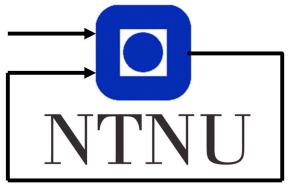


Department of Engineering Cybernetics

TTK4551 - Specialization project (7.5 p)

---

# Collision Free Path Planning for Operations in Dynamically Changing Underwater Environments

---

*Author:*
Torben Falleth Olsen

December, 2022

**Supervisors:**
Ass. prof. Martin Føre, NTNU

**Co-supervisors:**
Herman Biørn Amundsen, NTNU and SINTEF Ocean
Dr. Marios Xanthidis, SINTEF Ocean
Dr. Eleni Kelasidi, SINTEF Ocean

**Abstract**

This study compares the performance of three path planning algorithms in aquaculture: the Elastic Band Method, the Rapidly Random exploring Tree, and the RRT-based near-optimal planner. The Elastic Band Method, provided by SINTEF Ocean, served as the benchmark for comparison. The algorithms were tested in two different environments using a C++ based simulation tool called FhSim. The results showed that the Elastic Band Method performed better than the other two algorithms in both environments.

## Problem formulation

Aquaculture is an important global contributor to the production of seafood for human consumption, and in 2020, Norwegian aquaculture produced almost 1.5 mill. tons of marketable fish meat [1]. 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.

There is also a general desire within aquaculture to shift production methods from manual operations and experience-based reasoning towards a more objective approach using intelligent sensors, mathematical models, and 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 aim of the concept of Precision Fish Farming, which provides approaches for adapting technological solutions to applications in aquaculture.

Using unmanned underwater vehicles (UUVs) is a key element 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 autonomous UUVs 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 influenced by external forces (e.g., waves and currents). Researches in SINTEF Ocean are targeting research to address the challenges (e.g., minimizing the impact on living fish during autonomous fish-farm operations) of using UUVs 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 student projects have explored different planning methods for avoiding both static and dynamic obstacles, but there have been few activities on developing new methods enabling the vehicle to track the paths or trajectories produced by such methods. This specialization will focus on the development of path planning strategies for in-cage navigation of UUVs.

The specialization project will include the following elements:

- Literature review of collision free path planning methods for navigation of UUVs in subsea and aquaculture domains
  - Select three suitable methods and describe these in greater detail
- Implementation and testing:
  - Implement the three methods in FhSim
  - Formulate simulation setups that provide realistic case studies for applications in sea-cages/fish farms
- Simulation experiments:
  - Run the case studies with the methods to demonstrate their different properties
  - Compare the performance between the methods

# Contents

# List of Figures

# Abbreviations

# Part I
# Introduction

Aquaculture is the practice of farming aquatic organisms in complex environments. These environments can vary greatly in terms of their physical characteristics, such as the presence of obstacles, currents, and other factors that can affect the movement of the aquatic organisms [1, 3]. Automation plays a crucial role in maintaining efficient and effective aquaculture operations, particularly in challenging environments. It can assist with various tasks, including navigating and managing complex environments. To achieve success in aquaculture operations, it is important to use a robust path planner for efficient navigation through the aquatic environment. Path planning algorithms allow robots to navigate through the aquatic environment, avoiding obstacles and finding the most efficient routes to their destination. This is important for a number of reasons, including optimizing resource utilization, improving the health and welfare of the aquatic organisms, and increasing the overall productivity of the aquaculture operation. [1, 3]

SINTEF Ocean has implemented the Elastic band Method towards enabling autonomy prior to this project. SINTEF Ocean's implementation of the elastic band algorithm had been extensively tested and validated [2]. In this project, the task is to compare SINTEF Ocean's implementation of the Elastic Band Method against other path planning algorithms to test the performance. The algorithms that are being compared against the Elastic Band Method are the Random Rapidly exploring Trees(RRT) and the RRT-based near-optimal planner.

The contributions of this work are towards path planning in complex an aquatic environments. More specifically:

- Implementation of RRT and RRT*
- Testing the Path Planners in different simulation environments.

The structure of the report is divided into the following Sections: Path planners, Implementation of methods, Simulations, Discussion, Conclusion and Recommended future work.

# Part II
# Path Planners

Path planning [4] is the computational problem of finding a path to move an object from a starting pose to a goal pose within the free space. This is an essential aspect of navigation for autonomous systems, as it allows them to solve safe navigation by deciding safe motion. Path planners can be divided into two categories: *Global path planners* and *Local path planners*. Global path planning involves finding a path from a starting point to a destination within a global map, a global map means a map that covers the total space. Local path planning involves finding a path within a local map using sensor data to update the path as the object moves, local map means a map that covers just a portion of the total space.

The Guidance and Navigation Control (GNC) in Figure 1 is responsible for ensuring that the Remotely Operated Vehicle (ROV) remains on course and reaches its destination safely.

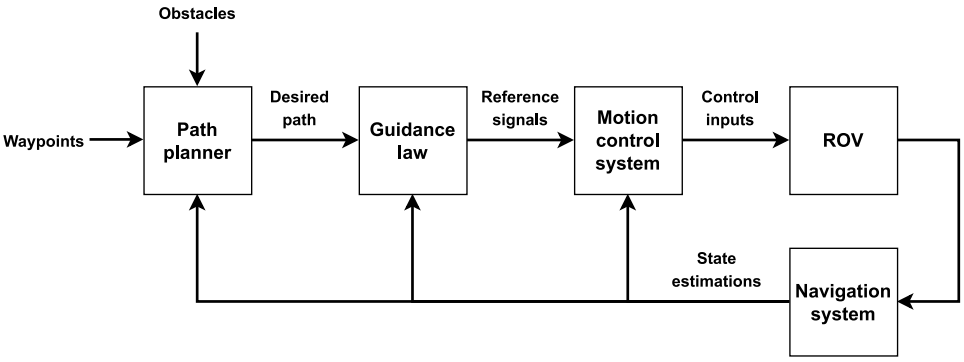


Figure 1: A block diagram illustrating the information flow between the main modules of the system, based on a figure in [2]

As depicted in the figure, the path planner receives input in the form of goals(waypoints), obstacles, and estimated states. The planner uses this information to generate a safe path, which is then passed to the guidance law. The guidance law uses the desired path to generate reference signals, which are sent to the motion control system. The motion control system uses these signals to generate control inputs for the ROV, which are used to steer the ROV along the desired path. This process allows the ROV to navigate its environment and achieve its goals while avoiding obstacles and staying within the configuration space.

# 1 Background

There are many different approaches to path planning, each with their own strengths and limitations. To present the possible paths that a path planner can take from its current location to a goal while avoiding obstacles in its work space is called the configuration space $C$ [5]. For path planners however the subset of the configuration space is typically used to search for paths, namely the free space $C_{free}$ [5]. The free space can then be defined as $C_{free} = [c^i_{min}, c^i_{max}] \times \cdots \times [c^n_{min}, c^n_{max}]$ with $n$ degrees of freedom, where $[c^n_{min}, c^n_{max}]$ is defining the maximum and minimum of the space [6]. The free space mainly represents the configurations that are not in collisions with any obstacles.

Optimality is a measure of how well a path planner performs in finding the best solution to a given problem. There are several different types of optimality, including near-optimal, sub-optimal, global optimality and local optimality. Near-optimal refers to a solution that is close to the optimal solution, within a specified tolerance. Sub-optimal refers to a solution that is arbitrary far from optimal solution. A global optimal solution refers to the best possible solution to a problem, regardless of where it is found. A locally optimal solution refers to the best solution within a limited range of candidate solutions. In the following paragraphs some common algorithms for path planning are to be presented.

One common approach is to use a search-based algorithm, such as A* or Dijkstra's algorithm [7], to find the shortest path between two points. These algorithms are widely used because they are relatively efficient and can find paths that are guaranteed to be optimal. However, they can be computationally expensive and may not be suitable for real-time applications.

Another approach is to use probabilistic methods, such as Rapidly exploring Random Trees (RRTs) or Probabilistic Roadmaps (PRMs), to generate paths in a probabilistic manner [7]. The RRT and PRM are known to be probabilistically complete, which means that if a problem is solvable, the probability that the planner will solve it approaches 1 as the running time increases. This means that given enough time, the planner will eventually find a solution to any solvable problem [8]. By improving these methods into RRT-based near-optimal planner (RRT*) and PRM-based near-optimal planner (PRM*) can prove to be more efficient than search-based algorithms and can generate paths that are near-optimal. However, they are not guaranteed to find the optimal path and can be sensitive to the parameters used.

In conclusion, path planning is a crucial problem in robotics and there are many different approaches to solving it. In this project, the focus is on

comparing two path planning algorithms against the already implemented Elastic band Method and observe the outcome. This allows the possibility to gain a deeper understanding of how different path planners work and how they perform compared to each other. By conducting this comparison, one can identify the strengths and weaknesses of each approach and gain insights into the factors that affect their performance.

In this project, the algorithms selected are Elastic Band Method (EBM), RRTs and RRT*. These algorithms each have a different approach to path planning, and each has its own strengths and limitations. The world in this project is defined using the NED coordinate system, and the vehicle has 6 Degrees Of Freedom (DOF), which are represented by the state $\eta = [x, y, z, \phi, \theta, \psi]^\top$. The position of the vehicle is represented by $[x, y, z]^\top$ , and its orientation is represented by $[\phi, \theta, \psi]^\top$.

## 2 Elastic Band Method (EBM)

EBM [9] is a technique that conceptualizes a path as a band of partly overlapping bubbles of varying sizes with a functionality as shown in Figure 2. The motivation behind the method is to treat the path as if it were a physical and dynamic system, giving it properties similar to those of a real physical system such as an elastic band. The elastic band is a good model for this because it contracts due to internal potential energy, causing it to become shorter. This idea can be applied to the path in order to give it similar dynamics.

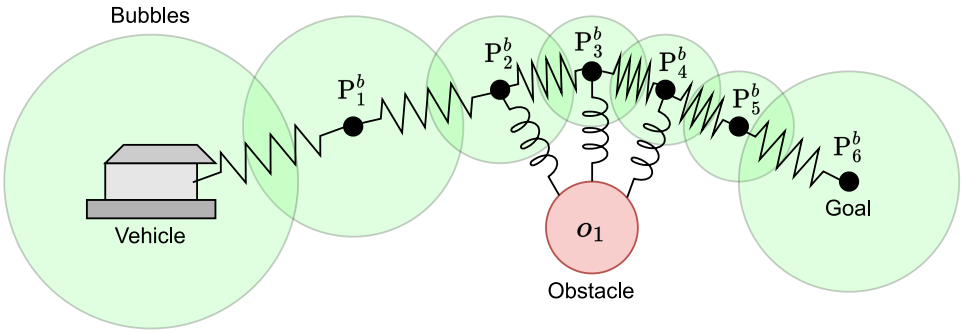


Figure 2: EBM - The figure shows a band of overlapping bubbles(Green) from a starting pose to a goal pose(Black) that avoids an obstacle(Red)

Figure 2 shows a vehicle that has created a path consisting of overlapping bubbles, where each bubble has an attraction force on its neighboring bub-

bles in the form of spring forces. Initially, the overlapping bubbles are formed in a straight line from a start point to a set of waypoints. Due to the contraction forces, the path will also change depending on its own state. To avoid collision within the environment the EBM provides a repellent force from the obstacles onto the bubbles, as shown in Figure 2. The figure illustrates a repellent spring that pushes the bubbles away from the obstacle, making the path avoid the obstacle.

The EBM consists of three phases: *Initial path build up*, *Elastic band deformation*, and *Bubble reorganization* [10]. In the following Sections the detailed procedures and methods mentioned previously are going be explained.

## 2.1 Phase-I: Initial Path Build-up

In Phase-I the aim is to initialize a set of straight line paths from the vehicle to a number of given navigation waypoints without considering any obstacles. The path consists of a sequence of partly overlapping bubbles, $\mathbf{b_i}$, each bubble has a position and radius, i.e., $\mathbf{b}_i = [\mathbf{p}_i^{b\top}\ r_i^b]^\top,\ i \in [0,\ n-1]$, where $\mathbf{p}_i^b \in \mathbb{R}^3$ contains the bubble position of bubble $i$, and $r_i^b > 0$ is the $i$-th bubble radius. The bubble nodes are inserted along the initial path and labeled with ascending numbers. Each bubble should be a subset of the free space, and its radius is therefore

$$0 \ \leq \ r_{min} \ \leq \ r_i^b, \tag{1}$$

where $r_{min}$ refers to the minimum bubble radius, which corresponds to the maximum distance from the vehicle center to its edge. [10, 11, 2]

## 2.2 Phase-II: Elastic Band Deformation

Phase-II in the Elastic Band Method is reserved to continuously modify the path in real-time to avoid static and dynamic obstacles. This is done by introducing artificial forces to contract the path from internal potential and to repel the path from obstacles. Each bubble node is subject to contracting forces from its neighboring bubbles. This will work to remove any "slack" from the path, thus making the path shorter. The internal force $\mathbf{f}_{int}^i$ on bubble $i$ can be computed as a geometric expression using

$$\begin{aligned}\mathbf{f}_{int}^i(\mathbf{p}_{i-1}^b, \mathbf{p}_i^b, \mathbf{p}_{i+1}^b) = k_{int} \Bigg( & \frac{\mathbf{p}_{i+1}^b - \mathbf{p}_i^b}{\left\|\mathbf{p}_{i+1}^b - \mathbf{p}_i^b\right\|} \left(\left\|\mathbf{p}_{i+1}^b - \mathbf{p}_i^b\right\| - r_{\min}\right) + \\ & \frac{\mathbf{p}_{i-1}^b - \mathbf{p}_i^b}{\left\|\mathbf{p}_{i-1}^b - \mathbf{p}_i^b\right\|} \left(\left\|\mathbf{p}_{i-1}^b - \mathbf{p}_i^b\right\| - r_{\min}\right) \Bigg),\end{aligned} \tag{2}$$

where $r_{min}$ mimics the natural spring length and $k_{int} > 0$ is the spring stiffness.

Given that a situation can contain multiple obstacles where some might be dynamic, obtaining the virtual external repulsion force requires that all obstacles are considered. This is important because the repulsion force represents the force to make the bubbles avoid obstacles, where the $j$-th obstacle is defined as $\mathbf{o}_j = [\mathbf{p}_j^{o\top} \;\; r_j^o]^\top,\ j \in [0,\ n-1]$ with the position $\mathbf{p}_j^o \in \mathbb{R}^3$ and the radius $r_j^o > 0$. The repulsion force for from obstacle $\mathbf{o}_j$ onto bubble $\mathbf{p}_i^b$ is then calculated as follows:

$$\mathbf{f}_{ext}^{i,j}\left(\mathbf{b}_i,\ \mathbf{o}_j\right) = k_{ext} \cdot e^{-D_a} \left( \frac{\mathbf{p}_i^b - \mathbf{p}_j^o}{\left\| \mathbf{p}_i^b - \mathbf{p}_j^o \right\|} \right), \tag{3}$$

where $k_{ext} > 0$ is the repulsive gain. The fading function $e^{-D_a}$ is to ensure that the repulsive effect of an obstacle decreases with distance $D_a$, which is calculated using the following equation:

$$D_a = ||\mathbf{p}_i^b - \mathbf{p}_i^o|| - r_{min} - r_j^o - d_{sm}, \tag{4}$$

where $d_{sm} \geq 0$ is a customizable safety margin. Next, the summarized external force acting upon bubble $i$ is calculated by

$$\mathbf{F}_{ext}^i = \sum_{j=0}^{N-1} \mathbf{f}_{ext}^{i,j}\left(\mathbf{b}_i,\ \mathbf{o}_j\right). \tag{5}$$

Since it is possible that the elastic band deforms above the sea surface or below the seafloor, virtual forces from the sea surface and the seafloor were included. This is to prevent the EBM from creating a path that travels above the sea surface or below the seafloor. The force will push the bubbles away from the sea surface and the seafloor. The sea surface force upon bubble $i$ is given by

$$\mathbf{f}^i_{\text{surface}}(\mathbf{b}_i) = k_{\text{surface}} e^{-z} \frac{[0,\, 0\,,\, z]^\top}{z} \tag{6}$$

where $k_{\text{surface}} > 0$ is a designer parameter. Similarly, the resulting force from the seafloor upon bubble $i$ is then

$$\mathbf{f}^i_{\text{seafloor}}(\mathbf{b}_i) = k_{\text{seafloor}} e^{-(z - z_{max} - r_{min} - d_{sm})} \frac{[0,\, 0\,,\, z - z_{max}]^\top}{z_{max} - z} \tag{7}$$

with $z_{max}$ indicating the the depth of the seafloor and $k_{\text{seafloor}} > 0$ is a design parameter. The total applied force upon bubble $i$ is then summarized by

$$\mathbf{f}^i_{\text{total}} = \mathbf{f}^i_{\text{int}} + \mathbf{f}^i_{\text{seafloor}} + \mathbf{f}^i_{\text{surface}} + \mathbf{F}^i_{ext} \tag{8}$$

The resulting force $\mathbf{f}^i_{\text{total}}$ is then used to find the updated bubble position for bubble $i$. A gradient descent [12] can then be used to find the new bubble position. The new bubble position is found by iteratively solving

$$\mathbf{p}^b_i[k+1] = \mathbf{p}^b_i[k] + \gamma \mathbf{f}^i_{\text{total}} \tag{9}$$

until $\left|\mathbf{p}^b_i[k+1] - \mathbf{p}^b_i[k]\right| < d_{\text{tol}}$ for some small constant $d_{\text{tol}} > 0$, where $\gamma > 0$ is an input that represents the step size for the gradient descent. [10, 11, 2]

### 2.3 Phase-III: Bubble Reorganization

In Phase-III of the EBM, the new configuration of the EBM is evaluated and updated. This is done by recalculating the bubble radii, removing redundant bubbles, and inserting new bubbles if needed [2]. The recalculation of the bubble radii is to ensure that each bubble is still a subset of the free space, where the radius for bubble $i$ is determined by

$$r^b_i = \begin{cases} r_{max}, & ||\mathbf{p}^b_i - \mathbf{p}^o_j|| > r_{max} - r^o_j \\ r_{min}, & ||\mathbf{p}^b_i - \mathbf{p}^o_j|| < r_{min} - r^o_j + d_{sm} \\ ||\mathbf{p}^b_i - \mathbf{p}^o_j|| - r^o_j - d_{sm}, & \text{otherwise,} \end{cases} \tag{10}$$

where $\mathbf{o}_j = [\mathbf{p}_j^{o\top} \;\; r_j^o]^\top$ is the closest obstacle, $r_{max}$ is the maximum bubble radius and $r_{min}$ is the minimum bubble radius.

When changing the bubbles and path of the elastic band, one must always check two properties. This is to ensure that the path maintains feasibility and continuity. The first step is to remove redundant bubbles, which means when a bubble is completely within another bubble's coverage, i.e.,

$$|r_{i-1}^b - r_i^b| \geq ||\mathbf{p}_{i-1}^b - \mathbf{p}_i^b|| \tag{11}$$

or if the bubble is within an area where two other bubbles overlap, i.e.

$$r_{i-1}^b + r_{i+1}^b > ||\mathbf{p}_i^b - \mathbf{p}_{i-1}^b|| + ||\mathbf{p}_{i+1}^b - \mathbf{p}_i^b|| + d_{ol} \tag{12}$$

where $d_{ol} > 0$ is a parameter representing the desired overlap between two neighboring bubbles, such that the radius of the intersection circle is always larger or equal to $r_{\min}$. The second step is to ensure that the path is continuously connected, which means that there are no gaps between consecutive bubbles in the path. This is done by checking the criteria below,

$$r_i^b + r_{i-1}^b - d_{ol} < ||\mathbf{p}_i^b - \mathbf{p}_{i-1}^b|| \tag{13}$$

Should the criteria for (13) hold, then the method will insert an extra bubble between $\mathbf{p}_{i-1}^b$ and $\mathbf{p}_i^b$ to fill the gap. [10, 11, 2]

# 3 Rapidly Exploring Random Trees (RRTs)

RRTs [13] is an algorithm that explores the free space $C_{free}$ by creating a random tree that stretches across the region.

In Figure 3, the steps necessary to create the random tree are illustrated. The first step is sampling a random point (red) within the configuration space $C$. When the random point is picked, the nearest neighbor (green) is found. When knowing both the random point and the nearest node the RRT creates a new node (yellow), which is restricted by an input step size. Next, the new node is connected to the nearest neighboring node and becomes a node of the tree. This is done iteratively until a solution is found or a predefined timeout.

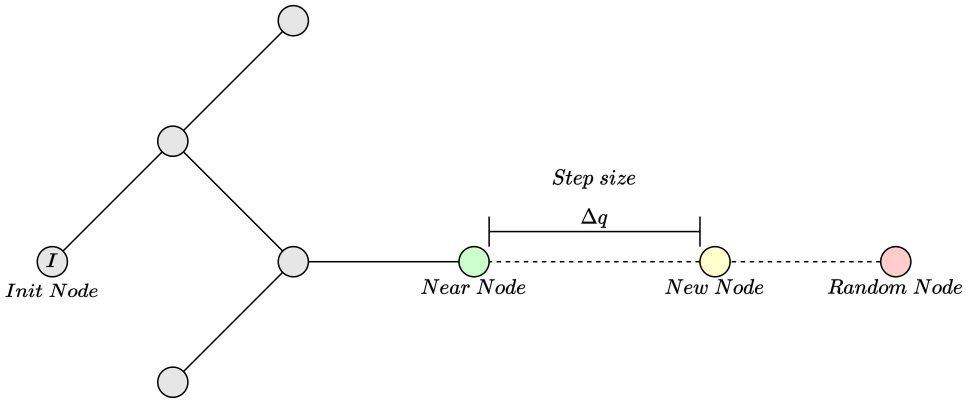


Figure 3: RRT - The figure shows a random point(red) being picked and the nearest node(green) being found, with a new node(yellow) being placed in the direction of the random point at a length equal to the step size

The RRTs holds four main steps that are repeated, these are *random sampling*, *nearest neighbor*, *tree expansion* and *path extraction*. In the following sections the detailed procedures and methods mentioned previously are going be explained. The implementation details of the method are given in the Section 6:

## 3.1 Random sampling

When sampling for the random point $q_{rand}$ the point needs to lie in the free space $\in \ C_{free}$ and it is sampled uniformly to preserve thew probabilistic completeness guarantees. The random point can then be found by the following equation:

$$
\begin{aligned}
q^i_{rand} &= \left(c^i_{max} - c^i_{min}\right) \frac{r_s}{r_s^{max}} \\
\mathbf{q}_{rand} &= [q^1_{rand}, q^2_{rand}, q^3_{rand}]
\end{aligned}
\tag{14}
$$

where $r_s$ is a random sample from a random sample function, $r_s^{max}$ is the maximum value that the random sample can have.

## 3.2 Nearest neighbor

The next step is to find the nearest neighbor $q_{near}$ to the random point. The nearest node can be found by calculating the minimum distance between the random point and all the nodes in the tree. The nearest neighbor can then be found by,

$$
\mathbf{q}_{near} = argmin \left\{ ||\mathbf{q} - \mathbf{q}_{rand}|| \right\} \quad \mathbf{q} \in \mathbf{T}, \tag{15}
$$

with $\mathbf{q}$ being a node already connected to the tree $\mathbf{T}$.

## 3.3 Tree expansion

When the nearest neighbor is found the next step is to calculate the new node $\mathbf{q}_{new}$. By using the position of the nearest node one can calculate the new node of the tree by using the direction of the random point and limit the branch with a input step size $\Delta q$. The equation can then be stated as:

$$
\mathbf{q}_{new} = \mathbf{q}_{near} + \frac{\Delta q}{d_{min}} (\mathbf{q}_{rand} - \mathbf{q}_{near}), \quad \Delta q \leq d_{min} \tag{16}
$$

If the distance to between the nearest node and the random point is less than the step size, the equation is as follows

$$
\mathbf{q}_{new} = \mathbf{q}_{near} + (\mathbf{q}_{rand} - \mathbf{q}_{near}), \quad \Delta q > d_{min} \tag{17}
$$

When the new node is computed the next step is to check if the new node and is collision free.

### 3.3.1 Collision avoidance

When checking for collisions in the RRTs one might use a method that calculates the minimum distance from a point a line segment [1]. The method consist of three cases, Should any of these cases have a lower distance than

[1] https://www.geeksforgeeks.org/minimum-distance-from-a-point-to-the-line-segment-using-vectors/

the obstacle radius, then the line segment collides with an obstacle and the node will not be connected to the tree.

### Case 1

In Case 1 the obstacle $O_1 = \mathbf{p}_1^o$ has a minimum distance $d_1$ to $P_2 = \mathbf{q}_{new}$ on the line segment $l_{P1}^{P2} = [(x_{P_2} - x_{P_1}),\ (y_{P_2} - y_{P_1}),\ (z_{P_2} - z_{P_1})]$ and $P_1 = \mathbf{q}_{near}$. If the following statement is true

$$l_{P1}^{P2} \cdot l_{P2}^{O1} > 0,$$

where $l_{P2}^{O1} = [(x_{O_1} - x_{P_2}),\ (y_{O_1} - y_{P_2}),\ (z_{O_1} - z_{P_2})]$, then the minimal distance is between $P_2$ and the obstacle, as shown in Figure 4.

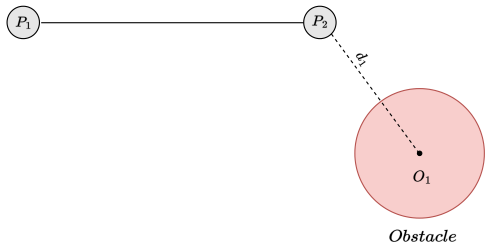


Figure 4: The figure shows the case 1 collision check, with the position of the obstacle (Red) compared to the line segment between $P_1$ and $P_2$.

The distance can then be calculated by the norm $d_1 = ||l_{P2}^{O1}||$, which results in the distance between the obstacle and $P_2$.

### Case 2

In Case 2 the obstacle $O_1$ has a minimum distance to $P_1$ on the line segment $l_{P1}^{P2}$. If the following statement is true

$$l_{P1}^{P2} \cdot l_{P1}^{O1} < 0$$

where $l_{P1}^{O1} = [(x_{O_1} - x_{P_1}),\ (y_{O_1} - y_{P_1}),\ (z_{O_1} - z_{P_1})]$, then the minimal distance is between $P_1$ and the obstacle, as shown in Figure 5.

The distance can then be calculated by the norm $d_2 = ||l_{P1}^{O1}||$, which results in the distance between the obstacle and $P_1$.

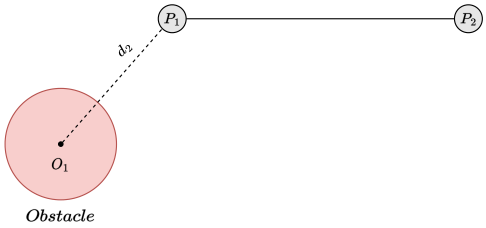


Figure 5: The figure shows the case 2 collision check, with the position of the obstacle (Red) compared to the line segment between $P_1$ and $P_2$.

### Case 3

In Case 3 the obstacle $O_1$ has a minimum distance perpendicular to the line segment $P_1P_2$, as shown in Figure 6.

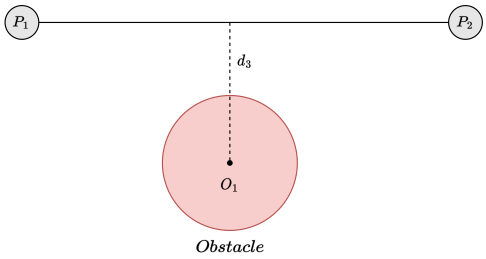


Figure 6: The figure shows the case 3 collision check, with the position of the obstacle (Red) compared to the line segment between $P_1$ and $P_2$.

The distance can then be calculated as:

$$d_3 = \frac{||l_{P1}^{P2} \times l_{P1}^{O1}||}{||l_{P1}^{P2}||}$$

which results in the perpendicular distance between the obstacle and the line segment $l_{P1}^{P2}$.

## 3.4 Path extraction

As the RRT iteratively finds new nodes for the tree, there is an goal bias when sampling for random points. The goal bias is the probability of sampling the goal node instead of a random point, thus creating a solution. If a solution exist the next step is to extract the path from the tree. This is done by following the parent nodes from the goal position back to the initial node as shown in Figure 7.

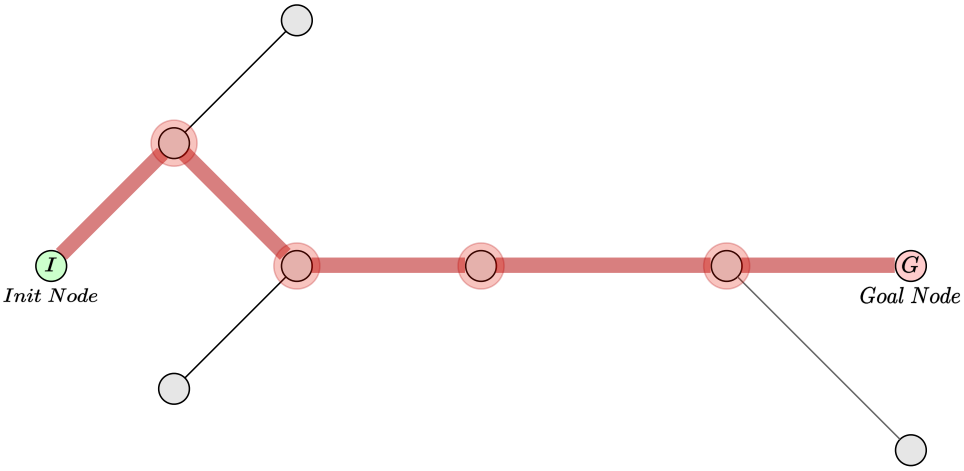


Figure 7: RRT - The figure shows the path(red) being extracted between the init node(green) and the goal(red).

# 4 RRT-based near-optimal planner (RRT*)

RRT* is an extension of the RRT algorithm presented in Section 3, that provides near-optimal solutions. One of the key features of the RRT* algorithm is its use of cost to guide the optimization process. Cost is a way of evaluating the quality or value of a path, which in this case would be the path length. This is important since knowing the cost of a path makes it is possible to improve the cost. The use of cost in the RRT* algorithm allows the algorithm to continuously improve the quality of the path by rewiring the tree as it iteratively adds new nodes. By prioritizing paths with lower cost, the algorithm can quickly identify and discard infeasible or sub-optimal paths, thereby reducing the search space and improving the efficiency of the algorithm.

## 4.1 Major Enhancements of RRT

The RRT* extension adds three major enhancements to the regular RRT: *incorporating cost*, *rewire node* and *rewire neighborhood*. Figure 8 shows an illustration of these enhancements.

Figure 8 shows a new node that has been connected to a preexisting tree. A low-opacity circle indicates the neighborhood region, and the neighboring nodes (in gray) inside this circle are checked against the current edge (shown as a firm line) of the new node (in yellow) using cost. The new node checks the cost of each neighboring node, plus the cost (shown as a dotted line) to reach the neighboring node. If it finds a neighboring node that would give the new node a reduced cost, it will disconnect from its current parent and

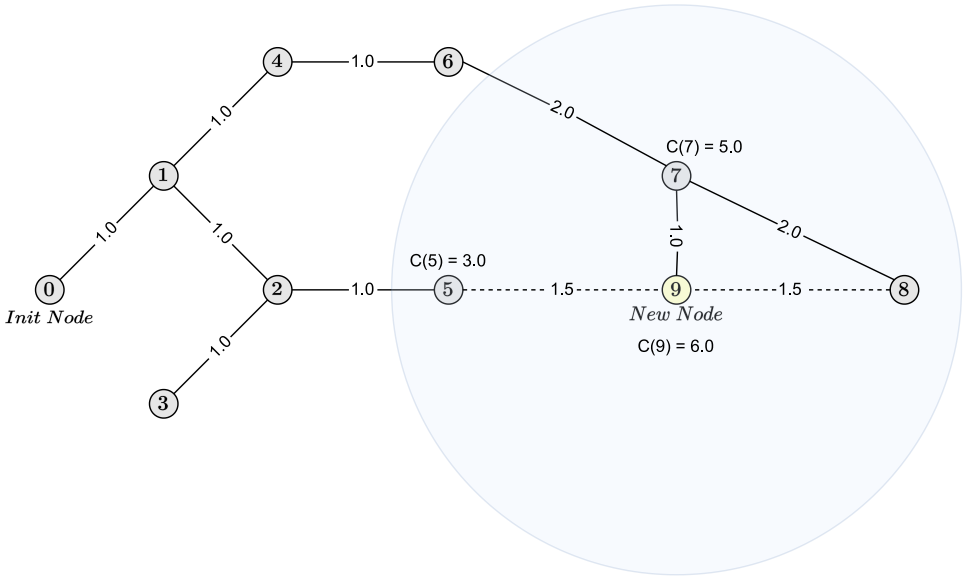


Figure 8: RRT* - The figure shows the rewiring of the new node (yellow) in a neighborhood(low opacity blue circle) in a tree(grey nodes).

connect to the neighboring node instead.

It is not only the parent of the new node that is updated in the tree. The neighboring nodes within the region are also checked against the new node to try to improve their own cost. The neighboring nodes check the cost of the new node, plus the cost to reach the new node, to see if it would reduce their cost. If the cost is reduced by connecting to the new node, the neighboring node will disconnect from its current parent and attach itself to the new node instead.

In the following sections, the detailed procedures and methods mentioned previously will be explained in more detail. The implementation details of the method are given in the Section 6:

### 4.1.1 Incorporating Cost

The most important enhancement of the RRT in RRT* is the incorporation of cost, this is important because it is the cost of the solution of the tree that the RRT* wants to minimize. The cost of a node in the RRT* is the sum of the weight of the edge that leads to that node added with the cost of the parent node. The equation can then be written as:

$$\text{cost}_{new} = \text{cost}_{near} + ||\mathbf{q}_{new} - \mathbf{q}_{near}|| \tag{18}$$

where $\text{cost}_{near}$ is the cost of the nearest node and $\text{cost}_{new}$ is the cost of the new node.

### 4.1.2 Rewire node

The next enhancement of the RRT is the rewiring of the new node, the reason that this is desirable is because it will reduce the cost of the new node. This is done by checking if the new node can be connected to another parent resulting in a reduced cost for the new node. Should this be the case then the new cost of the new node is be calculated by:

$$\text{cost}_{new} = \text{cost}_{neighbor} + ||\mathbf{q}_{new} - \mathbf{q}_{neighbor}|| \tag{19}$$

where $\text{cost}_{neighbor}$ is the cost of the better neighboring parent.

### 4.1.3 Rewire neighborhood

The last enhancement of the RRT is the rewiring of the neighborhood, the reason the reason for doing this is that when a new node is set the cost of one of the neighboring nodes might be improved. The RRT* then checks the resulting cost from the neighboring node to the new node. If the cost of the neighboring node is reduced, then the new cost of the neighboring node is calculated by:

$$\text{cost}_{neighbor} = \text{cost}_{new} + ||\mathbf{q}_{new} - \mathbf{q}_{neighbor}|| \tag{20}$$

# Part III
# Implementation of methods

## 5 Simulation Environment

Realistic simulation is an important part of underwater robotics research. It allows researchers to test algorithms and concepts in a controlled setting.

SINTEF Ocean provided a working simulation environment with a vehicle, control system, and net structures. The goal of the project is to integrate a path planning module with the existing simulation objects. The project focus on developing a path planning algorithm that could be integrated with the existing control system to improve the performance of the vehicle in the simulation environment.

### 5.1 FhSim - Marine Simulations

FhSim is a software platform implemented in C++ that is designed for mathematical modeling and numerical simulations, with a focus on simulation performance and marine systems modeling. Its modular design allows users to define simulations by interconnecting a variety of independent objects. FhSim also includes a collection of mathematical marine models that are essential for simulating a marine environment, with special focus on fish farming [14, 15].

## 6 Algorithms

The goal of this project is to implement three different path planning algorithms, as presented in section II. In this section, the implementation and description of these algorithms is presented.

## 6.1 EBM

In this project, the EBM algorithm was already implemented and is referred to as Algorithm 1.The inputs used in this algorithm are described in Line 1. In Line 2, the path is initialized. Line 4 contains a loop that runs from Lines 7 to 19, as long as the iteration index $i$ is not equal to the number of bubbles $N-2$. Line 5 set the new bubble position $\mathbf{p}_{new}$ is set equal to the current bubble position $\mathbf{p}_i^b$ and the variable $k$ is set to zero. In Line 6, there is a loop that continues as long as the distance between the old position $\mathbf{p}_{old}$ and the new bubble position is greater than a specified tolerance $d_{tol}$ or $k = 0$, run the loop. In Line 7, the old bubble position is set equal to the new bubble position. Line 8 calculates the elastic band deformation using the internal force $\mathbf{f}_{int}$ from equation 2.

In Line 9 the repulsion force $\mathbf{f}_{ext}$ is set to zero. In Line 10, a for-loop is used to loop through the obstacles. Line 11 calculates the distance from the bubble to the obstacle using equation 4. In Line 12, the repulsion force from the obstacle onto the bubble is calculated using equation 5 and added to the previous repulsion force. Line 13 then calculates the sea surface force $\mathbf{f}_{surface}$ using equation 6, and Line 14 calculates the seafloor force using equation 7. In Line 15, all the forces acting on the bubble are summarized using equation 8.

The new bubble position is calculated in Line 16 using equation 9, and the current bubble position is set to the new position in Line 17. Line 18 then recalculates the bubble radius to ensure that it remains a subset of the free space. Line 21 contains a while loop that loops through all the bubbles placed in Lines 6 to 19. Lines 22 to 23 remove any bubbles that are completely within another bubble, and Lines 24 to 25 remove any bubbles that are within an area where two bubbles overlap. Finally, Lines 26 to 28 check for gaps between consecutive bubbles in the bubble path. If there is a gap, an extra bubble is inserted between the two bubbles in Line 27.

**Algorithm 1** Elastic Band Method [2]

1: **Input**: $k_{\text{int}}\rightarrow$ Contraction gain. $k_{\text{ext}} \rightarrow$ Repulsion gain. $k_{\text{surface}} \rightarrow$ Surface gain. $k_{\text{seafloor}} \rightarrow$ Sea floor gain. $r_{\min} \rightarrow$ Min bubble radius. $r_{\max}\rightarrow$ Max bubble radius. $d_{\text{ol}}\rightarrow$ Bubble overlap. $d_{\text{sm}}\rightarrow$ Safety margin. $d_{\text{tol}}\rightarrow$ Gradient descent tolerence. $\gamma\rightarrow$ Gradient descent step size. $J \rightarrow$ Number of obstacles.

2: Construct the initial path $\mathbf{\Gamma}_{init}$ consisting of $N$ bubbles allocated evenly between the vehicle and the waypoints.

3: i $\leftarrow 0$

4: **while** $i \neq N-2$ **do**:

5: $\quad \boldsymbol{p}_{\text{new}} \leftarrow \boldsymbol{p}_i^b, k \leftarrow 0$

6: $\quad$ **while** $||\boldsymbol{p}_{\text{new}} - \boldsymbol{p}_{\text{old}}|| > d_{\text{tol}}$ or $k = 0$ **do**

7: $\quad\quad \boldsymbol{p}_{\text{old}} \leftarrow \boldsymbol{p}_{\text{new}}$

8: $\quad\quad \boldsymbol{f}_{\text{int}} \leftarrow k_{\text{int}}(\frac{\boldsymbol{p}_{i+1}^b - \boldsymbol{p}_{\text{old}}}{\|\boldsymbol{p}_{i+1}^b - \boldsymbol{p}_{\text{old}}\|}(\|\boldsymbol{p}_{i+1}^b - \boldsymbol{p}_{\text{old}}\| - r_{\min}) + \frac{\boldsymbol{p}_{i-1}^b - \boldsymbol{p}_{\text{old}}}{\|\boldsymbol{p}_{i-1}^b - \boldsymbol{p}_{\text{old}}\|}(\|\boldsymbol{p}_{i-1}^b - \boldsymbol{p}_{\text{old}}\| - r_{\min}))$

9: $\quad\quad \boldsymbol{f}_{\text{ext}} \leftarrow 0$

10: $\quad\quad$ **for** $j \leftarrow 0; j < J; j{+}{+}$ **do**

11: $\quad\quad\quad D_a = \left\|\boldsymbol{p}_{\text{old}} - \boldsymbol{p}_j^o\right\| - r_{\min} - r_j^o - d_{\text{sm}}$

12: $\quad\quad\quad \boldsymbol{f}_{\text{ext}} \leftarrow \boldsymbol{f}_{\text{ext}} + k_{\text{ext}} e^{-D_a}\left(\frac{\boldsymbol{p}_{\text{old}} - \boldsymbol{p}_j^o}{\|\boldsymbol{p}_{\text{old}} - \boldsymbol{p}_j^o\|}\right)$

13: $\quad\quad \boldsymbol{f}_{\text{surface}} \leftarrow k_{\text{surface}} e^{-z}\frac{[0,0,z]^\top}{z}$

14: $\quad\quad \boldsymbol{f}_{\text{seafloor}} \leftarrow k_{\text{seafloor}} e^{-(z - z_{\max} - r_{\min} - d_{\text{sm}})}\frac{[0,0,z-z_{\max}]^\top}{z_{\max} - z}$

15: $\quad\quad \boldsymbol{f}_{\text{total}} \leftarrow \boldsymbol{f}_{\text{int}} + \boldsymbol{f}_{\text{ext}} + \boldsymbol{f}_{\text{surface}} + \boldsymbol{f}_{\text{seafloor}}$

16: $\quad\quad \boldsymbol{p}_{\text{new}} \leftarrow \boldsymbol{p}_{\text{old}} + \gamma \boldsymbol{f}_{\text{total}}, k \leftarrow k + 1$

17: $\quad \boldsymbol{p}_i^b \leftarrow \boldsymbol{p}_{\text{new}}$

18: $\quad \boldsymbol{r}_i^b \leftarrow \min\{\max\{\min\{\|\boldsymbol{p}_i^b - \boldsymbol{p}_0^o\| - r_0^o - d_{\text{sm}}, \cdots, \|\boldsymbol{p}_i^b - \boldsymbol{p}_J^o\| - r_J^o - d_{\text{sm}}, \|z_i - z_{\max}\| - d_{\text{sm}}\}, r_{\min}\}, r_{\max}\}$

19: $\quad$ i $\leftarrow$ i+1

20: i $\leftarrow 0$

21: **while** $i < N$ -2 **do**:

22: $\quad$ **if** $|r_{i-1}^b - r_i^b| \geq \|\mathbf{p}_{i-1}^b - \mathbf{p}_i^b\|$ **then**

23: $\quad\quad$ Delete $\boldsymbol{p}_i^b$, $N \leftarrow N - 1$

24: $\quad$ **else if** $r_{i-1}^b + r_{i+1}^b > \|\mathbf{p}_i^b - \mathbf{p}_{i-1}^b\| + \|\mathbf{p}_{i+1}^b - \mathbf{p}_i^b\| + d_{ol}$ **then**

25: $\quad\quad$ Delete $\boldsymbol{p}_i^b$, $N \leftarrow N - 1$

26: $\quad$ **else if** $r_i^b + r_{i-1}^b - d_{ol} < \|\mathbf{p}_i^b - \mathbf{p}_{i-1}^b\|$ **then**

27: $\quad\quad$ Insert a new bubble at the midpoint between $\boldsymbol{p}_{i-1}^b$ and $\boldsymbol{p}_i^b$, $N \leftarrow N + 1$

28: $\quad\quad$ Go back to Line 3.

29: $\quad$ i $\leftarrow$ i+1

## 6.2 RRT

The RRTs algorithm is implemented and structured as Algorithm 2.The inputs used in this algorithm are described in Line 1. Lines 2 to 3 initialize the tree and the path. Line 4 contains a while-loop that continues as long as the new node $\mathbf{q}_{new}$ is not equal to the goal node $\mathbf{q}_{goal}$, and the code does not time out. Lines 5 to 7 contain functions that execute the equivalent of equations 14 to 17 to find the random point $\mathbf{q}_{rand}$, the nearest neighbor $\mathbf{q}_{near}$ and the new node $\mathbf{q}_{new}$.

After the new node is calculated in Line 7, the algorithm performs a collision check on the new node in Line 8. The collision check is described in Section 3.3.1 If the line segment between the new node and the neighboring node is collision-free, the algorithm progresses to Lines 9 and 10. In Line 9, the new node is added to a list containing all the nodes of the tree, and in Line 10, the parent is connected to the new node. In Lines 11 and 12, if a goal $\mathbf{q}_{goal}$ has been found, the path is extracted by starting at the goal node and adding all the parent nodes from the goal to the root into the path list. The last line returns the path.

**Algorithm 2** Rapidly exploring Random Tree (RRT)

1: **Input**: $\mathbf{q}_{init}$ → Initial configuration, $\Delta q$ → Incremental distance, C → Configuration space, TIMEOUT → The break time.

2: $\mathbf{T}$.init($\mathbf{q}_{init}$)
3: **Path**.init($\mathbf{q}_{goal}$)
4: **while** $\mathbf{q}_{new} \neq \mathbf{q}_{goal}$ **and not** TIMEOUT **do**:
5: $\quad \mathbf{q}_{rand}$ = RandomSamples(C)
6: $\quad \mathbf{q}_{near}$ = NearestNode($\mathbf{q}_{\text{rand}}, \mathbf{T}$)
7: $\quad \mathbf{q}_{new}$ = ExtendTree($\mathbf{q}_{\text{near}}, \mathbf{q}_{\text{rand}}, \Delta$q)
8: $\quad$ **if** CollisionFree($\mathbf{q}_{near}$, $\mathbf{q}_{new}$) **then**
9: $\quad\quad$ $\mathbf{T}$.AddNode($\mathbf{q}_{new}$)
10: $\quad\quad$ $\mathbf{T}$.AddEdge($\mathbf{q}_{near}, \mathbf{q}_{new}$)
11: **while** **Path**.Parent ≠ Root.Parent **do**:
12: $\quad$ **Path**.AddNode($\mathbf{T}$.Node(**Path**.Parent))
13: $\quad$ **Path**.Parent = $\mathbf{T}$.Edge(**Path**.Parent)
14: **return Path**

## 6.3 RRT*

The RRT* algorithm is implemented and structured as show in Algorithm 3. Since the RRT* algorithm is based on the RRT algorithm described in section 6.2,much of the code is similar. The inputs used in this algorithm are described in Line 1. Lines 2 to 3 initialize the tree and the path. Line 4 contains a for-loop that iterates until a timeout. Lines 5 to 7 contain functions that execute the equivalent of equations 14 to 17 to find the random point $\mathbf{q}_{rand}$, the nearest neighbor $\mathbf{q}_{near}$ and the new node $\mathbf{q}_{new}$.

After the new node is calculated in Line 7, the algorithm performs a collision check on the new node in Line 8. The collision check is described in section 3.3.1. If the line segment between the new node and the neighboring node is collision-free, the algorithm progresses to Lines 9 and 10. In Line 9, the new node is added to a list containing all the nodes of the tree, and in Line 10, the parent is connected to the new node.

When the RRT* algorithm reaches Line 11, the enhancements from section 4.1 begin. In Line 11, the cost of every new node is calculated using equation 18. Line 12 initializes the neighbor list. In Line 13 is a for-loop that loops though every neighbor within the neighborhood radius $R_n$. Line 14 then adds the node that is within that radius.

In Line 15, an if-statement checks if the cost sum of any of the nodes in the neighborhood to the new node is less than the current cost. If the if-statement is true, Line 16 sets the parent of the new node equal to the neighbor, and Line 17 updates the cost of the new node using equation 19. Line 18 ensures that the updated parent and cost are collision free, where in Line 19 the tree is rewired.

In Line 20, another if-statement checks if the cost sum of the new node to any of the neighboring nodes is less than the current neighbor cost. If the statement is true, Line 21 sets the neighbor's parent equal to the new node and Line 22 updates the cost of the new node using equation 20. Line 23 then ensures that the updated parent and cost are collision free, where in Line 24 the tree is rewired. In the Lines 25 to 27, the path is extracted by starting at the most recent goal $\mathbf{q}_{goal}$ solution and adding all the parent nodes from the goal to the root into the path list. The last line returns the path.

**Algorithm 3** Optimized Rapidly exploring Random Tree (RRT*)

1: **Input**: $\mathbf{q}_{init}$ → Initial configuration, $\Delta q$ → Incremental distance, C → Configuration space, TIMEOUT → The break time, $R_n$ → Neighborhood radius.

2: $\mathbf{T}$.init($\mathbf{q}_{init}$)
3: **Path**.init($\mathbf{q}_{goal}$)
4: **for** TIMEOUT **do**:
5: $\mathbf{q}_{rand}$ = RandomSamples(C)
6: $\mathbf{q}_{near}$ = NearestNode($\mathbf{q}_{\text{rand}}, \mathbf{T}$)
7: $\mathbf{q}_{new}$ = CreateNewNode($\mathbf{q}_{\text{near}}, \mathbf{q}_{\text{rand}}, \Delta$q)
8: **if** CollisionFree($\mathbf{q}_{near}$, $\mathbf{q}_{new}$) **then**
9: $\mathbf{T}$.AddNode($\mathbf{q}_{new}$)
10: $\mathbf{T}$.AddEdge($\mathbf{q}_{near}, \mathbf{q}_{new}$)
11: $\mathbf{T}$.AddCost($\mathbf{q}_{near}, \mathbf{q}_{new}, \mathbf{q}_{near}$.Cost))
12: **Neighbor**.init()
13: **for** $||\mathbf{q}_{new} - \mathbf{T}.\text{Node}|| < R_n$ **do**
14: **Neighbor**.AddNode($\mathbf{T}$.Node)
15: **if** ((**Neighbor**.Cost + $||\mathbf{q}_{new} - \mathbf{Neighbor}.\text{Node}||$) < $\mathbf{q}_{new}$.Cost) **then**
16: $\mathbf{q}_{new}$.Parent = **Neighbor**.Index
17: $\mathbf{q}_{new}$.Cost = **Neighbor**.Cost + $||\mathbf{q}_{new} - \mathbf{Neighbor}.\text{Node}||$
18: **if** CollisionFree($\mathbf{q}_{new}$.Parent, $\mathbf{q}_{new}$) **then**
19: $\mathbf{T}$.Exchange($\mathbf{q}_{new}$.Parent, $\mathbf{q}_{new}$.Cost)
20: **if** (($\mathbf{q}_{new}$.Cost + $||\mathbf{q}_{new} - \mathbf{Neighbor}.\text{Node}||$) < **Neighbor**.Cost) **then**
21: **Neighbor**.Parent = $\mathbf{q}_{new}$.Index
22: **Neighbor**.Cost = $\mathbf{q}_{new}$.Cost + $||\mathbf{q}_{new} - \mathbf{Neighbor}.\text{Node}||$
23: **if** CollisionFree(**Neighbor**.Parent, **Neighbor**) **then**
24: $\mathbf{T}$.Exchange(**Neighborhood**.Parent, **Neighborhood**.Cost)
25: **while** **Path**.Parent ≠ Root.Parent **do**:
26: **Path**.AddNode($\mathbf{T}$.Node(**Path**.Parent))
27: **Path**.Parent = $\mathbf{T}$.Edge(**Path**.Parent)
28: **return Path**

# 7 Case studies

The major goal of this project is to test the performance of different path planning algorithms in a realistic scenario related to aquaculture and complex environments. One of the case studies that was selected was path planning in a fish cage. An illustration of the fish cage used in this project is shown in Figure 9.

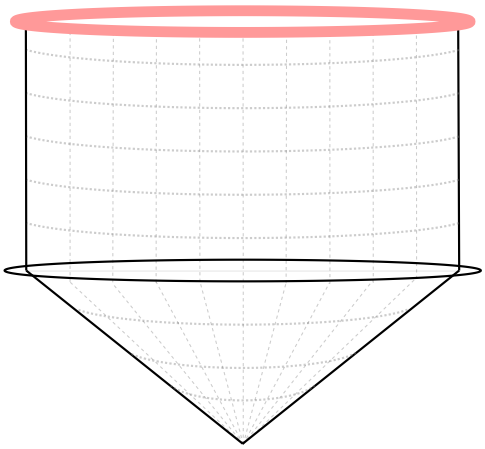

Figure 9: Fish cage

As shown in the figure, the fish cage consists of a floating ring platform (in red) at the top and a net. The simulation also included obstacles inside the fish cage.
The final case study involved path planning among multiple fish cages. To simulate this, there was placed four pairs of fish cages in a specific area. As an extra environment the RRT* was tested in a high obstacle environment, for the purpose of showing the different solutions of the RRT*.

# Part IV
# Simulations

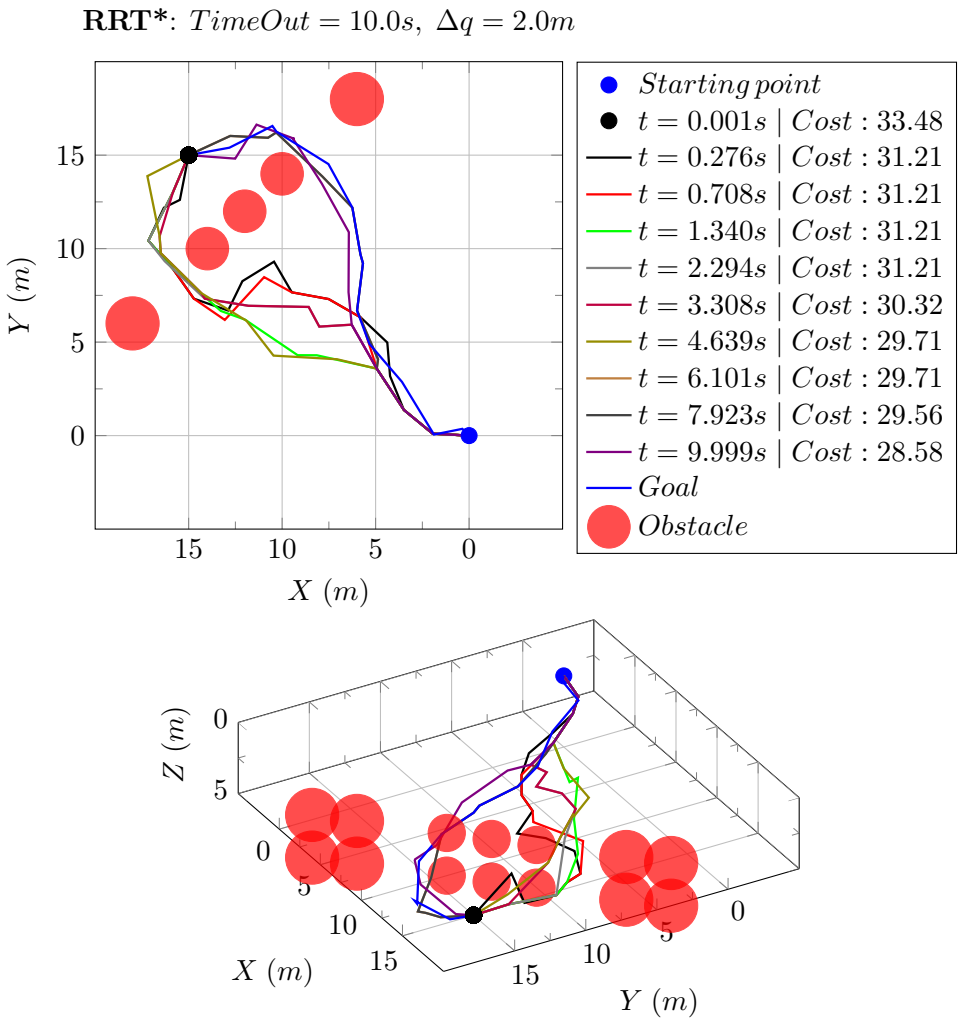


Figure 10: The RRT* algorithm was tested in a scenario with obstacles (shown in red) and a time out of 10 seconds.

Figure 10 shows the progression of the solution from the starting point (in blue) to the goal (in black) as it avoids obstacles. The figure also shows ten solutions that obtain a better cost at each time step, with the first solution (in black) taking a different route than the final solution (in blue). The obstacles in the simulation is placed in a way that technically allows the path planner to find a solution between the obstacles. This occurred previously

while simulating, but given the nature of the RRT/RRT* it won't happen every time. The first plot is a 2D representation, while the second is in 3D.

## 8 Case study: Path planning among fish cages

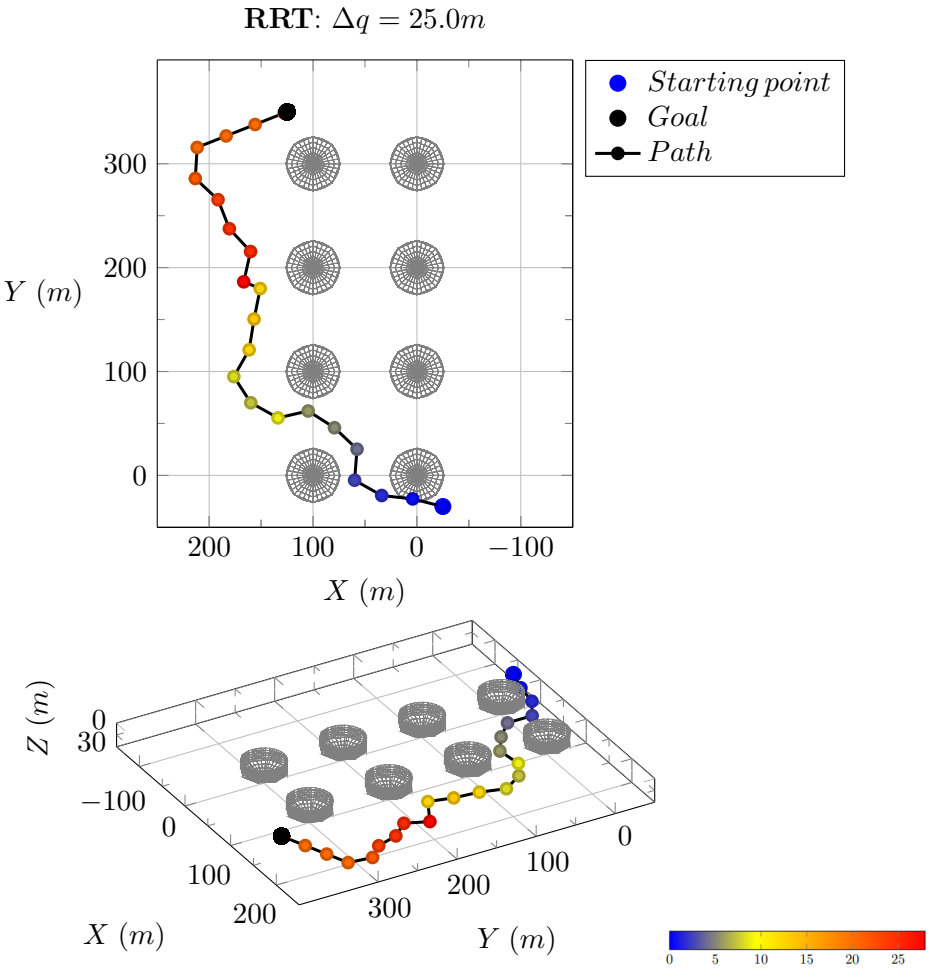


Figure 11: The figure shows the performance of the RRT algorithm among fish cages, starting from a point (in blue) and reaching the goal (in black).

Figure 11 generates a path (shown as a scatter line) that avoids the fish cages (in gray) by going around them. The path is color-coded, with red indicating lower depths ($Z = 30m$) and blue indicating surface level ($Z = 0$). The first plot is a 2D representation, while the second is in 3D.

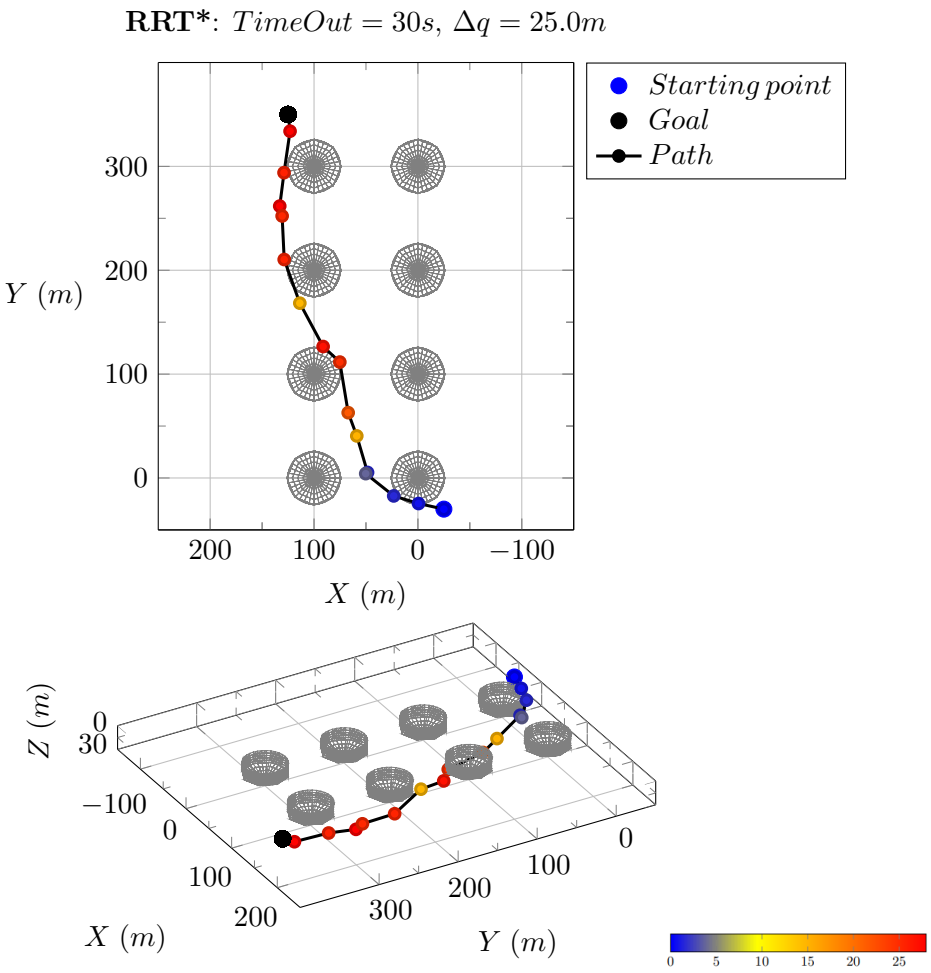


Figure 12: The figure shows the performance of the RRT* algorithm among fish cages, starting from a point (in blue) and reaching the goal (in black), with a time out of 30 seconds.

Figure 12 generates a path (shown as a scatter line) that avoids the fish cages (in gray) by staying close to them without colliding. The path is color-coded, with red indicating lower depths ($Z = 30m$) and blue indicating surface level ($Z = 0$). The first plot is a 2D representation, while the second is in 3D.

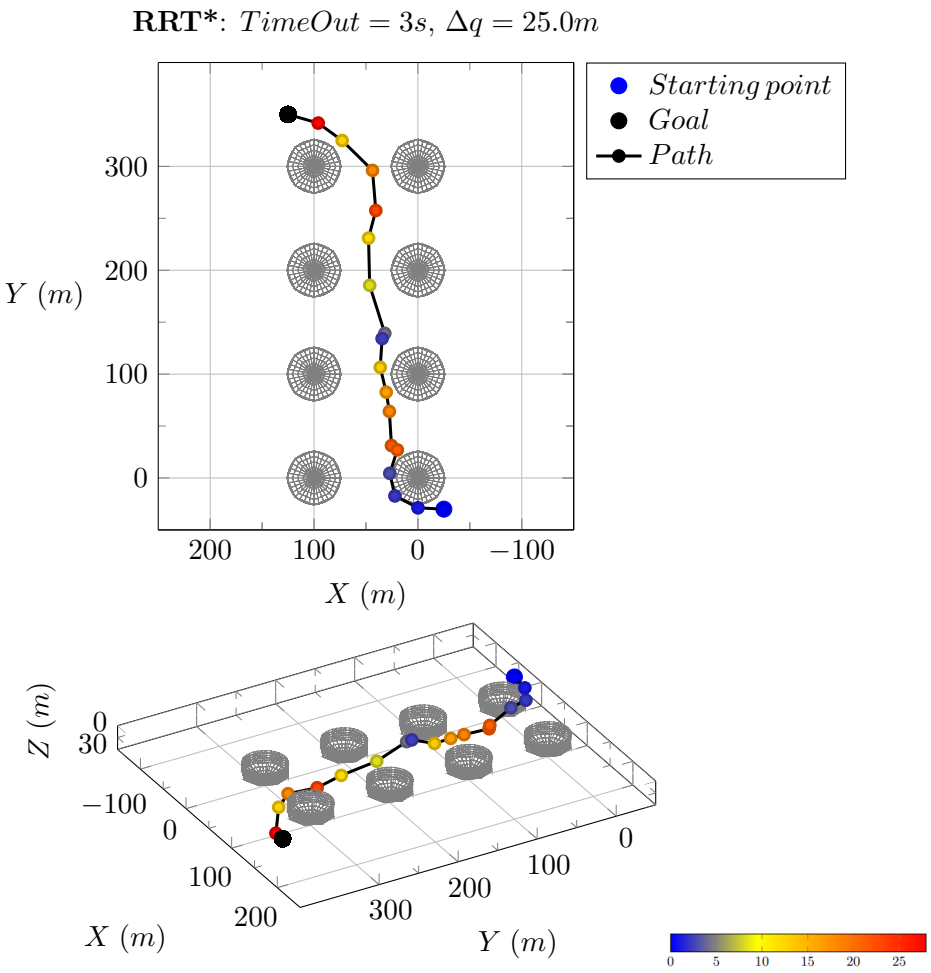


Figure 13: The figure shows the performance of the RRT* algorithm among fish cages, starting from a point (in blue) and reaching the goal (in black), with a time out of 3 seconds.

Figure 13 generates a path (shown as a scatter line) that avoids the fish cages (in gray) by traveling between the pairs of cages without colliding. The path is color-coded, with red indicating lower depths($Z = 30m$) and blue indicating surface level ($Z = 0$). The first plot is a 2D representation, while the second is in 3D.

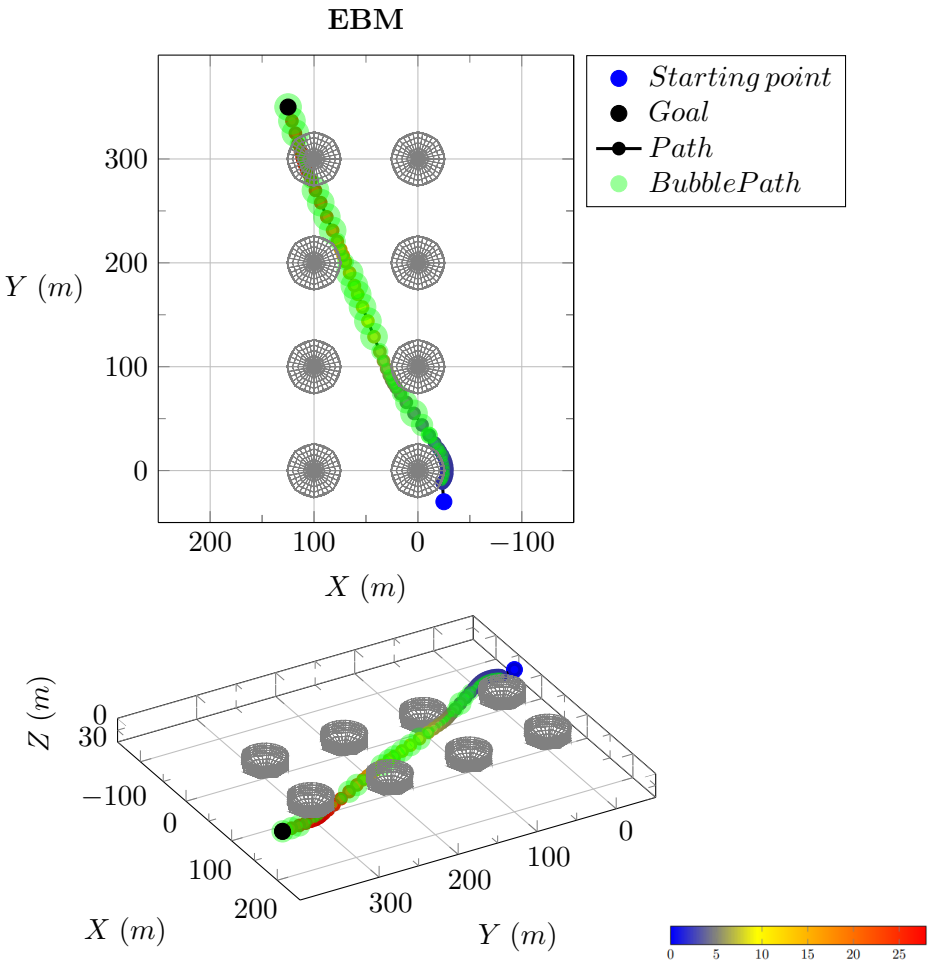


Figure 14: The figure shows the performance of the EBM algorithm among fish cages, starting from a point (in blue) and reaching the goal (in black).

Figure 14 creates a band of partly overlapping bubbles (in green), and the resulting path (shown as a scatter line) avoids the fish cages (in gray) by using different bubble sizes. In the figure, the EBM avoids the fish cages by going to the side. When it reaches the last cage, the bubbles go beneath and a little to the side. The path is color-coded, with red indicating lower depths ($Z = 30m$) and blue indicating surface level ($Z = 0$). The first plot is a 2D representation, while the second is in 3D.

# 9 Case study: Path planning in fish cage

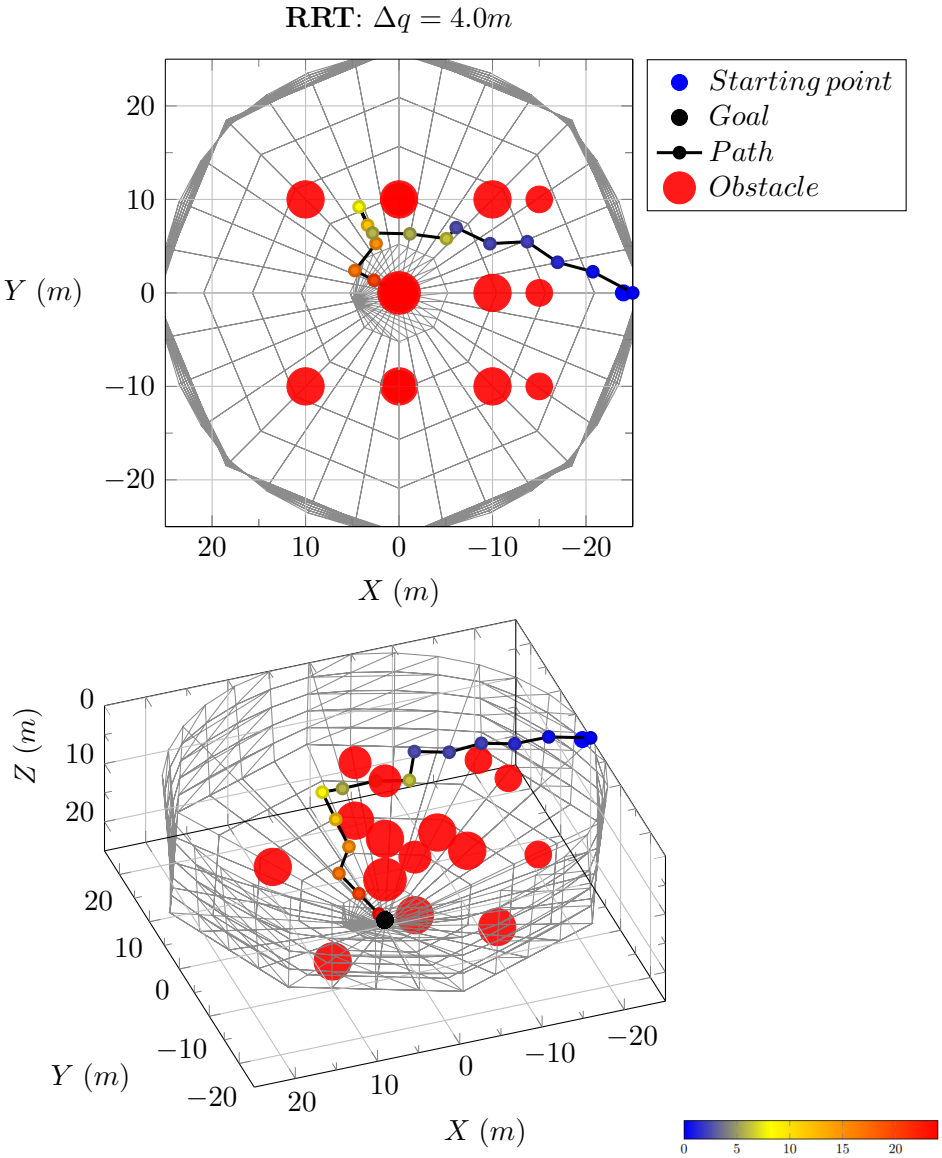


Figure 15: The figure shows the performance of the RRT algorithm within a fish cage, starting from a point (in blue) and reaching the goal (in black).

Figure 15 generates a path (shown as a scatter line) that avoids the obstacles (in red) by going between them, resulting in a choppy path. The path is color-coded, with red indicating lower depths($Z = 25m$) and blue indicates surface level ($Z = 0m$). The first plot is a 2D representation, while the second is in 3D.

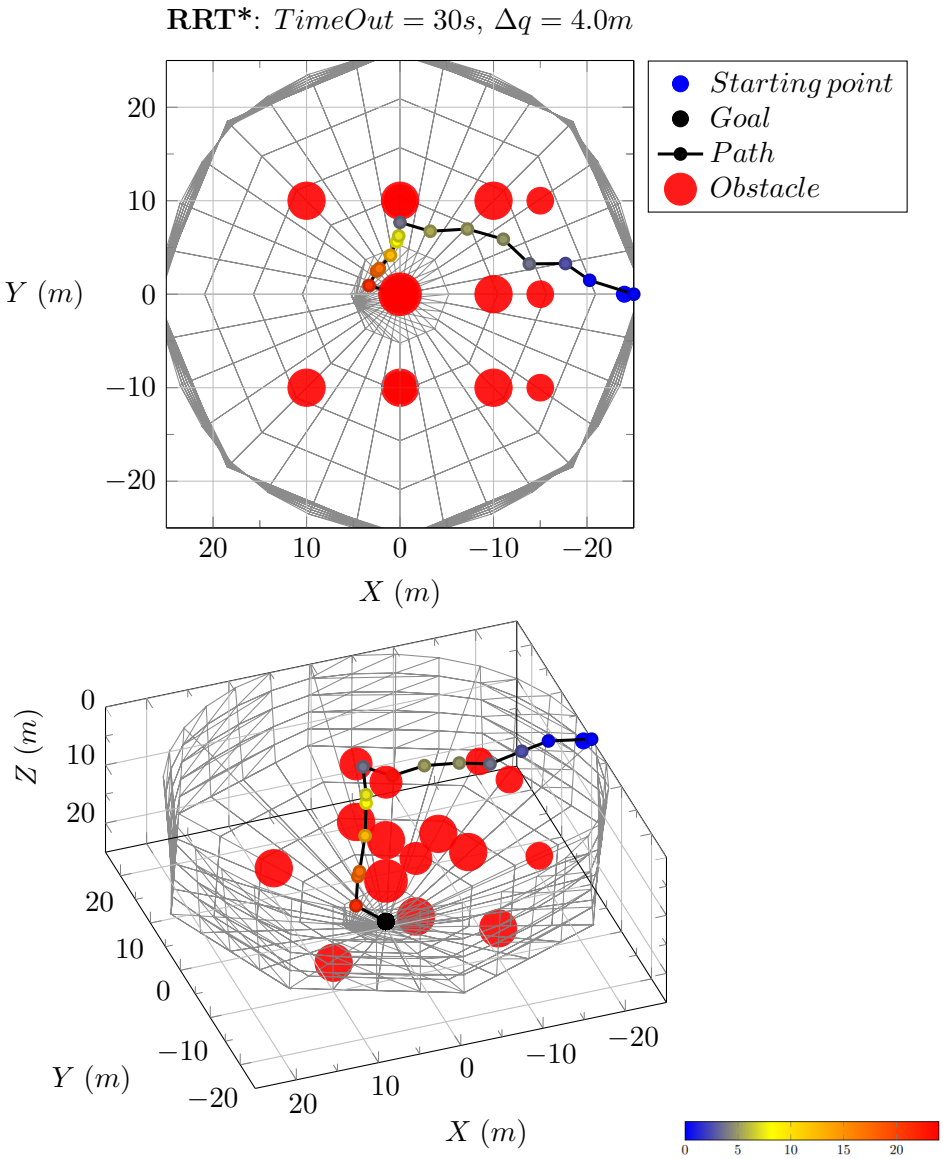


Figure 16: The figure shows the performance of the RRT* algorithm within a fish cage, starting from a point (in blue) and reaching the goal (in black), with a time out of 30 seconds.

Figure 16 generates a path (shown as a scatter line) that avoids the obstacles (in red) within the fish cage by navigating between the rows of obstacles, resulting in a path that is close to smooth. The path is color-coded, with red indicating lower depths ($Z = 25m$) and blue indicates surface level($Z = 0m$).The first plot is a 2D representation, while the second is in 3D.

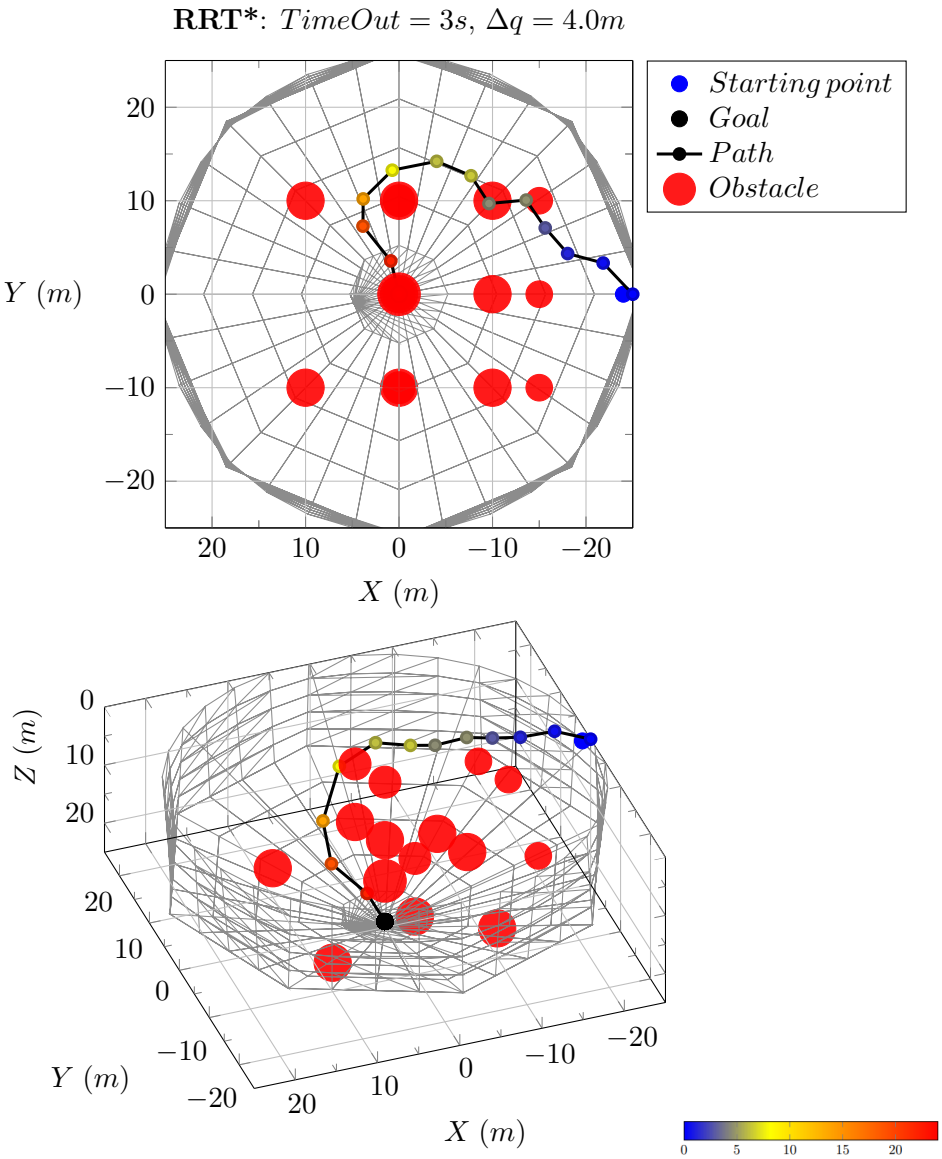


Figure 17: The figure shows the performance of the RRT* algorithm within a fish cage, starting from a point (in blue) and reaching the goal (in black), with a time out of 3 seconds, which is closer to real time.

Figure 17 generates a path (shown as a scatter line) that avoids the obstacles (in red) within the fish cage by navigating above and around the rows of obstacles, resulting in a path that is close to smooth. The path is color-coded, with red indicating lower depths ($Z = 25m$) and blue indicates surface level($Z = 0m$). The first plot is a 2D representation, while the second is in 3D.

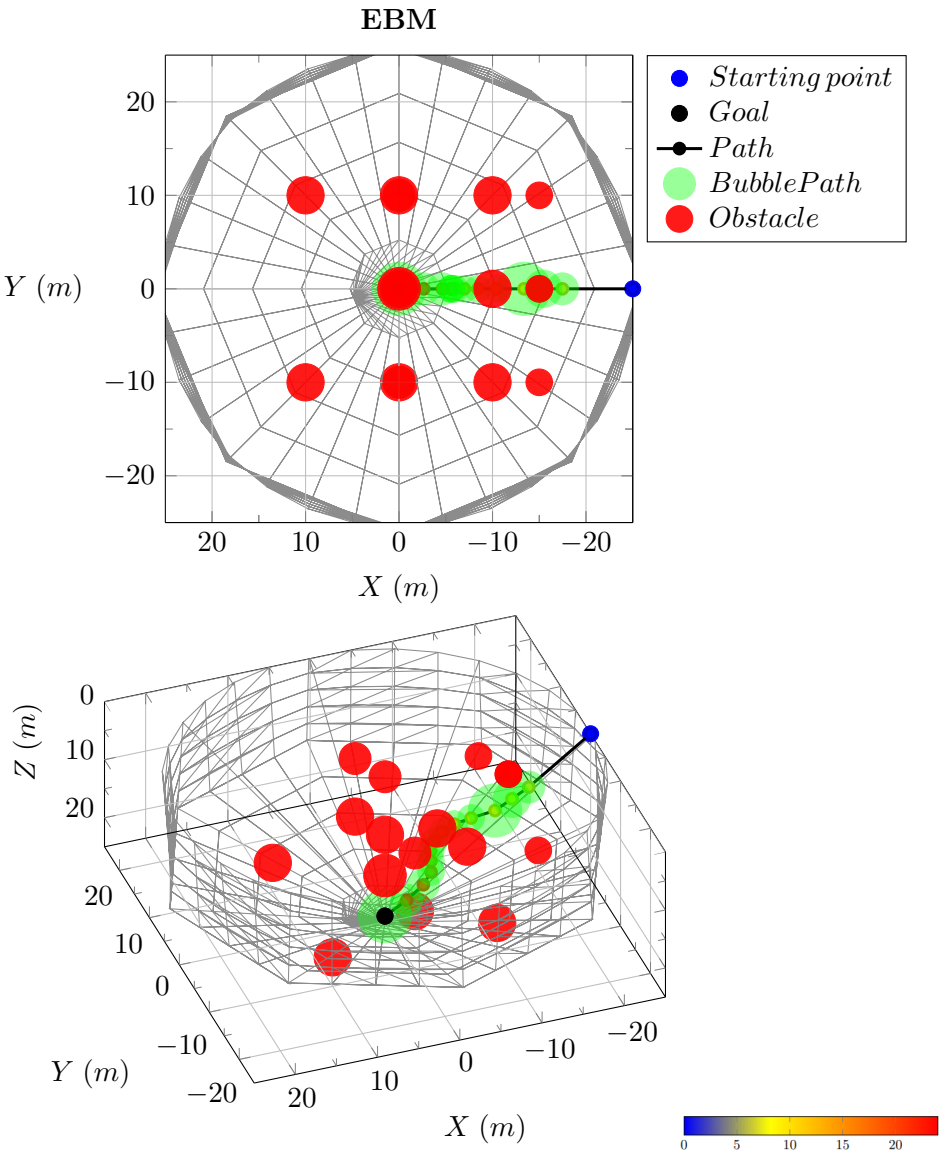


Figure 18: The figure shows the performance of the EBM algorithm within a fish cage, starting from a point (in blue) and reaching the goal (in black).

Figure 18 creates a band of partly overlapping bubbles (in green), and the resulting path (shown as a scatter line) avoids the obstacles (in red) by using different bubble sizes. In the figure, the EBM navigates the bubbles straight towards the goal while avoiding the obstacles. The path is color-coded, with red indicating lower depths($Z = 25m$) and blue indicates surface level($Z = 0$). The first plot is a 2D representation, while the second is in 3D.

# Part V
# Discussion

The simulation results in Section IV highlight the differences between the various path planning algorithms. Figure 10 specifically shows the improvement of the RRT* algorithm over its initial solution after 10 seconds. This is interesting because it shows how the RRT* algorithm is able to improve its initial solution and decrease the cost of the solution over time. The results of the simulation provide insight into the performance of these algorithms and can help inform decision-making for future path planning tasks.

The simulation results in Section 8 show the performance of different path planning algorithms among fish cages. The RRT algorithm is observed to take a longer route by avoiding obstacles, while the RRT* and EBM show improved performance by taking a shorter path. The RRT* was simulated twice with different timeouts to show real-time and more computation time solutions. In both cases, the RRT* was able to navigate smoothly between obstacles and reach the goal. The EBM generated a line from the starting point to the goal, with small changes in bubble positions and bubble sizes indicating the distance to obstacles. These results indicate that the RRT* and EBM perform better than the RRT in terms of finding the shortest path and navigating smoothly between obstacles. It is worth noting that when the map size changed for the RRT and RRT*, the step sizes needed to be adjusted in order for the difference in solutions to be noticeable.

The simulation results in Section 9 show the performance of different path planning algorithms within a fish cage. The RRT algorithm behaves similarly to the previous case study, navigating around obstacles and towards the goal located at the bottom center of the fish cage. The RRT* was simulated twice with different timeouts to show real-time and more computation time solutions. In both cases, the RRT* was able to navigate smoothly between obstacles and reach the goal. The EBM generated a bubble path, traveling in a straight line from the starting point to the goal, showing a clear difference in performance compared to the other algorithms. These results indicate that the EBM perform better than the RRT and the RRT* in terms of finding the shortest path and navigating smoothly between obstacles in this type of environment.

# Part VI
# Conclusions and Recommended Future Work

In conclusion, all the path planners performed well and managed to create a trajectory from a starting point to a goal pose without colliding in to any obstacles. The simulation results demonstrate the performance of different path planning algorithms in fish cage environments. The RRT algorithm was observed to take a longer route by avoiding obstacles, while the RRT* and EBM showed improved performance by taking a shorter path. When the configuration space changed for the RRT and RRT*, the step sizes needed to be adjusted in order for the difference in solutions to be noticeable. The EBM performed better than the RRT and the RRT* in terms of finding the shortest path and navigating smoothly between obstacles. EBM was also found to be the most efficient algorithm among RRT, RRT*, and itself in terms of computation time. Specifically, EBM had the fastest computation time, followed by RRT, and then RRT*. These results provide insight into the performance of these algorithms and can help inform decision-making for future path planning tasks.

## 10 Future Work

As a future direction, it may be worthwhile to investigate more complex path planners that can outperform and potentially replace the EBM that SINTEF Ocean is currently using. This could involve exploring different algorithms and techniques that are able to find even shorter paths and navigate more efficiently through complex environments, one approach could be to implement the *Trajopt* path planner from [16]. By doing so, it may be possible to improve the performance of path planning systems for applications such as fish cage environments.

# Part VII
# Appendices

## References

[1] Forskning og utvikling for realisering av havbruk til havs innspill til strategiske prioriteringer mot 2040. `https://www.ntnu.no/nyheter/wp-content/uploads/2022/11/Havbruk-til-havs_versjon-1.pdf`. Accessed: 2022-12-13.

[2] Herman Biørn Amundsen. Three-dimensional collision avoidance for unmanned underwater vehicles using elastic bands. Manuscript in preparation.

[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] 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.

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

[6] 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.

[7] 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.

[8] Dmitry Berenson and Siddhartha S. Srinivasaz. Probabilistically complete planning with end-effector pose constraints. In *2010 IEEE International Conference on Robotics and Automation*, pages 2724–2730, 2010.

[9] 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.

[10] Chi-Tai Lee and Ching-Chih Tsai. 3d collision-free trajectory generation using elastic band technique for an autonomous helicopter. In Tzuu-Hseng S. Li, Kuo-Yang Tu, Ching-Chih Tsai, Chen-Chien Hsu,

Chien-Cheng Tseng, Prahlad Vadakkepat, Jacky Baltes, John Anderson, Ching-Chang Wong, Norbert Jesse, Chung-Hsien Kuo, and Haw-Ching Yang, editors, *Next Wave in Robotics*, pages 34–41, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[11] Martin Føre, Sverre Fjæra, Sveinung Johan Ohrem, Eleni Kelasidi, Nina Bloecher, and Herman Biørn Amundsen. Adaptive motion planning and path following for permanent resident biofouling prevention robot operating in fish farms. In *OCEANS 2021: San Diego – Porto*, pages 1–10, 2021.

[12] Claude Lemaréchal. Cauchy and the gradient method. *Documenta Mathematica*, pages 251–254, 2012.

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

[14] 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.

[15] 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.

[16] Marios Xanthidis, Nare Karapetyan, Hunter Damron, Sharmin Rahman, James Johnson, Allison O'Connell, Jason M. O'Kane, and Ioannis Rekleitis. Navigation in the presence of obstacles for an agile autonomous underwater vehicle. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 892–899, 2020.