Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Peter Hatlebrekke Husebø

Whole-Body Motion Planning and Collision Avoidance for AIAUV

Master's thesis in Cybernetics and Robotics
Supervisor: Kristin Ytterstad Pettersen
June 2022

NTNU
Kunnskap for en bedre verden

Peter Hatlebrekke Husebø

Whole-Body Motion Planning and Collision Avoidance for AIAUV

Master's thesis in Cybernetics and Robotics
Supervisor: Kristin Ytterstad Pettersen
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Problem Description

Articulated Intervention Autonomous Underwater Vehicles (AIAUVs) have emerged from swimming snake robots and combine the slender, articulated body of snakes with the propulsion provided by thrusters. The AIAUVs were recently proposed by the NTNU snake robotics research group and are now industrialized by the spin-off company Eelume AS. The AIAUVs combine several benefits and capabilities of Remotely Operated Vehicles (ROVs) and survey AUVs into one robot: They have advantageous hydrodynamic properties and can travel long distances, like survey AUVs; they have hovering capabilities and can perform light intervention tasks, like ROVs. Moreover, their slender and flexible body provides access that supersedes previously existing marine robots. AIAUVs therefore mitigate the shortcomings of conventional marine robots, and thus enable autonomous operations for ocean sustainment and exploration, including both observation and intervention operations in the same mission, e.g., mapping the seabed and collecting sediments, inspecting and repairing the net of an aquaculture fish cage, and detecting and gathering plastic and other debris polluting the oceans. These new marine robots are already well on their way towards disrupting subsea operations.

Problem definition:
How to avoid collisions along the whole body, not only between the manipulator end and the environment, but between any point along the body and the environment? And how to avoid collisions between the robot and itself?

In order for the author to get more familiar to the problem domain at hand and what related work that has been done the thesis constitutes:

1. Literature study on
    (a) collision avoidance for elongated robots, including whole body motion planning and control.
    (b) collision avoidance for floating-base robot manipulators/vehicle-manipulator systems with a special emphasis on underwater vehicle-manipulator systems.
2. Apply and/or extend one of the methods in 1) for of AIAUVs.
3. Implement and validate the proposed control method(s) in simulation for an AIAUV.

Abstract

The Articulated Intervention Autonomous Underwater Vehicles (AIAUVs) give access to more efficient solutions to a wide range of tasks at sea and are especially suited for missions in cluttered environments. To solve tasks safely and reliably, it is crucial that the AIAUV neither collide with obstacles in the environment nor with itself. This thesis provides a pipeline for external collision avoidance in 3-dimensional cluttered environments. The pipeline consists of off-line global path planners, in form of a Rapidly Exploring Random Tree (RRT) and an Artificial Potential Field (APF), which search for the shortest collision-free path from a starting point to a desired target point in the environment. The AIAUV is guided onto the planned paths by utilization of a 3D Waypoint Line-of-Sight (WLOS) lookahead-based guidance method that incorporates slow-regions for safer path following. To optimize the APF's planned paths efficacy in combination with the WLOS method, this thesis proposes two different filtering methods: Bactracking Path planner (BPP) and Straight Lines of Constant Length (SLCL). In order to follow the guided references from the WLOS method, PID-based autopilot controllers in combination with *feed-forward* are utilized for reference tracking in *surge*, *pitch* and *heading*, while PD-controllers stabilize the joints. The proposed pipeline was shown to successfully guide the AIAUV from the starting point to the target point, while simultaneously avoiding collisions.

Sammendrag

Leddede, autonome undervannsroboter (AIAUVer) baner vei for økt optimalisering av varierte marine operasjoner og er spesielt egnet for operasjoner i områder med begrenset fremkommelighet. For å løse ulike operasjoner på en sikker måte er det ytterst nødvendig at undervannsrobotene unngår å kollidere med både seg selv og omkringliggende hindringer. Denne masteroppgaven presenterer en metode for å oppnå ekstern kollisjonsunngåelse, ved å kombinere ulike off-line baneplanleggere, banefølgingsalgoritmer og kontrollmetoder. To ulike baneplanleggingsmetoder benyttes for å generere globale kollisjonsfrie baner i kjente, avgrensede 3-dimensjonelle områder fylt med hindringer: *Hurtig-Utforskende Tilfeldige Trær* (RRT*) og *Kunstige Feltpotensialer* (APF). Baneplanleggerne søker etter korteste kollisjonsfrie bane fra en startlokasjon til en gitt endelokasjon i omgivelsene. AIAUVen styres til og følger den genererte banen ved bruk av en veipunkt-siktlinje-basert 3-dimensjonal banefølgingsmetode (WLOS) som benytter saktegående soner for sikrere banefølging. For å bedre kombinere APF-metoden og WLOS-metoden benytter denne masteroppgaven to ulike algoritmer for å filtrere banene fra APFen: *Tilbakesporende Baneplanlegger* (BPP) og *Rette Linjer med Konstant Lengde* (SLCL). For å følge referansene fra WLOS-metoden benyttes PID-baserte autopilot-kontrollere i kombinasjon med *foroverkobling*, mens leddene stabiliseres av PD-kontollere. Den presenterte løsningen oppnådde å styre AIAUVen kollisjonsfritt fra startlokasjonen til endelokasjonen.

Contents

Problem Description i

Abstract iii

Sammendrag iv

Preface xxiii

1 Introduction 1

1.1 Motivation 1
1.2 Literature Review 5
1.3 Assumptions 8
1.4 Contributions 8
1.5 Outline 9

2 System Architecture and Mechanical Configuration 11

2.1 System Architecture 11
2.2 Snake Robot Mechanical Configuration 12
2.2.1 Link Modules and Joints 13

3 Modeling 17

3.1 Modeling Prerequisits . . . 17
3.1.1 Reference Frames . . . 18
3.1.2 Pose and Velocity . . . 20
3.1.3 Configuration Space . . . 21
3.1.4 State Space . . . 21
3.1.5 Rotation Matrix . . . 22
3.1.6 Homogeneous Transformation Matrix . . . 23
3.1.7 Angular Velocity and Twists . . . 24
3.2 Kinematics . . . 26
3.2.1 Forward Kinematics . . . 27
3.2.2 Differential Kinematics . . . 29
3.3 Dynamics . . . 34
3.3.1 Equations of Motion . . . 34

4 Path Planning for Collision Avoidance **41**
4.1 Environment . . . 42
4.1.1 Cluttered Environment . . . 43
4.1.2 Configuration Space Obstacle . . . 44
4.2 Artificial Potential Fields . . . 45
4.2.1 Potential functions . . . 46
4.2.2 Gradient Descent for Path Generation . . . 49
4.2.3 Randomized Potential Field . . . 52
4.3 Rapidly Exploring Random Trees . . . 53
4.3.1 RRT . . . 54
4.3.2 RRT* . . . 55

5 Path Following **57**
5.1 Waypoints and Straight Line Paths . . . 58
5.2 Waypoint Generation . . . 60
5.2.1 Straight Lines of Constant Length . . . 61
5.2.2 Backtracking Path Planner . . . 62
5.3 Line of Sight Guidance . . . 66

5.3.1 Path-Tangential Reference Frame and Track-Errors 66
5.3.2 Lookahead-Based Guidance 67
5.3.3 Sideslip Angle and Angle of Attack 69
5.3.4 Forward Speed Law 70
5.3.5 Smoothing References 73

6 Control System **75**
6.1 PID Motion Control System 75
6.1.1 Forward Speed Autopilot 77
6.1.2 Pitch Autopilot 78
6.1.3 Heading Autopilot 78
6.1.4 Joint Controllers 78
6.2 Control Allocation 79

7 Implementation and Simulation **81**
7.1 Path Planning 82
7.1.1 Environment 82
7.1.2 APF Path Planner 85
7.1.3 RRT* Path Planner 93
7.1.4 RRT* vs APF as Global Path Planners 94
7.2 Path Following 97
7.2.1 Case 1: Following BPP-Filtered Path 99
7.2.2 Case 2: Following SLCL-Filtered Path 105
7.2.3 Case 3: Following RRT* Path 111
7.2.4 Case 4: Absence of Slow-Region 117
7.2.5 RRT* vs APF for Path Following 127

8 Conclusion and Future Work **129**
8.1 Conclusion 129
8.2 Future Work 131

A Algorithms **133**

A.1 Gradient Descent 133
A.2 Random Walk 134
A.3 Backtracking Path Planner (BPP) 135
A.4 RRT* 137

References **139**

List of Tables

2.1 Module properties of the snake robot as it is used in the simulation. . 16

7.1 Obstacles and target specifications of **environment 1**. 82
7.2 Obstacles and target specifications of **environment 2**. 83
7.3 Obstacles and target specifications of **environment 3**. 83
7.4 Hyperparameter values for the APF path planner. 87
7.5 Hyperparameter values for the RRT* path planner. 93
7.6 Distances of the paths planned in **environment 1** and **environment 2** by the APF planner with filtering and the RRT* planner. 96
7.7 Tuning parameters values for the guidance and forward speed laws. . 98
7.8 Tuning parameters values for the autopilot controllers. 98
7.9 Tuning parameters values for the joint PD-controllers. 98
7.10 Total thruster work, path distance and work done pr.metre from start position to target position for **Case 1-3**. 127

List of Figures

1.1 A ROV operating at the sea bed. 3
1.2 An illustration of an AUV scanning the sea bed to map it or detect certain objects. 3
1.3 An illustration of the Eelume AIAUV investigating a pipe line. Courtesy of Eelume AS 4

2.1 The system architecture of a general marine craft. 12
2.2 Revolute joint and its *parent* and *child* links. 13
2.3 A variant of the snake robot Eelume, used as basis for the simulation model. 15
2.4 An illustration of the revolute joints of the snake robot and their physical limits. 15
2.5 The snake robot's different modules and joints used in the simulation. 16

3.1 Snake robot model in the system architecture. 18
3.2 The inertial frame and the various frames related to the snake robot. 19

4.1 Guidance system in the system architecture. 42
4.2 Environment consisting of 4 spherical obstacles of different size, the target position (red) and the starting position (blue). 44
4.3 Illustration of the safe radius used for path planning. 45

4.4 A lamp represented by spheres. The approximation improves as the number of spheres used to represent the lamp increases. 45
4.5 Illustration of local minima and global minimum of objective functions in 1D and 2D. 50
4.6 Illustration of a planar path trapped by a local minimum due to an obstacle located right in front of the desired path to the target point. . 50
4.7 Illustration of a planar path that stops in a local minimum between two obstacles located in front of the target. 51
4.8 Illustration of a collision-free path in a 2D plane that succeeds to reach the target. 51
4.9 Illustration of *random walk* utilized to escape local minima in a 2D $\mathcal{W}$-space. 53
4.10 The process of randomly drawing a new sample and add it to the tree, following the constraint set by ϵ_{upper}. 55
4.11 The resulting tree $\mathcal{T}$ projected in the 2D space, showing how the tree expands from the initial configuration (red dot) to the goal (green dot), while avoiding the obstacles (blue dots). 56

5.1 Guidance system in the system architecture. 58
5.2 An example of a piecewise linear path of waypoints. 60
5.3 Visualize how the resulting piecewise linear path (dashed grey lines) from the SLCL-filtering algorithm will look like in 2D. 62
5.4 Visualize how the resulting piecewise linear path (dashed grey lines) from the BPP-filtering algorithm could look like in 2D. 63
5.5 Visualizing how d_{orth} can be used to decide if the straight line $\mathcal{P}_i$ stays clear of the obstacle. 65
5.6 Visualizing how d_{proj} can be used to decide if the straight line $\mathcal{P}_i$ stays clear of the obstacle. 65
5.7 The orientational relationship from $\mathcal{F}_I$ to $\mathcal{F}_{\mathcal{P}_i}$ 67
5.8 The path-tangential reference frame $\mathcal{F}_{\mathcal{P}_i}$ wrt $\mathcal{F}_I$, along with the main steering variables for 3D-LOS. 69

5.9 The horizontal ocean triangle, with the snake robot's base as the grey rectangle. 71
5.10 The vertical ocean triangle, with the snake robot's base as the grey rectangle. 72

6.1 Control system in the system architecture. 76

7.1 The three environments used to test the different path planners performance. 84
7.2 Different tuning of α results in different paths, where some might be impracticable. 88
7.3 The planned paths in **environment 1**, based on the APF planner. . . 90
7.4 The planned paths in **environment 2**, based on the APFplanner. . . 91
7.5 The APF planner fails to plan a path in **environment 3** because of the high obstacle density. 92
7.6 The planned paths in all three environments, based on the RRT*-planner. 94
7.7 The BPP-filtered path and the actual path of the centre link's CM. . . 100
7.8 The closest distance between the obstacles in the $\mathcal{W}$-space and the links and end effector of the snake robot. 101
7.9 The desired and actual *surge*, *pitch* and *heading* states of the snake robot while following the BPP-filtered path. 102
7.10 The joint angles and joint torques during path following of the BPP-filtered path. 103
7.11 The thruster forces during path following of the BPP-filtered path. . . 104
7.12 The SLCL-filtered path and the actual path of the centre link's CM. . 106
7.13 The distance between the obstacles in the $\mathcal{W}$-space and the links and end effector of the snake robot. 107
7.14 The desired and actual *surge*, *pitch* and *heading* states of the snake robot while following the SLCL-filtered path 108
7.15 The joint angles and joint torques during path following of the SLCL-filtered path. 109
7.16 The thruster forces during path following of the SLCL-filtered path. . 110

7.17 The RRT* path and the actual path of the centre link's CM. 112
7.18 The distance between the obstacles in the $\mathcal{W}$-space and the links and end effector of the snake robot. 113
7.19 The desired and actual *surge*, *pitch* and *heading* states of the snake robot while following the RRT* path. 114
7.20 The joint angles and joint torques during path following of the RRT* path. 115
7.21 The thruster forces during path following of the RRT* path. 116
7.22 The BPP-filtered path and the actual path of the centre link's CM when the slow-region is absent. 120
7.23 The desired and actual *surge*, *pitch* and *heading* states of the snake robot when following the BPP-filtered path and the slow-region is absent. 121
7.24 The SLCL-filtered path and the actual path of the centre link's CM when the slow-region is absent. 122
7.25 The desired and actual *surge*, *pitch* and *heading* states of the snake robot when following the SLCL-filtered path and the slow-region is absent. 123
7.26 The RRT* path and the actual path of the centre link's CM when the slow-region is absent. 124
7.27 The desired and actual *surge*, *pitch* and *heading* states of the snake robot when following the RRT* path and the slow-region is absent. . 125
7.28 The distance between the obstacles in the $\mathcal{W}$-space and the links and end effector of the snake robot when the slow-region is absent. 126

Acronyms

AIAUV Articulated Intervention Autonomous Underwater Vehicle.

APF Artificial Potential Field.

AUV Autonomous Underwater Vehicle.

BPP Bactracking Path planner.

CAP Collision Avoidance Points.

CB Centre of Buoyancy.

CM Centre of Mass.

CO Coordinate Origin.

DOF Degrees of Freedom.

ILOS Integral LOS.

IMR Inspection, Maintenance and Repair.

LOS Line-of-Sight.

PID Proportional, Integral and Derivative.

ROV Remotely Operated Vehicles.

RPF Randomized Potential Field.

RRT Rapidly Exploring Random Tree.

SLCL Straight Lines of Constant Length.

TCM Thrust Configuration Matrix.

UAV Unmanned Aerial Vehicle.

UVMS Underwater Vehicle-Manipulator System.

WLOS Waypoint Line-of-Sight.

WPG Waypoint Guidance.

Symbols

Domain Symbols

$\mathcal{W}$	Work-space domain
$\mathcal{C}$	Configuration-space domain
$\mathcal{S}$	State-space domain
$\mathcal{CO}$	Configuration space obstacle domain

Reference Frames

$\mathcal{F}_I$	Reference frame of inertial frame I, following the NED-convention
$\mathcal{F}_b$	Reference frame of the snake robot's base link
$\bar{\mathcal{F}}_j$	Reference frame of joint j
$\mathcal{F}_l$	Reference frame of link l
$\bar{\mathcal{F}}_{ee}$	Reference frame of end effector

Snake Robot Symbols

n	Number of links
m	Number of thrusters
$\boldsymbol{p}_{Ib}^{I}$	Position vector describing the position of frame $\mathcal{F}_b$ wrt frame $\mathcal{F}_I$, expressed in $\mathcal{F}_I$-coordinates, containing positional coordinates $[x^I,\ y^I,\ z^I]^T$
$\boldsymbol{\Theta}_{Ib}$	Orientational vector of frame $\mathcal{F}_b$ wrt frame $\mathcal{F}_I$, containing roll, pitch and yaw angles $[\phi,\ \theta,\ \psi]^T$
$\boldsymbol{q}_b^I$	Quaternion parametrization vector of the orientation of frame $\mathcal{F}_b$ wrt frame $\mathcal{F}_I$, containing $[\eta,\ \epsilon_1,\ \epsilon_2,\ \epsilon_3]^T$
$\boldsymbol{v}_{Ib}^{b}$	Linear velocity vector describing the linear velocity of frame $\mathcal{F}_b$ wrt frame $\mathcal{F}_I$, expressed in $\mathcal{F}_b$-coordinates, containing *surge*, *sway* and *heave* $[u,\ v,\ w]^T$
$\boldsymbol{\omega}_{Ib}^{b}$	Angular velocity vector describing the angular velocity of frame $\mathcal{F}_b$ wrt frame $\mathcal{F}_I$, expressed in $\mathcal{F}_b$-coordinates, containing $[p,\ q,\ r]^T$
$\boldsymbol{\eta}$	Pose vector of the snake robot base frame $\mathcal{F}_b$, containing $[\boldsymbol{p}_{Ib}^{I},\ \boldsymbol{\Theta}_{Ib}]^T$
$\boldsymbol{\nu}_{Ib}^{b}$	Velocity vector of the snake robot base frame $\mathcal{F}_b$, containing $[\boldsymbol{v}_{Ib}^{b},\ \boldsymbol{\omega}_{Ib}^{b}]^T$
q	Joint angle
$\boldsymbol{q}$	Joint angles vector, containing $n-1$ joint angles, $[q_1, .., q_{n-1}]^T$
$\dot{q}$	Joint angular velocity
$\dot{\boldsymbol{q}}$	Joint angular velocity vector, containing $n-1$ joint angle velocities, $[\dot{q}_1, ..., \dot{q}_{n-1}]^T$

$\boldsymbol{\xi}$ Configuration vector of the snake robot, containing pose and joint angles $[\boldsymbol{\eta},\ \boldsymbol{q}]^T$

$\boldsymbol{\zeta}$ Velocity vector of the snake robot, containing both velocity of the base and joints $[\boldsymbol{\nu}_{Ib}^b,\ \dot{\boldsymbol{q}}]^T$

$\boldsymbol{t}_{ik}^i$ Translation vector describing the distance between the CO of frame $\mathcal{F}_k$ relative to the CO of frame $\mathcal{F}_i$, expressed in $\mathcal{F}_i$

$\boldsymbol{R}_k^i$ Rotation matrix describing orientation of frame $\mathcal{F}_k$ wrt frame $\mathcal{F}_i$

$\boldsymbol{H}_k^i$ Homogeneous transformation matrix describing the pose of frame $\mathcal{F}_k$ wrt frame $\mathcal{F}_i$

$\boldsymbol{J}_j$ Velocity jacobian relating the velocities of the joint angles to the velocity of frame $\bar{\mathcal{F}}_j$ expressed in $\mathcal{F}_b$

$\boldsymbol{J}_e$ Velocity jacobian relating the velocities of the joint angles to the velocity of frame $\bar{\mathcal{F}}_{ee}$ expressed in $\mathcal{F}_b$

$\boldsymbol{J}_{\boldsymbol{H},j}$ Velocity jacobian relating the velocities of the base link and joint angles to the velocity of frame $\bar{\mathcal{F}}_j$ expressed in $\bar{\mathcal{F}}_j$

$\boldsymbol{J}_{\boldsymbol{H},e}$ Velocity jacobian relating the velocities of the base link and the joint angles to the velocity of frame $\bar{\mathcal{F}}_{ee}$ expressed in $\bar{\mathcal{F}}_{ee}$

α_{aoa} Angle of attack

β_{ss} Sideslip angle

$\boldsymbol{\tau}$ Generalized force vector for the snake robot's actuators

$\boldsymbol{\tau}_b$ Generalized force vector for the snake robot's thrusters

$\boldsymbol{\tau}_q$ Generalized force vector for the snake robot's joint torques

Artificial Potential Field Symbols

U Potential field

U_a Attractive potential field

U_r Repulsive potential field

∇U_a Gradient of attractive potential field U_a

∇U_r Gradient of repulsive Potential field U_r

K_a Scaling coefficient of an attractive potential field U_a

K_r Scaling coefficient of a repulsive potential field U_r

d_t^* Switching distance parameter for hybrid attractive potential field U_a

Q_o^* Smoothing parameter of a repulsive potential field U_r

α Step size for *gradient descent* algorithm

ϵ Gradient tolerance threshold for termination of *gradient descent* algorithm

r_{safe} Safe radius to incorporate safety margins related to the path following task

N_{rw} Number of random steps execute during the *random walk*

δ_{rw} The step size in *random walk*

$\epsilon_{\text{trapped}}$ The threshold value for detecting local minima

Rapidly Exploring Random Trees Symbols

N_{s} The number of nodes/samples to search for while building the RRT tree

r_{nh} The search radius for finding neighbourhood samples in order to optimize the RRT* tree

ϵ_{lower} The minimum length of a branch of the RRT tree

ϵ_{upper} The maximum length of a branch of the RRT tree

υ_{lower} The minimum angle between two straight line segments

Guidance Symbols

Δ Horizontal lookahead distance

μ Tuning parameter for vertical lookahead distance

κ Tuning parameter for desired forward speed for path following

r_{soa} Radius of the *sphere of acceptance*

r_{sr} Radius of the slow-region sphere

K_{sr} The forward speed scaling factor inside of a slow-region

Path and Obstacle Symbols

$\boldsymbol{wp}_i^I$	Position of waypoint i, expressed in the inertial frame $\mathcal{F}_I$
$\mathcal{P}_i$	Straight line segment from waypoint $\boldsymbol{wp}_i^I$ to $\boldsymbol{wp}_{i+1}^I$
$\chi_{\mathcal{P}_i}$	Azimuth angle of the straight line $\mathcal{P}_i$
$\gamma_{\mathcal{P}_i}$	Elevation angle of the straight line $\mathcal{P}_i$
$\boldsymbol{p}_t$	Target position, the desired goal of the snake robot
$\boldsymbol{p}_{\text{start}}$	Start position of the snake robot
N_o	Number of obstacles
$\mathcal{O}_i$	The i^{th} obstacle, where $i = 1, ..., N_o$
$\boldsymbol{p}_{\mathcal{O}_i}$	Position of the CO of $\mathcal{O}_i$
$r_{\mathcal{O}_i}$	Radius of obstacle $\mathcal{O}_i$

Preface

This master's thesis is a continuation of the specialization pre-project I conducted during the autumn of 2021. As is customary, the specialization project is not published. This means that important background theory and methods from the project report will be restated in full throughout this report to provide the best reading experience. Below, a complete list of the material included from the specialization project is listed.

- Chapter 1 (specifically sections 1.1 and 1.2, with some changes to Section 1.2)
- Chapter 2 (specifically Section 2.2, with some added material and reformulations)
- Chapter 3 (specifically sections 3.1.1, 3.1.2, 3.1.6 and 3.2.1, with some changes)
- Chapter 4 (specifically sections 4.2.1 and 4.2.2)
- Chapter 5 (specifically Section 5.1 and sections 5.3.1 and 5.3.2, with some changes)

The work presented in this thesis has been carried out under the supervision of Prof. Kristin Y. Pettersen at the Department of Engineering Cybernetics, NTNU. During the thesis, I have been provided multiple tools through PhD candidate Henrik Schmidt-Didlaukies, who gave access to the Matlab/Simulink simulator for the snake robot Eelume, to be used in the work.

Unless otherwise stated, all figures and illustrations have been created by the author.

I would like to thank Kristin Y. Pettersen, Henrik Schmidt-Didlaukies and Marianna Wrzos-Kaminska for their guidance and input throughout the project. Also, I would like to thank the whole "snakesnakk" research group for many meetings with both exciting and informative discussions and presentations regarding the snake robot.

Peter Hatlebrekke Husebø

Trondheim, June 2022

Chapter 1

Introduction

This introductory chapter starts by motivating the problem to be solved. Secondly, a literature review is presented, containing previous work related to path following and collision avoidance for autonomous vessels in general, but with special emphasize on work related to UVMSs. Note that both Section 1.1 and parts of Section 1.2 originally stems from the specializations pre-project conducted by the author, but as this thesis directly builds on the pre-project they are reused here. The scope of the work to be done in the thesis is then defined through a list of assumptions. Finally, the contributions of the thesis are defined and elaborated.

1.1 Motivation

The ocean covers approximately 71% of the earth's surface and is estimated to be 3.8 kilometres deep on average [28]. It's fair to say that the ocean is huge, and it has been a part of humans life throughout our history, now more than ever. The ocean has provided food and minerals, and large oil reservoirs beneath the ocean floor has been discovered and utilized as a main source of energy all over the world. In addition, the

majority of all cargo delivered in the world is transported by ships. Even though we have adequately conquered the ocean surface and proven our capability of gathering invaluable mineral resources and food hundreds of metres below sea level, there are still many undiscovered parts of the ocean that may show great potential. In fact, as of today approximately 80% of the ocean remains undiscovered [5]. Furthermore, many of the facilities at sea such as fish farms and oil rigs require periodically Inspection, Maintenance and Repair (IMR). Both further exploration of the ocean and IMR of existing facilities are very difficult or even impossible for a human being to execute directly. It would be physically impossible for a human to dive several kilometres down below sea level without getting crushed by the enormous pressure at that depth. The utilization of submersible vessels carrying humans, such as submarines, solves this problem, but being able to explore completely without risking human lives is unquestionably the top priority. The same arguments could also be given regarding the IMR tasks. Humans are not biologically built to operate effectively in the sea, and they may use a lot of time on even a simple task such as inspecting damages of a fish farm net installed offshore. This is where Underwater Vehicle-Manipulator Systems (UVMS) greatly improve the possibilities of solving such tasks.

ROVs, seen in Figure 1.1, accommodates the desire of safe operations in terms of putting no human lives at risk. They are commonly used for vessel hull inspection and object detection to prevent subsea navigation hazards. They are also equipped with manipulators, enabling light intervention tasks such as mounting hooks on submerged infrastructure that needs to be surfaced or collecting sediments at the sea bed. Even though they are very well suited to solve various subsea tasks they are still controlled by humans and may not function optimal in terms of both reaction time, precision and accuracy. To be able to remotely operate the ROVs they are connected to the control room through a cable, bounding their maximum travelling distance and restricting their motion in narrow submerged environments such as inside ship wrecks.

AUVs, illustrated in Figure 1.2, demand no real-time input or control from a human operator, and thus do not have any physical connection to a control room in terms of a cable. They can be equipped with many various sensors to be able to execute different

tasks. This makes them suited for large distance surveys, where tasks could range from looking for ship and plane wrecks to ocean mapping and measuring various states of the ocean at different locations, e.g. temperature, pollution and ocean currents.

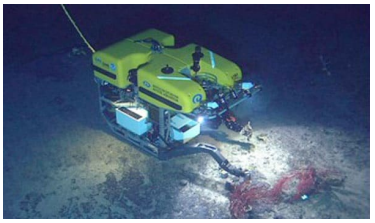


Figure 1.1: A ROV operating at the sea bed. It has turned on its lights to be able to see with its cameras and interact with its manipulators.

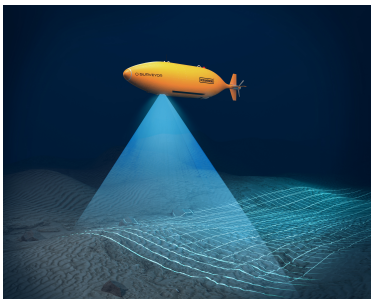


Figure 1.2: An illustration of an AUV scanning the sea bed to map it or detect certain objects.

The AIAUVs enable further exploration of the ocean and improve the exploitation of its discovered domain. Through an AIAUV's capability of executing light-intervention tasks it is able to interact with its environment in various ways, just like the ROVs. Such tasks could for instance be inspecting a pipe line, a fish net or a ship wreck, collecting sediments at and map the ocean floor. Additionally, the AIAUVs have also inherited advantageous hydrodynamic properties from the AUV, making them able to travel long distances. In particular, the AIAUV snake robot called Eelume, seen in Figure 1.3, is very well suited for subsea operations. Its body is articulated, meaning that the snake robot consists of an ensemble of links connected by joints, giving the snake robot a large Degrees of Freedom (DOF) and thus the ability to change its shape according to the task it needs to execute. As a result, AIAUVs are not only capable of solving a great number of various tasks, they have the potential to solve them in an optimal manner, both considering time efficiency and energy consumption.

While operating at sea all types of UVMSs need the ability to avoid collisions with objects in order to be able to execute their main tasks. Collision avoidance is thus a

task by itself, one of great importance in terms of operational safety. The potential colliding objects might be static objects that the UVMSs need to avoid while moving from one location to another, or dynamic objects that may hit them while they are standing still and inspect a vessel hull, gather minerals from the sea bed, or execute any other task. The task of collision avoidance is of the utmost importance, otherwise the UVMSs might not be able to execute their main goal, resulting in mission failure. In worst case they might also get severely damaged by larger impacts with other objects. The task of collision avoidance gets even more complicated for the Eelume snake robot, due to its high number of DOFs. In addition to the necessity of being able to prevent collisions with other objects it also needs to be aware of its own links positions relative to each other in order to not collide with itself. The large DOF of the AIAUV makes it a nontrivial task to decide how it should move to solve the collision avoidance task. This motivates the need for developing methods for motion planning and control schemes in order to prevent collisions, both with other objects and itself, that otherwise would result in task failure and potentially make damage to the AIAUV and its surroundings.

Figure 1.3: An illustration of the Eelume AIAUV investigating a pipe line. Courtesy of Eelume AS

1.2 Literature Review

The collision avoidance task of mobile robots operating in cluttered, dynamically varying environments has been a research topic for decades. There exist a lot of useful results that is worth looking into to learn and take advantage of to achieve collision avoidance for an UVMSs, not only limited to the collision avoidance research of underwater vessels. Ideas and results from both autonomous land based vehicles and Unmanned Aerial Vehicles (UAV) are also of great importance, as their solutions are built with the same goal in mind - collision avoidance. This chapter will present some earlier research on the matter that is relevant to the work presented in this thesis and potentially future work to be further investigated.

In [25], a collision avoidance task of a multi-agent system of N wheeled robots, each called an agent, is considered. The paper considers both collision avoidance between the agents as well as between an agent and static obstacles. The problem is solved by exploiting a differential game formulation, a game theoretic approach that enables numerous objectives, which may or may not be in conflict with each other, to be considered simultaneously. In [35], *Wrzos-Kaminska et al.* manage to prevent self collisions of the different links of an AIAUV changing its configuration. The solution builds on the differential game approach from [25] to be able to find the optimal control input to a system of double integrator dynamics. The double integrator dynamics are obtained through feedback linearization of the snake robot dynamics. To attain robustness and deal with disturbances due to estimation errors, a mixed H_2/H_∞ control problem is solved, where the control input and the disturbance act as players of a differential game. As stated in [24], the mixed H_2/H_∞ control has shown great performance in certain situations where it is of interest to seek both optimality and robustness. The utilization of mixed H_2/H_∞ control in [35] is based on [24], which considers H_2/H_∞ for a general class of nonlinear systems.

A path planner and collision avoidance method utilizing an Artificial Potential Field (APF) was developed in [15]. The APF is here used to plan a collision free path for an underwater snake robot to follow, and the operational area consists of obstacles and

a final target position that the snake robot is trying to reach. The main idea is that that the obstacles and the target create a potential field. The obstacles repel the snake robot while the target attracts it, potentially leading to a collision free path that the snake robot can follow to reach its target. To be able to follow the derived path the paper propose a number, N, of waypoints along the path that the snake robot should pass on its way to the target, a method known as Waypoint Guidance (WPG). The path thus consists of $N-1$ straight lines connecting two waypoints. In order to guide the snake robot and make it follow the waypoints the paper utilizes the well-known Line-of-Sight (LOS) lookahead-based guidance law. The control structure developed to enable the path following consists of an outer-loop controller for the formation of the reference joints for tracking the desired heading and an inner-loop controller to make the joints follow their references. The downside of this solution is the fact that the LOS guidance law does not consider ocean current disturbances, which may prevent the snake robot from converging to the path. This could be solved by utilizing Integral LOS (ILOS), which adds integral effect to the guidance law. This ILOS guidance law is in fact added in [14], where a constant irrotational current is disturbing an underwater snake robot following a path. Another fact that is worth mentioning is that the paper only considers movement in a 2D-plane, while the snake robot eventually will be operating in 3D-space.

Path following and obstacle avoidance are also considered in [17], where a set-based guidance strategy is utilized for underwater snake robots conducting planar sinusoidal motion. The idea of the guidance scheme is that the snake robot is set to follow a straight path, but when it gets to close to an obstacle along that path, in other words leave the set of where its movements are guaranteed collision free, its task switch to follow a circular path around the obstacle in order to avoid it. When the snake robot's reference velocity vector points out of the circular path it follows, the snake robot may switch back to the task of following the original straight line path. The limits of this paper is that it only considers motion in the plane, as was also the case for [15], while a 3D solution would be needed for most real life tasks of underwater vehicles. The paper also assumes that the snake robot is not exposed to ocean currents, which is

rarely the case in real life.

Implementation of path following and collision avoidance controllers in various cluttered 3D environments was done in a master thesis [11] for an underwater swimming manipulator. In this work the LOS guidance law was extended from the 2D case to the 3D case and tested for both straight line and various circular-like paths. The collision avoidance method of this master thesis is closely related to the solution from [17], where the collision avoidance task is activated when getting closer to an obstacle than desired. The difference is that this thesis considers multiple points on the robot, Collision Avoidance Points (CAP), that may collide and need to be maneuvered in a collision free manner, while in [17] a circle with a radius considering the length of the snake robot as well as the radius of the obstacle defines activation of the task, triggered by the distance between the overall centre of mass of the snake robot and the obstacle centre.

In [18], a path planner based on Rapidly-exploring Random Trees (RRTs) was developed for efficiently solving single-query path planning problems for moving an object from a start configuration to a goal configuration in a cluttered environment. The method consisted of two RRTs that grows towards each other, one from start to goal and the other from goal to start, which when connected to each other creates a path from the starting configuration to the goal configuration. The algorithm was tested on a 6-DOF PUMA arm and successfully generated collision-free motions in a 3D work-space.

The RRT algorithm was also utilized in [34], where they developed a specialized RRT (sp-RRT) algorithm for follow-the-leader motion of hyper-redundant manipulators in confined spaces. In a follow-the-leader strategy, the end effector moves along a path, while the rest of the body follows the end effector's motion, which is discussed in [6]. The method showed to efficiently generate a collision-free path that could be followed easily and guarantee the final pose of the end effector. In addition, the algorithm was adaptive to manipulators with different link segments.

1.3 Assumptions

This section presents the assumptions the author has made in order to accommodate the scope of this project. The assumptions are:

A1: The AIAUV is fully observable.

A2: The AIAUV has perfect knowledge of its surroundings, i.e knowledge about the positions of the external obstacles in the 3-dimensional work-space it operates in.

A3: The external obstacles are assumed to be static, but in general they can be both static and dynamic.

A4: The AIAUV has the ability to use all its thrusters at any time instant.

A5: The AIAUV is only in *transit* from A to B, not performing any *manipulation* tasks with its end effector.

A6: The AIAUV is not exposed to ocean currents.

A7: The AIAUV is passive stable in roll.

1.4 Contributions

This section presents and briefly elaborates on the contributions of this master's thesis. The main contributions are as follows:

C1: The implementation and validation of a Rapidly Exploring Random Tree (RRT)* and an Artificial Potential Field (APF) for global off-line collision-free path planning in known cluttered 3-dimensional environments. The APF is implemented with a *random walk* mode, giving it the ability to escape local minima and converge to the target.

C2: The development of a backtracking algorithm that divides a planned collision-free path into a shorter piecewise linear path. The algorithm searches for the smallest number of waypoints that still result in a collision-free path. This path is further sent to the *guidance* and *control* systems of the snake robot for path following purposes.

C3: The development of a WLOS guidance method that incorporates slow-regions for safer path following behaviour in cluttered environments.

C4: A simulation study evaluation of the proposed path planners' performance in various cluttered environments and the path following performance of the proposed guidance laws and control method.

1.5 Outline

The report is organized as follows: In Chapter 2, a general system architecture for marine crafts is presented, which relates well to the snake robot. In addition, the snake robot's mechanical configuration and the physical dimensions used in the simulation will be presented. In Chapter 3, the kinematic and dynamic models of the snake robot is presented, and important aspects of kinematic and dynamic modelling are introduced. Chapter 4 presents the task of planning a collision-free path in cluttered environments, focusing on the APF and RRT methods. Chapter 5 gives an overview of the path following task, focusing on the Waypoint Guidance (WPG) method and Waypoint Line-of-Sight (WLOS) guidance laws. In Chapter 6, the motion control system of the snake robot is presented. Chapter 7 presents the implementation and simulation of the aforementioned path planning and control systems for path following tasks, and discuss the simulation results. In Chapter 8, a conclusion of the work done throughout the master's thesis is presented, as well as related future work to be further investigated, related to the topic of motion planning and control for collision avoidance of the snake robot.

Chapter 2

System Architecture and Mechanical Configuration

As an introduction to the snake robot, this chapter will present a general overview of the *guidance*, *navigation* and *control* system architecture of a marine craft, which is well suited for the snake robot. Secondly, in Section 2.2, the mechanical configuration of the snake robot is presented in order for the reader to get more familiar with its various components and capabilities. The majority of Section 2.2 was first written by the author of this thesis as a part of the specialization pre-project, and is restated here as it is highly relevant for this thesis.

2.1 System Architecture

The system architecture shown in Figure 2.1 gives an overview of the different systems a general marine craft consists of, which is mainly three systems; *guidance*, *navigation* and *control*. In addition, a model of the marine craft is included, which is essential to

calculate and predict how the craft will respond to various inputs from the control system and for estimations of the various states of the marine craft, which for the snake robot often relates to position, attitude and velocity of its end effector. The highlighted systems in Figure 2.1 will be focused on throughout this thesis, which are the *guidance* and *control* systems and the marine craft model. The other systems' signals to these three, which relates to signals from the on-board sensors and the *navigation* system, will in this thesis be assumed perfect, as stated in Section 1.3.

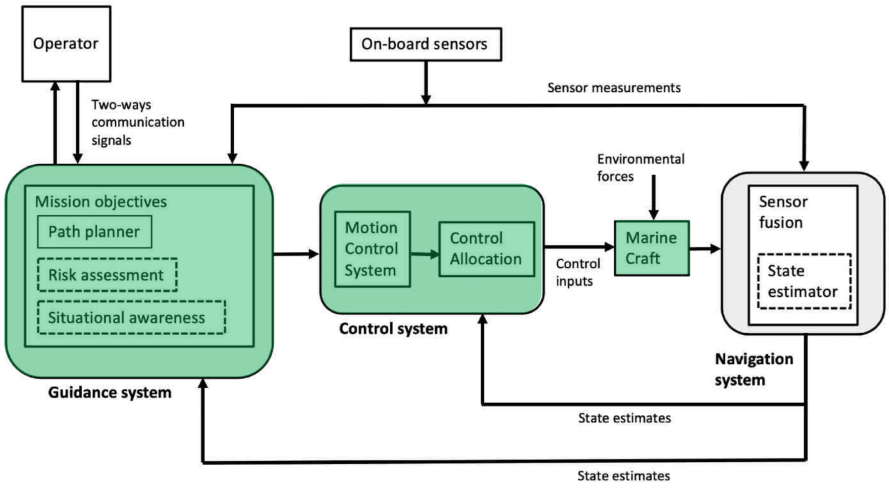


Figure 2.1: The system architecture of a general marine craft. Courtesy of [10].

2.2 Snake Robot Mechanical Configuration

The importance of understanding the physical limits and capabilities of a robotic system is essential to be able to control it in a reasonable manner. First, this section will present and briefly describe the different modules and joints the snake robot consists of. Then, an example of the snake robot's mechanical configuration will be presented, which originates from [30], the paper that the simulation model used for this thesis is based on.

2.2.1 Link Modules and Joints

Serial link manipulators, often referred to as open-chain manipulators, are well known in the field of robotics. The manipulators consist of rigid body links and actuated joints, where each joint is connected to a *parent* and a *child* link [23]. There exists various types of joints, where in this thesis only the revolute kind will be used for the snake robot. The revolute joint allows rotational motion about the joint axis, meaning that it has only 1 DOF [23]. An illustration of a revolute joint and its *parent* and *child* links is shown in Figure 2.2.

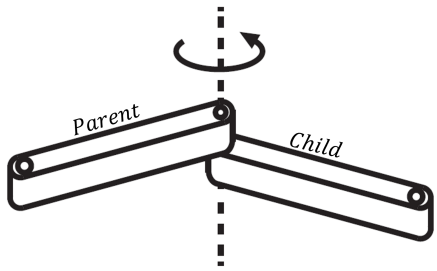


Figure 2.2: Revolute joint and its *parent* and *child* links. Courtesy of [23].

The snake robot discussed in this thesis is a type of serial link manipulator, with n cylindrical links and $n-1$ actuated joints in general. Due to the fact that the snake robot is not mounted to the ground but is free to move in 3D-space, it acts as a floating-base manipulator. Each joint is actuated by a servo motor that is capable of exerting a torque up to 10[Nm], and they have the ability to rotate $\pm 65^\circ$ about their axis of rotation. Even though all the links are considered to be rigid bodies, the snake robot in general is not, due to its shape-shifting ability. It can change its configuration by moving its joint angles. Nevertheless, the snake robot can be seen as a rigid body when the joints are stationary in a period of time.

The snake robot has 6 different module designs for the links, which all have different functions:

1. *Base module*:
   This is the rear end of the snake robot, where the tether is connected for remotely controlled operations. In addition, it is also mounted a camera on this module for visualization of the snake robot's surroundings.

2. *Coupling module w/o thrusters*:
   This is a small link module between two revolute joints. The joint mounted on the backside of the coupling link rotates about its Z-axis, while the joint mounted at the front rotates about its Y-axis. The two joint variants are illustrated in Figure 2.4, where also the limit of each joint is shown.

3. *Double module back*:
   This module is equipped with 2 tunnel thrusters, one acting in the Z-axis direction of the link and the other in the direction of its Y-axis.

4. *Center module*:
   This module is equipped with 3 thrusters, 2 acting in the direction of the link's X-axis and 1 tunnel thruster acting in its Z-axis direction.

5. *Double module front*:
   This module is similar to the *double module back*, except that its tunnel thrusters positions are interchanged compared to the ones in the *double module back*.

6. *Front module*:
   This module is the head of the snake robot and is used for operating tools for various intervention tasks, and is equipped with a camera and lights.

The modular design of the snake robot gives the ability to assemble several variants of snake robots for different purposes. In the snake robot simulator model, based on [30], the snake robot consists of 9 cylindrical links of radius 9cm and 8 joints. The links properties are as shown in Table 2.1. All the mounted thrusters are in the simulation capable of exerting a force up to 40[N]. The connection of the different links and joints of the snake robot are visualized in Figure 2.5. In addition, an illustration of how the

snake robot looks like in real life for the described configuration is shown in Figure 2.3, which is the same as in [30].

Figure 2.3: A variant of the snake robot Eelume, used as basis for the simulation model.

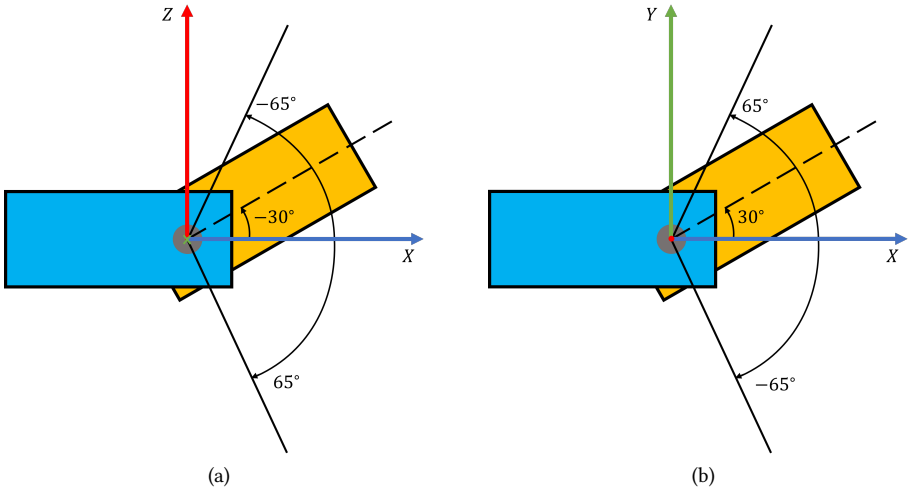


Figure 2.4: An illustration of the revolute joints of the snake robot and their physical limits. (a) Illustrates the joint rotating about its Y-axis. The Y-axis is here pointing into the sheet, following the right-hand rule. (b) Illustrates the joint rotating about its Z-axis. The Z-axis is here pointing out of the sheet, following the right-hand rule.

Link Properties				
Link No.	**Length [m]**	**Mass [kg]**	**Thrusters**	**Module Type**
1	0.62	14.3	None	Base Module
2, 4, 6, 8	0.10	6.0	None	Coupling Module w/o Thrusters
3	0.59	12.7	2: Z, Y	Double Module Back
5	0.80	9.8	3: X, X, Z	Centre Module
7	0.59	12.7	2: Y, Z	Double Module Front
9	0.37	7.8	None	Front Module

Table 2.1: Module properties of the snake robot as it is used in the simulation.

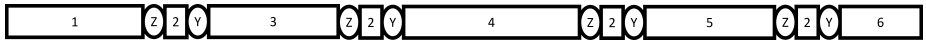


Figure 2.5: The snake robot's different modules and joints used in the simulation. The numbered rectangles represent links and their module type. The ellipses represent the joints, where Z and Y describe which axis the joints rotate about.

Chapter 3

Modeling

In order to make any robotic system execute different types of tasks we need to have a model of its motional behaviour. There exist mainly two models, kinematic and kinetic, where the kinematic aim to describe the motion of a system without considering the forces that causes the system to move. Kinetic models, or also called dynamic models, describes the motion of the system while also accounting for the forces acting on the system. This chapter will thus present the kinematic and dynamic modeling needed to describe and further control the motion of the snake robot, which relates to the system architecture as shown in Figure 3.1.

3.1 Modeling Prerequisits

Before introducing the different parts of the kinematic modeling it is important to establish some ground concepts that the modeling directly builds on. This relates to reference frames, pose and velocity of the robotic system, as well as its configuration and state space representation. In addition, rotation matrices and homogeneous transformation matrices will be presented. Some of the material in sections 3.1.1, 3.1.2

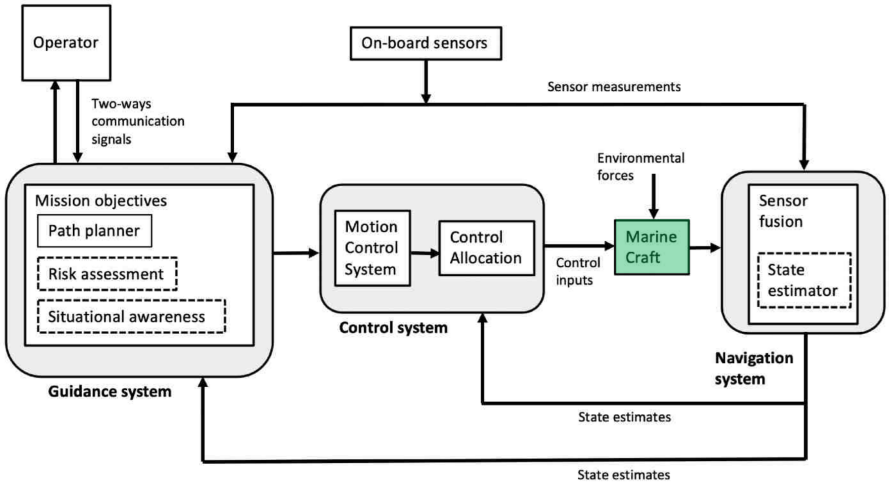


Figure 3.1: Snake robot model in the system architecture. Courtesy of [10]

and 3.1.6 is restated from the specialization pre-project.

3.1.1 Reference Frames

Reference frames are important in order to reason about a point or a rigid body's position, orientation and motion in space. The reference frames used in this thesis are cartesian coordinate systems, consisting of an origin and three orthogonal unit axes, regularly denoted

$$x = [1, 0, 0]^T \quad y = [0, 1, 0]^T \quad z = [0, 0, 1]^T \tag{3.1}$$

For ease of notation, frame(s) will from now on mean reference frame(s), unless otherwise specified. It is common to define an inertial world frame, which is a non-accelerating frame where the Newtonian laws of motion applies. There exist numerous ways to define such an inertial frame, where in this thesis the NED-convention is used, based on [10]. NED stands for *norh-east-down*, and the axes are defined as

- x^I - axis points towards true *North*
- y^I - axis points towards *East*
- z^I - axis points *downwards*, normal to the Earth's surface

following the right-hand-rule. The NED inertial frame is from now denoted $\mathcal{F}_I$. The base frame of the snake robot, denoted $\mathcal{F}_b$, is located at the very back of the snake robot's body. Each link has a reference frame, denoted $\mathcal{F}_l$, attached to its Centre of Mass (CM), where $l = 1, ..., n$. The joints frames are denoted $\bar{\mathcal{F}}_j$, where $j = 1, ..., n-1$, while the end effector frame is denoted $\bar{\mathcal{F}}_{ee}$. An illustration of the snake robot and its frames is shown in Figure 3.2.

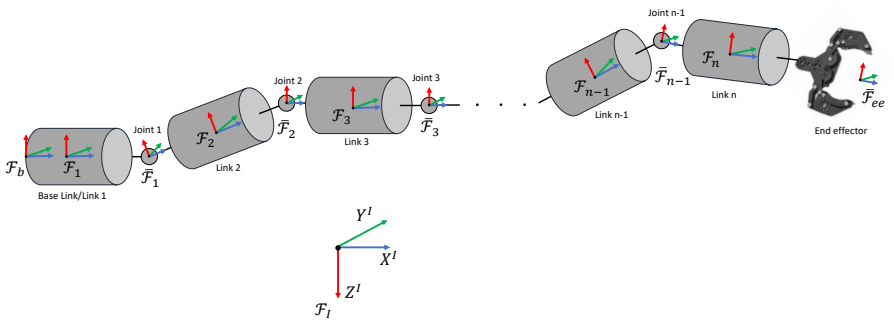


Figure 3.2: The inertial frame and the various frames related to the snake robot. The end effector is illustrated as a gripper, but could in general be any kind of end effector.

3.1.2 Pose and Velocity

Every rigid body could be, at any time instant, described by the position and orientation of its Coordinate Origin (CO) relative to a given frame. Also the rigid body motion, including linear and angular velocity, from time t_k to t_{k+1}, can be relatively described between two frames. These definitions are extensively discussed for motion relating a body-fixed frame and an inertial frame in [10], so only a main recapitulation is presented here.

The pose and velocity of a rigid body wrt to $\mathcal{F}_I$ are denoted

$$\boldsymbol{\eta} = \begin{bmatrix} \boldsymbol{p}_{Ib}^{I} \\ \boldsymbol{\Theta}_{Ib} \end{bmatrix} \in \mathbb{R}^6, \qquad \boldsymbol{\nu}_{Ib}^{b} = \begin{bmatrix} \boldsymbol{v}_{Ib}^{b} \\ \boldsymbol{\omega}_{Ib}^{b} \end{bmatrix} \in \mathbb{R}^6 \tag{3.2}$$

where b denotes a rigid body with body frame $\mathcal{F}_b$ attached to it, defining its CO. In terms of the snake robot the base frame will be the body frame of the whole robot. The pose vector $\boldsymbol{\eta}$ consists of the position vector $\boldsymbol{p}_{Ib}^{I} = \begin{bmatrix} x^I & y^I & z^I \end{bmatrix}^T$ and the orientation vector $\boldsymbol{\Theta}_{Ib} = [\phi \ \theta \ \psi]^T$, which express the position of the body and its orientation in world frame coordinates. The velocity vector $\boldsymbol{\nu}_{Ib}^{b}$ consists of the linear velocity vector $\boldsymbol{v}_{Ib}^{b} = [u \ v \ w]^T$ and the angular velocity vector $\boldsymbol{\omega}_{Ib}^{b} = [p \ q \ r]^T$, which are all expressed in the body frame. Note that the former definition utilize an euler angle parametrization of the orientation, which is singular for $\theta = \pm\frac{\pi}{2}$[rad]. An alternative singular-free parametrization of orientation is possible by utilizing unit quaternions, which is defined as

$$\boldsymbol{q}_b^I = \begin{bmatrix} \eta, \boldsymbol{\epsilon}^T \end{bmatrix}^T, \quad ||\boldsymbol{q}_b^I|| = 1, \quad \eta \in \mathbb{R} \text{ and } \boldsymbol{\epsilon} \in \mathbb{R}^3 \tag{3.3}$$

resulting in the pose and velocity vector

$$\boldsymbol{\eta} = \begin{bmatrix} \boldsymbol{p}_{Ib}^{I} \\ \boldsymbol{q}_b^I \end{bmatrix} \in \mathbb{R}^7, \qquad \boldsymbol{\nu}_{Ib}^{b} = \begin{bmatrix} \boldsymbol{v}_{Ib}^{b} \\ \boldsymbol{\omega}_{Ib}^{b} \end{bmatrix} \in \mathbb{R}^6 \tag{3.4}$$

For ease of notation, positional vectors and velocity vectors can be assumed expressed

in world frame and body frame, respectively, when no superscript is added. In addition, quaternions will always be denoted with sub- **and** superscript, as was done in Equation (3.3), in order to distinguish them from joint angles.

For an articulated robot, such as the snake robot, there are also the need for describing the pose and the velocity of its joints. The joints are rotational about 1 axis and thus only have 1 DOF, resulting in a joint j being represented by an angle $q_j \in \mathbb{R}$ and angular velocity $\dot{q}_j \in \mathbb{R}$. Hence, the overall joint angles and angular velocity vectors are denoted $\boldsymbol{q} = [q_1\ q_2 ... q_{n-1}]^T$ and $\dot{\boldsymbol{q}} = [\dot{q}_1\ \dot{q}_2 ... \dot{q}_{n-1}]^T$, respectively. Augmenting the rigid body pose vector, $\boldsymbol{\eta}$, and velocity vector, $\boldsymbol{\nu}^b_{Ib}$, with the joint angles and angular velocities, respectively, results in a final pose and velocity representation of the snake robot, denoted

$$\boldsymbol{\xi} = \begin{bmatrix} \boldsymbol{\eta} \\ \boldsymbol{q} \end{bmatrix} \in \mathbb{R}^{6+(n-1)} \text{ or } \mathbb{R}^{7+(n-1)}, \qquad \boldsymbol{\zeta} = \begin{bmatrix} \boldsymbol{\nu}^b_{Ib} \\ \dot{\boldsymbol{q}} \end{bmatrix} \in \mathbb{R}^{6+(n-1)} \tag{3.5}$$

depending on whether euler angles or unit quaternions are used to represent the orientation of the base.

3.1.3 Configuration Space

The configuration space, hereafter denoted $\mathcal{C}$-space, is a complete specification of the location of every point on the snake robot [31]. By knowing the pose of the base link and joint angles of the snake robot it is possible to determine the location of any point along the snake robot. The $\mathcal{C}$-space is thus specified by the snake robot's $n + 5$ DOF, which shows that $\boldsymbol{\xi}$ from section 3.1.2 is in fact representing the snake robot's $\mathcal{C}$-space.

3.1.4 State Space

The state of a manipulator, here denoted $\boldsymbol{x}$, is a set of variables that, together with a description of the manipulator's dynamics and actuating inputs, are sufficient to determine any future state of the manipulator [31]. The state space, hereafter denoted $\mathcal{S}$-space is thus the set of all possible states of the system. The snake robot's dynamics

are Newtonian, meaning that it can be described by generalizing newtons second law, $F = ma$. The state of the snake robot can be specified by the $\mathcal{C}$-space together with the linear and angular velocity of the base link and the joint velocities. This results in the $\mathcal{S}$-space dimension of the snake robot being $2(n+5)$, which shows that the state vector in the $\mathcal{S}$-space and its derivative with respect to time are given as

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{\xi} \\ \zeta \end{bmatrix}, \quad \dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\boldsymbol{\xi}} \\ \dot{\zeta} \end{bmatrix} \tag{3.6}$$

3.1.5 Rotation Matrix

Rotation matrices are widely used in the field of robotics to represent a body's orientation wrt a fixed frame or to change the frame in which a vector or a frame is represented [23]. In this thesis the fixed frame will be $\mathcal{F}_I$ and a body could be the snake robot or an obstacle located in the work-space. Rotation matrices consists of 9 elements, denoted

$$\boldsymbol{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \tag{3.7}$$

where each column represents a unit axis. Thus, the matrix could be written more compactly as $\boldsymbol{R} = [\hat{\boldsymbol{x}},\ \hat{\boldsymbol{y}},\ \hat{\boldsymbol{z}}]$. In addition to the fact that the columns are of unit length they must also be orthogonal to each other, which leads to the rotation matrix being subject to 6 constraints, mathematically represented as

$$||\hat{\boldsymbol{x}}|| = 1, \qquad ||\hat{\boldsymbol{y}}|| = 1, \qquad ||\hat{\boldsymbol{z}}|| = 1 \tag{3.8}$$

$$\hat{\boldsymbol{x}} \cdot \hat{\boldsymbol{y}} = 0, \qquad \hat{\boldsymbol{x}} \cdot \hat{\boldsymbol{z}} = 0, \qquad \hat{\boldsymbol{z}} \cdot \hat{\boldsymbol{y}} = 0 \tag{3.9}$$

which may be more compactly represented by the expression

$$\boldsymbol{R}^T \boldsymbol{R} = \boldsymbol{I}_{3\times 3} \tag{3.10}$$

Note also that by the definition that the frames follow the right-hand-rule, resulting in $\det(\boldsymbol{R}) = 1$, the rotation matrix is a part of the special orthogonal group $SO(3)$, a group of matrices satisfying

$$SO(3) := \{\boldsymbol{R} \in \mathbb{R}^{3\times 3} \mid \det(\boldsymbol{R}) = 1, \boldsymbol{R}^{-1} = \boldsymbol{R}^T\}$$

The rotation matrix that describes the orientation of a frame $\mathcal{F}_k$ wrt a frame $\mathcal{F}_i$ is denoted $\boldsymbol{R}_k^i$, where the super- and subscript is understood as $\boldsymbol{R}_{\text{from}}^{\text{to}}$. Thus, a general vector that is described in frame $\mathcal{F}_k$, denoted $\boldsymbol{v}^k$, can be described in $\mathcal{F}_i$ by pre-multiplying $\boldsymbol{R}_k^i$. Mathematically this is formulated as

$$\boldsymbol{v}^i = \boldsymbol{R}_k^i \boldsymbol{v}^k \tag{3.11}$$

To go back to represent $\boldsymbol{v}^i$ in $\mathcal{F}_k$

$$\boldsymbol{R}_i^k \boldsymbol{v}^i = \boldsymbol{v}^k \tag{3.12}$$

where $\boldsymbol{R}_i^k = \left(\boldsymbol{R}_k^i\right)^{-1} = \left(\boldsymbol{R}_k^i\right)^T$. If the origins of the two frames do not coincide, we may also need to express a translational relationship between the two frames, which is done by utilizing a homogeneous transformation matrix.

3.1.6 Homogeneous Transformation Matrix

Now that the reference frames and the rotation matrix have been established, and a definition of the pose and velocity vectors of the snake robot have been introduced, it is necessary to be able to transform vectors expressed in one frame, $\mathcal{F}_k$, to another, $\mathcal{F}_i$, where the origin of the two frames do not coincide. The position and orientation of a rigid body is, as discussed in section 3.1.2, expressed in world frame coordinates, while velocities are expressed in the frame of the body that experience the velocity. To compute the transformation from one frame to another one utilize a homogeneous

transformation matrix, generally denoted

$$\boldsymbol{H}_k^i = \begin{bmatrix} \boldsymbol{R}_k^i & \boldsymbol{t}_{ik}^i \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix} \in SE(3), \tag{3.13}$$

where the special euclidean group $SE(3)$ is defined as

$$SE(3) := \{\boldsymbol{H} \in \mathbb{R}^{4\times 4} \mid \boldsymbol{R} \in SO(3), \boldsymbol{t}_{ik}^i \in \mathbb{R}^3\}$$

The rotation matrix $\boldsymbol{R}_k^i$ is as described in Section 3.1.5 and $\boldsymbol{t}_{ik}^i$ is a translation vector indicating the distance from frame $\mathcal{F}_k$ wrt to $\mathcal{F}_i$, expressed in $\mathcal{F}_i$-coordinates. To describe the opposite transformation, i.e the transformation from i to k, the rotation will be reversed, and the translation goes in the opposite direction and is expressed in k instead of i

$$\boldsymbol{H}_i^k = \begin{bmatrix} (\boldsymbol{R}_k^i)^T & -(\boldsymbol{R}_k^i)^T \boldsymbol{t}_{ik}^i \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_i^k & -\boldsymbol{t}_{ik}^k \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix} \tag{3.14}$$

3.1.7 Angular Velocity and Twists

The rotation matrices and homogeneous transformation matrices will in general be time-varying for a dynamic system, due to the fact that the different frames attached to the snake robot will move relative to each other and the world frame. In order to describe a dynamic system's relative angular and/or linear velocities wrt to various frames we need to find a relationship between the velocities and the time-varying $\boldsymbol{R}$ and $\boldsymbol{H}$.

Angular Velocity

The body-fixed angular velocity, $\boldsymbol{\omega}_{Ib}^b$, which was introduced in Section 3.1.2, and the

spatial angular velocity, $\boldsymbol{\omega}_{Ib}^{I}$, are shown to relate to $\boldsymbol{R}_{b}^{I}$ as [9]

$$\left[\boldsymbol{\omega}_{Ib}^{b}\times\right] = \boldsymbol{R}_{I}^{b}\dot{\boldsymbol{R}}_{b}^{I} \tag{3.15}$$

and

$$\left[\boldsymbol{\omega}_{Ib}^{I}\times\right] = \dot{\boldsymbol{R}}_{b}^{I}\boldsymbol{R}_{I}^{b} \tag{3.16}$$

Here, the $[.\times]$-operator transforms a vector into its skew-symmetric matrix representation, which for a 3-dimensional vector $\boldsymbol{a} = [a_1\ a_2\ a_3]^T$ is computed as

$$[\boldsymbol{a}\times] = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \tag{3.17}$$

Twists

The combined linear and angular velocity vector, $\boldsymbol{\nu}$, which was introduced in Section 3.1.2, is often referred to as a twist. In general, a twist can be re-written in matrix form, as is done in [32]

$$[\boldsymbol{\nu}\wedge] = \begin{bmatrix} [\boldsymbol{\omega}\times] & \boldsymbol{v} \\ \boldsymbol{0}_{1\times 3} & 0 \end{bmatrix} \in \mathbb{R}^{4\times 4}, \tag{3.18}$$

where $[.\wedge]$-operator is used to transform a velocity vector of dimension $\mathbb{R}^6$ into its twist matrix representation. The body-fixed and spatial-fixed twists are defined in the same manner as for the angular velocity case, respectively as

$$\left[\boldsymbol{\nu}_{Ib}^{b}\wedge\right] = \boldsymbol{H}_{I}^{b}\dot{\boldsymbol{H}}_{b}^{I} \tag{3.19}$$

and

$$\left[\boldsymbol{\nu}_{Ib}^{I}\wedge\right] = \dot{\boldsymbol{H}}_{b}^{I}\boldsymbol{H}_{I}^{b} \tag{3.20}$$

To retrieve the velocity vector from the twist matrix there exists the adjoint action of the homogeneous tranformation matrix $\boldsymbol{H}_b^I$, written as $\mathrm{Ad}\left(\boldsymbol{H}_b^I\right) : \mathbb{R}^6 \to \mathbb{R}^6$ and in [30] defined such that

$$\left[\mathrm{Ad}\left(\boldsymbol{H}_b^I\right)\boldsymbol{\nu}_{Ib}^b\wedge\right] = \boldsymbol{H}_b^I\left[\boldsymbol{\nu}_{Ib}^b\wedge\right]\boldsymbol{H}_I^b, \tag{3.21}$$

where it is also stated that for $\boldsymbol{H}_b^I$ the adjoint action explicitely looks like

$$\mathrm{Ad}\left(\boldsymbol{H}_b^I\right) = \begin{bmatrix} \boldsymbol{R}_b^I & [\boldsymbol{t}_{Ib}^I\times]\boldsymbol{R}_b^I \\ \boldsymbol{0}_{3\times3} & \boldsymbol{R}_b^I \end{bmatrix} \in \mathbb{R}^{6\times6} \tag{3.22}$$

and

$$\mathrm{Ad}\left(\boldsymbol{H}_b^I\right)^{-1} = \mathrm{Ad}\left(\boldsymbol{H}_I^b\right) = \begin{bmatrix} \boldsymbol{R}_I^b & -\boldsymbol{R}_I^b[\boldsymbol{t}_{Ib}^I\times] \\ \boldsymbol{0}_{3\times3} & \boldsymbol{R}_I^b \end{bmatrix} \tag{3.23}$$

From eqs. (3.19), (3.20), (3.22) and (3.23) it is seen that the adjoint operator can be used to express velocities in different frames, following the explicit relationship

$$\boldsymbol{\nu}_{Ib}^I = \mathrm{Ad}\left(\boldsymbol{H}_b^I\right)\boldsymbol{\nu}_{Ib}^b \tag{3.24}$$

and

$$\boldsymbol{\nu}_{Ib}^b = \mathrm{Ad}\left(\boldsymbol{H}_I^b\right)\boldsymbol{\nu}_{Ib}^I \tag{3.25}$$

3.2 Kinematics

Kinematics is used to describe the relation of a system's configurations and its velocities, which for the snake robot means its joint angles and pose of the base link. The model will enable motion planning through prediction of its configuration trajectories due to velocity of its base and joint angles in the configuration space. Having introduced the reference frames, pose and velocity of the snake robot and the transformation matrix relating the different frames, this section stitch these concepts together to obtain the kinematic model for the snake robot. For a fixed-base robotic manipulator

this mainly consists of two parts: forward and differential kinematics of its joints. For a floating-base manipulator, which the snake robot is, there is also the need to relate the kinematics of the base expressed in both the body frame and the world frame. This section will thus introduce these three concepts, and is based on [32] and [30]. Note that the forward kinematics presented in Section 3.2.1 is restated from the specialization pre-project.

3.2.1 Forward Kinematics

The concept of forward kinematics, also called direct kinematics, of a robotic system with joints is that by knowing the joint angles vector $\boldsymbol{q}$ one can iterate through all the attached frames of different parts of the robot. This is done by utilizing the transformation matrices $\boldsymbol{H}_k^i$ defined in section 3.1.6. What forward kinematics is mostly used for is to find the pose of the end effector in the world frame [23], but along the way it also gives the other joints pose relative to the world frame.

The base frame of the snake robot, $\mathcal{F}_b$, is related to the world frame, $\mathcal{F}_I$, through a transformation matrix $\boldsymbol{H}_b^I$, which analogously to the definition from section 3.1.6 gives

$$\boldsymbol{H}_b^I = \begin{bmatrix} \boldsymbol{R}_b^I & \boldsymbol{t}_{Ib}^I \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix} \tag{3.26}$$

The base frame and the first joint frame, $\bar{\mathcal{F}}_1$, are related by the transformation matrix $\boldsymbol{H}_{j=1}^b$. The relationship between joint j and its following joint $j+1$ follows the same form through the entire snake robot's body. Thus, the transformation matrix defining the relationship between joint frame $\bar{\mathcal{F}}_j$ and $\bar{\mathcal{F}}_{j+1}$ is expressed as

$$\boldsymbol{H}_{j+1}^j = \begin{bmatrix} \boldsymbol{R}_{j+1}^j & \boldsymbol{t}_{j(j+1)}^j \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix}, \tag{3.27}$$

where $\boldsymbol{t}_{j(j+1)}^j$ is a known constant length from the joint frame $\bar{\mathcal{F}}_j$ to the joint frame

$\bar{\mathcal{F}}_{j+1}$. This length is in fact the length of the link between joint j and $j+1$, which expressed in frame $\bar{\mathcal{F}}_j$ will be aligned with the x-axis of the frame. This means that the translation expressed in $\bar{\mathcal{F}}_j$-coordinates is given as

$$\boldsymbol{t}^j_{j(j+1)} = l_l \boldsymbol{e}, \tag{3.28}$$

where $\boldsymbol{e} = [1\ 0\ 0]^T$ is the basis vector for the x-axis in frame $\bar{\mathcal{F}}_j$ and l_l is the length of the following link l. In terms of a transformation matrix this translation can be written as

$$\boldsymbol{H}^j_{j'} = \begin{bmatrix} \boldsymbol{I}_{3\times 3} & l_l \boldsymbol{e} \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix}, \tag{3.29}$$

where $\bar{\mathcal{F}}_{j'}$ is the intermediate frame after the translation. In addition, joint $j+1$ only have 1 DOF, meaning that the orientation of frame $\bar{\mathcal{F}}_{j+1}$ with respect to the intermediate $\bar{\mathcal{F}}_{j'}$ will only depend on one variable, the angle q_{j+1}. This rotation transformation in terms of a transformation matrix is written

$$\boldsymbol{H}^{j'}_{j+1} = \begin{bmatrix} \boldsymbol{R}^{j'}_{j+1} & \boldsymbol{0}_{3\times 1} \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix}, \tag{3.30}$$

resulting in the transformation between joint frame $\bar{\mathcal{F}}_j$ and $\bar{\mathcal{F}}_{j+1}$ expressed as

$$\boldsymbol{H}^j_{j+1} = \boldsymbol{H}^j_{j'} \boldsymbol{H}^{j'}_{j+1} = \begin{bmatrix} \boldsymbol{I}_{3\times 3} & l_l \boldsymbol{e} \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{R}^{j'}_{j+1} & \boldsymbol{0}_{3\times 1} \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix} \tag{3.31}$$

To express the pose of the end effector related to the world frame the procedure is to multiply all the transformation matrices from the world frame and iterate through the base frame and all the joint frames of the snake robot's body.

$$\boldsymbol{H}^I_e = \boldsymbol{H}^I_b \boldsymbol{H}^b_e = \begin{bmatrix} \boldsymbol{R}^I_b & \boldsymbol{t}^I_{Ib} \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{R}^b_e & \boldsymbol{t}^b_{be} \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix} \tag{3.32}$$

In the special case of the snake robot in *transit* from A to B, with all joints following a reference angle of 0, the transformation matrix $\boldsymbol{H}_e^b$ only contains translations. It can be rewritten as

$$\boldsymbol{H}_e^b = \begin{bmatrix} \boldsymbol{I}_{3\times 3} & \boldsymbol{t}_{be}^b \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix}, \tag{3.33}$$

where $\boldsymbol{t}_{be}^b = [S_l \; 0 \; 0]$ and S_l is the total length of the snake robot. This is a relevant simplification of the forward kinematic if the snake robot is assumed to only move as an AUV.

3.2.2 Differential Kinematics

The snake robot is a dynamic system with capabilities of moving not just its body frame relative to a world frame, but also different parts of its body relative to each other. Thus, the homogeneous transformation matrices that relates the snake robot to the world frame and different links to each other are time varying. Differential kinematics relates the velocities between the different links of the snake robot, utilizing these time varying homogeneous transformation matrices.

Differential Kinematics Between Links

As stated in [32], the body-fixed velocity of the base frame, $\boldsymbol{\nu}_{Ib}^b$, is related to the transformation matrix $\boldsymbol{H}_b^I$ by the twist matrix representation

$$\left[\boldsymbol{\nu}_{Ib}^b \wedge\right] = \left(\boldsymbol{H}_b^I\right)^{-1} \dot{\boldsymbol{H}}_b^I = \begin{bmatrix} \left[\boldsymbol{\omega}_{Ib}^b \times\right] & \boldsymbol{v}_{Ib}^b \\ \boldsymbol{0}_{1\times 3} & 0 \end{bmatrix} \in se(3) \tag{3.34}$$

The same way as for the base link the body-fixed velocity of link l of the snake robot wrt the world frame $\mathcal{F}_I$ can be expressed in $\bar{\mathcal{F}}_j$, the frame attached to joint j, having

link l as its *child* link. This relationship is expressed by

$$\begin{aligned}\left[\boldsymbol{\nu}_{Ij}^{j}\wedge\right] &= \left(\boldsymbol{H}_{j}^{I}\right)^{-1}\dot{\boldsymbol{H}}_{j}^{I} \\ &= \left(\boldsymbol{H}_{j}^{b}\right)^{-1}\left(\boldsymbol{H}_{b}^{I}\right)^{-1}\left(\dot{\boldsymbol{H}}_{b}^{I}\boldsymbol{H}_{j}^{b}+\boldsymbol{H}_{b}^{I}\dot{\boldsymbol{H}}_{j}^{b}\right) \\ &= {\boldsymbol{H}_{j}^{b}}^{-1}\left[\boldsymbol{\nu}_{Ib}^{b}\wedge\right]\boldsymbol{H}_{j}^{b}+\left[\boldsymbol{\nu}_{bj}^{j}\wedge\right]\end{aligned} \tag{3.35}$$

where the composite transformation relationship $\boldsymbol{H}_{j}^{I} = \boldsymbol{H}_{b}^{I}\boldsymbol{H}_{j}^{b}$ is used, and the differentiation result follows from the product rule. The first addend of the last line in Equation (3.35) is a similarity transform of the base link body-fixed twist in order to express the twist in $\bar{\mathcal{F}}_j$. Then, the transformed twist is added to the body-fixed twist of link l wrt $\mathcal{F}_b$, expressed in $\bar{\mathcal{F}}_j$, resulting in the complete body-fixed twist representation of link l wrt the world frame, expressed in $\bar{\mathcal{F}}_j$. The velocity twist of link l from Equation (3.35) in vector form can be found by utilizing the adjoint mapping in Equation (3.23) from Section 3.1.7

$$\boldsymbol{\nu}_{Ij}^{j} = \mathrm{Ad}\left(\boldsymbol{H}_{b}^{j}\right)\boldsymbol{\nu}_{Ib}^{b}+\boldsymbol{\nu}_{bj}^{j} \tag{3.36}$$

and the same relationship can be utilized to find the body-fixed velocity of the end effector

$$\boldsymbol{\nu}_{Ie}^{e} = \mathrm{Ad}\left(\boldsymbol{H}_{b}^{e}\right)\boldsymbol{\nu}_{Ib}^{b}+\boldsymbol{\nu}_{be}^{e} \tag{3.37}$$

Furthermore, the second addend in both eqs. (3.36) and (3.37) can be computed by utilizing the relationships

$$\boldsymbol{\nu}_{bj}^{j} = \mathrm{Ad}\left(\boldsymbol{H}_{b}^{j}\right)\boldsymbol{\nu}_{bj}^{b} = \mathrm{Ad}\left(\boldsymbol{H}_{b}^{j}\right)\boldsymbol{J}_{j}\dot{\boldsymbol{q}} \tag{3.38}$$

and

$$\boldsymbol{\nu}_{be}^{e} = \mathrm{Ad}\left(\boldsymbol{H}_{b}^{e}\right)\boldsymbol{\nu}_{be}^{b} = \mathrm{Ad}\left(\boldsymbol{H}_{b}^{e}\right)\boldsymbol{J}_{e}\dot{\boldsymbol{q}} \tag{3.39}$$

where $\boldsymbol{J}_j$ and $\boldsymbol{J}_e$ are defined as in [32]

$$\boldsymbol{J}_j(\boldsymbol{q}) := \left[\mathrm{Ad}\left(\boldsymbol{H}^b_{j=1}\right)\boldsymbol{X}^1_1, ..., \mathrm{Ad}\left(\boldsymbol{H}^b_j\right)\boldsymbol{X}^j_j, \boldsymbol{0}_{6\times(n-j)}\right] \tag{3.40}$$

$$\boldsymbol{J}_e(\boldsymbol{q}) := \left[\mathrm{Ad}\left(\boldsymbol{H}^b_{j=1}\right)\boldsymbol{X}^1_1, ..., \mathrm{Ad}\left(\boldsymbol{H}^b_e\right)\boldsymbol{X}^e_e\right] \tag{3.41}$$

The $\boldsymbol{X}^j_j$ are the joint twist vectors and given as either $\boldsymbol{X}^j_j = [0,\ 0,\ 0,\ 0,\ 0,\ 1]^T$ or $\boldsymbol{X}^j_j = [0,\ 0,\ 0,\ 0,\ 1,\ 0]^T$ depending on whether the joint rotates about its Z-axis or Y-axis, which was introduced in Section 2.2. In addition, $\boldsymbol{X}^e_e$ in Equation (3.41) depends on the end effector mounted on the snake robot. Eventually, eqs. (3.38) and (3.39) can be inserted into eqs. (3.36) and (3.37), resulting in

$$\boldsymbol{\nu}^j_{Ij} = \boldsymbol{J}_{\boldsymbol{H},j}(\boldsymbol{q})\boldsymbol{\zeta}, \quad \boldsymbol{J}_{\boldsymbol{H},j} := \left[\mathrm{Ad}\left(\boldsymbol{H}^j_b\right),\ \mathrm{Ad}\left(\boldsymbol{H}^j_b\right)\boldsymbol{J}_j\right] \tag{3.42}$$

$$\boldsymbol{\nu}^e_{Ie} = \boldsymbol{J}_{\boldsymbol{H},e}(\boldsymbol{q})\boldsymbol{\zeta}, \quad \boldsymbol{J}_{\boldsymbol{H},e} := \left[\mathrm{Ad}\left(\boldsymbol{H}^e_b\right),\ \mathrm{Ad}\left(\boldsymbol{H}^e_b\right)\boldsymbol{J}_e\right] \tag{3.43}$$

Differential Kinematics Between $\mathcal{F}_b$ and $\mathcal{F}_I$

The relationship of the snake robot's kinematics expressed in the base frame $\mathcal{F}_b$ relative to being expressed in $\mathcal{F}_I$ follows the same approach as was done in [10]. The transformation from $\mathcal{F}_b$-velocity to velocity in $\mathcal{F}_I$ is denoted

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}_k(\boldsymbol{\eta})\boldsymbol{\nu}^b_{Ib} \quad k \in \{\boldsymbol{\Theta}_{Ib}, \boldsymbol{q}^I_b\}, \tag{3.44}$$

where the pose vector $\boldsymbol{\eta}$ is represented by either euler angles or unit quaternions, as described in Section 3.1.2, represented in Equation (3.44) by $\boldsymbol{\Theta}_{Ib}$ and $\boldsymbol{q}^I_b$ respectively.

From [10] it can be seen that by utilizing euler angles as the orientational parametrization the linear and angular velocities are denoted

$$\dot{\boldsymbol{p}}^I_{Ib} = \boldsymbol{R}(\boldsymbol{\Theta}_{Ib})\boldsymbol{v}^b_{Ib} \tag{3.45}$$

and

$$\boldsymbol{\omega}_{Ib}^{b} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \boldsymbol{R}_{x,\phi}^{T} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \boldsymbol{R}_{x,\phi}^{T} \boldsymbol{R}_{y,\theta}^{T} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} := \boldsymbol{T}^{-1}(\boldsymbol{\Theta}_{Ib}) \dot{\boldsymbol{\Theta}}_{Ib} \tag{3.46}$$

where

$$\boldsymbol{T}^{-1}(\boldsymbol{\Theta}_{Ib}) = \begin{bmatrix} 1 & 0 & -\mathrm{s}_{\theta} \\ 0 & \mathrm{c}_{\phi} & \mathrm{c}_{\theta}\mathrm{s}_{\phi} \\ 0 & -\mathrm{s}_{\phi} & \mathrm{c}_{\theta}\mathrm{c}_{\phi} \end{bmatrix}, \quad \boldsymbol{T}(\boldsymbol{\Theta}_{Ib}) = \begin{bmatrix} 1 & \mathrm{s}_{\phi}\mathrm{t}_{\theta} & \mathrm{c}_{\phi}\mathrm{t}_{\theta} \\ 0 & \mathrm{c}_{\phi} & -\mathrm{s}_{\phi} \\ 0 & \mathrm{s}_{\phi}/\mathrm{c}_{\theta} & \mathrm{c}_{\phi}/\mathrm{c}_{\theta} \end{bmatrix} \tag{3.47}$$

and the simplifying notation $\mathrm{c}_a := \cos(a)$, $\mathrm{s}_a := \sin(a)$ and $\mathrm{t}_a := \tan(a)$, with $a \in \{\phi, \theta, \psi\}$, is used. This leads to the transformation being expressed as

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}_{\boldsymbol{\Theta}_{Ib}}(\boldsymbol{\eta}) \boldsymbol{\nu}_{Ib}^{b} = \begin{bmatrix} \boldsymbol{R}(\boldsymbol{\Theta}_{Ib}) & \boldsymbol{0}_{3\times 3} \\ \boldsymbol{0}_{3\times 3} & \boldsymbol{T}(\boldsymbol{\Theta}_{Ib}) \end{bmatrix} \boldsymbol{\nu}_{Ib}^{b} \tag{3.48}$$

The transformation matrix $\boldsymbol{T}(\boldsymbol{\Theta}_{Ib})$ suffers from a singularity at $\theta = \pm\frac{\pi}{2}$[rad], which is not a problem for surface vessels, but for underwater vessels, like the snake robot, the need to avoid this singularity might be present during operation.

The unit quaternion representation is a singularity-free representation for orientation and thus more suited for the snake robot. The linear and angular velocities of the snake robot expressed in $\mathcal{F}_I$ while utilizing quaternions are denoted

$$\dot{\boldsymbol{p}}_{Ib}^{I} = \boldsymbol{R}(\boldsymbol{q}_{b}^{I}) \boldsymbol{v}_{Ib}^{b} \tag{3.49}$$

and

$$\dot{\boldsymbol{q}}_{b}^{I} = \frac{1}{2} \begin{bmatrix} -\boldsymbol{\epsilon}^{T} \\ \eta \boldsymbol{I}_3 + [\boldsymbol{\epsilon}\times] \end{bmatrix} \boldsymbol{\omega}_{Ib}^{b} := \boldsymbol{T}(\boldsymbol{q}_{b}^{I}) \boldsymbol{\omega}_{Ib}^{b} \tag{3.50}$$

where

$$\boldsymbol{R}(\boldsymbol{q}_b^I) = \begin{bmatrix} 1-2(\epsilon_2^2+\epsilon_3^2) & 2(\epsilon_1\epsilon_2-\epsilon_3\eta) & 2(\epsilon_1\epsilon_3-\epsilon_2\eta) \\ 2(\epsilon_1\epsilon_2-\epsilon_3\eta) & 1-2(\epsilon_1^2+\epsilon_3^2) & 2(\epsilon_2\epsilon_3-\epsilon_1\eta) \\ 2(\epsilon_1\epsilon_3-\epsilon_2\eta) & 2(\epsilon_2\epsilon_3-\epsilon_1\eta) & 1-2(\epsilon_2^2+\epsilon_2^2) \end{bmatrix} \tag{3.51}$$

$$\boldsymbol{T}(\boldsymbol{q}_b^I) = \frac{1}{2}\begin{bmatrix} -\epsilon_1 & -\epsilon_2 & -\epsilon_3 \\ \eta & -\epsilon_3 & \epsilon_2 \\ \epsilon_3 & \eta & -\epsilon_1 \\ -\epsilon_2 & \epsilon_1 & \eta \end{bmatrix}, \quad \boldsymbol{T}(\boldsymbol{q}_b^I)^T\boldsymbol{T}(\boldsymbol{q}_b^I) = \frac{1}{4}\boldsymbol{I}_3 \tag{3.52}$$

This leads to the kinematic transformation from $\mathcal{F}_b$ to $\mathcal{F}_I$ and vice versa being expressed as

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}_{\boldsymbol{q}_b^I}(\boldsymbol{\eta})\boldsymbol{\nu}_{Ib}^b = \begin{bmatrix} \boldsymbol{R}(\boldsymbol{q}_b^I) & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{4\times3} & \boldsymbol{T}(\boldsymbol{q}_b^I) \end{bmatrix}\boldsymbol{\nu}_{Ib}^b \tag{3.53}$$

$$\boldsymbol{\nu}_{Ib}^b = \boldsymbol{J}_{\boldsymbol{q}_b^I}^{-1}(\boldsymbol{\eta})\dot{\boldsymbol{\eta}} = \begin{bmatrix} \boldsymbol{R}^T(\boldsymbol{q}_b^I) & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{4\times3} & 4\boldsymbol{T}^T(\boldsymbol{q}_b^I) \end{bmatrix}\dot{\boldsymbol{\eta}} \tag{3.54}$$

By having established a relationship between the velocities expressed in $\mathcal{F}_b$ and $\mathcal{F}_I$ we can now express the transformation between the velocities of the whole $\mathcal{C}$-space as

$$\dot{\boldsymbol{\xi}} = \begin{bmatrix} \boldsymbol{J}_k(\boldsymbol{\eta}) & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_{(n-1)\times(n-1)} \end{bmatrix}\boldsymbol{\zeta}, \tag{3.55}$$

where the dimensions of the zero-matrices on the anti-diagonal in Equation (3.55) depends on whether euler angles or quaternions are utilized for orientational representation.

3.3 Dynamics

The dynamical model of the snake robot considers the forces that acts on the snake robot as it operates in an underwater environment; added mass forces, dissipative drag forces, hydrostatic forces and control forces from its actuators. The model for the snake robot presented in this section is based on [32], [29] and a note on modeling made by PhD-candidate Henrik Schmidt-Didlaukies in conjunction with creation of the simulator used in this thesis.

3.3.1 Equations of Motion

The equations of motion for the snake robot can be written as [32]

$$\boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{\zeta}} + \boldsymbol{C}(\boldsymbol{q},\boldsymbol{\zeta})\boldsymbol{\zeta} + \boldsymbol{D}(\boldsymbol{q},\boldsymbol{\zeta})\boldsymbol{\zeta} + \boldsymbol{g}(\boldsymbol{q},\boldsymbol{R}_I^b) = \boldsymbol{\tau}(\boldsymbol{q}) \tag{3.56}$$

where

- $\boldsymbol{M}(\boldsymbol{q})$ is the system inertia matrix
- $\boldsymbol{C}(\boldsymbol{q},\boldsymbol{\zeta})$ is the coriolis-centripetal matrix
- $\boldsymbol{D}(\boldsymbol{q},\boldsymbol{\zeta})$ is the hydrodynamic damping matrix
- $\boldsymbol{g}(\boldsymbol{q},\boldsymbol{R}_I^b)$ is the matrix of gravitational and buoyancy forces
- $\boldsymbol{\tau}(\boldsymbol{q})$ is a vector of actuator forces

Inertia Matrix

The inertia matrix of the snake robot consists of a rigid body part and a hydrodynamic added mass part, denoted

$$\boldsymbol{M}(\boldsymbol{q}) = \boldsymbol{M}_{RB}(\boldsymbol{q}) + \boldsymbol{M}_A(\boldsymbol{q}), \tag{3.57}$$

of which themselves also consists of contributions from each cylindrical link of the

snake robot. Each link has its own inertia matrix, consisting of a rigid body part and a hydrodynamic added mass part, here given as

$$\boldsymbol{M}_l = \boldsymbol{M}_{RB,l} + \boldsymbol{M}_{A,l} = \begin{bmatrix} \boldsymbol{M}_{l,11} & \boldsymbol{M}_{l,12} \\ \boldsymbol{M}_{l,21} & \boldsymbol{M}_{l,22} \end{bmatrix}, \tag{3.58}$$

where $l = 1, 2, ...n$.

The rigid body part of a link's inertia matrix is calculated from its CO, taken to be the end of the link. In fact, its CO coincides with joint frame $\bar{\mathcal{F}}_j$, where joint j has link l as its *child* link, giving the inertia matrix

$$\boldsymbol{M}_{RB,l} = \begin{bmatrix} m_l \boldsymbol{I}_3 & -m_l[\boldsymbol{r}_{gj}^j \times] \\ m_l[\boldsymbol{r}_{gj}^j \times] & \boldsymbol{I}_{CO} \end{bmatrix}, \tag{3.59}$$

where

- m_l is the mass of link l
- $\boldsymbol{r}_{gj}^j$ is the distance from the link's CO, i.e $\bar{\mathcal{F}}_j$, to its CM, expressed in $\bar{\mathcal{F}}_j$
- $\boldsymbol{I}_{CO}$ is the link's inertia about its CO

It is possible to find the whole snake robot's rigid body mass, $\boldsymbol{M}_{RB}$, by adding all the links rigid body masses, but first they need to be expressed with respect to the base frame $\mathcal{F}_b$. By noting that the kinetic energy of the CO of link l can be written

$$\begin{aligned} \mathcal{K}_l &= \frac{1}{2} \left(\boldsymbol{\nu}_{Ij}^j \right)^T \boldsymbol{M}_{RB,l} \boldsymbol{\nu}_{Ij}^j \\ &= \frac{1}{2} \boldsymbol{\zeta}^T \boldsymbol{J}_{\boldsymbol{H},j}^T(\boldsymbol{q}) \boldsymbol{M}_{RB,l} \boldsymbol{J}_{\boldsymbol{H},j}(\boldsymbol{q}) \boldsymbol{\zeta} \\ &= \frac{1}{2} \boldsymbol{\zeta}^T \boldsymbol{M}_{RB,l}^*(\boldsymbol{q}) \boldsymbol{\zeta} \end{aligned} \tag{3.60}$$

where $\boldsymbol{M}_{RB,l}^*(\boldsymbol{q}) = \boldsymbol{J}_{\boldsymbol{H},j}^T(\boldsymbol{q}) \boldsymbol{M}_{RB,l} \boldsymbol{J}_{\boldsymbol{H},j}(\boldsymbol{q})$ is the rigid body inertia matrix of link

l, expressed with regards to the base frame $\mathcal{F}_b$. From this one can conclude that the rigid body mass of the whole snake robot is

$$\boldsymbol{M}_{RB}(\boldsymbol{q}) = \sum_{l=1}^{n} \boldsymbol{M}^*_{RB,l}(\boldsymbol{q}) \tag{3.61}$$

The hydrodynamic added mass of a single link l, expressed in frame $\bar{\mathcal{F}}_j$, is given in [30] as

$$\boldsymbol{M}_{A,l} = \rho\pi r_l^2 l_l C_a \begin{bmatrix} \alpha_l & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \frac{1}{2}l_l \\ 0 & 0 & 1 & 0 & -\frac{1}{2}l_l & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{2}l_l & 0 & \frac{1}{3}l_l^2 & 0 \\ 0 & \frac{1}{2}l_l & 0 & 0 & 0 & \frac{1}{3}l_l^2 \end{bmatrix} \tag{3.62}$$

where ρ is the density of water, r_l is the radius of the link, l_l is the length of link l and C_a is added mass coefficient for the cross-section. By performing the same transformation to the added mass as for the rigid body mass, the added mass of the whole snake robot can be expressed as

$$\boldsymbol{M}_A(\boldsymbol{q}) = \sum_{l=1}^{n} \boldsymbol{M}^*_{A,l}(\boldsymbol{q}) \tag{3.63}$$

where $\boldsymbol{M}^*_{A,l}(\boldsymbol{q}) = \boldsymbol{J}^T_{\boldsymbol{H},j}(\boldsymbol{q})\boldsymbol{M}_{A,l}\boldsymbol{J}_{\boldsymbol{H},j}(\boldsymbol{q})$. Having transformed both the rigid body mass and added mass of each link, the expression for the whole snake robot inertia matrix is

$$\boldsymbol{M}(\boldsymbol{q}) = \sum_{l=1}^{n} \boldsymbol{M}^*_{RB,l}(\boldsymbol{q}) + \sum_{l=1}^{n} \boldsymbol{M}^*_{A,l}(\boldsymbol{q}) \tag{3.64}$$

Coriolis-Centripetal Matrix

The coriolis-centripetal matrix of the snake robot consists of all the contributions from its links, expressed as

$$\boldsymbol{C}(\boldsymbol{q},\boldsymbol{\zeta}) = \sum_{l=1}^{n} \boldsymbol{C}_l(\boldsymbol{q},\boldsymbol{\zeta}) \tag{3.65}$$

The coriolis-centripetal matrix of a rigid body can always be parametrized such that $\boldsymbol{C}(\boldsymbol{\nu}) = -\boldsymbol{C}^T(\boldsymbol{\nu})$ [10]. For a rigid link l this means that its coriolis-centripetal matrix can be formulated as

$$\boldsymbol{C}_l(\boldsymbol{\nu}_{nj}^j) = -\begin{bmatrix} \boldsymbol{0}_{3\times 3} & \boldsymbol{C}_{l,11} \\ \boldsymbol{C}_{l,21} & \boldsymbol{C}_{l,22} \end{bmatrix}, \tag{3.66}$$

where

$$\begin{aligned} \boldsymbol{C}_{l,11} &= [(\boldsymbol{M}_{l,11}\boldsymbol{v}_{Ij}^j + \boldsymbol{M}_{l,12}\boldsymbol{\omega}_{Ij}^j)\times] \\ \boldsymbol{C}_{l,21} &= [(\boldsymbol{M}_{l,11}\boldsymbol{v}_{Ij}^j + \boldsymbol{M}_{l,12}\boldsymbol{\omega}_{Ij}^j)\times] \\ \boldsymbol{C}_{l,22} &= [(\boldsymbol{M}_{l,21}\boldsymbol{v}_{Ij}^j + \boldsymbol{M}_{l,22}\boldsymbol{\omega}_{Ij}^j)\times] \end{aligned}$$

As was done for the inertia matrix, the coriolis-centripetal matrix $\boldsymbol{C}_l(\boldsymbol{\nu}_{Ij}^j)$ can be expressed with regards to $\mathcal{F}_b$ as

$$\boldsymbol{C}_l^*(\boldsymbol{q},\boldsymbol{\nu}_{Ij}^j) = \boldsymbol{J}_{\boldsymbol{H},j}^T(\boldsymbol{q})\boldsymbol{C}_l(\boldsymbol{\nu}_{Ij}^j)\boldsymbol{J}_{\boldsymbol{H},j}(\boldsymbol{q}) \tag{3.67}$$

From [30] an additional term is added in the coriolis-centripetal matrix, due to the force on the link when the snake robot change its configuration, expressed in $\mathcal{F}_b$ as

$$\boldsymbol{C}_l(\boldsymbol{q}) = \boldsymbol{J}_{\boldsymbol{H},j}^T(\boldsymbol{q})\boldsymbol{M}_l\dot{\boldsymbol{J}}_{\boldsymbol{H},j}(\boldsymbol{q},\dot{\boldsymbol{q}}) \tag{3.68}$$

Thus, the coriolis-centripetal matrix of the whole snake robot, expressed in $\mathcal{F}_b$, is computed as

$$\boldsymbol{C}(\boldsymbol{q},\boldsymbol{\zeta}) = \sum_{l=1}^{n} \boldsymbol{C}_l(\boldsymbol{q}) + \boldsymbol{C}_l^*(\boldsymbol{q},\boldsymbol{\nu}_{nj}^j) \tag{3.69}$$

Damping Matrix

The viscosity of the fluid displaced by the snake robot in motion also causes the presence of dissipative drag forces on the body [1]. The total drag force of the snake robot consists of contributions from each of its links. The drag force of a link l consists of both a linear and a nonlinear term, resulting in the damping matrix of a link l being expressed as

$$\boldsymbol{D}_l(\boldsymbol{\nu}_{Ij}^j) = \boldsymbol{D}_{L,l} + \boldsymbol{D}_{N,l}(\boldsymbol{\nu}_{Ij}^j), \tag{3.70}$$

where the linear part is given in link frame $\mathcal{F}_l$ as [30]

$$\boldsymbol{D}_{L,l} = \rho\pi r_l l_l C_{d,L} v_{\text{ref}} \begin{bmatrix} \beta_l & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \frac{1}{2}l_l \\ 0 & 0 & 1 & 0 & -\frac{1}{2}l_l & 0 \\ 0 & 0 & 0 & \gamma_l r_l^2 & 0 & 0 \\ 0 & 0 & -\frac{1}{2}l_l & 0 & \frac{1}{3}l_l^2 & 0 \\ 0 & \frac{1}{2}l_l & 0 & 0 & 0 & \frac{1}{3}l_l^2 \end{bmatrix}, \tag{3.71}$$

with ρ, r_l and l_l as described in Equation (3.62). Additionally, $C_{d,L}$ is the linear cross-section drag coefficient, γ_l and β_l are the linear drag parameters in roll and surge, respectively. The nonlinear part of the drag is in general difficult to model accurately due to it being subject to highly complex hydrodynamic effects, which are out of the scope of this thesis. In addition, for low speeds the linear drag contribution dominates the nonlinear part, which will be the case for the snake robot in this thesis.

To find the total amount of drag force acting on the snake robot the drag forces acting on each link is expressed with regards to $\mathcal{F}_b$ and added together, giving the total damping matrix as

$$\boldsymbol{D}(\boldsymbol{q}, \boldsymbol{\zeta}) = \sum_{l=1}^{n} \boldsymbol{D}_l^*(\boldsymbol{q}, \boldsymbol{\zeta}), \tag{3.72}$$

where $\boldsymbol{D}_l^*(\boldsymbol{q}, \boldsymbol{\zeta}) = \boldsymbol{J}_{\boldsymbol{H},j}^T(\boldsymbol{q})\boldsymbol{D}_l(\boldsymbol{\nu}_{Ij}^j)\boldsymbol{J}_{\boldsymbol{H},j}(\boldsymbol{q})$.

Gravitational and Buoyancy Force Vector

The snake robot is subject to both gravitational and buoyancy forces, which acts through the snake robot's CM and Centre of Buoyancy (CB), respectively. The gravitational force and buoyancy force are in general denoted [10]

$$W = mg, \quad B = \rho g \nabla \tag{3.73}$$

where g is the gravitational acceleration constant, m is mass and ∇ is the volume of fluid displaced by the vessel. These two forces works in the vertical plane when expressed in the inertial world frame $\mathcal{F}_I$, with opposite direction

$$\boldsymbol{f}_g^n = [0\ 0\ W]^T, \quad \boldsymbol{f}_b^n = -[0\ 0\ B]^T \tag{3.74}$$

where the subscripts b and g here refers to the CB and CM of the vessel. Each link of the snake robot is subject to a gravitational and buoyancy force, and the forces on link l can be expressed in its CO according to

$$\begin{aligned} \boldsymbol{g}_l(\boldsymbol{q}, \boldsymbol{R}_I^j) &= -\begin{bmatrix} \boldsymbol{f}_g^j + \boldsymbol{f}_b^j \\ \boldsymbol{r}_{gj}^j \times \boldsymbol{f}_g^j + \boldsymbol{r}_{bj}^j \times \boldsymbol{f}_b^j \end{bmatrix} \\ &= -\begin{bmatrix} \boldsymbol{R}_I^j(\boldsymbol{f}_g^I + \boldsymbol{f}_b^I) \\ \boldsymbol{r}_{gj}^j \times \boldsymbol{R}_I^j \boldsymbol{f}_g^I + \boldsymbol{r}_{bj}^j \times \boldsymbol{R}_I^j \boldsymbol{f}_b^I \end{bmatrix} \end{aligned} \tag{3.75}$$

where the minus sign on the RHS is due to the fact that the forces are moved to the LHS in the equations of motion, Equation (3.56). The second row of the vector are the experienced moments about the CO when the reference frame does not coincide with CM or CB.

To obtain the hydrostatic forces on the whole snake robot, expressed with regards to the base frame $\mathcal{F}_b$, Equation (3.75) must be computed for every link, transformed

to the base frame and added together. This leads to the gravitational and buoyancy vector of the whole snake robot being expressed as

$$\boldsymbol{g}(\boldsymbol{q}, \boldsymbol{R}_I^b) = \sum_{l=1}^{n} \boldsymbol{J}_{\boldsymbol{H},j}^T(\boldsymbol{q}) \boldsymbol{g}_l(\boldsymbol{q}, \boldsymbol{R}_I^b) \tag{3.76}$$

Actuator Forces

The generalized control forces $\boldsymbol{\tau}(\boldsymbol{q})$ are calculated in order to move the snake robot in the world frame and change its configuration. It consists of a Thrust Configuration Matrix (TCM) and force input vector, expressed as in [32]

$$\boldsymbol{\tau}(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{\tau}_b(\boldsymbol{q}) \\ \boldsymbol{\tau}_q(\boldsymbol{q}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{B}(\boldsymbol{q}) & \boldsymbol{0}_{6\times(n-1)} \\ \boldsymbol{0}_{(n-1)\times m} & \boldsymbol{I}_{(n-1)\times(n-1)} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_{\text{thr}} \\ \boldsymbol{u}_q \end{bmatrix} \tag{3.77}$$

where $\boldsymbol{B}(\boldsymbol{q}) \in \mathbb{R}^{6\times m}$ is the TCM and m is the number of thrusters. The inputs $\boldsymbol{u}_{\text{thr}}$ and $\boldsymbol{u}_q$ are the thruster forces and joint torques, respectively. As can be seen from Equation (3.77) the TCM depends on the snake robot configuration. How the generalized control forces will be distributed over the different actuators of the system is a control allocation task, which will be introduced in Section 6.2.

Chapter 4

Path Planning for Collision Avoidance

The snake robot is operating in a subset of the 3D-world, also known as its work-space. How it moves in the work-space is decided by the *guidance* and *control* systems of the robot. The path planner is an integrated part of the *guidance system*, shown in Figure 4.1. The path planner plays an important part in deciding where the snake robot should move in order to reach its desired target in a collision-free manner. There exist mainly two types of path planners; global and local. Global planners require prior information about the environment, i.e. the positions of the obstacles in the work-space, while local planners in general do not [27]. When planning a path there are many different objectives to consider; shortest path, both regarding time spent and/or distance travelled, most energy efficient path and the safest path. This chapter will cover global path planning, using the Artificial Potential Field (APF) and Rapidly Exploring Random Tree (RRT) method, focusing on the objective of finding a feasible collision-free path for the snake robot, leading to the desired target. In addition, the

ability to find navigable short paths is preferable, as this can lower energy consumption and save time. First, Section 4.1 introduces the environment and its obstacles. Secondly, Section 4.2 presents the APF method, where sections 4.2.1 and 4.2.2 are restated from the specialization pre-project since the APFs applications were studied there and will be further utilized in this thesis. Lastly, Section 4.3 introduces the RRT path planning method.

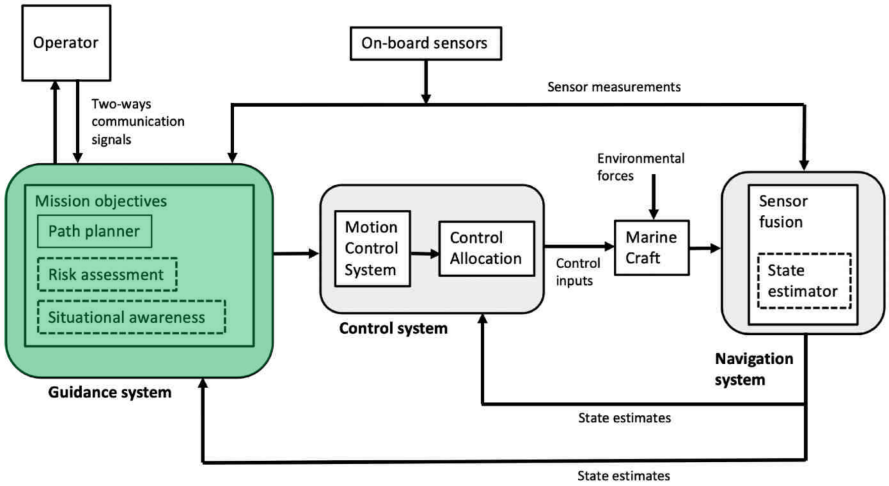


Figure 4.1: Guidance system in the system architecture. Courtesy of [10].

4.1 Environment

The environment the snake robot operates in, hereafter referred to as the $\mathcal{W}$-space, is defined as the total volume swept out by the end effector as the manipulator executes all possible motions [31]. In theory, this means that the $\mathcal{W}$-space of the snake robot, which is a floating-base manipulator, is the whole 3D-space. In practice the snake robot is not free to move wherever it wants because the domain will often be restricted by the presence of obstacles, cluttering the environment. For the snake robot's case these obstacles might be industrial underwater installations, ship wrecks, rocks or

even coral reefs. This section presents the cluttered environment used in this thesis, of which the APF and RRT* path planning algorithms will later be utilized to solve path planning wrt the collision avoidance task.

4.1.1 Cluttered Environment

The $\mathcal{W}$-space might be cluttered by obstacles in various ways. These obstacles may in general have any arbitrary shape and size, but in this thesis all obstacles are assumed to be spheres, due to their simple form and convenience regarding approximating more complex structures. Thus, each obstacle, denoted $\mathcal{O}_i$, will have positional coordinates, denoted $\boldsymbol{p}_{\mathcal{O}_i}$, and a radius, denoted $r_{\mathcal{O}_i}$. An example of a typical environment consisting of spherical obstacles is illustrated in Figure 4.2. Even though it is not realistic that obstacles will be completely spherical in real life, it is practical to work with spheres. Any obstacle could be approximated by using a number of spheres, which is discussed in [23]. Depending on how accurate one need to approximate an obstacle and how fast a collision avoidance algorithm need to be, the amount of spheres and their size vary. For accurate approximations one need to use many small spheres, but at the expense of being computationally heavier than less accurate approximations using less and bigger spheres. An example of different approximations are visualized in Figure 4.4. This approximation method is utilized in [15] for the planar case, where each 2D rectangular obstacle are approximated by a circle enclosing the whole rectangle. Spherical approximations of obstacles will not be explicitely done in this thesis, but one could imagine that inside each sphere there exists an obstacle of a different arbitrary shape that is approximated by the sphere. Additionally, to prevent the snake robot from getting too close to each obstacle, a safe radius can be added. This is to make sure that the planned path gives the snake robot some safety margins for maneuvering and stay clear of the obstacles when following the path. An illustration of the actual obstacle and the obstacle accounted for by the path planner is shown in Figure 4.3. The larger the safe radius, the more conservative the path planner will be. It might lead to paths that are not as optimal in terms of distance, due to the planner excluding more space for the snake robot to move through, but when a path is found it will be safer in terms

of potential collisions.

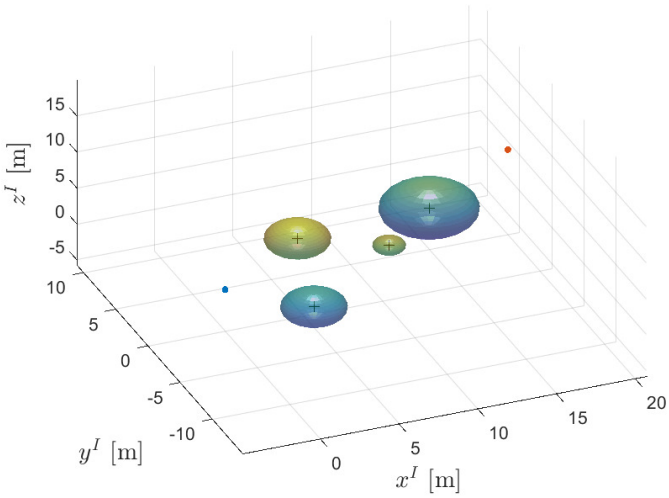


Figure 4.2: Environment consisting of 4 spherical obstacles of different size, the target position (red) and the starting position (blue).

4.1.2 Configuration Space Obstacle

The snake robot itself occupies a subset of the $\mathcal{W}$-space, denoted $\mathcal{A}(\boldsymbol{\xi})$. Here $\mathcal{A}$ denotes the snake robot and $\boldsymbol{\xi}$ its configuration. The configuration space obstacle, hereafter denoted $\mathcal{CO}$-space, is defined as the set of configurations for which the robot collides [31], mathematically denoted

$$\mathcal{CO} = \{\boldsymbol{\xi} \in \mathcal{C} \mid \mathcal{A}(\boldsymbol{\xi}) \cap \mathcal{O} \neq \emptyset\} \tag{4.1}$$

where $\mathcal{O}$ is the set of all obstacles in the $\mathcal{W}$-space. By combining the $\mathcal{CO}$-space and the $\mathcal{C}$-space from Section 3.1.3 we can define the free configuration space as the complement of the $\mathcal{CO}$-space

$$\mathcal{C}_{\text{free}} = \mathcal{C} \backslash \{\mathcal{CO}\} \tag{4.2}$$

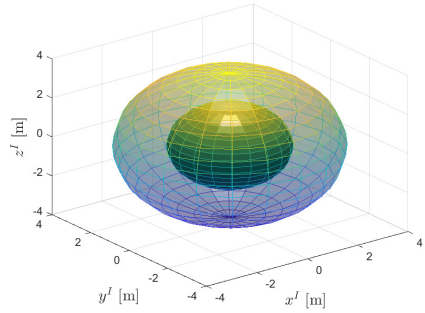


Figure 4.3: Illustration of the safe radius used for path planning. The inner solid sphere is the actual obstacle and the outer transparent is the sphere seen by the path planner.

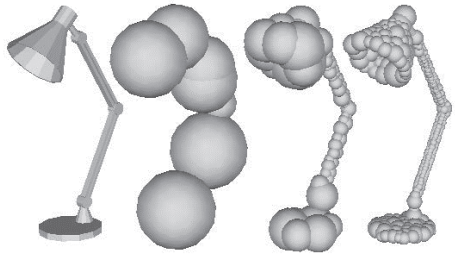


Figure 4.4: A lamp represented by spheres. The approximation improves as the number of spheres used to represent the lamp increases. Courtesy of [12].

4.2 Artificial Potential Fields

The method of creating an APF for collision-free path planning mainly consists of two parts; the obstacles that are present in the $\mathcal{W}$-space of the snake robot and the desired target in the same domain. The obstacles and the target are used to create a potential field, where the obstacles have repelling potential and the target has attractive potential. This section first presents the choice of potential functions in detail. Secondly, an explicit potential function for the task at hand will be derived, based on [16], [2] and [7]. Lastly, a more robust path planning variant of the APF, called Randomized Potential

Field (RPF), capable of escaping local minima, is presented.

4.2.1 Potential functions

The potential function is a differentiable real-valued function $U : \mathbb{R}^m \to \mathbb{R}$ [7]. In this project the $\mathcal{W}$-space of the snake robot is 3-dimensional, i.e $m = 3$. Essentially, the potential field is an objective function which one seeks to minimize in order to find a collision-free path between two chosen points in the $\mathcal{W}$-space domain. The field is a linear combination of several potential functions, both attractive and repelling ones. An attractive potential function, denoted U_a, is in this context related to the target, the desired end goal the snake robot wants to reach, while a repelling potential function, denoted U_r, is related to the obstacles. There only exists one target at a time, denoted $\boldsymbol{p}_t$, but there might be several obstacles in the domain. The total number of obstacles is denoted N_o, and the i^{th} obstacle is denoted $\mathcal{O}_i$, where $i = 1, ..., N_o$. Thus, there only exists one attractive potential function, but there exist o repelling potential functions. The potential field is defined the same way as in [15] and [2]

$$U = U_a + \sum_{i=1}^{N_o} U_r^i \tag{4.3}$$

Attractive Potential

The attractive potential, U_a, should be designed such that it is differentiable and monotonically decreasing as it approaches $\boldsymbol{p}_t$, meaning that the function value decreases as the distance to the target decreases. As stated in [7], the simplest choice is the scaled *conic potential*, formulated as

$$U_a(\boldsymbol{p}) = K_a d(\boldsymbol{p}, \boldsymbol{p}_t) \tag{4.4}$$

where the $d(\cdot)$ indicates the euclidean distance function, $\boldsymbol{p}$ is the position of a point in the $\mathcal{W}$-space and K_a is a scaling parameter to change the steepness of the potential

function. The *conic potential* has the gradient function

$$\nabla U_a(\boldsymbol{p}) = \frac{K_a}{d(\boldsymbol{p}, \boldsymbol{p}_t)}(\boldsymbol{p} - \boldsymbol{p}_t) \tag{4.5}$$

which suffers from numerical problems when approaching the target due to the fact that

$$\lim_{d(\boldsymbol{p}, \boldsymbol{p}_t) \to 0} \nabla U_a(\boldsymbol{p}) \to \infty \tag{4.6}$$

The design of another, still very simplistic, potential function deals with this problem, which is the *paraboloide potential*

$$U_a(\boldsymbol{p}) = \frac{1}{2} K_a d^2(\boldsymbol{p}, \boldsymbol{p}_t) \tag{4.7}$$

that has the gradient function

$$\nabla U_a(\boldsymbol{p}) = K_a(\boldsymbol{p} - \boldsymbol{p}_t) \tag{4.8}$$

The benefits from this potential, as stated in [7], is the fact that its gradient is bigger further away from the target. This makes the snake robot approach the target faster when it is far away, but slows down when it is closer to the target, which is good for preventing overshoot when planning a path. The downside of utilizing the *paraboloide function* is that if the snake robot starts too far away from the desired target, the gradient $\nabla U_a(p)$ might get very big, due to the fact that it is unbounded. Therefore one could combine the two variants of potential functions to utilize their strengths. This combined potential function also needs to be differentiable and monotonically decreasing when approaching the target, which is satisfied for

$$U_a(\boldsymbol{p}) = \begin{cases} \frac{1}{2} K_a d^2(\boldsymbol{p}, \boldsymbol{p}_t), & d(\boldsymbol{p}, \boldsymbol{p}_t) \leq d_t^* \\ d_t^* K_a d(\boldsymbol{p}, \boldsymbol{p}_t) - \frac{1}{2} K_a (d_t^*)^2, & d(\boldsymbol{p}, \boldsymbol{p}_t) > d_t^* \end{cases} \tag{4.9}$$

having the gradient function

$$\nabla U_a(\boldsymbol{p}) = \begin{cases} K_a(\boldsymbol{p} - \boldsymbol{p}_t), & d(\boldsymbol{p}, \boldsymbol{p}_t) \leq d_t^* \\ \frac{d_t^* K_a}{d(\boldsymbol{p}, \boldsymbol{p}_t)}(\boldsymbol{p} - \boldsymbol{p}_t), & d(\boldsymbol{p}, \boldsymbol{p}_t) > d_t^* \end{cases} \tag{4.10}$$

which is well defined at the boundary d_t^*, where the two gradient functions have the same value.

Repulsive Potential

The repulsive potential from an obstacle, $\mathcal{O}_i$, should be a non-negative continuous and differentiable function that tends to infinity as the snake robot approaches the obstacle's surface, as stated in [16]. The influence from the obstacle should be smaller the farther the distance is between the snake robot and the obstacle. At a certain distance the influence from the obstacle should also be non-influencing the snake robot due to the fact that it is located in a safe distance relative to the obstacle. These attributes are satisfied for

$$U_r^o(\boldsymbol{p}) = \begin{cases} \frac{1}{2} K_r \left(\frac{1}{d_o(\boldsymbol{p})} - \frac{1}{Q_o^*} \right)^2, & d_o(\boldsymbol{p}) \leq Q_o^* \\ 0, & d_o(\boldsymbol{p}) > Q_o^* \end{cases} \tag{4.11}$$

where $d_o(\cdot)$ is the shortest distance from the robot to the obstacle's surface, K_r is a scaling parameter and Q_o^* is a smoothing parameter that defines how the obstacle's potential shall fade away as the distance to it increases. Or in other words, it alters the size of the obstacle's influence domain. The gradient function is formulated as

$$\nabla U_r^o(\boldsymbol{p}) = \begin{cases} K_r \left(\frac{1}{Q_o^*} - \frac{1}{d_o(\boldsymbol{p})} \right) \frac{1}{d_o(\boldsymbol{p})^2} \nabla d_o(\boldsymbol{p}), & d_o(\boldsymbol{p}) \leq Q_o^* \\ 0, & d_o(\boldsymbol{p}) > Q_o^* \end{cases} \tag{4.12}$$

where $\nabla d_o(\boldsymbol{p}) = \frac{\boldsymbol{p} - \boldsymbol{c}}{d_o(\boldsymbol{p}, \boldsymbol{c})}$ and $\boldsymbol{c}$ is the closest point on the surface of obstacle $\mathcal{O}_i$ to the snake robot.

4.2.2 Gradient Descent for Path Generation

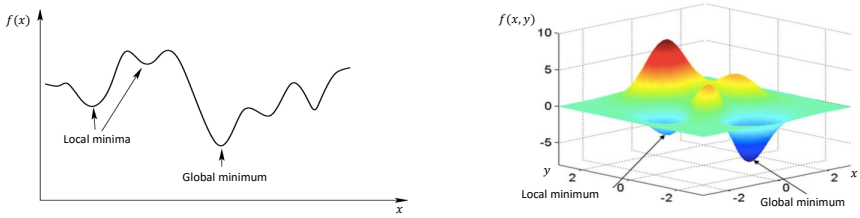
The path that essentially leads the snake robot from its current position to the target position is generated by utilizing the steepest descent algorithm, also known as *gradient descent*, to find the minimum of the objective function. The procedure is simple; choose a starting point and compute the gradient at that point. Then move in the opposite direction of the gradient to get to the next point and repeat the process. As a result one will iterate from one point to the other, always lowering the value of the potential field, until a minimum is found. The algorithm for *gradient descent* is found in [7], but is restated in Appendix A.1, where ϵ and α are hyperparameters and thus chosen dependently on the specific task. The parameter ϵ is chosen in order for the algorithm to terminate when the gradient is sufficiently small, while α, also called the step size, decides how fast the algorithm converge. If the value of α is relatively low the algorithm might get very slow, but if it is too big it might make the whole algorithm numerically unstable. Thus, this will be a tuning task, which will be looked at in Section 7.1. Ball park start for α could be 0.1, but there exists methods for choosing α, which are given in [26].

Local Minima Problem

The *gradient descent* algorithm suffers from the local minima problem, meaning that even though the algorithm converge to a minimum it is not guaranteed to be the global minimum. If there are several local minima, then which of those it converges to is decided by the initial condition, i.e $\boldsymbol{p}_{\text{start}}$. The local minima problem is illustrated for an objective function related to the 1D and 2D case respectively in Figure 4.5, where the robot can move on a line and a plane. In addition, figs. 4.6 and 4.7 illustrates where a local minimum could typically occur, and how the resulting path is not connected to the target. An objective function related to higher dimensional cases is difficult to visualize, but follows the same principles.

When *gradient descent* is able to converge to the global minimum of the objective function, a path from the starting point to the target is created by following/iterate through the points, and it will be collision-free. An example of such a path is illustrated

in Figure 4.8.



(a) (b)

Figure 4.5: Illustration of local minima and global minimum of objective functions in (a) 1D and (b) 2D.

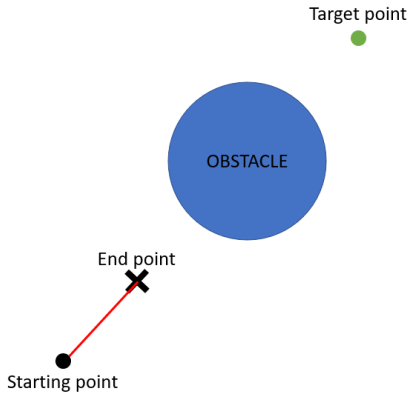


Figure 4.6: Illustration of a planar path trapped by a local minimum due to an obstacle located right in front of the desired path to the target point.

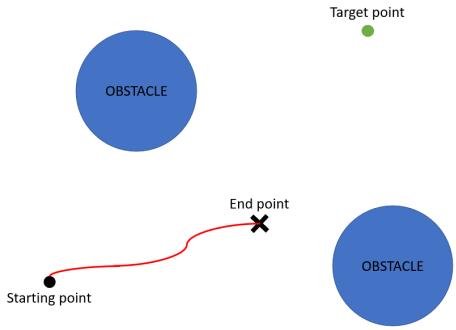


Figure 4.7: Illustration of a planar path that stops in a local minimum between two obstacles located in front of the target.

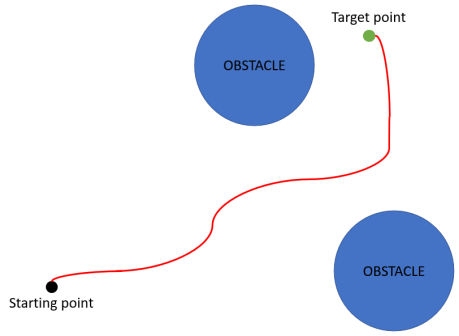


Figure 4.8: Illustration of a collision-free path in a 2D plane that succeeds to reach the target.

4.2.3 Randomized Potential Field

One way to cope with the local minima problem of the APF is described both in [31] and [20], and a solution is to introduce randomness. If the planned path gets stuck in a local minimum one could activate a *random walk* out of it and continue to search for the target with *gradient descent* from where the *random walk* stopped. In order to utilize this *random walk* method we need to be able to detect a local minimum when the path gets trapped in one and define the *random walk* such that it is able to escape the local minimum and eventually converge to the target.

Detect Local Minimum

For the case of detecting whether or not the path has been trapped by a local minimum we can use simple heuristics. When the path changes minimally over a sequence of iterations there is a high probability that it has been trapped. To detect this it is sufficient to choose a subset of the latest samples of the path to analyze how much it has changed. Pick the oldest sample in the subset as a reference and compute the euclidean distance between this reference and the rest of the samples in the subset. Choose a threshold value, $\epsilon_{\text{trapped}}$, that defines whether or not the path is trapped. If all the distances are below the threshold it's quite likely that that the path finds itself in a local minimum.

Random Walk

The *random walk* starts from the newest sample added to the path. The path is extended from this sample by a chosen number of random samples, N_{rw}, where each DOF in every sample is restricted to move a given amount $\pm\delta_{\text{rw}}$, where the sign is chosen based on the outcome of a fair coin toss [20]. How many samples that are needed and the size of δ_{rw} in order to successfully escape the local minimum depends on both the system and how cluttered the $\mathcal{W}$-space is. A pseudo-code for the *random walk* mode is provided in Appendix A.2. An example of how the *random walk* can escape local minima is visualized in Figure 4.9. The *random walk* might lead to paths that are not well suited for path following because of its random pattern, but methods for post-processing of the path which filter out a subset of the path as waypoints

can be used to retrieve a collision-free and feasible path for path following. Such post-processing methods will be further discussed in Section 5.2. It's important to mention that even though the *random walk* method can be used to escape local minima it's not guaranteed to, due to the fact that all the random steps must also be checked for possible collisions. In particular, if the environment is extremely cluttered around the local minimum it might be especially tricky for the *random walk* to navigate out of it.

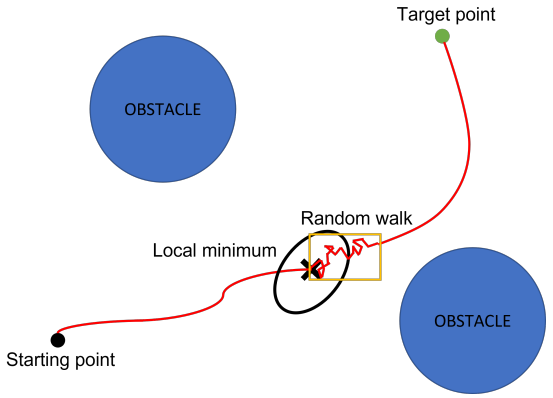


Figure 4.9: Illustration of *random walk* utilized to escape local minima in a 2D $\mathcal{W}$-space.

4.3 Rapidly Exploring Random Trees

The RRT algorithm is a sampling-based method for collision-free path planning. Sampling-based methods generally give up on the resolution-optimal solutions of a grid search in exchange for the ability to find satisfying solutions quickly in high-dimensional spaces, such as the $\mathcal{C}$-space or the $\mathcal{S}$-space [23]. The samples are drawn randomly from the selected space, here in general referred to as the *query*-space, and will eventually, while accounting for obstacles in the domain and motion constraints of the system, approximate the free space in which the snake robot can move. This section will give an introduction to the concepts of the RRT algorithm and the more optimal

version called RRT*, originating from the work done in [19]. Further improvements and analyses are discussed in [13].

4.3.1 RRT

The RRT algorithm results in a tree, denoted $\mathcal{T}$, that spans a subset of the *query*-space's free space. For ease of notation, the *query*-space is denoted $\mathcal{Q}$-space, where $\mathcal{Q} \in \{\mathcal{C},\ \mathcal{S}\}$. Thus, the tree $\mathcal{T}$ can be expressed as

$$\mathcal{T} \subset \mathcal{Q}_{\text{free}} \tag{4.13}$$

The algorithm starts exploring the domain from a given initial configuration, here denoted $\boldsymbol{\xi}_{\text{init}}$, looking for a path leading to the goal configuration, denoted $\boldsymbol{\xi}_{\text{goal}}$. Every sample, denoted $\boldsymbol{\xi}_{\text{rand}}$, is drawn randomly from the $\mathcal{Q}$-space, following a probability distribution of some sort. This could be the uniform distribution, but other distributions having a bias towards $\boldsymbol{\xi}_{\text{goal}}$ are also used [33]. Depending on the distribution utilized, the algorithm will possess different behaviour towards the target. The initial configuration is called the root node of $\mathcal{T}$, and the subsequently added samples are nodes of various branches of $\mathcal{T}$. The newest generated sample is connected to the nearest node in $\mathcal{T}$, where a metric, ρ_{RRT}, is used to define the nearest node. This metric depends on the units of the system's configuration space, where the configuration of the snake robot for instance consists of both units in radians/degrees and metres. Which configuration is the closest depends on the system's ability to move from one configuration to another. Various metrics can be found in [20]. In addition, when $\boldsymbol{\xi}_{\text{rand}}$ is generated and the nearest neighbour is selected we have to make sure that the path from the nearest node to $\boldsymbol{\xi}_{\text{rand}}$ is in fact feasible for the snake robot to follow. For collision avoidance purposes the algorithm checks that every sample is in fact inside of $\mathcal{Q}_{\text{free}}$, rejecting the samples that violates this condition. Additionally, a constraint, or steering method, on how far away the added sample can be from the nearest neighbour is also introduced, denoted ϵ_{upper}. This constraint ensures that the growth of the tree does not move too far in a potentially wrong direction related to the goal. How ϵ_{upper} bounds each branch's distance is visualized in Figure 4.10, and the new sample added

to the tree follows from

$$\xi_{\text{new}} = \begin{cases} \xi_{\text{near}} + \epsilon_{\text{upper}} \frac{\xi_{\text{rand}} - \xi_{\text{near}}}{||\xi_{\text{rand}} - \xi_{\text{near}}||}, & ||\xi_{\text{rand}} - \xi_{\text{near}}|| > \epsilon_{\text{upper}} \\ \xi_{\text{rand}}, & \text{otherwise} \end{cases} \tag{4.14}$$

The main accomplishment of the algorithm, regardless of the choice of distribution, is that the resulting tree $\mathcal{T}$ expands from the root node, covering a greater part of $\mathcal{Q}_{\text{free}}$ for every sample that is added. The number of samples, N_s, to draw from the domain decides how accurately $\mathcal{T}$ will approximate $\mathcal{Q}_{\text{free}}$ and whether it converges to ξ_{goal}. The smaller the allowed distance from on node to another, the higher the number of samples must be to ensure convergence to the target. An example outcome of the algorithm is visualized in Figure 4.11.

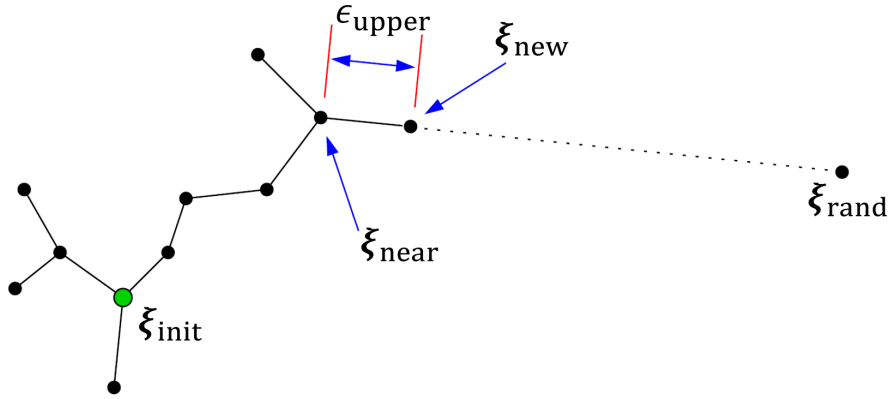


Figure 4.10: The process of randomly draw a new sample and add it to the tree, following the constraint set by ϵ_{upper}. Courtesy of [18], with small notational changes made by the author.

4.3.2 RRT*

Although the plain RRT algorithm converges to a path leading to ξ_{goal} it's almost surely a non-optimal path [13]. The RRT* version of the RRT algorithm is essentially built to avoid this problem and obtain optimality. To obtain optimality a cost function

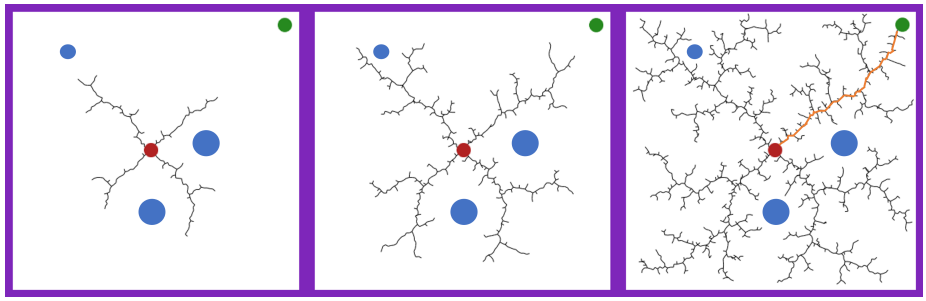


Figure 4.11: The resulting tree $\mathcal{T}$ projected in the 2D space, showing how the tree expands from the initial configuration (red dot) to the goal (green dot), while avoiding the obstacles (blue dots). The resulting path is visualized in orange. Courtesy of [19], with added obstacles and path by the author.

is introduced, which one seeks to minimize. As for all optimization problems, the cost function depends on the objective of the task. This could for instance be related to shortest distance travelled, shortest time used from $\boldsymbol{\xi}_{\text{init}}$ to $\boldsymbol{\xi}_{\text{goal}}$, least amount of fuel consumed along the path etc. In this thesis a shortest feasible path in the 3-dimensional $\mathcal{W}$-space will be searched for, and thus the cost function seeks to minimize the distance travelled. A pseudo-code of the RRT* algorithm is provided in Appendix A.4, based on the one from [13].

Chapter 5

Path Following

Having planned a desired path or trajectory that the snake robot shall follow, the task is to steer the snake robot onto and follow it in an acceptable manner. For a path this means that the snake robot must be able to move onto the path, in general preferred without too large overshoot, and stay on the path. For a trajectory there is also the essence of time, the snake robot needs to be at a specific point at a specific time as well. This chapter will focus on the path following part, which relates to the system architecture as shown in Figure 5.1. First, in Section 5.1 a Waypoint Guidance (WPG) method is introduced, where piecewise linear paths are utilized for path following purposes. Section 5.2 subsequently presents two methods for dividing paths into a sparser number of waypoints. Lastly, Section 5.3 presents a LOS guidance law, as described in [10] and applied in [15]. In addition, an extension to guidance in 3D is presented, where a Waypoint Line-of-Sight (WLOS) guidance law will be introduced to make the snake robot follow a desired 3D path, based on the work done in [21] and [4]. Note that parts of Section 5.1 and sections 5.3.1 and 5.3.2 are reused from the specialization pre-project.

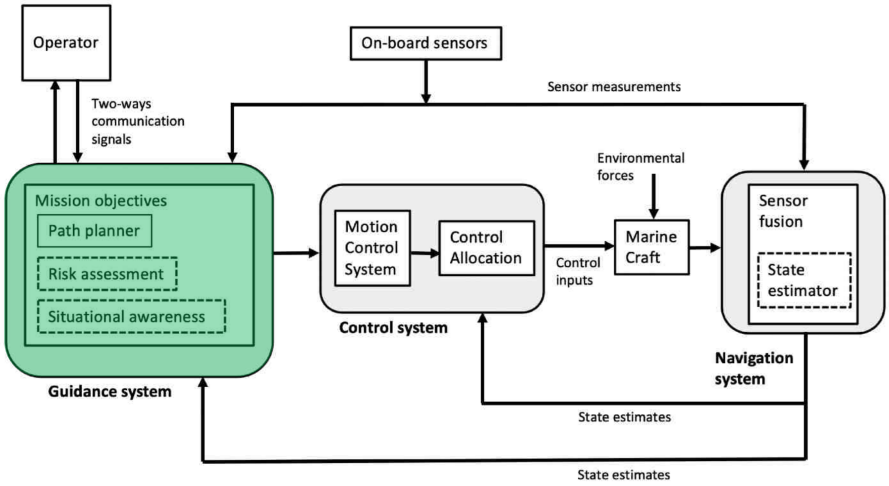


Figure 5.1: Guidance system in the system architecture. Courtesy of [10]

5.1 Waypoints and Straight Line Paths

Waypoint Guidance (WPG) is a widely used method to steer AUVs onto a desired path, and the method could also be utilized for the snake robot, as it inherits the slender body characteristics of an AUV. The path consists of several waypoints that are used to indicate changes in direction, speed and altitude along the desired path, as stated in [10]. Each waypoint is parametrized by cartesian coordinates in the world frame, $\mathcal{F}_I$, denoted

$$\boldsymbol{wp}_i = [x_i \; y_i \; z_i]^T \in \mathbb{R}^3, \quad i = 1, 2, ..., N_{\text{wp}} \tag{5.1}$$

where N_{wp} is the total number of waypoints. The common way of connecting the waypoints is by using straight lines and/or inscribed circles, and straight lines will be used in this thesis due to their simplicity. The waypoint guidance scheme will be utilized the same way as in [15], where the desired path is already planned. The planned path is split into N_{wp} waypoints, which is connected by $N_{\text{wp}} - 1$ straight lines. The path is hereafter denoted $\mathcal{P}$ and the straight line between $\boldsymbol{wp}_i$ and $\boldsymbol{wp}_{i+1}$

is denoted $\mathcal{P}_i$. Thus, the entire path $\mathcal{P}$ can be described as

$$\mathcal{P} = \bigcup_{i=1}^{N_{\text{wp}}-1} \mathcal{P}_i \tag{5.2}$$

It is important to note that the number of waypoints and their locations can not be chosen arbitrarily on the original path, as this procedure will inevitably change the shape of the path. In worst case we could end up with a new path that deviates a lot from the original collision-free path, which might not be collision-free. How the waypoints can be chosen will be further discussed in Section 5.2.

The snake robot will follow the waypoints in a predefined order defined by the path. To mark the current waypoint $\boldsymbol{wp}_{i+1}$ as visited and move on to the next one it utilizes a method called *sphere of acceptance* [10], formulated as

$$(x_{i+1} - x)^2 + (y_{i+1} - y)^2 + (z_{i+1} - z)^2 \leq r_{\text{soa},i+1}^2 \tag{5.3}$$

where $r_{\text{soa},i+1}$ is a pre-decided radius of a sphere, with the waypoint $\boldsymbol{wp}_{i+1}$ as its centre. A guideline is to choose $r_{\text{soa},i+1} = 2L$, where L is the length of the marine craft, but in the case where the followed path needs to be accurately followed in order to prevent collisions with obstacles, the *sphere of acceptance* might need to be smaller, depending on how cluttered the environment is and the safety margins used in the path planner. This will be further addressed in Section 5.2. A 3D path made of straight lines, with visualization of the *sphere of acceptance* at every waypoint succeeding the starting point, is illustrated in Figure 5.2.

A straight line $\mathcal{P}_i$ in $\mathbb{R}^3$ has an *azimuth angle*, $\chi_{\mathcal{P}_i}$, and a *flight path angle*, $\gamma_{\mathcal{P}_i}$. They will be the desired *heading* and *pitch* of the snake robot, relative to the inertial reference frame $\mathcal{F}_I$, when following the line $\mathcal{P}_i$. The relative *azimuth angle* and *flight path angle* for a straight line $\mathcal{P}_i$ between waypoint $\boldsymbol{wp}_i$ and $\boldsymbol{wp}_{i+1}$ are formulated as

$$\chi_{\mathcal{P}_i} = \text{atan2}\left(y_{i+1} - y_i, x_{i+1} - x_i\right) \in \mathbb{S} = [-\pi, \pi] \tag{5.4}$$

$$\gamma_{\mathcal{P}_i} = \text{atan2}\left(-(z_{i+1} - z_i), \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}\right) \in \mathbb{S} = [-\pi, \pi] \quad (5.5)$$

where $\text{atan2}(a, b)$ is in general the four-quadrant version of $\arctan\left(\frac{a}{b}\right) \in [-\frac{\pi}{2}, \frac{\pi}{2}]$, as stated in [4]. The line $\mathcal{P}_i$ can be parametrized by a scalar value, denoted $\bar{\omega} \in \mathbb{R}$, giving the following expression for an arbitrary point on the line

$$\boldsymbol{p}_{\mathcal{P}_i}(\bar{\omega}) = \boldsymbol{wp}_i + \bar{\omega}\frac{\boldsymbol{wp}_{i+1} - \boldsymbol{wp}_i}{||\boldsymbol{wp}_{i+1} - \boldsymbol{wp}_i||} \quad (5.6)$$

In order to steer the snake robot onto the line $\mathcal{P}_i$ a LOS guidance law will be utilized, which will be further discussed in Section 5.3.

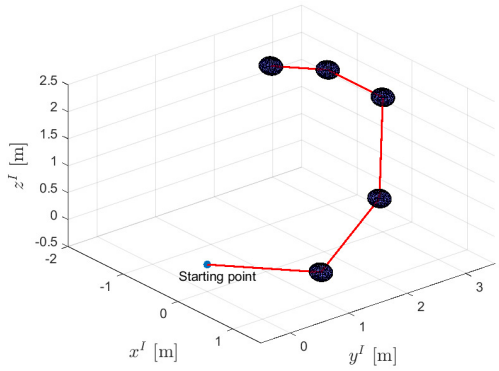


Figure 5.2: An example of a piecewise linear path of waypoints. The spheres visualize the *sphere of acceptance*.

5.2 Waypoint Generation

The waypoints used for the WLOS guidance scheme are generated from an already planned path, created by the APF path planner presented in section 4.2. This originally planned path normally consists of a huge amount of points, hereafter called redundant waypoints. The large number is caused by the small step length α used in *gradient*

descent, needed to make the path feasibly converge to the target. When generating the waypoints the idea is that the majority of these redundant waypoints are filtered out, and only some of them are used as final waypoints. It is important that the resulting straight line path $\mathcal{P}$ is also collision-free, as mentioned in section 5.1. In addition, there might be of interest to simultaneously search for a shortest straight line path in order for the snake robot to travel a minimum distance from A to B while still avoiding the obstacles. A third attribute of the path $\mathcal{P}$ to keep in mind is that the more complex the structure, the harder and more energy consuming the maneuvering executed by the snake robot must be to follow the path. For instance, paths with sharp corners and small distances between waypoints may be difficult to follow without larger deviations or the need for the snake robot to move very slowly to sufficiently visit each waypoint and reach the target. In this subsection two different waypoint generators will be presented.

5.2.1 Straight Lines of Constant Length

The waypoints created by using Straight Lines of Constant Length (SLCL) builds on the known physical dimensions of the snake robot. Every waypoint on the path are connected by two straight lines of a constant length, except the starting point and target point which are only connected to one straight line. This length is chosen to be equal to the half of the snake robot's length. The method ensures that the piecewise linear path is not violating the demand of being collision-free, due to the fact that the original path is already incorporating a safe radius that gives room for straight line adjustments, and the concept is visualized in Figure 5.3. This method might lead to many waypoints, and will not result in the most optimal path in terms of distance, due to its simple heuristic. Additionally, the snake robot also needs to maneuver onto a potentially large number of line segments and this might not be the most energy efficient solution either. The *sphere of acceptance* can't be chosen larger than half of the length of each line segment, or else the path could in practice change shape because subsets of waypoints are inside the same *sphere of acceptance* and seen as visited. Then the path is no longer known to be collision-free and therefore the *sphere of acceptance*

must be chosen according to the length of the line segments.

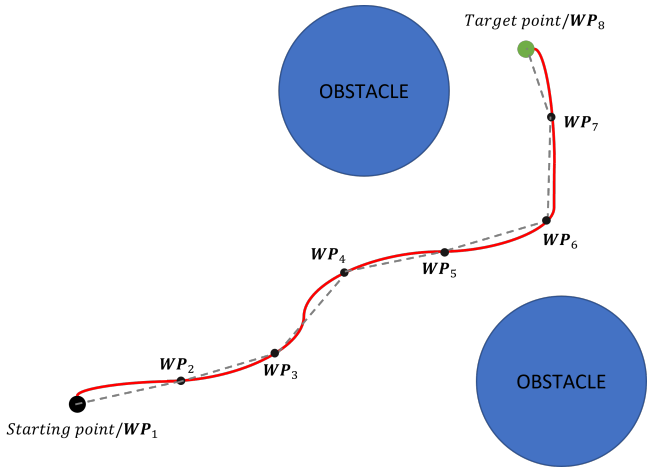


Figure 5.3: Visualize how the resulting piecewise linear path (dashed grey lines) from the SLCL-filtering algorithm will look like in 2D. The red line is the original path. Note that the safe radius r_{safe} is omitted from the illustration.

5.2.2 Backtracking Path Planner

The Bactracking Path planner (BPP) algorithm for waypoint generation is essentially searching for the minimum amount of waypoints needed to transform the original path into a collision-free piecewise linear path. In this way the resulting path $\mathcal{P}$ consists of less and longer straight lines that might ease the maneuvering task for the snake robot while following the path. Thus, it tries to create shorter and potentially less energy consuming paths for path following. A similar approach, utilizing a breadth-first search, was conducted in [22]. Conceptually, the BPP algorithm checks that the straight line path between two waypoints does not intersect with any of the obstacles in the $\mathcal{W}$-space. It starts by checking the straight line between the start point and the target point. If the line is collision-free the resulting path will only consist of one simple straight line between the starting point and the target point. If it does intersect

with an obstacle the algorithm moves along the path to the waypoint preceding the target point, checking whether this waypoint forms a collision-free straight line with the starting point. When it finds a suitable waypoint it considers this as a new virtual starting point and resumes its search for straight lines, but now from this virtual starting point to the target point. Eventually, the algorithm has found a number of waypoints connected by straight lines that make up a path $\mathcal{P}$ from the starting point to the target point. By always accepting the first collision-free straight line it finds the algorithm can be classified as greedy. A greedy algorithm always make the choice that looks best at the moment [8]. In this case it could be that the resulting piecewise linear path is not the shortest path possible, but the algorithm finds a shorter path than the SLCL algorithm from section 5.2.1. A pseudo-code of the algorithm is given in Appendix A.3. An example of a possible outcome of the algorithm is visualized in Figure 5.4.

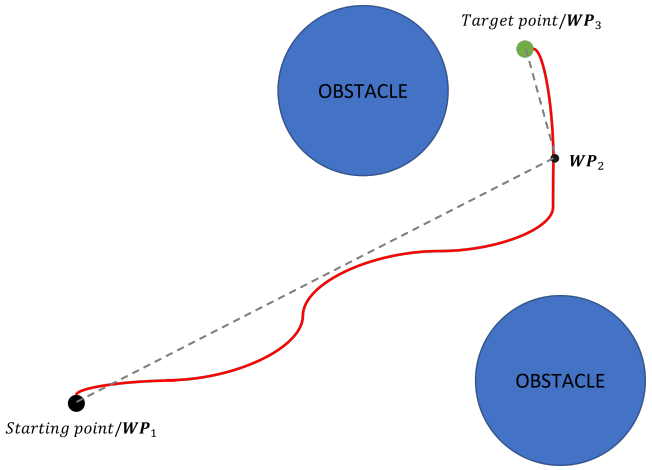


Figure 5.4: Visualize how the resulting piecewise linear path (dashed grey lines) from the BPP-filtering algorithm could look like in 2D. The red line is the original path. Note that the safe radius r_{safe} is omitted from the illustration.

In order to check for possible intersections between a straight line $\mathcal{P}_i$ the algorithm

utilizes the scalar product, in general formulated as

$$\boldsymbol{a}^T\boldsymbol{b} = |\boldsymbol{a}||\boldsymbol{b}|\cos(\theta), \quad \theta \in [0, \pi] \tag{5.7}$$

where $\boldsymbol{a}$ and $\boldsymbol{b}$ are 3-dimensional column vectors, θ is the angle between them and $|\cdot|$ is the euclidean norm of a vector. In the algorithm, the straight line $\mathcal{P}_i$ in question is seen as a vector from $\boldsymbol{wp}_i$ to $\boldsymbol{wp}_{i+1}$, which will be the $\boldsymbol{a}$-vector in the scalar product. The $\boldsymbol{b}$-vector will be the straight line from $\boldsymbol{wp}_i$ to an obstacle in the $\mathcal{W}$-space. By utilizing Equation (5.7) we are able to find θ, which is needed to compute the orthogonal projection of $\boldsymbol{b}$ onto $\boldsymbol{a}$ as well as the projected distance of $\boldsymbol{b}$ onto $\boldsymbol{a}$. Mathematically, these distances are computed as

$$d_{\text{orth}} = ||\boldsymbol{b}||\sin(\theta) \tag{5.8}$$

$$d_{\text{proj}} = ||\boldsymbol{b}||\cos(\theta) \tag{5.9}$$

Now, deciding whether the straight line stays clear of an obstacle $\mathcal{O}_i$ can be divided into 2 cases:

C1: If $d_{\text{orth}} > r_{\mathcal{O}_i} + r_{\text{safe}}$

C2: If $d_{\text{proj}} > ||\boldsymbol{a}|| + r_{\mathcal{O}_i} + r_{\text{safe}}$

These cases are also visualized in figs. 5.5 and 5.6.

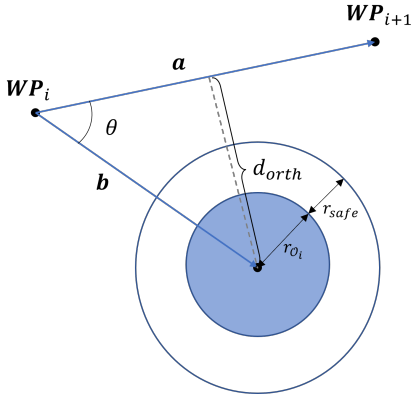


Figure 5.5: Visualizing how d_{orth} can be used to decide if the straight line $\mathcal{P}_i$ stays clear of the obstacle.

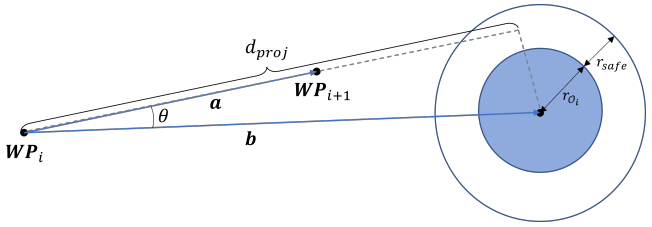


Figure 5.6: Visualizing how d_{proj} can be used to decide if the straight line $\mathcal{P}_i$ stays clear of the obstacle.

5.3 Line of Sight Guidance

Line-of-Sight (LOS) guidance is classified as a three-point guidance method since it involves a reference point, the intercepting marine craft and the target, as stated in [4]. In this thesis the intercepting marine craft will be the snake robot, the reference point will be the previously visited waypoint $\boldsymbol{wp}_i$ and the target will be the next waypoint $\boldsymbol{wp}_{i+1}$. The guidance law that will be utilized to guide the snake robot onto the straight line path is called lookahead-based guidance. Before presenting the lookahead-based approach a path-tangential reference frame and *along-track*, *cross-track* and *vertical-track errors* will be introduced.

5.3.1 Path-Tangential Reference Frame and Track-Errors

As the snake robot moves onto the piecewise linear path, its current position will deviate from the path at the beginning. The distance to the previous visited waypoint can be calculated in $\mathcal{F}_I$ as

$$\boldsymbol{p}_{\mathrm{e}}^I = \boldsymbol{p}_{Ib}^I(t) - \boldsymbol{wp}_i^I. \tag{5.10}$$

The error can also be expressed in a path-fixed reference frame, a frame from now on referred to as $\mathcal{F}_{\mathcal{P}_i}$. The current straight line the snake robot is following, $\mathcal{P}_i$, has an orientation described by the *azimuth angle* and *flight path angle*, as mentioned in section 5.1. By rotating an angle $\chi_{\mathcal{P}_i}$ about the path-tangential reference frame's z-axis we will obtain an intermediate reference frame, denoted $\mathcal{F}_{\mathcal{P}_i{}'}$. If we further rotate an angle $\gamma_{\mathcal{P}_i}$ about the y-axis of $\mathcal{F}_{\mathcal{P}_i{}'}$ we obtain the same orientation as the inertial reference frame $\mathcal{F}_I$. The orientational transformation from the inertial frame to the path frame is thus as shown in Figure 5.7. Mathematically, the orientation of frame $\mathcal{F}_{\mathcal{P}_i}$ wrt $\mathcal{F}_I$ is formulated as in [4]

$$\boldsymbol{R}(\chi_{\mathcal{P}_i}, \gamma_{\mathcal{P}_i}) := \boldsymbol{R}(\chi_{\mathcal{P}_i})\boldsymbol{R}(\gamma_{\mathcal{P}_i}), \tag{5.11}$$

where the right hand side rotation matrices are explicitly expressed as

$$\boldsymbol{R}(\chi_{\mathcal{P}_i}) := \begin{bmatrix} c_{\chi_{\mathcal{P}_i}} & -s_{\chi_{\mathcal{P}_i}} & 0 \\ s_{\chi_{\mathcal{P}_i}} & c_{\chi_{\mathcal{P}_i}} & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{R}(\gamma_{\mathcal{P}_i}) := \begin{bmatrix} c_{\gamma_{\mathcal{P}_i}} & 0 & s_{\gamma_{\mathcal{P}_i}} \\ 0 & 1 & 0 \\ -s_{\gamma_{\mathcal{P}_i}} & 0 & c_{\gamma_{\mathcal{P}_i}} \end{bmatrix} \tag{5.12}$$

where $c_a := \cos(a)$ and $s_a := \sin(a)$, $a \in [\chi_{\mathcal{P}_i}, \gamma_{\mathcal{P}_i}]$. From this relationship it is possible to express the error coordinates in $\mathcal{F}_{\mathcal{P}_i}$ by the transformation

$$\boldsymbol{p}_{\mathrm{e}}^{\mathcal{P}_i}(t) = \boldsymbol{R}(\chi_{\mathcal{P}_i}, \gamma_{\mathcal{P}_i})^T \boldsymbol{p}_{\mathrm{e}}^{I} \tag{5.13}$$

where $\boldsymbol{p}_{\mathrm{e}}^{\mathcal{P}_i}(t) = [s(t),\ e(t),\ h(t)]^T \in \mathbb{R}^3$ are the *along-track*, *cross-track* and *vertical-track errors* relative to the origin of $\mathcal{F}_{\mathcal{P}_i}$. The x-axis of $\mathcal{F}_{\mathcal{P}_i}$ coincides with the path, thus for path-following purposes the *along-track error* can be set to 0, i.e $s(t) = 0$.

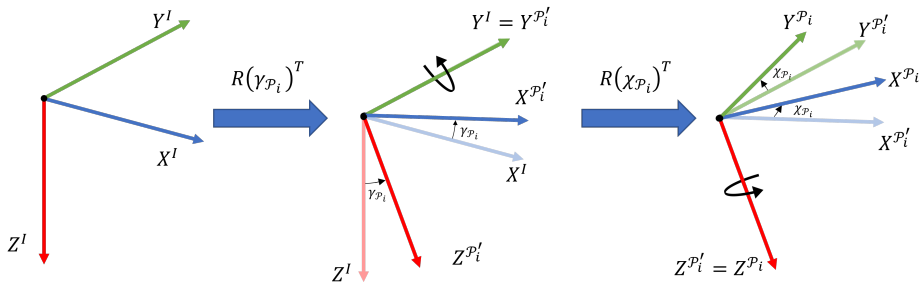


Figure 5.7: The orientational relationship from $\mathcal{F}_I$ to $\mathcal{F}_{\mathcal{P}_i}$

5.3.2 Lookahead-Based Guidance

The lookahead-based guidance method builds on the idea of having a point on the straight line path that the marine craft is guided towards. The lookahead distance is denoted Δ and represents the distance of the point relative to the origin of the path-tangential reference frame, directed along its x-axis. The lookahead distance is

a tuning parameter that decides how the marine craft shall encounter and converge onto the path. If Δ is very small, the craft will encounter the path with a large angle, meaning that it must be highly maneuverable or lower its speed to prevent large oscillations onto the path. If Δ is chosen very large, the craft will slowly converge onto the path. For the 3D-case, the lookahead-based approach is separated into two parts, as stated in [4]. The desired *azimuth angle* and *flight path angle* are formulated as

$$\chi_d(\chi_{\mathcal{P}_i}, \gamma_{\mathcal{P}_i}, \chi_r, \gamma_r) = \text{atan2}\left(f(\chi_{\mathcal{P}_i}, \gamma_{\mathcal{P}_i}, \chi_r, \gamma_r), g(\chi_{\mathcal{P}_i}, \gamma_{\mathcal{P}_i}, \chi_r, \gamma_r)\right) \tag{5.14}$$

$$\gamma_d(\gamma_{\mathcal{P}_i}, \chi_r, \gamma_r) = \text{arcsin}\left(\text{sin}\gamma_{\mathcal{P}_i}\text{cos}\chi_r\text{cos}\gamma_r + \text{cos}\gamma_{\mathcal{P}_i}\text{sin}\gamma_r\right), \tag{5.15}$$

where

$$\begin{aligned} f(\chi_{\mathcal{P}_i}, \gamma_{\mathcal{P}_i}, \chi_r, \gamma_r) = \text{cos}\chi_{\mathcal{P}_i}\text{sin}\chi_r\text{cos}\gamma_r - \text{sin}\chi_{\mathcal{P}_i}\text{sin}\gamma_{\mathcal{P}_i}\text{sin}\gamma_r + \\ \text{sin}\chi_{\mathcal{P}_i}\text{cos}\gamma_{\mathcal{P}_i}\text{cos}\chi_r\text{cos}\gamma_r \end{aligned} \tag{5.16}$$

$$\begin{aligned} g(\chi_{\mathcal{P}_i}, \gamma_{\mathcal{P}_i}, \chi_r, \gamma_r) = -\text{sin}\chi_{\mathcal{P}_i}\text{sin}\chi_r\text{cos}\gamma_r - \text{cos}\chi_{\mathcal{P}_i}\text{sin}\gamma_{\mathcal{P}_i}\text{sin}\gamma_r + \\ \text{cos}\chi_{\mathcal{P}_i}\text{cos}\gamma_{\mathcal{P}_i}\text{cos}\chi_r\text{cos}\gamma_r \end{aligned} \tag{5.17}$$

and

$$\chi_r(e(t)) := -\text{arctan}\left(\frac{e(t)}{\Delta}\right), \quad \Delta > 0 \tag{5.18}$$

$$\gamma_r(h(t)) := \text{arctan}\left(\frac{h(t)}{\mu\sqrt{e(t)^2 + \Delta^2}}\right), \quad \mu, \Delta > 0 \tag{5.19}$$

The reference angles in eqs. (5.18) and (5.19) are used to make the snake robot converge towards the xz-plane and xy-plane of $\mathcal{F}_{\mathcal{P}_i}$, respectively. Here, Δ and μ shape the references, i.e. decide how the snake robot will converge onto the path, both horizontally and vertically. The μ parameter gives more freedom regarding tuning of the convergence onto the xz-plane, where a larger μ results in slower convergence than a smaller μ. An illustration of the lookahead-based 3D-LOS guidance for the snake robot is shown given in Figure 5.8, inspired by the one in [4].

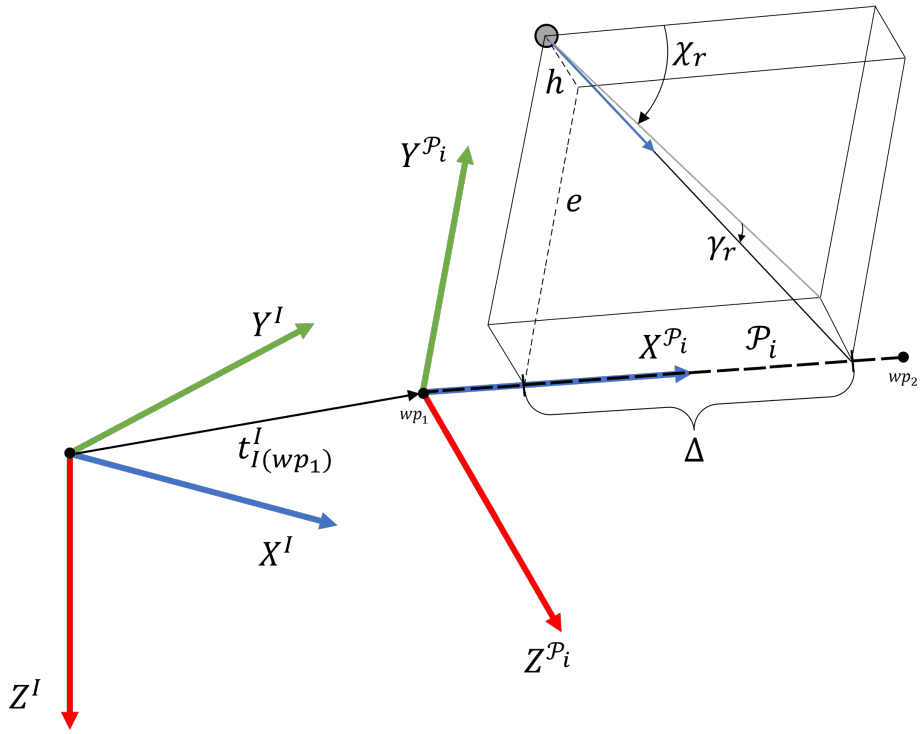


Figure 5.8: The path-tangential reference frame $\mathcal{F}_{\mathcal{P}_i}$ wrt $\mathcal{F}_I$, along with the main steering variables for 3D-LOS. The snake robot base frame origin is the grey circle at the top of the box. The blue arrow is the velocity vector of the snake robot.

5.3.3 Sideslip Angle and Angle of Attack

The desired angles χ_d and γ_d are calculated wrt the projection of the base frame velocity $\boldsymbol{\nu}^b_{Ib}$ in the xy-plane and the xz-plane of the body frame $\mathcal{F}_b$. Thus, they give the desired *course* and *flight-path* angles the snake robot should follow. We want to control the *heading* and *pitch* angles in order to also make the snake robot's body align well with the path. The benefits of aligning with the path is that sensors such as cameras can provide data about the environment in the direction the snake robot is

headed. In order to utilize χ_d and γ_d for this purpose it is important to account for the *sideslip angle* and *angle of attack*, respectively denoted β_{ss} and α_{aoa}. They appear when the ship is subject to external disturbances, or during turning, making a discrepancy between χ_d and ψ_d, and between γ_d and θ_d. The relationship between these angles are given as in [3]

$$\psi_d = \chi_d - \beta_{\text{ss}} \tag{5.20}$$

$$\theta_d = \gamma_d - \alpha_{\text{aoa}} \tag{5.21}$$

where the *sideslip angle* and *angle of attack* are given as

$$\beta_{\text{ss}} = \arcsin(v/\sqrt{(u^2 + v^2)}) \tag{5.22}$$

$$\alpha_{\text{aoa}} = \arctan(w/u) \tag{5.23}$$

The angles are found from the geometrical relationship visualized in figs. 5.9 and 5.10, inspired by the ones in [10].

5.3.4 Forward Speed Law

In addition to the guidance laws, the snake robot also needs to be fed a desired speed to track, in order to move along the path. The desired speed should be possible to achieve while still being able to stay on the path. Therefore it should somehow be related to the lookahead distances and how far away the snake robot is from the path, which is in fact done in [3]

$$u_d = \kappa\sqrt{\mu^2(e^2 + \Delta^2) + h^2}, \tag{5.24}$$

where $\kappa > 0$ is a constant gain parameter that gives the ability to adjust the desired speed wrt the kinematic abilities of the vessel used. From Equation (5.24) we can see that the desired forward speed is higher the farther away the snake robot is from the path, which relates to larger *cross-track error* and *vertical-track error*. The smaller the lookahead distance parameters Δ and μ are, the smaller the desired forward speed gets.

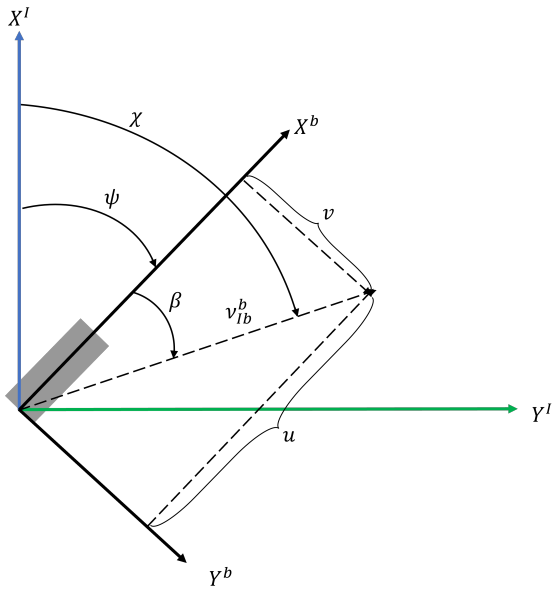


Figure 5.9: The horizontal ocean triangle, with the snake robot's base as the grey rectangle. The figure shows the horizontal velocity component of the base, decomposed in *surge* and *sway*. The relationship between the *course angle* χ, *heading angle* ψ and *sideslip angle* β_{ss} are also shown.

This relationship between the desired forward speed and the guidance parameters allows the snake robot to move faster when it is farther away from the path or has a less aggressive convergence onto it. On the other side, it prevents the snake robot from moving too fast and overshoot the path when it is close to and converges more aggressively onto it.

Slow-Regions

The reasonable forward speed of the snake robot depends on the environment and the shape of the path it's following. The shape of the path is defined by how cluttered the environment is and the algorithm used to obtain the path. When the snake robot is closing in on a waypoint, its velocity should be smaller in order to prevent large

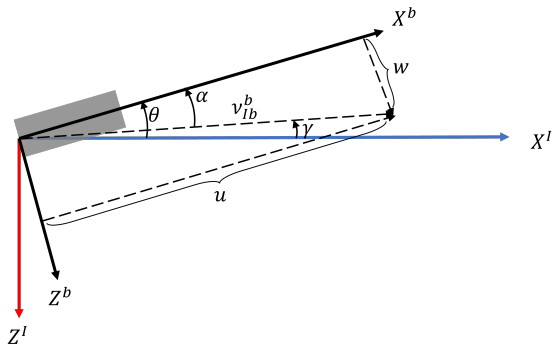


Figure 5.10: The vertical ocean triangle, with the snake robot's base as the grey rectangle. The figure shows the longitudinal velocity component of the base, decomposed in *surge* and *heave*. The relationship between the *flight-path angle* γ, *pitch angle* θ and *angle of attack* α_{aoa} are also shown.

an overshoot when switching from the currently followed line segment, $\mathcal{P}_i$, to the next line segment, $\mathcal{P}_{i+1}$. This depends on the angle between $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$, and the approaching speed should be smaller for a bigger change in both desired *pitch* and desired *heading*. If the snake robot overshoots too much it might collide with an obstacle or pass by a waypoint without visiting it, which would lead to failure in terms of leading the snake robot to the desired target. The forward speed law from Equation (5.24) does not consider the transition from $\mathcal{P}_i$ to $\mathcal{P}_{i+1}$, which could result in undesirable overshooting behaviour around waypoints. To cope with the potential problem of approaching waypoints with too large speed, slow-regions are implemented as spheres around each waypoint. In order to decrease the desired forward speed inside these regions we can divide the current output obtained from Equation (5.24) by a value proportional to the angle between $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$. If the line segments are close to parallel the desired forward speed shouldn't change much, but the sharper the angle the more the speed need to decrease in order to prevent large overshoot. Thus, we can define a mapping from the angle between $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$, given as

$$K_{\text{sr}} = \frac{\pi - \theta}{\pi}(K_{\text{sr,upper}} - 1) + 1, \tag{5.25}$$

where $K_{\text{sr}} \in [1, K_{\text{sr,upper}}]$ is a scaling factor and θ is here defined as the angle between the two line segments, not to be misunderstood as the *pitch* angle. The new desired forward speed is thus

$$u_d = \frac{u_d}{K_{\text{sr}}} \tag{5.26}$$

The size of each slow-region's sphere, denoted r_{sr}, and how much the speed should decrease inside a slow-region depends on the situation, and thus both r_{sr} and $K_{\text{sr,upper}}$ need to be tuned according to the path and the environment.

5.3.5 Smoothing References

The references from the guidance laws will be discontinuous in the transition from $\mathcal{P}_i$ to $\mathcal{P}_{i+1}$ when the snake robot switch from waypoint $\boldsymbol{wp}_i$ to $\boldsymbol{wp}_{i+1}$. This will result in huge velocity references, which in turn demands the actuators to provide large forces in order to follow the references. It's not desirable to have actuators operating at their max performance unless absolutely necessary, due to wear and tear of the equipment and high energy consumption. To cope with this problem the references are fed through a low-pass filter [10], which smooths the references in the transition from following line $\mathcal{P}_i$ to following line $\mathcal{P}_{i+1}$. Thus, the new references from the guidance laws are given as

$$\psi_d = h_{\text{lp}} \psi_d \tag{5.27}$$

$$\theta_d = h_{\text{lp}} \theta_d \tag{5.28}$$

where h_{lp} is the transfer function of a suitable low-pass filter to make the references smooth and feasible for the snake robot to track, without demanding too large outputs from its thrusters.

Chapter 6

Control System

The snake robot's desired trajectories and paths from the *guidance* system need to be followed sufficiently in order for the snake robot to execute its tasks, which for this thesis is collision avoidance. For this purpose a *control* system is needed, consisting of a motion control system and control allocation. The control allocation calculates the control inputs, which are fed to the snake robot's actuators in order to execute the desired motions, which is shown in Figure 6.1. This chapter will mainly cover the motion control system in the form of Proportional, Integral and Derivative (PID)-controllers, and control allocation will be briefly explained.

6.1 PID Motion Control System

The motion control system of a dynamic system, such as the snake robot, can be designed in various ways, depending on the dynamical characteristics of the system and the applications of which the system will be used for. The performance of various controller techniques depends on the parametric uncertainties of the model, disturbances acting on the system and unmodelled dynamics. For this thesis, where the

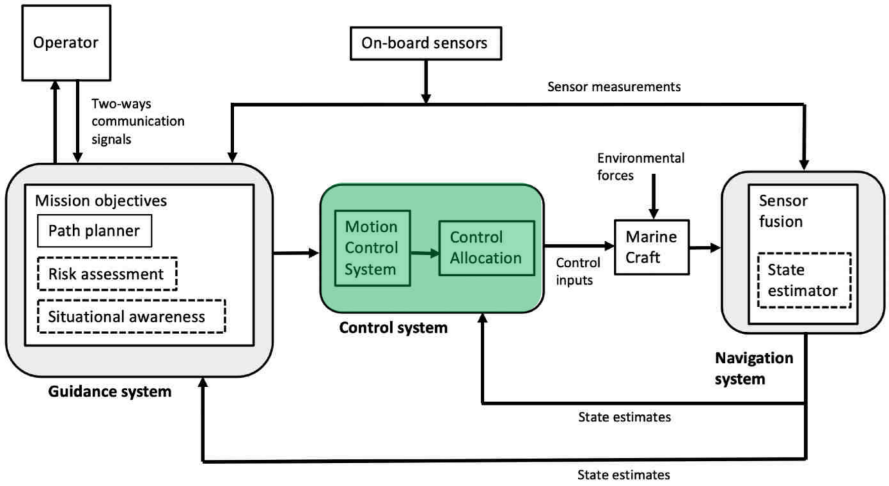


Figure 6.1: Control system in the system architecture. Courtesy of [10].

application is collision avoidance for a snake robot with assumed known parameters and not subject to environmental disturbances, we will utilize basic PID-controllers as our motion control system. In general, the commanded generalized force calculated by PID-controllers in combination with additional *feed-forward* terms are formulated as

$$\tau_c(t) = \tau_{\text{FF}}(t) + \tau_{\text{PID}}(t) = \tau_{\text{FF}}(t) - k_p e(t) - k_d \dot{e}(t) - k_i \int_0^t e(\rho)\, d\rho \tag{6.1}$$

where $k_p,\ k_d,\ k_i \geq 0$, depending on the type of PID-controller utilized. In addition, τ_{FF} is the *feed-forward* term and $e(t) := x(t) - x_d(t)$ is defined as the error of some state x of the system, relative to its desired value x_d. For ease of notation, the time-dependency of each state and the control inputs are omitted for the rest of this chapter.

The *feed-forward* term and the *proportional* (P), *integral* (I) and *derivative* (D) terms serves different purposes in the control system [36], and will be briefly recapitulated here:

- The *feed-forward* term's purpose is to eliminate the effects of disturbances before they create control errors. Thus, in order for the *feed-forward* method to work it requires that the disturbances are measured and a model of the system to be controlled. As this term handles disturbances before they have created errors the response time of the system is often faster than when utilizing integral action, and the need for integral action is smaller.

- The *proportional* term is proportional to the error e. It's purpose is to ensure a desirable response time of the system and handles sudden changes.

- The *integral* term is proportional to the integral of the error e. Integral action is added to the controller in order to remove steady-state offsets that occurs when the system experience some form for disturbance. Thus, this term handles slowly varying changes.

- The *derivative* term is proportional to the derivative of the error e. It's purpose is to improve the stability of the closed-loop system, and it does so by predicting the system output and provide additional damping to the system.

6.1.1 Forward Speed Autopilot

The task of the forward speed autopilot is to control the *surge* of the snake robot's base frame. In the form of a PID-controller with *feed-forward*, the autopilot does in general look like this

$$\tau_{c,u} = \tau_{\text{FF}} - k_{p,u}\tilde{u} - k_{d,u}\dot{\tilde{u}} - k_{i,u}\int_0^t \tilde{u}(\rho)\, d\rho, \tag{6.2}$$

where $\tilde{u} := u - u_d$ and $\dot{\tilde{u}} := \dot{u} - \dot{u}_d$. The *feed-forward* term for the forward speed autopilot is provided in order to cancel the gravitational and buoyancy forces that acts in the x-axis of the base frame and is thus given as the first component of the gravitational and buoyancy vector in Section 3.3.1, $\boldsymbol{g}(\boldsymbol{q}, \boldsymbol{R}_I^b)$.

6.1.2 Pitch Autopilot

The task of the pitch autopilot is to control the *pitch* of the snake robot's base frame. In the form of a PID-controller with *feed-forward*, the autopilot does in general look like this

$$\tau_{c,\theta} = \tau_{\text{FF}} - k_{p,\theta}\tilde{\theta} - k_{d,\theta}\dot{\tilde{\theta}} - k_{i,\theta}\int_0^t \tilde{\theta}(\rho)\, d\rho, \tag{6.3}$$

where $\tilde{\theta} := \theta - \theta_d$ and $\dot{\tilde{\theta}} := \dot{\theta} - \dot{\theta}_d$. The *feed-forward* term for the pitch autopilot is provided in order to cancel the moment about the y-axis of the base frame created by the gravitational and buoyancy forces. The fifth component of $\boldsymbol{g}(\boldsymbol{q}, \boldsymbol{R}_I^b)$ provides the moment about the y-axis, and is thus fed forward to prevent larger control errors in the *pitch* motion.

6.1.3 Heading Autopilot

The task of the heading autopilot is to control the *heading* of the snake robot's base frame. In the form of a PID-controller with *feed-forward*, the autopilot does in general look like this

$$\tau_{c,\psi} = \tau_{\text{FF}} - k_{p,\psi}\tilde{\psi} - k_{d,\psi}\dot{\tilde{\psi}} - k_{i,\psi}\int_0^t \tilde{\psi}(\rho)\, d\rho, \tag{6.4}$$

where $\tilde{\psi} := \psi - \psi_d$ and $\dot{\tilde{\psi}} := \dot{\psi} - \dot{\psi}_d$. The *feed-forward* term for the pitch autopilot is provided in order to cancel the moment about the z-axis of the base frame, created by the gravitational and buoyancy forces. The sixth component of $\boldsymbol{g}(\boldsymbol{q}, \boldsymbol{R}_I^b)$ provides the moment about the z-axis, and is thus fed forward to prevent larger control errors in the *heading* motion.

6.1.4 Joint Controllers

The joint controllers are implemented in order to calculate the joint torques needed to follow the desired joint trajectories, and each controller is formulated as

$$\tau_{c,q} = -k_{p,q}\tilde{q} - k_{d,q}\dot{\tilde{q}}, \tag{6.5}$$

where $\tilde{q} := q - q_d$ and $\dot{\tilde{q}} := \dot{q} - \dot{q}_d$.

6.2 Control Allocation

Control allocation is in general the process of distributing the generalized control forces $\boldsymbol{\tau}$ to the actuators in terms of the control inputs $\boldsymbol{u}$ [10]. For the snake robot this relates to the generalized control forces from its m thrusters, which for the snake robot in this thesis is seven. The thrusters are used to actuate the snake robot, wrt to the motion of the base frame $\mathcal{F}_b$. The relationship between the generalized control forces for the thrusters of the snake robot's base frame, $\boldsymbol{\tau}_b \in \mathbb{R}^6$, and the control inputs, $\boldsymbol{u}_{\text{thr}} \in \mathbb{R}^7$, are given as

$$\boldsymbol{\tau}_b = \boldsymbol{B}(\boldsymbol{q})\boldsymbol{u}_{\text{thr}}, \tag{6.6}$$

where $\boldsymbol{B}(\boldsymbol{q})$ is the TCM from Section 3.3.1. Due to the fact that the number of thrusters is bigger than the DOFs of the snake robot's base, the thrust allocation problem is an *overactuated control* problem. In general, *overactuated control* problems can be solved by utilizing the right *Moore-Penrose pseudoinverse* [10], denoted

$$\boldsymbol{u}_{\text{thr}} = \boldsymbol{B}(\boldsymbol{q})^{\dagger}\boldsymbol{\tau}_b, \quad \boldsymbol{B}(\boldsymbol{q})^{\dagger} = \boldsymbol{B}(\boldsymbol{q})^T \left(\boldsymbol{B}(\boldsymbol{q})\boldsymbol{B}(\boldsymbol{q})^T\right)^{-1} \tag{6.7}$$

Because the snake robot can change its configuration, leading to $\boldsymbol{B}(\boldsymbol{q})$ being time-dependent, the TCM may become singular. This needs to be dealt with, otherwise we lose control of the snake robot. A simple but effective strategy for solving this is to utilize the *damped pseudoinverse*

$$\boldsymbol{u}_{\text{thr}} = \boldsymbol{B}(\boldsymbol{q})^{\dagger}\boldsymbol{\tau}_b, \quad \boldsymbol{B}(\boldsymbol{q})^{\dagger} = \boldsymbol{B}(\boldsymbol{q})^T \left(\boldsymbol{B}(\boldsymbol{q})\boldsymbol{B}(\boldsymbol{q})^T + \lambda^2\right)^{-1} \tag{6.8}$$

where λ is the damping factor. In near-singular thruster configurations this method will allow the thrusters to deviate from the commanded thrust, which would otherwise lead to excessively high thrust to solve the task. This method was used in [32], and will be utilized for control allocation in this thesis.

Chapter 7

Implementation and Simulation

This chapter will introduce the results from the implementation of path planning and control for collision avoidance of the snake robot in a static cluttered environment. First, Section 7.1 presents the results from planning 3-dimensional global collision-free paths. To plan the paths the APF and RRT* planners will be utilized for global path planning, and their ability to find feasible paths will be evaluated. Secondly, in Section 7.2 the snake robot is set to follow the created paths from the planners by utilizing the WLOS guidance method, which additionally incorporates slow-regions, in order to avoid the obstacles in the environment. The resulting path following performance will be compared in terms of the snake robot's ability to follow the paths in a collision-free manner. In addition, the snake robot's energy consumption will also be considered, indicating how demanding the paths are to be followed.

7.1 Path Planning

The global path planning task in a 3-dimensional $\mathcal{W}$-space will be solved by the RRT* algorithm and the APF algorithm, incorporating a *random walk* mode. As the planned path from the APF is quite dense it is filtered into a much smaller number of waypoints to simplify the maneuvering needed to follow the path, which could lead to less energy consuming paths. The dense path is filtered by the use of both the BPP-filter and SLCL-filter from Section 5.2.

7.1.1 Environment

The three environments that the path planners are tested on are visualized in Figure 7.1, and the specifications of the environments, i.e the obstacles, target location and the start position of the snake robot, are listed in tables 7.1 to 7.3. The upper and lower bounds of all the environments are set to ± 25[m] in all three directions; *north*, *east* and *down*, except for **environment 3**, where a flat square approximation of the ocean floor is added at -4[m] in the *down* direction. The initial position of the snake robot in all the simulations is

$$\boldsymbol{p}_{Ib}^{I}(0) = [-1.7\ 0\ 0]^T$$

ENVIRONMENT 1			
	Obstacle type	**Position [m]**	**Radius [m]**
Target	N/A	[20, 5, 6]	0.5
Obstacle 1	Sphere	[5, 1, 4]	2
Obstacle 2	Sphere	[4, -4, 0]	3
Obstacle 3	Sphere	[10, -1, 3]	1
Obstacle 4	Sphere	[15, 5, 0]	3

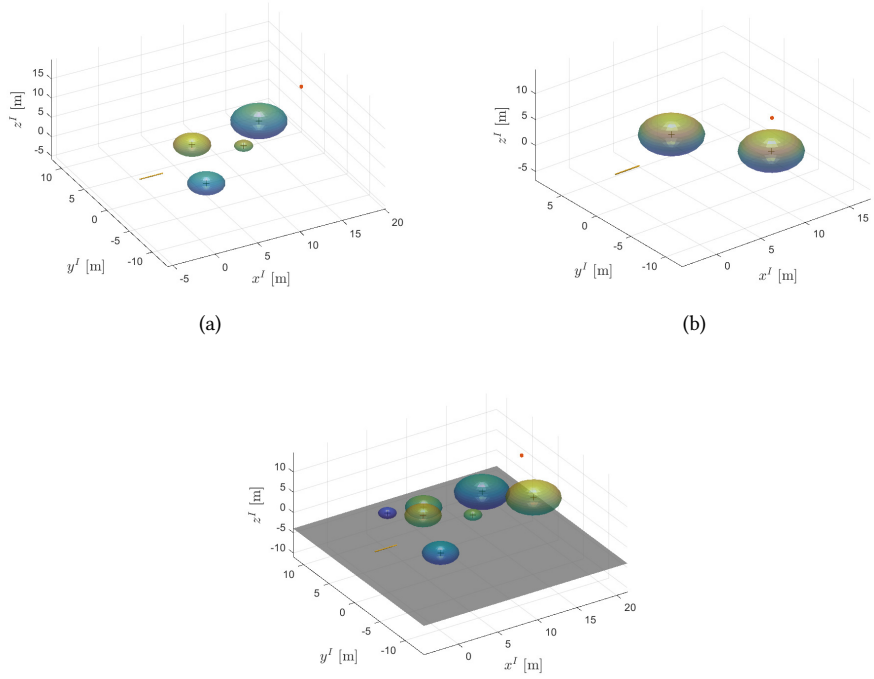
Table 7.1: Obstacles and target specifications of **environment 1**.

ENVIRONMENT 2			
	Obstacle type	**Position [m]**	**Radius [m]**
Target	N/A	[16, 0, 0]	0.5
Obstacle 1	Sphere	[7, 3, 0]	2
Obstacle 2	Sphere	[12, -5, 0]	3

Table 7.2: Obstacles and target specifications of **environment 2**.

ENVIRONMENT 3			
	Obstacle type	**Position [m]**	**Radius [m]**
Target	N/A	[20, 5, 6]	0.5
Obstacle 1	Sphere	[5, 1, 4]	2
Obstacle 2	Sphere	[4, -4, 0]	3
Obstacle 3	Sphere	[10, -1, 3]	1
Obstacle 4	Sphere	[15, 5, 0]	3
Obstacle 5	Sphere	[7, 4, 2]	2
Obstacle 6	Sphere	[17, -2, 4]	3
Obstacle 7	Sphere	[5, 8, -2]	1

Table 7.3: Obstacles and target specifications of **environment 3**.



(a) (b)

(c)

Figure 7.1: The three environments used to test the different path planners' performance. (a) Shows **environment 1**, consisting of 4 spherical obstacles, the target (red dot), and the snake robot (orange line) in its initial configuration. (b) Shows **environment 2**, consisting of 2 spherical obstacles, the target (red dot), and the snake robot (orange line) in its initial configuration. (c) Shows **environment 3**, consisting of 7 spherical obstacles, the target (red dot), and the snake robot (orange line) in its initial configuration. In addition, the lower bound is created by the ocean floor, which is illustrated as a grey square.

7.1.2 APF Path Planner

The APF path planner implemented here consists of three parts; a potential field, the *gradient descent* algorithm and a *random walk* for escaping local minima, based on Section 4.2. For the APF planner to obtain a collision-free path it needs to be tuned. The tuning process is usually based on trial and error, which is stated in [7]. The most important aspect is to find sufficient numerical values for the hyperparameters, in order for the planned path to be collision-free and navigable for the snake robot.

Tuning

The hyperparameters for the APF algorithm, including the hyperparameters for the potential field, *gradient descent* and the *random walk*, need to be tuned for acceptable path planning. The numerical values are given in Table 7.4 and an elaboration follows here:

- *Step size*, α. There are a few criterias to be considered for choosing the appropriate α for this task. Due to the fact that the *step size* is determining how far *gradient descent* will travel every iteration, it should be chosen such that the distance between every iteration avoids moving through or too close to the obstacles. If it moves through an obstacle, the path is in practice useless, or if it moves too close to an obstacle, where the repulsive potential is big, it might result in the next step being pushed far from the previous one and possibly in a non-optimal direction, leading away from the target. The approximate value where this non-desirable behaviour emerges is denoted $\alpha_{\text{zig-zag}}$. Such behaviour complicates the task of subsequently dividing the path into useful waypoints, and is thus avoided. Another criteria is the desire of implementing a planner which finds a path in a reasonable amount of time. Thus, $\alpha \in (\alpha_{\text{slow}}, \alpha_{\text{zig-zag}})$ is the acceptable set of α, preventing slow run-time and undesirable paths. From trial and error $\alpha = 0.1$. Illustrations of the discussed problems with $\alpha > \alpha_{\text{zig-zag}}$, leading to so called zig-zag paths and paths that travels through obstacles, are shown in fig. 7.2, for $\alpha = 1$ and $\alpha = 2$.

- *Gradient tolerance*, ϵ. This parameter is used for terminating the *gradient descent* algorithm. For these simulations ϵ is chosen to be 0.1.

- *Random steps*, N_{rw}. The total number of random steps to execute during the *random walk* procedure, which together with the length of each step defines how far the *random walk* is allowed to travel. For planning in **environment 1-3** the total step size is set to 60.

- *Random step length*, δ_{rw}. The length to travel in each direction for every random step in the *random walk*, which together with the total number of random steps defines how far the *random walk* can travel. Since every random step is also checked for collisions it is beneficial to not make the step length too big, in order to avoid collisions and rather have a larger number of total steps in order to travel a longer distance. Thus, the random step length is set to 0.15[m].

- *Local minimum threshold*, $\epsilon_{\text{trapped}}$. The threshold value determines whether or not the APF planner has been trapped by a local minimum. The four latest samples are chosen to check whether a local minimum is present. If the euclidean distance between the oldest sample and the other three samples are all below $\epsilon_{\text{trapped}}$, a local minimum is detected. The threshold is set to 0.2[m].

- *Potential gain*, K_a. This parameter decides the size of the gradient, and is set to 1.

- *Repulsive gain*, K_r. This parameter decides the size of the gradient, and is set to 5.

- *Switching distance*, d_t^*. The switching distance decides whether the attraction function should be a *conical* or *paraboloide* function. It is set to 25[m], in practice making the attractive potential solely a *paraboloide* function. This is due to the fact that K_a is not too large and the $\mathcal{W}$-space domain is not big enough in order for the gradient to grow too large, which was the reason for using the *conical* function.

- *Smoothing obstacle gradient*, Q_o^*. The last hyperparameter for the repulsive functions, Q_o^*, is after trial and error chosen to be 15. This makes a more gently increasing gradient relatively close to an obstacle, preventing the next step of *gradient descent* to be pushed far away from the last one, and perhaps in a non-optimal direction.

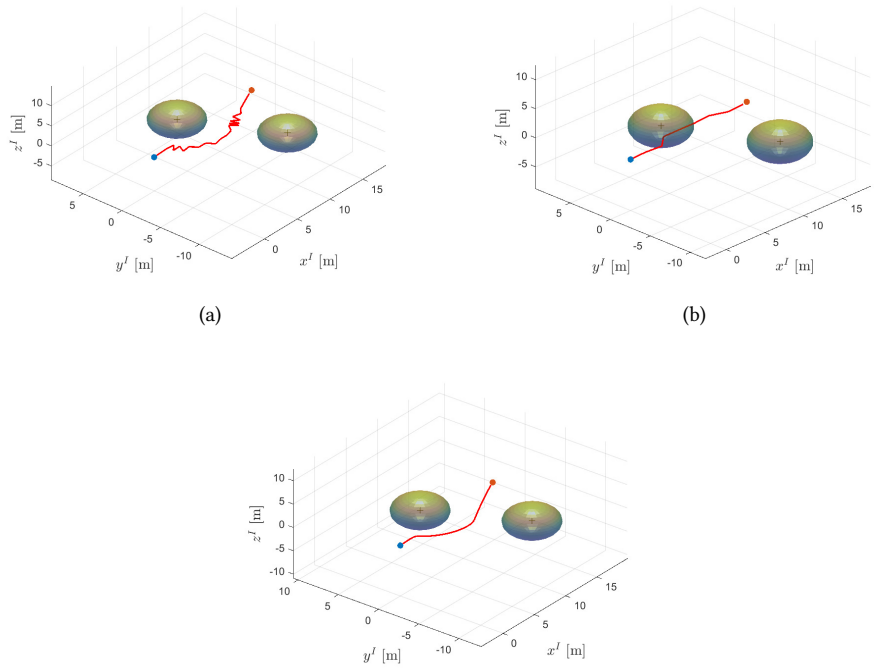
- *Safe radius*, r_{safe}. The *safe radius* is here chosen to be half of the snake robot's length in order to give the snake robot some path following margins. This results in a path that does not demand the snake robot to follow the path with 100% accuracy at all time to avoid collisions.

Hyperparameter	Numerical value	Unit
α	0.1	1
ϵ	0.1	m
K_a	1	1
K_r	5	1
d_t^*	25	m
Q_o^*	15	1
r_{safe}	1.7	m
N_{rw}	60	1
δ_{rw}	0.15	m
$\epsilon_{\text{trapped}}$	0.2	m

Table 7.4: Hyperparameter values for the APF path planner. The "1" in the **Unit** column specifies dimensionless quantities.

Performance of the APF planner

When all the hyperparameters are chosen, the path planner is tested on the environments visualized in Figure 7.1. The planner manages to find a collision-free path for **environment 1** and **environment 2**, which can be seen from figs. 7.3 and 7.4, indicating that the APF path planner is indeed capable of finding collision-free paths in a cluttered environment, after tuning of its hyperparameters. In addition, it managed to escape a local minimum in **environment 2**. **Environment 3** proved to be much



(a) (b)

(c)

Figure 7.2: Different tuning of α results in different paths, where some might be impracticable. (a) and (b) shows impracticable paths generated with $\alpha = 1$ and $\alpha = 2$, respectively. (c) shows a feasible path with $\alpha = 0.1$.

trickier to converge to the target, the APF planner seemed to get stuck in several local minima and failed to reach the target. Thus, even though the *random walk* manages to help the APF planner in converging to the target it can be really hard when the number of obstacles increase and the number of local minima grows. In addition, the more obstacles that are present, the harder it gets to find feasible collision-free steps during the *random walk*. For the succeeding paths, the waypoints are successfully filtered out by utilization of both the BPP-filter and SLCL-filter from Section 5.2, still resulting in

collision-free paths. The filtered paths in both **environment 1** and **environment 2** are shown in Figure 7.3 and Figure 7.4, respectively. In Figure 7.4 we can see that the resulting SLCL-filtered path inherit some of the random pattern from the *random walk* while the BPP-filtered path does not. In the scenario where a local minimum appears and the original path from the APF planner gets a random pattern the BPP will make better suited paths for path following due to its ability to literally go straight through the *random walk* areas and obtain paths that need less aggressive maneuvering to be followed.

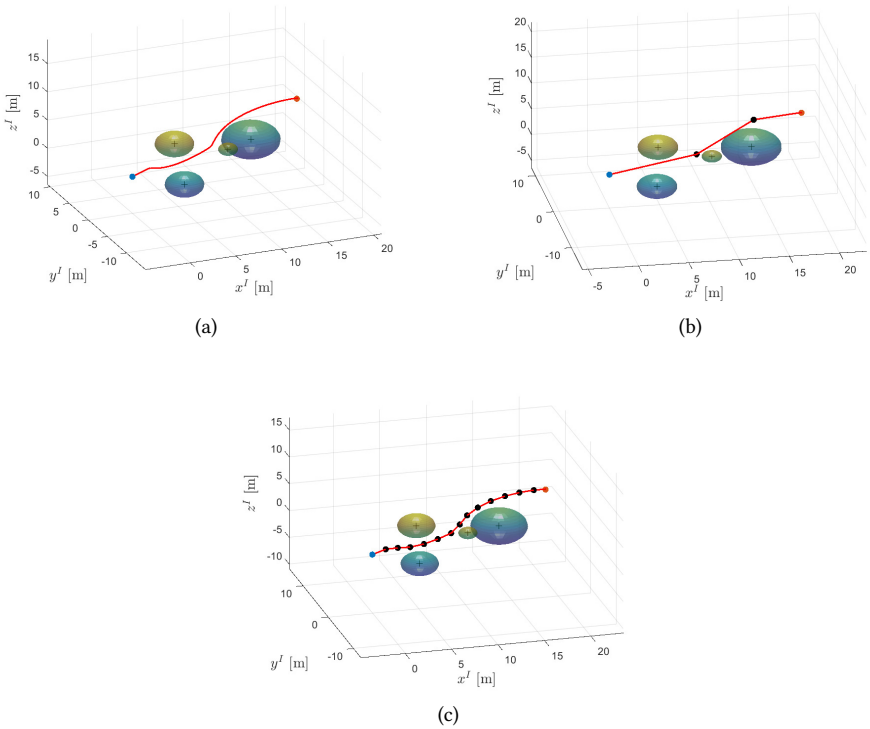
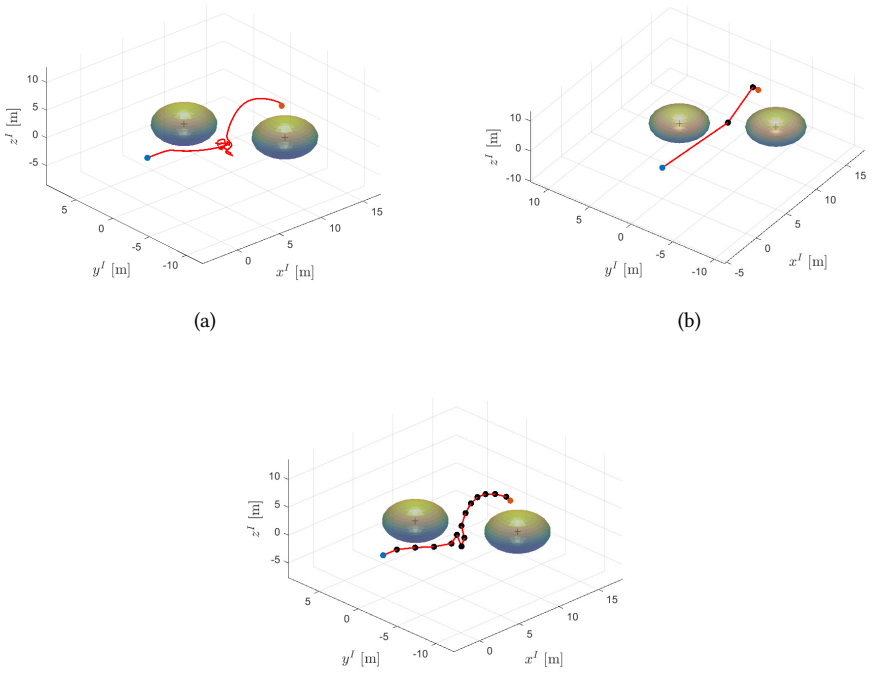


Figure 7.3: The planned paths in **environment 1**, based on the APF planner. The starting and target points are visualized as blue and red dots, respectively. (a) Path from the APF planner. (b) The APF path filtered by the BPP-filter. (c) The APF path filtered by the SLCL-filter. We can see that the path in (b) is simpler than the other two, which ease the path following task for the snake robot.



(a) (b)

(c)

Figure 7.4: The planned paths in **environment 2**, based on the APF planner. The starting and target points are visualized as blue and red dots, respectively. (a) Path from the APF planner. (b) The APF path filtered by the BPP-filter. (c) The APF path filtered by the SLCL-filter. We can see that the path in (b) is the most suitable for path following purposes.

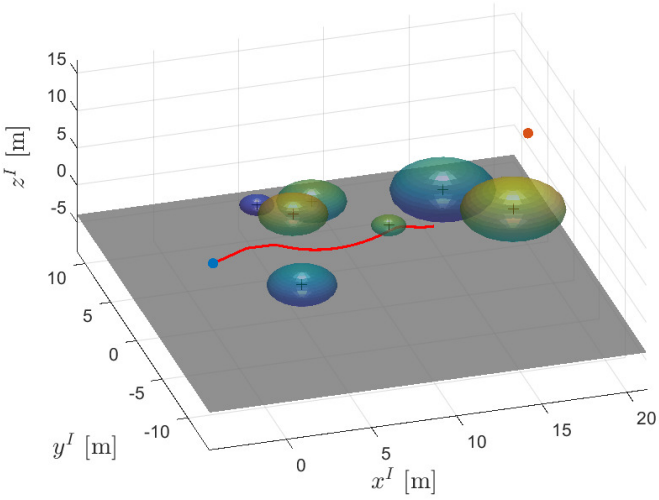


Figure 7.5: The APF planner fails to plan a path in **environment 3** because of the high obstacle density.

7.1.3 RRT* Path Planner

The RRT* path planner implemented for the snake robot acting as an AUV is explicitely searching for a collision-free path in the $\mathcal{W}$-space, a subset of $\mathbb{R}^3$. When searching for a path, the algorithm needs to consider the length of each branch in the tree and the angle between two branches, resulting in a feasible path for the snake robot to follow. To achieve this, the hyperparameters are tuned as shown in Table 7.5, and an elaboration follows here:

- *Branch lower bound*, ϵ_{lower}. The minimum euclidean distance between two connected nodes of the tree. This parameter is set to twice the length of the snake robot, aiming for a path $\mathcal{P}$ that is easier to follow with the WLOS guidance method than for paths consisting of smaller line segments.

- *Branch upper bound*, ϵ_{upper}. The maximum euclidean distance between two connected nodes of the tree. The distance is set to 15[m], which is to prevent the random samples to be placed too far from the target in a potentially wrong direction.

- *Angle lower bound*, υ_{lower}. The minimum angle between two following edges. The smaller the angle the harder it is to prevent overshoot when switching between two lines $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$. This parameter is set to 120° to create a path that does not demand aggressive turn during switching.

- *Neighbourhood radius*, r_{nh}. The algorithm searches for close nodes that might lead to a shorter path, and is here set to 8[m].

Hyperparameter	Numerical value	Unit
ϵ_{lower}	6.8	m
ϵ_{upper}	15	m
υ_{lower}	120	deg
r_{nh}	8	m

Table 7.5: Hyperparameter values for the RRT* path planner.

The RRT* planner manages to plan collision-free paths for all the environments, which is shown in Figure 7.6. Their efficacy in a path following scenario will be further investigated in Section 7.2.

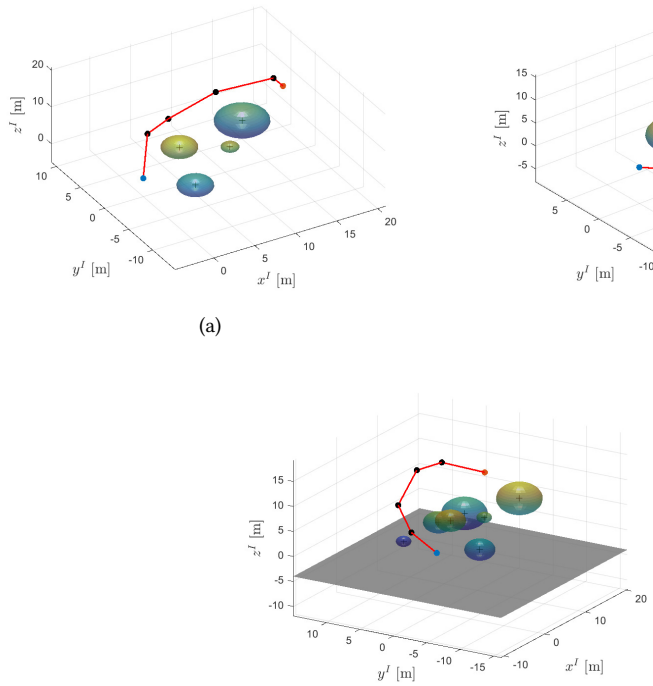


Figure 7.6: The planned paths in all three environments, based on the RRT*-planner. (a) Shows the planned path in **environment 1**. (b) Shows the planned path in **environment 2**. (c) Shows the planned path in **environment 3**.

7.1.4 RRT* vs APF as Global Path Planners

The APF and RRT* has been tested on three environments with different number of obstacles. Both the RRT* planner and the APF planner managed to plan collision-

free paths for **environment 1** and **environment 2**, but only the RRT* successfully accomplished it for **environment 3**. Thus, it seems that the RRT* is better than APF for global path planning the more cluttered the environment is. This makes sense as the APF tends to move straight towards the target, and the more obstacles that lies close to this path, the more it gets pushed away from it. The probability of getting trapped by a local minimum, which is one of the biggest drawbacks of the APF, rapidly increases with the number of obstacles. Even though the *random walk* mode could solve the problem, it was shown in Figure 7.5 that this need not be the case. The RRT* implemented here does not suffer from a local minimum problem, and it is able to exploit the whole domain better than the APF due to its exploring behaviour. Eventually, this is what makes RRT* more suited for path planning in cluttered environments, and the better choice for path planning in **environment 3**.

The APF planner tends to converge straight to the target and create curved paths around the obstacles when it gets close to them. The SLCL-filter and BPP-filter are both trying find waypoints along the path, resulting in a piecewise linear path that is better suited for WLOS guidance. The BPP-filter is additionally filtering the original path with the objective of finding a shorter path that is still collision-free. This method also makes it suited for filtering APF paths where the *random walk* mode has been activated, which the SLCL-filtering does not handle well. The RRT* is also searching for and converges towards the shortest path, but whether or not it finds it depends on the tuning of its hyperparameters, which is not easy to tune in order to find the theoretically shortest feasible path. For **environment 1** and **environment 2**, where both the APF and RRT* planners managed to plan collision-free paths, the shortest distance of a path is in fact planned by the APF planner with BPP-filtering, as shown in table 7.6. In terms of the path distance, the APF planner with BPP-filtering seems to be the best.

Thus, for a sparse cluttered environment, choosing to utilize the APF planner with BPP-filtering would create a shorter and more effective path towards the goal, but the more cluttered the environment gets the better it is to utilize the RRT* planner in favor of the APF planner.

PATH DISTANCES [m]		
	Environment 1	**Environment 2**
APF (BPP)	22.9	17.9
APF (SLCL)	23.6	24.6
RRT*	26.2	25.6

Table 7.6: Distances of the paths planned in **environment 1** and **environment 2** by the APF planner with filtering and the RRT* planner.

7.2 Path Following

Having planned the path, the next step is to make the snake robot sufficiently follow it. The snake robot in this section will follow the path while being straight at all times, meaning that its joint angles will follow a desired angle of 0. To accomplish this task, the guidance laws from Section 5.3 and the controllers from Section 6.1 need to be tuned appropriately. Essentially, we want the snake robot to reach the target and stay clear of the obstacles, and to achieve this the snake robot needs to stay close to the collision-free path at all times. Thus, we aim for guidance laws and controllers that together effectively steer the snake robot onto the path, without too large an overshoot and with fast convergence towards the path. The centre link's CM frame will be used for guidance purposes, which is beneficial in the area of switching from one waypoint to another. If the base frame or the end effector frame were used, the switching would respectively occur later or earlier, relative to when the centre link is used. This would introduce more deviations between the planned path and the actual path travelled by the snake robot, which is closely related to the matter of selecting an appropriate *sphere of acceptance*, addressed in Section 5.2. The numerical values for both the guidance laws and the controllers after the tuning process are given in tables 7.7 and 7.8, respectively. In addition, each joint is controlled by a PD-controller, with gains given as in Table 7.9. The tuning parameters were found through trial and error in order to make the snake robot's behaviour meet the criteria mentioned above, and they are the same for all case simulations, to enable a fair comparison. The paths created in **environment 1** from Section 7.1.1 will be used for path following in all the simulations because the path planning results in this environment were the best for comparison of the path following task. In addition, the subsequent simulations in sections 7.2.1 to 7.2.3 will make use of the slow-region concept from Section 5.3.4, while in Section 7.2.4 the slow-regions are absent.

Tuning parameters	Numerical value	Unit
Δ	6.8	m
μ	0.5	1
κ	0.05	1
r_{soa}	0.5	m
$K_{\text{sr,upper}}$	8	1
r_{sr}	2	m

Table 7.7: Tuning parameters values for the guidance and forward speed laws. The "1" in the **Unit** column specifies dimensionless quantities.

	Tuning parameters	Numerical values
Forward autopilot	$k_{p,u}$	70
	$k_{i,u}$	5
	$k_{d,u}$	0
Pitch autopilot	$k_{p,\theta}$	30
	$k_{i,\theta}$	0
	$k_{d,\theta}$	60
Heading autopilot	$k_{p,\psi}$	25
	$k_{i,\psi}$	0
	$k_{d,\psi}$	60

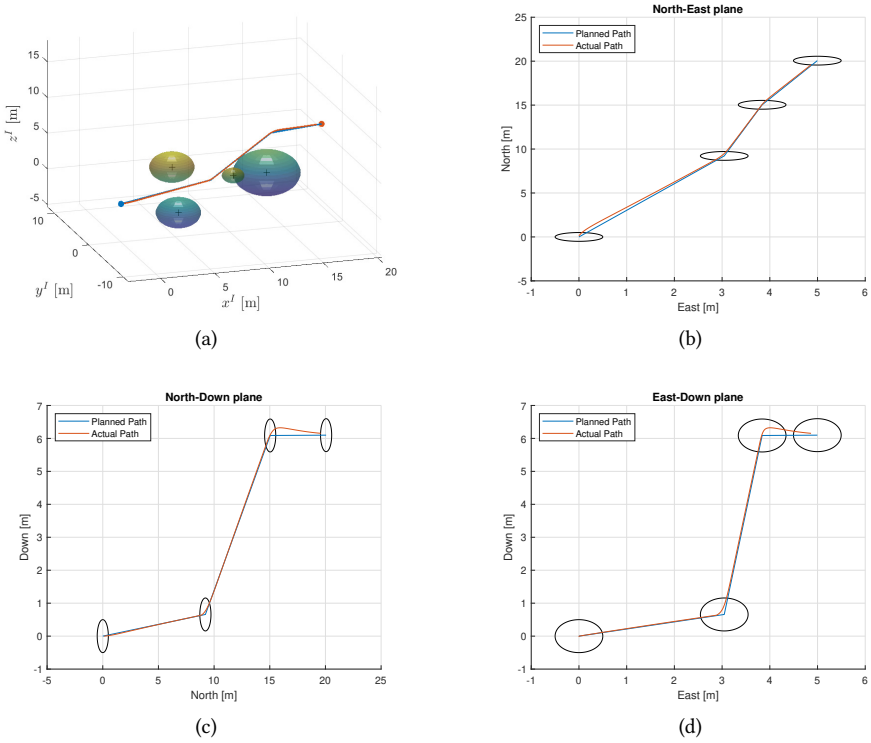
Table 7.8: Tuning parameters values for the autopilot controllers.

Tuning parameters	Numerical value
$k_{p,q}$	300
$k_{d,q}$	20

Table 7.9: Tuning parameters values for the joint PD-controllers.

7.2.1 Case 1: Following BPP-Filtered Path

The results from the path following method executed on the planned path from the BPP-filtering are summarized through figs. 7.7 to 7.11. In Figure 7.7 it is seen that the snake robot is able to follow the planned path quite accurately, which is necessary to avoid the obstacles. The distances from the obstacles to each link's rear end and the end effector are visualized in Figure 7.8, which shows that the snake robot is in fact avoiding all the obstacles while following the path. Furthermore, Figure 7.9 shows how the snake robot is able to track the desired *surge*, *pitch* and *heading* along the path. Note that when the snake robot enters the low velocity region it slows down in order to better follow the path and avoids large overshooting behaviour onto the path. Additionally, Figure 7.10 shows that the joints are kept close to 0° during the whole path following operation and the joint torques do not exceed their 10[Nm] limit. There are tiny motions in some of the joint angles when the snake robot performs larger turns, where the magnitude of the thruster forces are greatest, but they are kept within 1° and do not hinder the path following operation's overall performance. The thruster forces seen in Figure 7.11 shows that the thrusters are never demanded to provide excessively high forces and operate within their 40[N] limits at all times.



(a) (b) (c) (d)

Figure 7.7: The BPP-filtered path and the actual path of the centre link's CM. (a) The whole 3D environment. (b) The projection onto the North-East plane. (c) The projection onto the North-Down plane. (d) The projection onto the East-Down plane. The *sphere of acceptance* is visualized as black ellipses in each plane. Note that the difference in the axes make the circles appear elliptical.

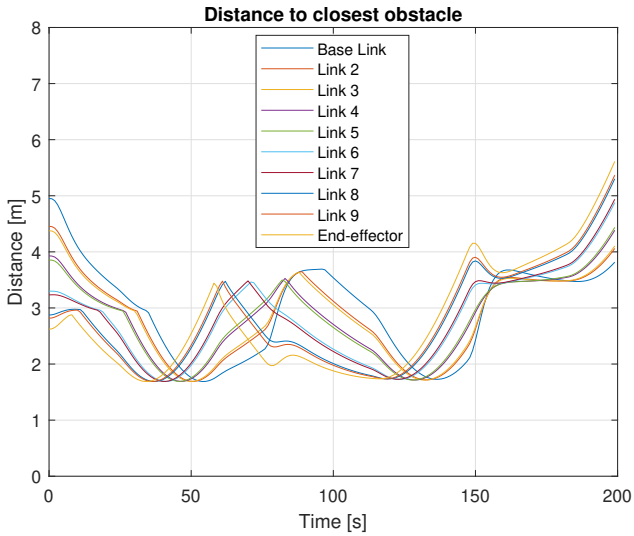


Figure 7.8: The closest distance between the obstacles in the $\mathcal{W}$-space and the links and end effector of the snake robot. It is seen that all the links and the end effector keep their distance from the obstacles when following the BPP-filtered path. Note that since the longest link is 0.8 metres, collision avoidance is guaranteed, as each calculated distance is above 0.8.

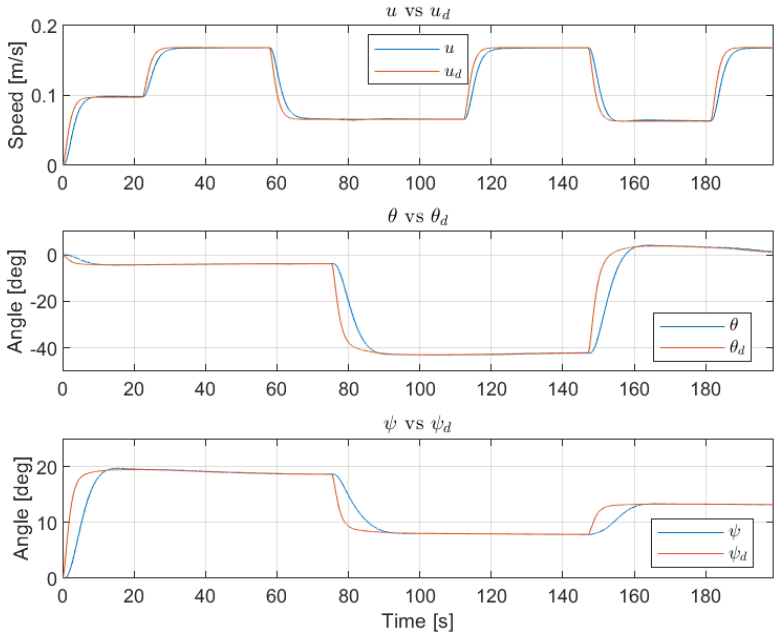
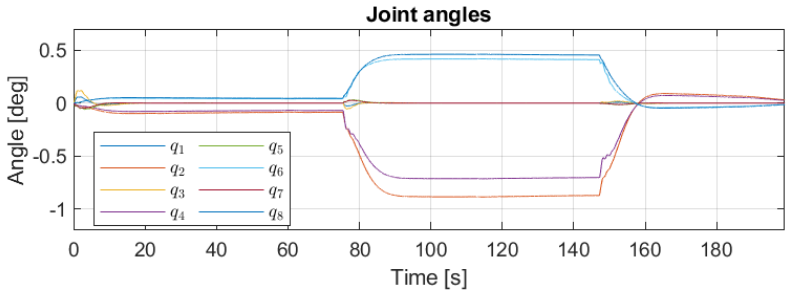
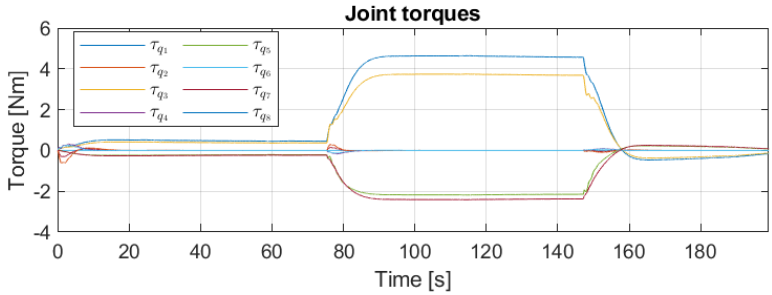


Figure 7.9: The desired and actual *surge*, *pitch* and *heading* states of the snake robot while following the BPP-filtered path. It is seen that their respective autopilots are tuned such that snake robot are able to follow the references in a reasonable amount of time.



(a)



(b)

Figure 7.10: The joint angles and joint torques during path following of the BPP-filtered path. (a) Shows that the joint angles are kept relatively close to 0° during the whole path following operation. When the snake robot use its thrusters to turn one can see some of the joints move slightly, but are kept within 1°. (b) Shows the joint torques used to keep the joints close to 0°.

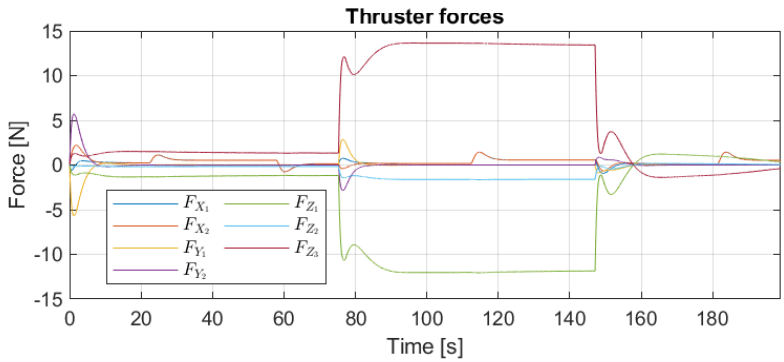


Figure 7.11: The thruster forces during path following of the BPP-filtered path. It is seen that thrusters Z_1 and Z_3 are the most active, in order to follow the demanded change in *pitch* about 40° after approx. 80 seconds.

7.2.2 Case 2: Following SLCL-Filtered Path

The results from the path following scheme executed on the planned path from the SLCL-filtering are summarized through figs. 7.12 to 7.16. In Figure 7.12 it is seen that the snake robot is also here able to follow the planned path quite accurately. The distances from the obstacles to each link's rear end and the end effector are visualized in Figure 7.13, which shows that the snake robot is in fact avoiding all the obstacles while following the path. Furthermore, Figure 7.14 shows how the snake robot is able to track the desired *surge*, *pitch* and *heading* along the path, but note that the reference values change more frequently than the ones in Figure 7.9. This is due to the fact that the SLCL-filtered path consists of a higher number of waypoints, which in turn demands the snake robot to execute more maneuvers to follow the path, compared to what was needed for following the BPP-filtered path. Due to the smaller distance between the waypoints, the angle between $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$ vary less across the entire path, which makes the scaling factor K_{sr} in the slow-region vary less. Thus, the resulting difference between the highest and lowest desired speed is smaller than for the BPP-filtered path, which is seen by comparing the desired *surge* in Figure 7.9 and Figure 7.14. Additionally, Figure 7.15 shows that the joints are kept close to 0° during the whole path following operation and the joint torques do not exceed their 10[Nm] limit. Similar to the path following of the BPP-filtered path in Section 7.2.1, some of the joint angles deviate from the reference when the snake robot performs large turns, where the magnitude of the thruster forces are greatest. The deviations are kept small and do not hinder the path following operation's overall performance. The thruster forces seen in Figure 7.16 shows that the thrusters are not demanded to provide excessively high forces and operate within their 40[N] limits at all times. Nevertheless, they are larger and more varying than the ones seen in Figure 7.11. This makes sense because the snake robot needs to execute more turning maneuvers to adequately follow the path.

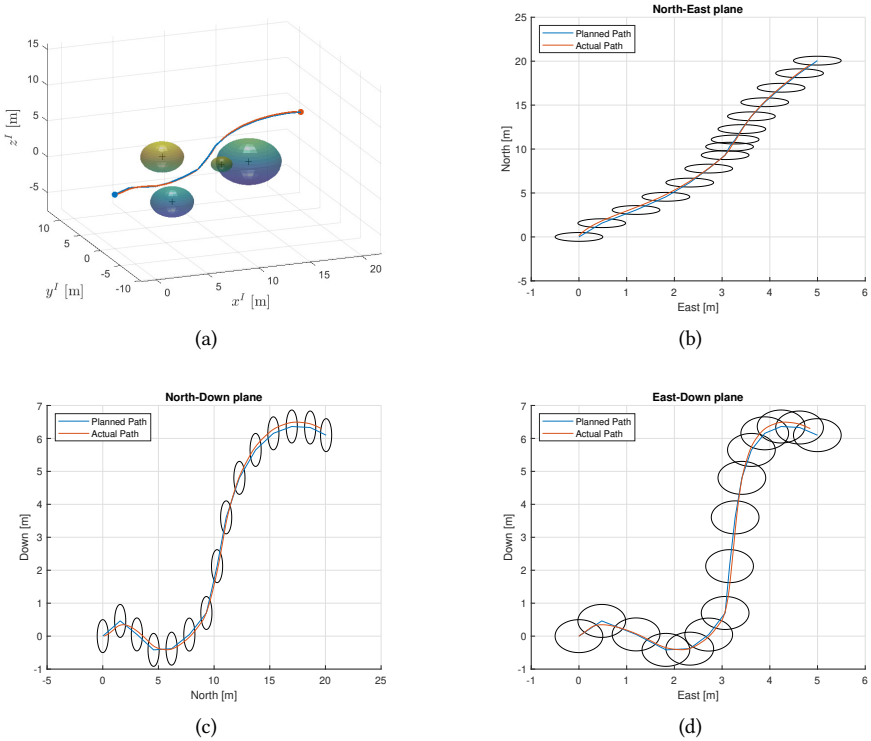


Figure 7.12: The SLCL-filtered path and the actual path of the centre link's CM. (a) The whole 3D environment. (b) The projection onto the North-East plane. (c) The projection onto the North-Down plane. (d) The projection onto the East-Down plane. The *sphere of acceptance* is visualized as black ellipses in each plane. Note that the difference in the axes make the circles appear elliptical.

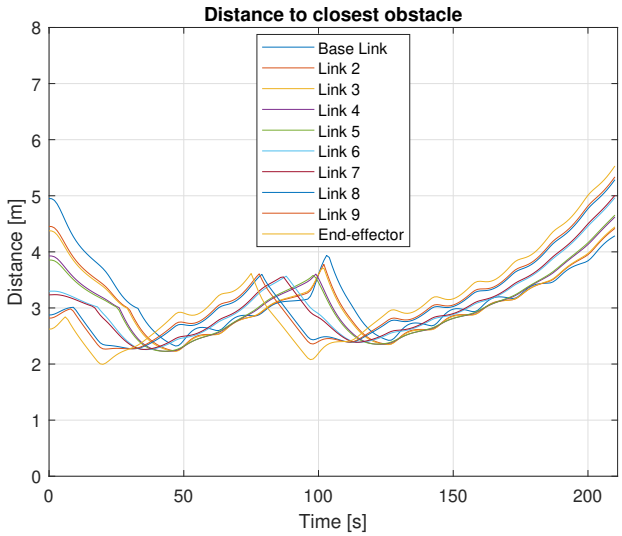


Figure 7.13: The distance between the obstacles in the $\mathcal{W}$-space and the links and end effector of the snake robot. It is seen that all the links and the end effector keep their distance from the obstacles when following the SLCL-filtered path. Note that since the longest link is 0.8 metres, collision avoidance is guaranteed, as each calculated distance is above 0.8.

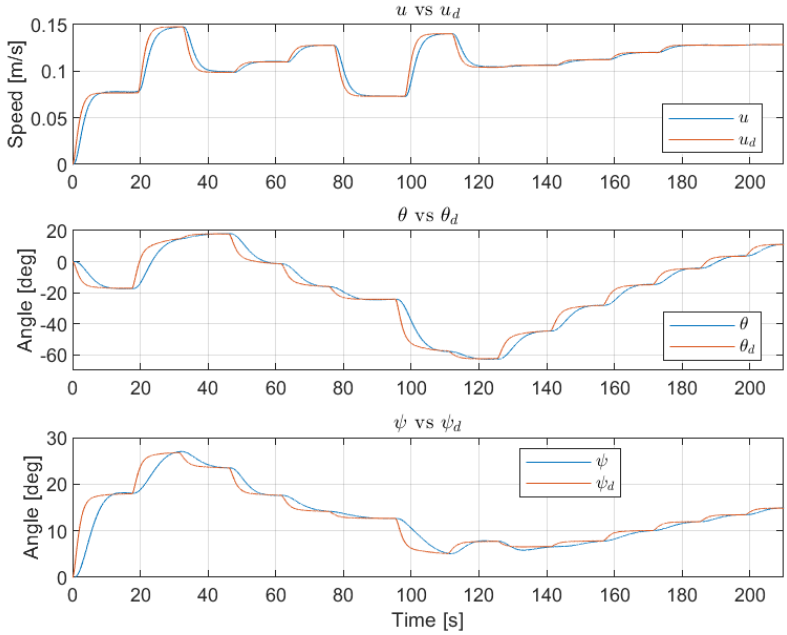
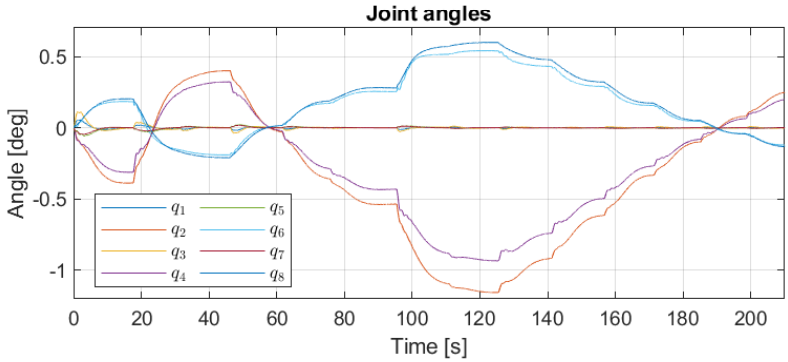
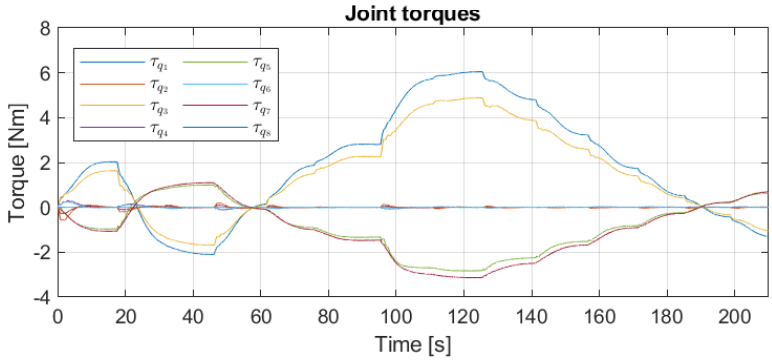


Figure 7.14: The desired and actual *surge*, *pitch* and *heading* states of the snake robot while following the SLCL-filtered path. It is seen that it manages to obtain the desired *surge*, *pitch* and *heading* values without lagging much behind.



(a)



(b)

Figure 7.15: The joint angles and joint torques during path following of the SLCL-filtered path. (a) Shows that the joint angles are kept relatively close to 0° during the whole path following operation. When the snake robot use its thrusters to turn we can see some of the joints move slightly, barely past 1° for q_2. (b) Shows the joint torques used to keep the joints close to 0°.

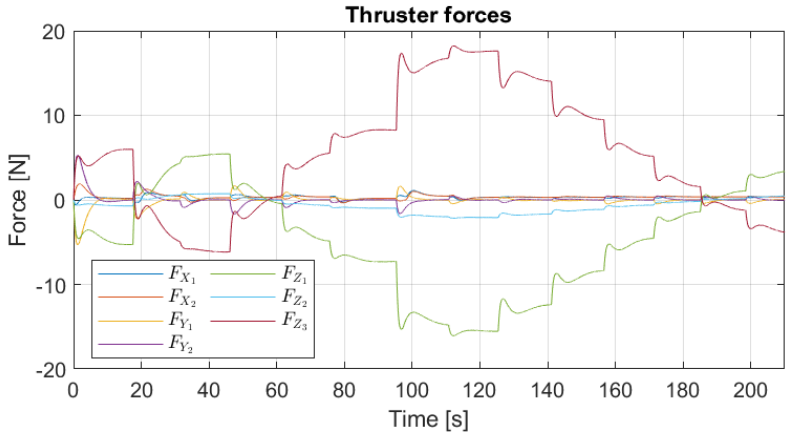


Figure 7.16: The thruster forces during path following of the SLCL-filtered path. It is seen that thrusters Z_1 and Z_3 are the most active, in order to follow the demanded change in *pitch* of about $40°$ after approx. 80 seconds.

7.2.3 Case 3: Following RRT* Path

The results from the path following scheme executed on the planned path from the RRT* algorithm are summarized through figs. 7.17 to 7.21. In Figure 7.17 we see that the snake robot sufficiently follows the planned path, and as a direct result avoids the obstacles. The distances from the obstacles to each link's rear end and the end effector are visualized in Figure 7.18. Furthermore, Figure 7.19 shows how the snake robot is able to track the desired *surge*, *pitch* and *heading* along the path. Note that when the snake robot enters the slow-region it slows down proportionally to the size of the angle between $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$ in order to better follow the path and avoid large overshooting behaviour onto it. Additionally, fig. 7.20 shows that the joints are kept close to 0° during the whole path following operation and the joint torques do not exceed their 10[Nm] limit. There are tiny motions in some of the joint angles when the snake robot makes large turns, where the thruster forces are large, but they are kept within 1° and do not hinder the path following operation's overall performance. The thruster forces seen in Figure 7.21 are not at any point demanded to output excessively high forces and operate within their 40[N] limits at all times.

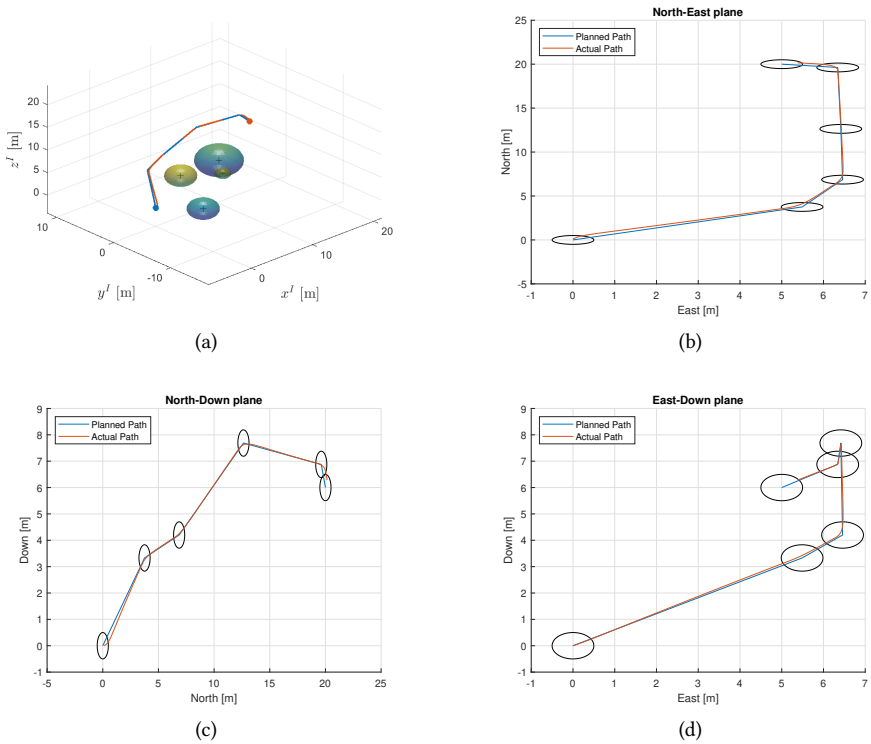


Figure 7.17: The RRT* path and the actual path of the centre link's CM. (a) The whole 3D environment. (b) The projection onto the North-East plane. (c) The projection onto the North-Down plane. (d) The projection onto the East-Down plane. The *sphere of acceptance* is visualized as black ellipses in each plane. Note that the difference in the axes make the circles appear elliptical.

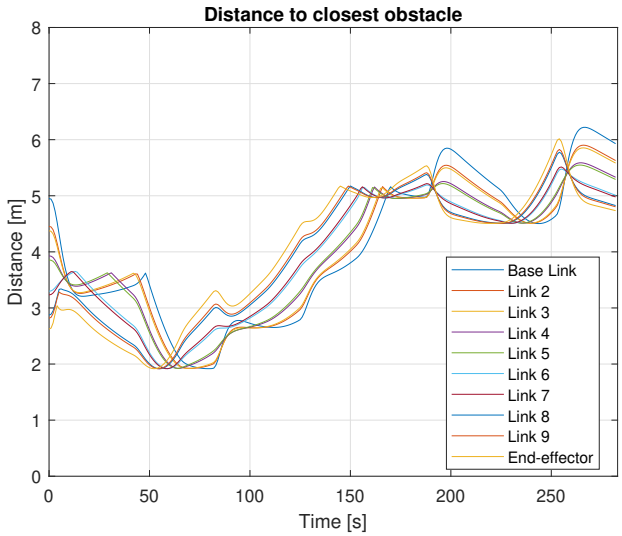


Figure 7.18: The distance between the obstacles in the $\mathcal{W}$-space and the links and end effector of the snake robot. It is seen that all the links and the end effector keep their distance from the obstacles when following the RRT* path. Note that since the longest link is 0.8 metres, collision avoidance is guaranteed, as each calculated distance is above 0.8.

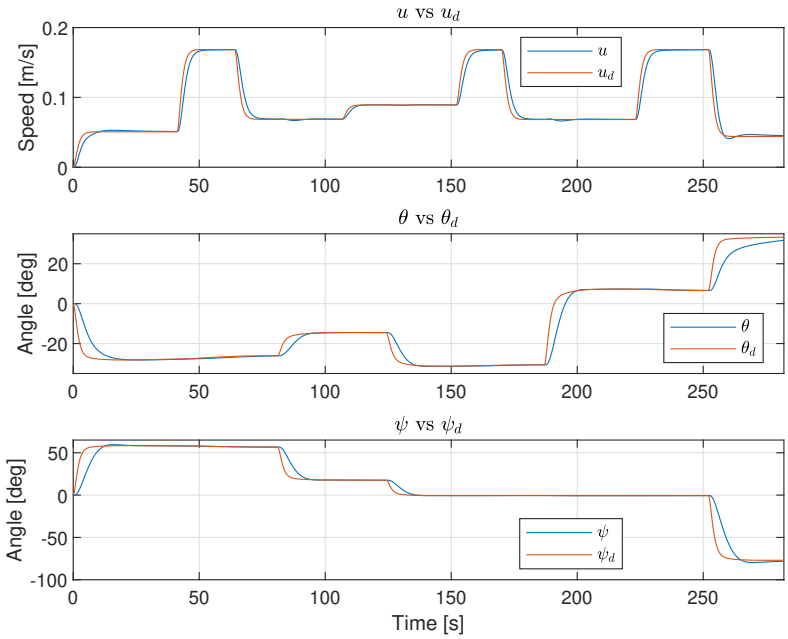
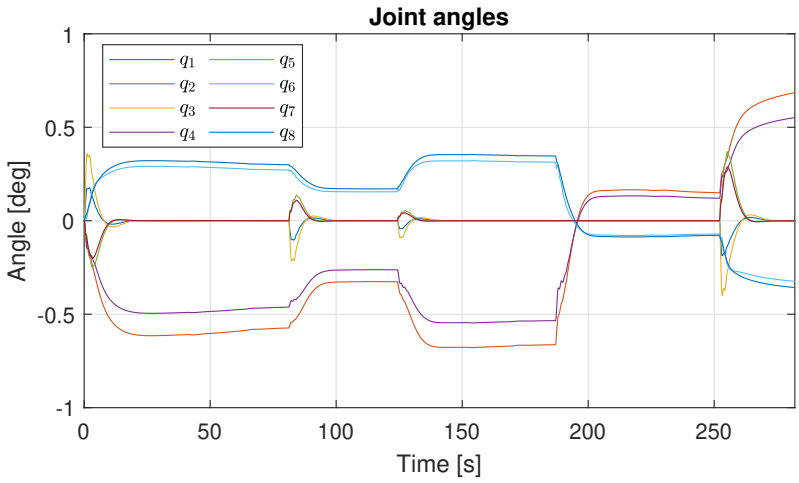
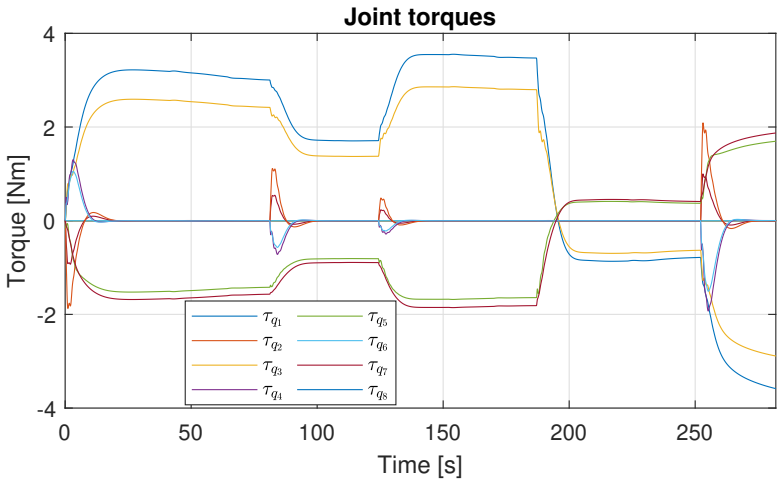


Figure 7.19: The desired and actual *surge*, *pitch* and *heading* states of the snake robot while following the RRT* path.



(a)



(b)

Figure 7.20: The joint angles and joint torques during path following of the RRT* path. (a) Shows that the joint angles are kept relatively close to 0° during the whole path following operation. When the snake robot use its thrusters to turn one can see some of the joints moves slightly, but are kept within 1°. (b) Shows the joint torques used to keep the joints close to 0°.

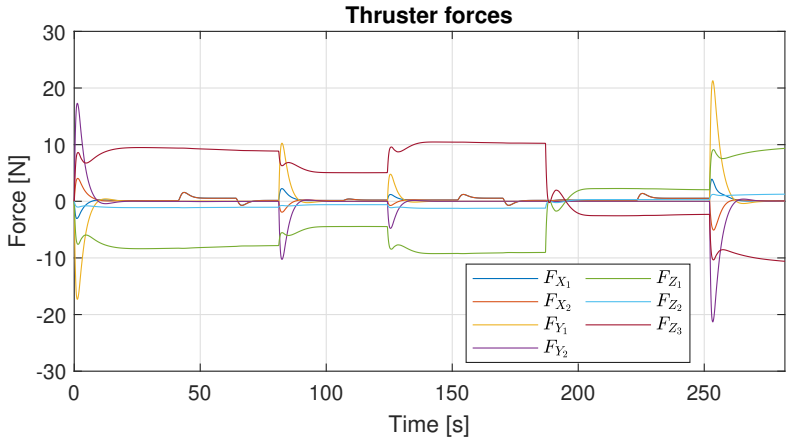


Figure 7.21: The thruster forces during path following of the RRT* path. Note that the thruster forces are largest at the beginning and the end of the path, where the snake robot's orientation vary the most from the desired orientation of the line segment it's set to follow.

7.2.4 Case 4: Absence of Slow-Region

In this case the same paths from sections 7.2.1 to 7.2.3 are used for path following, but here the slow-region is absent. For all three path following scenarios the consequence is that the resulting paths in total deviates more from the planned path compared to the path following results presented in sections 7.2.1 to 7.2.3, which is seen in Figure 7.22, Figure 7.24 and Figure 7.26. The BPP-filtered path is experiencing its worst overshooting behaviour at the last line segment, while the SLCL-filtered path has more or less the same overshooting behaviour through the whole path. Even though the path following task for these two scenarios overshoot and oscillates more about the planned path, they still manages to follow the path and reach the target. They also avoid colliding with obstacles, which is seen in Figure 7.28, though the distances to the objects are slightly closer compared to the results from sections 7.2.1 and 7.2.2. This is expected, as the closer the snake robot follows the path, the bigger the distance to the obstacles will be. When the speed is increased, the more difficult it is to stay on the path, leading to smaller distances to the obstacles. Staying close to the path is especially difficult when switching from one line segment to another, where bigger turning maneuvers are required. The biggest deviations from the planned path is thus found around the waypoints. Note that WLOS guidance method barely manages to guide the snake robot to the target while following the SLCL-filtered path. This could be seen from figs. 7.24 and 7.25, where the snake robot almost fails to visit some of the waypoints and struggles to follow the *heading* reference, respectively. The RRT* planned path experience the same increase in overshooting behaviour, but in this case it results in the snake robot not being able to respond fast enough to enter the *sphere of acceptance* and fails to reach the target. Thus, it is seen that if both the angle between the lines $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$ and the length of $\mathcal{P}_{i+1}$ are too small at the same time, the snake robot could fail to visit the waypoint. If this happens, the WLOS method fails in guiding the snake robot towards the target. A solution could be to have a larger *sphere of acceptance* radius, but here the task is to avoid obstacles by following a collision-free path, thus we would not want the radius to be too big. If the radius is too big the actual path would deviate more from the planned path due to the switching

from $\mathcal{P}_i$ to $\mathcal{P}_{i+1}$ happens earlier. The presence of a slow-region is thus shown to make the WLOS guidance method more robust regarding the shape of the paths planned. This is accomplished because the snake robot is set to slow down according to the size of the angle between the lines $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$.

It should also be mentioned that as an alternative to introducing a slow-region, the lookahead-distances Δ and μ can also be tuned different to deal with the problem encountered here. If for instance Δ is lowered compared to its value in these simulations the more aggressively the snake robot steers onto each line segment. The more aggressive the tuning, the more oscillating the snake robot's convergence onto the path will be, which could lead to collisions if the oscillations are too big. If this method is utilized the speed should be smaller to prevent overshooting behaviour. In addition, there exists methods utilizing time-varying lookahead-distances for the purpose of fast convergence onto the path and at the same time avoid overshooting behaviour [10]. The slow-region concept was instead implemented here to cope with the problem in a smaller area around the waypoints, where the overshooting behaviour occured. The slow-region also utilizes information about the shape of the path to slow down accordingly, which was shown to improve the WLOS guidance method's overall performance in these path following cases.

The slow-region and varying lookahead-distances are both possible solutions to prevent the WLOS method of failing. An alternative to these solutions could be to combine the WLOS path following method with a set-based method, as was done in [17]. The snake robot could start in WLOS mode and follow the planned path, but if it fails to visit a waypoint it could potentially collide with obstacles while trying to navigating back to the path. In these situations, the snake robot could switch to a set-based mode, which leads it back to the unvisited waypoint unharmed. The set-based method does not necessarily need to guide the snake robot back to the lost waypoint, but could choose from any of the waypoints or even go straight towards the target. What would be the best solution depends on the task. Such a method is not implemented in this thesis, but could definitely make the guidance method more robust and safer in terms of obstacle avoidance and reaching the target.

In the cases presented in this subsection, the WLOS guidance scheme successfully guided the BPP-filtered and the SLCL-filtered paths to the target, but failed for the RRT* planned path. This is related to how the paths are formed and the tuning of the guidance laws, including the *sphere of acceptance*. As the desired size of the *sphere of acceptance* is relatively small in order for the WLOS guidance scheme to closely follow the path, the relation between the hyperparameters of the RRT* planner need to be more carefully considered when the slow-region is absent. The APF planner creates a more deterministic curved path that can easily be divided into feasible waypoints to be used in the WLOS guidance, through either BPP-filtering or SLCL-filtering, which performs better without slow-regions than the paths created by RRT*.

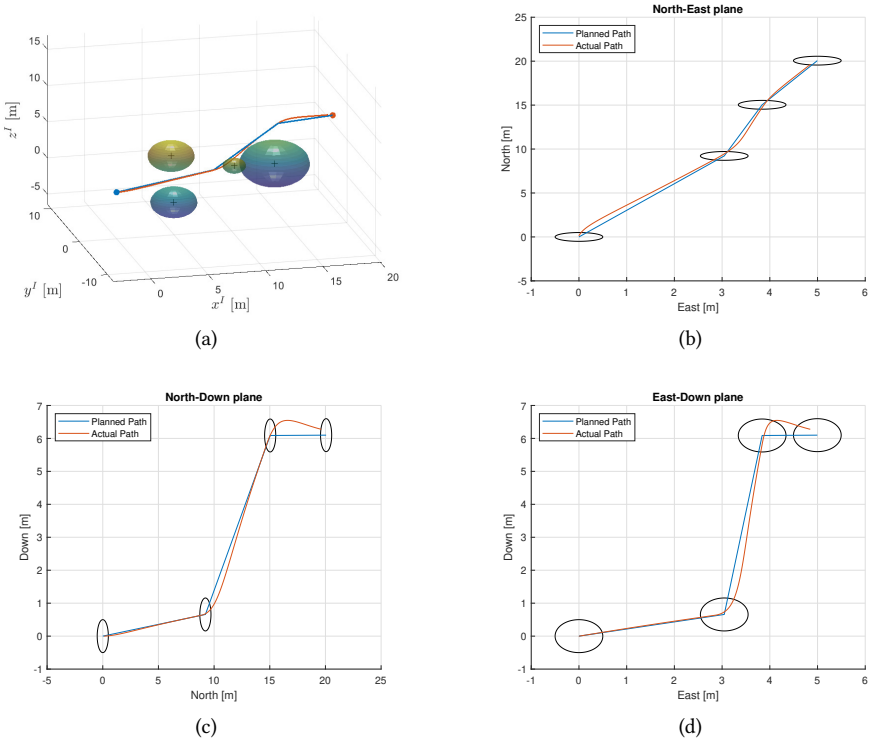


Figure 7.22: The BPP-filtered path and the actual path of the centre link's CM when the slow-region is absent. (a) The whole 3D environment. (b) The projection onto the North-East plane. (c) The projection onto the North-Down plane. (d) The projection onto the East-Down plane. The *sphere of acceptance* is visualized as black ellipses in each plane. Note that the difference in the axes make the circles appear elliptical.

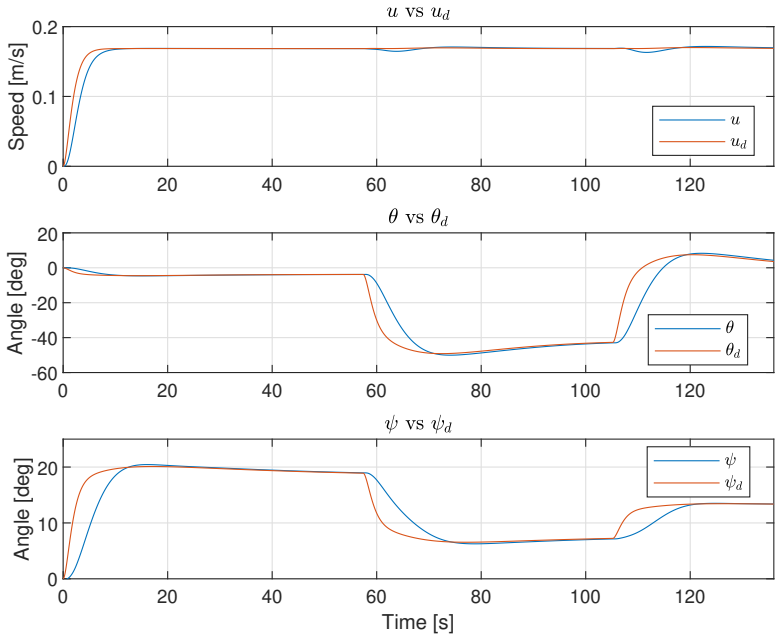


Figure 7.23: The desired and actual *surge*, *pitch* and *heading* states of the snake robot when following the BPP-filtered path and the slow-region is absent. Notice that the desired *surge* is constant after 10 seconds.

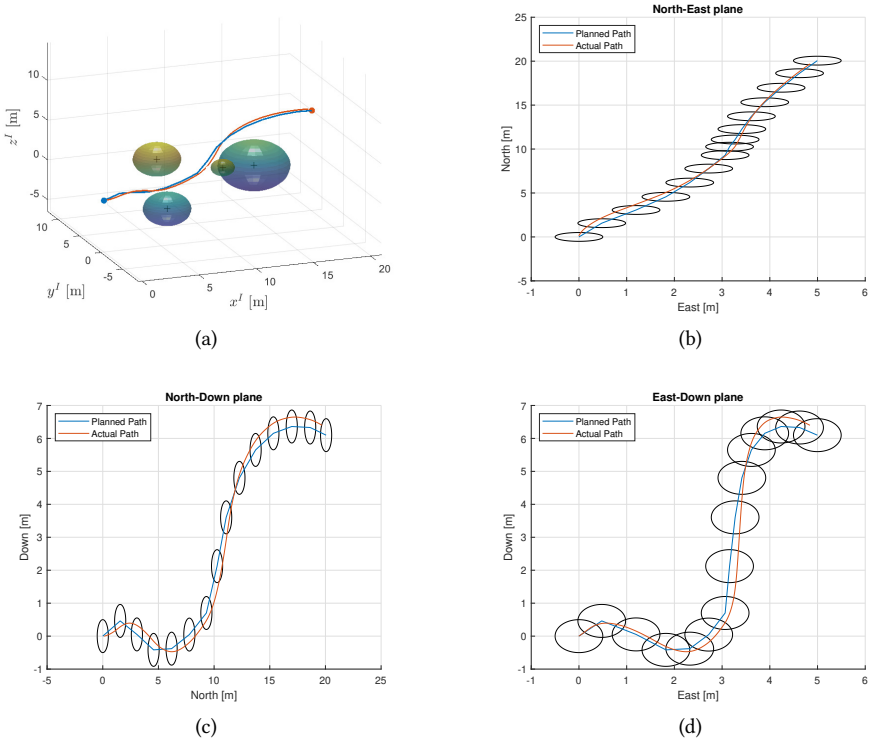


Figure 7.24: The SLCL-filtered path and the actual path of the centre link's CM when the slow-region is absent. (a) The whole 3D environment. (b) The projection onto the North-East plane. (c) The projection onto the North-Down plane. (d) The projection onto the East-Down plane. The *sphere of acceptance* is visualized as black ellipses in each plane. Note that the difference in the axes make the circles appear elliptical.

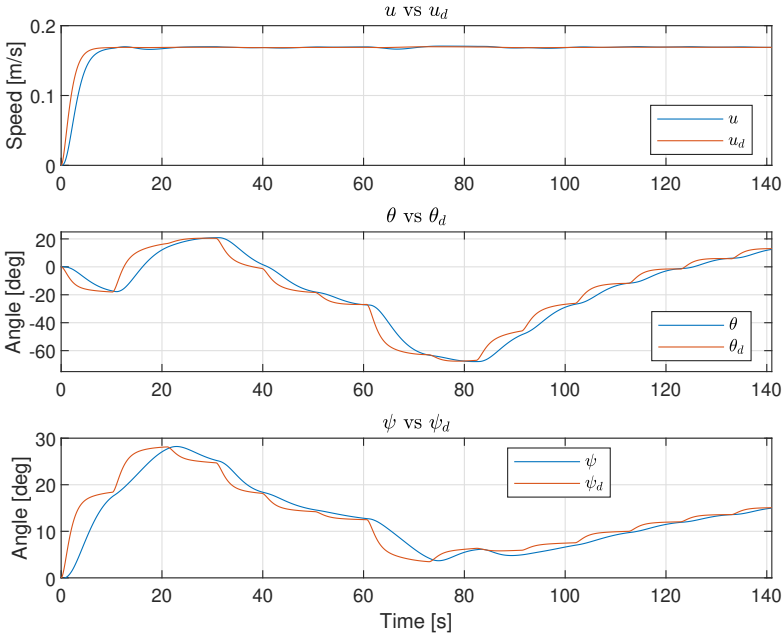
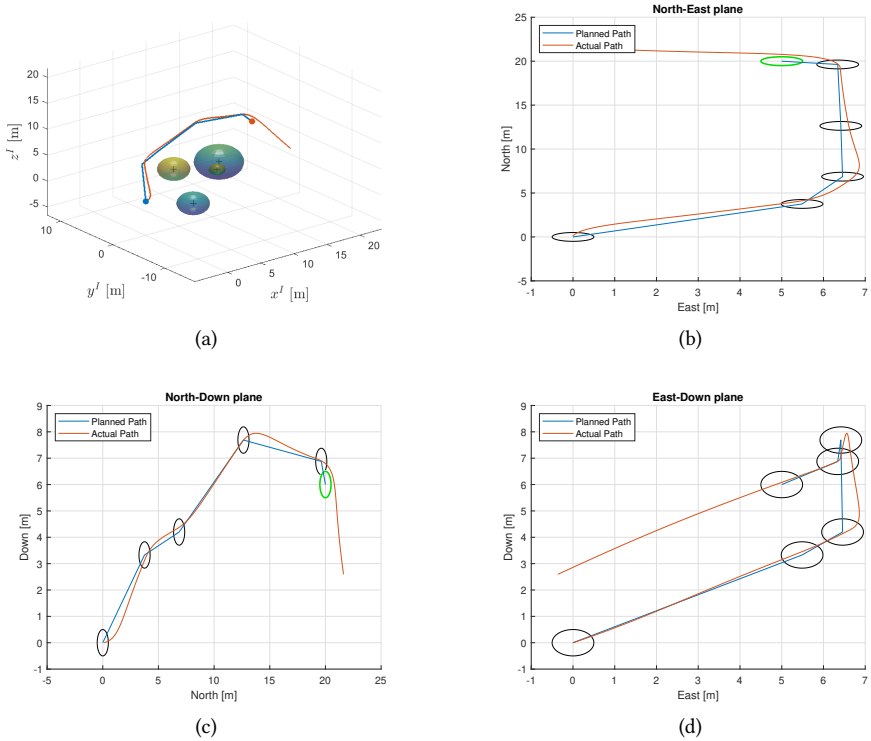


Figure 7.25: The desired and actual *surge*, *pitch* and *heading* states of the snake robot when following the SLCL-filtered path and the slow-region is absent. Notice that the desired *surge* is constant after 10 seconds and that its magnitude results in the *heading* barely reaching its desired value before a new *heading* reference is demanded.



(a) (b) (c) (d)

Figure 7.26: The RRT* path and the actual path of the centre link's CM when the slow-region is absent. (a) The whole 3D environment. (b) The projection onto the North-East plane. (c) The projection onto the North-Down plane. (d) The projection onto the East-Down plane. Note that the snake robot fails to enter the green *sphere of acceptance*, which causes the WLOS guidance to fail.

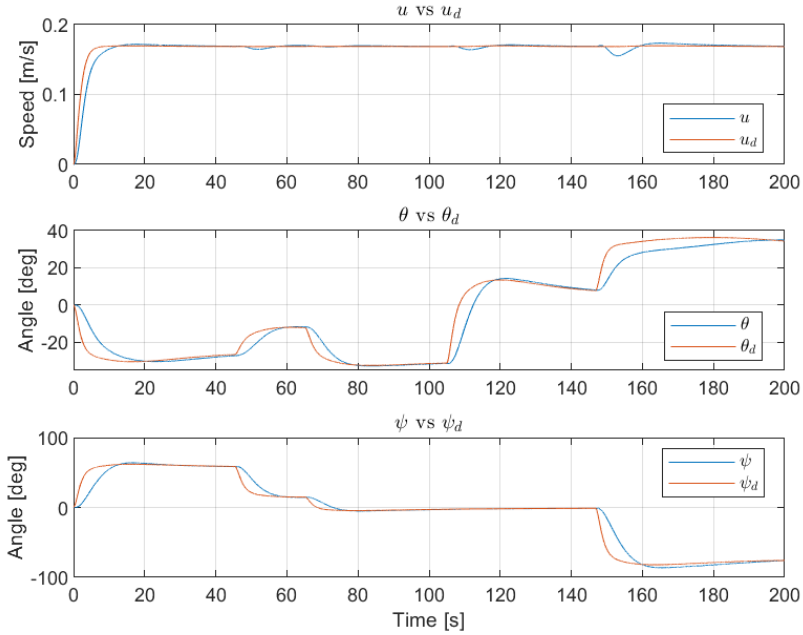


Figure 7.27: The desired and actual *surge*, *pitch* and *heading* states of the snake robot when following the RRT* path and the slow-region is absent. Notice that the desired *surge* is constant after 10 seconds.

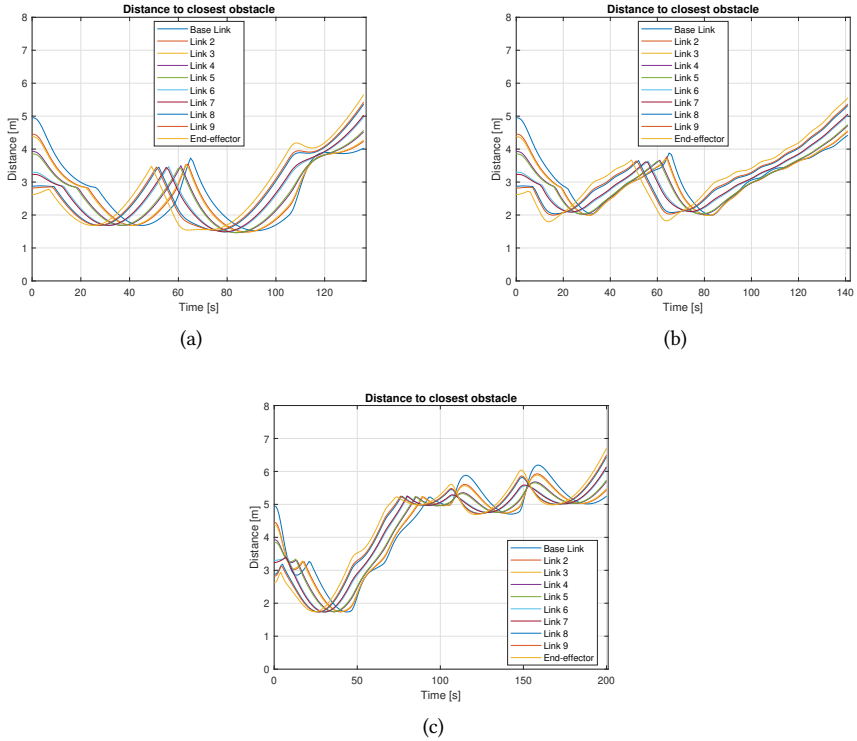


Figure 7.28: The distance between the obstacles in the $\mathcal{W}$-space and the links and end effector of the snake robot when the slow-region is absent. It is seen that all the links and the end effector keep their distance from the obstacles when following (a) BPP-filtered path, (b) SLCL-filtered path and (c) RRT* path. Note that since the longest link is 0.8 metres, collision avoidance is guaranteed, as each calculated distance is above 0.8.

7.2.5 RRT* vs APF for Path Following

The three planned paths in **environment 1**, created by the RRT* planner and the APF planner with both BPP-filtering and SLCL-filtering, have been used for path following throughout Section 7.2. The WLOS guidance method, which incorporated a slow-region for safer path following, was utilized together with PID-based autopilot controllers for *surge*, *pitch* and *heading*, respectively in the form of PI, PD and PD. The path following task was satisfyingly executed as the snake robot was able to closely follow the paths while travelling from its starting position to the target position, without colliding with the obstacles. Thus, all the paths were proven functional in combination with the proposed guidance and control methods to solve the collision avoidance task. We also want to compare them in terms of energy efficiency because this matter is of high practical importance. Being energy efficient while executing various tasks, for instance travelling safely from A to B, saves fuel and money. If the path chosen also is one of the shortest possible and easy to follow, we may also save a significant amount of time. In Table 7.10 we see the total work done by the thrusters throughout the three path following tasks, from where we can conclude that the BPP-filtered path is best wrt energy efficient operation for this environment. It is important to emphasize that it's the best option for this exact environment and tuning choice of the hyperparameters, as another environment and other values of the hyperparameters could make the paths look very different. Nevertheless, the BPP-filtering in combination with the APF worked as intended, creating an easy to follow and short path, resulting in being the most energy efficient.

	Total thrust work [J]	**Path distance [m]**	**Work pr. metre [J/m]**
Case 1	1240	24.1	51.5
Case 2	3140	25.3	124.1
Case 3	4180	30.9	135.3

Table 7.10: Total thruster work, path distance and work done pr.metre from start position to target position for **Case 1-3**.

Chapter 8

Conclusion and Future Work

This chapter first provides a summary of the results obtained in this thesis. Secondly, Section 8.2 elaborates on potential future work that can enhance the effectiveness of the solutions presented and other required measures to acquire self-collision avoidance.

8.1 Conclusion

In this thesis, the collision avoidance task for a snake robot operating in 3-dimensional cluttered environments was conducted. The task of collision avoidance for the snake robot was two-fold; avoiding obstacles in an environment and self-collision avoidance. A solution to the obstacle avoidance problem was provided and analyzed for the given environments, while the self-collision avoidance remains an unresolved problem.

The three cluttered environments investigated in the thesis consisted of a various number of spherical objects and flat surfaces. To solve the obstacle avoidance task in these environments, this thesis aimed at planning straight line paths for path following, by utilization of the WLOS guidance method in combination with autopilot controllers.

To plan the paths both the APF wit *random walk* mode and RRT* algorithms were used as path planners, creating collision-free paths from a starting point to a desired target point in the environments. In addition, the planners searched for easily navigable and short paths, as they would be better suited for the WLOS method and more optimal regarding energy consumption and time efficiency. To suit these criteria, the APF planned paths were filtered by two methods; the BPP-filter and the SLCL-filter. The APF planner with *random walk* mode managed to plan paths for 2/3 environments, failing at the most cluttered environment. In the environments where the APF planner managed to plan feasible paths, the BPP-filter was shown to provide shorter and easier navigable paths than the SLCL. Additionally, the SLCL-filter failed to create navigable paths where the *random walk* mode had been activated. The RRT* was shown to handle all three environments well, in terms of finding a collision-free path, but the paths were longer than the ones planned by the APF-filters. In sum, whe comparing the APF and RRT* methods, the APF planner was better at finding shorter paths in sparser environments, but the RRT* performed better the more cluttered the environments were. The paths were subsequently followed by the snake robot, utilizing a WLOS lookahead-based guidance law in combination with PID-based autopilot controllers for *surge*, *pitch* and *heading*, all with *feed-forward*, and PD-controlled joints. The WLOS method incorporated a slow-region, which was shown to increase the path following performance and the method's robustness and ability to handle different paths. The guidance laws and the controllers were shown capable of guiding the snake robot from the starting point to the target point, without colliding with the obstacles. The ability to efficiently follow the paths were also analyzed, where the BPP-filtered path was shown to be the most energy efficient in the environments investigated.

The main results from this thesis are the ability to off-line plan global collision-free piecewise linear paths, which in turn can be utilized to guide the snake robot from its starting point to a desired target point in a collision avoiding manner.

8.2 Future Work

The proposed method shows it's collision-free guidance capability in a 3-dimensional cluttered environment. Even though it shows promising results, the problem of the WLOS method when missing a waypoint should be handled to ensure safe operations. To make the proposed pipeline more robust, a suggested solution is to incorporate set-based methods, like the ones in [17], as was discussed in Section 7.2.4. Furthermore, in order to exploit the shape-shifting ability of the snake robot, two approaches should be further investigated. The first one suggests implementing a follow-the-leader approach, based on [34] and [6], where the constraints of the snake robot in configuration space is explicitly accounted for in the planning process, and thus handles both collision avoidance with obstacles in the environment and self-collisions. The second approach utilizes the exact same path planning process as was used in this thesis, but here the task is also to minimize the path-to-end-effector and the path-to-base errors. In order to prevent self-collisions while navigating on the path in this manner, it would be interesting to utilize a reactive-based form of the APF-method, where each link of the snake robot sees the other links as dynamic obstacles in the environment. By utilizing such an approach the path planning margins can be smaller, making the proposed planners able to plan paths in even more confined spaces.

Appendix A

Algorithms

This appendix contains pseudo-code for the *gradient descent*, *random walk*, BPP and RRT* algorithms used for the implementation of the path planner.

A.1 Gradient Descent

Algorithm 1 Gradient Descent

Input: A means to compute the gradient $\nabla U(\boldsymbol{p})$ at a point $\boldsymbol{p}$
Output: A sequence of points $\{\boldsymbol{p}(0), \boldsymbol{p}(1), ..., \boldsymbol{p}(i)\}$

$\boldsymbol{p} = empty\ array$
$\boldsymbol{p}(0) = \boldsymbol{p}_{\text{start}}$
$i = 0$
while $||\nabla U(\boldsymbol{p}(i))|| > \epsilon$ **do**
    $\boldsymbol{p}(i+1) = \boldsymbol{p}(i) + \alpha(i)\nabla U(\boldsymbol{p}(i))$
    $i = i + 1$
end while
return $\boldsymbol{p}$

A.2 Random Walk

Algorithm 2 Random Walk

Input: A point trapped in local minimum $\boldsymbol{p}_{\text{trapped}}$, number of steps to walk N_{rw}, random step length δ_{rw}, obstacles $\mathcal{O}$ and safe radius r_{safe}
Output: A sequence of random points, $\{\boldsymbol{p}(0), \boldsymbol{p}(1), ..., \boldsymbol{p}(i)\}$, leading out of the local minimum

$\boldsymbol{p} = empty\ array\ of\ dimension\ N_{\text{rw}} \times 3$
$\boldsymbol{p}(0) = \boldsymbol{p}_{\text{trapped}}$
$i = 1$
while $i < N_{\text{rw}}$ **do**
  $\boldsymbol{p}_{\text{random}} = \boldsymbol{p}(i-1) + randomStep(\delta_{\text{rw}})$
  $l_{length} = ||\boldsymbol{p}(i-1) - \boldsymbol{p}_{\text{random}}||$
  $l_{direction} = \frac{\boldsymbol{p}(i-1) - \boldsymbol{p}_{\text{random}}}{l_{length}}$
  if noCollisions$(\boldsymbol{p}(i-1), l, \mathcal{O}, r_{\text{safe}})$ **then**
    $\boldsymbol{p}(i) = \boldsymbol{p}_{\text{random}}$
    $i = i + 1$
  end if
end while
return $\boldsymbol{p}$

A.3 Backtracking Path Planner (BPP)

Algorithm 3 Backtracking Path Planner (BPP)

Input: Collision free path $\mathcal{P}$, obstacles $\mathcal{O}$ and safe radius r_{safe}
Output: A sequence of waypoints $\{\boldsymbol{wp}(0), \boldsymbol{wp}(1), ..., \boldsymbol{wp}(i)\}$

$\boldsymbol{wp} = empty\ array$
$\boldsymbol{wp_1} = start\ point\ of\ \mathcal{P}$
$\boldsymbol{wp_2} = end\ point\ of\ \mathcal{P}$
$l_{length} = ||\boldsymbol{wp_2} - \boldsymbol{wp_1}||$
$l_{direction} = \frac{(\boldsymbol{wp_2} - \boldsymbol{wp_1})}{l_{length}}$

$add\ \boldsymbol{wp}_1\ to\ \boldsymbol{wp}$

$j = size\ of\ path\ \mathcal{P}$
while $\boldsymbol{wp}_1\ != \ end\ point\ of\ \mathcal{P}$ **do**
  if noCollisions$(\boldsymbol{wp}_1, l, \mathcal{O}, r_{\text{safe}})$ **then**
    $add\ \boldsymbol{wp}_2\ to\ \boldsymbol{wp}$
    $\boldsymbol{wp}_1 = \boldsymbol{wp}_2$
    $\boldsymbol{wp}_2 = end\ point\ of\ \mathcal{P}$
    $j = size\ of\ path\ \mathcal{P}$
  else
    $j = j - 1$
    $\boldsymbol{wp}_2 = point\ j\ on\ \mathcal{P}$
    $l_{length} = ||\boldsymbol{wp_2} - \boldsymbol{wp_1}||$
    $l_{direction} = \frac{(\boldsymbol{wp_2} - \boldsymbol{wp_1})}{l_{length}}$
  end if
end while
return $\boldsymbol{wp}$

Algorithm 4 noCollisions

Input: Waypoint $\boldsymbol{wp}_1$, line l between $\boldsymbol{wp}_1$ and $\boldsymbol{wp}_2$, obstacles $\mathcal{O}$ and safe radius r_{safe}
Output: Boolean value {TRUE, FALSE}, stating whether a collision occurs

$collision\_free = FALSE$
$valid\_obstacles = 0$
for all $obstacles\ in\ \mathcal{O}$ **do**
  if noCollision$(\boldsymbol{wp}_1, l, \mathcal{O}_i, r_{\text{safe}})$ **then**
    $valid\_obstacles = valid\_obstacles + 1$
  else
    $break\ out\ of\ for\ loop$
  end if
end for

if $valid\_obstacles == size\ of\ \mathcal{O}$ **then**
  $collision\_free = TRUE$
end if
return $collision\_free$

Algorithm 5 noCollision

Input: Waypoint $\boldsymbol{wp}_1$, line l between $\boldsymbol{wp}_1$ and $\boldsymbol{wp}_2$, obstacle $\mathcal{O}_i$ and safe radius r_{safe}
Output: Boolean value {TRUE, FALSE}, stating whether a collision occurs

$collision\_free = FALSE$
$length = ||obstacle\_position - \boldsymbol{wp_1}||$
$direction = \frac{obstacle\_position - \boldsymbol{wp_1}}{length}$
$scalar\_product = l^T_{direction} * direction$
$\theta = arccos(scalar\_product)$ ▷ $\in [0, \pi]$
$distance_{orth} = length * sin(\theta) - obstacle\_radius - r_{safe}$
$distance_{proj} = length * cos(\theta) - obstacle\_radius - r_{safe}$

if $\theta > \frac{\pi}{2}$ *or* $distance_{orth} > 0$ *or* $distance_{proj} > l_{length}$ **then**
  $collision\_free = TRUE$
end if
return $collision\_free$

A.4 RRT*

Algorithm 6 Body of RRT*

Input: Initial configuration $\boldsymbol{\xi}_{\text{init}}$, target configuration $\boldsymbol{\xi}_{\text{t}}$, obstacles $\mathcal{O}$, constraints ϵ, neigbourhood search radius r_{nh}, number of nodes N_{s} and safe radius r_{safe}
Output: A random tree of nodes and branches (V, E), avoiding obstacles and violation of constraints

$V = \boldsymbol{\xi}_{\text{init}}$
$E = empty\ edge\ list$
$i = 0$
while $i < N_{\text{s}}$ **do**
  $T = (V, E)$
  $\boldsymbol{\xi}_{\text{random}} = \text{randomSample}(i)$
  $(V, E) = \text{Extend}(T, \boldsymbol{\xi}_{\text{random}}, \mathcal{O}, \boldsymbol{\epsilon}, r_{\text{nh}}, r_{\text{safe}})$
end while
return (V, E)

Algorithm 7 Extend

Input: Random configuration $\boldsymbol{\xi}_{\text{random}}$, node and edge list (V, E), obstacles $\mathcal{O}$, constraints $\boldsymbol{\epsilon}$, neigbourhood search radius r_{nh}, and safe radius r_{safe}
Output: An extended tree of nodes and branches (V', E')

$V' = V$
$E' = E$
$\boldsymbol{\xi}_{\text{nearest}} = \text{nearest}((V', E'), \boldsymbol{\xi}_{\text{random}}, r_{\text{nh}})$
$\boldsymbol{\xi}_{\text{new}} = \text{steer}(\boldsymbol{\xi}_{\text{nearest}}, \boldsymbol{\xi}_{\text{random}}, \boldsymbol{\epsilon})$
$l = \text{line}(\boldsymbol{\xi}_{\text{nearest}}, \boldsymbol{\xi}_{\text{new}})$
if noCollisions$(\boldsymbol{\xi}_{\text{nearest}}, l, \mathcal{O}, r_{\text{safe}})$ **then**
  $V' = V' \cup \{\boldsymbol{\xi}_{\text{new}}\}$
  $\boldsymbol{\xi}_{\text{min}} = \boldsymbol{\xi}_{\text{nearest}}$
  $\Xi_{\text{near}} = \text{near}((V', E'), \boldsymbol{\xi}_{\text{new}}, |V|)$ ▷ Ξ_{near} is a set of near configurations
  for *all* $\boldsymbol{\xi}_{\text{near}} \in \Xi_{\text{near}}$ **do**
    $l = \text{line}(\boldsymbol{\xi}_{\text{near}}, \boldsymbol{\xi}_{\text{new}})$
    if noCollisions$(\boldsymbol{\xi}_{\text{near}}, l, \mathcal{O}, r_{\text{safe}})$ **then**
      $c' = \text{cost}(\boldsymbol{\xi}_{\text{near}}) + \text{cost}(l)$
      if $c' < \text{cost}(\boldsymbol{\xi}_{\text{new}})$ **then**
        $\boldsymbol{\xi}_{\text{min}} = \boldsymbol{\xi}_{\text{near}}$
      end if
    end if
  end for
  $E' = E' \cup \{(\boldsymbol{\xi}_{\text{min}}, \boldsymbol{\xi}_{\text{new}})\}$
  for *all* $\boldsymbol{\xi}_{\text{near}} \in \Xi_{\text{near}} \backslash \boldsymbol{\xi}_{\text{min}}$ **do**
    $l = \text{line}(\boldsymbol{\xi}_{\text{near}}, \boldsymbol{\xi}_{\text{new}})$
    if noCollisions$(\boldsymbol{\xi}_{\text{nearest}}, l, \mathcal{O}, r_{\text{safe}})$ *and*
    $\text{cost}(\boldsymbol{\xi}_{\text{near}}) > \text{cost}(\boldsymbol{\xi}_{\text{new}}) + \text{cost}(l)$ **then**
      $\boldsymbol{\xi}_{\text{parent}} = \text{parent}(\boldsymbol{\xi}_{\text{near}})$
      $E' = E' \backslash \{(\boldsymbol{\xi}_{\text{parent}}, \boldsymbol{\xi}_{\text{near}})\}$
      $E' = E' \cup \{(\boldsymbol{\xi}_{\text{new}}, \boldsymbol{\xi}_{\text{near}})\}$
    end if
  end for
end if
return (V', E')

References

[1] Antonelli, G. [2018]. *Underwater Robots*, Springer.

[2] Arvanitakis, I. and Tzes, A. [2012]. Trajectory optimization satisfying the robot's kinodynamic constraints for obstacle avoidance, *2012 20th Mediterranean Conference on Control Automation (MED)*, pp. 128–133.

[3] Breivik, M. and Fossen, T. I. [2005]. Guidance-based path following for autonomous underwater vehicles, *Proceedings of OCEANS 2005 MTS/IEEE*, pp. 2807–2814 Vol. 3.

[4] Breivik, M. and Fossen, T. I. [2009]. *Guidance Laws for Autonomous Underwater Vehicles*.

[5] Brown, T. [2019]. Ocean, NationalGeographic. last checked: 01.06.2022.
URL: *https://www.nationalgeographic.org/encyclopedia/ocean/*

[6] Choset, H. and Henning, W. [1999]. A follow-the-leader approach to serpentine robot motion planning, *Journal of Aerospace Engineering* **12**(2).

[7] Choset, H. et al. [2005]. *Principles of Robot Motion. Theory, Algorithms, and Implementations*, The MIT Press.

[8] Cormen, T. H. et al. [2009]. *Introduction to Algorithms*, The MIT Press.

[9] Egeland, O. and Gravdahl, J. T. [2002]. *Modeling and Simulation for Automatic Control*, Marine Cybernetics.

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

[11] Hoffmann, B. H. [2018]. *Path following and collision avoidance for an underwater swimming manipulator*, Master's thesis, Department of Cybernetics and Robotics, Norwegian University of Science and Technology.

[12] Hubbard, P. M. [1996]. Approximating polyhedra with spheres for time-critical collision detection, *ACM Transactions on Graphics* **15**: 179–210.

[13] Karaman, S. and Frazzoli, E. [2011]. Sampling-based algorithms for optimal motion planning, *The International Journal of Robotics Research* **30**: 846 – 894.

[14] Kelasidi, E., Liljebäck, P., Pettersen, K. Y. and Gravdahl, J. T. [2017]. Integral line-of-sight guidance for path following control of underwater snake robots: Theory and experiments, *IEEE Transactions on Robotics* **33**(3): 610–628.

[15] Kelasidi, E., Pettersen, K. Y. and Gravdahl, J. T. [2014]. A waypoint guidance strategy for underwater snake robots, *22nd Mediterranean Conference on Control and Automation*, pp. 1512–1519.

[16] Khatib, O. [1985]. Real-time obstacle avoidance for manipulators and mobile robots, *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, Vol. 2, pp. 500–505.

[17] Kohl, A. M., Moe, S., Kelasidi, E., Pettersen, K. Y. and Gravdahl, J. T. [2017]. Set-based path following and obstacle avoidance for underwater snake robots, *2017 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Vol. 1, pp. 1206 – 1213.

[18] Kuffner, J. J. and LaValle, S. M. [2000]. Rrt-connect: An efficient approach to single-query path planning, *Proceedings 2000 ICRA. Millennium Conference. IEEE*

International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065), Vol. 2, pp. 995–1001 vol.2.

[19] LaValle, S. M. [1998]. Rapidly-exploring random trees : a new tool for path planning, *The annual research report* .

[20] LaValle, S. M. [2006]. *Path Planning Algorithms*, Cambridge University Press.

[21] Lekkas, A. M. and Fossen, T. I. [2013]. *Line-of-Sight Guidance for Path Following of Marine Vehicles*.

[22] Liu, Y. and Zuo, G. [2020]. Improved rrt path planning algorithm for humanoid robotic arm, *2020 Chinese Control And Decision Conference (CCDC)*, pp. 397–402.

[23] Lynch, K. M. and Park, F. C. [2017]. *Modern Robotics: Mechanics, Planning, and Control*, Cambridge University Press.

[24] Mylvaganam, T. and Astolfi, A. [2016]. A nash game approach to mixed H_2/H_∞ control for input-affine nonlinear systems, *International Federation of Automatic Control (IFAC)* **49**(18): 1024 – 1029.

[25] Mylvaganam, T. and Sassano, M. [2018]. Autonomous collision avoidance for wheeled mobile robots using a differential game approach, *European Journal of Control (EJC)* **40**(4): 53 – 61.

[26] Nocedal, J. and Wright, S. J. [2006]. *Numerical Optimization*, Springer.

[27] Patle, B. et al. [2019]. A review: On path planning strategies for navigation of mobile robot, *Defence Technology* **15**(4): 582–606.

[28] Pidwirny, M. [2006]. Introduction to the oceans, Fundamentals of Physical Geography, 2nd Edition. last checked: 01.06.2022.
URL: *http://www.physicalgeography.net/fundamentals/8o.html*

[29] Schjølberg, I. and Fossen, T. I. [1994]. Modelling and control of underwater vehicle-manipulator systems, *in Poc. rd Conf. on Marine Craft maneuvering and control*, pp. 45–57.

[30] Schmidt-Didlaukies, H., Sørensen, A. J. and Pettersen, K. Y. [2018]. Modeling of articulated underwater robots for simulation and control, *2018 IEEE/OES Autonomous Underwater Vehicle Workshop (AUV)*, pp. 1–7.

[31] Spong, M. W., Hutchinson, S. and Vidyasagar, M. [2020]. *Robot Modeling and Control*, John Wiley & Sons, Ltd.

[32] Sverdrup-Thygeson, J., Kelasidi, E., Pettersen, K. Y. and Gravdahl, J. T. [2018]. The underwater swimming manipulator—a bioinspired solution for subsea operations, *IEEE Journal of Oceanic Engineering* **43**(2): 402–417.

[33] Urmson, C. and Simmons, R. [2003]. Approaches for heuristically biasing rrt growth, *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, Vol. 2, pp. 1178–1183 vol.2.

[34] Wei, H., Zheng, Y. and Gu, G. [2021]. Rrt-based path planning for follow-the-leader motion of hyper-redundant manipulators, *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3198–3204.

[35] Wrzos-Kaminska, M., Mylvaganam, T., Pettersen, K. Y. and Gravdahl, J. T. [2020]. Collision avoidance using mixed H_2/H_∞ control for an articulated intervention-auv, *European Control Conference (ECC)* **42**(4): 881 – 888.

[36] Åström, K. J. and Hägglund, T. [1995]. *PID Controllers: Theory, Design and Tuning*, International Society of Automation.

Peter Hatlebrekke Husebø

Whole-Body Motion Planning and Collision Avoidance for AIAUV

NTNU
Kunnskap for en bedre verden