Master's thesis

NTNU
Norwegian University of Science and Technology
Department of Engineering Cybernetics

Torben Falleth Olsen

Model Predictive Control-based Path-planning and Obstacle Avoidance for Real-Time Safe Underwater Operations

Master's thesis in Cybernetics and Robotics
June 2023

NTNU
Norwegian University of
Science and Technology

Torben Falleth Olsen

Model Predictive Control-based Path-planning and Obstacle Avoidance for Real-Time Safe Underwater Operations

Master's thesis in Cybernetics and Robotics
June 2023

Norwegian University of Science and Technology
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Problem Description

Aquaculture is an important global contributor to the production of seafood for human consumption, and in 2020, Norwegian aquaculture produced almost 1.5 million. tons of marketable fish meat. As fish farming sites are moved further offshore and to more exposed locations, working conditions get increasingly challenging due to the harsher environments at such sites, and the sheer remoteness to land. The automation of certain important fish farm operations is therefore an industrial aim to ensure safe and efficient operation.

Aquaculture also desires to shift production methods from manual operations and experience-based reasoning towards a more objective approach using intelligent sensors, mathematical models, decision support, and autonomous systems in different stages of production. However, using unsuitable technological tools and immature automation solutions can lead to unwanted events and accidents, that may in turn lead to economic losses, damages to structures and fish, and increased personnel risks. Avoiding this is the main objective of Precision Fish Farming, which provides approaches for adapting technological solutions to applications in aquaculture.

Using autonomous underwater vehicles (AUVs) is key in automating several aspects of aquaculture operations. However, since the situation in a fish farm is highly complex and dynamic due to the living fish, deformable flexible structures and at times demanding environmental conditions, it is difficult to automate operations using conventional methods and tools. While existing models and control strategies for AUVs allow navigation among rigid structures in static environments, they are not designed for operations in a dynamic fish farm environment where they need to react to the presence of animals and deformable structures such as net cages. SINTEF Ocean is targeting research to address the challenges (e.g., minimizing the impact on living fish during autonomous fish-farm operations) of using AUVs in dynamically changing environments such as fish farms.

An important area within this topic is to develop methods enabling the vehicles to move within the cage without colliding with the fish or the net structure. Previous research have explored different planning methods for avoiding both static and dynamic obstacles, covering methods such as the Elastic Band Method (EBM), RRT, RRT* and reinforcement learning based approaches. Model Predictive Control (MPC) approaches using optimisation to find an optimal obstacle free path, could pose an alternative solution to these methods. Due to their high computational demands, MPC methods have traditionally been considered unsuitable for real-time navigational

purposes. However, recent advances in computer technology, particularly in embedded systems, have made such approaches more relevant.

This master project will focus on the development of an MPC-based local path planner. This planner will be used between the waypoints defined by a global path planner (e.g., RRT* from the previous project [1]) to avoid dynamic obstacles in the path of an AUV.

The following items should be considered:

- Literature study covering the applications and previous work on using MPC for path planning.
  - Identify examples from applications on land, in the air, and on the ground.
  - Identify potential optimization criteria suitable for path planning composed of an objective function and constraints
- Implementation and testing
  - Implement the local path planner in C++
  - Explore the potential of using MPC for path optimization
- Simulation experiments
  - Simulations running case studies based on relevant examples from autonomous operations in fish farms

Abstract

Underwater path planning enables autonomous underwater vehicles (AUVs) to navigate complex environments safely and efficiently, which is essential when operating in aquaculture. This thesis aims to develop and assess MPC-based path-planning for AUVs operating in challenging underwater environments, especially in aquaculture settings. Furthermore, the thesis investigates integrating active perception together with the MPC-based path planner to improve overall navigation. Through testing in diverse underwater environments, this research demonstrates the promising potential of the path planner in enabling efficient and safe navigation for AUVs in challenging underwater scenarios while observing points of interest. The findings provided valuable insights, which would greatly benefit the advancement of underwater navigation in various environments, including aquaculture operations.

Acknowledgement

This work is performed in collaboration with SINTEF Ocean and supported by the Norwegian Research Council project CHANGE [grant no. 313737] and ResiFarm [grant no. 327292]. I would also like to express my gratitude towards my supervisor, Associate Professor Martin Føre, and co-supervisors, Dr. Eleni Kelasidi, Dr. Marios Xanthidis, and Mr. Herman Biørn Amundsen, for contributing with informative discussions and guidance during the project.

Contents

List of Figures

Abbreviations

MPC Model Predictive Control 3

NMPC Nonlinear Model Predictive Control 3

AUV Autonomous Underwater Vehicle 2

SQP *Sequential Quadratic Programming* 11

IPM *Interior Point Methods* 11

QP Quadratic Programming 11

NLP Nonlinear Programming 11

LQR Linear Quadratic Regulator 10

PFM Potential Field Method 7

RRT Rapidly exploring Random Trees 2

DOF Degrees Of Freedom 6

EBM Elastic Band Method 2

PRM Probabilistic roadmaps 8

PRM* PRM-based near-optimal planner 8

RRT* RRT-based near-optimal planner 2

ICRA IEEE International Conference on Robotics and Automation . 4

Part I
Introduction

Aquaculture is an important global contributor to seafood production for human consumption. The practice of aquaculture involves the farming of aquatic organisms in diverse environments with varying physical characteristics. Automating fish farming operations becomes increasingly critical for safety and efficiency as the practice moves to more challenging and remote locations [2, 3]. To ensure the success of aquaculture operations, it is essential to have reliable autonomy for automated underwater vehicles [4]. To achieve autonomy and operate efficiently within an aquatic environment, the presence of a reliable path-planning system is of utmost importance. This system generates paths for vehicles to follow, ensuring efficient and safe navigation. With the help of path planning algorithms, underwater vehicles can navigate around obstacles and find the most efficient routes to their destination, resulting in reduced energy usage and lower operational costs. Additionally, precise and controlled movement within the environment is essential for the health and safety of aquatic organisms, as it minimizes disturbances and potential harm to marine life [2].

1 Aquaculture production practices

Aquaculture production practices aim to provide optimal conditions for aquatic organisms' growth, health, and productivity [2, 3]. This involves careful management of water quality, temperature, salinity, and nutrition. Aquaculture systems vary greatly, from extensive systems in natural water bodies to intensive systems in land-based tanks or offshore structures. Efficient and sustainable production is a significant challenge in aquaculture, which requires advanced and reliable technologies. Robotics and automation are crucial in improving aquaculture operations, allowing for automated tasks such as feeding, monitoring water quality, disease detection, and harvesting. This helps to reduce labor costs and enhance overall production efficiency [2, 3].

2 Autonomous navigation and path planning in aquaculture

Effective path planning is crucial in aquaculture for promoting sustainable environmental management through robotic navigation. An Autonomous Underwater Vehicle (AUV) must determine the most efficient path to navigate the underwater environment while avoiding obstacles and complex terrain to reach their destinations successfully.

In aquaculture, underwater robots require a seamless integration of perception and path planning for reliable navigation. By combining these components, trajectories can be created to track and monitor important features like fish, corals, and structures in the aquatic environment. This integration is beneficial when navigating through fish cages, as it allows for inspecting the cage's condition and gathering vital data on the fish population's well-being and behavior. With this capability, underwater robots can perform targeted and informed actions, contributing to the overall efficiency and effectiveness of aquaculture operations [5, 6, 4].

As part of the previous project [1], the aim was to evaluate the effectiveness of different global path planners for use in an aquaculture setting within an underwater environment. Global path planning involves analyzing the global map and obstacles to generate a path from point A to point B. The planners tested included Rapidly exploring Random Trees (RRT), RRT-based near-optimal planner (RRT*), and Elastic Band Method (EBM), which demonstrated varying levels of success. However, relying solely on a global path planner in a complex underwater environment is not advisable. While it can generate an optimal path globally, it will not adjust to environmental changes, resulting in collisions and an inefficient strategy. Therefore, it is necessary to pair a global path planner with a local path planner that continuously updates the path based on the surroundings, ensuring the robot's safety.

Local path planning is crucial for navigating complex underwater environments for AUVs. Local path planning involves identifying a safe route for a robot or vehicle in its immediate vicinity, considering the obstacles and surroundings. Various methods have been utilized in this field, with conventional approaches such as potential field method [7] or D* [8] being widely used to generate collision-free paths in cluttered underwater environments. However, these methods may encounter local minima, resulting in sub-optimal path quality.

More advanced local path planning methods, such as Model Predictive Control (MPC), have recently been introduced for underwater robotic path planning. MPC-based planners are a relatively new type of planner that uses a dynamic system model to optimize a future trajectory subject to constraints. Previous studies have tested and implemented MPC as a path planner for ground and aerial vehicles. These studies have shown that MPC is a reliable path planner that can create safe and comfortable paths for autonomous vehicles [9, 10]. However, the limitations of these studies include the absence of extra functions in the planner, the implementation of active perception, and the lack of testing in complex underwater environments.

The limitation of traditional MPC is that it relies on a model of the environment, which may not always be accurate. The system model is assumed to be linear in a standard linear MPC. However, this assumption may not hold in underwater environments where the rigid-body kinematics can be nonlinear. Employing a linear MPC in such scenarios can lead to complications since it requires linearizing the nonlinear system, which is an impractical simplification. To address this limitation, Nonlinear Model Predictive Control (NMPC) has been proposed. The NMPC's capability to manage complex dynamics, constraints, and changes in real-time makes it ideal for underwater environments. NMPC uses a nonlinear model of the system to optimize the trajectory and is more robust to model uncertainties than traditional MPC.

Considering the success of MPC-based path planning in ground and aerial vehicles and the lack of testing on underwater vehicles, it would be interesting to explore its potential as a path planner in dynamically changing underwater environments. This thesis aims to develop, implement, and test a path planner for underwater robots, mainly focusing on MPC-based local path planning. The aim is to thoroughly examine the challenges and opportunities associated with this approach, emphasizing the real-world application of NMPC in underwater environments. By investigating the potential of NMPC to enhance the navigation capabilities of autonomous underwater vehicles, this project aims to advance underwater robotics and facilitate more efficient and successful underwater exploration and operations.

3 Contributions

The following contribution has been made to this thesis:

- The results obtained in this thesis are aimed for conference publications in IEEE International Conference on Robotics and Automation (ICRA).
- Designed the optimization problem for the NMPC-based path-planner. This is described in Section 8.
- Creation of additional functions used in the path planning. This is described in Section 9.
- Implementation of active perception together with the NMPC-based path planner. This is described in Section 9.5.
- Development of the total path planning algorithm. This is described in Section 9.6
- Comparison of the performance of the planning time with different numbers of obstacles. This is presented in Section 11.4
- Case studies evaluating the performance of the path planner. This is presented in Section 11.
- Case studies evaluation the performance of the path planner with active perception. This is presented in Section 12.

4 Outline

- Chapter I introduces and explains the problem this project aims to solve. It gives a detailed explanation of the problem statement and its relevance.
- Chapter II offers a detailed summary of the key theoretical background that serves as the basis for the following work. Its goal is to provide readers with the essential knowledge and comprehension required to understand and value the ensuing discussions and analyses presented throughout the project.
- Chapter III focuses on the practical implementation aspects of the software, calculations, and functional concepts. It provides a detailed account of the steps to develop and deploy the software system, including the various calculations and algorithms employed.
- Chapter IV validates the path planner in different situations. It explains the evaluation and presents the path planner's performance results.
- Chapter V analyzes the research results by examining the data collected during experimentation and evaluation to better understand the outcomes' significance and implications.
- Chapter VI summarizes the essential findings and insights from the research. It provides a thorough overview of the results, emphasizing the primary outcomes and their implications for the field.

Part II
Theory

This chapter introduces the theory of path planning, which builds upon some of the pre-project work in [1]. The goal is to establish a strong theoretical foundation and provide a comprehensive understanding of the path-planning principles used.

5 Path Planning

Path planning [11] is a computational problem that involves finding a valid sequence of configurations to determine an object's path from its starting pose to its goal pose within the free space, C_{free}, which is crucial for safe navigation in autonomous systems. Path planning is usually presented as a search problem in a state space [12], where the state refers to the relevant properties of the robot, and the state space is made up of all possible states. The state space is created by including the Degrees Of Freedom (DOF), each adding a dimension to the issue, leading to a space with multiple dimensions. A specific robot configuration or state corresponds to a point in the state space. The configuration space [12] is the collection of all feasible robot configurations represented by $C \in \mathbb{R}^n$ and is defined as $C = [c^i_{min}, c^i_{max}] \times \cdots \times [c^n_{min}, c^n_{max}]$ where n is the number of degrees of freedom and $[c^n_{min}, c^n_{max}]$ defines the minimum and maximum boundaries of the space [13]. In path planning problems with obstacles, C can be divided into two complementary subsets, namely free space C_{free} and obstacle space C_{obs} [12]. Free space refers to all the safe states where the robot can move without colliding with obstacles or being in any invalid states. The obstacle space refers to any states considered invalid within the system, which collisions or other limitations can cause.

Regarding path planning, optimality refers to how effectively the planner can find the best solution for a given problem [12]. There are different types of optimality, including near-optimal, global optimality, and local optimality. A near-optimal solution is a solution that is close to the optimal within a specified tolerance. Regardless of location, a globally optimal solution is the best solution for the problem. A locally optimal solution is the best solution within a limited range of options.

There are two types of path planning to consider when planning a safe and efficient route: global and local [14]. Global path planning creates a path from the start to the final destination, taking into account the entire environment shown on a map. On the other hand, local path planning focuses

on generating a path in real time that avoids obstacles based on the current surroundings and the vehicle's status. Although global path planning is crucial for determining the overall route, it is not enough for real-time control of a AUV, particularly when faced with time-sensitive tasks or unexpected obstacles. Therefore, local path-planning techniques are necessary to ensure safe and efficient navigation in dynamic environments.

5.1 Analysis of Common Path Planners

Various path-planning algorithms exist in robotics, each with unique strengths and limitations. Understanding the differences between these algorithms is crucial for selecting the most suitable approach for a given robotic application. This section presents some of the popular path planners in robotics, aiming to explain their differences.

One particularly influential path planner is the Potential Field Method (PFM) [15]. The PFM algorithm utilized attractive and repulsive forces to guide a robot towards a goal while avoiding obstacles [16]. The combined force of the two determines the direction of where the vehicle moves. Its simplicity and real-time computation made it a popular choice [7]. However, one of the drawbacks of PFM is its tendency to get stuck in local minimums, oscillations in narrow passages, and close to obstacles, leading to sub-optimal path quality [17]. The EBM[18] is an enhanced version of the PFM. It incorporates an elastic band model to refine the robot's path and enhance the overall path quality. The method combines attractive and repulsive forces of PFM with the band's flexibility to generate smoother and more efficient paths. However, it will face limitations as the PFM when it is close to the goal.

Another commonly used path planner is A* [19]. This algorithm uses heuristic functions to guide the optimal path generation by minimizing the path cost [16]. The strength of this method is its efficiency, simplicity, and modularity [20]. However, its disadvantage is that it can be computationally expensive when dealing with complex problems [20]. In dynamic environments where the map must be updated continuously based on new observations, specialized variants of A*, such as D* [8], are commonly used. This variant update previously explored states, resulting in significant computational efficiency improvements without sacrificing the optimality guarantees of the path planning algorithm. Although it is effective, the computational cost of this method is high, and its action space is discretized, limiting the number of available actions. Furthermore, it is not suitable for use in high-dimensional spaces.

Probabilistic methods, such as RRT [21] or Probabilistic roadmaps (PRM) [22], is also commonly used path planning method, which generates paths in a probabilistic manner. The RRT is a sampling-based path-planning method that constructs a tree by randomly sampling configurations, and it's great for handling complex constraints and high-dimensional spaces [16]. However, its paths may not always be optimal as it can settle for a local minimum [16]. PRM also uses a sampling-based approach and constructs a roadmap from randomly sampled configurations. This method is effective in high-dimensional spaces and generates non-intuitive paths [16]. However, constructing the roadmap can be computationally expensive [16]. The quality of PRM's paths depends heavily on the connectivity of the roadmap, and sparse sampling or weak local connections can result in inefficient or suboptimal paths. Improving these methods into RRT* [23] and PRM-based near-optimal planner (PRM*) [23] have been developed to almost guarantee a solution. These can be used anytime, with extra iterations improving the path. However, updating and rewiring can be computationally expensive compared to the original planners' performance.

Sampling-based techniques can solve problems, but their solutions are often sub-optimal, leading to rough and aggressive movements. Optimization-based planners such as Trajopt [24, 25] and CHOMP [26] take constraints and cost functions as inputs and generate an optimized path. CHOMP is a reliable algorithm that can solve various problems starting with a linear interpolation between the initial and goal configurations. However, it may take longer and be less accurate when dealing with complex problems. On the other hand, Trajopt is a popular path optimization framework that can quickly produce paths that meet constraints and optimize desired cost functions both in 2D and 3D space [27].

Recently path planning based on MPC has been adopted [28, 10, 29], which has shown great promise. The MPC uses a dynamic system model to optimize the future trajectory subjected to constraints. The method has shown the ability to produce safe and comfortable paths [9]. The strength of this method lies in its ability to handle constraints and generate dynamically feasible paths. However, it requires a model of the system and can be computationally expensive.

While some methods, like PFM and RRT, provide solutions, they may lead to sub-optimal paths or encounter challenges in certain environments. Algorithms like A* can face challenges when dealing with complex problems. Therefore, it is crucial to choose path-planning techniques that are efficient, optimal, and robust for the specific context or problem domain. Careful evaluation is necessary to make the right choice. Recent technological advancements have led to adopting MPC for path planning. MPC has shown

great promise in producing safe and comfortable paths for autonomous vehicles. Therefore, it would be valuable to test the performance of MPC-based path planning in real-world underwater scenarios to evaluate its effectiveness for underwater navigation further.

6 Model Predictive Control for Path Planning

To explore how MPC can be used for navigation, it is essential to understand its basic principles and concepts. This section provides a comprehensive overview of MPC, laying the groundwork for a deeper exploration of how it can be applied to navigation and path planning. By understanding the basics of MPC, one can better understand how it can successfully face the obstacles of autonomous navigation in various environments.

6.1 Model Predictive Control

MPC [30] is an advanced process control technique that calculates the current control action by solving a finite horizon open-loop optimal control problem at each sampling instance, also meeting a series of constraints. The optimization process begins with the system's present state as the starting point and produces an optimal control sequence. The initial control in the sequence is then subsequently applied to the system. The equation below shows how the optimization problem is usually structured in a MPC [31],

$$
\begin{aligned}
\min_{z \in \mathbb{R}^n} \quad & f(z) && (1.) \\
\text{s.t.} \quad & && \\
& c_1(z) = g_1(z) && (2.) \\
& c_2(z) \geq g_2(z) && (3.) \\
& c_3(z) \leq g_3(z) && (4.)
\end{aligned}
\tag{1}
$$

where $f(z)$ is the objective function, $c_i(z)$ are the constraints and $g_i(z)$ are the contraint functions. The objective function (1.) measures how good a solution is to a problem's goals. It takes input variables and produces a scalar value to be maximized or minimized. The constraints in (2.)-(4.) limit possible problem solutions by defining requirements and restrictions and optimizing the objective function. There are many ways to formulate the objective function. One typical formulation is [31]:

$$
\begin{aligned}
f(z) \;=\; & \sum_{i=0}^{N-1} x_{i+1}^{\top}\, Q_{i+1}\, x_{i+1} + d_{xi+1}\, x_{i+1} \\
& + \frac{1}{2}\, u_i^{\top} R_i u_i + d_{ui} u_i + \frac{1}{2} \Delta u_i^{\top} R_{\Delta i} u_i
\end{aligned}
$$

The variables in the equation include the state of the plant represented by x_{i+1}, the input by u_i, input change by Δu_i, prediction horizon N and tunable variables such as Q_{i+1}, d_{xi+1}, R_i, d_{ui}, and $R_{\Delta i}$. The Q and R matrices assign weights to variables in the cost function, tuning performance, effort, and stability for controller modification. However, the form previously shown is just a basic example of a quadratic cost function, and there are numerous variations that the objective function can take.

6.2 MPC vs LQR

Linear Quadratic Regulator (LQR) and MPC are two different methods for designing control strategies for dynamic systems, with significant differences despite some similarities.

LQR is a feedback control technique that designs a linear controller to minimize a quadratic cost function [31]. It is a one-step control strategy that computes the entire control sequence based on the system's current state. LQR is best suited for linear systems with no constraints.

MPC, on the other hand, is a receding horizon control technique that predicts the system's future behavior using a dynamic model and optimizes a control sequence over a finite time horizon [31]. Constraints on the system's state and input are considered during optimization. Only the first optimized control action is applied at each time step, and the optimization process repeats with a new initial state at the next step. MPC is an effective strategy for systems with dynamics and constraints.

It's worth noting that the MPC approach is more relevant than the simpler LQR approach. While LQR computes the entire control sequence in a single step based on the current system state, MPC considers future behavior by predicting it and optimizing a control sequence over a finite time horizon using a dynamic system model, all while considering constraints. Therefore, the MPC approach provides more flexibility and adaptability in controlling the system than LQR.

6.3 Nonlinear Model Predictive Control

NMPC [31] is a form of MPC that uses nonlinear system models in its calculations. Similarly to MPC, NMPC solves the optimal control problems iteratively over a finite prediction horizon. However, unlike linear MPC, these problems may not be convex in NMPC, presenting complications for both the stability and the numerical solutions. Non-convex problems are optimization problems that do not follow the rules of convexity. Convexity means that a curve in a function or set is either flat or curves upward, and any line connecting two points within the curve or set is above or on the curve. To effectively use NMPC, it is crucial to have a solver that solves the Nonlinear Programming (NLP) problem. Therefore, a nonlinear solver is required. The two primary classes of solvers available are *Sequential Quadratic Programming* (SQP) and *Interior Point Methods* (IPM) [31].

Sequential Quadratic Programming (SQP)

SQP is a numerical optimization method that utilizes an iterative approach to find the optimal solution. It achieves this by solving Quadratic Programming (QP) sub-problems sequentially [32]. There are traditionally two types of classes that are used in the SQP method, and these are *line search* and *trust-region* methods [32]. These classes ensure global convergence of locally convergent minimization in the SQP method. The **Line search** method controls the step length taken along the computed SQP direction [32]. The **Trust-region** method, on the other hand, aims to control the step length at the same time as computing the search direction [32].

Interior Point Methods (IPM)

IPM are numerical optimization techniques that transform constrained optimization problems into unconstrained ones by introducing a barrier function that penalizes constraint violations [32]. In practice, there are two effective classes of IPMs. The first class is an extension of IPMs for linear and quadratic programming, which enforces convergence through line searches and computes steps using direct linear algebra. The second class employs a quadratic model to define the step and includes a trust-region constraint to ensure stability. These two approaches share similarities with line search and trust-region SQP methods [32].

Part III
Method Description

An NMPC-based local path planner was implemented in Visual Studio using C++. The planner used **CasADi** [33] library with the *Opti stack* class to solve the optimization problem formulated in Section 8. To ultimately solve the path planning problem presented in Section 7.

7 Problem Statement

This project aims to produce safe paths for underwater vehicles operating in a constrained underwater aquaculture environment with unknown underwater conditions, such as aquatic organisms or cage instruments. To simplify the problem, the AUV is assumed to be holonomic, meaning that the number of DOF matches the number of controllable DOF. A typical 3D configuration space C will be utilized in the environment, meaning $C = X$ x Y x Z. Let $\mathbf{x}_{init} = [x_{init}\ y_{init}\ z_{init}]$ indicate the initial state of the vehicle, and the goal defined as $\mathbf{x}_{goal} = [x_{goal}\ y_{goal}\ z_{goal}]$, where $\mathbf{x}$ is the location of a state within the configuration space. Additionally, let $\mathbf{P}$ define a path from the initial state $\mathbf{x}_0$ to a goal $\mathbf{x}_{goal}$. Lastly, let $\mathbf{O} = [\mathbf{o}_0\ \mathbf{o}_1 \dots \mathbf{o}_n]$ be a set of known convex spherical obstacles within the environment, where $\mathbf{o}_j = [\mathbf{x}_j\ r_j^{obs}]^\top \in \mathbf{O}$ is the state of obstacle j. The objective is to produce locally optimal paths with minimal length, avoiding collision between the vehicle and any obstacle within the environment.

8 Optimization Problem Formulation for NMPC-Based Path Planning

To address the challenges inherent in underwater navigation and path planning, the initial phase of development involved creating a design based on the NMPC principles presented in 6. This design optimizes the path-planning process, allowing the planner to generate safe and efficient trajectories while considering surrounding obstacles. This section presents the key components implemented in the NMPC-based path planner, highlighting how these enhancements improve the navigation capabilities of the vehicle. The design of the NMPC is presented by the optimization problem shown in (2).

$$
\begin{aligned}
&\min_{\mathbf{x}\in\mathbb{R}^3} \quad \sum_{i=0}^{N-1} ||\mathbf{x}_{i+1} - \mathbf{x}_i||^2 && (1.)\\
&\text{s.t.}\\
&\qquad \mathbf{x}_0 = \mathbf{x}_{init} && (2.)\\
&\qquad \mathbf{x}_N = \mathbf{x}_N && (3.)\\
&\qquad ||\mathbf{x}_i - \mathbf{x}_j|| > r_{robot} + r_j^{obs} && (4.)\\
&\qquad \left|\left|\frac{\mathbf{x}_{i+1} + \mathbf{x}_i}{2} - \mathbf{x}_j\right|\right| > r_{robot} + r_j^{obs} && (5.)\\
&\qquad r_{robot} < \min_{k\in P}\{||\mathbf{x}_i - \mathbf{x}_k||\} \leq r_{robot} + d_{obsrv} && (6.)
\end{aligned}
\tag{2}
$$

The NMPC-based path planner's objective function aims to minimize the path length within the prediction horizon (N). To achieve this, the sum of distances between consecutive states represented by $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ is calculated in (1.), as shown in Equation (3). By minimizing the length, the path planner can find the most efficient and compact path from the initial state to the destination, leading to optimal navigation.

$$
\min_{\mathbf{x}\in\mathbb{R}^3} \quad \sum_{i=0}^{N-1} ||\mathbf{x}_{i+1} - \mathbf{x}_i||^2 \tag{3}
$$

where $\mathbf{x}_i$ refers to the state of path node i, represented as a three-dimensional vector $\mathbf{x}_i = [x_i\ y_i\ z_i]^\top \in \mathbf{P},\ i \in [0\,,\ N-1]$. The equality constraints outlined in Equation (4) fix the initial (2.) and final (3.) points of the prediction horizon, thereby preventing the path from collapsing to a zero length. This ensures that the path maintains a non-zero length throughout the prediction horizon.

$$
\begin{aligned}
\mathbf{x}_0 &= \mathbf{x}_{init}\\
\mathbf{x}_N &= \mathbf{x}_N
\end{aligned}
\tag{4}
$$

Where $\mathbf{x}_{init}$ represents the current vehicle state, while $\mathbf{x}_N$ indicates the state located at the end of the prediction horizon. Equation (5) enforces an inequality constraint in line (4.), preventing any states from intersecting with a set of obstacles $\mathbf{O}$ that may cause a collision.

$$||\mathbf{x}_i - \mathbf{x}_j|| > r_{robot} + r_j^{obs} \tag{5}$$

where $\mathbf{x}_j$ corresponds to the position of an obstacle, $r_j^{obs} > 0$ denotes the obstacle radius, and $r_{robot} > 0$ represents the radius of the vehicle. The following equation from line (5.) illustrates an additional inequality constraint, which prohibits the middle point between $\mathbf{x}_{i+1}$ and $\mathbf{x}_i$ from colliding with any obstacles.

$$\left|\left|\frac{\mathbf{x}_{i+1} + \mathbf{x}_i}{2} - \mathbf{x}_j\right|\right| > r_{robot} + r_j^{obs} \tag{6}$$

8.1 Active Perception

An additional constraint was added to the optimization problem to enhance the planner for aquaculture use. Apart from the constraints that help navigate from the starting point to the endpoint, this new constraint enables observing points of interest close to the path. Equation (7) shows the constraint used, which applies an attractive force on the path towards a specific observation point (6.).

$$r_{robot} < \min_{k \in \mathbf{K}} \{||\mathbf{x}_i - \mathbf{x}_k||\} \leq r_{robot} + d_{obsrv} \tag{7}$$

The variable $\mathbf{x}_k \in \mathbf{K}$ denotes a point of interest, specifically, a point the path is attracted to but does not collide into so that the vehicle can safely observe the point. The scalar d_{obsrv} represents the maximum distance over which observations can be made.

9 Planner Formulation

This section will discuss the functions added to the NMPC-based local path planner. The purpose of these functions is to enhance navigation in underwater environments and create safer paths. The goal is to improve the performance and reliability of the path planner, allowing it to handle the unique challenges posed by underwater conditions. These functions were added to optimize the path planner's ability to navigate complex underwater terrains, avoid obstacles, and ultimately increase the safety and efficiency of underwater robotic operations. The functions are:

- **Initial Path Update** updates the initial path used in the optimization based on the vehicle's state.
- **Path Resolution** regulates the number of states in the path based on a predefined step length.
- **Collision Prediction** predicts collisions with moving obstacles, creating static obstacles at collision points.
- **Prediction Horizon Extension** moves the last state of the prediction horizon to a safe state if the current is not feasible.
- **Active Perseption** does not assist with collision avoidance but is designed to enhance navigation and efficiency during underwater missions. This function pushes the path toward a specific point of interest, allowing the vehicle to observe the point safely.
- **Collision Avoidance** creates an optimized path that avoids obstacles, combining the NMPC framework and the previous functions.

9.1 Initial Path Update

The **Initial Path Update** function recalculates and generates the initial path from the vehicle's current state to the goal. This function is used as a starting point for the path-planning process. To better understand this process, refer to figure 1 for a visual representation.

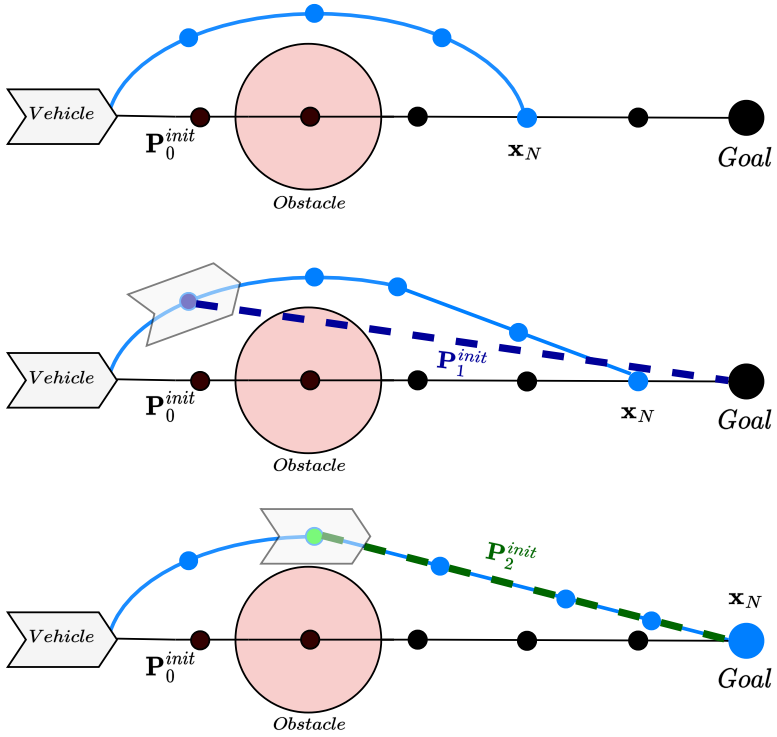


Figure 1: The figure shows the process of updating the initial path and computing the locally optimal path based on the vehicle's movement along the path.

Figure 1 depicts a vehicle in a gray color that progressively computes the initial path ($\mathbf{P}_i^{init}$) and the locally optimal path (blue) using the sliding window goal $\mathbf{x}_N$ and the global goal (Black). The figure shows transparent versions of the vehicle at different times. These vehicles have recalculated the initial path based on their current state, depicting a dark blue line indicated as the first updated path and a green line indicated as the second.

The function begins by calculating the distance between the goal $\mathbf{x}_{goal}$ and the current state of the vehicle $\mathbf{x}_0$ in x, y, and z directions as shown below.

$$\mathbf{x}_{goal} - \mathbf{x}_0 \tag{8}$$

After computing the distance, the function determines the number of states between the vehicle's current state and the goal. This is calculated using the following equation:

$$n = \frac{||\mathbf{x}_{goal} - \mathbf{x}_0||}{|\mathbf{k}|} + 1 \tag{9}$$

Where $|\mathbf{k}| > 0$ is the Euclidean step length between the path states, and n is the number of states between the current state and the goal, the next step involves calculating the updated initial path by determining the new step length $\mathbf{k}$ in x, y, and z. The new step length can then be expressed as:

$$\mathbf{k} = \frac{\mathbf{x}_{goal} - \mathbf{x}_0}{n-1}, \quad n > 1 \tag{10}$$

A new initial path between the current state and the goal is then generated using $\mathbf{k}$ as the length between each state.

9.1.1 Algorithm

Algorithm 1, which utilizes the equations previously presented, is used to create the function that updates the initial path. The inputs required for this algorithm are specified in Line 1. The output coming from this algorithm is specified in Line 2. Line 3 computes the number of states necessary to reach the goal, given a desired step length. The new step length for the initial path is calculated in Line 4, and in Lines 6 and 7, the new initial path is generated. Finally, the algorithm returns the path in Line 8.

Algorithm 1 InitialPathUpdate()

1: **Input**: $\mathbf{x}_i \rightarrow$ State, $\mathbf{x}_{goal} \rightarrow$ Goal, $|\mathbf{k}| \rightarrow$ Desired step length.
2: **Output**: $\mathbf{P} \rightarrow$ Path.
3: $n = \frac{||\,\mathbf{x}_{goal} - \mathbf{x}_0\,||}{|\mathbf{k}|}$
4: $\mathbf{k} = \frac{\mathbf{x}_{goal} - \mathbf{x}_0}{n-1}$
5: $i = 0$
6: **for** $i < n$ **do**
7: $\quad \mathbf{x}_i \leftarrow [x_0 + ik_x,\ y_0 + ik_y,\ z_0 + ik_z] \in \mathbf{P}$
8: **return P**

9.2 Path Resolution

The **Path Resolution** function adjusts the path resolution by adding or removing states to ensure a smoother trajectory, which will help to navigate more safely. Figure 2 visually represents this process.

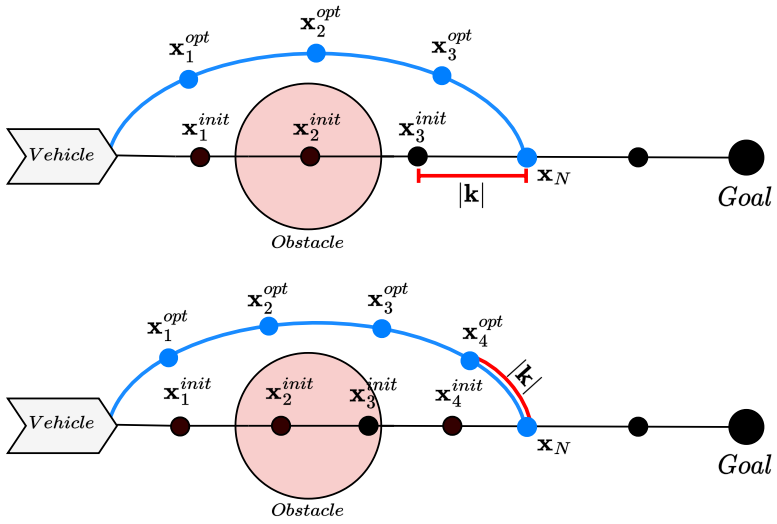


Figure 2: The figure shows the first optimal path calculated from the initial path with step length $|\mathbf{k}|$, where it has three nodes in the horizon. The function then uses the previous solution to calculate the new number of states for the path to maintain step length $|\mathbf{k}|$.

In Figure 2, the gray vehicle has created a path from its current state to its destination, like in previous functions. The optimized path (blue) goes around the obstacle (pink), where it obtains a different step length ($|\mathbf{k}|$) than the initial path (black). On the next iteration, the initial path is adjusted to maintain consistency in step length, resulting in the optimized path's step length obtaining the same step length as the initial path. In this example, this is done by inserting a new state in the local horizon.

The function uses the optimized path's Euclidean distance, represented as l_f, to determine how many states a path needs to contain. This distance is calculated using Equation (11), which considers the distance between adjacent states in the horizon.

$$l_f = \sum_{i=0}^{N-1} || \mathbf{x}_{i+1}^{opt} - \mathbf{x}_i^{opt} || \tag{11}$$

where $\mathbf{x}_i^{opt}$ is an optimal state, $\mathbf{x}_{i+1}^{opt}$ is the adjacent optimal state and N is the number of states in the prediction horizon. Next is calculating the new

number of states n needed for the optimal path to match the desired step length $|\mathbf{k}|$. The following equation calculates this.

$$n = \frac{l_f}{|\mathbf{k}|} \tag{12}$$

When the state index n is calculated, the new initial path step length $\mathbf{k}$ must be calculated within the prediction horizon using the following equation.

$$\mathbf{k} = \frac{\mathbf{x}_N - \mathbf{x}_0}{n} \tag{13}$$

where $\mathbf{x}_0$ is the current state and $\mathbf{x}_N$ is the last state in the prediction horizon. Then, the new number of states can be placed using the new step length to create step length $|\mathbf{k}|$ on the optimal path.

9.2.1 Algorithm

The equations previously mentioned are implemented in Algorithm 2 to update the path resolution. The required inputs for the algorithm are listed in Line 1. The algorithm outputs are listed in Line 2. Lines 4 and 5 calculate the Euclidean distance of the optimal path, while Line 6 calculates the number of states needed to match the desired step size in the optimal path. Line 7 computes the new step length for the initial path within the horizon. The updated number of states within the prediction horizon is added to the initial path in Lines 9 to 10. Finally, Line 11 returns the updated initial path.

Algorithm 2 PathResolution()

1: **Input**: $\mathbf{x}_i \rightarrow$ States, $\mathbf{x}_i^{Opt} \rightarrow$ Locally Optimal States, $|\mathbf{k}| \rightarrow$ Desired step length, $N \rightarrow$ The number of states in the prediction horizon.

2: **Output**: $\mathbf{P} \rightarrow$ Path.

3: $i = 0$

4: **for** $i < N$ **do**

5: $\quad l_f = \sum_{i=0}^{N-1} || \mathbf{x}_{i+1}^{opt} - \mathbf{x}_i^{opt} ||$

6: $n = \frac{l_f}{|\mathbf{k}|}$

7: $\mathbf{k} = \frac{\mathbf{x}_N - \mathbf{x}_0}{n}$

8: $i = 0$

9: **for** $i < n$ **do**

10: $\quad \mathbf{x}_i \leftarrow [\ x_0 + (ik_x,\ y_0 + iky,\ z_0 + ik_z\ ] \in \mathbf{P}$

11: **return P**

9.3 Collision Prediction

The **Collision Prediction** function uses a similar idea to predict the obstacle trajectory presented in [34], where it predicts future collision between the vehicle and a dynamic obstacle within the prediction horizon. This function enables the local path optimization to predict and avoid these collisions ensuring the vehicle's safety. Figure 3 offers an illustration to understand the action better.

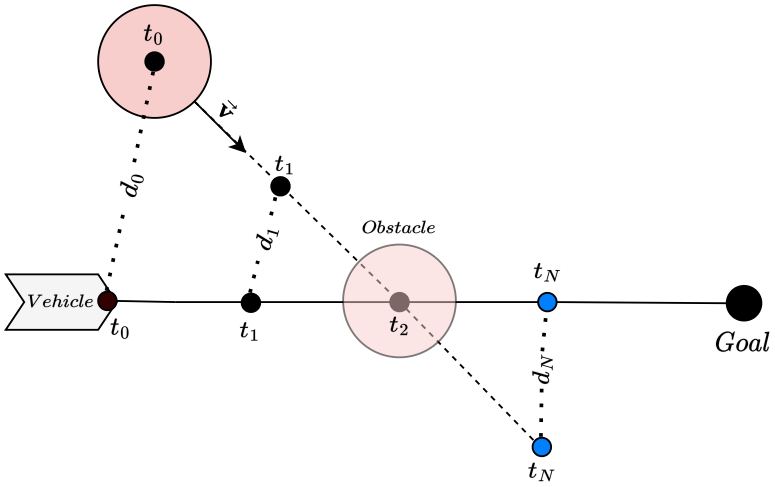


Figure 3: NMPC - The figure shows the trajectory of a dynamic obstacle (pink) and the path of the vehicle, where prediction is made as to where they will collide (see-through pink)

Figure 3 shows an initial path (black) between the vehicle (gray) and the goal. On the left corner of the figure, there is an obstacle (pink) moving with a velocity $\vec{\mathbf{v}}$. The optimization uses a time-based state trajectory to predict where the obstacle and vehicle will collide along the path. This means that the optimization uses time-based states instead of position-based states. This is done by calculating the time it takes for the vehicle to go from state $\mathbf{x}_i$ to $\mathbf{x}_{i+1}$. The function then calculates the distance (d_i) between the vehicle states corresponding to the obstacle states, predicting the collision and creating a static obstacle at the collision point (see-through pink).

The vehicle's state time $t_i \in \mathbf{T}$ is calculated to obtain the time it takes for the vehicle to reach state $\mathbf{x}_i$ from its current state. Equation (14) is the equation used for this.

$$t_{i+1} = t_i + \frac{||\mathbf{x}_{i+1} - \mathbf{x}_i||}{||\dot{\mathbf{x}}_0||}, \quad ||\dot{\mathbf{x}}_0|| > 0 \tag{14}$$

Here, a state of the path is represented by $\mathbf{x}_i$, while the velocity of the vehicle is indicated by $\dot{\mathbf{x}}_0$. Once the time-based states are found, the obstacle trajectory $\mathbf{x}_j^{t_i}$ is calculated using equation (15).

$$\mathbf{x}_j^{t_i} = \mathbf{x}_j + (\dot{\mathbf{x}}_j t_i) \tag{15}$$

where the current position of an obstacle is represented by $\mathbf{x}_j$, while its current speed is denoted by $\dot{\mathbf{x}}_j$. To determine whether the vehicle collides with an obstacle, it is necessary to evaluate the associated obstacle trajectory, denoted by $\mathbf{x}_j^{t_i}$, against the following condition. This is to check if a state in the local path collides with the corresponding obstacle trajectory state.

$$||\mathbf{x}_i - \mathbf{x}_j^{t_i}|| \leq r_{robot} + r_j^{obs}$$

If the condition is met, the obstacle will collide with the vehicle. The position of the obstacle $\mathbf{x}_j^{t_i}$ will then be considered as an inequality constraint in (5), i.e., $||\mathbf{x}_i - \mathbf{x}_j^{t_i}|| > r_{robot} + r_j^{obs}$.

9.3.1 Algorithm

The algorithm in Algorithm 3 utilizes the equations mentioned earlier to update the path. Line 1 lists the required inputs for the algorithm. Line 2 lists the algorithm outputs. Lines 3-6 create time-based states from the distance between path states and the vehicle's velocity. Lines 7-11 create an obstacle trajectory using the time-based states and obstacle velocity. Checking for potential collisions between path states and obstacle trajectory states is done in Lines 12-19. If a collision is detected, Line 20 is triggered. Lines 20-21 add any obstacles on a collision course with the vehicle as static obstacles positioned on the collision point. Lastly, Line 22 returns the updated obstacle positions.

Algorithm 3 CollisionPrediction()

Input: $\mathbf{x}_i \rightarrow$ States, $\dot{\mathbf{x}}_0 \rightarrow$ Vehicle velocity, $\dot{\mathbf{x}}_j \rightarrow$ Obstacle velocity, $\mathbf{x}_j \rightarrow$ Obstacle position, $N \rightarrow$ The number of states in the prediction horizon, $|\mathbf{O}| \rightarrow$ Number of obstacles.

Output: $\mathbf{O} \rightarrow$ Obstacles.

$t_0 = 0.0$
$i = 0$
for $i < N$ **do**
    $t_{i+1} = t_i + \frac{||\mathbf{x}_{i+1} - \mathbf{x}_i||}{||\dot{\mathbf{x}}_0||}$,
$j = 0$
for $j < |\mathbf{O}|$ **do**
    $i = 0$
    for $i < N$ **do**
        $\mathbf{x}_j^{t_i} = \mathbf{x}_j + (\dot{\mathbf{x}}_j t_i)$
$j = 0$
Collisions = *empty*
for $j < |\mathbf{O}|$ **do**
    if $||\dot{\mathbf{x}}_j|| = 0$ **then**
        $i = 0$
        for $i < N$ **do**
            if $||\mathbf{x}_i - \mathbf{x}_j^{t_i}|| \leq r_{robot} + r_j^{obs}$ **then**
                Collisions $\leftarrow \mathbf{o}_j^{t_i} = [\mathbf{x}_j^{t_i} \;\; r_j^{obs}]^\top$
if $len($**Collisions**$) > 0$ **then**
    $\mathbf{O} \leftarrow$ **Collisions**
return O

9.4 Prediciton Horizon Extension

In some cases, the final state's ($\mathbf{x}_N$) location may be in collision, and the optimization process might not converge into a safe solution. However, implementing the **Prediction Horizon Extension** function has solved this issue. The function extends the prediction horizon if the last state of the horizon collides with an obstacle. The function moves the last state ($\mathbf{x}_N$) to the next available state outside the obstacle's range to ensure that the vehicle can safely follow the path. Figure 4 offers a visual representation to understand this process better.

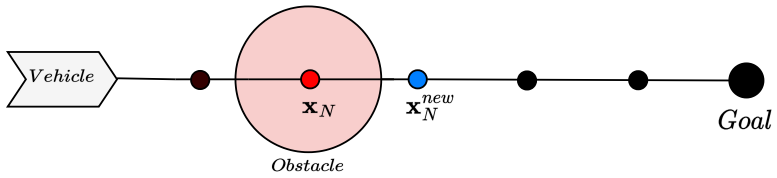


Figure 4: The figure shows the last state $\mathbf{x}_N$ of the prediction horizon being extended from its previous location (red) to a safe state (blue) outside of the obstacle range on the path

In Figure 4, the vehicle (gray) wants to reach a goal, following an initial path (black) between its current position and the goal. An obstacle (pink) lies in the path between the two states. The length of the prediction horizon is such that the last state, $\mathbf{x}_N$ (red), collides with the obstacle. To address this, the function relocates the state $\mathbf{x}_N$ to a position outside the range of the obstacle, represented by $\mathbf{x}_N^{new}$ (blue).

The process involves comparing the distance between $\mathbf{x}_N$ and obstacle $\mathbf{x}_j$ to determine if $\mathbf{x}_N$ collides with an obstacle and how many states it needs to move to be outside the obstacle's range. The condition in equation (16) determines if a state is in collision with an obstacle.

$$||\mathbf{x}_n - \mathbf{x}_j|| \leq r_{robot} + r_j^{obs} \tag{16}$$

The condition involves $\mathbf{x}_n \in \mathbf{P}$ which represents a state further down the path, where $n \in [N, goal]$, and r_{robot}, r_j^{obs} represent the radii of the vehicle and obstacle, respectively. The states will continue to test against this condition until it is no longer true. The new $\mathbf{x}_N$ will become the state that did not fulfill the condition.

9.4.1 Algorithm

The equations mentioned earlier are utilized to create the algorithm in Algorithm 4. Specific inputs required for the algorithm are listed in Line 1. The algorithm outputs are listed in Line 2. The for-loop in Line 5 goes through the states from the end of the prediction horizon to the end of the path, and the for-loop in Line 8 goes through the number of obstacles detected. Line 9 sets a condition to check if the state collides with an obstacle. If it does, collision is declared true in Line 10. If a collision occurs, the index variable is incremented by 1 in Lines 11-12. Once there is no collision, the loop is exited in Lines 13-14. The value of the index variable is added to the number of prediction states in Line 15 and is then returned in Line 16.

Algorithm 4 ExtendHorizon()

1: **Input**: $\mathbf{x}_i \rightarrow$ States, $N \rightarrow$ The number of states in the prediction horizon, $|\mathbf{O}| \rightarrow$ Number of obstacles, $|\mathbf{P}| \rightarrow$ number of path states.
2: **Output**: $N \rightarrow$ The number of states in the prediction horizon.
3: Index = 0
4: $i = N$
5: **for** $i < |\mathbf{P}|$ **do**
6: Collision = **FALSE**
7: $j = 0$
8: **for** $j < |\mathbf{O}|$ **do**
9: **if** $||\mathbf{x}_n - \mathbf{x}_j|| \leq r_{robot} + r_j^{obs}$ **then**
10: Collision = **TRUE**
11: **if** Collision = **TRUE then**
12: Index += 1
13: **if** Collision = **FALSE then**
14: **Break**
15: N = N + Index
16: **return** N

9.5 Active Perception

The last addition to the path planner is the **Active Perception** function. This function adds an attraction on the path to an observation point based on the constraint in (7), which is inspired by AquaVis [35]. The function involves the observation of visual objectives and modifying the planning process to enable observation of the point. Figure 5 is provided to enhance the understanding of this function.

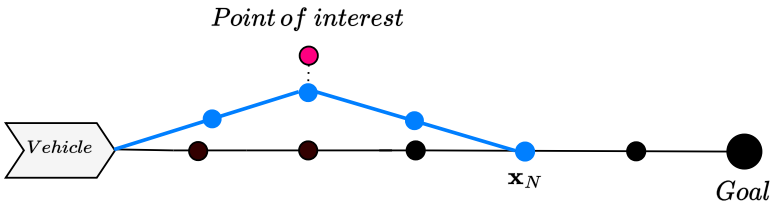


Figure 5: NMPC - The figure shows a point of interest placed in the aeria of the path, making the optimal path being attracted to the point

Figure 5 shows the initial path(Black) between the vehicle state and the goal. The closest state in the optimal path(Blue) is then pulled toward the observation point(magenta) until it reaches an observation distance. The vehicle is then able to observe the point from a safe distance.

9.6 Collision avoidance

The **Collision avoidance** function ensures that the path generated by the planner avoids obstacles in the environment, combining the NMPC framework with the previously presented functions. This is achieved from the constraints in Equation (5) and (6) that prevent the vehicle from colliding with any obstacles in its path. The function can be better understood through visualization in Figure 6.

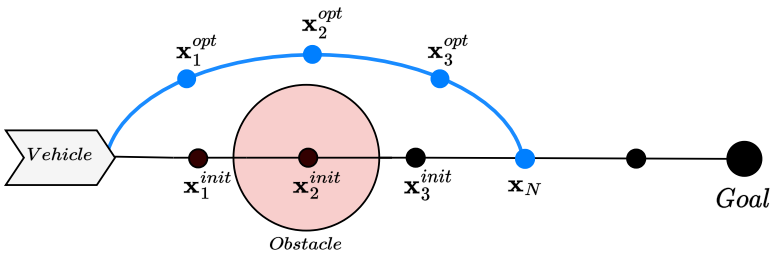


Figure 6: The figure shows a locally optimal path (blue) generated from the initial path (black), effectively avoiding the obstacle (pink) encountered.

Figure 6 displays a path indicated with black that connects the vehicle to a goal position, with a red obstacle intersecting the path. The path planner calculates an optimal path that avoids collisions by circumventing the obstacle, creating a safe path within the prediction horizon.

9.6.1 Algorithm

The path planner algorithm is summarized in Algorithm 5. The required inputs are listed in Line 1. The outputs are listed in Line 2. Line 3 initializes the optimal path to be the same as the initial path. In Line 4, the algorithm loops Lines 5 to 24 until it reaches the goal. Line 5 sets a condition to limit the loop to run at a non-zero input frequency. Line 6 updates the initial path using the function from Algorithm 1. Line 7 updates the number of states in the initial path using the function from Algorithm 2. Line 8 uses the function from Algorithm 3 to create static obstacles from potential dynamic collisions. Line 9 extends the prediction horizon using the function from Algorithm 4. Line 10 initializes the function variable of the MPC. Lines 11 to 12 initialize the objective function from equation (3). Line 13 Initializes the minimization function enabling the minimization of the objective function. In Lines 14 to 21, the constraints presented in equations (4) (5) (6) (7). Line 22 initializes the NLP solver, which is the SQP method in this case. Line 23 initializes the quadratic solver used in the SQP method. Line 24 utilizes the initial guess from Line 23 to solve the optimization problem, which is then extracted into the optimal path. Line 25 resets the timer used in Line 4, and the optimal

path is ultimately returned in Line 26.

Algorithm 5 Nonlinear Model Predictive Control (NMPC)

1: **Input**: $N \rightarrow$ number of states in the Prediction Horizon, $fz \rightarrow$ Update frequency, $\mathbf{x}_{goal} \rightarrow$ Goal, $|\mathbf{k}| \rightarrow$ Desired distance between the states, $\mathbf{P} \rightarrow$ Initial Path, $\mathbf{O} \rightarrow$ Obstacles, $\mathbf{K} \rightarrow$ Points of interest.

2: **Output**: $\mathbf{P}_{opt} \rightarrow$ Locally optimal Path
3: $\mathbf{P}_{opt} = \mathbf{P}$

4: **while** $\mathbf{x}_0 \neq \mathbf{x}_{goal}$ **do**
5: **if** timer $\geq \frac{1}{fz}$ **then**
6: $\mathbf{P}$ = InitialPathUpdate($\mathbf{P}$, $\mathbf{x}_{goal}$, $|\mathbf{k}|$)
7: $\mathbf{P}$ = PathResolution($\mathbf{P}$, $\mathbf{P}_{Opt}$, $|\mathbf{k}|$, N)
8: $\mathbf{O}$ = CollisionPrediction($\mathbf{P}$, $\dot{\mathbf{x}}_0$, $\dot{\mathbf{x}}_j$, $\mathbf{x}_j$, N, $|\mathbf{O}|$)
9: N = ExtendHorizon($\mathbf{P}$, N, $|\mathbf{O}|$, $|\mathbf{P}|$)
10: $f = 0$
11: **for** $i < N - 1$ **do**
12: $f = f + ||\mathbf{x}_{i+1} - \mathbf{x}_i||^2$
13: $Minimize(f)$
14: $SubjectTo(\mathbf{x}_0 = \mathbf{P}_{init})$
15: $SubjectTo(\mathbf{x}_N = \mathbf{P}_N)$
16: **for** $i <$ N **do**
17: **for** $j < |\mathbf{O}|$ **do**
18: $SubjectTo(|| \mathbf{x}_i - \mathbf{x}_j || > r_{robot} + r_j^{obs})$
19: $SubjectTo(\left|\left| \frac{\mathbf{x}_{i+1}+\mathbf{x}_i}{2} - \mathbf{x}_j \right|\right| > r_{robot} + r_j^{obs})$
20: **for** $k < |\mathbf{K}|$ **do**
21: $SubjectTo(r_{robot} < \min_{k \in \mathbf{K}} \{||\mathbf{x}_i - \mathbf{x}_k||\} \leq r_{robot} + d_{obsrv})$
22: $InitializeSolver$(SQP METHOD)
23: $InitializeQuadraticSolver$(QRQP)
24: $\mathbf{P}_{opt} = Solve(\mathbf{P})$
25: timer.reset()
26: **return** $\mathbf{P}_{opt}$

10 Simulation Experiments

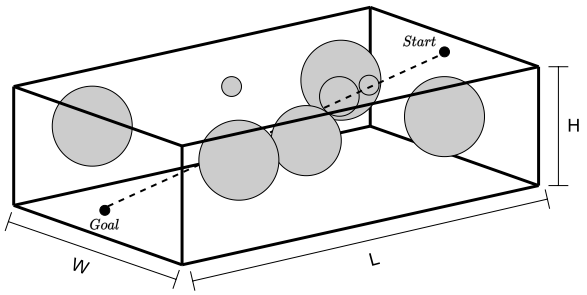
The simulation environment adopts FhSim to execute the simulations using the NED coordinate system. The vehicle's state is defined by a six-dimensional vector $\eta = [x, y, z, \phi, \theta, \psi]^\top$, where x, y, and z represent the position of the vehicle, and ϕ, θ and ψ represent its orientation.

10.1 FhSim - Underwater Simulations

FhSim is a software platform for mathematical modeling and numerical simulations in C++. It places great emphasis on simulation performance and marine systems modeling. Its modular design lets users define simulations by connecting independent objects. FhSim features a collection of essential mathematical marine models for simulating a marine environment, particularly on fish farming [36, 37].

10.2 Case Study 1: Cluttered Environment

The first environment chosen to test the path planner is a cluttered box, highly dense with obstacles. This assesses the planner's performance in a complex and restricted environment. The case study will help evaluate the path planner's ability to avoid collisions in a challenging scenario. Figure 7 represents the environment where a generic box with length(L) = $31m$, width(W) = $16m$, and height(H) = $8m$ is filled with obstacles. The test is conducted by assigning the vehicle to travel between two predefined points within the volume and letting the path planner identify a feasible route by avoiding the obstacles. For more details see A.1

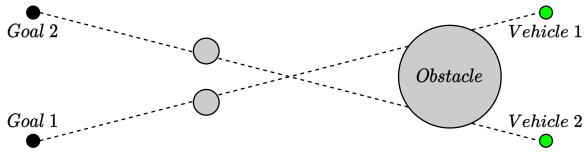


Prediction Horizon (N)	Step Length ($\|\mathbf{k}\|$)
10	1.0 (m)

Figure 7: Case Study 1 - Cluttered Box

10.3 Case Study 2: Multi-robot coordination

For Case Study 2, the objective is to test the path planner's ability to handle complex situations by simulating a scenario where two vehicles will collide with each other. The vehicles will use the same path planner and logic without communicating with each other in an environment that includes obstacles. Figure 8 shows the environment, which includes two vehicles, three obstacles, and two different goals. The two vehicles start from a common starting point connected to the goals through an initial path. For more details see A.2

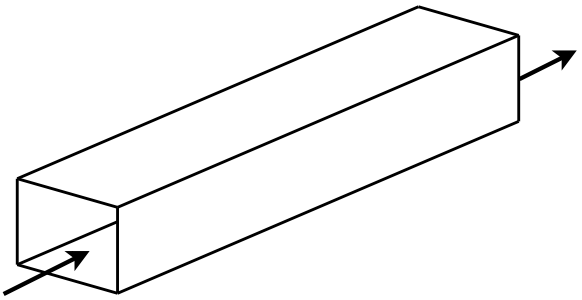


Prediction Horizon (N)	Step Length ($\|\mathbf{k}\|$)
10	1.0 (m)

Figure 8: Case Study 2 - Several Vehicles & Obstacles

10.4 Case Study 3: Narrow Passage

Case Study 3 aims to test the path planner's capability to safely navigate through a 10x1.4x1.4m narrow passage, which will evaluate its effectiveness in real-world scenarios and its ability to handle challenging situations. This environment was selected because other path planners, including the potential field method, have struggled with similar environments [17]. Figure 9 illustrates the narrow passage, with the entrance and exit indicated by an arrow. For more details see A.3

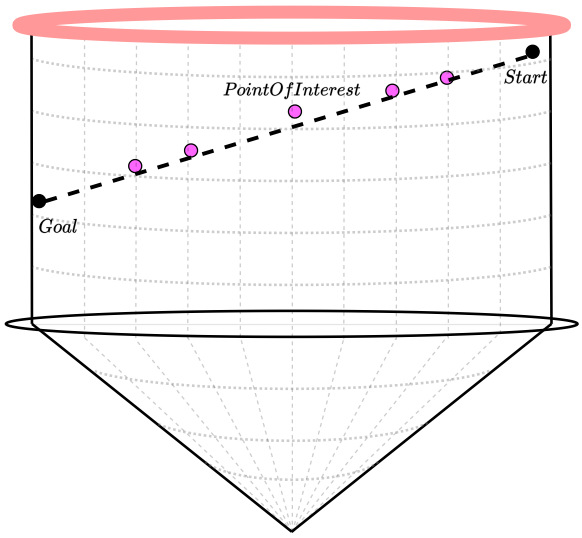


Prediction Horizon (N)	Step Length ($\|\mathbf{k}\|$)
10	1.0 (m)

Figure 9: Case Study 3 - Narrow Passage

10.5 Case Study 4: Inspect Fish Cage

The Environment in Case Study 4 is intended to evaluate the path planner's ability to navigate to inspect points of interest within a fish cage environment, with a radius = $25m$ and depth = $25m$. The presence of points of interest will enable assessing the planner's ability to observe them effectively. Figure 10 displays the fish cage environment used for this study. The test is conducted by assigning the vehicle to navigate between two predetermined points within the fish cage and letting the path planner identify a feasible route and be attracted by points of interest placed near the cage wall. For more details see A.4

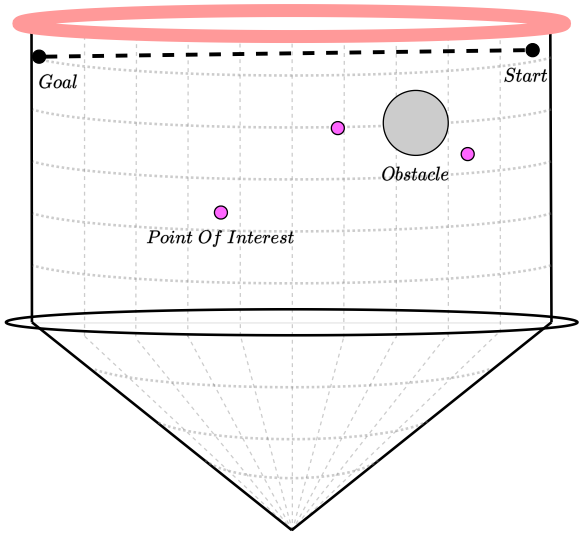


Prediction Horizon (N)	Step Length ($\|\mathbf{k}\|$)
20	1.0 (m)

Figure 10: Case Study 4 - Fish Cage

10.6 Case Study 5: Inspect Fish Cage with Obstacles

Case Study 5 is focused on aquaculture and utilizes an environment similar to the one in 10.5 but with added obstacles. The aim is to test the path planner's ability to handle attraction and repulsion functions, specifically obstacle avoidance, while navigating to points of interest placed on the cage wall. Figure 11 visually represents the environment. For more details see A.5

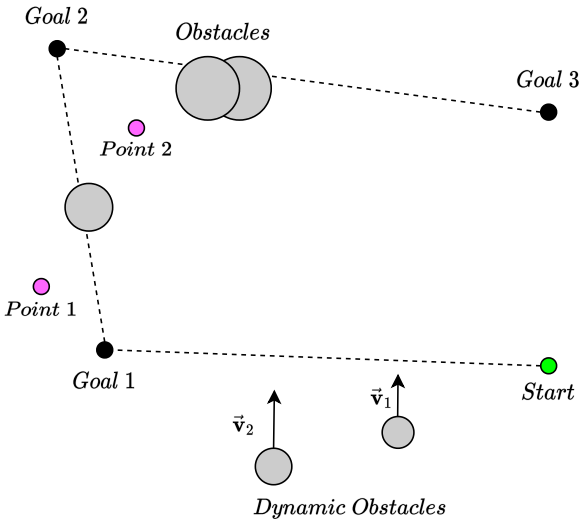


Prediction Horizon (N)	Step Length ($\|\mathbf{k}\|$)
10	1.0 (m)

Figure 11: Case Study 5 - Fish Cage with obstacle and points of interest

10.7 Case Study 6: Obstacle-rich Environment with Points of Interest

Case Study 6 is designed to challenge the path planner by presenting a variety of situations that require the use of all its implemented functions, demonstrating its capability to navigate effectively in a diverse environment. The environment, illustrated in figure 12, features multiple dynamic obstacles, points of interest(Dark Pink), and static obstacles. The scenario begins at a starting point (marked in green) and is linked to an initial path via several intermediate goals. For more details see A.6



Prediction Horizon (N)	Step Length ($\|\mathbf{k}\|$)
10	1.0 (m)

Figure 12: Case Study 6 - Obstacle-rich Environment with Points of Interest

Part IV
Results

11 Path-Planning

11.1 Case Study 1: Cluttered Environment

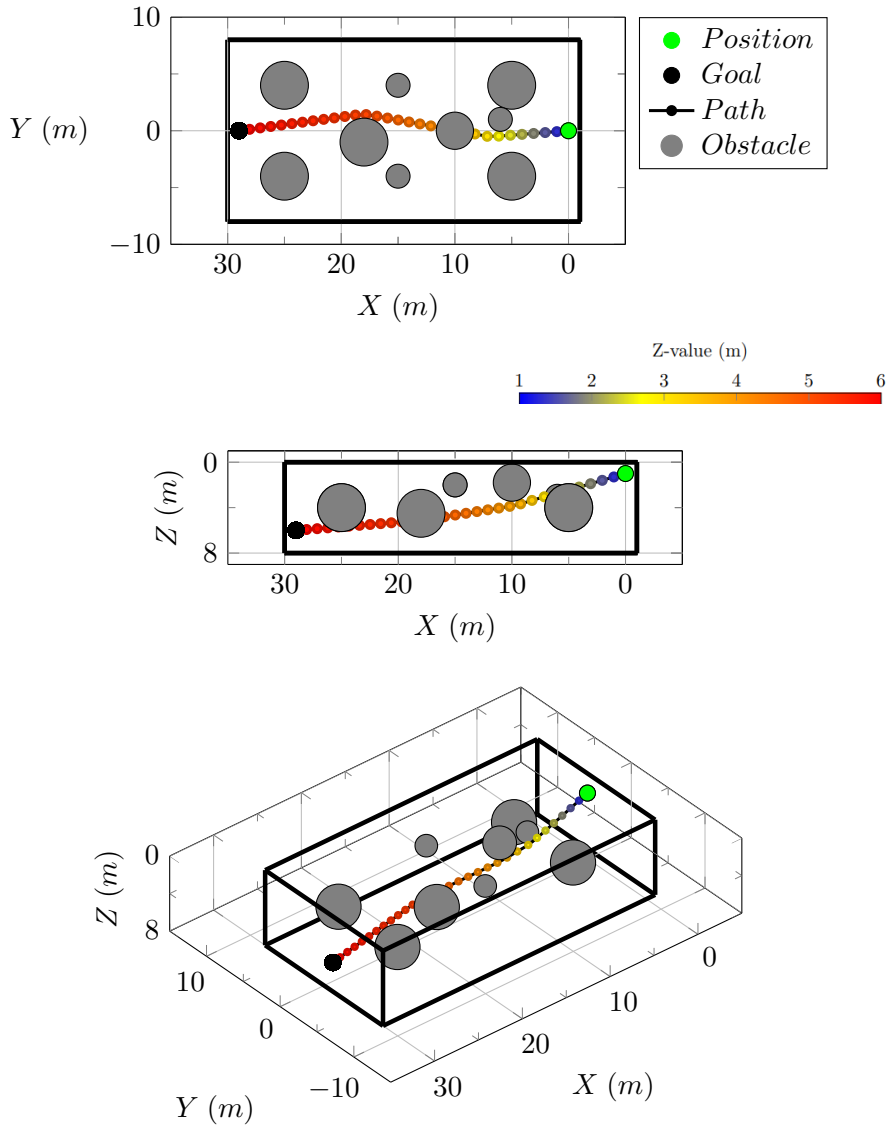


Figure 13: NMPC (N = 10) - The figure shows the path planner performing in a cluttered restricted environment from three different points of view.

Figure 13 shows the optimized path obtained from three viewpoints. The first viewpoint shows the path on the x,y-plane, while the second shows it on the x,z-plane. The third view is a 3D view that provides a complete path visualization. The path navigates from a starting point (represented by the color green) towards a goal (represented by black) while avoiding obstacles and staying within the box's confines. A color gradient shows the path's depth, where the depth is positive.

11.1.1 Time-varying Visualization

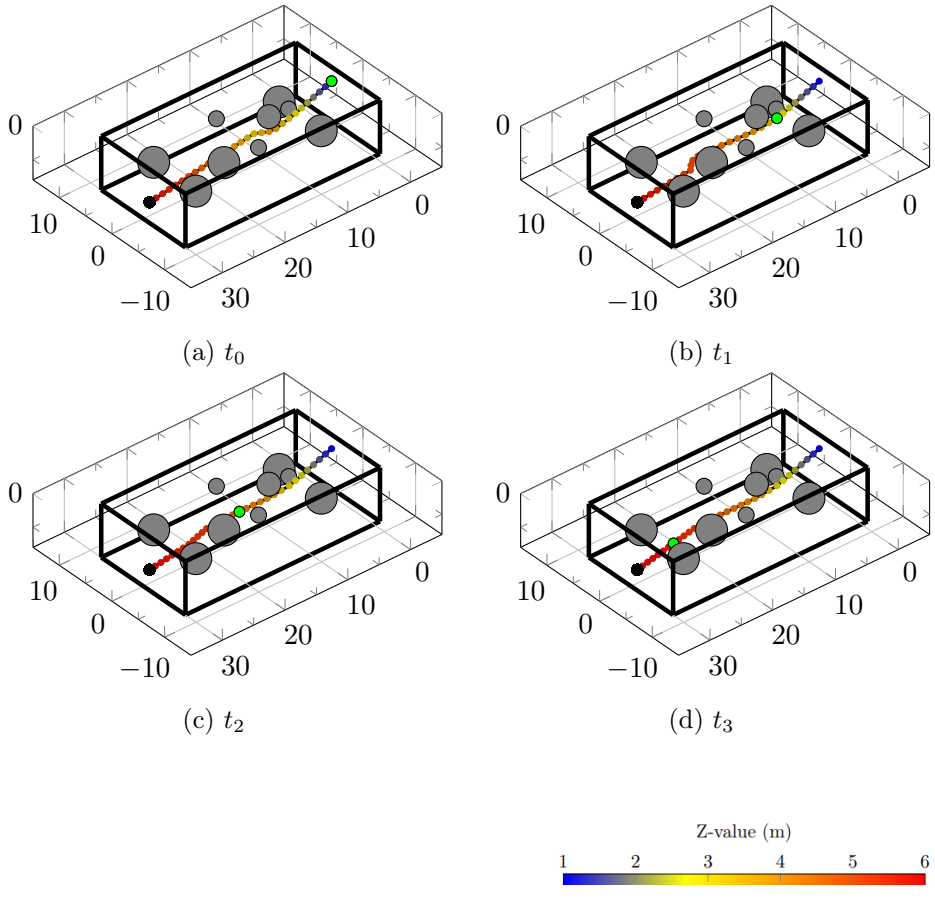


Figure 14: NMPC (N = 10) - The figure shows the position of the vehicle and the navigation of the path at different points in time, navigating in a cluttered environment while being restricted by a box.

Figure 14 shows the path evolution from time t_0 (start) to t_3 (end). The path adapts and updates as the vehicle moves toward its destination. A color gradient indicates the positive depth along the path.

11.2 Case Study 2: Multi-robot coordination

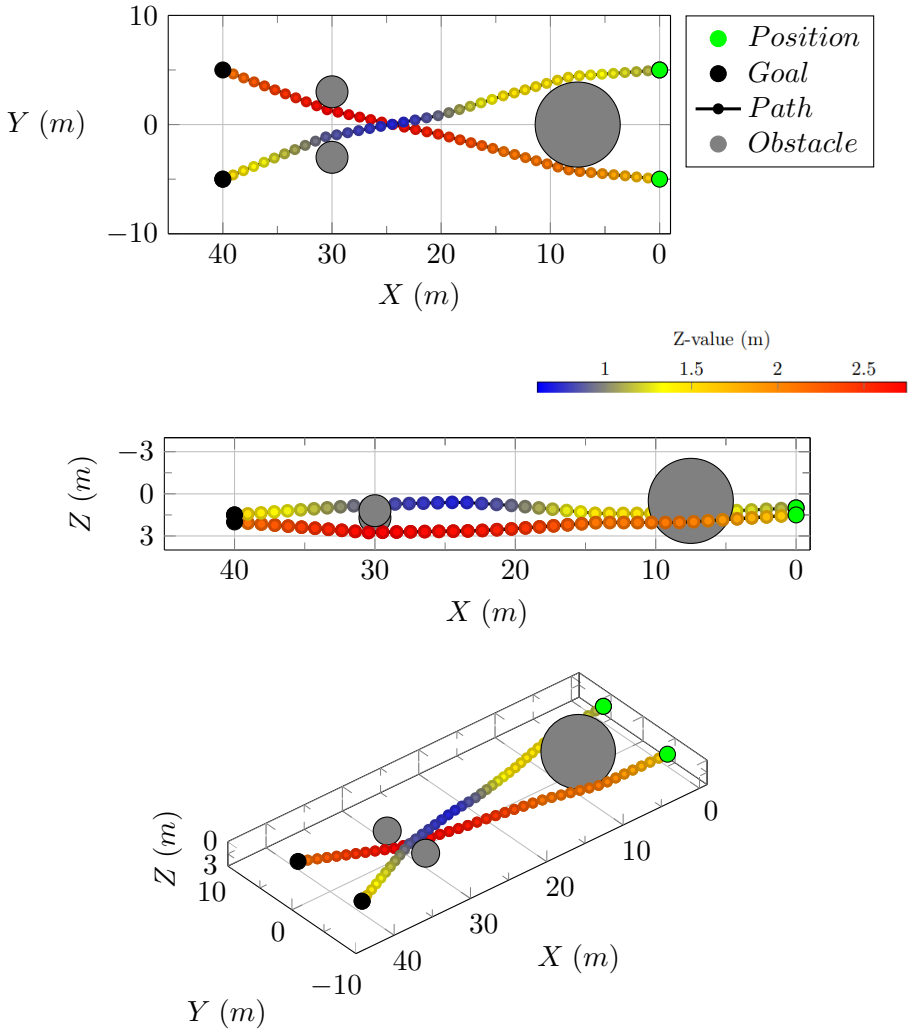


Figure 15: NMPC (N = 10) - The figure shows two vehicles on a collision course, navigating to not collide with each other or in any other obstacle.

Figure 15 displays the path of two vehicles from three perspectives, the xy-plane, the xz-plane, and a full 3D view. The green dots indicate the vehicles which navigate around gray obstacles. Additionally, the vehicles avoid colliding by passing over and under each other. A color gradient shows the path's depth, where the depth is positive.

11.2.1 Time-varying Visualization

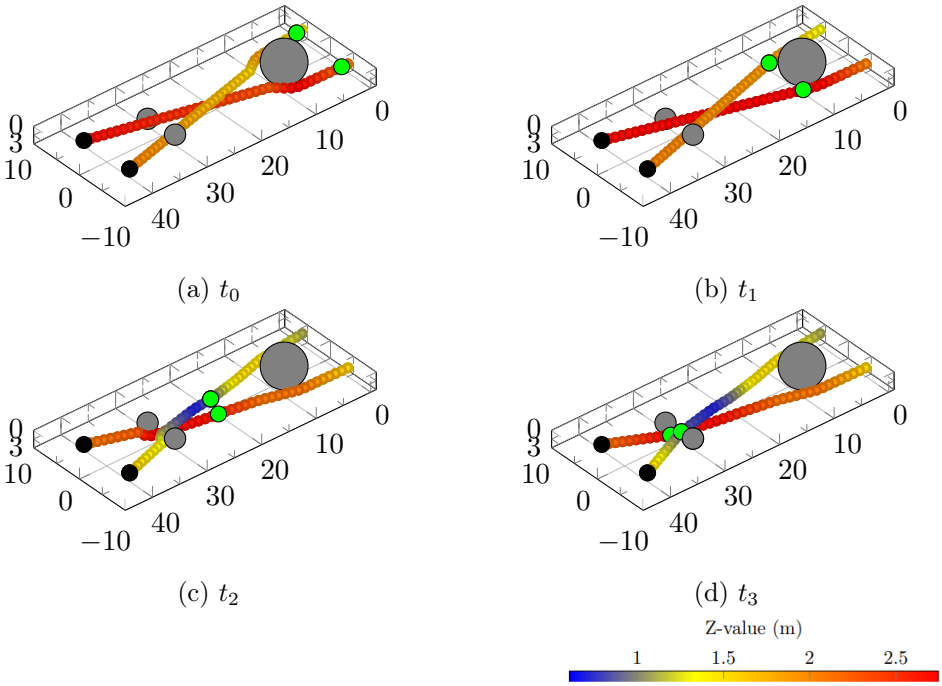


Figure 16: NMPC (N = 10) - The figure shows two vehicles avoiding collision with obstacles and each other at different points in time.

Figure 16 displays the paths of two vehicles as they advance toward their respective destinations. The paths are recalculated at each step, with t_0 representing the time at the beginning of the path when the vehicle is close to the starting point and t_3 representing the time at the end of the path when the vehicle is close to the goal. The time in between is represented by t_1,t_2 respectively. The green dots represent the two vehicles, which calculate their unique paths. The plot illustrates that the vehicles were initially headed toward a collision, but they predict and avoid the collision by choosing different paths. Additionally, they successfully navigated around any other obstacles that came their way. A color gradient shows the path's depth, where the depth is positive.

11.3 Case Study 3: Narrow Passage

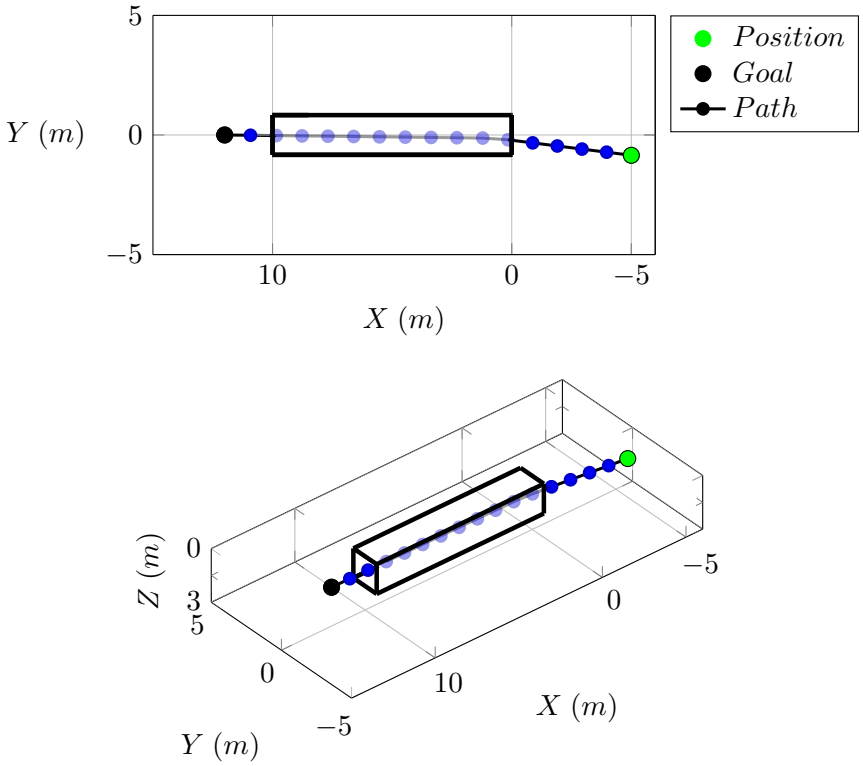
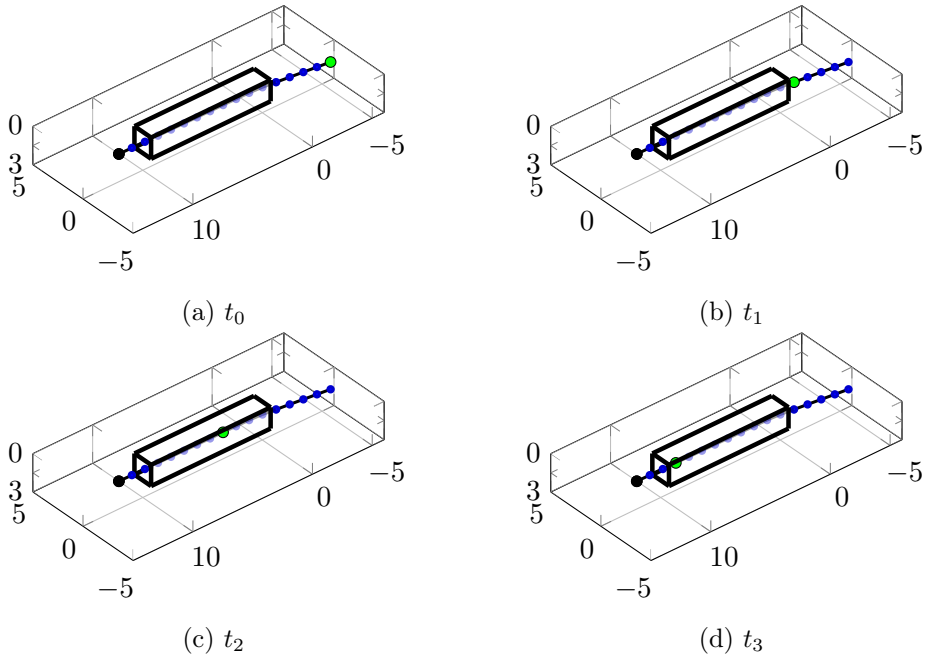


Figure 17: NMPC (N = 10) - The figure shows the performance of the path planner as it navigates through a narrow passage.

Figure 17 shows two viewpoints of path planning within a fish cage. The first viewpoint is a 2D representation of the x,y-plane, while the second is a full 3D representation. The plot shows a path from a starting position and through a narrow passage to a goal position.

11.3.1 Time-varying Visualization



(a) t_0 (b) t_1

(c) t_2 (d) t_3

Figure 18: NMPC (N = 10) - The figure shows the path planner navigating through a narrow passage at different points in time.

Figure 18 shows the vehicle's path towards its destination, with t_0 representing the time at the beginning of the path when the vehicle is close to the starting point and t_3 representing the time at the end of the path when the vehicle is close to the goal. The time in between is represented by t_1,t_2 respectively.

11.4 Planning efficiency

During the testing phase of the path planner, it was crucial to assess its planning time to determine its usefulness in real-world situations. The planning time was tested in three simulation environments, where the first was an empty space with no obstacles. In the second simulation, the path avoids one obstacle. In the third simulation, the path avoids four obstacles. The performance of the planning time can be seen in figure 19.

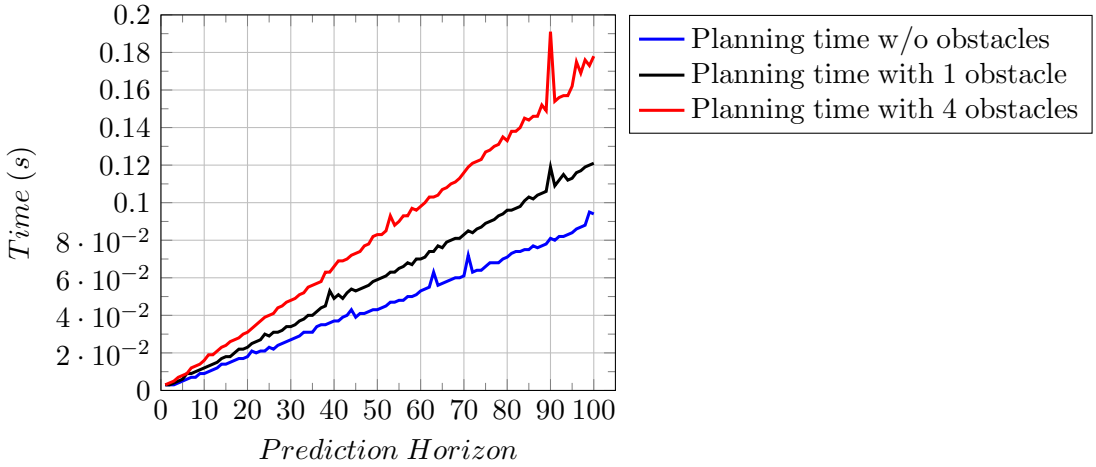


Figure 19: The figure shows the planning time of the MPC, which suggests that it is linear scaling with respect to the prediction horizon, minimal, and dependent on the number of obstacles.

The plot depicted in Figure 19 showcases the relationship between the planning time for the optimal path, the number of states in the prediction horizon, and the impact of obstacles on the planning time. It is suggested from the result that the planning time of the NMPC increases linearly as the number of encountered obstacles increases.

12 Path Planning with Active Perception

12.1 Case Study 4: Inspect Fish Cage

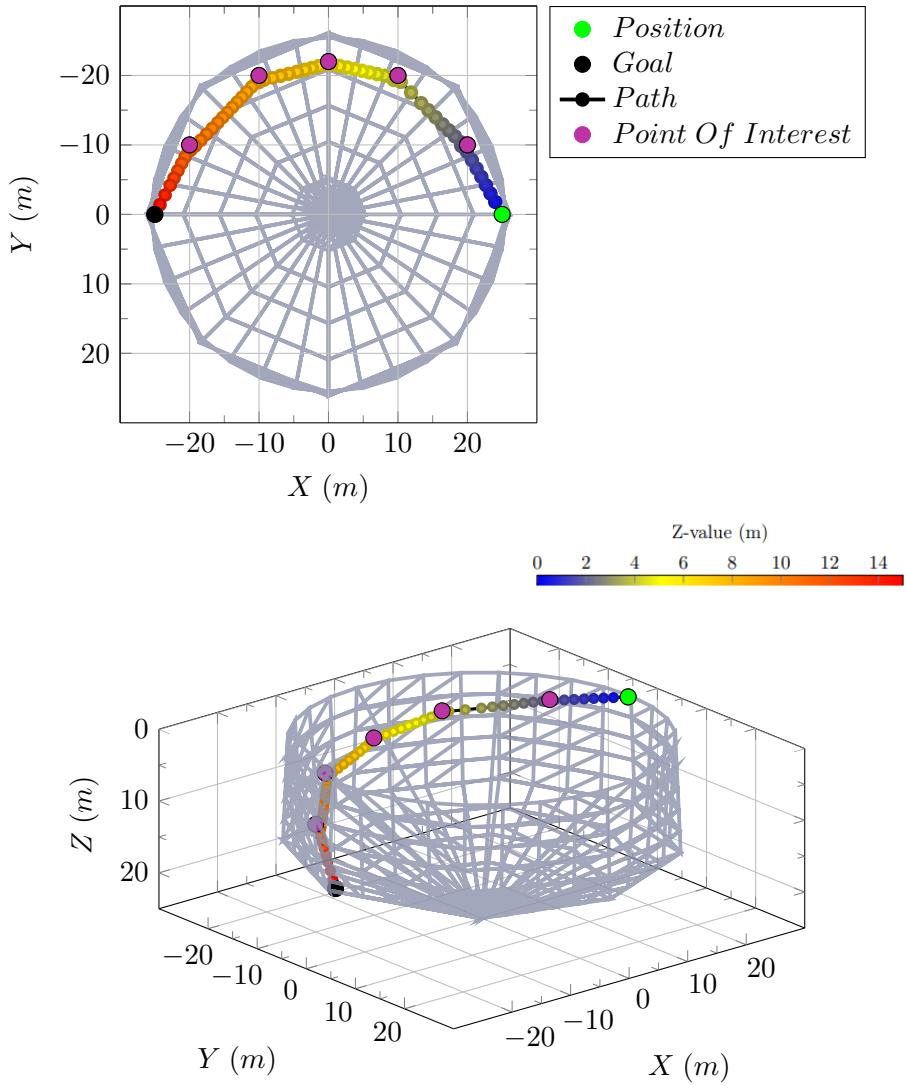


Figure 20: NMPC (N = 20) - The figure shows the path planner performing in a fish cage, where there are five points of interest within the cage.

Figure 20 provides two viewpoints of the path planning result within a fish cage. The first viewpoint is a 2D representation of the x,y-plane, while the second is a full 3D representation. The figure showcases the optimized path from a starting position to a goal, with five points of interest along the

way. The path follows a curved trajectory, passing the points of interest and heading toward the goal. A color gradient shows the path's depth, where the depth is positive.

12.1.1 Time-varying Visualization

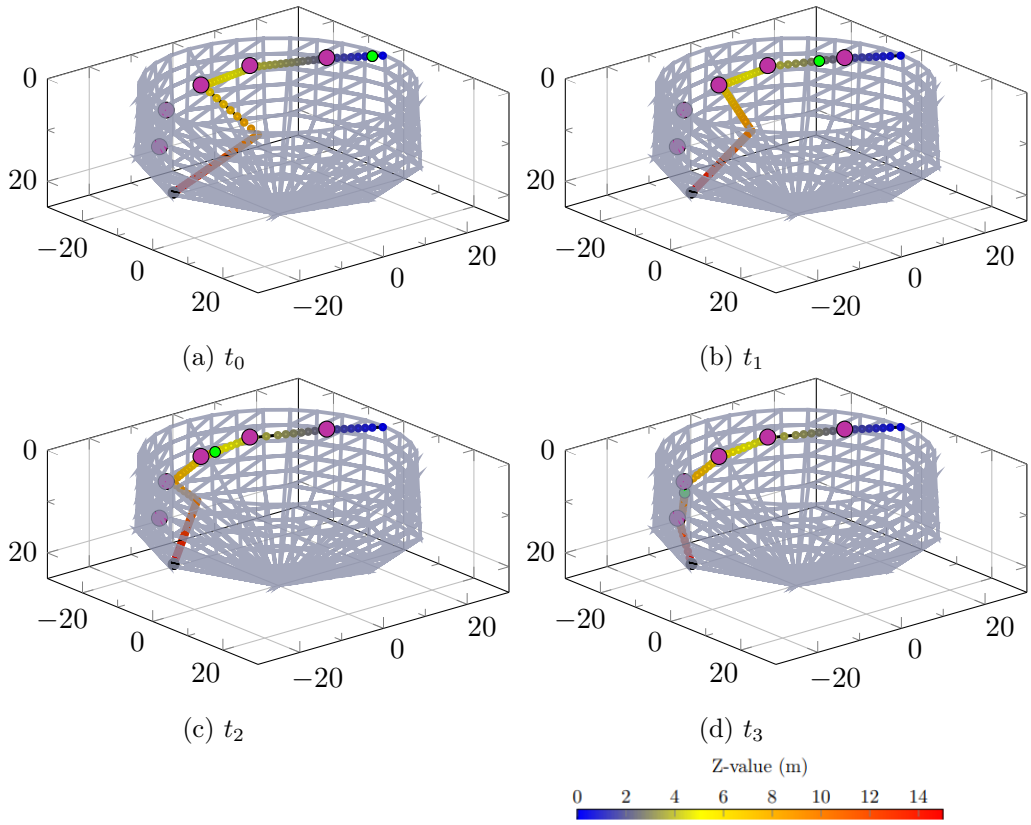


Figure 21: NMPC (N = 20) - The figure shows the navigation of the path planner at different points in time, showing how the path updates as the vehicle moves.

Figure 21 shows the evolution of the path over time as the vehicle moves toward its destination. The time at the beginning of the path is represented by t_0, and t_3 represents the time at the end of the path when the vehicle is close to the goal. The time in between is represented by t_1,t_2 respectively. It is a visual representation of the path adapting and updating as the vehicle moves forward. The angle at different points represents the deviation between the initial and optimal paths. The initial path follows the vehicle, so the last part of the path is directed toward it. A color gradient shows the path's depth, where the depth is positive.

12.2 Case Study 5: Fish cage with obstacle

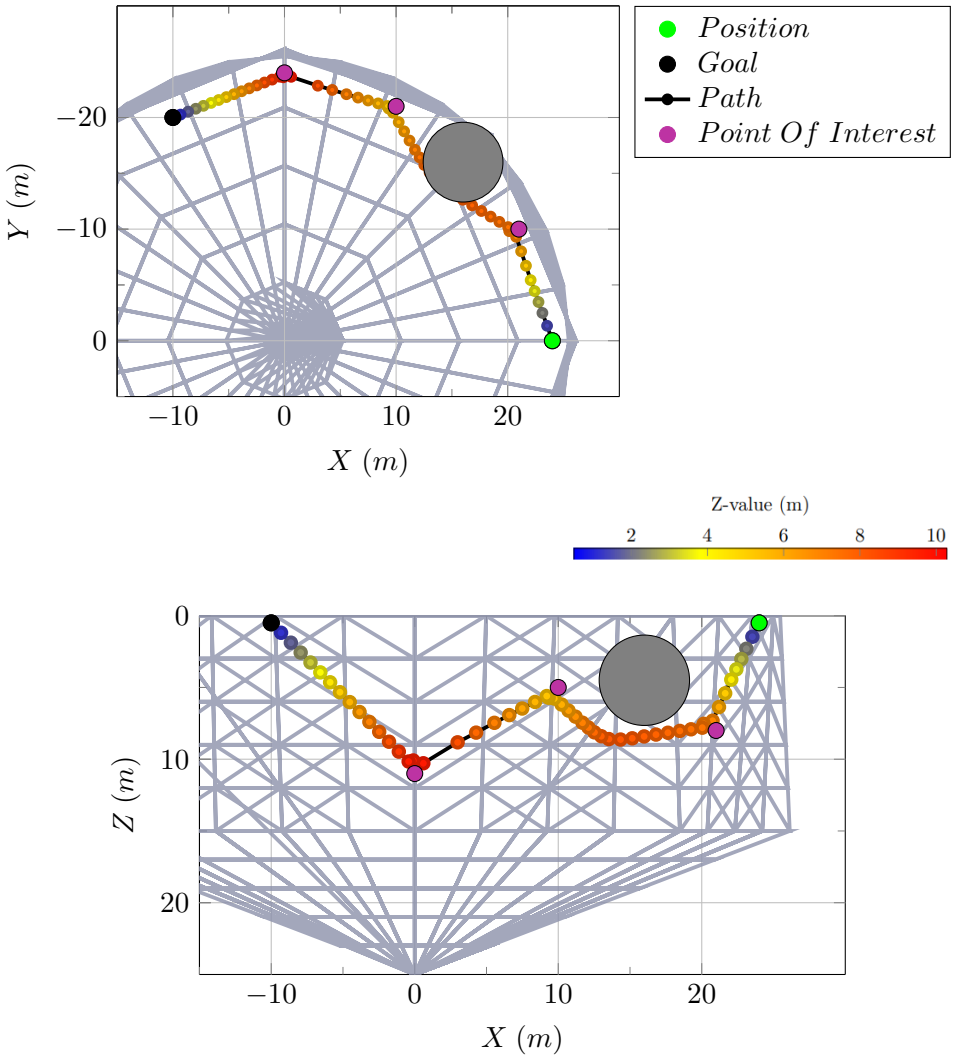


Figure 22: NMPC (N = 10) - The figure shows the path planner navigating in a fish cage attracted to points of interest and repulsed by obstacles.

Figure 22 displays two viewpoints of the path within a fish cage, one in the xy-plane and the other in the xz-plane. The path has a fixed starting point and a goal point. The path is affected by an attraction force from points of interest and a repulsion force from obstacles. The color gradient indicates the depth of the path. A color gradient shows the path's depth, where the depth is positive.

12.2.1 Time-varying Visualization

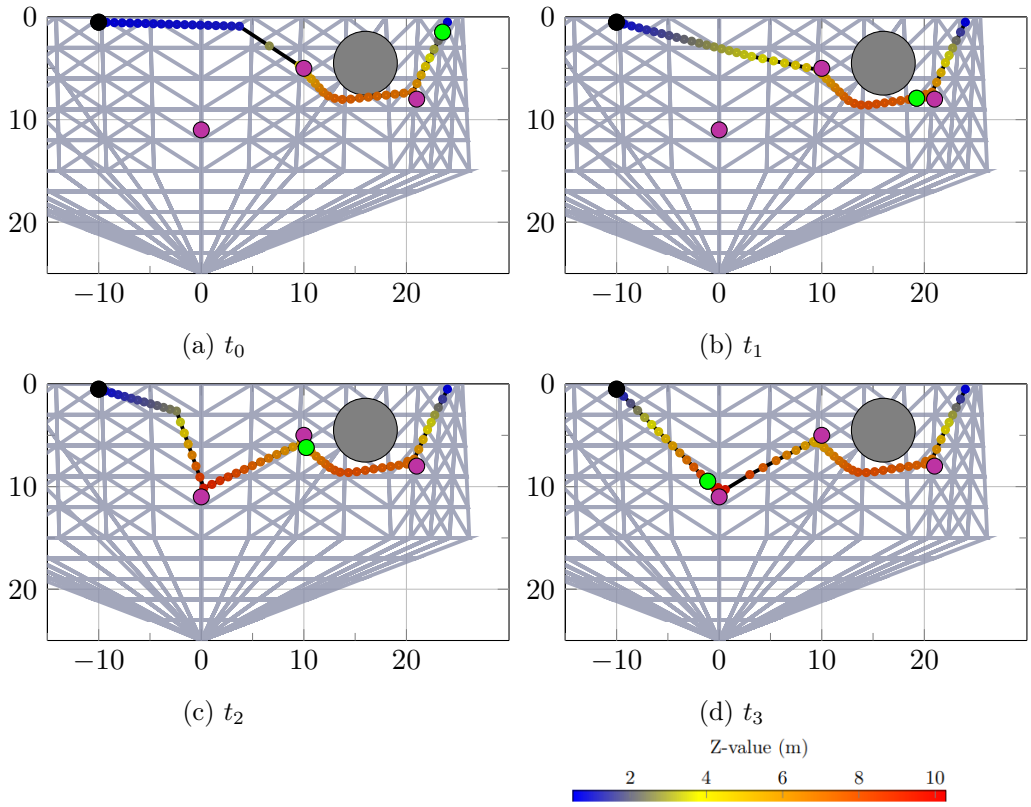


Figure 23: NMPC (N = 10) - The figure shows the path planner navigating in a fish cage at different points in time, attracted to points of interest and repulsed by obstacles.

Figure 23 displays the path of a vehicle as it advances toward its respective destinations. The path is constantly recalculated with t_0 representing the time at the beginning of the path when the vehicle is close to the starting point and t_3 representing the time at the end of the path when the vehicle is close to the goal. The time in between is represented by t_1,t_2 respectively. The figure indicates that the path is drawn towards points of interest as the prediction horizon progresses while avoiding an obstacle by passing underneath it. A color gradient shows the path's depth, where the depth is positive.

12.3 Case Study 6: Obstacle-rich Environment with Points of Interest

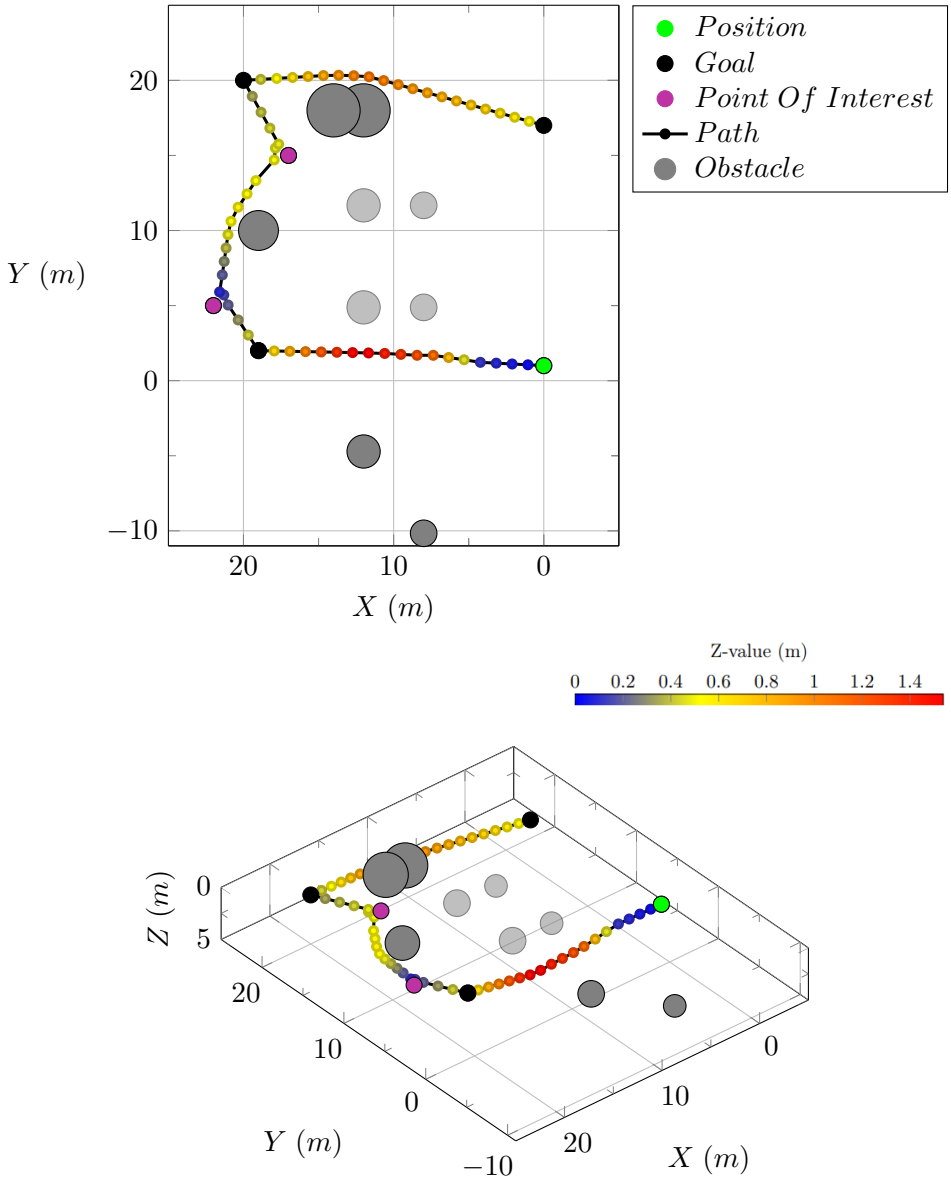
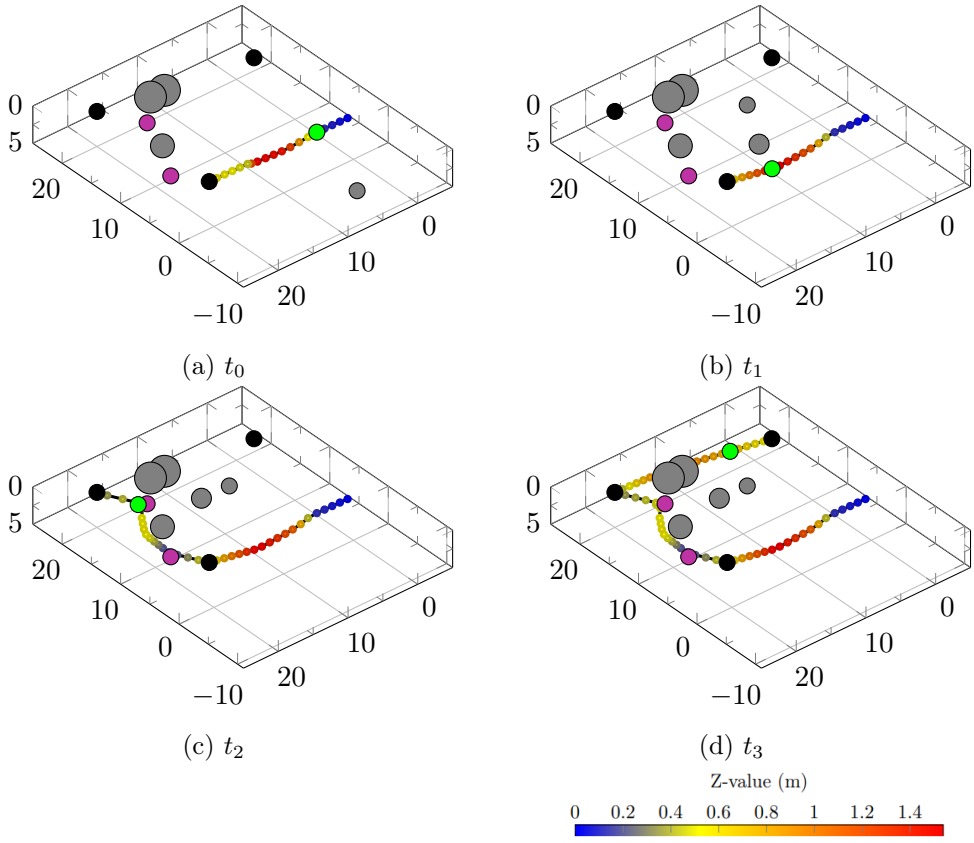


Figure 24: NMPC (N = 10) - The figure shows the path planner navigating in a diverse and complex environment, testing all the functions of the path planner.

Figure 24 illustrates the optimized path from two viewpoints, 2D on the x,y-plane and 3D. The path starts at the initial position and goes to the

first goal while avoiding two dynamic obstacles that are meant to collide with the vehicle. The figure also shows the path from the first goal to the second goal. In this part of the path, the vehicle avoids a static obstacle while simultaneously observing two points of interest. The last scenario in the figure is between the second and third goals, where the path planner successfully avoids two larger static obstacles. A color gradient shows the path's depth, where the depth is positive.

12.3.1 Time-varying Visualization



(a) t_0 (b) t_1 (c) t_2 (d) t_3

Figure 25: NMPC (N = 10) - The figure shows the path navigating in a diverse and complex environment at different points in time.

Figure 25 shows the vehicle's path from t_0 (start) to t_3 (end), with t1 and t2 in between. Each position represents the vehicle's location along the sub-paths, showcasing updates for different scenarios and multi-goal navigation capability. A color gradient indicates positive depth along the path

Part V
Discussion

The results presented in Section IV demonstrate the effectiveness and efficiency of the MPC-based path planner in various complex environments, testing path planning with and without active perception.

13 Path-Planning

The planner's performance was evaluated through multiple case studies. Each intended to showcase different aspects of its performance. In the first case study, presented in Section 11.1, the planner demonstrated its effectiveness in navigating a challenging environment characterized by restrictions and clutter. The planner successfully navigated the vehicle around static obstacles by incorporating obstacle avoidance constraints, ensuring safe path generation. The dynamic nature of the planner's path planning was visually depicted through time-varying visualizations, showcasing the continuous updates to the planned path as the vehicle made its way through the environment. This real-time path adaptation allows the vehicle to respond to changing obstacle configurations and ensure efficient and safe navigation. The results from this case study highlight the planner's ability to handle cluttered and restricted environments while maintaining the desired path toward the goal. However, it is essential to note that this case study does not address the planner's performance when faced with dynamic obstacles.

Section 11.2 demonstrates the planner's ability to navigate and avoid collisions with other vehicles in real time. The planner predicts and prevents potential collisions by treating other vehicles as dynamic obstacles and simultaneously avoiding static obstacles, as shown in the dynamic visualization. These results highlight the importance of reliable collision avoidance mechanisms in complex environments, ultimately elevating the safety and reliability of autonomous navigation. The performance and findings presented in Section 11.2 highlight the planner's effectiveness in ensuring safe and reliable navigation in challenging underwater environments.

While the previous case studies highlighted the planner's reliable navigation in cluttered environments, there are instances where the vehicle needs to navigate through tight spaces. In Section 11.3, the case study showcases the planner's ability to maneuver through narrow passages while avoiding collisions with the surrounding environment. Despite being a significant challenge for many path planners, such as the potential field method [17], the results unequivocally demonstrate the planner's proficiency in this task.

Navigating through narrow passages without colliding with the environment is crucial for autonomous underwater vehicles operating in complex and constrained environments. The planner successfully showcases its exceptional aptitude for path planning and collision avoidance, even in challenging scenarios. Its effectiveness in navigating through tight spaces without collisions opens up possibilities for various applications in underwater exploration, inspection, and maintenance tasks. The planner's capability to maneuver successfully through confined spaces showcases its potential to enhance autonomous underwater navigation systems' safety, efficiency, and versatility.

The planning efficiency of the NMPC presented in 11.4 is depicted in Figure 19, which shows that the planning time is small and linear with respect to the prediction horizon, enabling the NMPC to be applied in real-time path planning. Although the planning time increases with the number of obstacles, the effect on planning time is negligible in underwater robotics, given that the horizon is shorter than 100 states and the number of colliding obstacles is low.

14 Path-Planning with Active Perception

The previous case studies only evaluated the path planner's safe navigation in challenging underwater environments. However, a new function was introduced in Section 12 - active perception. This function enables the path to shift towards points of interest, enabling the planner to concentrate on exploring and monitoring specific areas. The case study showcased in 12.1 demonstrates how the planner adjusts its trajectory to navigate a cage with points of interest. By incorporating these points, the planner effectively tracks and monitors them, enhancing exploration and monitoring in underwater environments. The time-varying visualization reveals the planner's dynamic path updates based on new points of interest, enabling informed decision-making and optimal path selection. This enhances its versatility and adaptability, making it a valuable tool for underwater exploration, research, and monitoring tasks requiring attention to specific areas or objects.

The case study in Section 12.2 demonstrates the planner's ability to simultaneously avoid obstacles and be attracted to points of interest in an enclosed environment. By adding attraction to points of interest in the planning process, the planner can focus on exploring and monitoring specific areas while ensuring safe navigation. This capability enhances the planner's adaptability and versatility in complex underwater scenarios, improving efficiency and information gathering during missions.

The case study in Section 12.3 highlights the planner's exceptional ability to handle all the presented functions simultaneously. By integrating the prediction and avoidance of dynamic obstacles, attraction to points of interest, and navigation around static obstacles, the planner offers a comprehensive solution for autonomous underwater navigation. This capability ensures safe and collision-free navigation in dynamic underwater environments, enables efficient exploration and monitoring of specific areas of interest and ensures path generation without collisions with fixed objects. The successful demonstration of these functions showcases the planner's robustness and reliability, with implications for various underwater applications such as research, exploration, environmental monitoring, and infrastructure inspection. The planner's capacity to handle multiple functions simultaneously opens up new opportunities for optimizing underwater missions and improving overall mission success.

15 Areas of Improvement

During the testing of the planner, despite its overall safe navigation within the environment, several issues came to light. The path planner utilized in this project was coded in C++ via Visual Studio, mainly using the SQP method along with the QRQP as the QP solver for the NLP solver. However, it was discovered that the SQP method with QRQP solver was fragile and sensitive to failure in specific environments [38]. To address this issue, simple tests were conducted in Python, utilizing a robust solver, IPOPT [38], leading to improved performance and the ability to handle environments SQP and QRQP could not. However, this improvement could not be implemented because of restrictions in the build system of FhSim. Another disadvantage of the path planner is that it does not consider vehicle dynamics in the optimization problem, which could improve its quality and robustness. Additionally, it was discovered that the optimization problem was technically not nonlinear. This further emphasizes the importance of including dynamics since the problem is nonlinear in real-life underwater robotics.

Part VI
Conclusions and Recommended Future Work

In conclusion, the MPC-based path planner demonstrates effectiveness and efficiency in navigating complex environments. This planner can handle multiple functions simultaneously, such as avoiding static and dynamic obstacles while being drawn to points of interest. This planner has undergone successful testing in numerous case studies. Its planning time is small and linear, making it suitable for real-time path planning, and the number of obstacles does not significantly affect its planning time. However, the robustness of the NLP solver was an issue, but it can be improved by replacing it with a more robust planner such as IPOPT. Overall, the performance of this planner is highly promising for real-world navigation applications in underwater environments, particularly for Autonomous Underwater Vehicles. The planner has proven efficient and effective in navigating complex scenarios like cluttered, narrow, and dynamically changing environments. With more refinement and development, this planner could greatly benefit the advancement of underwater robotics and related industries.

16 Future Work

In terms of future work, several directions can be explored to enhance the path planner further. One avenue to investigate is using different NLP solving methods to improve the robustness of the planner. For instance, the IPOPT method has shown promise in previous studies and could be a worthwhile option to explore as an alternative solver [38].

Another possibility is to assess the performance of the SQP method with a different quadratic solver. Using a different solver may enhance the robustness of the NLP solver, thereby improving the overall performance of the path planner.

Additionally, integrating vehicle dynamics into the optimization process is worth exploring. This enhancement involves incorporating the system's dynamics into the path planning algorithm, leading to more accurate predictions and improving the quality and robustness of the generated paths.

Furthermore, updating the collision prediction function could be beneficial. Currently, the function does not account for future collisions along the optimized path, which may result in potential collisions with dynamic obstacles. Developing a collision prediction mechanism that considers the optimal path

would enhance the planner's ability to avoid collisions and improve overall safety.

Overall, investigating different NLP solving methods, integrating vehicle dynamics, and improving the collision prediction function are promising avenues for future research. These enhancements can further improve the path planner's robustness, performance, and safety, ultimately advancing the field of autonomous underwater navigation.

Part VII
Appendices

A Case Study Parameters and Configurations

A.1 Case Study 1: Cluttered Environment

Start Position [x,y,z]	**Goal** [x,y,z]
[0.0 , 0.0 , 1.0]	[29.0 , 0.0 , 6.0]

Obstacle $\mathbf{o}_j$	**Position** [x,y,z]	**Radius**(m)
$\mathbf{o}_0$	[18.0 , −1.0 , 4.5]	2.0
$\mathbf{o}_1$	[10.0 , 0.0 , 1.8]	1.5
$\mathbf{o}_2$	[6.0 , 1.0 , 3.0]	1.0
$\mathbf{o}_3$	[5.0 , 4.0 , 4.0]	2.0
$\mathbf{o}_4$	[5.0 , −4.0 , 4.0]	2.0
$\mathbf{o}_5$	[25.0 , 4.0 , 4.0]	2.0
$\mathbf{o}_6$	[25.0 , −4.0 , 4.0]	2.0
$\mathbf{o}_7$	[15.0 , 4.0 , 2.0]	1.0
$\mathbf{o}_8$	[15.0 , −4.0 , 2.0]	1.0

A.2 Case Study 2: Multi-robot coordination

Vehicle	**Start Position** [x,y,z]	**Goal** [x,y,z]
1	[0.0 , 5.0 , 1.0]	[40.0 , −5.0 , 1.5]
2	[0.0 , −5.0 , 1.5]	[40.0 , 5.0 , 2.0]

Obstacle $\mathbf{o}_j$	**Position** [x,y,z]	**Radius**(m)
$\mathbf{o}_0$	[7.5 , 0.0 , 4.5]	0.5
$\mathbf{o}_1$	[30.0 , −3.0 , 1.2]	1.5
$\mathbf{o}_2$	[30.0 , 3.0 , 1.7]	1.5

A.3 Case Study 3: Narrow Passage

Start Position [x,y,z]	**Goal** [x,y,z]
[−5.0 , −0.85 , 0.83]	[12.0 , 0.0 , 0.83]

A.4 Case Study 4: Inspect Fish Cage

Start Position [x,y,z]	**Goal** [x,y,z]
[25.0 , 0.0 , 0.0]	[−25.0 , 0.0 , 15.0]

Point Of Interest $\mathbf{x}_k$	**Position** [x,y,z]
$\mathbf{x}_0$	[20.0 , −10.0 , 2.0]
$\mathbf{x}_1$	[10.0 , −20.0 , 4.0]
$\mathbf{x}_2$	[0.0 , −22.0 , 6.0]
$\mathbf{x}_3$	[−10.0 , −20.0 , 8.0]
$\mathbf{x}_4$	[−20.0 , −10.0 , 10.0]

A.5 Case Study 5: Inspect Fish Cage with Obstacles

Start Position [x,y,z]	**Goal** [x,y,z]
[24.0 , 0.0 , 0.5]	[10.0 , −20.0 , 0.5]

Obstacle $\mathbf{o}_j$	**Position** [x,y,z]	**Radius**(m)
$\mathbf{o}_0$	[16.0 , −16.0 , 4.5]	4.0

Point Of Interest $\mathbf{x}_k$	**Position** [x,y,z]
$\mathbf{x}_0$	[21.0 , −10.0 , 8.0]
$\mathbf{x}_1$	[10.0 , −21.0 , 5.0]
$\mathbf{x}_2$	[0.0 , −24.0 , 11.0]

A.6 Case Study 6: Obstacle-rich Environment with Points of Interest

Start Position [x,y,z]	**Goal** [x,y,z]
[0.0 , 1.0 , 0.0]	[19.0 , 2.0 , 0.5]
[19.0 , 2.0 , 0.5]	[20.0 , 20.0 , 0.3]
[20.0 , 20.0 , 0.3]	[0.0 , 17.0 , 0.5]

Obstacle $\mathbf{o}_j$	**Position** [x,y,z]	**Radius**(m)
$\mathbf{o}_0$	[14.0 , 18.0 , 0.0]	2.0
$\mathbf{o}_1$	[12.0 , 18.0 , 0.0]	2.0
$\mathbf{o}_2$	[19.0 , 10.0 , 0.0]	1.5
$\mathbf{o}_3$	[8.0 , $40Sin(0.2t)$+ 40 , 1]	1.0
$\mathbf{o}_4$	[12.0 , $60Sin(0.2t)$+ 40 , 1]	1.2

Point Of Interest $\mathbf{x}_k$	**Position** [x,y,z]
$\mathbf{x}_0$	[22.0 , 5.0 , 0.0]
$\mathbf{x}_1$	[17.0 , 15.0 , 0.5]

References

[1] T.F.Olsen. Ttk4551 - specialization project. Technical report, Trondheim, Dec. 2022.

[2] Forskning og utvikling for realisering av havbruk til havs innspill til strategiske prioriteringer mot 2040. `https://www.sintef.no/globalassets/sintef-ocean/a4_havbruk-til-havs_korrektur3.pdf`. Accessed: 2022-12-13.

[3] Offshore sites or the unavoidable need to renew new forms of farming. `https://weareaquaculture.com/featured/offshore-closed-pen-indepth/26663/`. Accessed: 2022-12-13.

[4] Eleni Kelasidi and Eirik Svendsen. Robotics for sea-based fish farming. In Qin Zhang, editor, *Encyclopedia of Smart Agriculture Technologies*, pages 1–20, Cham, 2022. Springer International Publishing.

[5] Yogesh Girdhar. *Unsupervised Semantic Perception, Summarization, and Autonomous Exploration for Robots in Unstructured Environments*. PhD thesis, 01 2014.

[6] Eric Bourque and Gregory Dudek. On the automated construction of image-based maps. *Autonomous Robots*, 8(2):173–190, Apr 2000.

[7] Ding Fu-guang, Jiao Peng, Bian Xin-qian, and Wang Hong-jian. Auv local path planning based on virtual potential field. In *IEEE International Conference Mechatronics and Automation, 2005*, volume 4, pages 1711–1716 Vol. 4, 2005.

[8] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 3310–3317 vol.4, 1994.

[9] Chang Liu, Seungho Lee, Scott Varnhagen, and H. Eric Tseng. Path planning for autonomous vehicles using model predictive control. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 174–179, 2017.

[10] Björn Lindqvist, Sina Sharif Mansouri, and George Nikolakopoulos. Non-linear mpc based navigation for micro aerial vehicles in constrained environments. In *2020 European Control Conference (ECC)*, pages 837–842, 2020.

[11] Jacob T Schwartz and Micha Sharir. On the "piano movers" problem. ii. general techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4(3):298–351, 1983.

[12] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[13] Marios Xanthidis, Joel M Esposito, Ioannis Rekleitis, and Jason M O'Kane. Motion planning by sampling in subspaces of progressively increasing dimension. In *Journal of intelligent and Robotic systems*, page 777–789, 2020.

[14] Yuncheng Lu, Xue Zhucun, Gui-Song Xia, and Liangpei Zhang. A survey on vision-based uav navigation. *Geo-spatial Information Science*, pages 1–12, 01 2018.

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

[16] Yinjing Guo, Hui Liu, Xiaojing Fan, and Wenhong. Lyu. Research progress of path planning methods for autonomous underwater vehicle. In *Mathematical Problems in Engineering*, 2021.

[17] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pages 1398–1404 vol.2, 1991.

[18] S. Quinlan and O. Khatib. Elastic bands: connecting path planning and control. In *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 802–807 vol.2, 1993.

[19] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[20] Daniel Foead, Alifio Ghifari, Marchel Budi Kusuma, Novita Hanafiah, and Eric Gunawan. A systematic literature review of a* pathfinding. *Procedia Computer Science*, 179:507–514, 2021. 5th International Conference on Computer Science and Computational Intelligence 2020.

[21] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[22] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[23] S. Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for optimal motion planning. 06 2010.

[24] John Schulman, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. 06 2013.

[25] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.

[26] Nathan Ratliff, Matt Zucker, J. Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *2009 IEEE International Conference on Robotics and Automation*, pages 489–494, 2009.

[27] Marios Xanthidis, Nare Karapetyan, Hunter Damron, Sharmin Rahman, James Johnson, Jason O'Kane, and Ioannis Rekleitis. Navigation in the presence of obstacles for an agile autonomous underwater vehicle, 03 2019.

[28] Davide Falanga, Philipp Foehn, Peng Lu, and Davide Scaramuzza. Pampc: Perception-aware model predictive control for quadrotors. 04 2018.

[29] Masahiro Ono, Marco Quadrelli, and Terrance L. Huntsberger. Safe maritime autonomous path planning in a high sea state. In *2014 American Control Conference*, pages 4727–4734, 2014.

[30] D.Q. Mayne, J.B. Rawlings, C.V. Rao, and P.O.M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.

[31] Bjarne Foss and Tor Aksel N. Heirung. *Merging Optimization and Control*. 03 2016.

[32] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, 2e edition, 2006.

[33] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

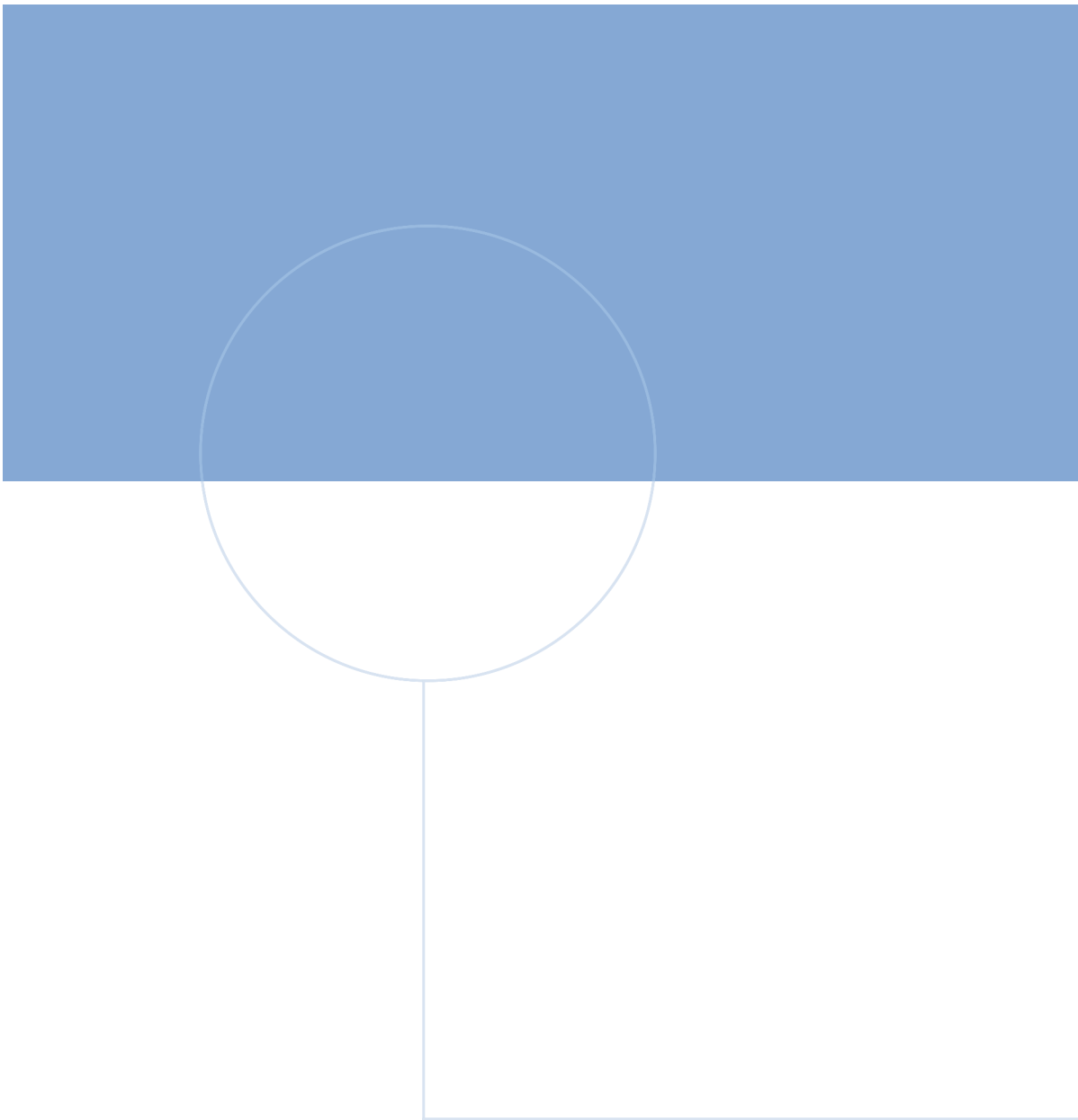
[34] Chonhyon Park, Jia Pan, and Dinesh Manocha. Itomp: Incremental trajectory optimization for real-time replanning in dynamic environments. 06 2012.

[35] Marios Xanthidis, Michail Kalaitzakis, Nare Karapetyan, James Johnson, Nikolaos Vitzilaios, Jason O'Kane, and Ioannis Rekleitis. Aquavis: A perception-aware autonomous navigation framework for underwater vehicles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5387–5394, Prague, Czech Republic, 2021.

[36] Karl-Johan Reite, Martin Føre, Karl Gunnar Aarsæther, Jørgen Jensen, Per Rundtop, Lars T. Kyllingstad, Per Christian Endresen, David Kristiansen, Vegar Johansen, and Arne Fredheim. FhSim - time domain simulations of marine systems. In *Proc. ASME 33rd International Conference on Ocean, Offshore and Arctic Engineering*, 2014.

[37] Biao Su, Karl-Johan Reite, Martin Føre, Karl Gunnar Aarsæther, Morten Alver, Per Christian Endresen, David Kristiansen, Joakim Haugen, Walter Caharija, and Andrei Tsarau. A multipurpose framework for modelling and simulation of marine aquaculture systems. In *Proc. ASME 38th International Conference on Ocean, Offshore and Arctic Engineering*, 2019.

[38] Joel A.E. Andersson and James B. Rawlings. Sensitivity analysis for nonlinear programming in casadi. *IFAC-PapersOnLine*, 51(20):331–336, 2018. 6th IFAC Conference on Nonlinear Model Predictive Control NMPC 2018.



NTNU

Norwegian University of
Science and Technology