describing the different phases of the transition between  $\mathbf{p}_k$  and  $\mathbf{p}_{k+1}$ . When these parameters are established, the time, jerk, acceleration, velocity, and position for each phase in the algorithm are calculated, enabling the transit. Moreover, [33] describes the algorithm in general.

In practice, the application of this position referencing methodology for Minerva 2 implies that between every waypoint generated by the path planner, the vehicle situates its heading towards the next waypoint,  $\mathbf{p}_{k+1}$ . Depending on the step in position from the current waypoint,  $\mathbf{p}_k$ , to  $\mathbf{p}_{k+1}$ , the algorithm goes through up to all of the seven phases described above. When Minerva 2 reaches the waypoint  $\mathbf{p}_{k+1}$ , the ROV takes on zero forward velocity as it will, by its over-actuated nature, stay at the fixed (N,E)-position and change its heading towards the waypoint  $\mathbf{p}_{k+2}$ . When the new heading is obtained, the algorithm runs all over again. Note that when the ROV is adjusting its altitude/depth, the pitch and roll then can be assumed constant. Thus, the ROV only adjusts its heading in two dimensions.

## 3.6 Altitude Control

Altitude control is a control scheme shown to be useful in autonomous docking missions. In altitude control, the altitude is automatically controlled to follow the terrain at a fixed altitude. In Section 5.6.2, an autonomy mode called auto-altitude mode refers to autonomous docking missions utilizing altitude control. Moreover, details on the development of altitude control for the AUR-Lab ROVs is given in [7]. A brief walk-through of altitude control based on [7] is given in this Section to give relevant insight into its application in autonomous docking operations.

Initially, to perform altitude control, seafloor approximation is carried out utilizing the DVL. As already mentioned, Minerva 2 is equipped with a DVL providing four acoustic beams. In order to obtain a linear seafloor approximation, measurements from three of the four DVL beams are utilized. Figure 3.4 demonstrates how the DVL can be used to provide a linear seafloor approximation using measurements from three of those beams. In this Figure, the red lines represent the DVL beams, the white plane represents the linear seafloor approximation, while the blue and the green lines represent the estimated altitude and seafloor gradient. Thus, by application of

such methodology, the ROV is not only able to estimate the altitude but also the seafloor gradient for future state estimation. Additionally, visual inspection of Figure 3.4 yields that the seafloor approximation is prone to seafloor steepness in the most forward-pointing acoustic beam if the seafloor is too steep for this beam to obtain a reliable measurement.

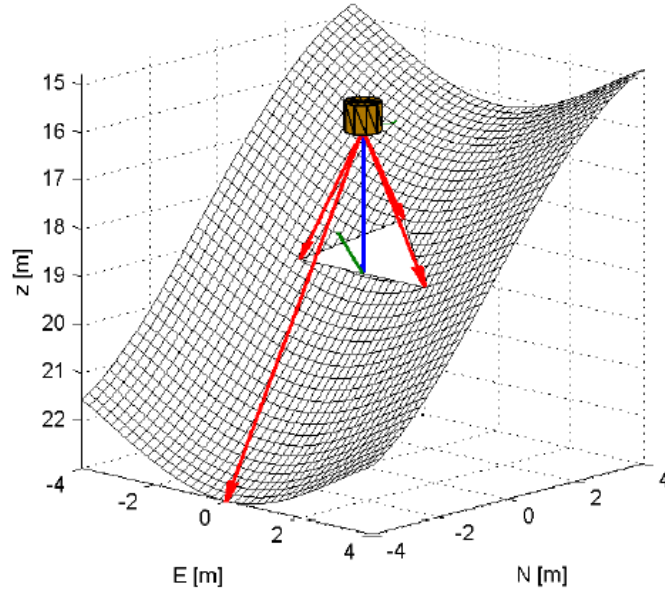


Figure 3.4: Linear seafloor approximation using DVL [7]

Furthermore, for conservative reasons, the seafloor approximation is based on the lowest DVL altitude measurement and the gradient from the least-squares method. In this context, assuming that the seafloor can be expressed as the surface given in (3.11), which has a continuous first-order derivative.

$$F(x, y, z) = f(x, y) - z = 0, \quad \frac{\partial F}{\partial t} = 0 \quad (3.11)$$

Moreover, assuming that the ROV is in the global NED-position  $\mathbf{p}^n = [x_p, y_p, z_p]^T$ , the altitude of the ROV at this time instant can be expressed as in (3.12a), and the altitude rate of change as in (3.12b).

$$a = f(x_p, y_p) - z_p = F(x_p, y_p, z_p) \quad (3.12a)$$

$$\dot{a} = \Delta F(\mathbf{p}^n) \cdot \dot{\mathbf{p}} \quad (3.12b)$$

Thus, by the approach illustrated in Figure 3.4, the linear seafloor approximation,  $\hat{z}_{\text{sf}}$  is given by the lowest DVL altitude measurement,  $a_{\text{min}}$  as expressed in (3.13).

$$\hat{z}_{\text{sf}} = a_{\text{min}} + bx + cy, \quad (3.13)$$

where  $b$  and  $c$  are coefficients.

Furthermore, due to noise issues and possible discontinuities in the seafloor, the need for an altitude observer arises. The altitude observer provides estimates of the approximated altitude and the

altitude rate of change. The altitude observer is implemented in cascade with the vehicle observer described in Section 3.4, because the altitude observer utilizes input as estimated vehicle orientation,  $\hat{\Theta}_{nb}$  and estimated vehicle linear velocities,  $\hat{\mathbf{v}}^b$  from the vehicle observer. Moreover, the altitude observer is implemented as a Kalman filter, taking on the same structure as the observer described in detail in Section 3.4.

Moreover, the altitude control is implemented by defining a guidance law providing the controller with the desired depth corresponding to the desired altitude. The guidance law is expressed in terms of depth, not altitude, since depth measurements usually are more reliable than altitude measurements. Depth is a state of the ROV and is continuous, while altitude can be discontinuous. A feed-forward term concerning the altitude rate of change included in the guidance law to enhance the performance of the system. Finally, the guidance law is applied to the controller. As mentioned, the methodology behind this is given in detail in [7].

### 3.7 Optimal Path Planning

Now, theory on optimal path planning is proposed, a methodology considered a part of the high-level autonomy module in Figure 3.2. This Section aims to define the optimal path planning problem enabling Minerva 2 to take on autonomous docking missions. The optimal path problem is defined to enable the high-level autonomy module to provide the guidance module with a waypoint based path fulfilling the desired properties.

A path,  $\mathcal{P}$ , consists of  $n$  waypoints including the initial position of the ROV,  $\mathbf{p}_0 = (x_0, y_0, z_0)$ , and the position of the docking station,  $\mathbf{p}_n = (x_n, y_n, z_n)$ .  $\mathbf{p}_0$  and  $\mathbf{p}_n$  are known, fixed points, while the interior waypoints,  $\mathbf{p}_1, \dots, \mathbf{p}_{n-1} = (x_1, y_1, z_1), \dots, (x_{n-1}, y_{n-1}, z_{n-1})$  are to be determined by the path planner. The number of interior waypoints, denoted  $K$ , is a design parameter. Thus, a mathematical representation of a path is expressed in (3.14).

$$\mathcal{P} = [\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_{n-1}, \mathbf{p}_n], \quad (3.14)$$

where  $\mathbf{p}_1, \dots, \mathbf{p}_{n-1}$  are the unknowns to be determined by the path planner. In addition, the environment is defined within global variable bounds in north, east and down directions, as show in (3.15).

$$N = [x_{\min}, \quad x_{\max}] \quad (3.15a)$$

$$E = [y_{\min}, \quad y_{\max}] \quad (3.15b)$$

$$D = [0, \quad z_{\text{seafloor}}] \quad (3.15c)$$

Furthermore, the environment contains static obstacles with known positions,  $\mathbf{p}_{\text{obstacle},i}$ , described mathematically in (3.16).

$$\mathbf{p}_{\text{obstacle},i} = (x_{\text{obstacle},i}, \quad y_{\text{obstacle},i}, \quad z_{\text{obstacle},i}) \quad (3.16)$$

For simplicity, obstacles are modelled as spheres with a known radius,  $r_{\text{obstacle}}$ .

To render the optimal path, the path planning problem is solved by optimization. One way to solve an optimization problem is subject to one or more objectives, assigning the optimal solution certain attributes. In such algorithms, the optimization problem is solved by minimizing optimization functions. The number of objectives is denoted  $M$ . Theory on optimization is given in Chapter 4.

When solving the path planning problem for underwater vehicles, the two clear objectives are ensuring safety in terms of collision avoidance with obstacles and minimizing the total path length. By minimizing the total path length, the travel time is likely minimized, assuming that the transit velocity is constant and thus obtaining minimum energy consumption. The reason behind collision avoidance with obstacles is obvious from a safety point of view. Additionally, two more objectives are being considered here. The first one concerns the avoidance of sharp turns in the xy-plane, aiming to minimize wear and tear on the actuators and producing a smoother trajectory. The latter is ensuring reliable DVL-measurements while avoiding collision with the seafloor, which is obtained by terrain following. These objectives are given more attention in the following Sections, where they are given a mathematical representation denoted  $O_1(\mathcal{P})$ ,  $O_2(\mathcal{P})$ ,  $O_3(\mathcal{P})$  and  $O_4(\mathcal{P})$ , respectively. The objectives are describes corresponding to the path  $\mathcal{P}$  in question.

### 3.7.1 Objective Function - Path Length

Given the mathematical representation of a path expressed in (3.14), the total length of the path consisting of  $n - 1$  path segments connecting the  $n$  waypoints is given as expressed in (3.17)[31].

$$O_1(\mathcal{P}) = \sum_{k=1}^{n-1} \sqrt{(x_{k+1} - x_k)^2 + (y_{k+1} - y_k)^2 + (z_{k+1} - z_k)^2} \quad (3.17)$$

The first objective function,  $O_1(\mathcal{P})$ , is given (3.17). The path planner should be able to calculate the numerical value of  $O_1(\mathcal{P})$ , and path candidates with lower values of  $O_1(\mathcal{P})$  are preferred.

### 3.7.2 Objective Function - Safety Margin

The second objective function,  $O_2(\mathcal{P})$ , concerns collision avoidance with obstacles in the operational environment. The objective function value is penalized if the path is closer than a safety margin,  $m$  to the obstacle. The closer the path is to the obstacle, the more the objective function value approaches zero. On the other hand, the objective function takes a negative value if the path is more than a safety margin away from the obstacle. The further away from the obstacle the path is, the more negative the objective function value. Given the mathematical representation of the path given in (3.14),  $O_2(\mathcal{P})$  is mathematically expressed in (3.18)[31].

$$O_2(\mathcal{P}) = \begin{cases} m - r_{\min}, & \text{if } r_{\min} > m \\ e^{r_{\min} - m}, & \text{otherwise} \end{cases} \quad (3.18)$$

where the safety margin  $m$  is a design parameter. Moreover,  $r_{\min}$  is the closest distance from any path segment spanned between two adjacent waypoints  $\mathbf{p}_k$  and  $\mathbf{p}_{k+1}$  and an obstacle with known position  $\mathbf{p}_{obstacle}$ , as shown in Figure 3.5.

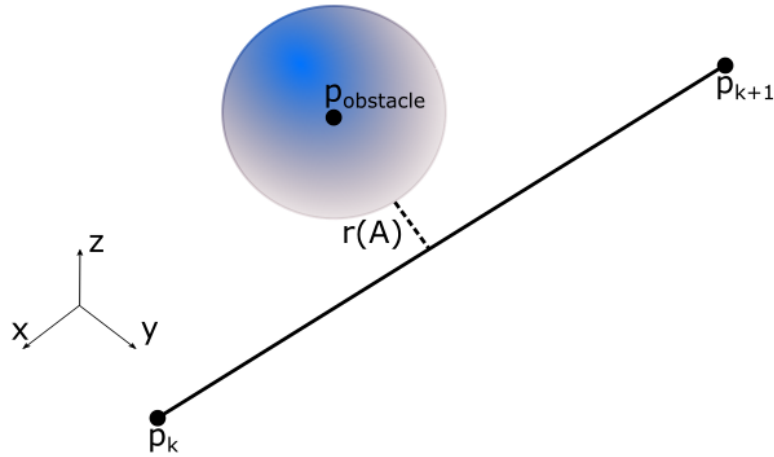


Figure 3.5: Illustration of the closest distance between an obstacle and any path segment

As mentioned, the obstacles are assumed to have known positions and are being modelled as spheres, with a known radius. The value of  $r_{\min}$  is being calculated as expressed in (3.19)[34].

$$r_{\min} = \min \left[ \frac{|(\mathbf{p}_{\text{obstacle}} - \mathbf{p}_k) \times (\mathbf{p}_{\text{obstacle}} - \mathbf{p}_{k+1})|}{|\mathbf{p}_{k+1} - \mathbf{p}_k|} \right] \quad (3.19)$$

Summing up, a more negative value of  $O_2(\mathcal{P})$  is preferable. If the distance  $r_{\min}$  dominates the safety margin,  $m$ , the value of  $O_2(\mathcal{P})$  grows exponentially, and thus, gives a higher penalty. On the other hand, the larger the distance  $O_2(\mathcal{P})$  compared to the safety margin  $m$ , the objective function takes a lower penalty.

### 3.7.3 Objective Function - Avoid Sharp Turns in the xy-plane

Given the consecutive waypoints  $\mathbf{p}_k$ ,  $\mathbf{p}_{k+1}$  and  $\mathbf{p}_{k+2}$  interconnected by the path segments  $\mathbf{v}_k$  and  $\mathbf{v}_{k+1}$ , the path's change in heading,  $d\psi_i$ , in the xy-plane can be calculated as expressed in (3.20)[35].

$$d\psi_i = \arccos \left( \frac{\mathbf{v}_k \cdot \mathbf{v}_{k+1}}{|\mathbf{v}_k| |\mathbf{v}_{k+1}|} \right) \quad (3.20)$$

Figure 3.6 illustrates the change of heading,  $d\psi_i$ , in the xy-plane of a path  $\mathcal{P}$  consisting of consecutive waypoints  $\mathbf{p}_k, \dots, \mathbf{p}_{k+3}$  interconnected by path segments  $\mathbf{v}_k, \dots, \mathbf{v}_{k+2}$ .

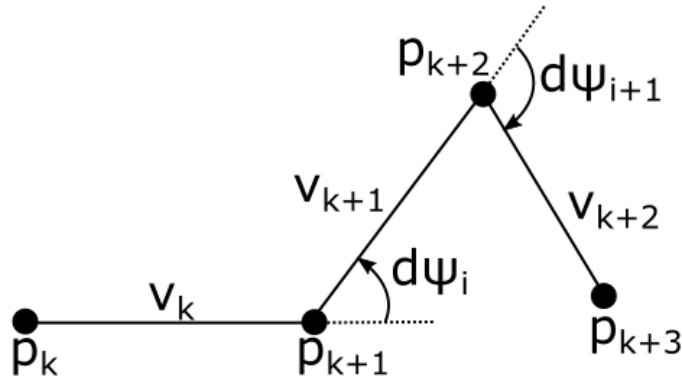


Figure 3.6: Illustration of the Change of Heading in the xy-plane

The third objective function,  $O_3(\mathcal{P})$ , makes the path planner favour paths with lower changes in heading. Paths with the lowest possible maximum change of heading  $d\psi_i$  between any two path segments in the path are preferable. A mathematical representation of  $O_3(\mathcal{P})$  is given in (3.21).

$$O_3(\mathcal{P}) = \max [d\psi_i] \quad (3.21)$$

Summing up, the unit of this objective function is simply radians. Thus, lower values of  $O_3(\mathcal{P})$  indicates that the sharpest turn of a path  $\mathcal{P}$  is less sharp.

### 3.7.4 Objective Function - Ensuring Reliable DVL Measurements

In real ROV operations, the availability and reliability of DVL measurements are essential. The absence of reliable DVL measurements deteriorates the ROVs capability of performing reliable state estimation, causing the ROV to take on worse behaviour. DVL measurements tend to become unreliable if the distance from the ROV to the seafloor is too large.

Therefore, in real ROV operations altitude control, that is, keeping the ROV at a fixed altitude above the seafloor, is often applied. Thus, requiring online, real-time update of the altitude, which is provided by DVL measurements.

For offline optimal path planning, knowledge about the seafloor is not necessarily available. However, an objective function,  $O_4(\mathcal{P})$ , is implemented to examine the performance of the path planner while constraining the placement of the z-coordinates of the waypoints. Satisfactory performance in the offline case could motivate the possibility of an online extension.

The objective function,  $O_4(\mathcal{P})$ , takes in a desired depth corresponding to a desired altitude above the seabed,  $z_{\text{desired}}$ . Thus, the preference that the z-coordinate of interior waypoints,  $z_n$  are located within the interval  $[z_{\text{desired}} - z_{\text{limit}}, z_{\text{desired}} + z_{\text{limit}}]$  arises.  $z_{\text{limit}}$  acts as a margin on the tolerated deviation between  $z_{\text{desired}}$  and the z-coordinate assigned the interior waypoint in question by the path planner. If the z-coordinate is placed outside of the interval, the path is penalized exponentially worse, thus  $O_4(\mathcal{P})$ . Figure 3.7 illustrates the methodology behind  $O_4(\mathcal{P})$ .

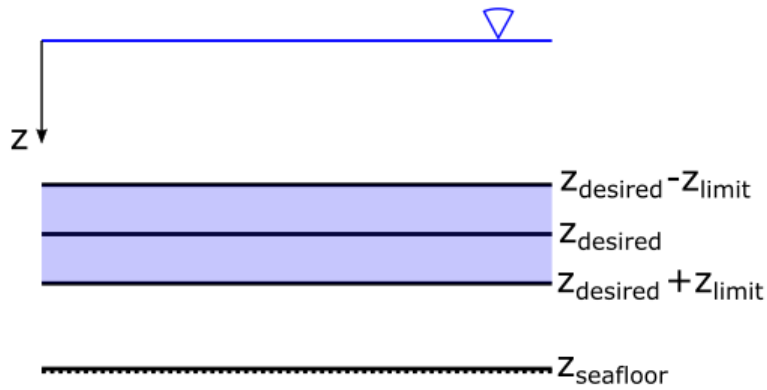


Figure 3.7: Illustration of Objective Function Evaluating Placement of Waypoint  $z$ -coordinates

A mathematical representation of  $O_4(\mathcal{P})$  is expressed in (3.22).

$$O_4(\mathcal{P}) = \begin{cases} e^{(z_{\text{deviation}} - z_{\text{limit}})}, & \text{if } z_{\text{deviation}} > z_{\text{limit}} \\ z_{\text{deviation}} - z_{\text{limit}}, & \text{otherwise} \end{cases} \quad (3.22)$$

where  $z_{\text{deviation}}$  denotes the deviation between the  $z$ -coordinate of any interior waypoint,  $z_n$ , and the reference value  $z_{\text{terrain}}$ , as shown in (3.23).

$$z_{\text{deviation}} = \max[\text{abs}(z_k - z_{\text{terrain}})] \quad (3.23)$$

Summing up, the objective function  $O_4(\mathcal{P})$  grows exponentially if the  $z$ -coordinate of the worst placed interior waypoint is placed outside of the interval  $[z_{\text{desired}} - z_{\text{limit}}, z_{\text{desired}} + z_{\text{limit}}]$ . However, if the  $z$ -coordinate is contained in the interval, the objective value will take a more negative value the closer the  $z$ -coordinate is to  $z_{\text{desired}}$ . A more negative value for  $O_4(\mathcal{P})$  is preferable.

### 3.7.5 Constraint - Ensuring Safe Docking

To ensure safe docking, the heading of the ROV should remain constant and equal to the heading of the docking station,  $\psi_n$  from the penultimate waypoint,  $\mathbf{p}_{n-1}$ , to the terminal waypoint,  $\mathbf{p}_n$ . That is, the ROV should take on a constant heading on the last path segment. To be able to assign the heading  $\psi_n$  to the last path segment, the  $z$ -coordinate of  $\mathbf{p}_{n-1}$  and  $\mathbf{p}_n$  are set to be equal, that is,  $z_{n-1} = z_n$ , and thus, reducing the terminal phase of the docking problem to a two-dimensional problem.

**Assumption 1:** The heading of the docking station,  $\psi_n$ , is known

**Assumption 2:** There are no obstacles present in the area around the terminal position,  $\mathbf{p}_n$ , of the docking operation, such that the penultimate waypoint,  $\mathbf{p}_{n-1}$ , can be placed at a fixed position from the terminal waypoint. Such that, the last path segment connection  $\mathbf{p}_{n-1}$  and  $\mathbf{p}_n$  takes on a fixed length.

Due to assumption two, the objective function concerning the safety margin to obstacles,  $O_2(\mathcal{P})$ , will not be evaluated for the last path segment. The reason is that for a fixed path segment in all candidate paths, a perpendicular distance to an obstacle may not exist. This results in this objective function taking a high, non-correct value, causing the algorithm to fail. Additionally,

$O_4(\mathcal{P})$  is not evaluated for  $\mathbf{p}_{n-1}$  and  $\mathbf{p}_n$  as the depth of the docking station in the environment is assumed independent of the previously assigned desired depth.

An illustration of the terminal phase of the docking mission is given in Figure 3.8. According to assumption 2, no obstacles are conflicting with this area. Further, the radius of the red circle in the Figure represents the length of the last path segment, denoted  $d$ , which is a known distance.

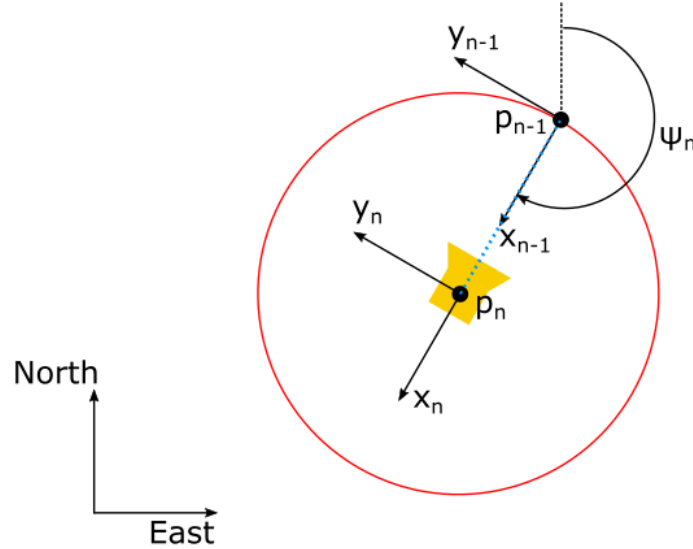


Figure 3.8: Illustration of Terminal Phase of the Docking Operation

Moreover, the penultimate waypoint is assigned a fixed position for all candidate paths, as expressed in (3.24).

$$\mathbf{p}_{n-1} = \mathbf{p}_n + \mathbf{R}_{z,\psi}^{2D}(\psi_n) [-d, \quad 0]^T, \quad (3.24)$$

where  $\mathbf{P}_{z,\psi}^{2D}$  is the two-dimensional version of the rotation matrix, expressed in (3.25).

$$\mathbf{R}_{z,\psi}^{2D} = \begin{bmatrix} \cos(\psi) & \sin(\psi) \\ -\sin(\psi) & \cos(\psi) \end{bmatrix} \quad (3.25)$$

### 3.7.6 Path Selector

In optimization problems solved by multi-objective optimization algorithms, the applied algorithm generates a set of Pareto-optimal solutions, that is, equally optimal solutions that are indistinguishable among each other. Furthermore, to choose one of the paths for the vehicle to follow, a path selector is designed selecting a suitable path based on a different weighing of the objective functions. Selecting the desired path,  $\mathcal{P}_{\text{selected}}$  in this manner is done by applying the path selector algorithm given in Algorithm 1 to the final population of Pareto-optimal paths,  $P$ . The path selector algorithm is based on numeric values related to the different objective functions, namely, the average path length,  $\bar{O}_1$ , the minimum path length,  $O_{1,\min}$ , the required safety margin  $O_{2,\text{required}}$  and the required depth/altitude deviation factor  $O_{4,\text{required}}$  corresponding to  $O_1(\mathcal{P}), O_2(\mathcal{P}), O_4(\mathcal{P}) \in P$ , respectively.



**Algorithm 1** Path Selector

---

```

1: procedure PATH-SELECTOR( $P$ )
2:    $O_{1,\min} = \min [O_1] \in P$  ▷ Calculating the shortest path length
3:    $\bar{O}_1 = \text{mean} [O_1] \in P$  ▷ Calculating the average path length
4:    $O_{1,\text{ideal}} = (L_{\text{shortest}} + L_{\text{mean}})/2$  ▷ Defining desirable path length for selection
5:    $O_{2,\text{required}} = \beta_{O2} \cdot \text{mean}(O_2)$  ▷ Defining maximum allowed safety margin
6:    $O_{4,\text{required}} = \beta_{O4} \cdot \text{mean}(O_4)$  ▷ Defining maximum allowed altitude deviation factor
7:   flag = 0
8:   while flag  $\neq$  1 do
9:      $\mathcal{P}_{\text{proposed}} = \text{find} [\mathcal{P}(O_1 \approx O_{1,\text{ideal}})] \in P$ 
10:    if  $\mathcal{P}_{\text{proposed}}(O_2) > O_{2,\text{required}}$  or  $\mathcal{P}_{\text{proposed}}(O_4) > O_{4,\text{required}}$  then
11:      remove( $\mathcal{P}_{\text{proposed}} \in P$ ) ▷ If selection criteria not satisfied, discard  $\mathcal{P}_{\text{proposed}}$ 
12:    else
13:       $\mathcal{P}_{\text{selected}} = \mathcal{P}_{\text{proposed}}$  ▷ If selection criteria i satisfied save  $\mathcal{P}_{\text{selected}}$ 
14:      flag = 1 ▷ Abort path selection, as a satisfactorily path,  $\mathbf{P}_{\text{selected}}$  is obtained
15:    end if
16:  end while
17: end procedure

```

---

In Algorithm 1,  $\beta_{O2}$  and  $\beta_{O4}$  are scaling factors deciding how strict the selection process should be.

### 3.7.7 The Path Planning Problem

Now, this Section aims to sum up the optimal path planning problem described throughout this Sections in a reader-friendly manner. Initially, several parameters have to be assigned numerical values. Note that the number of parameters that need to be defined may vary on the application. Table 3.2 provides an overview of the maximum number of parameters that may need to be defined.

Table 3.2: User defined path planning parameters

Property	Symbol	Unit
Initial position	$(x_0, y_0, z_0)$	[m]
Final position	$(x_n, y_n, z_m)$	[m]
Obstacle 1 position	$(x_{o1}, y_{o1}, z_{o1})$	[m]
Obstacle 2 position	$(x_{o2}, y_{o2}, z_{o2})$	[m]
Obstacle radius		[m]
Final heading	$\psi_n$	[°]
Number of interior waypoints	$K$	[-]
Number of objectives	$M$	[-]
Sea depth	$z_{\max}$	[m]
Desired depth	$z_{\text{desired}}$	[m]
Safety margin to obstacles acting in $O_2$	$m$	[m]
Path selector scaling factor	$\beta_{O2}$	[-]
Path selector scaling factor	$\beta_{O4}$	[-]

Next, the multi-objective path planning problem can be solved by optimization utilizing a GA, minimizing the objective functions expressed in (3.26a)-(3.26d). Note that the number of objectives

may vary depending on the application, for instance, (3.26d) is omitted for some applications.

$$O_1(\mathcal{P}) = \sum_{k=1}^{n-1} \sqrt{(x_{k+1} - x_k)^2 + (y_{k+1} - y_k)^2 + (z_{k+1} - z_k)^2} \quad (3.26a)$$

$$O_2(\mathcal{P}) = \begin{cases} m - r_{\min}, & \text{if } r_{\min} > m \\ e^{r_{\min} - m}, & \text{otherwise} \end{cases} \quad (3.26b)$$

$$O_3(\mathcal{P}) = \max [d\psi_i] \quad (3.26c)$$

$$O_4(\mathcal{P}) = \begin{cases} e^{\gamma(z_{\text{deviation}} - z_{\text{limit}})}, & \text{if } z_{\text{deviation}} > z_{\text{limit}} \\ z_{\text{deviation}} - z_{\text{limit}}, & \text{otherwise} \end{cases} \quad (3.26d)$$

Summing up, the objective functions aim to provide the set of obtained paths with the following desired properties through optimization:

- $O_1$ , expressed in (3.26a), short path length.
- $O_2$ , expressed in (3.26b), sufficient safety margin to obstacles.
- $O_3$ , expressed in (3.26c), smaller magnitudes of turns in the xy-plane.
- $O_4$ , expressed in (3.26d), ensures sufficient depth in order to achieve reliable DVL-measurements.

The objective functions are minimized subject to constraints such as bounds on the defined problem space in NED-coordinates, expressed in (3.27a)-(3.27c), fixed initial position  $\mathbf{p}_0$  and terminal position  $\mathbf{p}_n$ , expressed in (3.27d)-(3.27e), as well as constraint on the final heading resulting in a fixed penultimate position,  $\mathbf{p}_{n-1}$ , expressed in (3.27f).

$$N = [x_{\min}, \quad x_{\max}] \quad (3.27a)$$

$$E = [y_{\min}, \quad y_{\max}] \quad (3.27b)$$

$$D = [0, \quad z_{\text{seafloor}}] \quad (3.27c)$$

$$\mathbf{p}_0 = (x_0, y_0, z_0) \quad (3.27d)$$

$$\mathbf{p}_n = (x_n, y_n, z_n) \quad (3.27e)$$

$$\mathbf{p}_{n-1} = \mathbf{p}_n + \mathbf{R}_{z,\psi}^{2D}(\psi_n) [-d, \quad 0]^T \quad (3.27f)$$

Finally, as a post-processing step, a path,  $\mathcal{P}_{\text{selected}}$ , is selected by the path selector proposed in Algorithm 1 in accordance with suitably chosen scaling factors  $\beta_{O2}$  and  $\beta_{O4}$ .

The optimal path planning problem defined in this Section is somewhat general, however, in this thesis, it is applied to autonomous docking missions for Minerva 2. In this thesis, the optimal path planning problem is solved by the genetic multi-objective evolutionary algorithm NSGA-II. The theory behind this algorithm is given in Chapter 4.

## Chapter 4

# Optimization Theory

### 4.1 Non-Dominated Sorting Genetic Algorithm (NSGA-II)

The optimization algorithm used to solve path planning problems in this thesis is the Non-Dominated Sorting Genetic Algorithm (NSGA-II) proposed in [8]. Section 1.5.3, provides a brief overview of optimization techniques, including genetic algorithms, acting as a background for this Chapter. Moreover, this Chapter provides a walk-through of all the operators necessary to apply NSGA-II to the optimal path planning problem defined in Section 3.7.7, mainly based on [8].

To entirely explain NSGA-II, some essential operations utilized throughout the optimization process need further attention. These operations are a fast non-dominated sorting approach, crowding distance assignment in addition to standard genetic algorithm operators such as binary tournament selection, simulated binary crossover and polynomial mutation.

#### 4.1.1 Fast Non-Dominated Sorting Approach

In [8], the fast non-dominated sorting approach is highlighted as a major improvement to NSGA-II, compared to the former version NSGA. This Section gives the theory of how the sorting is performed. Firstly, note that the purpose of non-dominated sorting is to sort the solutions in the population into different non-dominated fronts based on the objective function values, that is, the overall fitness of each solution. The first non-dominated front contains all solutions of the first non-dominated level, meaning the solutions of this front are the best among the population as they are equally good and not dominated by any other solution in the population in terms of objective function values. The solutions of the second non-dominated front are dominated by the solution of the first non-dominated front, but dominates the solutions of the third non-dominated front and so on. A graphical representation based on figures in [8] is provided in Figure 4.1, where  $P$  represents the population that is being sorted into five fronts of non-domination,  $\mathcal{F}_i$ ,  $i = 1, 2, \dots, 5$ .

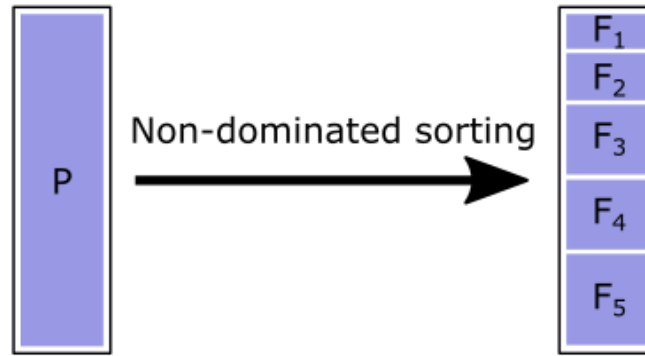


Figure 4.1: Graphical explanation of non-dominated sorting

To initialize this sorting in a computationally efficient way, [8] introduces two entities, namely the domination count,  $n_p$ , and the set  $S_p$ . The domination count,  $n_p$ , is an index assigned to each solution,  $p$ , in the population  $P$ , counting the total number of solutions in the population that dominates the solution  $p$ . On the other hand, the set,  $S_p$ , is a set of all solutions in the population,  $P$ , that the solution in question,  $p$ , dominates. Domination refers to objective function values. Initially, the domination count,  $n_p$ , is set to zero and the set  $S_p$  is an empty set, as shown in (4.1)[8].

$$n_p = 0 \quad (4.1a)$$

$$S_p = \emptyset \quad (4.1b)$$

Furthermore, for all solutions in the first non-dominated front,  $\mathcal{F}_1$ , the non-domination count,  $n_p$ , remains zero throughout the sorting process. At this point, all the solutions not belonging to the first non-dominated front,  $\mathcal{F}_1$ , are contained in the set  $S_p$ , and thus, the first front is at this moment identified. The solutions of the first front are assigned a rank,  $p_{\text{rank}} = 1$ [8].

Moreover, subsequent fronts are identified. This is done by further examination of each solution with  $n_p = 0$ . For each solution with  $n_p = 0$ , each solution  $q$  contained in the corresponding set  $S_p$  has its domination count reduced by one. After doing so, if the domination count becomes zero,  $n_q = 0$ , for any solutions  $q$  contained in the set  $S_p$  in question, these solutions are put in a separate list,  $Q$ . The members of the list  $Q$  belong to the second non-dominated front,  $\mathcal{F}_2$ , and are assigned a rank. The process continues with each member of the list  $Q$  in order to identify the third non-dominated front,  $\mathcal{F}_3$  and so on. This is done as an iterative process until all non-dominated fronts,  $\mathcal{F}_i$   $i = 1, 2, 3, \dots$ , in the population are identified and all solutions are assigned a rank  $q_{\text{rank}} = i + 1$ [8].

Pseudo-code for the fast non-dominated sorting algorithm proposed in [8] is reproduced in Algorithm 2.

**Algorithm 2** Fast non-dominated sorting algorithm

---

```

1: procedure FAST-NON-DOMINATED-SORTING( $P$ )
2:   for each  $p \in P$  do
3:      $S_p = \emptyset$ 
4:      $n_p = 0$ 
5:     for each  $q \in P$  do
6:       if  $p \prec q$  then                                     ▷ If  $p$  dominates  $q$ 
7:          $S_p = S_p \cup \{q\}$                                ▷ Add  $q$  to the set  $S_p$  of solutions dominated by  $p$ 
8:       else if  $p \succ q$  then                               ▷ if  $q$  dominates  $p$ 
9:          $n_p = n_p + 1$                                      ▷ Increment the domination counter for  $p$ 
10:      end if
11:      if  $n_p = 0$  then                                     ▷ Identifying the first front
12:         $p_{rank} = 1$                                        ▷ Assigning members of the first front rank 1
13:         $\mathcal{F}_1 = \mathcal{F}_1 \cup \{p\}$ 
14:      end if
15:    end for
16:  end for
17:   $i = 1$                                                  ▷ Front counter
18:  while  $\mathcal{F}_i \neq \emptyset$  do                             ▷ Identifying subsequent fronts
19:     $Q = \emptyset$ 
20:    for each  $p \in \mathcal{F}_i$  do                                 ▷ Iterating over each solution  $p$  in front  $i$ 
21:      for each  $q \in S_p$  do                                 ▷ For each solution  $q$  in each set  $S_p$  corresponding to  $p$ 
22:         $n_q = n_q - 1$                                      ▷ Decrementing domination counter
23:        if  $n_q = 0$  then
24:           $q_{rank} = i + 1$                                  ▷ Assigning rank to subsequent fronts
25:           $Q = Q \cup \{q\}$ 
26:        end if
27:      end for
28:    end for
29:     $i = i + 1$                                            ▷ Incrementing front counter
30:     $\mathcal{F}_i = Q$                                            ▷ Identifying front  $i$ 
31:  end while
32: end procedure

```

---

**4.1.2 Crowding Distance Assignment**

In addition to the non-dominated sorting, which arranges the population according to fitness, crowding distance assignment is performed in order to determine the diversity among the different solutions in the population. In GAs, it is desirable to preserve a spread of solutions within the search space. In terms of diversity preservation, NSGA-II also claims to be significantly improved compared to NSGA [8] which depended on a sharing function requiring user input. Thus, causing the necessity of a comprehensive comparison among the solutions in the population, resulting in a quite high computational complexity. The improved NSGA-II avoids these challenges by introducing the crowded-comparison approach, not requiring any user input while reducing computational complexity.

Moreover, the crowding distance,  $i_{\text{distance}}$ , is computed within each non-dominated front in the population. This is done by sorting the solutions within each front according to each objective function value in ascending order of magnitude. Next, for each objective function, boundary solutions, i.e. the smallest and largest function value are assigned an infinite distance value,  $i_{\text{distance}} = \infty$ . All intermediate solutions are assigned a distance value for each objective function, equal to the normalized corresponding objective function values of two adjacent solutions. Summing up, the crowding distance is calculated individually for each objective function. Finally, the overall crowding distance is calculated as the sum of the individual distances corresponding to each

objective[8].

Figure 4.2, inspired by [8], gives a graphical example of the crowding distance calculation in a case with two objective functions,  $f_1$  and  $f_2$ . The red dots represent the boundary values for  $f_1$  and  $f_2$ , and thus, these solutions are assigned an infinite distance value. Furthermore, the green dot represents the solution under examination in this example. The two blue dots represent its adjacent solutions. The solution represented by the green dot is assigned a distance metric equal to the sum of the normalized side lengths of the dashed cuboid interconnecting the two blue dots. Finally, the curly brackets represent the side lengths of the dashed cuboid. Note that, even though this example only represents a case with two objective functions, the algorithm is still applicable to problems with more than two objective functions.

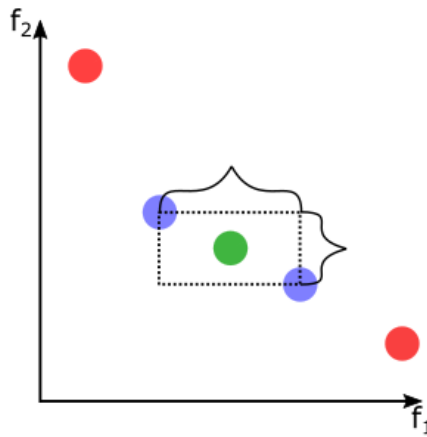


Figure 4.2: Crowding distance calculation with two objective functions,  $f_1$  and  $f_2$

Pseudo-code for the crowding distance assignment algorithm proposed in [8] is reproduced in Algorithm 3, where the crowding distances in front  $\mathcal{F}_i$  is examined. The front  $\mathcal{F}_i$  contains the solutions  $\mathcal{F}_i[k]$ ,  $k = 1, 2, 3, \dots, l$ , each which are assigned a crowding distance metric  $\mathcal{F}_i[k]_{\text{distance}}$ . Furthermore,  $m$  denotes the  $m$ th objective function value, while  $f_m^{\max}$  and  $f_m^{\min}$  denotes the maximum and minimum value of the  $m$ th objective function value for normalization.

---

**Algorithm 3** Crowding distance assignment

---

```

1: procedure CROWDING-DISTANCE-ASSIGNMENT( $\mathcal{F}_i$ )
2:    $l = |\mathcal{F}_i|$  ▷ Number of solutions,  $j = 1, 2, \dots, l$  in front  $\mathcal{F}_i$ 
3:   for each  $k$  do
4:      $\mathcal{F}_i[k]_{\text{distance}} = 0$ 
5:     for each objective  $m$  do
6:        $\mathcal{F}_i = \text{sort}(\mathcal{F}_i, m)$ 
7:        $\mathcal{F}_i[k = 1] = \infty$ 
8:        $\mathcal{F}_i[k = l] = \infty$ 
9:       for  $k = 2$  to  $(l - 1)$  do
10:         $\mathcal{F}_i[k] = \mathcal{F}_i[k] + (\mathcal{F}_i[k + 1].m - \mathcal{F}_i[k - 1].m) / (f_m^{\max} - f_m^{\min})$ 
11:      end for
12:    end for
13:  end for
14: end procedure

```

---

### Crowded-Comparison Operator

After performing the non-dominated sorting and the crowding distance calculation, the crowded-comparison operator can be defined. The selection process utilizes the crowded-comparison operator throughout NSGA-II, and thus, ensuring diversity among non-dominated solutions. In short, the crowded-comparison operator compares two solutions for their extent of proximity to other solutions. It also takes rank into account, as lower (better) ranks are preferable. If two solutions have the same rank, the solution located in a less crowded region, that is, the solution with the higher crowding distance is preferred. Take into account the two solutions  $i$  and  $j$ , with corresponding ranks  $i_{\text{rank}}$ ,  $j_{\text{rank}}$  and crowding distances  $i_{\text{distance}}$  and  $j_{\text{distance}}$ . The criterion for the solution  $i$  dominating the solution  $j$ , i.e.  $i \prec j$  is given in Algorithm 4[8].

---

#### Algorithm 4 Crowded-comparison operator

---

```

1: procedure CROWDED-COMPARISON OPERATOR( $i, j$ )
2:   if ( $i_{\text{rank}} < j_{\text{rank}}$ ) or ( $i_{\text{rank}} = j_{\text{rank}}$  and  $i_{\text{distance}} > j_{\text{distance}}$ ) then
3:      $i \prec j$ 
4:   end if
5: end procedure

```

---

### 4.1.3 Selection

An illustration of the selection process is given in Figure 4.3[8], where  $R_t$  represents the concatenation of the  $t$ th parent population,  $P_t$ , and the  $t$ th offspring population  $Q_t$ , forming the intermediate population  $R_t = P_t \cup Q_t$ . Thus, the population  $R_t$  has size  $2N$ , while the populations  $P_t$  and  $Q_t$  have size  $N$ . Elitism is ensured since all individuals from  $P_t$  and  $Q_t$  are present in  $R_t$ . The parent population of the next iteration,  $P_{t+1}$  of size  $N$ , called the mating pool, is used for reproduction generating next offspring population  $Q_{t+1}$  of size  $N$ [8].

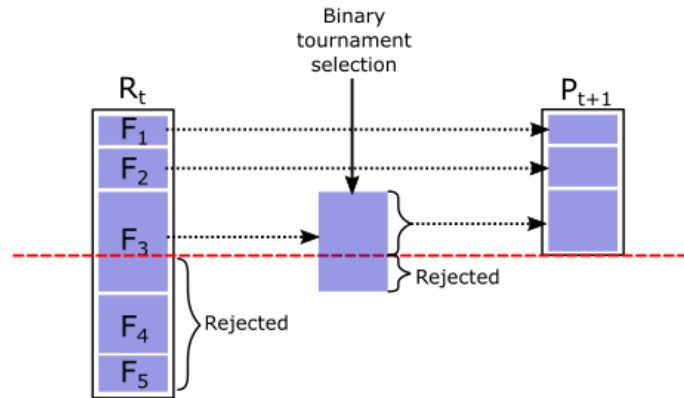


Figure 4.3: Selection in the  $i$ -th iteration of NSGA-II[8]

Furthermore, solutions from fronts with the lowest rank are preserved into the mating pool. The process goes on as long as there is space for an entire front containing all solutions in the mating pool. When an entire front no longer can be accommodated, and if there still is space for more solutions in the mating pool, binary tournament selection based on the crowded-comparison operator is used to fill the remaining spots in  $P_{t+1}$  such that the size of  $P_{t+1}$  becomes  $N$ . Furthermore,  $P_{t+1}$  is used to generate the next offspring population,  $Q_{t+1}$ , utilizing the simulated binary crossover(SBX) and polynomial mutation operators.

## Binary Tournament Selection

In general, tournament selection consists of choosing a number  $t$  of individuals at random from the population and copying the best individuals from the group of  $t$  individuals into the intermediate population.  $t$  is known as the tournament size, and such tournaments are often held between two individuals. Tournament selection with  $t = 2$  is called binary tournament selection[36].

As briefly mentioned, the remaining slots of the next parent population are resided by solutions selected by binary tournament selection based on the crowded comparison operator, forming  $P_{t+1}$  of size  $N$ . Thus, two solutions from the front that cannot be entirely resided in  $P_{t+1}$  are contending against each other, and the solution with the largest crowding distance is transmitted into  $P_{t+1}$ .

### 4.1.4 Simulated Binary Crossover (SBX)

The SBX operator initially proposed in [37], is shown to be particularly useful as a genetic operator in multi-objective optimization problems where GAs are applied as it outperforms several other crossover operators. Furthermore, crossover operators ability to combine good portions of parent genes into better offspring individuals reflects the performance of the crossover operator. In SBX a child solution is created from a probability distribution that depends on the parent genes[37].

Moreover, the SBX operator is augmented to account for constraint handling such as dimensions of the search space, for instance as in the optimal path planning problem defined in Section 3.7.7, in [38]. As a result of this, for instance, no offspring solution is generated outside of the search space that is defined in the problem parameters. This approach is considered suitable for such applications, and thus, the mathematical representation of the SBX operator proposed [38] is presented in this Section.

The procedure of generating an offspring population using the SBX operator is based on the genes of two parent solutions, namely,  $y_1$  and  $y_2$ , from which two child solutions, namely  $c_1$  and  $c_2$ , are generated. Figure 4.4 presents an illustration explaining crossover, where the blue and the red line represents the genes of the parents  $y_1$  and  $y_2$  respectively, while the recombined lines to the right in the figure represent the genes of the offspring  $c_1$  and  $c_2$ .

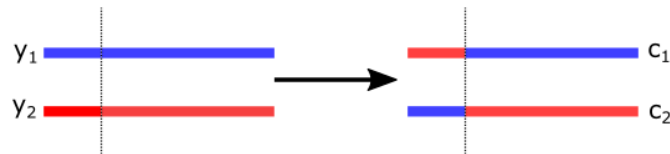


Figure 4.4: SBX

The properties of  $c_1$  and  $c_2$  are calculated based on the properties of the parents  $y_1$  and  $y_2$  as expressed in (4.2a) and (4.2b)[38], respectively for each child.

$$c_1 = \frac{1}{2} \cdot [(y_1 + y_2) - \beta_q(y_2 - y_1)] \quad (4.2a)$$

$$c_2 = \frac{1}{2} \cdot [(y_1 + y_2) + \beta_q(y_2 - y_1)] \quad (4.2b)$$

The parameter  $\beta_q$  is based on the crossover probability distribution index,  $\eta_c$ , and a randomly generated number,  $u$ , between 0 and 1. The properties of the offspring depend on the magnitude



of  $\eta_c$ , as a high value of  $\eta_c$  renders a higher probability of generating "near-parent" offspring and vice versa[37]. The expression for  $\beta_q$  is a probability distribution as expressed in (4.3)[38].

$$\beta_q = \begin{cases} (u\alpha)^{\frac{1}{\eta_c+1}} & \text{if } u \leq \frac{1}{\alpha} \\ \left(\frac{1}{2-u\alpha}\right)^{\frac{1}{\eta_c+1}}, & \text{otherwise} \end{cases} \quad (4.3)$$

Furthermore, the parameter  $\alpha$  is expressed in (4.4)[38].

$$\alpha = 2 - \beta^{-(\eta_c+1)} \quad (4.4)$$

Lastly, the parameter  $\beta$ , called the spread factor, is expressed in (4.5), such that the members of the offspring population do not escape the range of the constrained domain in which the problem is defined[38][38].

$$\beta = \begin{cases} 1 + \frac{2}{y_2 - y_1} \min [(y_1 - y_{lower}), (y_{upper} - y_2)] & \text{if } y_1 < y_2 \\ 1 + \frac{2}{y_2 - y_1} \min [(y_2 - y_{lower}), (y_{upper} - y_1)] & \text{if } y_1 > y_2 \end{cases} \quad (4.5)$$

In NSGA-II the SBX operator is applied for generating the next offspring population  $Q_{t+1}$  with a crossover probability,  $p_c$ .

#### 4.1.5 Polynomial Mutation

By applying the polynomial mutation operator, a child,  $c$  is generated based on one parent,  $y$ , and added to the offspring population[39]. As for the SBX operator, the theory behind polynomial mutation is augmented in [38], such that the child's properties do not escape the range of the constrained domain in which the problem is defined. The mathematical description proposed in [38] is presented in this Section. The properties of child,  $c$ , is calculated as expressed in (4.6), such that, the mutated value is changed by a perturbation factor  $\delta_q$  to a neighbourhood value using a polynomial distribution with its mean at the current value and its variance as a function of the mutation distribution index,  $\eta_m$ [39][38].

$$c = y + \delta_q(y_{upper} - y_{lower}) \quad (4.6)$$

Furthermore, the perturbation factor  $\delta_q$  is defined as expressed in (4.7), where  $u$  is a randomly generated number between 0 and 1[38].

$$\delta_q = \begin{cases} [2u + (1 - 2u)(1 - \delta)^{\eta_m+1}]^{\frac{1}{\eta_m+1}} - 1, & \text{if } u \leq 0.5 \\ 1 - [2(1 - u) + 2(u - 0.5)(1 - \delta)^{\eta_m+1}]^{\frac{1}{\eta_m+1}}, & \text{otherwise} \end{cases} \quad (4.7)$$

Lastly, the parameter  $\delta$  is defined as expressed in (4.8)[38].

$$\delta = \frac{\min [(y - y_{lower}), (y_{upper} - y)]}{y_{upper} - y_{lower}} \quad (4.8)$$

In NSGA-II the polynomial mutation operator is applied subsequently after the SBX operator with a small mutation probability,  $p_m$ , to increase the search power of the algorithm. The search power is a GAs ability to create arbitrary offspring in the search space [37].

### 4.1.6 Computational Complexity

The computational complexity of NSGA-II is also paid attention to in [8]; in fact, the algorithm is claimed to do quite well in this context. By the big-O notation, the worst-case complexity of NSGA-II is in general estimated to be  $\mathcal{O}(MN^2)$ , where  $M$  is the number of objectives and  $N$  is the population size. This complexity is governed by the non-dominated sorting approach proposed in NSGA-II, which in fact, is one of the significant improvements to this iteration of the algorithm compared to the previous iteration, NSGA. Indeed, the worst-case computational complexity of the older NSGA is  $\mathcal{O}(MN^3)$ [8].

### 4.1.7 NSGA-II Parameters

Summing up, an overview of necessary NSGA-II parameters are provided in Table 4.1. These parameters are to be assigned numerical values, in accordance with suggested values given in the literature [8], [37], [38], [39].

Table 4.1: NSGA-II user defined parameters

<b>Property</b>	<b>Symbol</b>
Population size	$N$
Number of generations	$G$
SBX distribution index	$\eta_c$
SBX probability	$p_c$
Mutation distribution index	$\eta_m$
Mutation probability	$p_m$

Overall, this Section, along with Section 3.7.7 should provide the minimum required information on solving an optimal path planning problem by applying the NSGA-II based optimal path planner proposed in this thesis. By applying NSGA-II and defining the parameter settings given in Table 4.1 to the optimal path planning problem defined in section 3.7.7, a NSGA-II based optimal path planner is created. By the operators described throughout this Chapter, the optimal path planner ensures that a population of paths with the desired properties by minimizing the objectives  $O_1$ ,  $O_2$ ,  $O_3$  and  $O_4$  is generated. The path planner aims to obtain a set of so-called Pareto-optimal solutions, that is, solutions that are equally optimal but cannot be distinguished from each other due to the multi-objective nature of the optimal path planning problem.

# Chapter 5

## Method

This Chapter aims to put the operators presented throughout Chapter 4 in context by addressing the development of the NSGA-II based path planner corresponding to the optimal path planning problem defined in Section 3.7.7 in Chapter 3. As for the other Sections in Chapter 3 and Chapter 2, act as background when implementing the path planner into the control system of Minerva 2.

A NSGA-II based path planner is coded in `Matlab`, Appendix A provides the source code that is the path planner. Moreover, for the integration of the path planner into the autonomy framework in the control system of the ROV Minerva 2, the graphical programming language `LabVIEW` is used. The implementation enables carrying out simulations of autonomous docking missions and field trials. Since `LabVIEW` is compatible with `Matlab`-code, the implementation of the path planner into the control system uses some of the `Matlab` code. Some background behind the development of the path planner in `Matlab` is given in Section 5.1 and the `Matlab` directory is later used for verification of the optimal path planning technique, in Section 6.1. The method behind the integration of the path planner into `LabVIEW` is given in Section 5.6. The latter appears as an integrated system, allowing simulations of the ROV Minerva 2 doing autonomous docking missions following optimal paths generated by the NSGA-II based path planner. In addition, Sections 5.2 - 5.5 aims to give some insight into the already existing `LabVIEW` based ROV motion control system.

### 5.1 Optimal NSGA-II Based Path Planner

Furthermore, Algorithm 5 is meant to give an overview of the method behind the NSGA-II based path planner coded in `Matlab`. The Algorithm utilizes the operators described throughout Chapter 4, and it is based on pseudo code provided in [8]. By applying the optimal path planning problem defined in Section 3.7.7 to Algorithm 5, an optimal path is proposed. By the genetic nature of NSGA-II, the algorithm produces a population of paths. Ideally, the population is Pareto-optimal.

**Algorithm 5** Non-Dominated Sorting Genetic Algorithm, NSGA-II

---

```

1: procedure NSGA-II(input)
2:   parameters = input                                ▷ Initialize problem parameters
3:    $R_0 = \text{initial-population}(\text{parameters})$       ▷ Create initial population
4:   procedure FAST-NON-DOMINATED-SORTING( $R_0$ )          ▷ See algorithm 2
5:   end procedure
6:   procedure CROWDING-DISTANCE-ASSIGNMENT( $\mathcal{F}_i$ )      ▷ See algorithm 3
7:   end procedure
8:    $P_0 = \text{binary-tournament-selection}(R_0)$           ▷ Generate first parent population
9:    $Q_0 = \text{generate-offspring-population}(P_0)$         ▷ Generate first offspring population
10:  while Stop criterion is not reached do           ▷ Main loop
11:     $R_i = P_i \cup Q_i$                                 ▷ Concatenation of parent and offspring populations
12:    procedure  $\mathcal{F} = \text{FAST-NON-DOMINATED-SORTING}(R_i)$   ▷ See algorithm 2
13:    end procedure
14:    procedure CROWDING-DISTANCE-ASSIGNMENT( $\mathcal{F}_i$ )      ▷ See algorithm 3
15:    end procedure
16:     $P_{i+1} = \text{selection}(\mathcal{F})$                        ▷ Selecting next parent population, see section 4.1.3
17:     $Q_{i+1} = \text{generate-offspring-population}(P)$      ▷ Generating next offspring population
18:  end while
19: end procedure

```

---

Moreover, Algorithm 5 aims to clarify the logic behind the `Matlab`-coded NSGA-II based path planner. Next, this paragraph gives a walk-through of Algorithm 5. Firstly, the numeric input required to solve an optimal path planning problem using this optimal path planner consists of the general path planning problem parameters summed up in Section 3.7.7, as well as NSGA-II parameters provided in Section 4.1.7. Next, to initialize the NSGA-II, an initial population  $R_0$  of size  $2N$  is generated. The array  $R_0$  consists of random waypoints located within the constrained space defined in the problem parameters. Also, constraints such as fixed initial and terminal positions and the constraint on the final heading are taken into account. Additionally, when a population is generated, the objective functions are evaluated, and thus,  $R_0$  also contains objective function values for all its solutions. Also, more information about the initial population is obtained by applying fast non-dominated sorting and crowding distance assignment, namely, assigning rank and crowding distance to each solution, respectively. At this point, all information about the initial population is obtained, and thus, binary tournament selection is applied to choose the first parent population,  $P_0$ , of size  $N$ . Based on  $P_0$ , the first offspring population  $Q_0$  of size  $N$  is generated by applying the SBX and polynomial mutation operators in coherence with relevant parameters. Again, whenever a new population is generated, objective function evaluations are carried out, assigning objective function values to each solution in the population. Now, the initialization of the optimization process is done, and the Algorithm goes into the main loop and performs a pre-defined number  $G$  of generations to obtain the final population of optimal solutions. The main loop is quite simple, as it initially concatenates the  $i$ th parent population  $P_i$  and the  $i$ th offspring population  $Q_i$ , forming the intermediate population  $R_i$  of size  $2N$ . Next, more information is obtained about the  $i$ th intermediate population,  $R_i$ , by applying fast non-dominated sorting and crowding distance assignment. Furthermore, the selection performed in the main loop to obtain the next parent population,  $P_{i+1}$ , is slightly different from in the initial phase of the Algorithm, as described in 4.1.3. Finally, the selection chooses the next parent population,  $P_{i+1}$ , and then, the next offspring population,  $Q_{i+1}$ , is generated by the SBX and polynomial mutation operators. Then, the main loop starts over again until the stopping criterion of  $G$  number of generations, or  $G$  number of iterations, in the main loop are performed.

Furthermore, the flowchart in Figure 5.1 aims to provide a graphical overview of the NSGA-II based optimal path planner. In addition to the optimization described in Algorithm 5, some post-optimization processing is done. In the first place, the path selector algorithm provided in Algorithm 1 is applied to the final population to obtain a path that not only is considered optimal but also satisfies additional user-defined preferences on the objective function values. Also, in the

post-processing phase, plots are obtained, showing the objective function values of each solution in the final population. Trends in these plots are used to comment on the performance of the optimization.

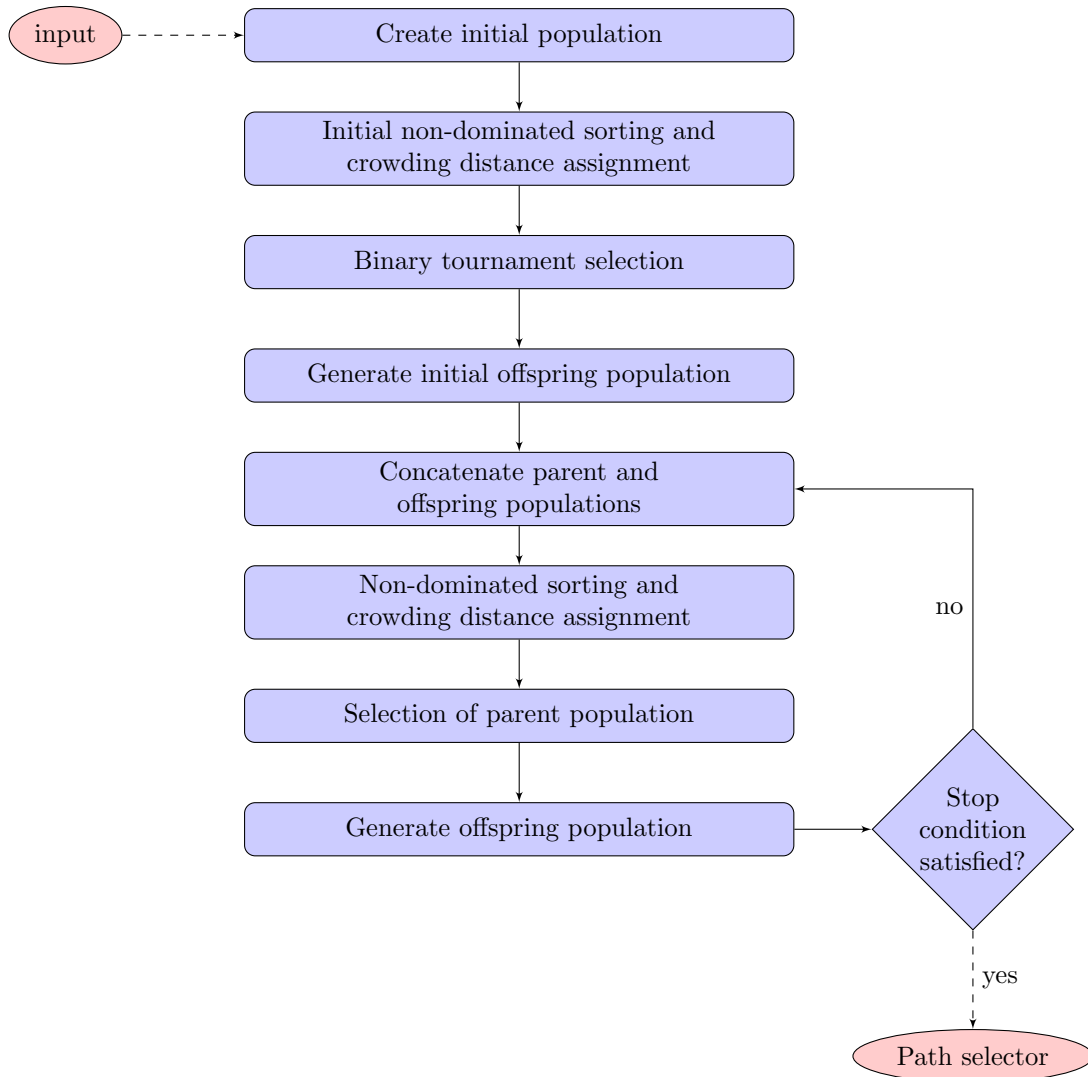


Figure 5.1: Flowchart of the NSGA-II based path planner

## 5.2 LabVIEW based ROV Motion Control System

The ROV motion control system is implemented in practice using an object-oriented approach in the graphical programming language LabVIEW, which has been developed for the AUR-Lab ROVs over the years. The literature review in Section 1.5.1 addresses the development of the control system. Due to its object-oriented nature, the control system applies to multiple ROVs. In general, LabVIEW programs are called virtual instruments (VIs). Such VIs consist of a user interface, or front panel, containing controls and parameters for input and output to objects in the code. The front panel is based on code in the form of a block diagram [40]. The block diagram can contain nodes utilizing Matlab-code and subsystems, making the code more readable. Due to its object-oriented nature, the code is somewhat generic.

The ROV motion control system has been developed by MSc students, Ph.D. candidates, professors,

and postdoctoral researchers since 2010. The motion control system consists of three main VIs, the overall control system called Njord, a graphical user interface (GUI) called Frigg, and a hardware-in-the-loop (HIL) simulator called Verdandi. Additionally, there are some optional VIs extending the capabilities of these main VIs, such as the autonomy module which includes functions for computer vision using both sonar and cameras. The majority of the contributions in Section 1.5.1 have been developed and tested in LabVIEW. The control system follows the control architecture provided in Figure 3.2. This Chapter aims to give an overview of the experimental setup for real ROV operations, that is, field trails, and HIL-simulations, as well as an overview over the key functionalities in the three main VIs, namely Njord, Frigg, and Verdandi.

### 5.2.1 Software and Hardware Platforms

For the control system to run, the different VIs need to interact with each other. In real ROV operations, that is, field trails, the HIL-simulator can be discarded. The control system is instead implemented using LabVIEW software on a compact Reconfigurable Inputs and Outputs (cRIO) platform, which is a real-time hardware platform by National Instruments commonly used in the control system industry[41]. The cRIO consists of a user-programmable Field-Programmable Gate Array (FPGA) ensuring communication between software modules and Input-output (IO) modules enabling signal processing for sensor output. The GUI is provided by a host computer running the LabVIEW based GUI VI, which is FPGA-compatible. Additional information from cameras and sonars can be accessed via User Datagram Protocol (UDP) link.

### 5.2.2 Hardware-in-the-Loop (HIL) Simulations

The LabVIEW VI Verdandi is providing HIL-simulation capabilities, which are useful in terms of adding new functionalities and modifications to the control system. Through such simulations, performance testing and debugging can be done more easily before field trials are carried out. HIL-simulations enables simulation of realistic scenarios due to the incorporation of hardware components combined with mathematical models[32]. Verdandi enables HIL-simulations by inter-connecting the cRIO platform-based control system using Transmission Control Protocol (TCP) connection[42].

## 5.3 Frigg VI - Graphical User Interface

The ROV motion control system is operated by the user from the Frigg VI acting as the GUI, depicted in Figure 5.2, while the control system, Njord VI, and alternatively the Verdandi VI are running simultaneously. When the control loop is initiated a suitable position for the field ROV operation, or HIL-simulation is defined. To the left in Figure 5.2, the user can choose among different missions with different levels of autonomy. The different missions are initiated by some desired input, acting as input to the guidance module, following Figure 3.2. In the middle of Figure 5.2, the user has an overview of the different sensors utilized by the ROV for navigation. Green lights indicate that the sensors function properly, while red lights indicate offline or faulty sensors. Furthermore, Figure 5.2 is a screenshot taken during a tracking operation. The map in the middle of the Figure represents the ROV by the small green box, while the red line represents the previous trajectory of the ROV and the blue line and dots represent the further planned path and waypoints, respectively. In the upper right corner, the user can access information about the ROV, such as desired and estimated states like heading, altitude, depth. In the lower right corner, the user can control other functionalities of the ROV, such as cameras and lights.

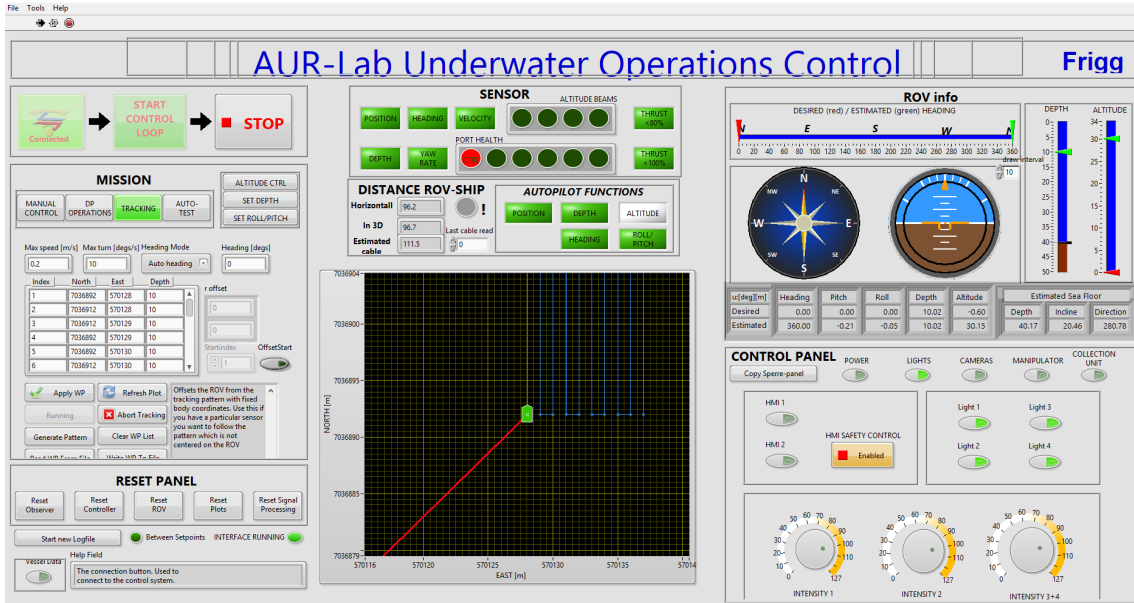


Figure 5.2: Frigg VI

## 5.4 Njord VI - ROV Motion Control System

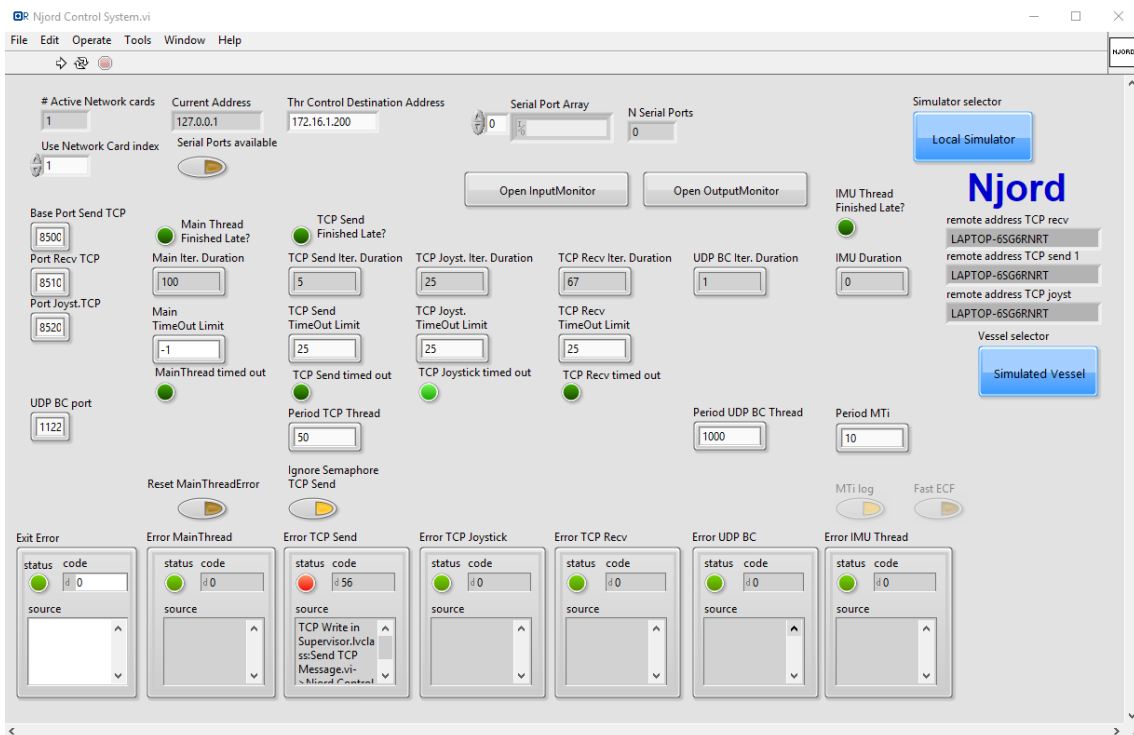


Figure 5.3: Njord VI

The control system itself is represented in the Njord VI, depicted in Figure 5.3. The Njord VI has to be running simultaneously with Frigg, however during ROV operations or simulations it mostly provides error messages (if any) and the overall status of the control system.

## 5.5 Verdandi VI - Hardware-In-The Loop (HIL) Simulation

In the case of HIL-simulations, the Verdandi VI is running together with the Njord VI and the Frigg VI. A screenshot of the Verdandi VI is given in Figure 5.4. As mentioned in Section 5.2.2, HIL-simulations enables simulations of realistic scenarios without access to actual hardware. By using Verdandi, the user can simulate different environmental forces and sensor noise acting on the ROV. Also, Verdandi enables customizing the seafloor for HIL-simulations. The seafloor is defined as a plane as shown in (5.1) and can be defined by the user by changing the coefficients, namely,  $a_0$ ,  $a_1$ ,  $a_2$ ,  $b_1$  and  $b_2$ .

$$z(x, y) = a_0 + a_1 \cdot x + b_1 \cdot y + a_2 \cdot x^2 + b_2 \cdot y^2 \quad (5.1)$$

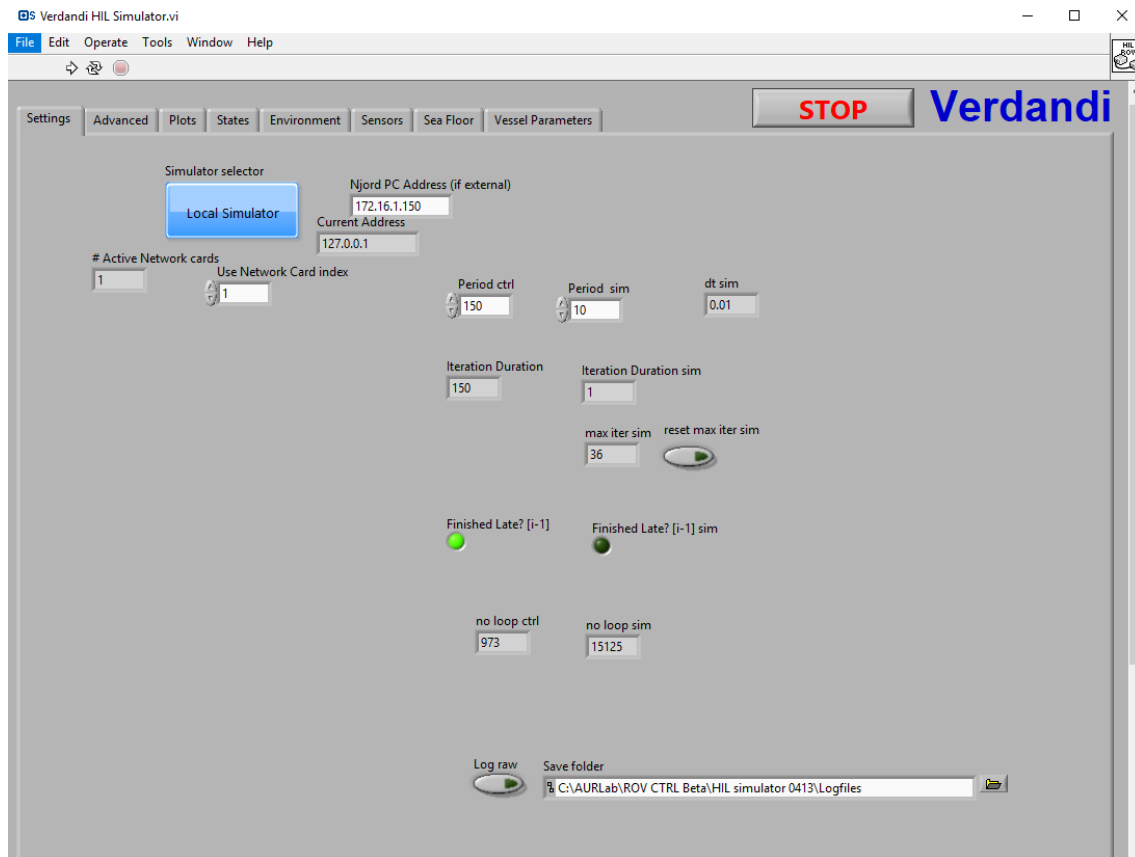


Figure 5.4: Verdandi VI

## 5.6 Integrated Autonomous System

The Autonomy Window, which is an extension to the Frigg VI described in Section 5.3, enables the user to initiate highly autonomous ROV missions. The autonomy window was initially developed in [2] is based on the control architecture reviewed in Section 1.5.1. This window acts as a framework for further implementation of autonomous functionality into the control system. In this thesis, the autonomy window is augmented to include optimal path planning capabilities for autonomous docking missions.

A screenshot of the autonomy windows graphical user interface is depicted in Figure 5.5, where the



part of the GUI enclosed by the yellow rectangle represents the optimal path planning capabilities, while the part enclosed by the red rectangle represents the autonomy module enabling the ROV to perform autonomous tasks. The autonomy module can perform different missions in accordance with different modes, such as a transit state where the ROV performs an autonomous trajectory between waypoints. Eventually, the transit state is used together with the path planning module in simulations of the integrated system. The method behind this is quite simple. The waypoints generated from the path planning module are saved in the autonomy window as transit waypoints for the ROV to follow in the transit state.

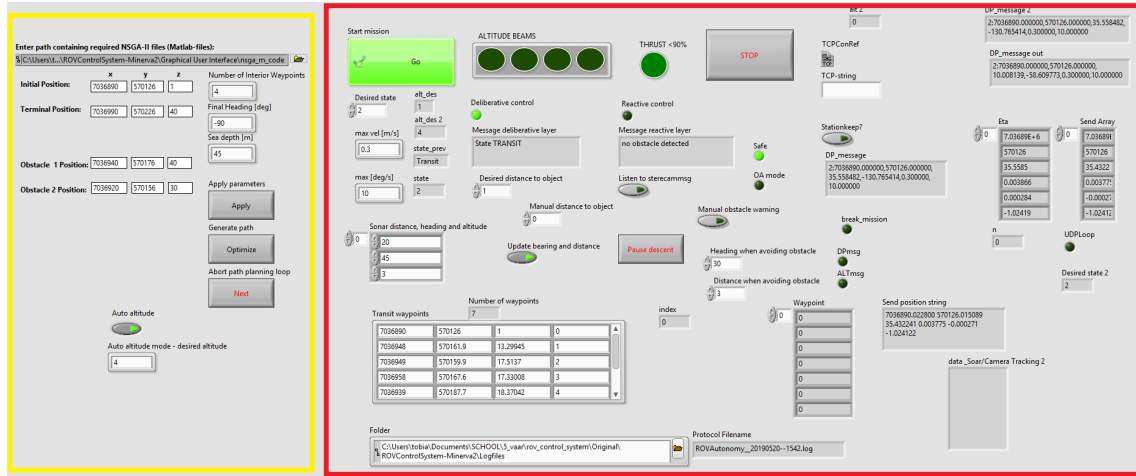


Figure 5.5: Autonomy window with optimal path planning capabilities

A more close-up screenshot of the graphical user interface concerning the path planning module, i.e. the part of the screenshot in Figure 5.5 enclosed by the yellow rectangle, is provided in Figure 5.6.

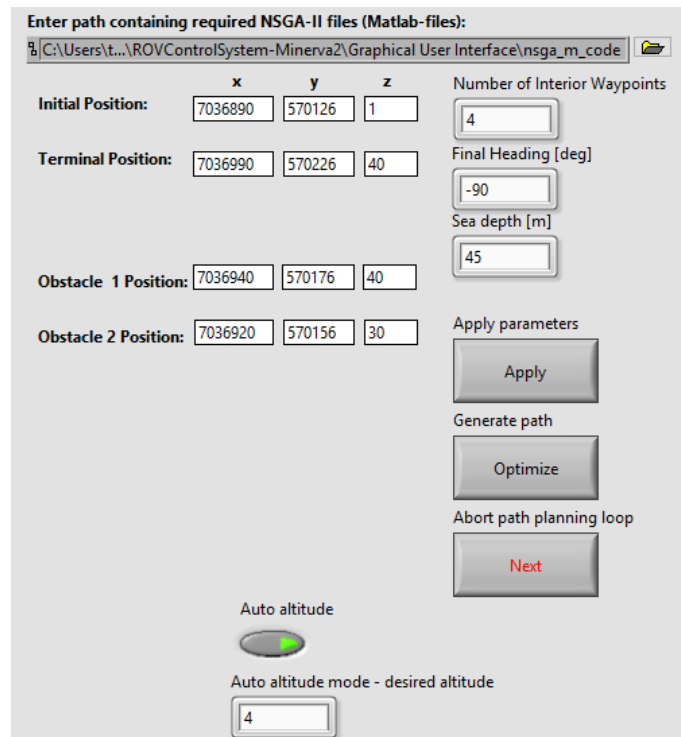


Figure 5.6: Zoomed path planning module graphical user interface

Now, it is time to describe the functionality of the LabVIEW based path planning module. By utilization of the MatlabScript Node, LabVIEW is able to call user-defined functions coded in Matlab. The LabVIEW optimal path planning module runs within a separate loop in the autonomy window. When the optimization is performed, and the path selector chooses a path, a Matlab plot showing the selected path in three dimensions and other plots pops up, allowing examination of the optimization performance. Thus, the user gets the opportunity to perform the path planning several times before applying the generated path to transit waypoints, in case the selected path or optimization performance is not found satisfactorily by visual inspection.

In the uppermost part of Figure 5.6, LabVIEW takes in a path referring to a directory containing Matlab code similar to the one described in accordance with Algorithm 5 in Section 5.1, however, there are some differences in terms of input/output handling. For post-processing purposes, whenever an optimization is complete, the selected path and the final population are saved to separate text files in the same path as the Matlab code.

Next, the path planning module requires several numeric user-defined inputs for the optimization process. Example values of those inputs are presented in Figure 5.6, namely, initial, terminal, and two obstacle positions all represented in the global NED-frame. Also, the number of interior waypoints, that is non-fixed waypoints, the final heading, and the sea depth in the environment are user-defined inputs. Other parameters, such as the NSGA-II parameters presented in Table 4.1 and the remainder of the parameters presented in table 3.2 are assigned fixed values according to preference and suggestions in literature. These parameters can, of course, be changed from a developer point of view. However, they can be claimed to be assigned suitable values in the design of the optimal path planner.

Furthermore, the path planning module graphical user interface, Figure 5.6, includes three buttons featuring functionalities of the module, namely the "Apply"-button, the "Optimize"-button and the "Next"-button. The "Apply"-button applies/resets the user-defined parameters described above to the optimization problem, while the "Optimize"-button generates an optimal path by NSGA-II and the path selector proposed in Algorithm 1. Finally, the "Next"-button aborts the path planning loop such that the user can go on with the autonomous mission, which is transit.

Moreover, when the optimal path planning is done, the autonomous docking mission can start. Here, it is possible to carry out the transit in two different modes, namely auto-altitude mode and auto-depth mode. In the graphical user interface in Figure 5.6, it is possible to switch between these two modes by the on/off switch named "Auto altitude". The path planning is identical for these two modes, whereas the subsequent transit is different, as explained in the following Sections.

### 5.6.1 Auto-Depth Mode

Firstly, an explanation of the auto-depth mode. At this point, the NSGA-II based path planning module has generated a three-dimensional optimal path in global NED-coordinates based on the four-objective optimal path planning problem defined in Section 3.7.7. Furthermore, the auto-depth mode is quite straightforward, as the subsequent autonomous mission consists of transit between the generated waypoints as they are given from the path planning module. The path can be assumed to have sufficient safety margin to obstacles due to the objective function  $O_2$ , whereas, the objective function  $O_4$  is supposed to ensure sufficiently magnitude of the distance to the seafloor, altitude, such that DVL measurements are available while eliminating the risk of bottom collision. The user-defined parameter sea depth is critical in this context.

### 5.6.2 Auto-Altitude Mode

Now, an explanation of the auto-altitude mode. Identically to the auto-depth mode, the NSGA-II based path planning module has generated a three-dimensional optimal path in global NED-

coordinates based on the four-objective optimal path planning problem defined in Section 3.7.7. However, this is where the auto-altitude mode differs from the auto-depth mode. The auto-altitude mode overwrites the D-coordinates of the generated path as it makes the ROV follow the seafloor at a constant desired altitude. The desired altitude applied in the auto-altitude mode is a user-defined input defined in the lowermost part of the graphical user interface, seen in Figure 5.6.

Moreover, it is assumed that sufficient safety margin to obstacles renders a safe and collision-free trajectory even though the D-coordinates are slightly different than those generated by the path planning module. A collision-free trajectory is assumed because the objective function  $O_4$  is included in the path planning problem also for the auto-altitude mode. The auto-altitude mode utilized online altitude control, described in Section 3.6, and thus, a gentle altitude profile can be expected. This approach aims to eliminate the risk of bottom collision while guaranteeing the availability of DVL measurements.



# Chapter 6

## Results

This Chapter contains results from different phases of the development of an optimal path planner based on NSGA-II. Note that the problem that is solved is the optimal path planning problem defined in Section 3.7.7, by applying the NSGA-II based optimal path planner described in Section 5.1. Simulations of autonomous docking missions are carried out by the integrated system described in Section 5.6. Overall the results can be divided into two different categories:

1. Results on the `Matlab`-coded NSGA-II based path planner as a stand-alone application, to examine the performance and highlight interesting factors in solving such optimization problems. Section 6.1 presents these results.
2. Results from the integrated system that is the implementation of the NSGA-II path planner into the control system of the ROV Minerva 2 tested in HIL simulations. Section 6.2 and 6.3 presents these results.

### 6.1 Optimization

Post-processing functionality in the `Matlab` directory enables generating different results from the offline path planning and examining the performance of the NSGA-II based optimal path planner. In all results, the general NSGA-II parameter settings are as given in Table 6.1.

Table 6.1: NSGA-II parameters

Property	Symbol	Value
Population size	$N$	100
Number of generations	$G$	500
SBX distribution index	$\eta_c$	5
SBX probability	$p_c$	0.9
Mutation distribution index	$\eta_m$	20
Mutation probability	$p_m$	0.1

General parameter setting for all results obtained for path planning done using the `Matlab`-code as a standalone application is given in Table 6.2.

Table 6.2: General path planning problem parameters

Property	Symbol	Value	Unit
Initial position	$(x_0, y_0, z_0)$	(1.0, 1.0, 1.0)	[m]
Final position	$(x_n, y_n, z_m)$	(100, 100, 50)	[m]
Obstacle 1 position	$(x_{o1}, y_{o1}, z_{o1})$	(50, 50, 40)	[m]
Obstacle 2 position	$(x_{o2}, y_{o2}, z_{o2})$	(30, 30, 30)	[m]
Obstacle radius		10	[m]

In addition to the parameters given in Table 6.2, some additional parameters are defined individually depending on the problem at hand.

### 6.1.1 Path Planner with Three Objectives, $O_1$ , $O_2$ and $O_3$

Initially, the optimal path problem defined in Section 3.7.7 is solved by the NSGA-II based optimal path planner concerning three objectives. Objectives  $O_1$ ,  $O_2$  and  $O_3$  are considered, thus  $O_4$  is omitted in this case. In addition to the general parameters applied to the optimal path planning problem, given in Table 6.2, the parameters given in Table 6.3 are applied when solving the problem utilizing a three-objective optimal path planner.

Table 6.3: Additional path planning problem parameters

Property	Symbol	Value	Unit
Final heading	$\psi_n$	-90	[°]
Number of interior waypoints	$K$	4	
Number of objectives	$M$	3	
Sea depth	$z_{\max}$	100	[m]
Safety margin to obstacles acting in $O_2$	$m$	5	[m]

Furthermore, in the post-optimization path selection, the parameter setting for the path selector shown in Table 6.4 is applied. Note that  $\beta_{O_4}$  is undefined since, for this application, the objective function  $O_4$  is omitted.

Table 6.4: Path selector parameters

	Value
$\beta_{O_2}$	0.8

Figure 6.1 depicts a three-dimensional plot of the selected optimal path,  $\mathcal{P}_{\text{selected}}$ . The path  $\mathcal{P}_{\text{selected}}$  is a member of the final population generated by the three-objective NSGA-II based optimal path planner and is selected by the path selector proposed in Algorithm 1. Figures 6.2 and 6.3 depict the projection of  $\mathcal{P}_{\text{selected}}$  in the NE-plane and the ED-plane, respectively. In the Figures 6.1-6.3 the initial position and the terminal position are indicated by a blue and a green dot, respectively, corresponding to the positions given in table 6.2. Furthermore, the remaining waypoints are indicated by black dots. The path consisting of straight lines interconnecting the waypoints are indicated by red lines. The two obstacles present in the environment are indicated by the two spheres in the plots, located at the positions indicated in Table 6.2.

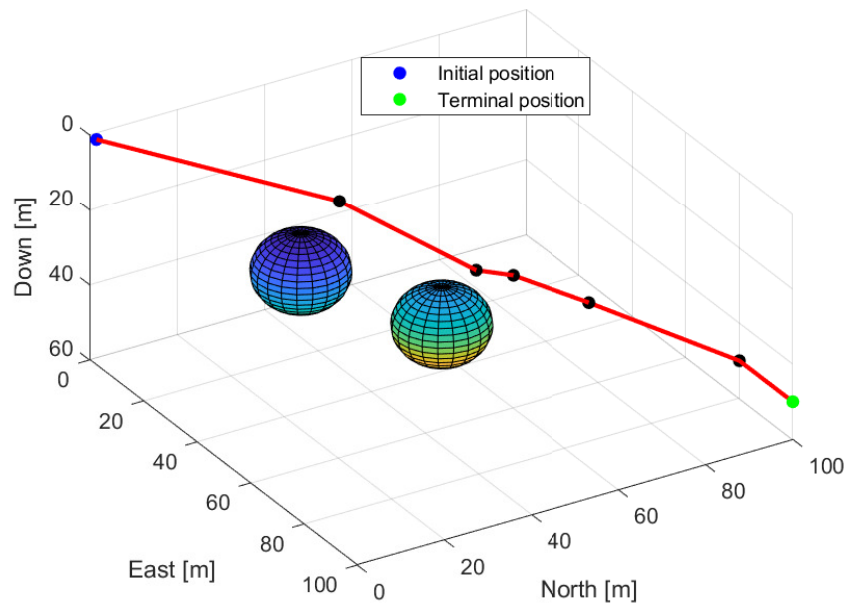


Figure 6.1: Selected path depicted in 3D

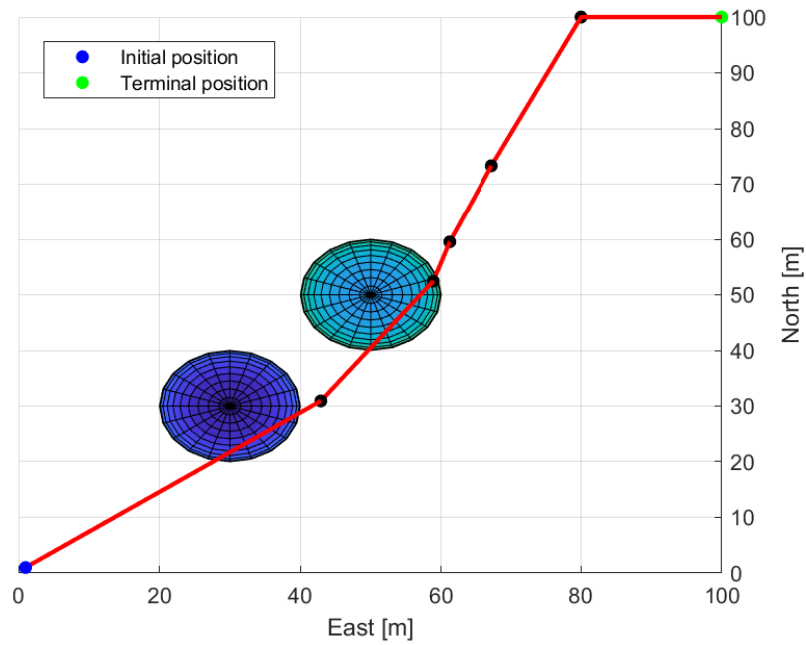


Figure 6.2: Selected path in the NE-plane

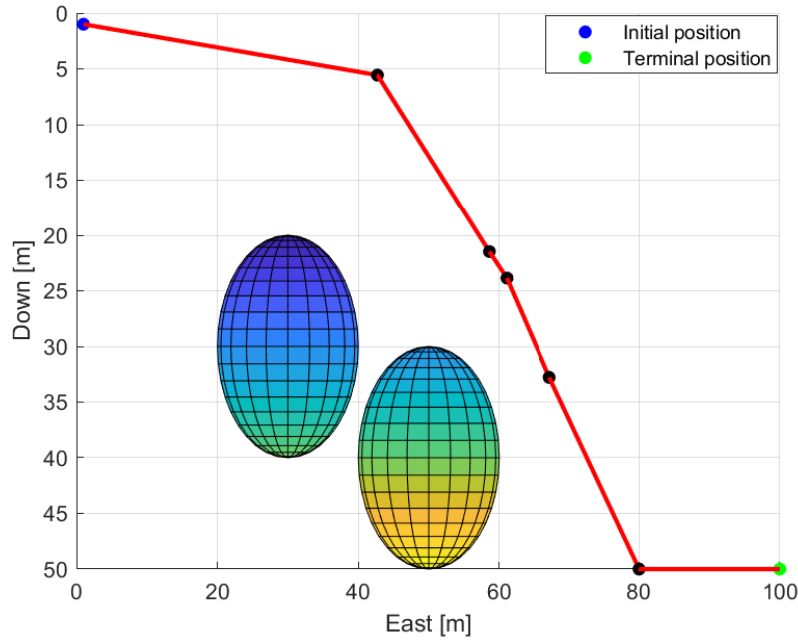


Figure 6.3: Selected path in the ED-plane

Objective function values of the selected path,  $\mathcal{P}_{\text{selected}}$  is given in Table 6.5.

Table 6.5: Properties of the selected path ( $\mathcal{P}_{\text{selected}}$ ),  $M = 3$ 

Objective	Value	Unit	
Path length	$O_1(\mathcal{P}_{\text{selected}})$	162.2595	[m]
Safety margin	$O_2(\mathcal{P}_{\text{selected}})$	-11.1240	[-]
Sharpest turn in the NE-plane	$O_3(\mathcal{P}_{\text{selected}})$	0.3011	[rad]

Furthermore, post-processing results representing data on the final population of solutions,  $P$ , generated by the three-objective NSGA-II based optimal path planner are given. A three-dimensional plot of the three objective function values,  $O_1, O_2, O_3$ , for every solution in  $P$  is provided in Figure 6.4. Similarly, two-dimensional plots comparing the objective function values  $O_1$  versus  $O_2$ ,  $O_1$  versus  $O_3$ , and  $O_2$  versus  $O_3$ , of every solution in the final population  $P$  are depicted in Figures 6.5, 6.6 and 6.7, respectively. For curiosity, the chosen solution,  $\mathcal{P}_{\text{selected}}$  is represented by a red dot in these figures.



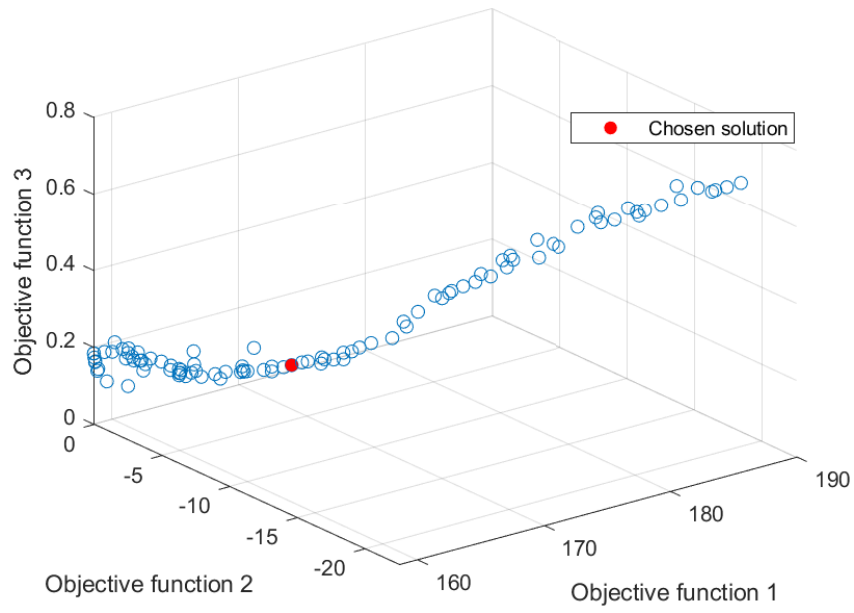
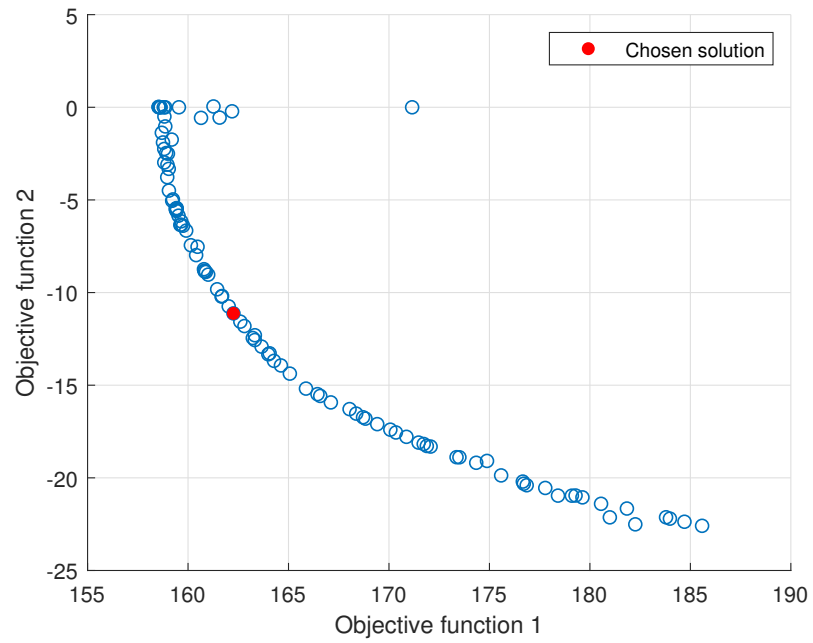
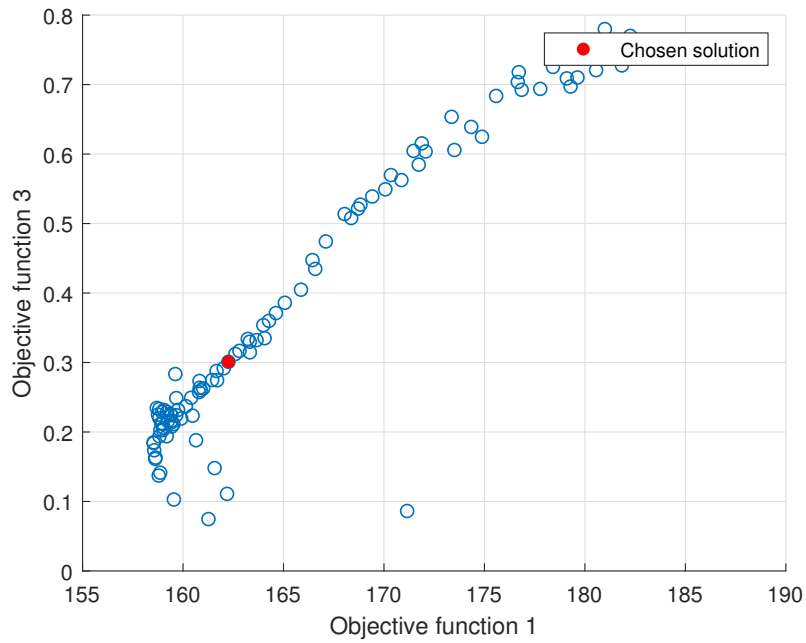
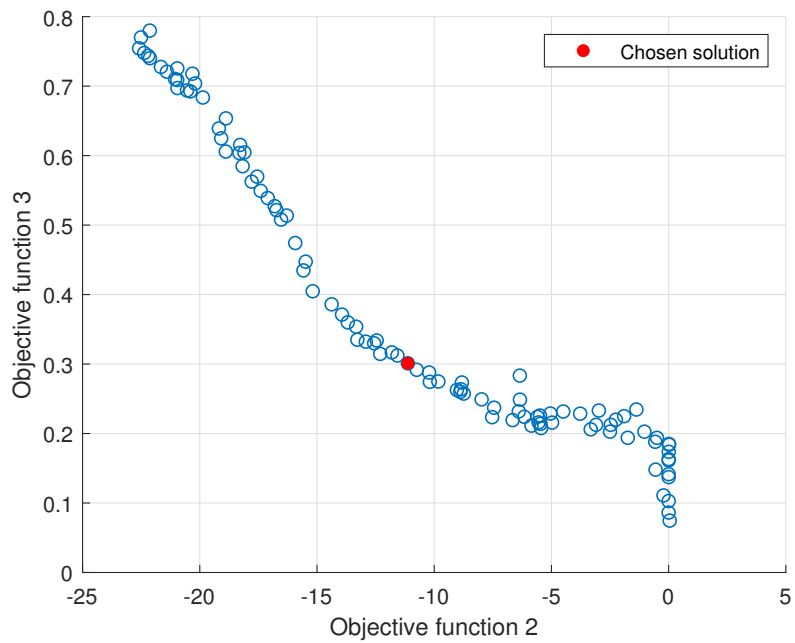


Figure 6.4: 3D plot of objective function values in the final population

Figure 6.5: 2D plot of objective function values  $O_1$  versus  $O_2$  in the final population

Figure 6.6: 2D plot of objective function values  $O_1$  versus  $O_3$  in the final populationFigure 6.7: 2D plot of objective function values  $O_2$  versus  $O_3$  in the final population

Finally, Table 6.6 provides some numerical data on the final population of paths,  $P$ , generated by the three-objective NSGA-II based optimal path planner.

Table 6.6: Final population,  $P$ , numeric properties,  $M = 3$ 

	<b>Value</b>	<b>Unit</b>
$O_{1,\min} \in P$	158.5337	[m]
$\bar{O}_1 \in P$	165.8948	[m]
$\bar{O}_2 \in P$	-10.7911	[-]
$O_{2,\text{required}}$	-8.6329	[-]
$\bar{O}_3 \in P$	0.3811	[rad]
Number of paths discarded by path selector	1	

### 6.1.2 Path Planner with Four Objectives, $O_1$ , $O_2$ , $O_3$ and $O_4$

Next, the optimal path problem defined in Section 3.7.7 is solved by the NSGA-II based optimal path planner concerning four objectives. Objectives  $O_1$ ,  $O_2$ ,  $O_3$  and  $O_4$  are considered.

In addition to the general parameters applied to the path planning problem, given in Table 6.2, the parameters given in Table 6.7 are applied when solving the problem utilizing a four-objective path planner.

Table 6.7: Additional path planning problem parameters

<b>Property</b>	<b>Symbol</b>	<b>Value</b>	<b>Unit</b>
Final heading	$\psi_n$	-90	[°]
Number of interior waypoints	$K$	4	
Number of objectives	$M$	4	
Sea depth	$z_{\max}$	100	[m]
Safety margin to obstacles acting in $O_2$	$m$	5	[m]
Desired depth	$z_{\text{desired}}$	70	[m]
Altitude limit acting in $O_4$	$z_{\text{limit}}$	10	[m]

Furthermore, in the four-objective optimal path planning scenario, in the post-optimization path selection the parameter setting for the path selector is shown in Table 6.8 is applied.

Table 6.8: Path selector parameters

	<b>Value</b>
$\beta_{O2}$	0.8
$\beta_{O4}$	1.0

The selected path from the four-objective NSGA-II based optimal path planner is depicted in Figures 6.8, 6.9 and 6.10. These Figures takes on the same format as Figures 6.1, 6.1 and 6.3, respectively.

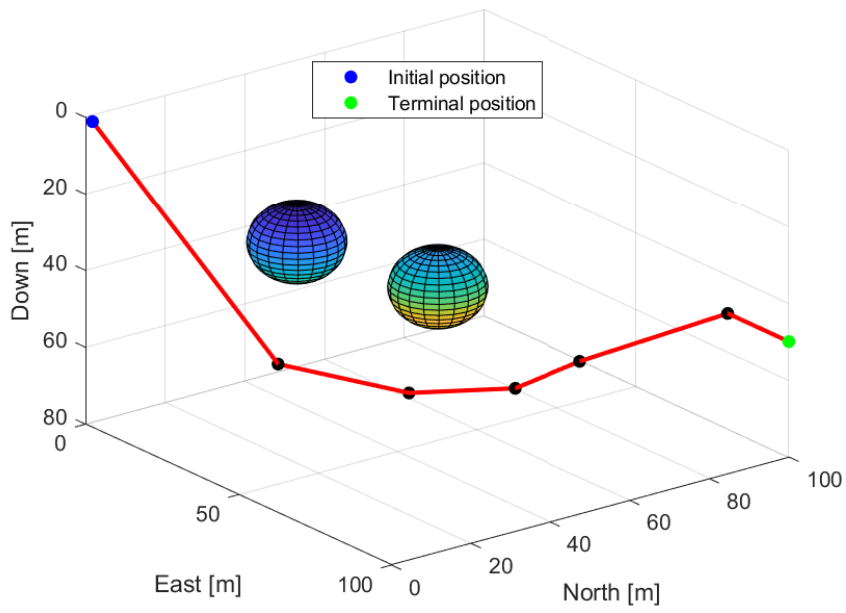


Figure 6.8: Selected path from four-objective optimal path planner

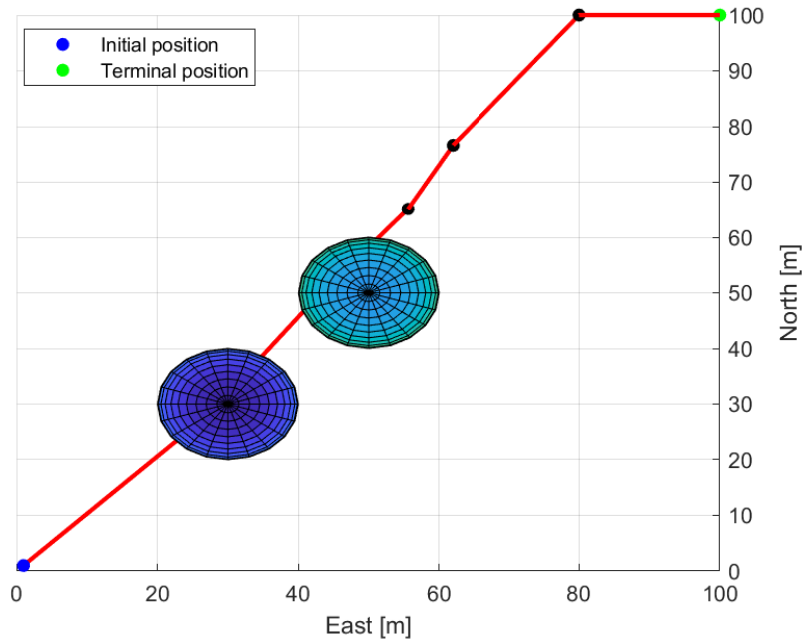


Figure 6.9: Selected path in the NE-plane

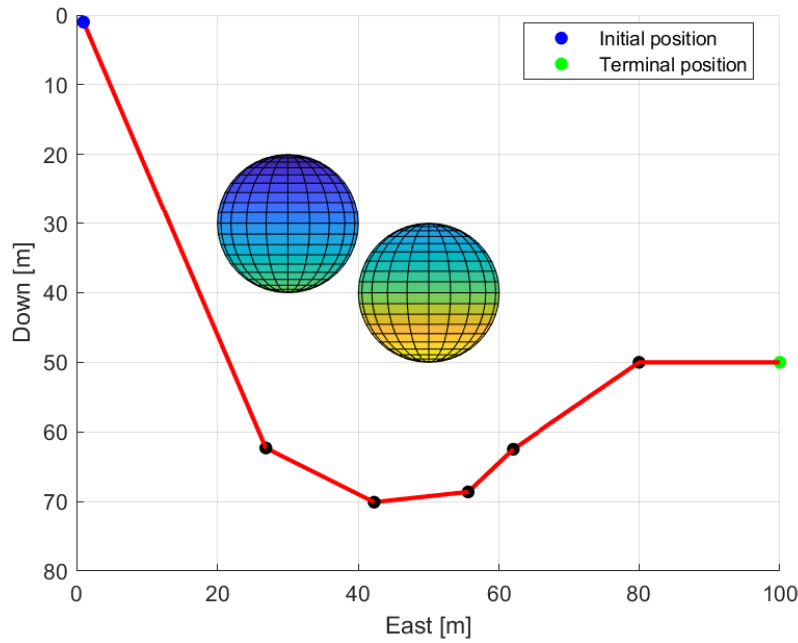


Figure 6.10: Selected path in the ED-plane

Objective function values of the selected path,  $\mathcal{P}_{\text{selected}}$ , in the four-objective NSGA-II based optimal path planner scenario is given in Table 6.9.

Table 6.9: Properties of the selected path,  $\mathcal{P}_{\text{selected}}$ ,  $M = 4$ 

Objective	Value	Unit	
Path length	$O_1(\mathcal{P}_{\text{selected}})$	186.6444	[m]
Safety margin	$O_2(\mathcal{P}_{\text{selected}})$	-5.0394	[-]
Sharpest turn in NE-plane	$O_3(\mathcal{P}_{\text{selected}})$	0.1730	[rad]
Largest depth deviation	$O_4(\mathcal{P}_{\text{selected}})$	-2.3090	[-]

Moreover, several plots are provided highlighting the performance of the optimization in the four-objective NSGA-II based optimal path planner scenario by comparison of the objective function values of the final population of paths,  $P$ . Figures 6.11, 6.12, 6.13 and 6.14 represents three-dimensional plots of the objective function values. More specific, Figure 6.11 compares  $O_1$ ,  $O_2$  and  $O_3$ , Figure 6.12 compares  $O_1$ ,  $O_2$  and  $O_4$ , Figure 6.13 compares  $O_1$ ,  $O_3$  and  $O_4$  and Figure 6.14 compares  $O_2$ ,  $O_3$  and  $O_4$ .

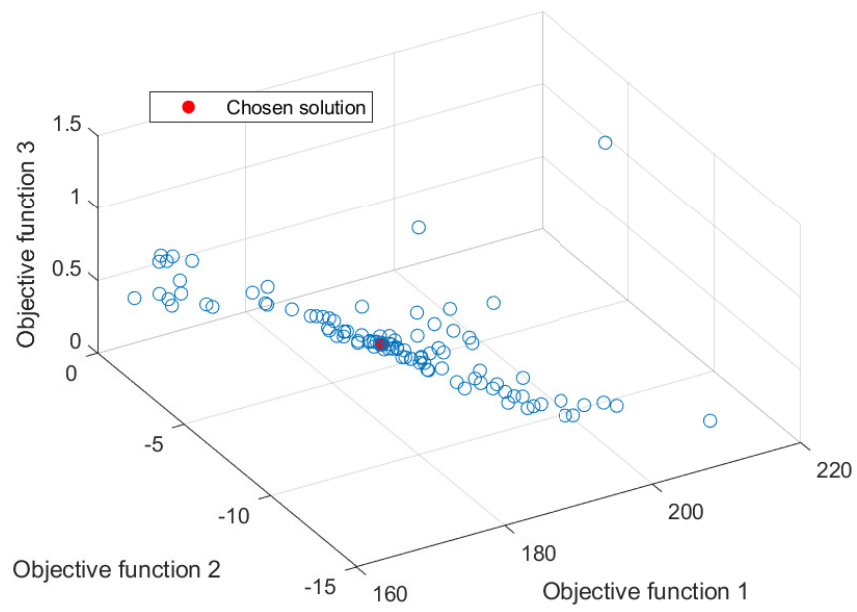


Figure 6.11: 3D plot of objective function values  $O_1$ ,  $O_2$ ,  $O_3$  in the final population

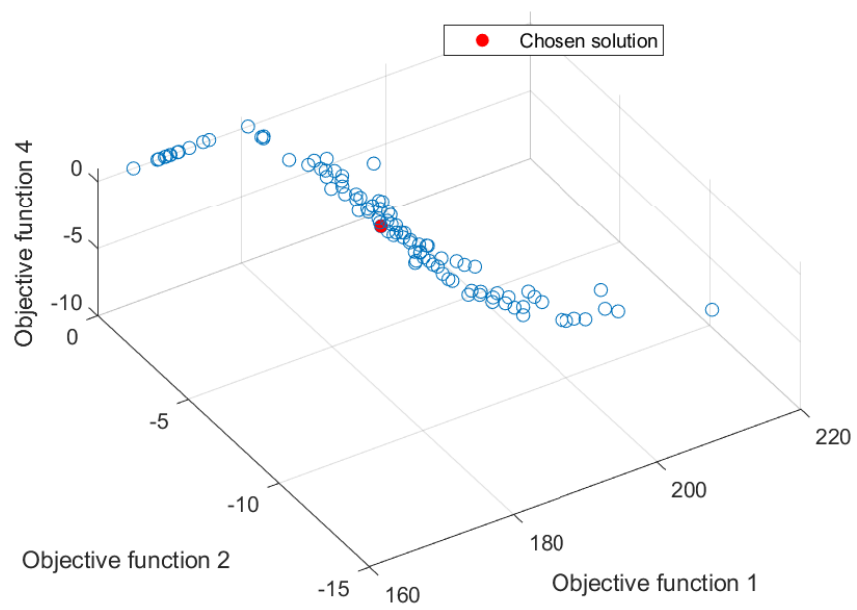


Figure 6.12: 3D plot of objective function values  $O_1$ ,  $O_2$ ,  $O_4$  in the final population

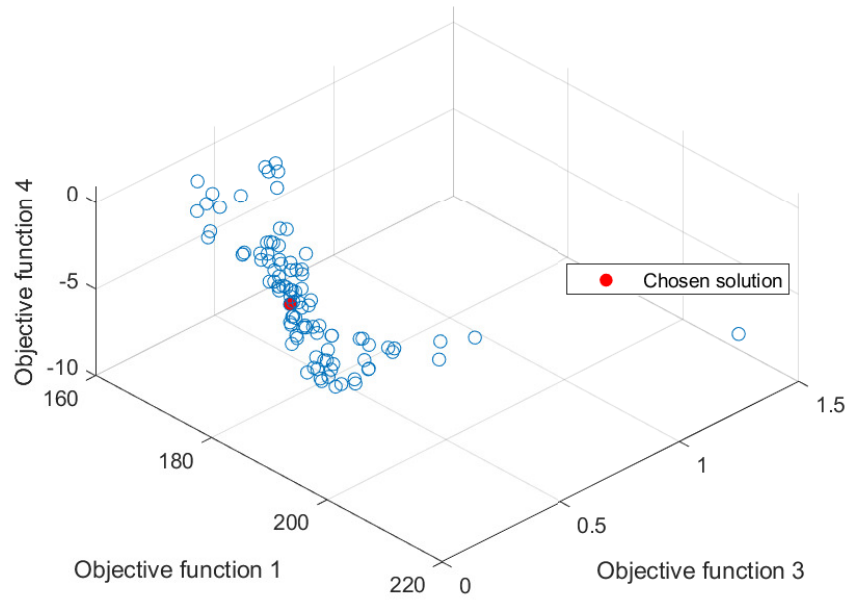


Figure 6.13: 3D plot of objective function values  $O_1$ ,  $O_3$ ,  $O_4$  in the final population

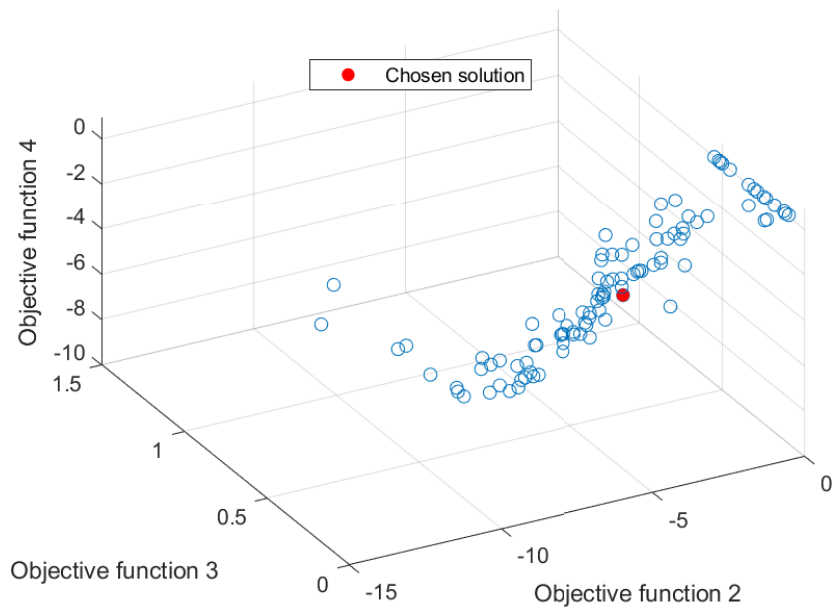


Figure 6.14: 3D plot of objective function values  $O_2$ ,  $O_3$ ,  $O_4$  in the final population

Finally, Table 6.6 provides some numerical data on the final population of paths,  $P$ , generated by the four-objective NSGA-II based optimal path planner.

Table 6.10: Final population,  $P$ , numeric properties,  $M = 4$ 

Property	Symbol	Value	Unit
Shortest path in the final population	$O_{1,min} \in P$	164.9809	[m]
Average objective function value	$\bar{O}_1 \in P$	187.1052	[m]
Average objective function value	$\bar{O}_2 \in P$	-5.3019	[-]
Maximum allowed margin in path selector	$O_{2,required}$	-4.2415	[-]
Average objective function value	$\bar{O}_3 \in P$	0.2219	[rad]
Average objective function value	$\bar{O}_4 \in P$	-2.1799	[-]
Maximum allowed margin in path selector	$O_{4,required}$	-2.1799	[-]
Number of paths discarded by path selector		41	

## 6.2 Integrated System

For the integrated systems the optimal path planning parameters are defined such that they match those applied when performing real ROV operations using the LabVIEW-coded control system described in Sections 5.2 - 5.6. Thus, the positions applied when solving the optimal path planning problem correspond to positions in the actual ROV field trial testing area. Furthermore, the optimization problem applied for this application is defined in Section 3.7.7. All four objectives, namely  $O_1$ ,  $O_2$ ,  $O_3$  and  $O_4$  are applied and the NSGA-II parameters are as defined in Table 6.1.

For comparability reasons, the same path is applied for simulations both in auto-altitude mode and auto-depth mode. Parameters corresponding to the optimal path planning problem defined in Section 3.7.7 for the NSGA-II based optimal path planner applied in the integrated system is given in Table 6.11. The ROV starts from the initial position, which is the origin in the test environment and aims to dock at the final position. The docking station is placed at a depth of 36 meters, that is, 4 meters above the seafloor. The environment is assumed to have an approximately flat seafloor at a depth of 40 meters. Furthermore, the applied path selector parameters are as defined in Table 6.8.

Table 6.11: General path planning problem parameters for the integrated system

Property	Symbol	Value	Unit
Initial position	$(x_0, y_0, z_0)$	(7036891, 570126, 0)	[m]
Final position	$(x_n, y_n, z_n)$	(7036990, 570226, 36)	[m]
Obstacle 1 position	$(x_{o1}, y_{o1}, z_{o1})$	(7036940, 570176, 30)	[m]
Obstacle 2 position	$(x_{o2}, y_{o2}, z_{o2})$	(7036920, 570156, 30)	[m]
Obstacle radius		10	[m]
Final heading	$\psi_n$	-90	[°]
Number of interior waypoints	$K$	4	
Number of objectives	$M$	4	
Safety margin to obstacles acting in $O_2$	$m$	5	[m]
Sea depth	$z_{max}$	40	[m]
Desired depth	$z_{desired}$	30	[m]
Altitude limit acting in $O_4$	$z_{limit}$	5	[m]

The selected path from the NSGA-II based optimal path planner in the integrated system is depicted in Figures 6.15, 6.16 and 6.17. These Figures takes on the same format as Figures 6.1, 6.1 and 6.3, respectively.



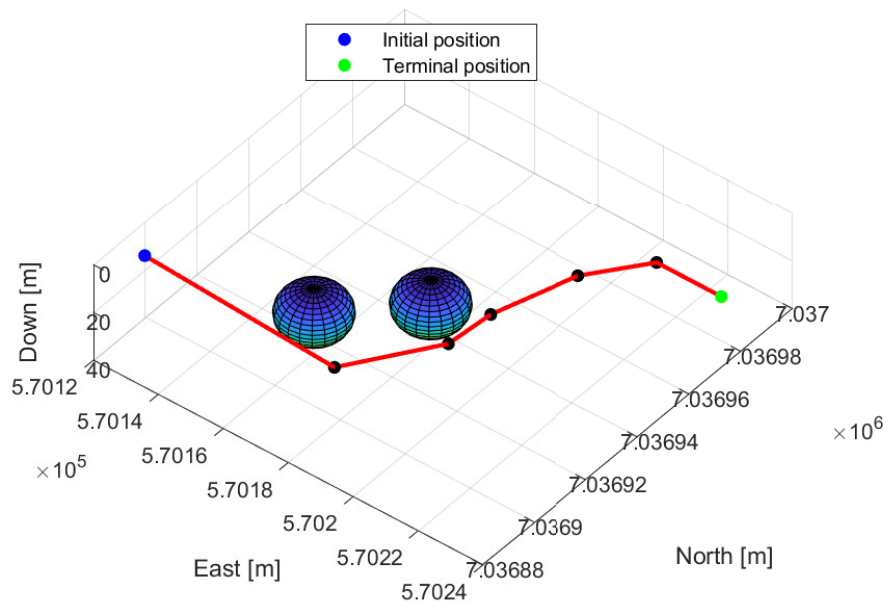


Figure 6.15: 3D plot of the selected path for simulations in the integrated system

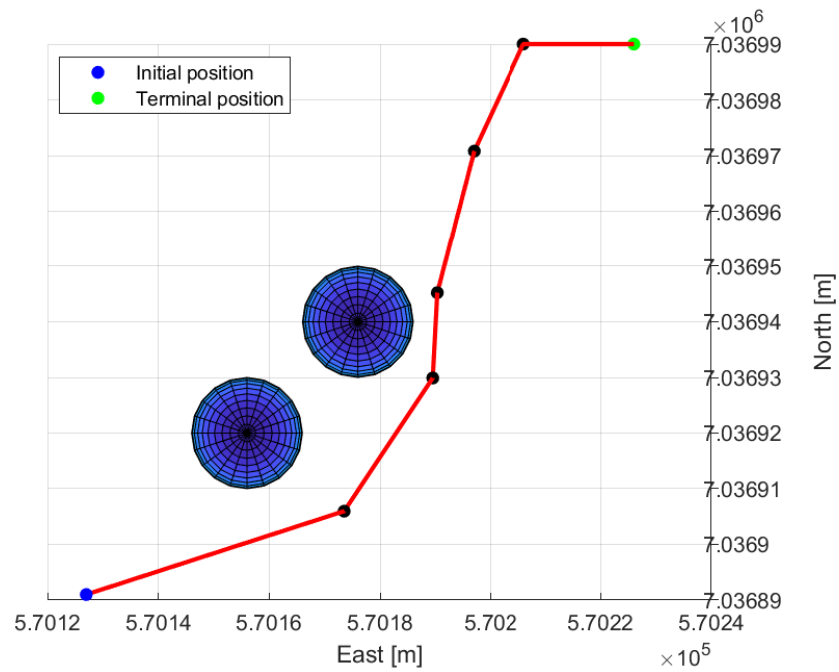


Figure 6.16: Selected path projected in the NE-plane

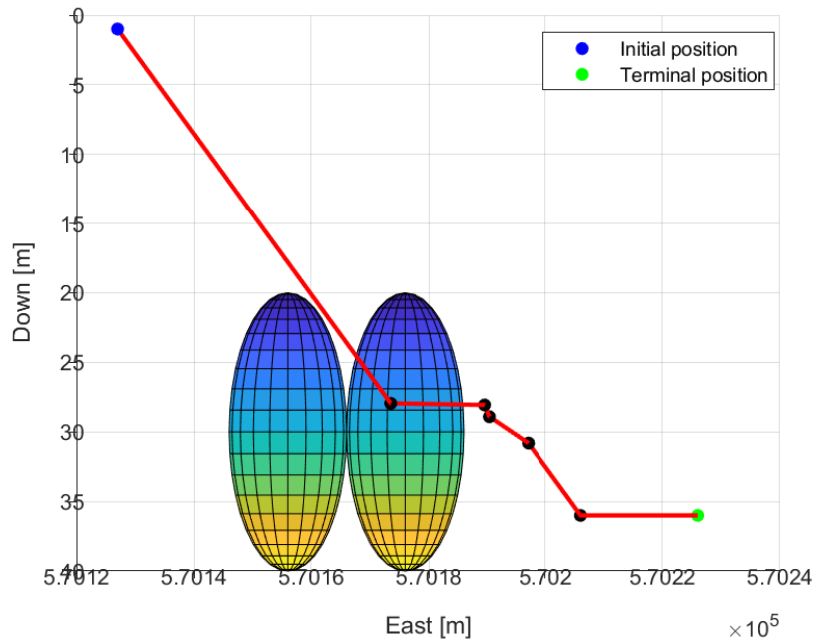


Figure 6.17: Selected path projected in the ED-plane

Objective function values of the selected path,  $\mathcal{P}_{\text{selected}}$ , in the integrated system is given in Table 6.12.

Table 6.12: Properties of the selected path,  $\mathcal{P}_{\text{selected}}$ , auto-altitude mode

Objective	Value	Unit	
Path length	$O_1(\mathcal{P}_{\text{selected}})$	168.2879	[m]
Safety margin	$O_2(\mathcal{P}_{\text{selected}})$	-5.2720	[-]
Sharpest turn in NE-plane	$O_3(\mathcal{P}_{\text{selected}})$	0.6704	[rad]
Largest depth deviation	$O_4(\mathcal{P}_{\text{selected}})$	-2.9496	[-]

Additionally, the waypoints of  $\mathcal{P}_{\text{selected}}$  in NED-coordinates for the integrated system is given in Table 6.13.

Table 6.13: Waypoints of  $\mathcal{P}_{\text{selected}}$  for the integrated system

Waypoint index	North	East	Down
0	7036891	570127	1
1	7036905	570173	27
2	7036929	570189	28
3	7036945	570190	28
4	7036970	570197	30
5	7036990	570206	36
6	7036990	570226	36

Subsequently,  $\mathcal{P}_{\text{selected}}$  is applied to simulations of autonomous docking missions in auto-depth mode and auto-altitude mode.

### 6.2.1 Auto-Depth Mode

Furthermore, simulations of the integrated system in auto-depth mode described in Section 5.6.1 are carried out. Figure 6.18 depicts the trajectory of the ROV Minerva 2 in the three-dimensional environment with the obstacles represented by red spheres. Estimated, measure and desired positions are represented by the blue, dashed black and green lines, respectively.

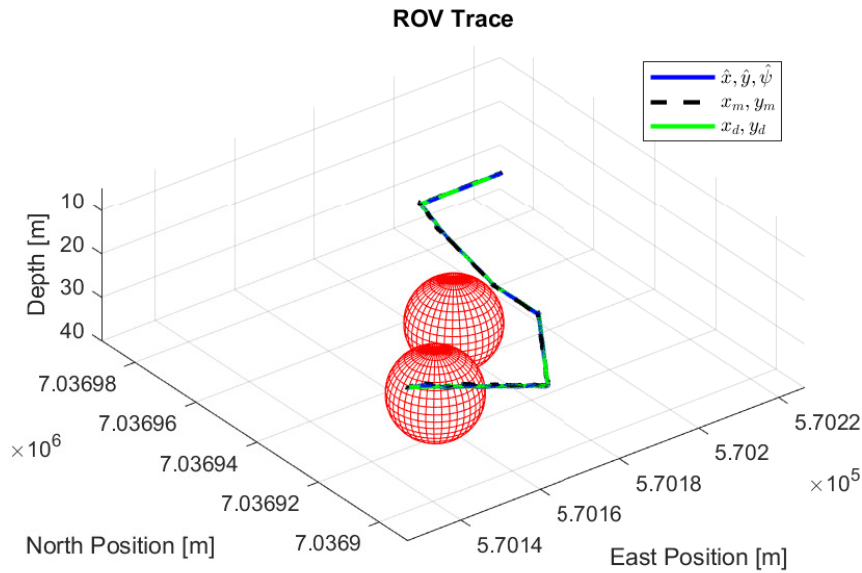


Figure 6.18: 3D ROV trajectory in auto-depth mode

Similarly, Figure 6.19 illustrates the trajectory of Minerva 2 in the NE-plane. The ROV is illustrated by blue rectangles at time instants throughout the simulations, enabling visual inspection of the vehicles heading.

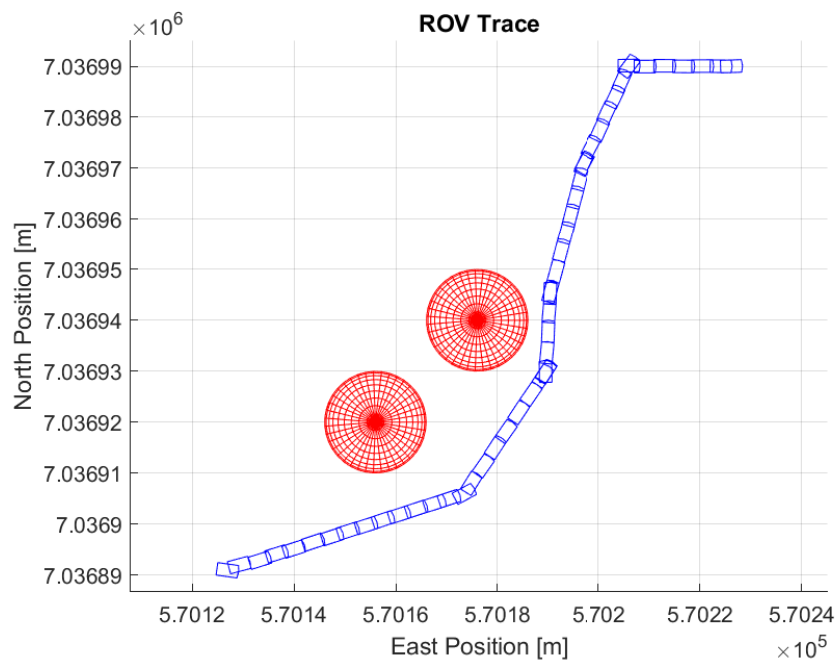


Figure 6.19: 2D ROV trajectory in the NE-plane in auto-depth mode

Moreover, Figure 6.20 gives the time-evolution of the NED-coordinates during the simulation in auto-depth mode. Measured, estimated, and desired coordinates are given by blue, red, and green lines, respectively.

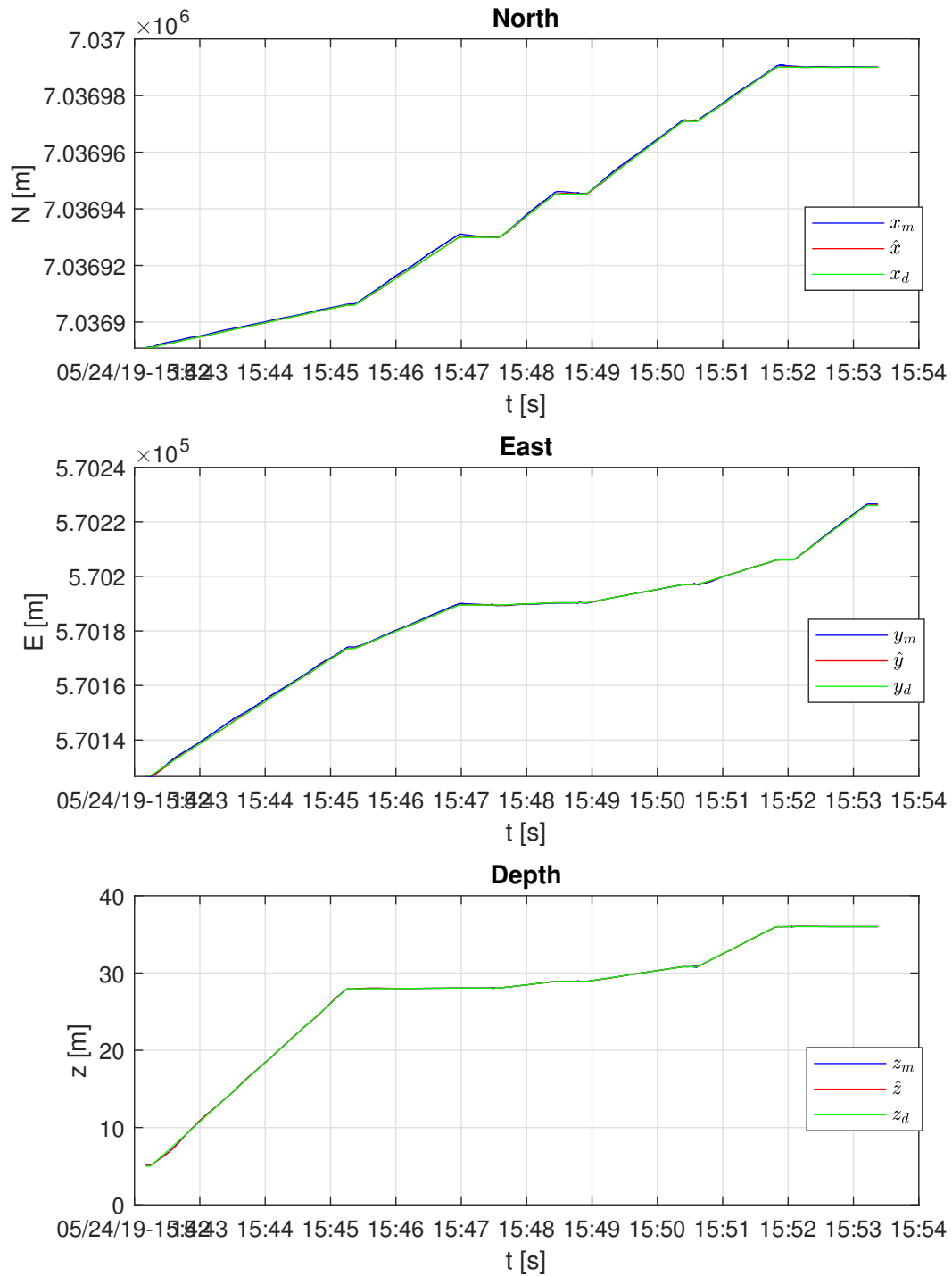


Figure 6.20: Time evolution of NED-coordinates in auto-depth mode

Also, Figure 6.21 represents the thruster rpm percentage according to a constraint of a maximum rpm of 1450 rpm defined in the control system.

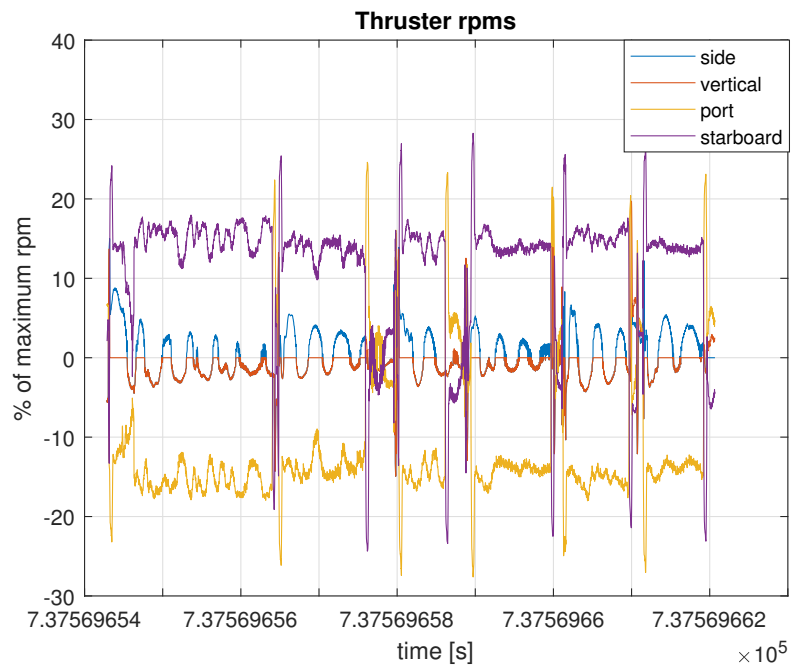


Figure 6.21: Time evolution of thruster %-rpm in auto-depth mode

Lastly, Figure 6.22 represents the time-evolution of the ROV altitude profile throughout the simulation.

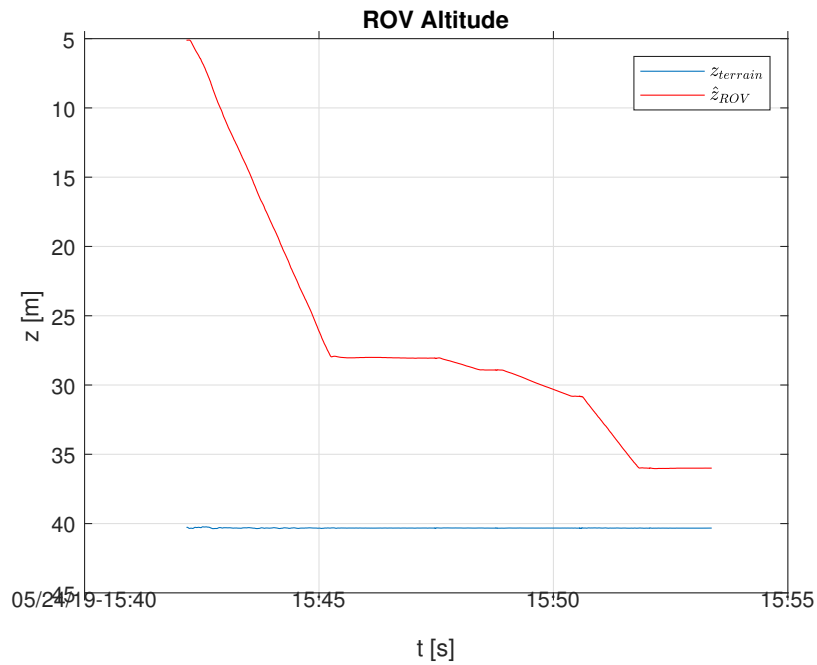


Figure 6.22: Time evolution of ROV altitude in auto-depth mode

### 6.2.2 Auto-Altitude Mode

Moreover, simulation of the integrated system in auto-altitude mode described in Section 5.6.2 is performed. Simulations in auto-altitude mode require additional input, namely desired altitude, which is defined as given in Table 6.14.

Table 6.14: Additional auto-altitude path planning problem parameters

Property	Symbol	Value	Unit
Desired altitude		4	[m]

Figures 6.23 and 6.24 depicts the ROV Minerva 2 trajectory, analogous to to the plots given in Section 6.2.1.

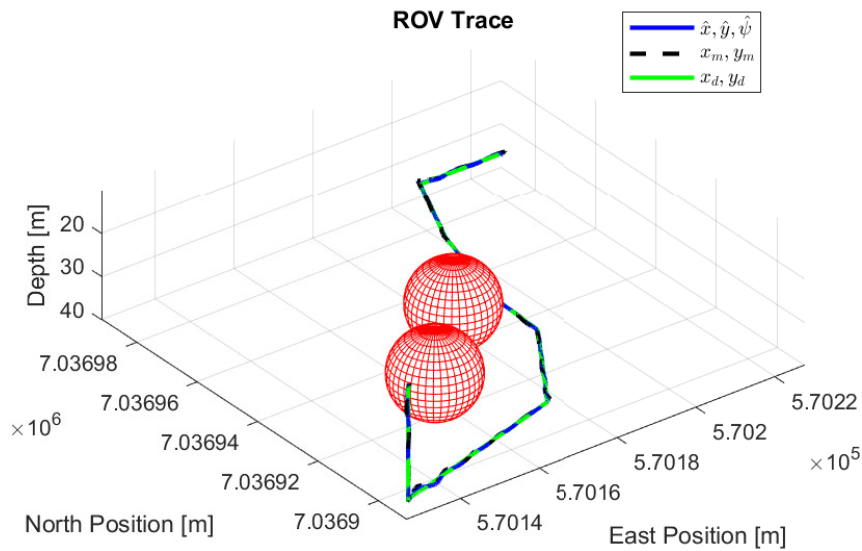


Figure 6.23: 3D ROV trajectory in auto-altitude mode

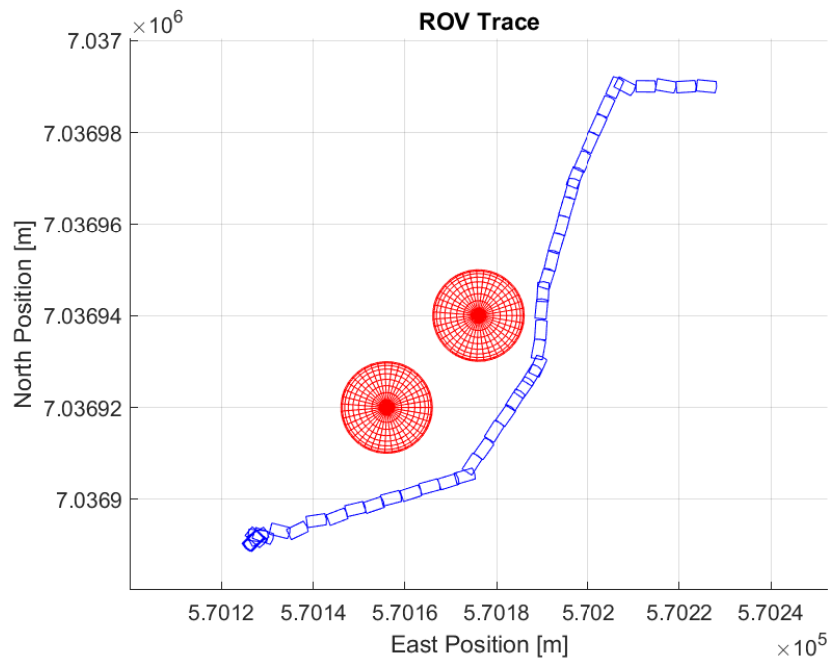


Figure 6.24: 2D ROV trajectory in the NE-plane in auto-altitude mode

Similarly as in section 6.2.2, plots of time evolution of the NED-coordinates, time evolution of thruster rpm percentage and time evolution of ROV altitude are given in Figures 6.25, 6.26 and 6.27, respectively.



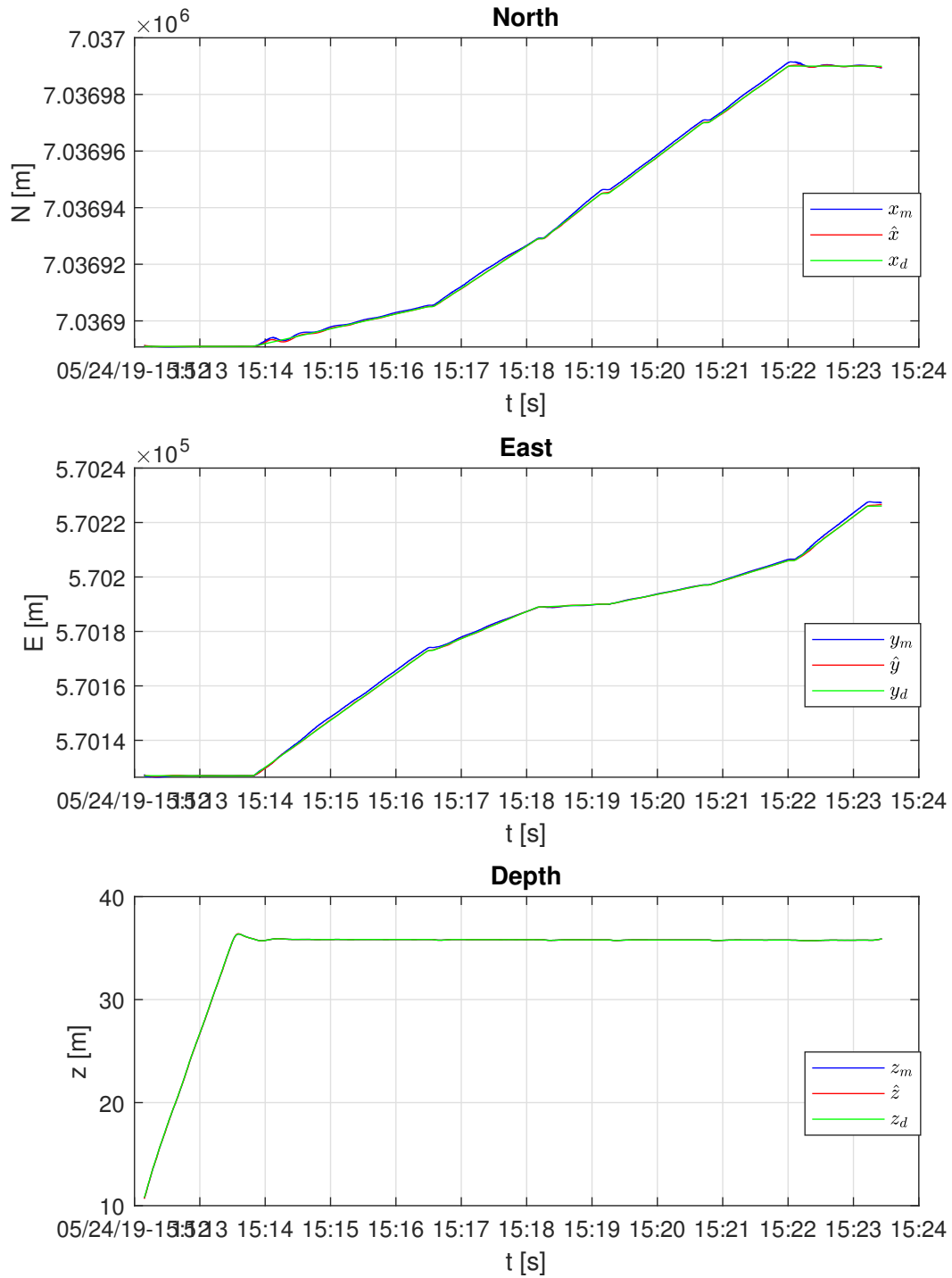


Figure 6.25: Time evolution of NED-coordinates in auto-altitude mode

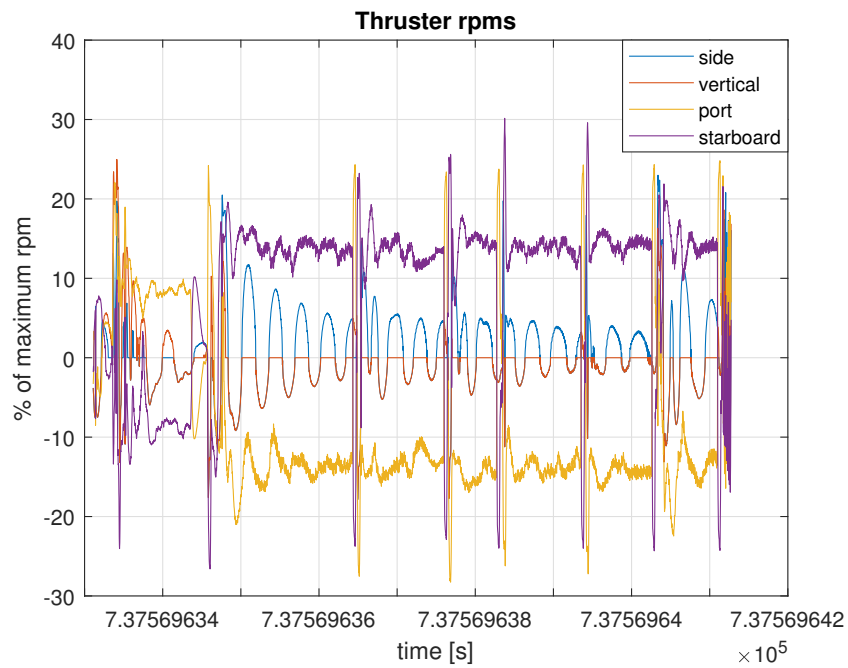


Figure 6.26: Time evolution of thruster %-rpm in auto-altitude mode

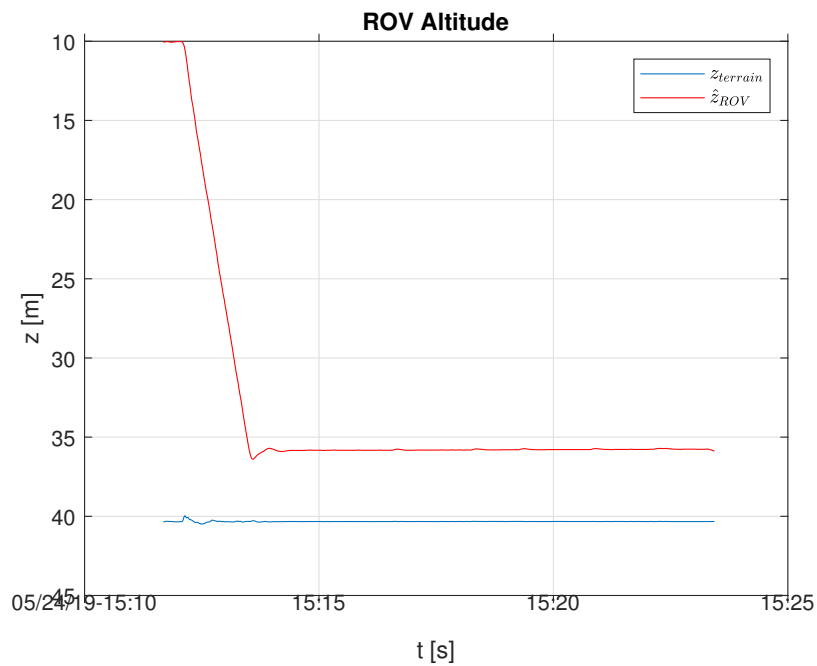


Figure 6.27: Time evolution of ROV altitude in auto-altitude mode

### 6.3 Extended Case: Curved Seafloor

An extended case is defined for simulation and examination of the integrated system in auto-altitude mode. The simulation takes place in an environment with a curved seafloor defined in

Verdandi as described in Section 5.5. The coefficient defining this seafloor corresponding to the coefficients in (5.1) given in Table 6.15, describing a curved seafloor.

Coefficient	Value
$a_0$	40
$a_1$	0.001
$b_1$	0.001
$a_2$	0.004
$b_2$	0.004

Table 6.15: Verdandi seafloorcoefficients

Furthermore, the optimal path planning problem parameters for this scenario is given in Table 6.16. These parameters are applied to the optimal path planning problem defined in Section 3.7.7, including all four objectives, namely,  $O_1$ ,  $O_2$ ,  $O_3$  and  $O_4$ . The NSGA-II parameters are as defined in Table 6.1 and the path selector parameters are as defined in Table 6.8. Note that knowledge about the sea depth is lacking. However, a desired depth of 60 meters is applied to avoid obstacles. Also, the final position representing the position of the docking station is situated four meters above the seafloor.

Table 6.16: Path planning problem parameters for extended case

Property	Symbol	Value	Unit
Initial position	$(x_0, y_0, z_0)$	(7036891, 570126, 0)	[m]
Final position	$(x_n, y_n, z_m)$	(7036990, 570226, 115)	[m]
Obstacle 1 position	$(x_{o1}, y_{o1}, z_{o1})$	(7036940, 570176, 45)	[m]
Obstacle 2 position	$(x_{o2}, y_{o2}, z_{o2})$	(7036920, 570156, 35)	[m]
Obstacle radius		10	[m]
Final heading	$\psi_n$	-90	[°]
Number of interior waypoints	$K$	4	
Number of objectives	$M$	4	
Safety margin to obstacles acting in $O_2$	$m$	5	[m]
Sea depth	$z_{\max}$		[m]
Desired depth	$z_{\text{desired}}$	60	[m]
Altitude limit acting in $O_4$	$z_{\text{limit}}$	5	[m]
Desired altitude		4	[m]

Figure 6.28 provides a three-dimensional graphical representation of the extended case of the optimal path planning problem. The Figure displays the selected path as before, the curved seafloor represented by the plane, and the two obstacles represented by red spheres. Note that the selected path intersects the curved seafloor, and thus, the simulation is performed using auto-altitude mode at the desired altitude indicated in Table 6.16.

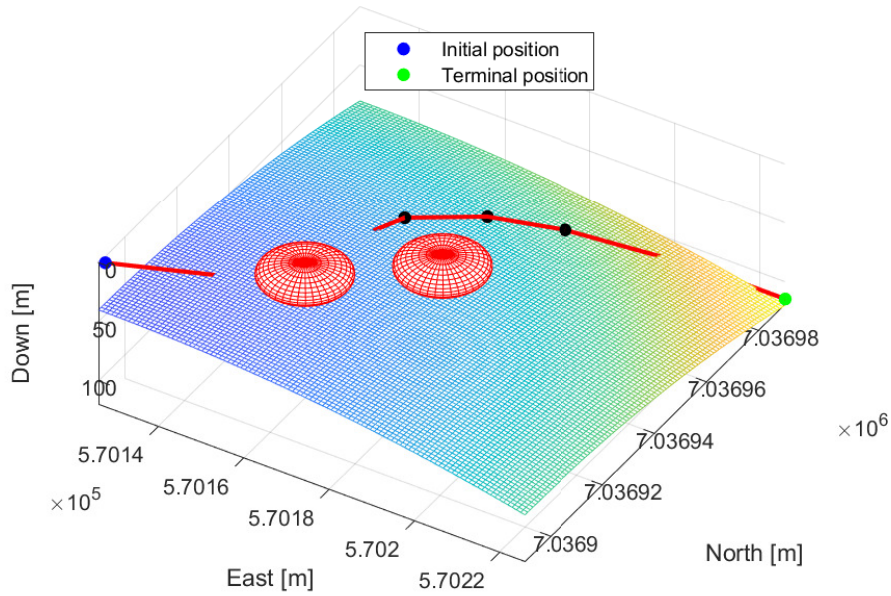


Figure 6.28: 3D plot of generated path for extended case

The objective function values of the selected path,  $\mathcal{P}_{\text{selected}}$ , for the extended case is given in Table 6.17.

Table 6.17: Properties of the selected path,  $\mathcal{P}_{\text{selected}}$ ,  $M = 4$ 

Objective	Value	Unit	
Path length	$O_1(\mathcal{P}_{\text{selected}})$	291.867	[m]
Safety margin	$O_2(\mathcal{P}_{\text{selected}})$	-7.1160	[-]
Sharpest turn in NE-plane	$O_3(\mathcal{P}_{\text{selected}})$	0.6377	[rad]
Largest depth deviation	$O_4(\mathcal{P}_{\text{selected}})$	-1.5439	[-]

Additionally, the waypoints of the selected path,  $\mathcal{P}_{\text{selected}}$  in NED-coordinates for the extended case is given in Table 6.18.

Table 6.18: Waypoints of  $\mathcal{P}_{\text{selected}}$  for the extended case

Waypoint index	North	East	Down
0	7036891	570127	1
1	7036920	570143	58
2	7036953	570158	56
3	7036963	570172	58
4	7036967	570188	57
5	7036990	570206	115
6	7036990	570226	115

Next, simulation results from the extended case simulated in the integrated system using auto-altitude mode is given. Figures 6.29 and 6.30 depicts the ROV Minerva 2 trajectory in NED-coordinates, analogous to plots given in section 6.2.1, also including the curved seafloor.

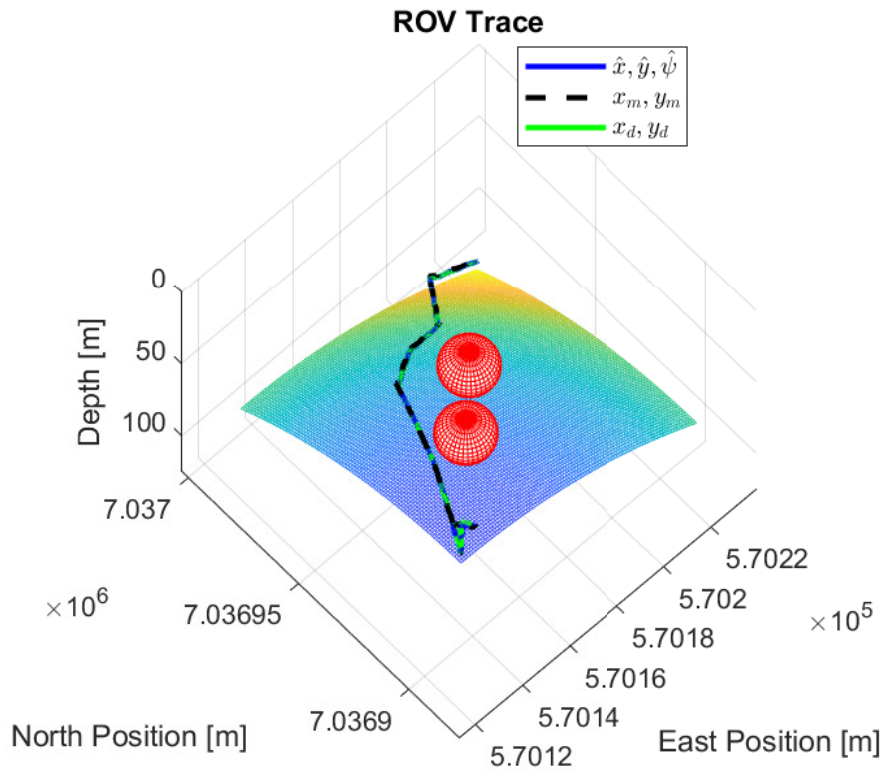


Figure 6.29: 3D ROV trajectory in the extended case

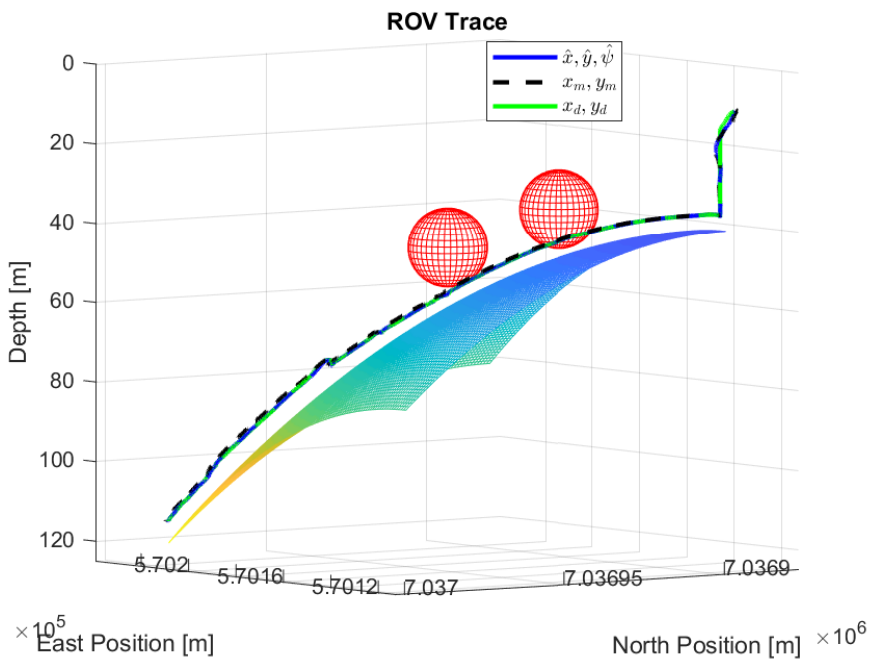


Figure 6.30: 3D ROV trajectory in the extended case

Similar as in previous sections, Figure 6.31 represents the time-evolution of the ROV altitude.

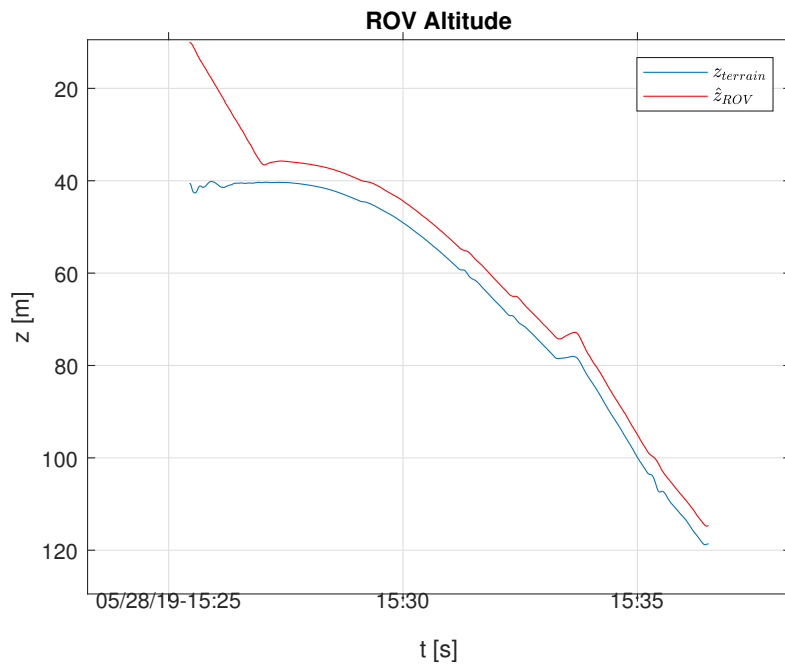


Figure 6.31: Time-evolution of ROV altitude in the extended case

Additionally, Figure 6.32 represents the time-evolution of the measurements, and corresponding estimates of the four acoustic beams from the DVL mounted on the ROV Minerva 2.

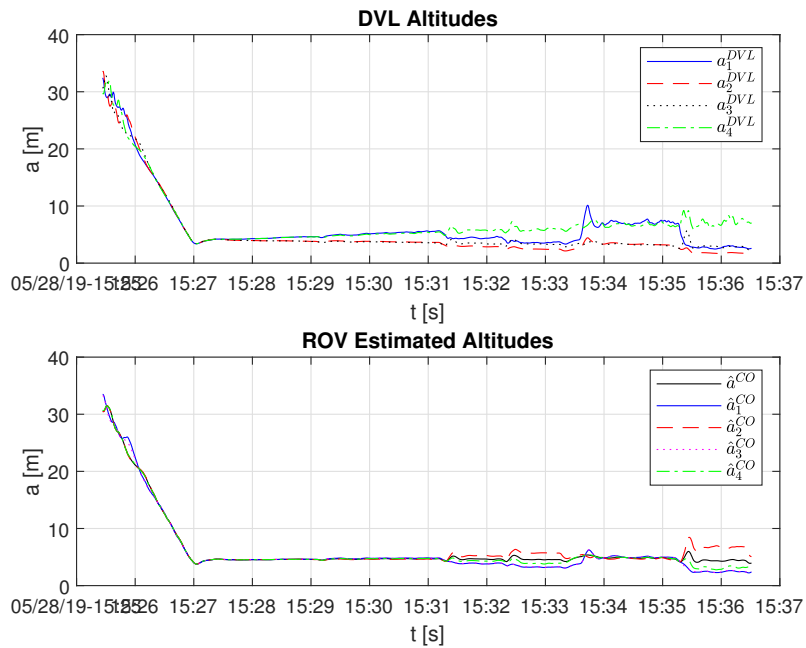


Figure 6.32: DVL measurements and estimates from extended case simulation

# Chapter 7

## Discussion

This Chapter consists of a discussion on the results provided in Chapter 6. Similarly, this Chapter is also divided into two overall categories corresponding to the different categories of results in Chapter 6:

1. Discussion on the results obtained from the `Matlab`-coded NSGA-II based path planner given in Section 6.1. This discussion is given in Section 7.1.
2. Discussion corresponding to the results obtained from `LabVIEW` simulations of the integrated system given in Section 6.2. This discussion is given in Section 7.2.

### 7.1 Optimization

Initially, the choice of NSGA-II parameters given in table 6.1 should be commented on. This set of parameters is applied to solving optimal path planning problems subject to both three and four objectives, of which graphical results are provided in Sections 6.1.1 and 6.1.2, respectively.

The choice of NSGA-II parameters presented in Table 6.1 governs from two factors, namely, recommendations from the literature [8], [37], [3], as well as trial and error. For instance, testing yields that reducing the SBX distribution index,  $\eta_c$ , from 20 to 5 rendered a population with a larger spread in the solution space, and thus an overall more satisfactory choice of selected path by the path selector provided in Algorithm 1. Smaller  $\eta_c$  provides NSGA-II with broader searching abilities, while larger  $\eta_c$  implies higher precision, as addressed in Section 4.1.4. The fact that  $\eta_c$  is constant in NSGA-II seems to make the algorithm for this application favour a smaller value for  $\eta_c$  since the path selector provides additional preferences when it comes to selecting a suitable path. As for the polynomial distribution index,  $\eta_m = 20$  seems to be a sufficient choice as this parameter determines the magnitude of the perturbation of the child solution relative to the parent solution, as addressed in Section 4.1.5. Moreover, the SBX probability,  $p_c$ , and the mutation probability,  $p_m$  are chosen according to literature, whereas, the population size,  $N$ , and the number of generations,  $G$ , are set reasonably large.

Note that there are not made any effort of finding the best possible values for  $N$  and  $G$ . In this context, the best possible values are high enough to generate satisfactory performance while minimizing computational requirements. Obtaining ideal parameter settings for  $N$  and  $G$  should be done if the optimal path planner is to be applied to scenarios where computational requirements are crucial, such as real-life applications or online enhancements. This thesis emphasizes the exploration of the functionalities of the path planner; however, work on ideal parameter settings could be a topic in case of further development of the NSGA-II based optimal path planner.

Nevertheless, establishing an ideal choice for  $G$  could be done by running the optimal path planner for different numbers of generations  $G$  and examine the performance of the results obtained with the different settings. For instance, the mean objective function values could be calculated for path planners with different  $G$  to determine at what  $G$  the objective function values converge to values that can be reviewed as optimal. The convergence is in coherence with decent optimization performance, as stated in Chapter 4. To eliminate uncertainties, the optimization on each  $G$  can be run multiple times, thus, the review-able objective function values would be the mean among the runs at the  $G$ -value in question.

Also, determining a sufficient value of  $N$  would affect the search power of the path planner, that is, the path planners ability to explore the entire search space. As stated in Chapter 4, diversity among the paths contained in the final population is a desired property in the optimization.

As stated in Section 4.1.6, the computational complexity is proportional to  $N^2$ . Also,  $G$  will affect the running time of the main loop in Algorithm 5. These facts indicates the possibility of increasing the efficiency of the NSGA-II based optimal path planner by ideal parameter settings.

Moreover, when examining the performance of the NSGA-II based optimal path planner in the results provided in Section 6.1 some general parameters are applied to all simulations, given in Table 6.2. This is motivated by achieving comparability between the different simulated scenarios.

Furthermore, another parameter that should be reviewed in terms of computational complexity is the number of interior waypoints,  $K$ . The parameter  $K$  is defined individually for the results obtained throughout Chapter 6. Increasing  $K$  results in the search space having a higher dimension, and thus, more computational power is required and vice verca. Note that the magnitude of  $K$  directly affects the dimension of  $N$ . Of course, the suitable choice of  $K$  depends on the mission at hand.

### 7.1.1 Path Planner with Three Objectives, $O_1$ , $O_2$ and $O_3$

Now, contemplation of the results in Section 6.1.1 obtained from the three-objective path planner, namely taking into account the objectives  $O_1$ ,  $O_2$  and  $O_3$ . In the subsequent post-processing of the NSGA-II solution to the optimal path planning problem defined in Section 3.7.7, a suitable path is chosen by the path selector. Visual inspection of the selected path,  $\mathcal{P}_{\text{selected}}$ , given in Figures 6.1, 6.2 and 6.3 shows that a relatively short path, with sufficient safety margin and absence of sharp turns in the NE-plane is obtained. These are desired properties in coherence with the applied objective functions, in the three-objective case  $O_1$ ,  $O_2$  and  $O_3$ , defined throughout Section 3.7.

Furthermore, examination of the performance of the optimization by visual inspection of Figures 6.4, 6.5, 6.6 and 6.7 concerning data on the final population of paths  $P$ , suggests that there are clear trends in all these Figures. The objective function values of the final population seem to somewhat converge to something that looks like a Pareto-optimal front for a constrained optimization problem. Convergence to a Pareto-optimal front is desirable, as addressed in Section 4.1.7. Additionally, the objective function values have a decent spread within the search space defined in the optimal path planning problem. The latter indicates diversity among the final population, which addressed as another desired property of optimization in Chapter 4. Thus, the performance of the NSGA-II based optimal path planner can be claimed to be good for this application.

Also, by reviewing the numeric data provided on  $\mathcal{P}_{\text{selected}}$  and the final population  $P$ , given in Tables 6.5 and 6.6, respectively, several comments can be made. By the path selector in Algorithm 1, the length of the selected path,  $O_1(\mathcal{P}_{\text{selected}})$ , is defined as the approximate mean of the shortest path length,  $O_{1,\text{min}} \in P$ , and the average path length,  $\bar{O}_1 \in P$ , in the final population  $P$ . The numerical values show that  $O_1(\mathcal{P}_{\text{selected}})$  is contained in the interval  $[O_{1,\text{min}}, \bar{O}_1]$ , indicating that the path selector could find a path satisfying this requirement while also satisfying the requirement on  $O_2$ . The path selector cannot simply obtain  $\mathcal{P}_{\text{selected}}$  as the path with the property closes to



the requirement on  $O_1$ , because the requirement on  $O_2$  is also taken into account. This causes a deviation between the length of the selected path  $O_1(\mathcal{P}_{\text{selected}})$  and the ideal length defined according to preference by the path selector.

Furthermore, as Table 6.4 indicates, the scaling factor  $\beta_{O_2}$  takes a value such that the maximum allowed safety margin to obstacles,  $O_{2,\text{required}}$ , for the path chosen by the path selector is 80% of the mean safety factor in the final population,  $\bar{O}_2 \in P$ . Note that the values of  $O_2$  are negative! Contemplation of the numerical values of these parameters shows that  $\mathcal{P}_{\text{selected}}$  takes on, by far, a better safety margin to obstacles than required by the path selector. And thus, the indication that the optimal path planner could be able to generate a path satisfying stricter requirements on  $O_2$ .

Finally, the last objective, that is, avoidance of sharp turns in the NE-plane  $O_3$ , is not taken into consideration by the path selector. Thus,  $O_3$  can be said to be the lesser important objective, intentionally, because this objective does not affect safety and energy consumption in the same extent as the two former objectives, namely  $O_1$  and  $O_2$ . On the other hand, the presence of  $O_3$  in the optimal path planner is still beneficial as this objective, in fact, helps the path planner get rid of paths with sharp turns throughout the optimization. Presence of sharp turns in the selected path can cause unnecessary wear and tears on the actuators of the ROV and indirectly cause a longer path length. However, for curiosity, it can be seen that  $O_3(\mathcal{P}_{\text{selected}})$  takes a lower value than the mean in the final population,  $\bar{O}_3 \in P$ .

Moreover, only one path is discarded by the path selector before a proposed path that suffices the criteria is obtained as  $\mathcal{P}_{\text{selected}}$ . Summing up, taking the above comments into account, it can be said that the three-objective optimal path planning problem for autonomous docking missions is easily solved with decent performance by the NSGA-II based path planner and the subsequent path selector.

### 7.1.2 Path Planner with Four Objectives, $O_1$ , $O_2$ , $O_3$ and $O_4$

Next, the results from the four objective NSGA-II based path planner are examined. In this scenario, the optimal path planning problem for autonomous docking missions defined in Section 3.7.7 includes all four objectives, namely,  $O_1$ ,  $O_2$ ,  $O_3$  and  $O_4$ . Post-processing results are provided in Section 6.1.2, where Figures 6.8, 6.9 and 6.10 depict a graphical representation of the selected path. As in the three-objective case, visual inspection yields that a relatively short path with sufficient safety margin and absence of sharp turns in the NE-plane is obtained. Additionally, it can be observed that in this case the interior waypoints contained in the selected path are located at a greater depth, underneath the obstacles in the environment. The deeper location of the interior waypoint is caused by including  $O_4$  to the optimal path planning problem and can be described as beneficial as the likelihood of achieving reliable DVL measurements is far better at higher depth. Of course, presupposing that the sea depth in the environment is known is necessary. Overall, the selected path has the desired properties in coherence with the applied objective functions, in the four-objective case  $O_1$ ,  $O_2$ ,  $O_3$  and  $O_4$ , defined throughout Section 3.7.

Moreover, graphical post-processing results on the final population of paths  $P$  in the four-objective case are provided in Figures 6.11, 6.12, 6.13 and 6.14. For simplicity, two-dimensional plots are not included since the increment in the number of objectives introduces a significant increase in the number of plots, and thus, the visual inspection becomes somewhat more complicated. However, by contemplation of the provided Figures trends can still be observed in all four plots, even though, these trends may not be quite as distinct as in the three-objective scenario. There is still convergence to something that looks like a Pareto-front, still with a decent spread in the search space. These findings are in coherence with desired properties indicating good optimization performance, addressed in Chapter 4.

Furthermore, the numerical data in tables 6.9 and 6.10 contains interesting information about the four-objective path planner performance, concerning  $\mathcal{P}_{\text{selected}}$  and the final population of paths,

respectively. Indications show that the path length of the selected path,  $O_1(\mathcal{P}_{\text{selected}})$  is contained in the interval  $[O_{1,\min}, \bar{O}_1] \in P$ , similarly as discussed for the three-objective case in Section 7.1.1.

Also, the requirements on  $O_2$  and  $O_4$  according to the path selector in Algorithm 1 are satisfied, namely,  $O_2(\mathcal{P}_{\text{selected}}) < O_{2,\text{required}}$  and  $O_4(\mathcal{P}_{\text{selected}}) < O_{4,\text{required}}$ . The former indicates sufficient safety margin to obstacles while the latter indicates sufficiently small deviation between the desired depth and the depth of the interior waypoints, in coherence with the objective functions defined throughout Section 3.7.

For curiosity, it can be mentioned that even though it is not a requirement defined in the path selector,  $O_2(\mathcal{P}_{\text{selected}})$  is slightly smaller than  $\bar{O}_3 \in P$ . The benefits of  $O_3$  are discussed in Section 7.1.1.

Moreover, some comparisons could be drawn to the three-objective case to distinguish between these two scenarios. The number of paths discarded by the path selector is 41 in the four-objective scenario, versus only one in the three-objective scenario. In the four-objective case,  $O_4$  is taken into account in the path selector such that yet another requirement concerning  $\mathcal{P}_{\text{selected}}$  has to be satisfied. With the path selector parameters  $\beta_{O_2} = 0.8$  for both scenarios and  $\beta_{O_4} = 1$ , for the four-objective scenario, it can be said that the requirement on  $O_4$  is the stricter one. Including  $O_4$  can be justified from a safety point of view, as it, by a decent likelihood, ensures reliability and availability of DVL measurements. Thus, autonomous docking missions can be carried out with proper state estimation. In the absence of proper state estimation, the operation would probably be too unsafe to perform. Nevertheless, the introduction of the additional objective,  $O_4$ , impacts the properties of both the final population of paths  $P$  from the optimization as well as the selected path,  $\mathcal{P}_{\text{selected}}$ . Numerical data on the two solutions of the two optimal path planning problems, with three and four objectives, respectively, is given in Table 7.1 to illustrate differences between these two scenarios.

Table 7.1: Comparison path planners in sections 6.1.1 and 6.1.2

Property	Value, M=3	Value, M=4	Comments
$O_1(\mathcal{P}_{\text{selected}})$ [m]	162.2595	186.6444	
$O_2(\mathcal{P}_{\text{selected}})$ [-]	-11.1240	-5.0394	
$O_3(\mathcal{P}_{\text{selected}})$ [rad]	0.3011	0.1730	
$O_4(\mathcal{P}_{\text{selected}})$ [-]		-2.3090	
$O_{1,\min} \in P$ [m]	158.5337	164.9809	Mean path length increases with introduction of $O_4$ .
$\bar{O}_1 \in P$ [m]	165.8948	187.1052	
$\bar{O}_2 \in P$ [-]	-10.7911	-5.3019	Mean safety margin decreases with introduction of $O_4$ .
$O_{2,\text{required}}$ [-]	-8.6329	-4.2415	
$\bar{O}_3 \in P$ [rad]	0.3811	0.2219	
$\bar{O}_4 \in P$ [-]		-2.1799	
$O_{4,\text{required}}$ [-]		-2.1799	

As Table 7.1 indicates, the mean path length increases with the introduction of the objective  $O_4$ . Naturally, this can be explained by comparison of Figure 6.1 and Figure 6.8, as the optimization makes the path approach the bottom to fulfill  $O_4$  when it is included. Furthermore, it can be seen that  $O_1(\mathcal{P}_{\text{selected}})$  is shifted more towards  $\bar{O}_1 \in P$  than  $O_{1,\min} \in P$  in the four-objective scenario versus the three-objective scenario, because the path selector is not able to find paths with length closer to the ideal length fulfilling the other requirements. The relation between the ideal length defined in the path selector and the actual length of  $\mathcal{P}_{\text{selected}}$  discussed in Section 7.1.1, is still valid in this comparison.

Moreover, the mean safety margin does indeed also deteriorate, which can be explained by the fact that the optimization problem becomes harder as  $O_4$  is taken into account. However, critical values of  $O_2$  are in the magnitude around  $-1$  or higher, and NSGA-II is able to discard solutions

with such  $O_2$ -values throughout the optimization process. As a matter of fact, this concludes that  $\beta_{O_2} = 0.8$  dictates a sufficient requirement in the path selector. On the other hand, even though the safety margins are slightly worse in the four-objective case, as briefly mentioned already, the introduction of  $O_4$  poses as an increase in safety for practical purposes. In real ROV operations, one can be more certain that a worse safety margin on  $O_2$  is sufficient if  $O_4$  is introduced since by this application the likelihood of achieving a well-behaved autonomous vehicle during an autonomous docking mission through reliable state estimation is higher because of reliable DVL measurements, as mentioned.

Finally, a general comment on the path selector, which does not search through the entire final population but instead assigns the first and best candidate to  $\mathcal{P}_{\text{selected}}$ . Once a candidate path that satisfies the selection criteria is found, the path selection is aborted and the candidate in question becomes  $\mathcal{P}_{\text{selected}}$ . The motivation behind this is to save computational time. However, there can exist paths within the final population that are better than  $\mathcal{P}_{\text{selected}}$ .

### 7.1.3 In General on the NSGA-II Based Optimal Path Planner

At this point, the discussion explores the research question on which this thesis is built, considering how a NSGA-II based optimal path planner could enable persistent autonomy in autonomous docking missions. The optimization performance is thoroughly addressed in the above Sections. However, as indicated in the literature review in Section 1.5.3, other EAs could achieve persistent autonomy as well. More specifically, as given in the literature review, [3] proposes a QPSO based optimal path planner for AUVs. If the QPSO based optimal path planner is redefined to take on the optimal path planning problem for ROVs, it could be a satisfactory contender, if not an even better path planner, to the NSGA-II based optimal path planner developed in this thesis.

## 7.2 Integrated System

Next, the performance of the integrated system described in Section 5.6 is examined in terms of the results provided in Section 6.2. Note that the results presented in Sections 6.2.1 and 6.2.2 origins from a simplified simulation environment with a flat seafloor.

### 7.2.1 Path Generation in the Integrated System

First, some comments on the path planner integrated into the control system are made. As mentioned, the scenario and the generated path upon which the autonomous docking mission in both the auto-depth mode and the auto-altitude mode is based on are the same. An overview of the path planning problem for autonomous docking and the corresponding selected path can be obtained by contemplating Figures 6.15-6.17 and Table 6.11. In addition to the plots, numerical values of the waypoints are given in Table 6.13. Note that the sea depth relative to the obstacle positions renders an environment such that the ROV Minerva 2 cannot transit underneath the obstacles as before. Visual inspection of the plots yields that a relatively short and collision-free path,  $\mathcal{P}_{\text{selected}}$  is generated as a member of the final population in the path planner and chosen by the path selector. The path has properties like an absence of sharp turns in the NE-plane and small deviations from the desired depth,  $z_{\text{desired}}$  in D-coordinates by inspection of Table 6.13. The sea depth in the simulated environment is 40 meters, and the desired depth is 30 meters, such that it is desirable that Minerva 2 transits 10 meters above the seafloor as taken into account by  $O_4$ . By inspection of the D-coordinates of the interior waypoints in Table 6.13, it can be seen that all D-coordinates are contained in the interval  $[z_{\text{desired}} - z_{\text{limit}}, z_{\text{desired}} + z_{\text{limit}}]$ . Throughout the optimization, paths that do not fulfill this will be penalized worse due to  $O_4$ . The problem parameters for scenarios simulated in the integrated system are different than before, such that the

path length  $O_1(\mathcal{P}_{\text{selected}})$  is not comparable to previous path lengths, whereas, the other objective function values of  $\mathcal{P}_{\text{selected}}$  are comparable and, in fact, very similar to those obtained in the four objective path planner as a stand-alone application in Section 6.1.2.

Furthermore, the same path is applied in both auto-depth mode and auto-altitude mode. This is done due to comparability reasons, in addition to a previously stated assumption supporting the application of all four objectives, namely,  $O_1$ ,  $O_2$ ,  $O_3$  and  $O_4$ . This assumption, see Section 5.6.2, states that the objective function  $O_4$  should be included also when generating paths for the auto-altitude mode, due to an assumed small deviation in obtained D-coordinates in the generated path and D-coordinate followed by the ROV in auto-altitude mode. Thus, rendering a safe and collision-free trajectory.

## 7.2.2 Comparison of Auto-Depth Mode and Auto-Altitude Mode

Moreover, the overall performance of Minerva 2 during docking operation in both auto-depth mode and auto-altitude mode is examined in order to distinguish between the general ROV behaviour obtained by these two modes.

Comparison of the three-dimensional Minerva 2 trajectory depicted in Figures 6.18 and 6.23 represents the auto-depth mode and the auto-altitude mode behavior, respectively, yields that the vehicle is able to perform collision-free trajectory in both modes. Visual inspection yields that estimated, measured and desired positions coincide quite well in both cases. However, one difference between the two modes is that in auto-altitude mode Minerva 2 dives down to the desired altitude before starting to move in the NE-plan, whereas, in auto-depth mode, Minerva 2 simply follows the path as it is. This is expected behaviour in coherence with the modes defined in Section 5.6, indicating that the travel time in auto-altitude mode is longer than in auto-depth mode assuming that Minerva 2 is deployed from the surface.

Furthermore, Figures 6.19 and 6.24 reveals information about Minerva 2's orientation in the NE-plane, i.e. heading, in the auto-depth mode and the auto-altitude mode, respectively. Figure 6.24 is somewhat distorted around the initial position, which corresponds to the initial diving down to the desired altitude in the auto-altitude mode trajectory. Visual inspection yields similar performance in both modes, however, the heading appears as slightly more oscillatory in auto-altitude mode. This behaviour is likely caused by the integrated use of the altitude controller and might be solved by tuning of controller gains, which is out of scope in this thesis.

Contemplation of the time evolution of the NED-coordinates throughout simulations in auto-depth mode and auto-altitude mode, given in Figures 6.20 and 6.25, respectively, gives a more detailed review of the relationship between measured, estimated and desired positions. Comparison supports the previous statement that the behaviour is slightly more oscillatory in auto-altitude mode compared to in auto-depth mode. However, the measured, estimated and desired positions still seems to coincide quite well. This observation indicates satisfactory performance of the control system described throughout Chapter 3. The measurements, represented by the blue lines in the Figures indicate that the measurements are slightly off, as expected in underwater navigation. Nevertheless, the observer seems to be able to produce sufficient state estimates. The discontinues in the North and East coordinate time evolution occurring in transitions between waypoints is expected due to the nature of constant jerk guidance, which finds the fastest path between two waypoints before redirecting the heading of the vehicle toward the next waypoint. By the nature of the constant jerk guidance, the heading should always be directed towards the next waypoint. This may cause a somewhat unsmooth trajectory, however, constant jerk guidance is suitable for over-actuated ROVs in environments requiring high precision. Constant jerk guidance benefits from its applicability to waypoint based paths consisting of straight-line path segments, as no path refinement, for instance, to splines is necessary, and thus, saving computational resources. On the other hand, speed reference guidance schemes could be implemented such that the ROV does not attain zero forward velocity during longer transits.

Moreover, Figures 6.21 and 6.26 depicts the time evolution of thruster rpm percentage of maximum rpm constrained by the control system at 1450 rpm during simulations in auto-depth mode and auto-altitude mode, respectively. The plots display similar results in both modes. However, the maximum amplitude has a slightly higher magnitude of about 30 % in auto-altitude mode versus just outside of 25 % in auto-depth mode. This difference can be considered insignificant due to the small deviation in rpm percentage in the two scenarios, and as there are loads of available rpm for unexpected events. Nevertheless, the difference can simply represent the already mentioned slightly more oscillatory behavior in auto-altitude mode and might be solved by tuning controller gains for altitude control.

Visual inspection of Minerva 2's altitude profile during simulations in auto-depth mode and auto-altitude mode, given in Figure 6.22 and 6.27, respectively, yields expected differences. The altitude profile obtained in auto-depth mode reveals that Minerva 2 transits at an altitude corresponding to the waypoint D-coordinates of  $\mathcal{P}_{\text{selected}}$  given in Table 6.13, whereas, after the initial diving, Minerva 2 takes on the desired altitude in the auto-altitude scenario. In the latter scenario, however, there seems to be some overshoot in the transition between diving and transit.

Finally, it should be noted that the two scenarios simulated and examined so far are two simplified scenarios designed to distinguish between auto-depth mode and auto-altitude mode. More sophisticated scenarios will typically be defined such that the assumption stating that the seafloor is flat does not hold. In addition, detailed knowledge about the sea depth is not necessarily known, however, the depth at the position from which Minerva 2 is deployed, that is, the initial position, can easily be measured by the mother-ship. The position of the docking station is known since the docking station at some point in time is installed there. Thus, the path planner could use an approximate input for sea depth and generate a path for the application of auto-altitude mode without the risk of bottom collision. Alternatively, the desired depth could be defined in coherence with known static obstacles in the environment.

### 7.2.3 Extended Case

Next, the performance of the extended case simulation results presented in Section 6.3 is examined. Note that this is a different scenario than simulated before, with parameters given in Table 6.16 acting as inputs in the optimal path planning problem defined in Section 3.7.7. Contemplation of Figure 6.28 and Table 6.18 illustrating the generated path,  $\mathcal{P}_{\text{selected}}$  rules out the possibility of performing autonomous docking utilizing auto-depth mode, due to the fact that  $\mathcal{P}_{\text{selected}}$  intersects the seafloor. The intersections are caused by the fact that the seafloor is curved, and such seafloors cannot directly be defined in the path planner as it is. Simply, the intersections mean that autonomous docking cannot be performed in auto-depth mode while avoiding bottom collision. Thus, the simulation is carried out in auto-altitude mode with at the desired altitude of four meters as indicated by Table 6.16. Furthermore, Table 6.17 displays the objective function values of  $\mathcal{P}_{\text{selected}}$  for the extended case. It is indicated that  $\mathcal{P}_{\text{selected}}$  has a sufficient safety margin to obstacles,  $O_2(\mathcal{P}_{\text{selected}})$ . Also,  $O_4$ -parameters are defined according to obstacle positions, solely to provide an obstacle collision-free path in auto-altitude mode in coherence with the previously stated assumption that the operation depth difference between auto-depth mode and auto-altitude mode is not too large. Moreover, by visual inspection,  $\mathcal{P}_{\text{selected}}$  has properties like absence of sharp turns and is relatively short.

Both the plots representing the trajectory of Minerva 2 throughout the simulation, namely Figure 6.29 and 6.30, and Figure 6.31 representing the time evolution of the ROV altitude, indicates that the autonomous docking is successfully carried out at a constant desired altitude in accordance with Table 6.16. However, the behaviour seems to be slightly oscillatory with some overshoot tendencies in waypoint transitions. This is mentioned before as a problem that can be solved by tuning the control system. As this thesis emphasizes path planning, this issue is out of scope and it is assumed that the performance can be enhanced by tuning of the altitude control scheme.

Additionally, the time evolution of measured and estimated DVL altitudes is given in Figure 6.32. Also, as indicated by Figure 6.28 and Table 6.15, the seafloor in this scenario gets steeper the further away from the initial position Minerva 2 transits. Thus, visual inspection of Figure 6.32 yields that acquiring reliable DVL altitudes becomes more difficult as the seafloor becomes steeper, as expected based on Section 3.6. However, the estimated, measured and desired positions plotted in Figures 6.29 and 6.30 still coincides well, indicating satisfactory state estimation, and does not seem to suffer significantly from this slight deterioration of DVL altitude availability. On the other hand, performing autonomous docking in an environment that is as steep as in this case can be considered a somewhat extreme case. Thus, this scenario demonstrates that the capabilities of the integrated system performing autonomous docking missions in auto altitude mode are satisfactory.

Summing up, it seems like the auto-depth mode outperforms auto-altitude in terms of less oscillatory behaviour and less overshoot, however, it is assumed that this difference in behaviour can be neutralized by tuning of controller gains. Moreover, assuming that the performance can be improved by tuning of controller gains, the auto-altitude is believed to be superior over the auto-depth mode for more sophisticated scenarios where the seafloor cannot be assumed to be flat and/or detailed knowledge about the sea depth is lacking. The benefits of online altitude control support this preference, as safety increases in this mode when the trajectory gets difficult. The increased likelihood of reliable DVL measurements in auto-altitude mode is essential for satisfactory state estimation and elimination of the risk of bottom collision, and thus, safe autonomous docking. The use of online altitude control renders the auto-altitude mode a semi-online path planning scheme, which can be further extended in future studies for making the NSGA-II path planner fully online with integrated obstacle avoidance and online re-planning.

# Chapter 8

## Conclusions

This Chapter presents some concluding remarks based on the discussions given throughout Chapter 7, corresponding to the research topic and objectives stated in Section 1.3.

### 8.1 Conclusions

This master's thesis explores the research hypothesis stating that the ROV control system will benefit from a high-performance path planner based on the non-dominated sorting genetic algorithm (NSGA-II). The contribution of this master thesis is a NSGA-II based optimal path planner allowing the ROV Minerva 2 to take on autonomous docking missions, through the implementation of the path planner into the ROV control system, forming an integrated system. The integrated system enables Minerva 2 to perform an optimal trajectory for autonomous docking by following a path consisting of waypoints.

The thesis emphasizes path planning, an essential component in the high-level autonomy module in the ROV control system, which is essential for achieving persistent autonomy. The work is motivated by the desire of increasing the level of autonomy in ROV operations. Increasing the level of autonomy is believed to streamline ROV operations by saving cost, increasing accuracy and efficiency, while maintaining safety. A higher level of autonomy will make ROVs less dependent on human intervention, and thus, saving costs. The advanced control structure, including an optimal path planner, will increase accuracy and efficiency. Lastly, maintaining safety is obtained by satisfactory performance of the integrated control system, and indirectly, by less human intervention.

The optimal path planning problem for autonomous docking missions proposed in this thesis is defined as a multi-objective optimization problem, assigning optimal paths four desirable properties. The four objectives corresponding to these properties are the path length, safety margin to obstacles, the absence of sharp turns in planar motion, and sufficient depth, ensuring reliable altitude measurements by the Doppler velocity log.

The optimal path planner is developed by applying NSGA-II to the optimal path planning problem. The performance of the NSGA-II based optimal path planner is examined through simulations concerning the path planner as a stand-alone application. Through optimization, the path planner generates a population of Pareto-optimal paths, that is, paths that all are optimal and equally good in terms of objective function evaluation. One of these paths is chosen for ROV autonomous docking trajectory by the path selector. The path selector is based on additional user-defined preferences on the path-objectives.

The integrated system consists of the path planner implemented in the control system of Minerva 2. Hardware-in-the-loop (HIL) simulations are carried out to examine the performance of the integrated system in different autonomous docking scenarios. The performance of the integrated system is reviewed in context with the control architecture of the entire control system. The integrated system is developed to enable autonomous docking trajectories in two different modes. The first mode, called auto-depth mode, tracks the path as it is generated by the path planner. The latter mode, auto-altitude mode, updates the depth-coordinates of the generated path online by following the seafloor at the desired altitude by altitude control. Both modes generate trajectories avoiding static obstacles, that also take on desired properties as defined by the objectives. The auto-altitude mode causes the ROV to dive down to the desired altitude initially in the docking mission, but also guarantees reliable DVL measurements, and thus, eliminates the risk of bottom collision. Hence, the auto-altitude mode is considered the preferred choice for autonomous docking missions. The performance in terms of the control system is satisfactory in both modes. However, there are slightly more oscillatory behaviour in the auto-altitude mode. This challenge is likely solvable by tuning of the control system. Coinciding desired and estimated states from simulations indicate that the performance of the observer is sufficient when DVL measurements are reliable.

Moreover, the NSGA-II path planner is efficient in terms of computational complexity. The control system uses constant jerk guidance for position reference, which is a waypoint based guidance scheme. Such waypoint based guidance schemes eliminate the need for further path refinement, and thus, save computational resources.

Lastly, it can be concluded that the NSGA-II based optimal path planner is a high-performance path planner for autonomous docking missions of Minerva 2.

## 8.2 Further work

During the work with this master thesis, several research topics for further work have come to mind. Initially, due to technical difficulties with the ROV Minerva 2 and its mother-ship, RV Gunnerus, field trials have unfortunately not been possible to carry out. Testing the NSGA-II based optimal path planner in real-life operations would have been interesting, as field experience on the performance of path planners is an incomplete research area.

Another topic for a future study could be developing a quantum-behaved particle swarm optimization (QPSO) based optimal path planner for autonomous ROV docking missions and compare the performance with the NSGA-II based optimal path planner.

Furthermore, augmentation of the NSGA-II based optimal path planner to include more online and re-planning functionalities is an interesting research topic. A reactive component able to detect and avoid obstacles will probably enhance the integrated system. Once the detected obstacle is avoided, the NSGA-II based optimal path planner could re-plan its path from its current position to the destination due to the sufficient computational complexity of the path planner.



# Bibliography

- [1] A. J. Sørensen and M. Ludvigsen, “Towards integrated autonomous underwater operations,” *IFAC PapersOnLine*, vol. 48, no. 2, pp. 107–118, 2015.
- [2] I. Rist-Christensen, “Autonomous robotic intervention using rovs,” 2016. [Online]. Available: <http://hdl.handle.net/11250/2440564>
- [3] Z. Zeng, K. Sammut, L. Lian, F. He, A. Lammas, and Y. Tang, “A comparison of optimization techniques for auv path planning in environments with ocean currents,” *Robotics and Autonomous Systems*, vol. 82, pp. 61–72, 2016.
- [4] T. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley & Sons, 2011.
- [5] “Kongsberg maritime,” 2018. [Online]. Available: <https://www.km.kongsberg.com/ks/web/nokbg0240.nsf/AllWeb/9DC12B00C48A0B63C1257F0900319BCF?OpenDocument>
- [6] S. M. Nornes, “Guidance and control of marine robotics for ocean mapping and monitoring,” Trondheim, 2018.
- [7] F. Dukan, “Rov motion control systems,” Trondheim, 2014.
- [8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, April 2002.
- [9] Z. Zeng, L. Lian, K. Sammut, F. He, Y. Tang, and A. Lammas, “A survey on path planning for persistent autonomy of autonomous underwater vehicles,” *Ocean Engineering*, vol. 110, no. PA, pp. 303–313, 2015.
- [10] M. Candeloro, “Tools and methods for autonomous operations on seabed and water column using underwater vehicles,” Trondheim, 2016.
- [11] StandardNorge, *Remotely operated vehicle (ROV) services*, 2016. [Online]. Available: <https://www.standard.no/en/webshop/ProductCatalog/ProductPresentation/?ProductID=797500>
- [12] “The applied underwater robotics laboratory,” 2018. [Online]. Available: <https://www.ntnu.edu/aur-lab>
- [13] N. R. Council, *Autonomous Vehicles in Support of Naval Operations*. Washington, DC: The National Academies Press, 2005. [Online]. Available: <https://www.nap.edu/catalog/11379/autonomous-vehicles-in-support-of-naval-operations>
- [14] P. Ridao, M. Carreras, D. Ribas, P. J. Sanz, and G. Oliver, “Intervention auvs: The next challenge,” *Annual Reviews in Control*, vol. 40, no. C, pp. 227–241, 2015.

- [15] A. Sundaresan, “Docking of underwater vehicle: Model, autopilot design, and guidance,” 2016. [Online]. Available: [uuid:3c87fdd0-8ed1-4e4b-a5ae-de400756d0fe](https://doi.org/10.1109/400756d0fe)
- [16] “Oceaneering,” 2018. [Online]. Available: <https://www.oceaneering.com/>
- [17] I. Dyrendahl, A. E. S. A. Sakshaug, and T. R. Svinåmo, “Iris - a concept for inspection and maintenance of deep water installations,” pp. 1–11.
- [18] D. d. A. Fernandes, “An output feedback motion control system for rovs : guidance, navigation, and control,” Trondheim, 2015.
- [19] M. Candeloro, A. J. Sørensen, S. Longhi, and F. Dukan, “Observers for dynamic positioning of rovs with experimental results,” *IFAC Proceedings Volumes*, vol. 45, no. 27, pp. 85–90, 2012.
- [20] A. M. Lekkas, M. Candeloro, and I. Schjøberg, “Outlier rejection in underwater acoustic position measurements based on prediction errors,” *IFAC PapersOnLine*, vol. 48, no. 2, pp. 82–87, 2015.
- [21] I. Schjøberg and I. B. Utne, “Towards autonomy in rovs operations,” 2015. [Online]. Available: <http://hdl.handle.net/11250/2475778>
- [22] F. Dukan, M. Ludvigsen, and A. J. Sorensen, “Dynamic positioning system for a small size rovs with experimental results.” *IEEE*, 2011, pp. 1–10.
- [23] L. S. Brusletto, “Computer vision based obstacle avoidance for a remotely operated vehicle,” 2016. [Online]. Available: <http://hdl.handle.net/11250/2418439>
- [24] L. Xue, “Computer vision based autonomous panel intervention for a remotely operation vessel,” 2018. [Online]. Available: <http://hdl.handle.net/11250/2566945>
- [25] E. N. Agdestein, “Visual estimation of motion for rovs - increasing accuracy for rovs navigation,” 2018. [Online]. Available: <http://hdl.handle.net/11250/2564522>
- [26] E. B. Holven, “Control system for rovs minerva 2,” 2018. [Online]. Available: <http://hdl.handle.net/11250/2564521>
- [27] A. Sans-Muntadas, “Path-planning, guidance and navigation tools for docking underactuated auvs,” 2018.
- [28] J. Canny and J. Reif, “New lower bound techniques for robot motion planning problems.” *IEEE*, 1987, pp. 49–60.
- [29] M. Wiig, K. Pettersen, and T. Krogstad, “A 3d reactive collision avoidance algorithm for nonholonomic vehicles,” 08 2018.
- [30] E. Besada-Portas, L. de La Torre, A. Moreno, and J. Risco-Martin, “On the performance comparison of multi-objective evolutionary uav path planners,” *Information Sciences*, vol. 238, pp. 111–125, 2013.
- [31] M. Ataei and A. Yousefi-Koma, “Three-dimensional optimal path planning for waypoint guidance of an autonomous underwater vehicle,” *Robotics and Autonomous Systems*, vol. 67, pp. 23–32, 2015.
- [32] A. J. Srensen, *Marine Control Systems. Propulsion and Motion Control of Ships and Offshore Structures*. Akademia Forlag, 2014.
- [33] S. M. Nornes, *Marine Cybernetics - Guidance and Navigation*. (Unpublished), 2018.
- [34] W. MathWorld, “Point-line distance–3-dimensional,” 2019. [Online]. Available: <http://mathworld.wolfram.com/Point-LineDistance3-Dimensional.html>

---

[//mathworld.wolfram.com/Point-LineDistance3-Dimensional.html](http://mathworld.wolfram.com/Point-LineDistance3-Dimensional.html)

- [35] OnlineMSchool, “Angle between two vectors,” 2019. [Online]. Available: <https://onlinemschool.com/math/library/vector/angl/>
- [36] D. E. Goldberg and K. Deb, “A comparative analysis of selection schemes used in genetic algorithms,” vol. 51, 12 1991.
- [37] R. Bhusan Agrawal, K. Deb, and R. Bhushan Agrawal, “Simulated binary crossover for continuous search space,” *Complex Systems*, vol. 9, 06 2000.
- [38] K. Deb and S. Agrawal, “A niched-penalty approach for constraint handling in genetic algorithms,” 01 1999.
- [39] K. Deb and M. Goyal, “A combined genetic adaptive search (geneas) for engineering design,” 1996.
- [40] N. Instruments, “Getting started with labview,” 2013. [Online]. Available: <http://www.ni.com/pdf/manuals/373427j.pdf>
- [41] —, “Compactrio systems,” 2019. [Online]. Available: <http://www.ni.com/en-no/shop/compactrio.html>
- [42] —, “Basic tcp/ip communication in labview,” 2019. [Online]. Available: <http://www.ni.com/white-paper/2710/en/>



# Appendices

## A Matlab-code

In the following appendices, the `Matlab`-code that is the proposed NSGA-II optimal path planner is presented. The `Matlab` directory is developed in coherence with the Algorithm 5 provided in Section 5.1. Additionally, Appendix A2 provides an example parameter setting for an optimal path planning problem concerning autonomous docking, corresponding to the problem defined in Section 3.7.7.

### A1 Main.m

```
1 %% Main NSGA-II optimal path planner
2 % *****
3 % This is the main program for the NSGA-II based optimal path planner
4 % designed to take on autonomous docking missions for the ROV Minerva 2.
5 % Followingly, a description of the subfiles and functions and a user-guide
6 % for definintion of the path planning problem:
7 %
8 % 1. parameters.m: Contains the optimal path planning parameters and the
9 %                 NSGA-II parameters. This file should be modified if other
10 %                 path planning scenarios are to be examined.
11 %
12 % 2. initialize_population.m: This function creates an initial population
13 %                            of size 2N at random based on the input from
14 %                            parameters.m. This function calls the function
15 %                            OFE_multiple_obstacles() for objective functon
16 %                            evaluation of all the members, i.e. paths, in
17 %                            the population. The function returns a matrix
18 %                            where each row corrsponds to one path.
19 %
20 % 3. NDS_CDA.m: This function handles the non-dominated sorting and
21 %              crowding distance assigment of the population in question. The
22 %              function appends rank and crowding distance to the matrix
23 %              containing information about the population
24 %
25 % 4. BTS.m: This function carries out binary tournament selection based on
26 %           an input population and returns the winners of the tournament.
27 %
28 % 5. make_new_pop.m: This function returns an offspring population based
29 %                   on a parent population utilizing the genetic operators
30 %                   simulated binary crossover and polynomial mutation. The
31 %                   function calls the function OFE_multiple_obstacles()
```

```

32 %           for objective function evaluation of all the members,
33 %           i.e. paths, in the population.
34 %
35 % 6. selection.m: This function carries out selection of paths for
36 %           reproduction, i.e. determines the parent population,
37 %           based on rank and crowding distance. If paths cannot be
38 %           distinguished from one another, the function
39 %           calls BTS() to complete the selection such that the size of
40 %           the parent population becomes N.
41 %
42 % 7. path_selector.m: This is a post-processing file selecting one path
43 %           among the obtained Pareto-optimal paths that cannot be
44 %           distinguished from one another. This file should be
45 %           modified in accordance with the preferred properties of
46 %           the selected path.
47 %
48 % 8. plot_tool.m. This file plots the results of the optimization, namely,
49 %           the objective functions values of the final population
50 %           compared among one another and the selected path in the
51 %           defined environment.
52 %
53 % For more background on the theory and method behind the development of
54 % this path planner, see:
55 % Hansen, Tobias Lars, "A NSGA-II Based Optimal Path Planner for Autonomous
56 % Docking of ROV Using Waypoint Guidance", 2019.
57 % *****
58 % Programmed by: Tobias Lars Hansen, 2019.
59 % *****
60 %%
61 clear all; close all; clc;
62
63 %% Initializing parameters
64 parameters;
65
66 %% Generating random population R_0
67 % R_0: size 2*N
68 % Evaluating objective functions for initial population, i.e. calls OFE()
69 R_0 = initialize_population(N, V, K, M, start_pos, final_pos,...
70                             min_range, max_range,...
71                             min_dist, obs_vec, z_terrain, pen_pos);
72
73 %% Non-dominated sorting and crowding distance assignment of R_0
74 R_0 = NDS_CDA(R_0, V, K, M);
75
76 %% Binary Tournament Selection - generating initial parent population P_0
77 % P_0: size N
78 P = BTS(R_0, N, V, K, M);
79
80 %% Generating initial offspring population Q_0
81 % Q_0: size N
82 % Objective function evaluation is carried out for offspring population,
83 % i.e. calls OFE()
84 Q = make_new_pop(P, V, K, M,...
85                 p_c, eta_c, p_m, eta_m,...
86                 min_range, max_range,...
87                 min_dist, obs_vec, start_pos, final_pos,...

```

---

```
88         z_terrain, pen_pos);
89
90 %% Main loop
91 for i = 1:G
92     %% Intermediate Solution R
93     % R_1: size 2*N
94     % R_1 is obtained by concatenation of P_0 and Q_0
95     R(1:N,:) = P;
96     R(N+1:2*N,:) = Q;
97
98     %% Non-dominated sorting and crowding distance assignment of R
99     R = NDS_CDA(R, V, K, M);
100
101     %% Selecting next parent generation, P
102     P = selection(R, N, V, K, M);
103
104     %% Generating next offspring population Q
105     Q = make_new_pop(P, V, K, M,...
106                    p_c, eta_c, p_m, eta_m,...
107                    min_range, max_range,...
108                    min_dist, obs_vec, start_pos, final_pos,...
109                    z_terrain, pen_pos);
110
111 end
112
113 %% Post processing
114 % Selecting path according to preferences
115 path_selector;
116 nbPath = a_index;
117 % plotting results
118 plot_tool;
```

**A2** parameters.m

```
1 %% Description:
2 % *****
3 % This file contains the parameters settings for NSGA-II and the parameters
4 % defining the optimal path planning problem for autonomous docking
5 % missions.
6 %
7 % This file should be modified if desirable to solve the path planning
8 % problem for different scenarios.
9 %
10 % For explanation of the parameters, please read comments throughout the
11 % code and/or see:
12 % Hansen, Tobias Lars, "A NSGA-II Based Optimal Path Planner for Autonomous
13 % Docking of ROV Using Waypoint Guidance", 2019.
14 % *****
15 % Programmed by: Tobias Lars Hansen, 2019.
16 % *****
17 %% Parameters
18 deg2rad = pi/180;
19 %% NSGA-II parameters:
20 N = 100; % Population size
21 G = 500; % Stopping criterion, number of generations
22 V = 3;
23 M = 4; % Number of objective functions
24
25 %% Parameters for Genetic Operations
26 % Simulated Binary Crossover (SBX) parameters
27 p_c = 0.9; % Crossover probability
28 eta_c = 5; % Distribution index, 20 suggested by literature
29 % Polynomial Mutation
30 p_m = 0.1; % Mutation probability
31 eta_m = 20; % Distribution index
32
33 %% Environment parameters
34 % Initial position
35 x0 = 1;
36 y0 = 1;
37 z0 = 1;
38 start_pos = [x0, y0, z0];
39
40 % Destination
41 x_final = 100;
42 y_final = 100;
43 z_final = 190;
44 final_pos = [x_final, y_final, z_final];
45
46 % Final heading
47 heading_final = -90*deg2rad; % 0 deg is towards north
48
49 % Length of last path segment
50 length_last_segment = 20;
51
52 % Penultimate position
53 x_pen = final_pos(1) + cos(heading_final)*(-length_last_segment);
54 y_pen = final_pos(2) - sin(heading_final)*(-length_last_segment);
```



```
55 z_pen = z_final;
56 pen_pos = [x_pen, y_pen, z_pen];
57
58 % Variable bounds
59 xxmin = [x0, x_final, x_pen];
60 xxmax = [x0, x_final, x_pen];
61 yymin = [y0, y_final, y_pen];
62 yymax = [y0, y_final, y_pen];
63
64 x_min = min(xxmin) - 1;
65 x_max = max(xxmax) + 1;
66
67 y_min = min(yymin) - 1;
68 y_max = max(yymax) + 1;
69
70 z_min = 30;
71 z_max = 70;
72
73 % Range vectors
74 max_range = [x_max, y_max, z_max];
75 min_range = [x_min, y_min, z_min];
76
77 % f(4)
78 z_terrain = z_max - 10;
79
80 K = 4; % Number of interior waypoints
81
82 % Obstacle position(s)
83 x_obs = 50;
84 y_obs = 50;
85 z_obs = 60;
86 % obs_vec = [x_obs, y_obs, z_obs];
87
88 x_obs_2 = 30;
89 y_obs_2 = 30;
90 z_obs_2 = 40;
91 obs_vec = [x_obs, y_obs, z_obs, x_obs_2, y_obs_2, z_obs_2];
92
93 % Assuming spherical obstacle, with the following radius
94 radius_obs = 10;
95
96 % Minimum accepted distance from obstacle in meters
97 min_dist = radius_obs + 5;
```

### A3 OFE\_multiple\_obstacles.m

```

1 function f = OFE_multiple_obstacles( vec,...
2     K, V, min_dist, obs_vec, z_terrain)
3 %% Description
4 % *****
5 % This function evaluates the calculates the objective functions for each
6 % path. The function takes one path, f, as input at the time.
7 %
8 % The first objective function, f(1), calculates the path length.
9 %
10 % The second objective function, f(2), calculates the lowest safety margin
11 % to obstacles applies penalty accordingly.
12 %
13 % The third objective function, f(3), calculates the sharpest turn in the
14 % xy-plane.
15 %
16 % The fourth objective function, f(4), calculates the largest deviation
17 % between the z-coordinate and desired depth in the path and applies
18 % penalty accordingly.
19 %
20 % *****
21 % Programmed by: Tobias Lars Hansen, 2019.
22 % *****
23 %% Objective function f(1): Length of path - low f(1) is good
24 % Count indicies:
25 xc = 1;
26 yc = 2;
27 zc = 3;
28
29 df_1 = zeros(1,K+2);
30 for i = 1:K+2
31     % Length of path segment between two adjacent waypoints
32     df_1(i) = sqrt( ( vec(xc+V) - vec(xc) )^2 + ...
33                   ( vec(yc+V) - vec(yc))^2 + ...
34                   ( vec(zc+V) - vec(zc))^2 );
35     % Updating counting indicies
36     xc = xc + V;
37     yc = yc + V;
38     zc = zc + V;
39 end
40 clear xc yc zc;
41 f(1) = sum(df_1);
42
43
44 %% - Objective function f(2): Margin of safety
45 % more negative f(2) is better
46 % Calculating the shortest distance between every line segment and the
47 % obstacle.
48
49 %Number of obstacles
50 nb_obs = length(obs_vec)/3;
51 obs_mat = zeros(nb_obs, 3);
52
53 % Count indicies:
54 xc = 1;

```

---

```

55 yc = 2;
56 zc = 3;
57
58 for i = 1:nb_obs
59     obs_mat(i,:) = [obs_vec(xc), obs_vec(yc), obs_vec(zc)];
60     % Updating counting idicies
61     xc = xc + V;
62     yc = yc + V;
63     zc = zc + V;
64 end
65
66 dist = zeros(nb_obs,K+1);
67
68 % for each obstacle
69 for j = 1:nb_obs
70     % Count idicies:
71     xc = 1;
72     yc = 2;
73     zc = 3;
74
75     for i = 1:K+1
76         % Adjacent waypoints on vetor form
77         wp1 = [vec(xc), vec(yc), vec(zc)];
78         wp2 = [vec(xc+V), vec(yc+V), vec(zc+V)];
79
80         % Reference: Algorithm used for calculating distance from point to line:
81         % http://mathworld.wolfram.com/Point-LineDistance3-Dimensional.html
82         num = norm(cross(obs_mat(j) - wp1, obs_mat(j) - wp2));
83         den = df_1(i);
84         dist(j,i) = num / den;
85
86         % Updating counting idicies
87         xc = xc + V;
88         yc = yc + V;
89         zc = zc + V;
90     end
91
92 end
93 clear xc yc zc;
94
95 dist_vec = [ dist(1,:), dist(2,:)];
96
97 % Calculating the objective function value.
98 d_min = min(dist_vec);
99
100 if (d_min < min_dist)
101     % If the path is situated less than a safety distance away from the
102     % obstale, f_2 will approach zero or even be positive
103     f(2) = exp(d_min-min_dist);
104 else
105     % If the path is situated more than a safety distance away from the
106     % obstacle, the f_2 will take a negative value.
107     f(2) = min_dist - d_min;
108 end
109
110 %% Objective function f(3) - penaltizing sharp turns in paths

```

```

111
112 xc = 1;
113 yc = 2;
114
115 theta = zeros(1,K);
116
117 for i = 1:K
118     wp1 = [vec(xc), vec(yc)];
119     wp2 = [vec(xc + V), vec(yc + V)];
120     wp3 = [vec(xc + 2*V), vec(yc + 2*V)];
121
122     dp_1 = wp2 - wp1;
123     dp_2 = wp3 - wp2;
124
125     theta(i) = acos( (dot(dp_1, dp_2))/(norm(dp_1)*norm(dp_2)) );
126
127     xc = xc + V;
128     yc = yc + V;
129 end
130
131
132 f_3_theta = max(abs(theta));
133
134 f(3) = f_3_theta;
135
136 %% Objective function f(4) - ensuring available DVL measurements
137 zc = V + V;
138 z_wp = zeros(1,K);
139
140 for i = 1:K
141     z_wp(i) = abs(vec(zc) - z_terrain) ;
142     zc = zc + V;
143 end
144
145 f_terr = zeros(1,K);
146 z_limit = 5;
147 for i = 1:length(z_wp)
148     z_dev = z_wp(i);
149     if z_dev > z_limit
150         f_terr(i) = exp((z_limit - z_dev));
151     else % z_dev < z_limit
152         f_terr(i) = z_dev - z_limit;
153     end
154 end
155 f_terr_pen = max(f_terr);
156
157 f(4) = f_terr_pen;
158 end %fcn

```

## A4 initialize\_population.m

```

1 function f = initialize_population(N, v_dim, k_interior, M,...
2                                     start_pos, final_pos,...
3                                     min_range, max_range,...
4                                     min_dist, obs_vec, z_terrain, pen_pos)
5 %% Description
6 % *****
7 % This function generates an initial population, f, of paths at random
8 % based on the input from the file parameters.m
9 %
10 % This function calls the function OFE_multiple_obstacles for evaluation of
11 % the objective function values of the generated paths.
12 %
13 % For explanation of inputs, see parameters.m / Main.m
14 % *****
15 % Programmed by: Tobias Lars Hansen, 2019.
16 % *****
17 %%
18 nbFixedPoints = 3;
19 V = v_dim*(k_interior + nbFixedPoints);
20
21 pop_data = zeros(2*N, V + M + 2);
22 for i = 1:2*N
23     %% Generate random population of size N, i.e. generate N random paths
24     % between start and destination
25
26     % Assigning starting point
27     for j = 1:v_dim
28         pop_data(i,j) = start_pos(j);
29     end
30     % Assigning internal points
31     count = v_dim + 1;
32     for j = 1:k_interior
33         for k = 1:v_dim
34             pop_data(i,count) = min_range(k) +...
35                 (max_range(k)-min_range(k))*rand(1);
36             count = count + 1;
37         end
38     end
39     clear count;
40     % Assigning penultimate point
41     for j = 1:v_dim
42         pop_data(i, V - 2*v_dim + j) = pen_pos(j);
43     end
44
45     % Assigning destination point
46     for j = 1:v_dim
47         %pop_data(i,V+V*K+j) = final_pos(j);
48         pop_data(i,V - v_dim + j) = final_pos(j);
49     end
50     %% Calculating the objective function value for each individual, i.e.
51     % evaluate the performance of each path
52     pop_data(i,V+1:V+M) = ...
53         OFE_multiple_obstacles(pop_data(i,:), k_interior, v_dim, ...
54                                 min_dist, obs_vec, z_terrain);

```

```
55 end
56 f = pop_data;
57 end %fcn
```

## A5 NDS\_CDA.m

```

1 function f = NDS_CDA(x, v_dim, k_interior, M)
2 %% Description
3 % *****
4 % This function takes in a population, x, as input and 1) sorts it in terms
5 % of non-domination, 2) calculates the crowding distance for each path in
6 % the population.
7 %
8 % For explanation of inputs, see parameters.m / Main.m
9 %
10 % * This code is based on pseudo-code from the article:
11 %   https://ieeexplore.ieee.org/document/996017
12 % *****
13 % Programmed by: Tobias Lars Hansen, 2019.
14 % *****
15 %%
16 % Simplifying notation for user readability
17 [N, ~] = size(x);
18 nbFixedPoints = 3;
19 V = v_dim*(k_interior + nbFixedPoints);
20
21 % Extracting objective function values
22 f_1 = x(:,V+1);
23 f_2 = x(:,V+2);
24 f_3 = x(:,V+3);
25 f_4 = x(:,V+4);
26
27 %% Explanations:
28 % *****
29 % * Domination count, n_p: The number of solutions that dominate the
30 % solution p. Here expressed by notation: sol(p).n
31 % * S_p: A set of solutions that the solutions p dominates. Here expressed
32 % by notation: sol(p).S
33 % *****
34 %% Fast non-dominated sorting
35 % Pre-allocation
36 sol(:).n = 0;
37 sol(:).sp = [];
38 rank = zeros(N,1);
39 f_i = 1;
40 F(f_i).f = [];
41 for p = 1:N
42     % Domination count, initialized n_p = 0:
43     sol(p).n = 0;
44     % Set Sp, initially an empty set:
45     sol(p).S = [ ];
46
47     for q = 1:N
48
49         % if p dominates q
50         if f_1(p) < f_1(q) && f_2(p) < f_2(q) && f_3(p) < f_3(q) && ...
51             f_4(p) < f_4(q) ...
52             || f_1(p) == f_1(q) && f_2(p) < f_2(q) &&...
53             f_3(p) < f_3(q) && f_4(p) < f_4(q)...
54             || f_1(p) < f_1(q) && f_2(p) == f_2(q) &&...

```

```

55         f_3(p) < f_3(q) && f_4(p) < f_4(q)...
56         || f_1(p) < f_1(q) && f_2(p) < f_2(q) &&...
57         f_3(p) == f_3(q) && f_4(p) < f_4(q)...
58         || f_1(p) < f_1(q) && f_2(p) < f_2(q) &&...
59         f_3(p) < f_3(q) && f_4(p) == f_4(q)
60
61         % Add q to the set of solutions dominated by p
62         sol(p).S = [sol(p).S, q];
63
64         % if q dominates p
65         elseif f_1(p) > f_1(q) && f_2(p) > f_2(q) &&...
66             f_3(p) > f_3(q) && f_4(p) > f_4(q)...
67             || f_1(p) == f_1(q) && f_2(p) > f_2(q) &&...
68             f_3(p) > f_3(q) && f_4(p) > f_4(q)...
69             || f_1(p) > f_1(q) && f_2(p) == f_2(q) &&...
70             f_3(p) > f_3(q) && f_4(p) > f_4(q)...
71             || f_1(p) > f_1(q) && f_2(p) > f_2(q) &&...
72             f_3(p) == f_3(q) && f_4(p) > f_4(q)...
73             || f_1(p) > f_1(q) && f_2(p) > f_2(q) &&...
74             f_3(p) > f_3(q) && f_4(p) == f_4(q)
75
76         % Increment the domination counter of p
77         sol(p).n = sol(p).n + 1;
78
79         end % if domination
80
81     end % for q
82
83     % Identifying the first non-dominated front
84     if sol(p).n == 0
85         rank(p,1) = 1;
86         F(f_i).f = [F(f_i).f, p];
87     end
88
89
90 end %for p
91
92 x(:, V + M + 1) = rank;
93
94 % Identifying subsequent non-dominated fronts
95 while ~isempty(F(f_i).f)
96
97     % Empty set list used to store members of the next front
98     Q = [ ];
99
100    % For each solution p with n_p = 0, visiting each member if its set S_p
101    % and reducing its domination count by one
102    for p = 1:length(F(f_i).f)
103
104        for q = 1:length( sol( F(f_i).f(p) ).S )
105
106            % Reducing domination count by one
107            sol( sol( F(f_i).f(p) ).S(q) ).n = ...
108                sol( sol( F(f_i).f(p) ).S(q) ).n -1;
109
110            % if n_p = 0 for any members, these members are put in the

```



```

111         % list Q as they belong to the next non-dominated front
112         if sol( sol( F(f_i).f(p) ).S(q) ).n == 0
113
114             x( sol(F(f_i).f(p)).S(q), V + M + 1) = f_i + 1;
115             Q = [Q sol(F(f_i).f(p)).S(q)];
116
117         end % end if n_p = 0
118     end
119 end
120 % Incrementing front counter
121 f_i = f_i + 1;
122 F(f_i).f = Q;
123 end
124 % Sorting the population based on ascending rank
125 x_r_s = sortrows(x,V + M + 1);
126
127 %% Crowding distance assignment
128 % Calculating the crowding distance for each solution in each non-dominated
129 % set, i.e in each front
130
131 % Making a temporary population data vector
132 x_temp = x_r_s;
133
134 row_i = 1;
135 for i = 1: ( length(F) - 1 )
136
137     % Initializing distance, i_dist = 0
138     i_dist = 0;
139
140     % Retrieving the length of each front
141     length_f = length(F(i).f);
142
143     % If the length of the front is higher than 2 intermediate solutions
144     % exists
145     if length_f > 2
146
147         % Sorting the population based on objective function values
148         [~, f_1_i] = sortrows(x_temp(row_i:(row_i+length_f - 1) , V + 1));
149         [~, f_2_i] = sortrows(x_temp(row_i:(row_i+length_f - 1) , V + 2));
150         [~, f_3_i] = sortrows(x_temp(row_i:(row_i+length_f - 1) , V + 3));
151         [~, f_4_i] = sortrows(x_temp(row_i:(row_i+length_f - 1) , V + 4));
152
153         % Assigning boundary solutions infinite distance
154         x_temp( f_1_i(1) + row_i - 1 , V + M + 2 ) = inf;
155         x_temp( f_1_i(end) + row_i - 1 , V + M + 2 ) = inf;
156
157         x_temp( f_2_i(1) + row_i - 1 , V + M + 3 ) = inf;
158         x_temp( f_2_i(end) + row_i - 1 , V + M + 3 ) = inf;
159
160         x_temp( f_3_i(1) + row_i - 1 , V + M + 4 ) = inf;
161         x_temp( f_3_i(end) + row_i - 1 , V + M + 4 ) = inf;
162
163         x_temp( f_4_i(1) + row_i - 1 , V + M + 5 ) = inf;
164         x_temp( f_4_i(end) + row_i - 1 , V + M + 5 ) = inf;
165
166         % Retrieving objective function extreme values for normalization

```

```

167     f_1_min = x_temp(f_1_i(1) + row_i - 1 , V + 1);
168     f_1_max = x_temp(f_1_i(end) + row_i - 1 , V + 1);
169
170     f_2_min = x_temp(f_2_i(1) + row_i - 1 , V + 2);
171     f_2_max = x_temp(f_2_i(end) + row_i - 1 , V + 2);
172
173     f_3_min = x_temp(f_3_i(1) + row_i - 1 , V + 3);
174     f_3_max = x_temp(f_3_i(end) + row_i - 1 , V + 3);
175
176     f_4_min = x_temp(f_4_i(1) + row_i - 1 , V + 4);
177     f_4_max = x_temp(f_4_i(end) + row_i - 1 , V + 4);
178
179     % Assigning intermediate solutions a distance
180     for j = 2:length_f - 1
181
182         % If more boundary solutions exists
183         if (f_1_min - f_1_max == 0) || (f_2_min - f_2_max == 0)...
184             || (f_3_min - f_3_max == 0)
185             x_temp (f_1_i(j)+row_i-1, V+M+2) = inf;
186             x_temp (f_2_i(j)+row_i-1, V+M+3) = inf;
187             x_temp (f_3_i(j)+row_i-1, V+M+4) = inf;
188         % Intermediate solutions - normalized
189         else
190             x_temp (f_1_i(j)+row_i-1, V+M+2) =...
191                 (x_temp(f_1_i(j+1) + row_i - 1, V+1)...
192                 - x_temp(f_1_i(j-1) + row_i - 1, V+1))...
193                 /(f_1_max - f_1_min);
194
195             x_temp (f_2_i(j)+row_i-1, V+M+3) =...
196                 (x_temp(f_2_i(j+1) + row_i - 1, V+2)...
197                 - x_temp(f_2_i(j-1) + row_i - 1, V+2) )...
198                 /(f_2_max - f_2_min);
199
200             x_temp (f_3_i(j)+row_i-1, V+M+4) =...
201                 (x_temp(f_3_i(j+1) + row_i - 1, V+3)...
202                 - x_temp(f_3_i(j-1) + row_i - 1, V+3) )...
203                 /(f_3_max - f_3_min);
204
205             x_temp (f_4_i(j)+row_i-1, V+M+5) =...
206                 (x_temp(f_4_i(j+1) + row_i - 1, V+4)...
207                 - x_temp(f_4_i(j-1) + row_i - 1, V+4) )...
208                 /(f_4_max - f_4_min);
209         end
210     end
211
212     % Else only boundary solutions exists
213     else
214         % Assigning boundary solutions infinite distance
215         x_temp(row_i:(row_i+length_f-1), V+M+2:V+M+5) = inf;
216     end
217     row_i = row_i + length_f;
218 end
219 % Calculating the overall crowding distance, i.e. the sum of the individual
220 % crowding distances.
221 crowding_dist = zeros(N,1);
222 for i = 1:N

```

```
223     crowding_dist(i) = x_temp(i,V+M+2) + x_temp(i,V+M+3) +...
224         x_temp(i,V+M+4) + x_temp(i,V+M+5);
225 end
226 x_r_s(:, V + M + 2) = crowding_dist;
227
228 f = x_r_s;
229
230 end
```

**A6** BTS.m

```
1 function f = BTS (x, P, v_dim, k_interior, M)
2 %% Description
3 % *****
4 % This function returns the winners of binary tournament selection based on
5 % an input population.
6 %
7 % * P = size of the mating pool
8 % * x = input-matrix containing population data
9 % For additional explanation of inputs, see parameters.m / Main.m
10 % *****
11 % Programmed by: Tobias Lars Hansen, 2019.
12 % *****
13 %%
14 % Simplifying notation
15 nbFixedPoints = 3;
16 V = v_dim*(k_interior + nbFixedPoints);
17
18 % Retrieving the rank and crowding distance from the population data:
19 rank_index = V+M+1;
20 cdist_index = V+M+2;
21
22 %% Binary tournament selection
23 % Two random solutions are compared in order to figure out which one is
24 % best fit to be a parent.
25 % The comparison is based on: 1) lower rank is preferred, 2) if the ranks
26 % are equal, the solution with the higher crowding distance is preferred
27
28 % Generating random unique indicies for which individuals to compare
29 rand_nb = randperm(2*P)';
30 compare_i(:,1) = rand_nb(1:P);
31 compare_i(:,2) = rand_nb(P+1:end);
32
33 % Pre allocation
34 mating_pool = zeros(P,V+M+2);
35
36 for p = 1:P
37
38     % Contender 1
39     contender_1 = compare_i(p,1);
40     rank_1 = x(contender_1, rank_index);
41
42     % Contender 2
43     contender_2 = compare_i(p,2);
44     rank_2 = x(contender_2, rank_index);
45
46     % If contenders have different rank
47     if rank_1 ~= rank_2
48
49         if rank_1 < rank_2
50             winner = contender_1;
51         else %if rank_1 > rank_2
52             winner = contender_2;
53         end
54
55     end
56 end
```

```
55     mating_pool(p,:) = x(winner,:);
56
57     % if contenders have the same rank
58     else
59         % Retrieving the crowding distance for the contenders
60         cdist_1 = x(contender_1, cdist_index);
61         cdist_2 = x(contender_2, cdist_index);
62
63         if cdist_1 > cdist_2
64             winner = contender_1;
65         else % cdist_1 < cdist_2
66             winner = contender_2;
67         end
68         mating_pool(p,:) = x(winner,:);
69     end
70 end
71 f = mating_pool;
72 end
```

## A7 make\_new\_pop.m

```

1 function f = make_new_pop(x, v_dim, k_interior, M,...
2                               p_c, eta_c, p_m, eta_m,...
3                               min_range, max_range,...
4                               min_dist, obs_vec, start_pos, final_pos,...
5                               z_terrain, pen_pos)
6 %% Description
7 % *****
8 % This function returns generated offspring population, f, generated by
9 % applying the genetic operators simulated binary crossover and polynomial
10 % mutation. The input is the parent population.
11 %
12 % This function calls the function OFE_multiple_obstacles for evaluation of
13 % the objective function values of the generated paths.
14 %
15 % For additional explanation of inputs, see parameters.m / Main.m
16 % *****
17 % Programmed by: Tobias Lars Hansen, 2019.
18 % *****
19 %%
20 [N, ~] = size(x);
21 nbFixedPoints = 3;
22 V = v_dim*(k_interior + nbFixedPoints);
23 p_i = randperm(N);
24 nb = k_interior*nbFixedPoints;
25 vmax = repmat(max_range, 1, nb);
26 vmin = repmat(min_range, 1, nb);
27 child_pop = zeros(N,V+M+2);
28 for p = 1:N
29     u = zeros(1, V);
30     % SBX
31     if rand(1) < p_c && p < N-1
32         parent_1 = x(p_i(p), 1:V);
33         parent_2 = x(p_i(p+1), 1:V);
34
35         beta = zeros(1, V);
36         alpha = zeros(1, V);
37         beta_q = zeros(1,V);
38
39         % Variable by variable
40         for i = 1:V
41             if parent_1(i) < parent_2(i)
42                 beta(i) = 1 + (2/(parent_2(i)-parent_1(i)))...
43                     *min((parent_1(i)-vmin(i)), (vmax(i)-parent_2(i)));
44             else % parent_1 > parent_2
45                 beta(i) = 1 + (2/(parent_1(i)-parent_2(i)))...
46                     *min((parent_2(i)-vmin(i)), (vmax(i)-parent_1(i)));
47             end
48
49             alpha(i) = 2 - beta(i)^(eta_c + 1);
50             u(i) = rand(1);
51
52             if u(i) <= 1/alpha(i)
53                 beta_q(i) = ( u(i)*alpha(i) )^( 1/(eta_c + 1) );
54             else

```

---

```

55         beta_q(i) = (1/(2- u(i)*alpha(i)))^( 1/(eta_c + 1) );
56     end
57 end
58
59     child_1 =0.5*((parent_1 + parent_2)-beta_q.*(parent_2 - parent_1));
60     child_2 =0.5*((parent_1 + parent_2)+beta_q.*(parent_2 - parent_1));
61
62     child_pop(p, 1:V) = child_1;
63     child_pop(p+1, 1:V) = child_2;
64
65     % If SBX occurs, p needs to be changed
66     p = p + 1;
67
68     % Polynomial mutation
69     else
70
71         parent_mut = x(p_i(p), 1:V);
72         delta = zeros(1, V);
73         delta_q = zeros(1, V);
74         child_mut = zeros(1,V);
75
76         for i = 1:V
77             u(i) = rand(1);
78             delta(i) = min( (parent_mut(i) - vmin(i)),....
79                             (vmax(i) - parent_mut(i)) )...
80                             /(vmax(i) - vmin(i));
81             if u(i) < 0.5
82                 delta_q(i) = (2*u(i)+(1-2*u(i))*(1-delta(i)))...
83                             ^((eta_m + 1))^(1/(eta_m + 1)) - 1;
84             else
85                 delta_q(i) = 1 - (2*(1-u(i)) + ...
86                                 2*(u(i) - 0.5)*(1 - delta(i)))...
87                                 ^((eta_m + 1))^(1/(eta_m + 1));
88             end
89             child_mut(i) = parent_mut(i) + delta_q(i)*(vmax(i) - vmin(i));
90         end
91         child_pop(p, 1:V) = child_mut;
92
93         % Manipulating the fixed point, i.e. start and destinator
94         child_pop(p, 1:3) = start_pos;
95         child_pop(p, V-5:V-3) = pen_pos;
96         child_pop(p, V-2:V) = final_pos;
97     end
98 end %for genetic operations
99 %% Evaluating objective functions for offspring population
100 for p = 1:N
101     child_pop(p, V+1: V + M) = OFE_multiple_obstacles(child_pop(p,:),...
102                                                         k_interior, v_dim,...
103                                                         min_dist, obs_vec, z_terrain);
104 end
105 f = child_pop;
106 end %fcn

```

## A8 selection.m

```

1 function f = selection (intermediate_pop, N, V, K, M)
2 %% Description
3 % *****
4 % This function performs selection of the next parent population based on
5 % rank and crowding distance.
6 %
7 % This function calls the function BTS if paths cannot be distinguished
8 % from each other in terms of rank and crowding distance.
9 %
10 % Input: Intermediate population - size 2*N
11 % Output: Parent population for next iteration - size N
12 % *****
13 % Programmed by: Tobias Lars Hansen, 2019.
14 % *****
15 %%
16 rank_f = intermediate_pop(100, V*(K+2)+M+1);
17
18 for i = 1:2*N
19     rank_i = intermediate_pop(i, V*(K+2)+M+1);
20     if rank_i == rank_f
21         final_front(i, :) = intermediate_pop(i, :);
22     end
23 end
24
25 nb_prev_sol = 0;
26 for i = 1:length(final_front)
27     row = sum(final_front(i, :));
28     if row == 0
29         nb_prev_sol = nb_prev_sol + 1;
30     end
31 end
32
33 pop_next(1:nb_prev_sol, :) = intermediate_pop(1:nb_prev_sol, :);
34 pool_size = N-length(pop_next);
35
36 if pool_size > 0
37     contenders = intermediate_pop(nb_prev_sol+1:nb_prev_sol+2*pool_size, :);
38     winners = BTS(contenders, pool_size, V, K, M);
39     winners = sortrows(winners, V*(K+2)+M+1);
40     pop_next(nb_prev_sol+1:nb_prev_sol+pool_size, :) = winners;
41 end
42
43 f = pop_next;
44 end %fcn

```



## A9 path\_selector.m

```

1 %% Path Selector
2 %% Description
3 % *****
4 % This file does post-optimization path selection by selecting a path that
5 % fulfils requirements in terms of desired path properties on objective
6 % function values.
7 %
8 % This file searches for path candidates in the final population.
9 % *****
10 % Programmed by: Tobias Lars Hansen, 2019.
11 % *****
12 %%
13 clear PP;
14 PP = P;
15 %% Scaling factors
16 beta_o2 = 0.8;
17 beta_o4 = 1;
18
19 %% Computations
20 avg_obj_1 = sum(PP(:,V*(K+3)+1))/N;
21 avg_obj_2 = sum(PP(:,V*(K+3)+2))/N;
22 avg_obj_3 = sum(PP(:,V*(K+3)+3))/N;
23 avg_obj_4 = sum(PP(:,V*(K+3)+4))/N;
24
25 min_safety_margin = beta_o2 * avg_obj_2;
26 min_terrain_factor = beta_o4 * avg_obj_4;
27
28 a_shortest_path = min(PP(:,V*(K+3)+1));
29 a_longest_path = max(PP(:,V*(K+3)+1));
30
31 ideal_length = (avg_obj_1 + a_shortest_path)/2;
32
33 flag = 0;
34 counter_discarded = 0;
35 while flag ~= 1
36     [~, a_index] = min(abs(ideal_length - PP(:,V*(K+3)+1)));
37     safety_factor = PP(a_index,V*(K+3)+2);
38     terrain_factor = PP(a_index,V*(K+3)+4);
39
40     if safety_factor > min_safety_margin ||...
41         terrain_factor > min_terrain_factor
42
43         PP(a_index, V*(K+3) + 1) = a_longest_path;
44         counter_discarded = counter_discarded + 1;
45
46     else
47         flag = 1;
48     end
49 end

```

## A10 plot\_tool.m

```
1 close all;
2 %% Description
3 % *****
4 % Post-optimization visualisation of performance.
5 % *****
6 % Programmed by: Tobias Lars Hansen, 2019.
7 % *****
8 %% Plotting objective function values
9 nbPath = a_index;
10
11 x_points = P(:, V*(K+3) + 1);
12 y_points = P(:, V*(K+3) + 2);
13 z_points = P(:, V*(K+3) + 3);
14 zz_points = P(:, V*(K+3) + 4);
15
16 x_ccc = P(nbPath, V*(K+3) + 1);
17 y_ccc = P(nbPath, V*(K+3) + 2);
18 z_ccc = P(nbPath, V*(K+3) + 3);
19 zz_ccc = P(nbPath, V*(K+3) + 4);
20
21 %% 3D plots: Objective function values
22 %% 0_1, 0_2, 0_3
23 figure()
24 scatter3(x_points,y_points, z_points); hold on;
25 xlabel('Objective function 1');
26 ylabel('Objective function 2');
27 zlabel('Objective function 3');
28 schosen = scatter3(x_ccc,y_ccc,z_ccc, 'r', 'filled');
29 LL = legend(schosen,{'Chosen solution'});
30
31 %% 0_1, 0_2, 0_4
32 figure()
33 scatter3(x_points,y_points, zz_points); hold on;
34 schosen = scatter3(x_ccc,y_ccc,zz_ccc, 'r', 'filled');
35 LL = legend(schosen,{'Chosen solution'});
36 xlabel('Objective function 1');
37 ylabel('Objective function 2');
38 zlabel('Objective function 4');
39
40 %% 0_1, 0_3, 0_4
41 figure()
42 scatter3(x_points,z_points, zz_points); hold on;
43 schosen = scatter3(x_ccc,z_ccc,zz_ccc, 'r', 'filled');
44 LL = legend(schosen,{'Chosen solution'});
45 xlabel('Objective function 1');
46 ylabel('Objective function 3');
47 zlabel('Objective function 4');
48
49 %% 0_2, 0_3, 0_4
50 figure()
51 scatter3(y_points,z_points, zz_points); hold on;
52 schosen = scatter3(y_ccc,z_ccc,zz_ccc, 'r', 'filled');
53 LL = legend(schosen,{'Chosen solution'});
54 xlabel('Objective function 2');
```

```
55 ylabel('Objective function 3');
56 zlabel('Objective function 4');
57
58 %% 2D plots: Objective function values
59 %% 0_1, 0_2
60 figure()
61 scatter(x_points,y_points); hold on;
62 sc = scatter(x_ccc,y_ccc, 'r', 'filled');
63 LLL = legend(sc,{'Chosen solution'});
64 xlabel('Objective function 1');
65 ylabel('Objective function 2');
66 grid on;
67
68 %% 0_1, 0_3
69 figure()
70 scatter(x_points,z_points); hold on;
71 sc = scatter(x_ccc,z_ccc, 'r', 'filled');
72 LLL = legend(sc,{'Chosen solution'});
73 xlabel('Objective function 1');
74 ylabel('Objective function 3');
75 grid on;
76
77 %% 0_1, 0_4
78 figure()
79 scatter(x_points,zz_points); hold on;
80 sc = scatter(x_ccc,zz_ccc, 'r', 'filled');
81 LLL = legend(sc,{'Chosen solution'});
82 xlabel('Objective function 1');
83 ylabel('Objective function 4');
84 grid on;
85
86 %% 0_2, 0_3
87 figure()
88 scatter(y_points,z_points); hold on;
89 sc = scatter(y_ccc,z_ccc, 'r', 'filled');
90 LLL = legend(sc,{'Chosen solution'});
91 xlabel('Objective function 2');
92 ylabel('Objective function 3');
93 grid on;
94 %% 0_2, 0_4
95 figure()
96 scatter(y_points,zz_points); hold on;
97 sc = scatter(y_ccc,zz_ccc, 'r', 'filled');
98 LLL = legend(sc,{'Chosen solution'});
99 xlabel('Objective function 2');
100 ylabel('Objective function 4');
101 grid on;
102
103 %% 0_3, 0_4
104 figure()
105 scatter(z_points,zz_points); hold on;
106 sc = scatter(z_ccc,zz_ccc, 'r', 'filled');
107 LLL = legend(sc,{'Chosen solution'});
108 xlabel('Objective function 3');
109 ylabel('Objective function 4');
110 grid on;
```

```

111
112 %% Plotting the selected path
113 % selected by path selector
114 figure()
115 nbWps = K + 3;
116 xc = 1;
117 yc = 2;
118 zc = 3;
119 Path = P(nbPath, :);
120
121 % Waypoints
122 s0 = scatter3(Path(xc), Path(yc), Path(zc), 'b', 'filled'); hold on;
123 xc = 1 + V;
124 yc = 2 + V;
125 zc = 3 + V;
126 for wp = 2: nbWps-1
127     s = scatter3(Path(xc), Path(yc), Path(zc), 'k', 'filled');
128     hold on;
129     s.LineWidth = 0.6;
130     xc = xc + V;
131     yc = yc + V;
132     zc = zc + V;
133 end
134 st = scatter3(Path(xc), Path(yc), Path(zc), 'g', 'filled'); hold on;
135
136 % Line segments
137 xc = 1;
138 yc = 2;
139 zc = 3;
140 for n = 1:nbWps
141     X(n) = Path(xc);
142     Y(n) = Path(yc);
143     Z(n) = Path(zc);
144     xc = xc + V;
145     yc = yc + V;
146     zc = zc + V;
147 end
148 p = plot3(X,Y,Z, 'r'); hold on;
149 p.LineWidth = 2;
150 set( gca, 'Zdir', 'reverse');
151 set( gca, 'Ydir', 'reverse');
152 grid on;
153
154 % Obstacles
155 [XX, YY, ZZ] = sphere;
156 surf(radius_obs*XX+obs_vec(1),...
157     radius_obs*YY+obs_vec(2),...
158     radius_obs*ZZ+obs_vec(3) );
159
160 surf(radius_obs*XX+obs_vec(4),...
161     radius_obs*YY+obs_vec(5),...
162     radius_obs*ZZ+obs_vec(6) );
163
164 % Seabed
165 % xd=linspace(0,150);
166 % yd=linspace(0,150);

```

---

```
167 % [xxx,yyy]=meshgrid(xd,yd);
168 % zzz = 0.007*xxx.^2 + 0.007*yyy.^2 + 0.05*xxx + 0.05*yyy + 40;
169 % CC = del2(zzz);
170 % sbed = mesh(xxx,yyy,zzz);
171
172 % View
173 % 3D settings: None
174
175 % NE settings
176 %view([0 0 1]);
177 %camroll(-90);
178 %view(2);
179
180 % ED settings
181 %view([-1 0 0]);
182
183 % Layout
184 xlim([0, 100]);
185 ylim([0, 100]);
186 zlim([0,200]);
187
188 L = legend([s0 st], {'Initial position', 'Terminal position'});
189 xlabel('North [m]');
190 ylabel('East [m]');
191 zlabel('Down [m]');
```





