FEEDBACK MOTION PLANNING OF UNMANNED UNDERWATER VEHICLES VIA RANDOM SEQUENTIAL COMPOSITION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

EMRE EGE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2019

Approval of the thesis:

FEEDBACK MOTION PLANNING OF UNMANNED UNDERWATER VEHICLES VIA RANDOM SEQUENTIAL COMPOSITION

submitted by **EMRE EGE** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İlkay Ulusoy
Head of Department, **Electrical and Electronics Engineering** _____

Prof. Dr. Umut Orguner
Supervisor, **Electrical and Electronics Eng., METU** _____

Assist. Prof. Dr. M. Mert Ankaralı
Co-supervisor, **Electrical and Electronics Eng., METU** _____

Examining Committee Members:

Prof. Dr. M. Kemal Leblebicioğlu
Electrical and Electronics Engineering, METU _____

Prof. Dr. Umut Orguner
Electrical and Electronics Engineering, METU _____

Prof. Dr. M. Önder Efe
Computer Engineering, Hacettepe University _____

Prof. Dr. A. Egemen Yılmaz
Electrical and Electronics Engineering, Ankara University _____

Assoc. Prof. Dr. Erol Şahin
Computer Engineering, METU _____

Date: 19.09.2019

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Emre Ege

Signature :

ABSTRACT

FEEDBACK MOTION PLANNING OF UNMANNED UNDERWATER VEHICLES VIA RANDOM SEQUENTIAL COMPOSITION

Ege, Emre
Ph.D., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Umut Orguner
Co-Supervisor: Assist. Prof. Dr. M. Mert Ankaralı

September 2019, 93 pages

In this thesis, we propose a new motion planning method to robustly and computationally efficiently solve (probabilistic) coverage, path planning, and navigation problems for unmanned underwater vehicles (UUVs). Our approach is based on synthesizing two existing methodologies: sequential decomposition of dynamic behaviors and rapidly exploring random trees. The main motivation for this integrated solution is a robust feed-back based and computationally feasible motion planning and navigation algorithm that takes advantage of these two planning approaches. To illustrate the main approach and show the feasibility of the method, we first performed 2D simulations in MATLAB. We then implemented our method using a realistic fully dynamic 3D UUV simulation environment based on a platform built on the Robot Operating System (ROS)/Gazebo interface to test the overall performance and applicability for real applications. We also tested the robustness of the method under extreme environmental uncertainty (water current that is half the maximum speed of the UUV). 2D and realistic 3D simulation results indicate that our method can produce robust and computationally feasible solutions for a broad class of UUVs and Unmanned Surface

Vehciles (USVs).

Keywords: Unmanned Underwater Vehicles(UUV), Feedback Motion Planning; UUV Simulation, Sequential Composition, RRT, ROV, AUV

ÖZ

İNSANSIZ SUALTI ARAÇLARININ RASTSAL SIRALI BİLEŞİM METODU ARACILIĞIYLA GERİBESLEMELİ HAREKET PLANLAMASI

Ege, Emre
Doktora, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Umut Orguner
Ortak Tez Yöneticisi: Dr. Öğr. Üyesi. M. Mert Ankaralı

Eylül 2019 , 93 sayfa

Tez çalışması kapsamında, insansız sualtı araçlarında tarama, yol planlama ve seyrüsefer problemlerinin çözümünde kullanılabilecek gürbüz ve hesap etkin yeni bir hareket planlama algoritması önerilmiştir. Yaklaşımımız literatüre yer alan iki faklı teknikiğin sentezlenmesi üzerine kurulmuştur: dinamik denetleyicilerin sıralı dizimi ve hızlı büyüyen rastlantısal ağaçlar. Birleştirilmiş metodun temel motivasyonu, bu iki metodun avantajlarını kullanan yeni bir gürbüz, geri-beslemeli ve hesaplama etkin yeni bir metod oluşturmaktır. Önerilen metodun uygunluğunu göstermek amacıyla ilk olarak MATLAB'da iki boyutlu benzetimler gerçekleştirilmiştir. Ardından, metod ROS/Gazebo arayüzü üzerinde, gerçekçi ve tam dinamik üç boyutlu sualtı aracı benzetiminde gerçek hayat uygulamalarına uygunluğu test edilmiştir. Ayrıca, metodun gürbüzlüğü, uç çevresel belirsizlikler (su akıntısının araç hızının yarısı kadar olduğu durumlarda) altında test edilmiştir. İki boyutlu ve üç boyutlu benzetim sonuçları önerilen metodun sualtı ve suüstü araçları gürbüz ve hesap etkin bir yöntem olduğunu doğrulamıştır.

Anahtar Kelimeler: İnsansız Sualtı Araçları, Geribeslemeli Hareket Planlama, İSA Simulasyon, Sıralı Bileşim, RRT, ROV, AUV

To my family...

ACKNOWLEDGMENTS

I would like to thank my supervisor Prof. Umut ORGUNER, for his constant support, guidance, and patience. It has been a great honor to work with him. I also would like to thank Prof. M. Kemal LEBLEBİCİOĞLU and Prof. M. Önder EFE for their valuable feedback on the progress of this work.

It is especially a pleasure for me to express my sincere gratitude to my thesis co-supervisor, Assist. Prof. Dr. M. Mert ANKARALI for his belief, patience, encouragement and guidance throughout the study. His guidance helped me throughout the research and writing of this thesis. I could not have imagined having a better mentor for my Ph.D. study.

This study is partially supported by the SWARMs European project (Smart and Networking Underwater Robots in Cooperation Meshes; grant number 662107-SWARMs-ECSEL-2014-1, partially supported by the ECSEL JU) and The Scientific and Technological Research Council of Turkey (TÜBITAK) (project number 118E195).

I must also express my very profound gratitude to the members of my family. I would like to thank my parents, my loving mother, Azize EGE, and my dear father, Cemil EGE, whose love and guidance are with me in whatever I pursue. I would like to offer my special thanks to my elder brother, Yunus EGE, for his constant encouragement and moral support throughout my research work. Finally, I owe my loving thanks to my wife, Gülhan TURAN EGE. I would like to thank her for standing beside me throughout my life and this study. I would also thank our two wonderful children, Kadir Tuna and İdil, for making my life enjoyable and meaningful.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ALGORITHMS

ALGORITHMS

LIST OF ABBREVIATIONS

ABBREVIATIONS

2D	2-Dimensional
3D	3-Dimensional
AUV	Autonomous Underwater Vehicle
CO	Center of Orientation
COB	Center of Buoyancy
COG	Center of Gravity
DOF	Degree of Freedom
DVL	Doppler Velocity Logger
IMU	Inertial Measurement Unit
NED	North-East-Down
PID	Proportional-Integral-Derivative
ROS	Robot Operating System
ROV	Remotely Operated Vehicle
RPM	Revolutions Per Minute
RRT	Rapidly Exploring Random Trees
URDF	Unified Robot Description Format
USBL	Ultra-Short Baseline Tracking
UUV	Unmanned Underwater Vehicle
XML	Extensible Markup Language

CHAPTER 1

INTRODUCTION

Motion and path planning is a mature field in the robotics community and has been practically used for many applications, from aerial vehicles to underwater vehicles, which are the main application domain for this thesis. There are numerous different definitions for motion and path planning. One definition can be summarized as follows: given a robot (or group of robots), model of the environment, initial conditions, and goal task (this can be a final position, set of positions or desired trajectories), find a set of inputs that will achieve the goal task such that the robot(s) will not collide with any of the obstacles in the environment.

Numerous planning algorithms have been developed in the last decade, each having different advantages and disadvantages, from the naive bug algorithms to more advanced approaches such as sampling-based methods [3].

Our goal in this thesis is to develop a computationally feasible, robust feedback-based motion planning algorithm for a class of unmanned underwater (and surface) vehicles. A huge majority of planning studies concentrate on generating "open-loop" trajectories that are then presumably tracked using separate feedback controllers. Even the definition of motion planning is based on an open-loop like navigation methodology. Such a decoupling or separation can be problematic in the sense that resulting trajectories can be difficult to navigate and control, and non-robustness can be unavoidable to external uncertainties. In the end, the main disadvantage of open-loop solutions– compared to closed-loop solutions– in all engineering fields is their general relative lack of robustness.

Another well-known problem associated with many planning algorithms is their com-

putational and practical complexity, which is of course highly dependent on the nature of the problem, robot, and environment. For example, there are some methods [4] that solve the well-known piano movers problem [5], but many of the solutions suffer from computational complexity and are not feasible for hardware-based applications. Another interesting example is the method of cell decompositions [6], which provides powerful solutions and performance for limited configuration spaces and relatively simple environments, since it relies on explicit representations of the obstacles in the configuration space.

There are also many feedback-based methods that provide effective solutions for many problems, such as potential field approaches [7] or navigation functions [8]. The main drawback of the potential field approach is the existence of local minima [9], which are unavoidable for high-dimensional problems, whereas navigation functions could easily add substantial computational burden for even fairly complex environments and robots.

In this respect, our goal is to develop a computationally feasible and robust feedback based motion planning method that solves planning, navigation, and control problems for a class of unmanned underwater or surface vehicles.

1.1 Background I: Sampling-Based Methods

Deterministic motion planning methods that guarantee a solution rely on explicit representation of the environment to solve the planning problem. As mentioned before, this type of approach can result in an excessive computational burden in high dimensional configuration spaces, and relatively complex environments.

To overcome the limitations of such deterministic algorithms, a variety of sampling-based methods have been developed in the last 20 to 30 years [10, 3], including very recent work [11, 12]. These algorithms produce effective solutions even in high-dimensional spaces. For this reason, such methods have gained great popularity in the robotics community.

Unlike deterministic approaches, sampling-based planners rely on a module that checks

for collisions (between a "sample" and the obstacles). Planning algorithms use this module to find a set of collision-free points, which are then connected to construct a roadmap [10] or a tree [3] of collision-free (generally open-loop) trajectories. Such a graph or tree can be used to find one or more solution for the motion-planning paradigm. These methods can provide a probabilistically guaranteed completeness to the problem.

1.1.1 Rapidly Exploring Random Trees

Rapidly exploring random tree (RRT) is a popular sampling-based planning algorithm that can effectively search complex, high-dimensional configuration spaces by randomly building a tree of samples [3]. The original RRT algorithm constructs a tree incrementally from an initial condition toward a goal state by sampling the configuration space. Due to the nature of the RRT algorithm, the tree is biased to grow toward untouched regions of the space. The RRT tree is then used to construct an open-loop trajectory for the robot. Given the incremental structure of the original algorithm, it provides only a single path between the initial and goal states.

At each sampling instant, a connected path is constructed between the random sample and the nearest node/state in the tree. If the connected path is collision-free and feasible then this new node is added to the tree with the established connection. The *distance* between the new sample and existing nodes is limited by a threshold. If the distance is above this threshold, a new node candidate is created at the maximum distance threshold in the same *direction* as the original sample from the nearest existing node. This new candidate is used to construct a path and grow the tree. In this context, the random samples from the environment indeed control the direction of the growth, while the threshold controls the growth rate. This property specifically results in the space-filling bias of the RRT. The RRT algorithm for a given configuration space $\mathcal{C}$ is given in Algorithm 1. There are some extensions of the original algorithm in the literature [13, 14, 15] to improve the performance for specific settings. Our method also extends the RRT algorithm by combining it with a feedback-based navigation and planning method.

Algorithm 1: Basic RRT Algorithm

Input : Initial configuration q_{init}, number of vertices in RRT K, incremental distance Δq

Output: RRT graph $\mathcal{G}$

$\mathcal{G}$.init(q_{init});
for $k = 1$ **to** K **do**
    $q_{rand} \leftarrow$RAND_CONF() ;
    $q_{near} \leftarrow$NEAREST_VERTEX($\mathcal{G}, q_{rand}$);
    $q_{new} \leftarrow$NEW_CONF($q_{near}, \Delta q$);
    $\mathcal{G}$.add_vertex(q_{new});
    $\mathcal{G}$.add_edge(q_{near}, q_{new});
end
return $\mathcal{G}$

1.2 Background II: Sequential Composition of Robot Behaviors

One of the pioneering concepts in feedback-based motion planning for robotic systems is the idea of sequential composition of a set of feedback control policies introduced by [1]. This idea has been applied (with some extensions and modifications) to a variety of robotic applications [16, 17, 18, 19]. This study uses the *funnel* abstraction [1, 20] for describing the individual feedback control policies. In this abstraction, for a dynamical robot behavior that is controlled via a stabilizing feedback controller, the funnel inlet and outlet correspond to metaphors for basins of attraction and stable equilibrium respectively. The major problem is that it is in general very difficult (or sometimes impossible according to [21]) to find a single control policy that will attract all the initial states of a robotic system to the goal state in a relatively complex environment with obstacles and other constraints.

The idea behind this approach is introducing a palette of different funnels (control policies) whose inlets and outlets are located inside obstacle-free and unconstrained regions of the state space such that the union of inlets (domains of attraction) of all funnels will cover all the regions of interest (presumably large) and each funnel's outlet is placed inside another funnel.

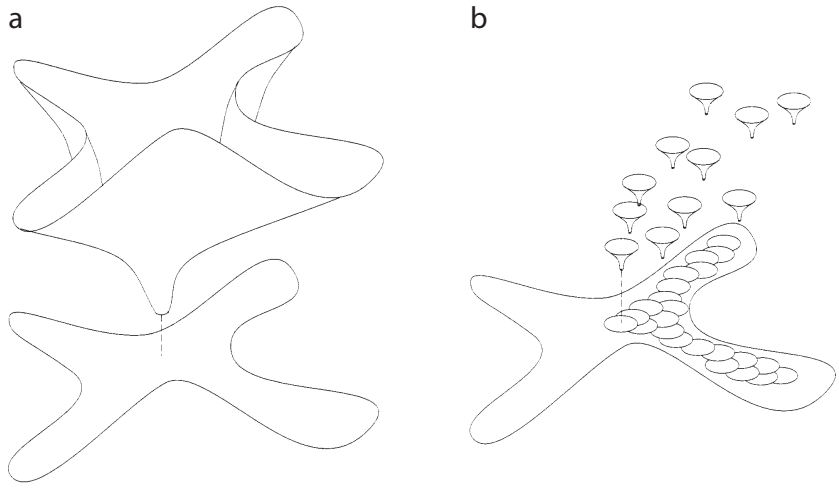


Figure 1.1: Illustrative comparison of global control policy and sequential composition of control policies approaches. Both figures have been acquired from [1]. **a)** A single "global" funnel, the inlet of which is the whole obstacle-free region of the space. As illustrated in the figure, the shape of the funnel is very complex. **b)** One method for sequential composition of funnels. A particular funnel is activated based on the location and priority queue. As illustrated, the system can reach the goal state with this approach by combining simpler control policies.

Using such a funnel (control policy) palette, it would be possible to bring any state inside the combined region of attraction to the goal state through a backchaining operation based on proper activation and switching operations among different funnels. Figure 1.1 compares the global control policy approach and the sequential composition of controllers (SCC) approach for a sample environment.

The main disadvantage of the implementation of sequentially combined controllers for planning is the computational complexity. The majority of applications of this idea rely on explicit representations of the obstacles in the state space and finite-element-like numerical discretization [16].

CHAPTER 2

FEEDBACK MOTION PLANNING USING RANDOM SEQUENTIAL COMPOSITION

2.1 Introduction

The core goal of this thesis is to develop a robust, computationally feasible, and sparse methodology for the motion planning problem for a class of unmanned surface vehicles. Our main approach is combining two different planning approaches, RRT and SCC, by eliminating the respective disadvantages of both worlds and taking into account the specific application domain of unmanned underwater vehicles (UUVs).

Most RRT-based planners determine discretized open-loop trajectories. Indeed, there exist closed-loop RRT extensions [22, 23], which basically adopt an integrated feedback-based approach and enhance the robustness of the RRT. However, neither of these extensions addresses the concept of sparsity. One of the main objectives of this study is obtaining a relatively sparse random tree structure from the environment by integrating the idea of SCC. We believe that sparsity is desirable for remotely operated vehicles (ROV)/ autonomous underwater vehicles (AUV) applications, since it is entirely unnecessary to fill empty regions with dense samples. The methods proposed in [24], [25] and [26], extend the RRT algorithm to cover the configuration space sparsely but they generate open-loop paths.

An important difference between RRT-based methods (open- or closed- loop) and SCC is that RRT-like methods rely on sampling *points* from the configuration space, whereas SCC involves designing a set of *funnels* (or regions). Due to nature of the sampling strategy, RRT-like methods densely fill the state-space with state samples (which also can require significant storage and computational complexity for some

problems), whereas the SCC method can cover a region with sparse funnels. In other words, robustness and sparsity constitute the strength of SCC. However, RRT-like approaches have gained their popularity because of their relative computational feasibility.

It should be also noted that some recent studies combine the fundamental idea of sequential composition with other sampling-based planning approaches [27, 28, 12]. Although our approach shares similarities with these studies, we present important differences and methods in this study. We also aim to develop an effective method specifically for ROV/AUV applications.

Key differences exist between our approach and the methods developed by Tedrake and his colleagues [29, 27, 11, 12]. They mainly developed their techniques to solve motion control problems for highly dynamic and non-linear robotic systems that is, systems in which second-order dynamics are critical. In this respect, they adopted a three-step motion planning strategy. In the first step, using some existing path planning techniques, they generate an open-loop trajectory between the initial and goal configurations. In the second stage, they generate a sparse feedback policy around this trajectory. In the third step, they generate random trajectories and feedback control policies that connect to the main branch to form a sparse tree structure. This class of methods, because of their multi-staged nature and other features, require heavy optimization and numerical steps to solve the motion planning problem. Practical implementation of our method is significantly more feasible, and computational complexity is greatly reduced. Moreover, the main limitation of this approach (from our perspective) is its dependence on pre-generated open-loop trajectories. Our goal in this study is to develop a standalone feedback motion planning approach that is free from open-loop trajectories. For these reasons, the methods in this class are fundamentally different from what we describe in our study.

The method most relevant for our study is one developed by [28]. Similar to our motion planning solution, these authors also utilize randomly generated and connected regions (these are circular regions in both their method and our) and generate feedback motion strategies inside these regions. However, they build a random connected graph structure, whereas we generate a connected tree structure. Probabilistic graphs

and random trees are widely used in the sampling-based motion planning literature, and they both bring advantages and disadvantages. Thus, from this limited perspective, our approach can be considered as an RRT extension of the approach of [28], which is also important for the robotics community. However, there is another significant difference between our study and that done by [28]. In their paper, they consider only robotic systems that are fully holonomic, and thus, they use navigation functions [8] for their feedback control policy inside each region. However, the ROV model that we adopt in this study is a non-holonomic model and we use a Lyapunov-based feedback control policy for navigation.

2.2 Problem Definition and Algorithm

The aim of our new motion planning algorithm is to allow a UUV in a given environment to navigate from any initial state (position and orientation) to a desired goal position. In our solution, we prioritize robustness and sparsity such that, instead of relying on open-loop trajectories, we develop a feedback-based method that covers large regions inside the space.

The basic idea in our method is to generate random sequentially attached sparse funnels, where the probabilistic tree generation idea is borrowed from the original RRT. Similar to [27], we grow the tree starting from the goal state, since our aim is to find a set of funnels that will eventually drive covered states to the goal position.

In classical sampling-based methods, the first step is drawing a sample from the state space to start growing the tree or roadmap. Our goal, however, is finding funnels, and thus we start by constructing a funnel for which the outlet is the goal position. In other words, the goal position is the ultimate equilibrium of the dynamical system structure that we want to construct. The fundamental approach in the methods deployed by [27], [28] and [12] is to first generate a control policy and then try to generate a funnel (a basin of attraction) using optimization techniques. The numerical and optimization steps of these other methods make them computationally less feasible than our approach. In contrast, we design a region of attraction first and later define a set of possible control policies that guarantee the asymptotical stability of the

equilibrium (under some assumptions). Considering the typical 2D environments for ROV applications, we use circles as our funnel inlets (or the region of attraction). Assuming a circular funnel provides us two main advantages: relative ease of finding a set of controllers with guaranteed stability inside the funnel (i.e., compared to funnels with polygonal shapes) and computational feasibility of finding collision-free regions (compared to, for example, funnels with ellipsoid shapes).

We name the first generated funnel the *master* funnel. The creation of the master funnel is illustrated in Figure 2.1(b) for a sample environment. After that, we start the random funnel generation process by sampling a data point inside the Cartesian space. Note that unlike similar methods in [27], we do not sample from the whole state space. Instead, we reduce the sampling space by omitting the angular coordinates, since our control policies (See Section 2.3) guarantee convergence for all initial angular positions. If the random sample is located inside the basin of attraction of one of the funnels then we omit the sample, which is one of the main differences from the classical RRT approaches. We then find the closest funnel (which minimizes the distance between its boundary and the sample) and draw a line between the outlet (center) of this funnel and the sample point. We create a new point along this line such that the distance between the new point and funnel outlet is equal to $d = \eta r_f$, where r_f is the radius of the funnel, and $\eta \in (0, 1)$ is a predefined threshold. We generally prefer values between $[0.8, 0.9]$. In other words, we locate the new point inside the funnel but close to the boundary to increase the sparsity of the funnel tree. This process is illustrated in Figure 2.1(c & d). We then generate a new collision-free funnel by using this point as its center/outlet. This new funnel is added to our tree structure with a proper connection to the connected funnel. It is obvious that similar to RRT the initial sample indeed controls the growth direction and size of the funnels (which depends on the sparsity of the region) defines the growth rate. Thus, inside sparse regions this method greatly reduces the density compared to RRT.

We repeat this sampling and funnel tree generation process until some user-defined stopping condition is triggered. In the simulation results that we present in this study, we stop the process when the new funnel covers the initial condition. It is also possible to continue the process until the whole region of interest is probabilistically covered. Algorithm 2 details the algorithmic steps of our method, and Figure 2.1

Algorithm 2: Random Sparse Funnel Tree Generation

$\mathcal{G} \leftarrow$ INITIALIZE_TREE();
$\mathcal{G} \leftarrow$ INSERT_NODE(q_{goal}, R_{attr});
for $k = 1$ **to** K **do**
    $q_{rand} \leftarrow$ RAND_CONF();
    $q_{nearest} \leftarrow$ NEAREST_VERTEX($\mathcal{G}, q_{rand}$);
    if $q_{rand} \notin \mathcal{B}$ **then**
        $q_{new} \leftarrow$ EXTEND($q_{nearest}, q_{rand}, \alpha$);
        $r_{new} \leftarrow$ COMPUTE_ATTRACTION_REGION(q_{new});
        $\mathcal{G} \leftarrow$ INSERT_NODE(q_{new}, R_{new});
        if $q_{init} \in \mathcal{B}$ **then**
            return $\mathcal{G}$
return G

illustrates the algorithmic steps for a sample environment.

2.3 Control Policy and Stability Analysis

In this section, we propose our velocity-based control policy for planar (2D) environments. The goal of a control policy in the motion planning algorithm is to produce a set of velocity commands. This approach relies on a key assumption that first-order dynamics of the ROV/AUV and environment dominate the system behavior. This assumption is reasonable under several conditions: high damping generated by an underwater environment, slow operating speeds, and a high-gain low-level velocity controller. Note that this assumption depends on a low-level velocity controller, which we utilize in the 3D simulation environment (see Chapter 4).

These conditions are also satisfied by many of the commercial ROV/AUV applications [30]. Here, we also concentrate on UUV systems in which the vehicle is not capable of a lateral direction thrust force. Clearly adding like these degrees of freedom (DOF) would simplify the control problem. Under these assumptions and constraints, we can simplify and reduce the 2D dynamics to a first-order unicycle model,

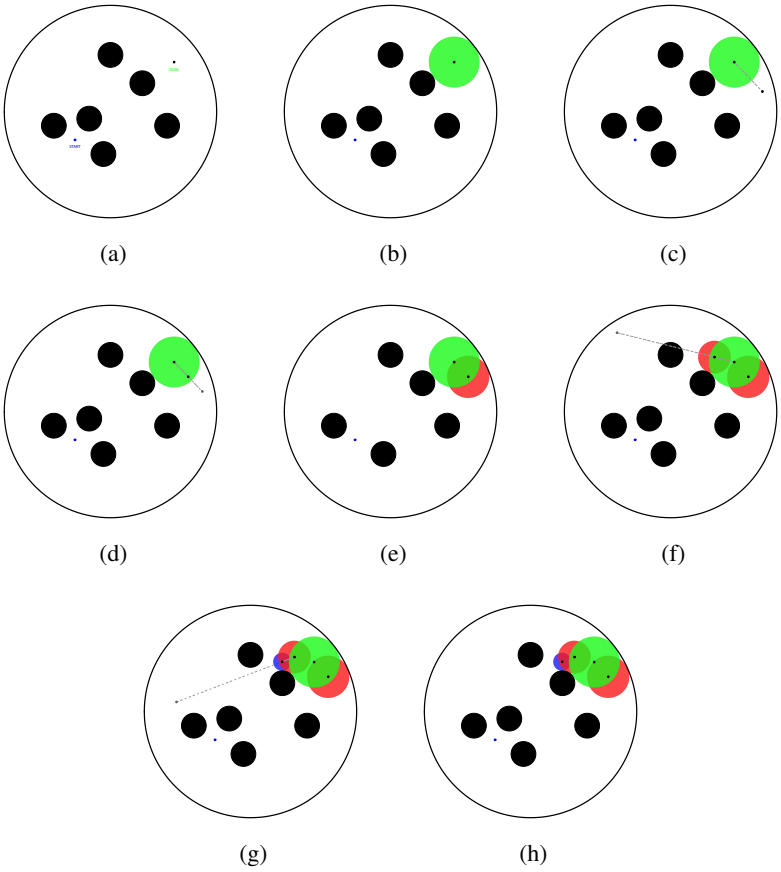


Figure 2.1: Illustration of proposed method for a sample environment. **a)** Initial configuration. Black solid circles, and the outer hollow circle denote the obstacles and boundary, respectively. **b)** An obstacle-free *funnel* (green) is created, centered on the *GOAL* node. **c)** A new sample is drawn from the 2D Cartesian space. **d)** The new sample is projected into the nearest funnel. **e)** The basin of attraction of the new node is computed, and this node is added to the tree. **f)** Steps **c**, **d**, and **e** are repeated to create the new funnel shown in red. Again, steps **c**, **d**, and **e** are repeated to create a new funnel. Note that both this funnel and previous funnel are connected to the master funnel, thus they are at the same level of the tree. **g)** Steps **c**, **d**, and **e** are repeated again to create a new funnel shown in blue. Note that this funnel is connected to one of the red funnels, thus it is located at a lower level of the tree. **h)** Overall view of the *funnel* tree. The green (master) funnel has the highest priority, the red funnels are at the second level of the priority queue, and the blue funnel has the lowest priority.

illustrated in Figure 2.2, where the inputs to the 2D ROV/AUV model are the forward speed, v, and angular speed, ω, of the model.

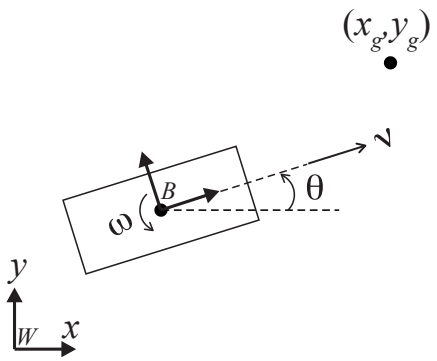


Figure 2.2: Illustration of the 2D unicycle model

The simplified dynamics of the unicycle with respect to the world-fixed reference frame takes the form

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} \tag{2.1}$$

For each *funnel* (or region), the goal of the control policy is this: from any initial condition inside the funnel, the trajectory will converge to the equilibrium of that funnel and the trajectories will strictly stay inside the basins of attraction. The second condition is necessary to avoid collisions, since the funnels define safe (collision-free) regions. Note that the control policy will directly stabilize only the Cartesian position not the angle at the equilibrium point.

Since we do not directly control the heading angle at the equilibrium, this generates a dynamical *symmetry* and thus if we rewrite the dynamics w.r.t to the instantaneous body-fixed reference frame (using polar coordinates), we can reduce the number of states to two, which is helpful for presenting and proving the stability of our simple control policy. The illustration of the model in polar coordinates with respect to the body-fixed reference frame can be seen in Figure 2.3.

In the model, ρ represents the distance between the body center and the outlet (equi-

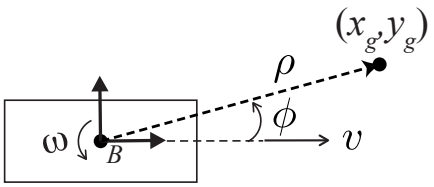


Figure 2.3: Unicycle model and coordinates with respect to body-fixed reference frame.

librium) of the funnel, and ϕ represents the angle between the ROV heading and the vector from the body center to the goal point. In polar coordinates, the target control policy is simply finding a controller that drives ρ and θ to zero as well as $\dot{\rho} < 0$ in order to strictly stay inside funnel. Before we pursue the details of the control policy, we first write the dynamics with respect to state variables ρ and ϕ as

$$\begin{bmatrix} \dot{\rho} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} v\cos\phi & 0 \\ \frac{\sin\phi}{\rho} & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{2.2}$$

Our simple non-linear control policy (for all funnels) is as follows:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} K_v \rho \cos\phi \\ -K_\phi \phi \end{bmatrix} \tag{2.3}$$

Before presenting a stability proof, we must understand the function of the control policy. Angular speed is simply given by a linear proportional control law to minimize the angular heading error. On the other hand, the forward velocity controller uses a non-linear control policy that depends on ρ and $\cos\phi$. We first analyze the extreme cases of $\cos\phi$. When $\phi = 0$, $v = rK_v$, which means that the ROV/AUV heading is perfectly aligned with the target, so it moves at its maximum speed. When $\phi = \pm\pi/2$, $v = 0$, which means that the ROV/AUV moves neither forward nor backward. This is an important behavior, since in this case any kind of forward or backward motion would increase the radius, allowing the ROV to leave the region if it is close to the boundary. However, since there is an error between heading angle and target angle, the angular speed controller will apply an input for correction, thus as soon as the ROV leaves this singular point, it will again move toward the goal point. When $\phi = \pm\pi$, $v = -rK_v$, which means that the angle difference between the ROV heading

and target is 180^o. In this case, and in general $\forall\phi, \cos\phi < 0$, the ROV/AUV will move to the target by going backwards. However as soon as the ϕ drops to the interval of $\pi/2 < \phi < \pi/2$, the vehicle will move only in a forward direction. In practice, backward motion is very limited (but necessary).

Next, we prove the stability of the controller using a Lyapunov-like stability analysis. Let the candidate Lyapunov function be

$$V(\rho, \phi) = \alpha\rho^2 + \phi^2,\ \alpha > 0 \tag{2.4}$$

$$V(\rho, \phi) = 0 \iff (\rho, \phi) = (0, 0) \tag{2.5}$$

The derivative of the Lyapunov function then takes the form

$$\dot{V} = 2\alpha\rho\dot{\rho} + 2\phi\dot{\phi} \tag{2.6}$$

$$= -2\alpha K_v\rho^2(\cos\phi)^2 - 2K_\phi\phi^2 + K_v\phi\sin(2\phi) \tag{2.7}$$

It is easy to show that

$$\frac{K_\phi}{K_v} > 1 \implies \dot{V} < 0, \forall(\rho, \phi) \neq 0 \tag{2.8}$$

In conclusion, if we select a pair of K_v and K_ϕ values such that $\frac{K_\phi}{K_v} > 1$, then we guarantee the stability of each funnel using Lyapunov's second method (note that this stability analysis is potentially conservative). This is an interesting and yet intuitive result. This stability condition simply states that the controller should prioritize correcting the errors in relative angle between the robot and the target rather than the relative distance. If we combine the dynamics in (2.2) and the controller in (2.3) and concentrate on only the dynamics in the ϕ direction we obtain

$$\dot{\phi} = K_V\frac{\sin(2\phi)}{2} - K_\phi\phi \tag{2.9}$$

When $\phi \in (-\pi/2, -pi/2)$, K_ϕ has a stabilizing effect, whereas K_V has a destabilizing effect on the angular dynamics. For this reason, one can easily deduce that if K_{phi} is sufficiently larger than K_V s.t. $K_\phi\phi > K_V\frac{\sin(2\phi)}{2}$, $\forall\phi \in (-\pi/2, -pi/2)$, then the dynamics in angular direction become stable. Indeed, this condition has already been found based on Lyapunov analysis.

Since the outlet of each funnel is connected to a parent funnel and the master funnel drives the initial conditions to the goal position, we can say that each initial condition

that is inside at least one funnel is guaranteed to converge to the goal state asymptotically.

2.4 Sparsity Enhancement

In this section, sparsity enhancement methods are described.

One of the powerful aspects of the proposed motion planning algorithm is its sparsity compared with other random sampling methods such as RRT. Sparse methods are more computationally efficient than dense methods, which enables the use of such algorithms in real-life applications. In the developed algorithm, the nodes are represented as circles, as shown in Figure 2.1. In the algorithm to append a new node to the tree, the radius of the new node is enlarged until it hits an obstacle or boundary. Examples can be seen in Figure 2.1(e) and Figure 2.1(f).

The main disadvantage of this method is that we stop increasing the radius of the circle when it hits a boundary or an obstacle. However, it is still possible to grow the circle in the direction opposite the collision. In this way, it is possible to obtain nodes with a greater radius, which means they cover larger areas, and we can cover the area with a smaller number of nodes.

Two methods are described for enlarging the covering area of a new node. In the first method, the circles are enlarged using an elliptical shape. In the second method, the nodes are enlarged keeping the shape as circular but increasing the radius.

2.4.1 Elliptical Nodes

In the standard algorithm, the enlargement of a circle stops when the boundary of a new node collides with either the boundary of a region or an obstacle. It is possible to enlarge the circular funnel without changing the place of the outlet point, which guarantees that the outlet of the new funnel remains inside the previous funnel.

In this algorithm, we first find a circular node as described in Algorithm 2. After finding the new node, we enlarge it in parallel with the tangent that passes through the

intersection of the obstacle and the node. The original node and elliptically enlarged node are given in Figure 2.4.

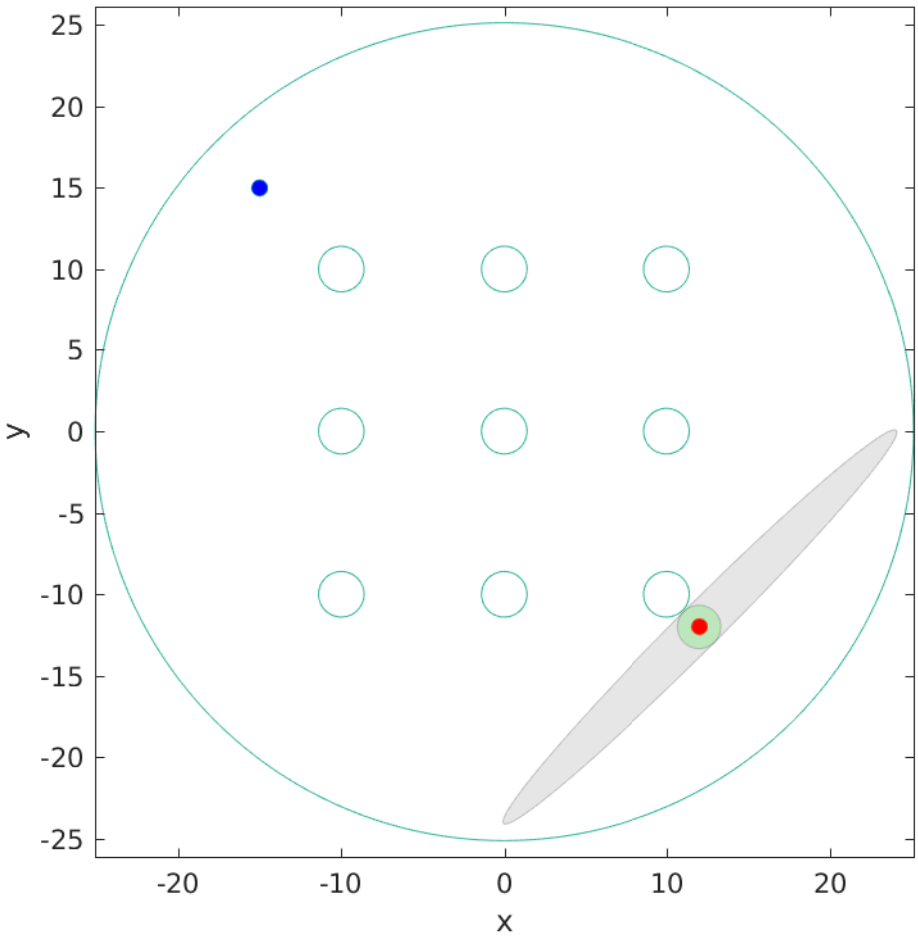


Figure 2.4: Illustration of elliptical enlargement. The green circle denotes the node generated with the original algorithm and the gray ellipse denotes the elliptical enlarged node.

2.4.2 Enlarged Circular Nodes

In this method, basically, the new node is appended to ensure that it covers the largest area and that the outlet of the funnel is still inside the parent funnel. The method is

illustrated in Figure 2.5.

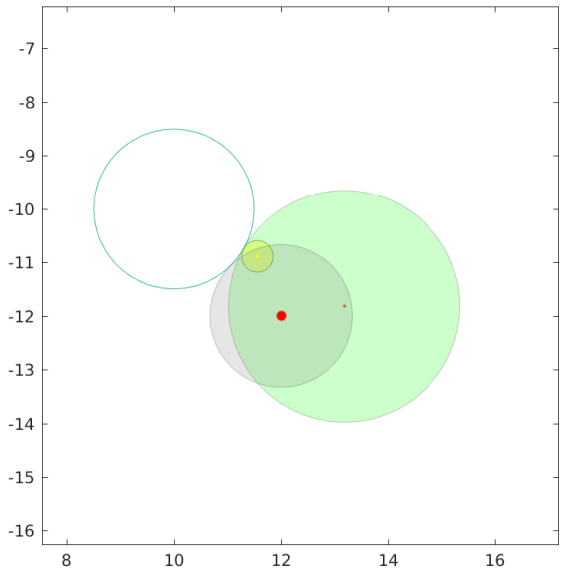


Figure 2.5: Illustration of circular enlargement. The yellow circle denotes the node generated with the original algorithm and the green circle denotes the circular enlarged node. Note that the origin of the new node is also changed but it is still in the previous connected funnel which is the gray one in the figure.

2.5 Extension of the Feedback Motion Planning Method to 3D

In this section, we extend the proposed algorithm to the more realistic and practically important 3D case. The class of underwater vehicles that we consider in this study are assumed to have independent vertical thrusters [30]; thus we omit the class of UUVs that have a single thruster at the back and in which the direction of thrust force is controlled via the regulation of fins. Note that our main motion planning approach can also be extended to this class, which is among our goals in the near future. Indeed the fundamental algorithm that is proposed in Algorithm 2 is directly applicable to the 3D case and also to different types of robotic systems, including the class of UUVs described in the previous paragraph. The main differences between 3D and 2D methods are that the shape of funnels, distance function, method for extending random samples, and control policy are adjusted to take into account the 3D kinematics and

dynamics.

In 3D case, the basins of attraction of the funnels should be of volumetric shape (since we sample only from Cartesian coordinates). One obvious candidate shape is the sphere, which would certainly produce effective results. However, one should note the fact that z-axis of the ROV/AUV can be controlled independently. Thus, if one uses cylinders instead of spheres as the basins of attraction, then it is possible to cover larger areas and thus boost the sparsity.

For a given 3D environment with obstacles and boundaries, the generation of a funnel around a selected equilibrium point is illustrated in Figure 2.6. We begin the process (Figure 2.6(b)) by generating an obstacle-free 2D disk in the X–Y plane that passes through the selected equilibrium point. Then we vertically extrude this disk in both directions until the cylinder hits an obstacle or constraint. This 3D cylinder forms the basin of attraction of the funnel. Note that the equilibrium point is not at the center of the volume in the z-direction. Indeed, due to independent control of the z-axis, we can even change the z position of the equilibrium point, which we will use for height smoothing described in the following steps.

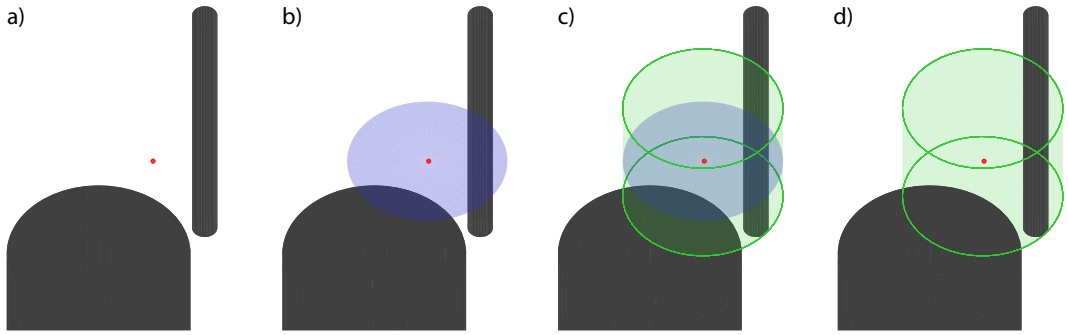


Figure 2.6: Illustration of funnel generation in 3D. **a)** Example 3D environment where black solid cylinders and red dot illustrate the obstacles and the funnel outlet (equilibrium), respectively. **b)** An obstacle-free 2D disk (blue) is generated around the funnel outlet. **c)** The 2D disk is extruded vertically in both directions to form an obstacle-free cylinder (green), which corresponds to the basins of attraction of the funnel. At the bottom, an obstacle defines the lower limit, whereas from the top, the sea level is the limit. **d)** The blue disk is removed to better illustrate the funnel's basins of attraction.

We start the method for 3D environments by generating a funnel around the goal position of the UUV. After that we draw a sample from 3D Cartesian space, similar to the 2D case. If the drawn sample is inside one of the funnels then we discard the sample. Otherwise, we find the closest funnel by computing the minimum Euclidean distance between the sample point and cylinder.

We perform this operation by projecting the point to the funnel boundary. Based on the location of the point there are three possibilities for this projection: onto the surface of the cylinder, onto the surface of one of the circles, or onto the perimeter of one of the circles.

The next step is finding a point inside the target funnel that will be the outlet of this new funnel candidate. To find such a point, we draw a line from the projection to the funnel equilibrium and then find the projection inside the funnel along this line such that the distance to the equilibrium will be $d = \eta||p||_2$, where $||p||_2$ is the distance of the original projection to the equilibrium and $\eta \in (0, 1)$ is a predefined threshold. Once we locate the new outlet (equilibrium), we construct the basins of attraction of this new node using the procedure above and this node is then connected to the funnel tree.

We perform one more step after this operation, which we call height "smoothing". We observed that if we do not perform such a smoothing, there can be unnecessary oscillations in the z-axis of the ROV/AUV. Indeed, for the class of UUVs that we consider, it is generally less desirable to move in the z-direction frequently, because of practical problems such as higher drag and less thrust force. In this smoothing operation, we take advantage of the fact that we can change the z position of the funnel equilibrium point even after we have generated basins of attraction. Basically, if the z-axis coordinate of the equilibrium of the *connected* funnel is inside the z-axis boundaries of the new funnel, then we equalize the z-axis coordinates of the connected funnels by moving the new equilibrium.

As in the 2D case, this combined sampling, projection, funnel generation and height smoothing process is repeated until the one of the funnels includes the initial condition, or the union of the funnels sufficiently covers the regions of interest.

2.5.1 Control Policy in 3D

From the perspective of the motion controller, we rely on same assumptions used with the 2D case. The only difference is the addition of a vertical degree of freedom. In this extended model of the UUV, the inputs will be the forward speed, v_p, angular speed, ω, and vertical speed, v_v. Under these inputs, the simplified 3D dynamics of the model with respect to the world-fixed reference frame take the form

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v_p \cos\theta \\ v_p \sin\theta \\ v_v \\ \omega \end{bmatrix}$$

Again, by taking into account the dynamical *symmetry* in the X–Y plane, we can rewrite the dynamics w.r.t. the instantaneous body-fixed reference frame using cylindrical coordinates:

$$\begin{bmatrix} \dot{\rho} \\ \dot{\phi} \\ \dot{h} \end{bmatrix} = \begin{bmatrix} v\cos\phi & 0 & 0 \\ \frac{\sin\phi}{\rho} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_p \\ \omega \\ v_v \end{bmatrix}$$

In the 3D model, ρ represents the distance between the body center and the outlet (equilibrium) of the funnel in the X–Y plane, ϕ represents the angle between the ROV heading and the X–Y projection of the vector from the body center to the goal point, and h is the vertical position with respect to the funnel outlet. The extended control policy is simply

$$\begin{bmatrix} v_p \\ \omega \\ v_v \end{bmatrix} = \begin{bmatrix} K_v \rho \cos\phi \\ -K_\phi \phi \\ -K_h \end{bmatrix}$$

Since the z-axis dynamics and control are independent from other coordinates, to have a stable complete system we need to make sure of the stability in this coordinate.

Simple scalar linear dynamics make it easy to see that if $K_h > 0$ and we satisfy the conditions for the 2D case, the simplified model is guaranteed to strictly stay inside the funnel and asymptotically converge to the funnel outlet.

CHAPTER 3

MATHEMATICAL MODELING OF SAGA ROV

3.1 Introduction

To develop our motion planning algorithm, we made some claims regarding the dynamics of the vehicle. We assumed a velocity-based kinematic model and motion controller that produced a set of velocity commands that were assumed to be controlled independently and reactively. However, even if our assumptions were reasonable, it is important to extend and test the methods to handle full (second-order) dynamic models and systems. We derive the model of our own SAGA ROV,[1] since its motion capabilities and constraints closely reflect the class of UUVs that we want to control.

When developing a controller for any system, constructing a mathematical model of the system must be the first step. The controller algorithms may be tested on the simulator using the model and the performance of the controller can be easily assessed. The simulation approach is typically cheaper and easier. The robustness of the controller can be readily tested using simulation that takes disturbances into account. However, the most critical part of simulating any system is obtaining an accurate mathematical model of the system being controlled.

The main difficulty in mathematical modeling is obtaining a realistic model. There are many components in ROV modeling. To model all of the forces affecting a vehicle underwater is very difficult and may not actually be possible, which means some assumptions should be made to simplify the modeling.

[1] www.desistek.com

3.2 Mathematical Modeling of SAGA ROV

Basically, the SAGA ROV has a differential drive thruster structure in the X–Y plane that can be used to control the angular velocity and forward speed. SAGA also has a single vertical thruster that is controlled to regulate vertical speed. The simulation environment utilizes the full 6DOF non-linear dynamics of the vehicle. The mathematical model based on parametric system identification using experimental data is presented in [31] and [32]. The simulation package [33] adopts this model and its parameters inside its physics engine.

An accurate dynamical model is required for obtaining realistic results. This is difficult, however, for underwater vehicles which include many modeling uncertainties and nonlinearities. During the modeling process, some reasonable assumptions must be made for the sake of simplicity while not moving far from the vehicle's true behavior.

In this study, the underwater vehicle SAGA, which is an inspection class ROV manufactured by Desistek (Figure 3.1) is used. The dimensions of this vehicle are 420 mm $\times$ 330 mm $\times$ 270 mm. The mass of the vehicle is 10 kg, and maximum depth is 250 m. Maximum surge speed is 3 knots and maximum heave speed is 1 knot. The vehicle has three thrusters, which supply a maximum force of 10 N. One of these thrusters is directed vertically, while the other two are located at the right and left sides and are directed horizontally.

The dynamics of the vehicle can be obtained in two stages: the *kinematic model* and the *dynamic model* [34]. The *kinematic model* considers the geometrical aspects of the motion, while the *dynamic model* analyzes the forces that cause the motion. In this study, the ROV is assumed to be a rigid body, meaning that while operating under water, no deformation occurs in the vehicle. The following sections present the *kinematic* and *dynamic* modeling of the ROV.

Figure 3.1: SAGA ROV vehicle manufactured by Desistek Robotics.

3.3 Kinematic Modeling

3.3.1 Reference Frames

Two reference frames are used in this study, as proposed in [35]: the north-east-down (NED) and the body-fixed coordinate frames. The definitions are given as follows.
NED: The earth-fixed frame is denoted as North-East-Down and usually defined as the tangent plane on the surface of the Earth. The origin is positioned at the sea surface, possibly the position of the surface vehicle and the x-axis points *north*, the y-axis points *east* and the z-axis points *down*. This coordinate system is denoted as $\{n\} = \{x_n, y_n, z_n\}$ with the origin o_n (Figure 3.2).

BODY: The body-fixed reference frame is a moving coordinate frame that is fixed to the ROV. The x-axis is the longitudinal axis directed from aft to fore, y-axis is the transversal axis directed to starboard and z-axis is the normal axis directed from top to bottom. This coordinate system is denoted as $\{b\} = \{x_b, y_b, z_b\}$ with the origin o_b (Figure 3.3). The notation used in the modeling is given in Table 3.1. Using the notation in [35], the kinematics can be described as:

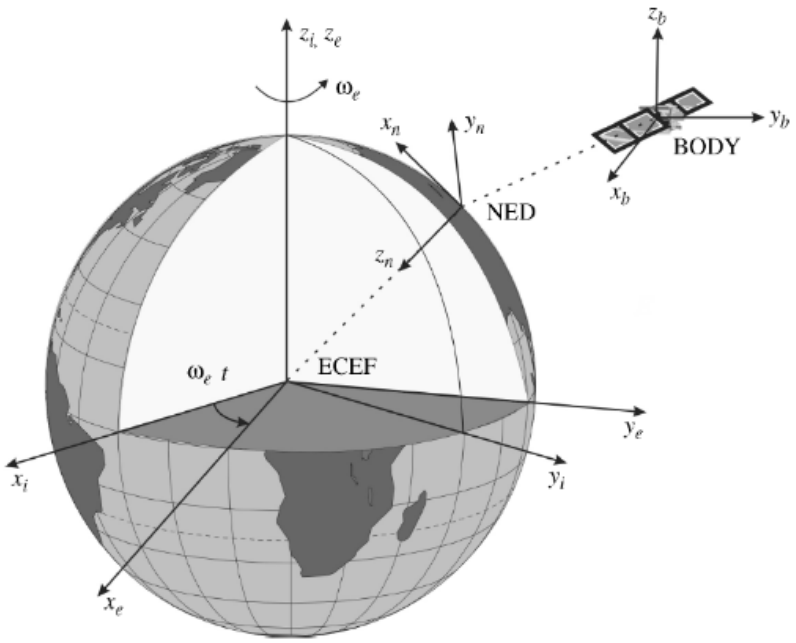


Figure 3.2: NED reference frame.

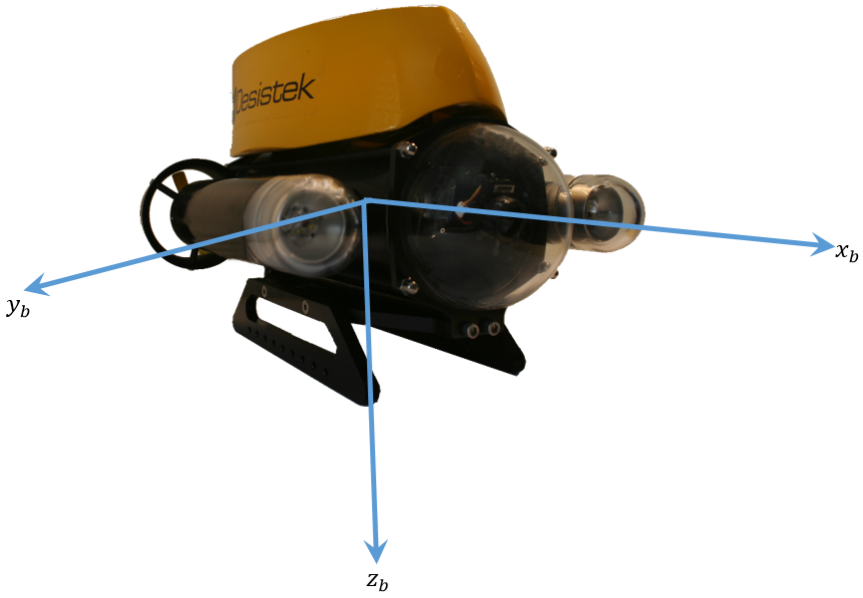


Figure 3.3: Body-fixed reference frame.

Table 3.1: The notation of SNAME (1950) for marine vessels.

DOF		**Forces and moments**	**Linear and angular velocities**	**Positions and Euler angles**
1	Motions in the x direction(surge)	X	u	x
2	Motions in the y direction(sway)	Y	v	y
3	Motions in the z direction(heave)	Z	w	x
4	Rotations in the x axis(roll)	K	p	ϕ
5	Rotations in the y axis(pitch)	M	q	θ
6	Rotations in the z axis(yaw)	N	r	ψ

NED Position	$P^n_{b/n} = [x, y, z]^T \in \mathbb{R}^3$
Attitude (Euler Angles)	$\Theta_{bn} = [\phi, \theta, \psi]^T \in S^3$
Body-fixed linear velocity	$v^b_{b/n} = [u, v, w]^T \in \mathbb{R}^3$
Body-fixed angular velocity	$\omega^b_{b/n} = [p, q, r]^T \in \mathbb{R}^3$
Forces and moments	$\tau = \begin{bmatrix} X & Y & Z & K & M & N \end{bmatrix}^T \in \mathbb{R}^6$

where $\mathbb{R}^3$ is the Euclidean space of dimension three and S^3 denotes a sphere defined by three angles on the interval $[0, 2\pi]$. The general motion for a marine craft in 6-DOF is described by the following vectors [36]:

$$\eta = \begin{bmatrix} P^n_{b/n} \\ \Theta_{bn} \end{bmatrix}, \quad v = \begin{bmatrix} v^b_{b/n} \\ \omega^b_{b/n} \end{bmatrix} \tag{3.1}$$

where:

$P^n_{b/n}$: Position of body frame with respect to NED frame expressed in NED frame

$v^b_{b/n}$: Linear velocity of body frame with respect to NED-frame expressed in body frame

τ: Forces and moments acting on the craft in the body-fixed frame

The kinematic relation between the body-fixed velocity v and the position η in the NED frame is expressed as:

$$\dot{\eta} = J(\Theta_{nb})v \tag{3.2}$$

where $J(\Theta_{nb}) \in \mathbb{R}^{6x6}$ is the transformation matrix given as:

$$J\left(\Theta_{nb}\right) = \begin{bmatrix} R\left(\Theta_{nb}\right) & 0_{3x3} \\ 0_{3x3} & T\left(\Theta_{nb}\right) \end{bmatrix} \tag{3.3}$$

The matrices $R\left(\Theta_{nb}\right)$ and $T\left(\Theta_{nb}\right)$ are given as:

$$R\left(\Theta_{nb}\right) = \begin{bmatrix} c\psi c\theta & -s\psi c\phi + c\psi s\theta s\phi & s\psi s\phi + c\psi c\phi s\theta \\ s\psi c\theta & c\psi c\phi + s\phi s\theta s\psi & -c\psi s\phi + s\theta s\psi c\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \tag{3.4}$$

$$T\left(\Theta_{nb}\right) = \begin{bmatrix} 1 & s\phi t\theta & c\phi t\theta \\ 0 & c\phi & -s\phi \\ 0 & s\phi/c\theta & c\phi/c\theta \end{bmatrix}, \quad \theta \neq \frac{\pi}{2} \tag{3.5}$$

where $s(.) = \sin(.)$, $c(.) = \cos(.)$, $t(.) = \tan(.)$.

3.4 Dynamic Modeling

In this section, the forces and moments caused by the rigid-body inertia and the hydrodynamics effects due to the movement of the ROV under water are modeled. First, rigid-body dynamics are derived

3.4.1 Rigid-Body Dynamics

In this section, the rigid-body dynamics of a marine vehicle are given as derived in [35]. The derivation is based on Newton's second law, which specifies the force acting on a system is equal to the product of the mass and acceleration of the system: $\sum F = ma$. Euler's axioms state that this is also valid for angular momentum [35, 37]. Using Newton's laws and Euler's axioms, the rigid-body equations of motions around an

arbitrary origin (CO) can be expressed as follows:
Translational:

$$m\left[\dot{v}^b_{b/n} + \dot{\omega}^b_{b/n} \times r^b_g + \omega^b_{b/n} \times v^b_{b/n} + \omega^b_{b/n} \times \left(\omega^b_{b/n} \times r^b_g\right)\right] = f^b_b \tag{3.6}$$

Rotational:

$$I_b\dot{\omega}^b_{b/n} + \omega^b_{b/n} \times I_b\omega^b_{b/n} + mr^b_g \times \left(\dot{v}^b_{b/n} + \omega^b_{b/n} \times v^b_{b/n}\right) = m^b_b \tag{3.7}$$

The equations 3.6 and 3.7 are usually written in component form according to SNAME (1950) notation by defining the following:

$f^b_b = [X, Y, Z]^T$	Force through o_b in $\{b\}$ (body-fixed coordinate system)
$m^b_b = [K, M, N]^T$	Moment about o_b in $\{b\}$
$v^b_{b/n} = [u, v, w]^T$	Linear velocity o_b relative to o_n expressed in$\{b\}$
$\omega^b_{b/n} = [p, q, r]^T$	Angular velocity of b relative to n (NED coordinate system) expressed in $\{b\}$
$r^b_g = [x_g, y_g, z_g]^T$	Vector from o_b to CG expressed in $\{b\}$

Applying this notation, 3.6 and 3.7 become

$$\begin{aligned}
m\left[\dot{u} - vr + wq - x_g(q^2 + r^2) + y_g(pq - \dot{r}) + z_g(pr + \dot{q})\right] &= X \\
m\left[\dot{v} - wp + ur - y_g(r^2 + p^2) + z_g(qr - \dot{p}) + x_g(qp + \dot{r})\right] &= Y \\
m\left[\dot{w} - uq + vp - z_g(p^2 + q^2) + x_g(rp - \dot{q}) + y_g(rq + \dot{p})\right] &= Z \\
I_x\dot{p} + (I_z - I_y)qr - (\dot{r} + pq)I_{xz} + (r^2 - q^2)I_{yz} + (pr - \dot{q})I_{xy} + \\
m\left[y_g(\dot{w} - uq + vp) - z_g(\dot{v} - wp + ur)\right] &= K \\
I_y\dot{q} + (I_x - I_z)rp - (\dot{p} + qr)I_{xy} + (p^2 - r^2)I_{zx} + (qp - \dot{r})I_{yz} + \\
m\left[z_g(\dot{u} - vr + wq) - x_g(\dot{w} - uq + vp)\right] &= M \\
I_z\dot{r} + (I_y - I_x)pq - (\dot{q} + rp)I_{yz} + (q^2 - p^2)I_{xy} + (rp - \dot{p})I_{zx} + \\
m\left[x_g(\dot{v} - wp + ur) - y_g(\dot{u} - vr + wq)\right] &= N
\end{aligned} \tag{3.8}$$

The first three equations represent translational motion, while the last three represent rotational motion. The rigid-body kinetics can be expressed in vectorial form as follows:

$$M_{RB}\dot{v} + C_{RB}(v)v = \tau_{RB} \tag{3.9}$$

where $v = [u, v, w, p, q, r]^T$ is the generalized velocity vector expressed in $\{b\}$ (body-fixed coordinate system), and $\tau_{RB} = [X, Y, Z, K, M, N]^T$ is a generalized vector of

external forces and moments. The matrix M_{RB} is the rigid-body mass matrix and C_{RB} is the rigid-body Coriolis and centripetal matrix due to the rotation of the body frame about the inertial frame. These matrices contain coefficients that can be determined based on gravity and the inertial properties of the vehicle. The representation of M_{RB} is unique and satisfies

$$M_{RB} = M_{RB}^T > 0, \quad \dot{M}_{RB} = 0_{6x6} \tag{3.10}$$

where

$$\begin{aligned} M_{RB} &= \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \\ &= \begin{bmatrix} m & 0 & 0 & 0 & mz_g & -my_g \\ 0 & m & 0 & -mz_g & 0 & mx_g \\ 0 & 0 & m & my_g & mx_g & 0 \\ 0 & -mz_g & my_g & I_x & -I_{xy} & -I_{xx} \\ mz_g & 0 & -mx_g & -I_{yx} & I_y & -I_{yz} \\ -my_g & mx_g & 0 & -I_{zx} & -I_{zy} & I_z \end{bmatrix} \end{aligned} \tag{3.11}$$

Here $r_g^b := [x_g, y_g, z_g]^T$ is the position of the CG with respect to the body-fixed frame origin. According to [34], the Coriolis and centripetal matrix can be parameterized such that $C_{RB}(v) = -C_{RB}(v)$ by choosing

$$C_{RB}(v) = \begin{bmatrix} 0_{3\times3} & -S\left(M_{11}v_1 + M_{12}v_2\right) \\ -S\left(M_{11}v_1 + M_{12}v_2\right) & -S\left(M_{21}v_1 + M_{22}v_2\right) \end{bmatrix} \tag{3.12}$$

where $v_1 = v_{b/n}^b, v_2 = \omega_{b/n}^b$ and S is the skew symmetric matrix

$$S([a,b,c]) = \begin{bmatrix} 0 & -c & b \\ c & 0 & -a \\ -b & a & 0 \end{bmatrix} \tag{3.13}$$

In the current study, the body-fixed frame is chosen to be coincident with the three principal axes of the SAGA (Figure 3.1). As a result of this property, $I_{xz} = I_{yz} =$

$I_{xy} = 0$. For SAGA, the M_{RB} and C_{RB} matrices are defined as follows:

$$M_{RB} = \begin{bmatrix} m & 0 & 0 & 0 & mz_g & -my_g \\ 0 & m & 0 & -mz_g & 0 & mx_g \\ 0 & 0 & m & my_g & -mx_g & 0 \\ 0 & -mz_g & my_g & I_{xx} & 0 & 0 \\ mz_g & 0 & -mx_g & 0 & I_{yy} & 0 \\ -my_g & mx_g & 0 & 0 & 0 & I_{zz} \end{bmatrix} \tag{3.14}$$

$$C_{RB}(v) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ -m(y_g q + z_g r) & m(y_g p + w) & m(z_g p - v) \\ m(x_g q - w) & -m(z_g r + x_g p) & -m(z_g q + u) \\ m(x_g r + v) & m(y_g r - u) & -m(x_g p + y_g q) \end{bmatrix} \begin{bmatrix} m(y_g q + z_q r) & -m(x_g p - w) & -m(x_g r + v) \\ -m(x_g q + w) & m(z_g r + x_g p) & -m(y_g r - u) \\ -m(x_g r - v) & -m(z_g q + u) & m(x_g p + y_g q) \\ 0 & I_{zz} q & -I_{yy} p \\ -I_{zz} q & 0 & I_{xx} p \\ I_{yy} q & -I_{xx} p & 0 \end{bmatrix} \tag{3.15}$$

3.4.2 Hydrodynamic Forces and Moments

The hydrodynamic forces acting on the vehicle can be listed as follows [38]:

- Added mass is a concept of pressure-induced forces and moments due to a forced harmonic motion of the body. These are proportional to the acceleration of the body.
- Hydrodynamic damping due to effects such as skin friction, vortex shedding and energy carried away by generated surface waves. The hydrodynamic damping forces usually consist of linear and quadratic damping terms. These are in a direction opposite to the vehicle's velocity.

- Restoring forces due to the vehicle's weight and buoyancy. Restoring forces consist of gravity and buoyancy forces.
- Currents.
- Thruster/propeller forces.
- Control surface/rudder forces.

All of these effects are included in Fossen's equations of motion for marine crafts as described in equation 8.151 in [35]:

$$M\dot{v} + C(v)v + D(v)v + g(\eta) + g_0\left(v_r\right)v_r = \tau_{wind} + \tau_{wave} + \tau_{RB} \tag{3.16}$$

where

$M = M_{RB} + M_A$ - system inertia matrix (including added mass)
$C(v) = C_{RB}(v) + C_A(v)$ - Coriolis-centripetal matrix (including added mass)
$D(v)$ - damping matrix
$g(\eta)$ - vector of gravitational/buoyancy forces and moments
g_o - vector used for pertrimming (ballast control)
τ - vector of control inputs
τ_{wind} - vector of wind forces
τ_{wave} - vector of wave-induced forces

τ_{wind} and τ_{wave} are forces and torques due to wind and waves, which we disregard and assume to be 0, since we focus solely on the underwater case.

3.4.3 SAGA Mathematical Model

The mathematical model parameters for the SAGA vehicle are obtained from [31], [39], and [32]. In these studies, the system identification is performed using two methods: acoustic-based and vision-based. These methods are used to find the position of the vehicle. In future studies, it may possible to use Ultra Short Baseline Tracking System (USBL) to estimate the position of the vehicle more accurately. In many ROV applications, the vehicle usually moves at low speeds. If the vehicle also

has three planes of symmetry, this suggests that we can neglect the contributions from the off-diagonal elements in the added inertia M_A. The mass matrix of the vehicle is the sum of the rigid-body mass and the added inertia matrices:

$$M = M_{RB} + M_A \tag{3.17}$$

The rigid-body mass is a diagonal matrix since the vehicle is assumed to have three symmetry planes. The off-diagonal contributions are neglected. The simplified rigid-body mass matrix is defined as

$$M_{RB} = \begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & I_x & 0 & 0 \\ 0 & 0 & 0 & 0 & I_y & 0 \\ 0 & 0 & 0 & 0 & 0 & I_z \end{bmatrix} \tag{3.18}$$

The following simple expressions for M_A and C_A are preferred for SAGA:

$$M_A = -\operatorname{diag}\{X_{\dot{u}}, Y_{\dot{v}}, Z_{\dot{w}}, K_{\dot{p}}, M_{\dot{q}}, N_{\dot{r}}\} \tag{3.19}$$

$$C_A(v) = \begin{bmatrix} 0 & 0 & 0 & 0 & -Z_{\dot{w}}w & Y_{\dot{v}}v \\ 0 & 0 & 0 & Z_{\dot{w}}w & 0 & -X_{\dot{u}}u \\ 0 & 0 & 0 & -Y_{\dot{v}}v & X_{\dot{u}}u & 0 \\ 0 & -Z_{\dot{w}}w & Y_{\dot{i}}v & 0 & -N_{\dot{i}}r & M_{\dot{q}}q \\ Z_{\dot{w}}w & 0 & -X_{\dot{u}}u & N_{\dot{i}}r & 0 & 0 \\ -Y_{\dot{v}}v & X_{\dot{u}}u & 0 & -M_{\dot{q}}q & 0 & 0 \end{bmatrix} \tag{3.20}$$

The rigid-body Coriolis and centripetal matrix is related to the speed of the vehicle and is defined as

$$C_{RB}(v) = \begin{bmatrix} 0 & 0 & 0 & 0 & m(w) & -m(v) \\ 0 & 0 & 0 & m(-w) & 0 & m(u) \\ 0 & 0 & 0 & m(v) & m(-u) & 0 \\ 0 & m(w) & -m(v) & 0 & I_z r & -I_y q \\ m(-w) & 0 & m(u) & -I_z r & 0 & 0 \\ m(v) & m(-u) & 0 & I_y q & 0 & 0 \end{bmatrix} \tag{3.21}$$

The gravitational force f_G acts through the center of gravity r_G of the vehicle. Similarly, the buoyancy force f_G acts through the center of buoyancy r_B. For SAGA, the center of gravity and buoyancy are chosen respectively as

$$r_G = \begin{bmatrix} x_G & y_G & z_G \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \tag{3.22}$$

$$r_B = \begin{bmatrix} x_B & y_B & z_B \end{bmatrix} = \begin{bmatrix} 0 & 0 & z_B \end{bmatrix} \tag{3.23}$$

The restoring force and moment matrix is defined as

$$g(\boldsymbol{\eta}) = \begin{bmatrix} (W-B)s\theta \\ -(W-B)c\theta s\phi \\ -(W-B)c\theta c\phi \\ -c\theta c\phi\left(y_G W - y_B B\right) + c\theta s\phi\left(z_G W - z_B B\right) \\ c\theta c\phi\left(x_G W - x_B B\right) + s\theta\left(z_G W - z_B B\right) \\ -c\theta s\phi\left(x_G W - x_B B\right) - s\theta\left(y_G W - y_B B\right) \end{bmatrix} \tag{3.24}$$

where

W is the weight of body
B is the buoyancy force, and
Z_B is the z-axis component of the buoyancy center.

The damping force matrix is assumed to consist of only linear terms, since the vehicle is able to move only at low speeds. The contribution of the quadratic damping terms is neglected. Hence, the following simple expression for D for SAGA is obtained:

$$D = -\operatorname{diag}\left\{X_u, Y_v, Z_w, K_p, M_q, N_r\right\} \tag{3.25}$$

CHAPTER 4

REALISTIC DYNAMIC SIMULATION OF UUV

4.1 Introduction

In this section, we describe how our motion controller can be used for more realistic scenarios using a high-fidelity simulation infrastructure build on top of the Gazebo and robot operating system (ROS) packages [33].

4.2 Simulation Environment

The employment of a simulation environment in the development and optimization process for subsea applications allows an early evaluation of both the vehicle's system and the mission strategy. The aim of this section is to provide a simulation environment for the underwater vehicles and the scenarios in which they will be deployed.

Upon evaluation of the requirements, the following functional requirements are identified:

- The simulation should be able to communicate with external programs (such as MATLAB) using ROS messages and modules. This includes the implementation of ROS communication interfaces for all simulated vehicles, sensors, actuators, control systems and environment scenarios.
- It must be possible to simulate multiple instances of underwater vehicles in one scenario to allow the simulation of their interaction in cooperative missions.
- The simulation must include in its physics engine the simulation of rigid-body

dynamics, hydrodynamic and hydrostatic forces and contact forces.

- The inclusion of new vehicles and scenarios should be possible without changes to the source code by combining sensor, actuator and hydrodynamic modules in a standard configuration file.
- The simulator should provide a graphic interface with a rendering engine for visualization of the current scenario.

Since there is already a range of robotics simulation software solutions that provide both physics and rendering engines in the literature, the focus is to extend an existing robotics simulation tool and adapt it for the needs of underwater robotics simulation. The comparison of simulation platforms and justification for the chosen solution can be found in the next section.

4.2.1 Selection of Suitable Environment

To select a suitable robotics simulation environment to fulfill the requirements in the previous section, a survey of existing solutions was made based on the evaluation criteria given in [40] for a similar survey. The adapted criteria are listed below

- Physical fidelity for simulation of rigid-body dynamics and collisions,
- Programmatic interface with external clients,
- Possibility of extensibility for inclusion of new dynamic models for sensor, vehicle and environment modules,
- Adequate documentation,
- Previous use in research,
- Regularity of further development and maintenance.

In Table 4.1, a preliminary list of benchmarked simulators is presented with the corresponding license, the maintainer and a short description. The most relevant features of these environments are described in the following sections and they are compared as well.

Table 4.1: List of robotics simulation environments.

Simulator	License	Maintainer	Description
V-REP	Dual license possibility (Commercial or GNU/GPL)	Coppelia Robotics	General purpose robotics simulator with a ROS interface with a plug-in-based extendibility framework in several programming languages.
UWSim	BSD	IRSLab, Jaume-I University, Spain	A ROS-based simulation environment for underwater robotics that uses OpenSceneGraph for rendering underwater environments.
MORSE	BSD	LAAS-CNRS,France	General purpose robotics simulator based on the Blender Game Engine and Bullet as the physics engine with interfaces to six middlewares including ROS.
Gazebo	Apache 2.0	OSRF (Open Source Robotics Foundation)	Open-source general purpose robotics simulator with a plugin-based extendibility framework and fully integrated with ROS.

4.2.1.1 V-REP

V-REP is a general purpose robot simulator developed by Coppelia Robotics. V-Rep EP uses a custom rendering engine (Bullet, ODE and Vertex) which can be changed at run-time. It allows simulations to be programmed using plug-ins, embedded scripts or ROS nodes and includes an API for C++. Proximity and vision sensors are already implemented in the simulator and the other sensors can be implemented via plug-ins.

According to our research, no AUV simulation is implemented using V-REP. The simulator supports underwater robots in general and simulation of an underwater ve-

hicle is possible by creating the appropriate underwater environment with the use of plug-ins.

4.2.1.2 UWSim

This underwater simulator is an open-source project from the University Jaume I, Castellón, Spain, built in the scope of marine robotics research and development [41]. It uses OpenSceneGraph and its physics engine, Bullet, as the simulation backend and provides an interface with external clients using the ROS middleware [40]. It provides very realistic visual fidelity to underwater environments (see Figure 4.1), the possibility for multi-robot simulations, sensor modules out-of-the-box and dynamic modeling based on Fossen's equations of motion for marine crafts [35].

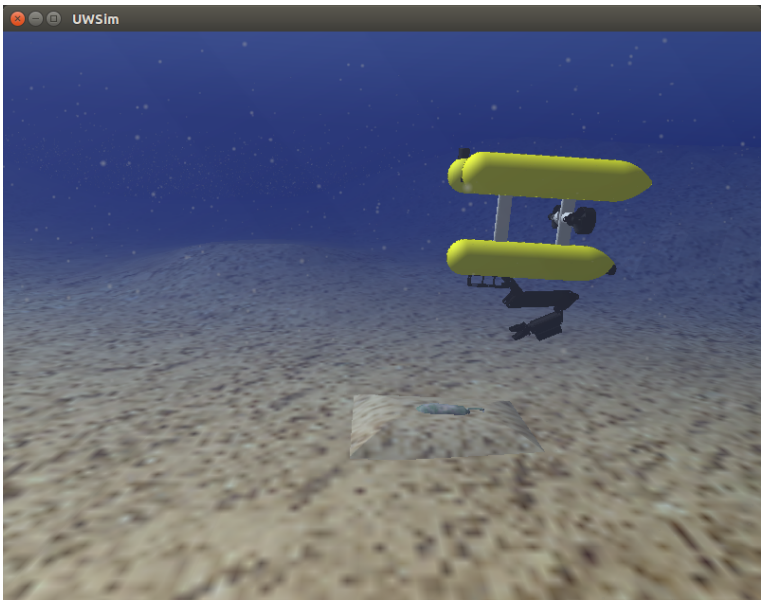


Figure 4.1: UWSim scenario with Girona 500 AUV

New scenes and vehicles can be created through an extensible markup language (XML) file. The vehicle is described in the usual unified robot description format (URDF) format from ROS. Sensor modules include camera, pressure sensor, Doppler velocity logger (DVL), inertial measurement unit (IMU), among others. The simulation of the vehicle dynamics is implemented in a monolithic Python script config-

urable through a YAML file, making the internal physics engine responsible only for the contact physics. This simulator has been widely used in the field of underwater robotics, including the RAUVI and the EU-funded TRIDENT projects.

The drawbacks of UWSim involve the extendibility of the simulation environment to other purposes. Although the simulation of multiple robots is supported, this has to be manually configured in the scenario's XML file along with settings for the environment, vehicles and other objects, which is a tedious and error-prone task [40]. Furthermore, modifications and/or extensions to other dynamic, actuator and sensor models must be done directly in the code. Although documentation is available, the simulator suffers from long periods of time without maintenance and/or development of additional features.

4.2.1.3 MORSE

MORSE [42] is a general purpose simulator developed by the LAAS-CNRS laboratory. It provides a library of sensor, actuator and robot models ready to use and is based on the Blender Game Engine for rendering and Bullet for the physics simulation (See Figure 4.2). It can be extended with Python plug-ins and supports the use of six different open-source middlewares, including ROS.

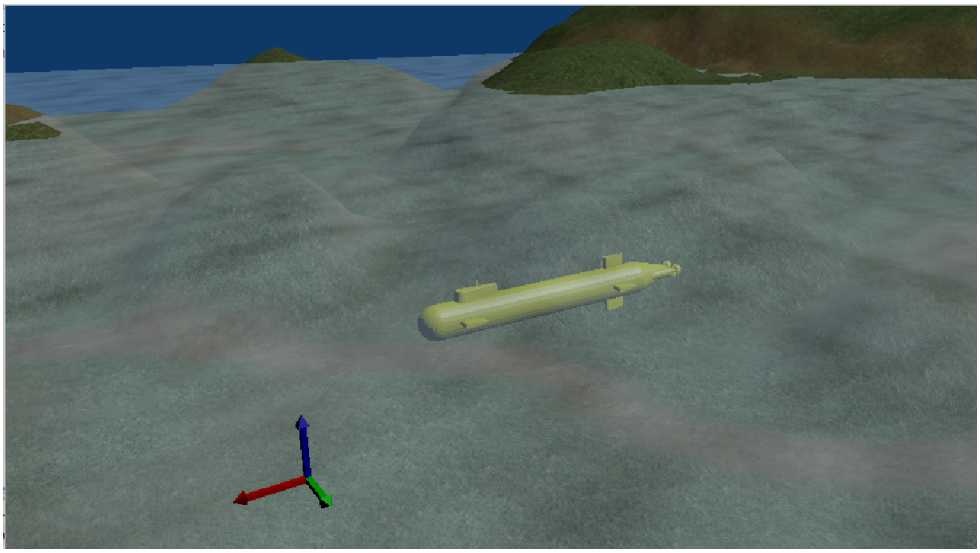


Figure 4.2: Simulation of a submarine in MORSE.

Although not configured specifically for underwater robotics simulation, it can be extended for this purpose by adding the necessary plug-ins and using the game engine for rendering.

The Blender engine has also a drawback in that the configuration of new scenarios is quite difficult if the user is not familiar with the Blender rendering software for the creation of the correct shaders [40].

4.2.1.4 Gazebo

Gazebo [43] is a general purpose robotics simulator developed and maintained by the Open Source Robotics Foundation (OSRF) and is, at the time of this study, one the most popular open-source platforms for robotics simulation. It gives the option of interfacing with any of four physics engines (ODE, Bullet, Simbody, or DART) and uses OGRE for the visualization rendering.

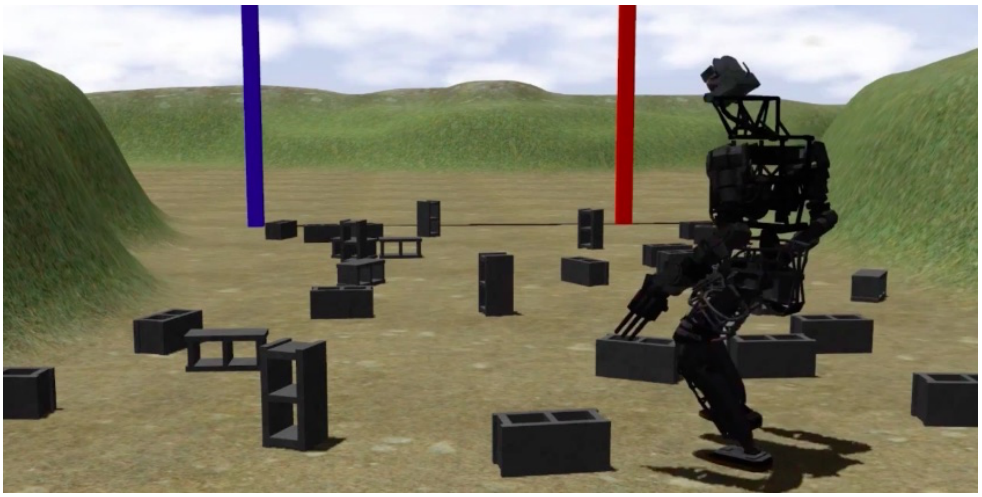


Figure 4.3: ATLAS humanoid robot from the DARPA Robotics Challenge in the Gazebo simulator environment.

This platform allows the development of custom plug-ins for building new features and modules for sensors, environments and dynamics using its C++ API, which al-

lows a high degree of modularization of the simulation. The interface between plug-ins is a socket-based approach that uses Google's Protobufs, allowing the server to run remotely from the graphical user interface (GUI). Multi-robot simulation is also supported and all entities in the simulation environment can be configured through an XML-based SDF format. ROS also has bindings to Gazebo, making extension of the simulation to the ROS middleware quite straightforward.

Gazebo delivers a vast library of sensor, actuator and robot models, having also several other contributions from the robotics community. It has also been employed in projects such as the DARPA Robotics Challenge, which has played a large role in spurring investment in further improvements to the software. Although it offers no specific applications for underwater robotics, implementations of plug-ins extending Gazebo's functionalities to this purpose have been published in [44]. However, there is great interest in extending the platform for underwater robotics with the release of new drag, lift and buoyancy plug-ins.

Furthermore, the project features a clear roadmap and simulation schedule and a clear code and integration standard for new contributions delivered by developers external to OSRF. This ensures that Gazebo is continuously under improvement, and new features and bug fixes are officially released every six months.

The drawbacks of the platform are the lack of official plug-ins for the computation of hydrodynamic forces and the lack of visual features for a more realistic depiction of underwater scenarios. This last issue, however, is expected soon to be solved given the current development of a generic interface for multiple graphics engines.

4.2.1.5 Discussion and Selection of a Simulation Environment

The benchmarking for choosing a simulation environment showed that all open-source options require some level of adaptation to fulfill the requirements listed in Section 4.2. A detailed list of the advantages and disadvantages of each simulation platform is presented in Table 4.2.

Table 4.2: Comparison of the analyzed simulation platforms.

Simulator	**Advantages**	**Disadvantages**
V-REP	• Cross-platform (Windows,Linux and Mac), • Interface with ROS, • Multiple physics engines to choose, • User-friendly GUI, • Particle dynamics already available.	• Would require a commercial license to all involved in the implementation of this simulation environment (only free for educational purposes and for the visualization tool)
UWSim	• Good visualization of underwater environments (e.g. floating particles, light damping), • Several examples of publications using the simulator as proof-of-concept for different algorithms, • Several implementations of underwater sensors already available, • Interface with ROS.	• Requires manual editing of a single XML file to define new simulation scenarios (environment, vehicles, objects) – bad modularization, • Monolithic implementation of the dynamic model for vehicle and actuators, • Physics engine used mostly for contact physics, • Unstable maintenance schedule.

MORSE	• Allows use of six different robotics middleware solutions (including ROS), • Has examples of implementations for underwater environments, • Extendible through plugins (modular simulation concept).	• Requires knowledge of usage of Blender.
Gazebo	• Extendible through plugins (modular simulation concept), • Vast library of sensors and actuator out-of-the-box, • Interface with ROS, • Multi-robot simulation is possible, • A new simulation scenario can be built out of modular configuration files, • Regular and clear roadmap for new software releases, • Extensive contribution from the robotics community for new modules and bug fixes, • Good documentation, • Big community (several sources of information when searching for examples for new implementations).	• Underwater worlds are not available, have to be implemented, • Underwater visualization effects like light damping and floating particles are hard to implement in the current stage, • GUI for construction of new models still very primitive.

Gazebo is a general purpose, accurate and effective simulation environment that offers high degree of fidelity physics in simulation, several sensors, and interfaces for users and developers.It is obvious that physically realistic simulations are desirable for all robotics applications but they are particularly critical for underwater systems due to the high equipment and operational costs.

The Gazebo simulation environment is compatible with ROS which is now the most popular software and communication framework in the robotics community. The advantages of Gazebo and the support for it make it desirable for us as a simulation environment. For these reasons, we selected the Gazebo as simulation environment.

4.3 Extending Gazebo to Subsea Robotics Simulation

As explained in the previous section, Gazebo is a powerful simulation tool, but it does not support underwater simulation out-of-the-box. Missing features that are important for underwater simulation are buoyancy, damping forces, underwater current, and added mass. These features can be added by implementing model plug-ins using the Gazebo API to simulate additional forces acting on underwater objects.

An ROV is a tethered, unmanned, remotely operated underwater vehicle. In general, an operator controls the ROV from an operator console on the surface with the help of feedback from onboard sensors such as camera, sonar, depth sensor and compass. The ROV may also feature manipulators to perform underwater tasks such as cutting and welding. Controlling an ROV to perform a specific task can be difficult and time consuming due to the harsh underwater environment. The success of such missions depends greatly on the experience of the ROV operator. In many situations, an operator alone would not be able to achieve the mission goals within the allotted time frame, thus a degree of ROV autonomy becomes critical. Increasing the level of autonomy can save time and money in underwater operations. Controlling an ROV with a control system with feedback from sensors reduces the dependency on the operator's skills, thus increasing efficiency. Operators can then focus on the mission itself rather than on controlling the vehicle. Developing such a control system, however, is not easy. The performance of the control system depends greatly on parameters such

as the accuracy of the sensors and the parameters of the autopilot.

There may be many sensors on an ROV, depending on its mission. Sensor options include camera (mono, stereo, multiple cameras), sonar (Side Scan Sonar, Forward-Looking Sonar), depth sensor, IMU, gyro, USBL, acoustic modem, conductivity temperature and depth (CTD) probe and altimeter. One of the most frequently performed tasks with ROVs is inspection of underwater objects such as pipelines or shipwrecks [45]. The difficulty in these tasks is that the operator should focus on the target rather than controlling the underwater vehicle [46]. In such a task, a control for holding the ROV in the right position is very important. Currents in the underwater environment make such control difficult for operators, so an autopilot plays a crucial role.

4.4 Implementation in Gazebo

The simulator environment that is used in this study is presented in [33]. This simulation environment enables the modeling of multiple types of underwater vehicles, sensors and environments. We integrated our SAGA ROV model, its dynamics and basic control algorithms into this simulation environment. The main purpose in using this simulation environment is to develop a solution that is compatible with ROS and the other frameworks that can be integrated with the Gazebo framework. To achieve this goal, the simulation is implemented in a modular way with plug-in modules.

There are two types of plug-in modules developed in the simulator. The first type is used for modeling vehicles, sensors and the environment. These plug-ins extend functionalities that are currently available in Gazebo. The second type is used to interact with ROS. These plug-ins exchange data between Gazebo and ROS via ROS topics and are used as bridges to implement control and navigation algorithms.

The simulation environment also has rendering and collision-detection capabilities. The objects, vehicles, and worlds are described in a Gazebo-specific format called SDF format, which is similar to XML format. The applications that use the ROS framework use URDF format (which is also similar to XML) to describe the robot's features.

The SDF files store information on all aspects of a robot, such as kinematic and dynamic attributes, sensors, surface properties, textures, joint friction, and many other properties. In the simulator, the vehicle and sensor descriptions must also contain a geometry file to be used in rendering and collision detection. The properties used in rendering are stored in COLLADA format and the properties stored for collision detection are in STL format. Two different formats are used to distinguish the rendering and collision detection-jobs. Since collision detection can take considerable time for larger geometries, its properties are supplied in a different file.

We implemented an *UnderwaterWorld* world plug-in that simulates underwater current and distributes it to all underwater objects by publishing it via a Gazebo topic. We simulate underwater current to either be constant or vary randomly by following a Gauss-Markov process.

The modified equations of motion described above were implemented within an *UnderwaterObject* model plug-in that is instantiated for each rigid body that should experience underwater effects.

The *UnderwaterObject* plug-in requires its physical parameters to be set when instantiated via URDF and it subscribes to the underwater current Gazebo topic published by an *UnderwaterWorld* to compute the relative velocity of the object with respect to the surrounding water. An example instantiation is shown below.

```
<gazebo>
  <plugin name="uuv_plugin" filename="libunderwater_object_ros_plugin.so">
    <!-- Fluid density required to compute buoyancy force -->
    <fluid_density>1028.0</fluid_density>
    <!-- Topic where current underwater flow velocity is published. -->
    <flow_velocity_topic>/hydrodynamics/flow_velocity</flow_velocity_topic>
    <link name="${namespace}/base_link">
      <!-- Volume required to compute buoyancy force. -->
      <volume>${volume}</volume>
      <!-- Center of buoyancy: Position at which buoyancy force acts. -->
      <center_of_buoyancy>${cob}</center_of_buoyancy>
      <hydrodynamic_model>
        <type>fossen</type>
        <!-- Added mass matrix -->
        <added_mass>
          779.79 -6.8773 -103.32 8.5426 -165.54 -7.8033
         -6.8773 1222 51.29 409.44 -5.8488 62.726
          -103.32 51.29 3659.9 6.1112 -386.42 10.774
          8.5426 409.44 6.1112 534.9 -10.027 21.019
          -165.54 -5.8488 -386.42 -10.027 842.69 -1.1162
          -7.8033 62.726 10.775 21.019 -1.1162 224.32
        </added_mass>
        <!-- Linear damping coefficients on diagonal of D -->
        <linear_damping>-75 -69 -728 -269 -310 -105</linear_damping>
        <!-- Quadratic damping coefficients on diagonal of D -->
        <quadratic_damping>-748 -993 -1821 -672 -774 -523</quadratic_damping>
      </hydrodynamic_model>
    </link>
  </plugin>
</gazebo>
```

There are different physics engines in the Gazebo package, but they do not support underwater dynamic effects directly. These effects are integrated by developing plug-ins for hydrodynamic and hydrostatic forces. In the following sections, important plug-ins in the simulator are described.

4.4.1 Thruster Plug-Ins

Thrusters are the essential means of locomotion of ROVs and AUVs: they are a combination of motors and propellers that can often be operated in both directions and produce a thrust force parallel to the rotation axis of the propeller. This phenomenon is simulated using newly-implemented thruster plug-ins for both Gazebo alone and Gazebo in conjunction with ROS. These include a variety of steady-state conversion functions from angular velocity to obtained thrust forces and a dynamic model to account for the limited rate of change of the angular velocity of the thruster. The thrust

force computed by the physical model is then applied to the thruster link at the correct pose of the thruster.

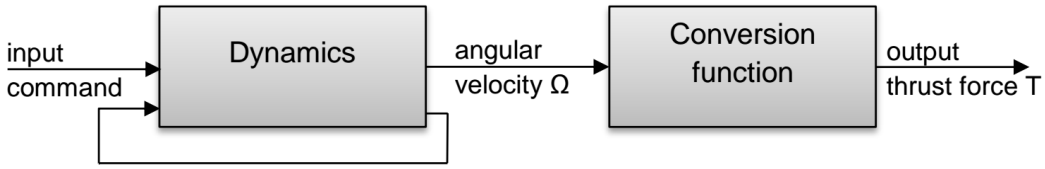


Figure 4.4: Overview simulation of thrust force.

Thrusters can be controlled either via Gazebo directly or via ROS-topics when using the Gazebo/ROS plug-in *libThrusterROSPlugin.so*.

An example of an instantiation of a thruster using this thruster plug-in is shown below:

```
<gazebo>
 <plugin name="thruster_model" filename="libThrusterROSPlugin.so">
  <linkName>thruster_link_name</linkName>
  <jointName>thruster_joint_name</jointName>
  <thrustTopic>thruster_thrust_topic</thrustTopic>
  <inputTopic>thruster_input_topic</inputTopic>
  <conversion>
   <type>Basic</type>
   <rotorConstant>0.00031</rotorConstant>
  </conversion>
  <dynamics>
   <type>FirstOrder</type>
   <timeConstant>0.05</timeConstant>
  </dynamics>
 </plugin>
<gazebo>
```

The parameter linkName refers to the link model used to describe the visual, collision and rigid-body model of the rotating part of the thruster object. It is connected to the fixed part through a joint, the label for which should be given in jointName. This joint lets the propeller rotate according to the angular velocity of the thruster.

If the ROS version of the plug-in is used, two further arguments must be provided for the necessary ROS topics. *thrustTopic* is the output topic that publishes the thrust

force generated of type *std_msgs::Float64* and *inputTopic* is reserved to receive the input command, also as *std_msgs::Float64*. The configuration of the conversion and dynamics blocks are described in the following sections.

4.4.1.1 Conversion Curve

The conversion curve maps the angular velocity of the thruster to the actual amount of thrust force it produces. This relation is often provided by manufacturers and shown as a plot within the technical specifications. If it is not available, it can be determined experimentally by connecting the thruster to a force sensor. The relationship between angular velocity and produced thrust is often a simple quadratic one (Basic) or is quadratic with a dead-zone interval (Bessa). We can, however, also model arbitrarily complex conversion curves using linear interpolation.

4.4.1.1.1 Basic Conversion The basic conversion curve assumes that the control input is the angular velocity of the rotor and the thrust force is proportional to the angular velocity squared. Since most thrusters can be operated in both directions, it follows for the thrust equations that

$$T = k\Omega|\Omega| \tag{4.1}$$

where k is a rotor constant and Ω is the input command, the angular velocity of the rotor. An example URDF configuration is shown below.

```
<conversion>
  <type>Basic</type>
  <rotorConstant>0.00031</rotorConstant>
</conversion>
```

4.4.1.1.2 Bessa Conversion The Bessa conversion function is the implementation of the dead-zone function described in [2] and depicted in Figure 4.5. Except for the added dead-zone interval, it is identical to the basic conversion function described

above:

$$(\Omega|\Omega|) = \begin{cases} k_l\left(\Omega|\Omega| - \delta_l\right), & \Omega|\Omega| \leq \delta_l \\ 0, & \delta_l < \Omega|\Omega| < \delta_r \\ k_r\left(\Omega|\Omega| - \delta_r\right), & \Omega|\Omega| \geq \delta_r \end{cases} \tag{4.2}$$

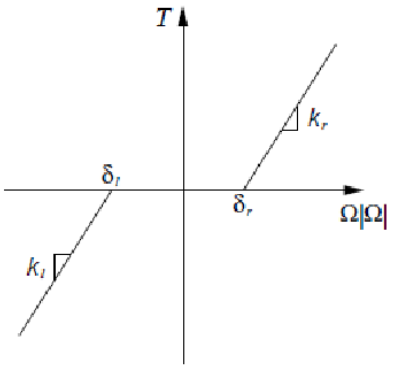


Figure 4.5: Dead-zone nonlinearity[2].

k_l and k_r are rotor constants for the reverse and forward motion, respectively, and δ_l and δ_r delimit the range of the dead-zone for the input $\Omega|\Omega|$, Ω being the propeller's angular velocity.

An example of how the Bessa conversion function can be instantiated within a URDF description is shown below.

```
<conversion>
  <type>Bessa</type>
  <rotorConstantL>0.00031</rotorConstantL>
  <rotorConstantR>0.00031</rotorConstantR>
  <deltaL>0.5</deltaL>
  <deltaR>0.5</deltaR>
</conversion>
```

4.4.1.1.3 Linear Interpolation Finally, the most general conversion function linearly interpolates an arbitrary conversion function, given enough interpolation points. It takes a list of input/output pairs which may be provided by the manufacturer of a thruster, taken from a conversion function plot, or measured in experiments, and interpolates between these points. Input values that fall outside of the provided interval

(above the maximum or below the minimum input values) are simulated by copying the nearest neighbor.

An example instantiation of the linear interpolation conversion function with only five interpolation points is shown below.

```
<conversion>
  <type>LinearInterp</type>
  <inputValues>-10 -1 0 1 10</inputValues>
  <outputValues>-5 -1 0 1 5</outputValues>
</conversion>
```

4.4.1.2 Dynamics

Considering the conversion function of a thruster alone is often not enough: particularly for large vehicles, the propeller of a thruster cannot instantaneously accelerate from one angular velocity to another. We simulate this effect using one of the following dynamics models.

4.4.1.2.1 Zero Order This dynamic model corresponds to not simulating dynamics at all. Instead, the angular velocity of the thruster immediately jumps to the latest commanded value and thus does not require any additional parameter. It can be instantiated from URDF with the following block.

```
<dynamics>
  <type>ZeroOrder</type>
</dynamics>
```

4.4.1.2.2 First Order The first-order dynamic model lets the angular velocity follow the commanded value with a first-order behavior. It requires as its only parameter the time constant in s^{-1}. This option is configured as follows:

```
<dynamics>
  <type>ZeroOrder</type>
  <timeConstant>0.2</timeConstant>
</dynamics>
```

4.4.1.2.3 Yoerger's Model

Whereas the previous two models assumed that the input command is the desired angular velocity of the thruster, the following two models are different: Yoerger's model for the energy-based approach for propeller dynamics described in [47] considers a commanded torque as input and can be expressed using the following differential equation:

$$\dot{\Omega} = \beta \tau_{input} - \alpha \Omega |\Omega| \tag{4.3}$$

where τ_{input} is the commanded input torque, and β and α are constant model parameters (for a detailed explanation of this model, see [47]). This option is configured as follows.

```
<dynamics>
  <type>Yoerger</type>
  <alpha>0.2</alpha>
  <beta>0.3</beta>
</dynamics>
```

4.4.1.2.4 Bessa's Model

Motivated by experimental results, Bessa proposed a slightly refined model in [48], which is there called Model 2 (Yoerger's model is called Model 1):

$$J_{msp}\dot{\Omega} + K_{v1}\Omega + K_{v2}\Omega|\Omega| = \frac{K_t}{R_m} V_m \tag{4.4}$$

where voltage V_m is the input command. The motor torque constant K_t, winding resistance R_m are constructive characteristics that need to be experimentally determined. K_{v1} and K_{v2s} are model parameters.

```
<dynamics>
  <type>Bessa</type>
  <Jmsp>0.2</Jmsp>
  <Kv1>0.3</Kv1>
  <Kv2>0.3</Kv2>
  <Kt>0.3</Kt>
  <Rm>0.3</Rm>
</dynamics>
```

4.4.2 Sensor Plug-Ins

There are already many sensor models in Gazebo, but they are mostly suitable for ground and air applications. In the UUV simulator package, some of these sensors

are extended to be used as underwater sensors. The IMU model used in the UUV simulator is based on the work given in [49]. The default IMU sensor model does not take bias into account. The sensor in the simulator includes a zero-mean Gaussian noise and a bias that follows a random walk with a zero-mean white Gaussian noise. There is a default compass plug-in in Gazebo which was developed in [50]. In the future, we will integrate sensors such as 2D imaging sonar and mechanical scanning sonar into our SAGA ROV model in the environment.

4.4.3 SAGA Implementation

The underwater vehicles are composed of the vehicle's base link, actuators (thruster and/or fins), sensors and, if available, manipulators. The body of the vehicle is configured so that the hydrodynamic and hydrostatic forces and moments are generated by the *underwater object plug-in*, which implements Fossen's equations of motion. The vehicle's main body is considered a single link with its sensors and actuators attached to it through fixed joints. This configuration simplifies the generation of hydrodynamic and hydrostatic forces, since they have to be applied then only on the main body link.

To allow the use of the NED coordinates, a NED link is also available in addition to the base link (which, under ROS standards, must always be created for a robot model), which is represented in the conventional east-north-up (ENU) coordinate frame convention, which is the only one available in Gazebo currently.

Adding the thrusters as separated and individual links attached to the main body allows the automatic computation of thrust allocation matrices for the vehicles in the simulation without the need to add them to a configuration file. It also allows the simulation of a vehicle with multiple thruster configurations, for example, with no changes to the source code. It is important to highlight that the actuator and sensor links are only place holders for a frame; they have no physical importance to the body of the vehicle. This is done in this manner because of the fact that many vehicle models require sensors and actuators to be positioned inside of the collision geometry of the vehicle's main body. This can lead to errors at the initialization of the simulation because of internal collisions.

The SAGA vehicle model is included in the *models* folder with the following folder structure:

- **launch** contains the ROS launch files that will upload the SAGA vehicle model and respective plug-ins to Gazebo at a user-defined world position.
- **mesh** contains the Collada and STL files necessary to describe both visual and collision geometries, respectively.
- **robots** contains the full URDF description of each available robot configurations, having at least one with the label default as the standard configuration. File names have to be set in the standard <robot_name>_<configuration_name>.urdf.xacro format to allow automation of the construction of new simulation scenarios.
- **urdf** contains the URDF robot description files with vehicle description, actuators, sensors and plug-ins to be loaded. The vehicles main description file must be named <robot_name>_base.xacro.

An example of the folder standard folder structure for a simulated SAGA can be seen in Figure 4.6.

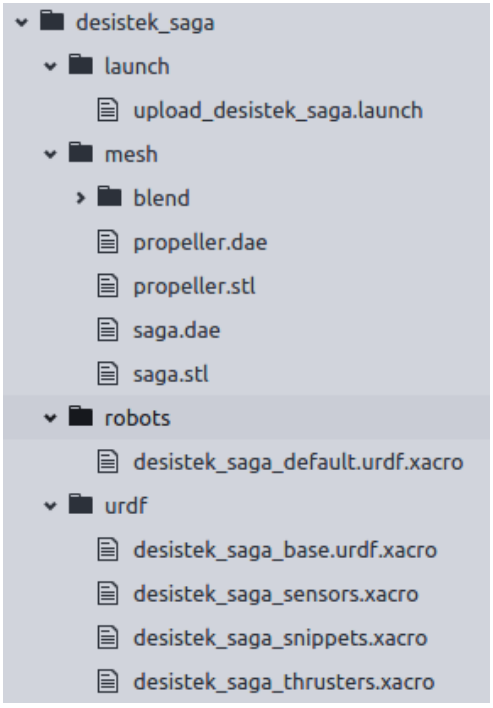


Figure 4.6: Standard folder structure for simulated SAGA.

ROS and Gazebo integration provides an effective simulation solution for integration and communication between different platforms in a modular way. We effectively use this modularity by constructing a closed loop communication protocol between MATLAB and the Gazebo simulation. Our motion controller and high-level control policy are built in MATLAB and this communicates with the SAGA ROV in Gazebo in real (soft) time.

The low-level modularity is established through plug-in modules. Some modules are used for modeling vehicles, sensors and the environment, whereas other types of plug-in modules are used to interact with ROS. These types of plug-ins exchange data between Gazebo and ROS via ROS topics and services. The ROS modules are used as a bridge between ROS, Gazebo and MATLAB to implement control and navigation algorithms in the ROS framework. We provide a screen shot from our SAGA ROV in the simulation environment in Figure 4.7.

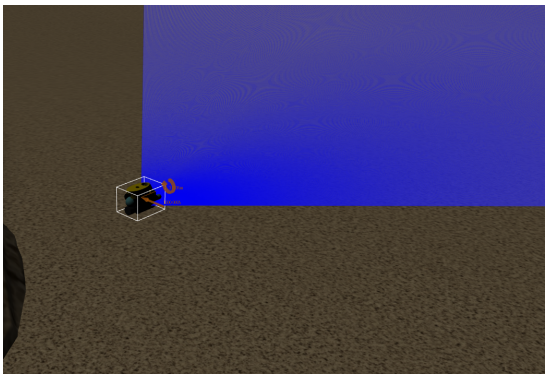


Figure 4.7: Screenshot of our SAGA ROV in the simulation environment

4.5 Control and Navigation

The simulation package [33] allows us to implement algorithms using the ROS framework and communicate with Gazebo via plug-ins. We control the dynamics of the vehicle using this communication framework. To implement our motion planning methodology inside a dynamic underwater environment (simulation or real experiment), we adopted a two-level (cascaded) control policy to navigate the vehicle in this simulation environment.

The two-level controller has two components. The low-level (inner-loop) module is a PID-based high-gain speed controller implemented inside the ROS/Gazebo UUV simulator [33]. This controller is shown as the inner-loop controller in Figure 4.8. Inputs to this block are a set of desired velocity and speed commands with respect to body fixed reference frame, whereas the output is the thrust force commands. A high-gain controller is used to minimize the error between the commanded and actual speeds. Indeed this block together with other properties (damping in underwater environments and slow operating speeds) ensures the feasibility of our assumptions in the motion controller. The high-level controller basically corresponds to our proposed motion planning controller. The nonlinear motion controller produces a set of velocity commands based on the proposed feedback rule given the current state of the robot. Produced commands are then fed to the low-level speed controller.

The complete closed-loop control diagram is illustrated in Figure 4.8. The $q_d = [x_d \ y_d \ z_d]^T$ denotes the desired global Cartesian coordinates in the 3D world coordinate system, which serves as the input for the motion planner. The motion planner then produces a set of control polices using our random sequential composition methodology to produce the output of $[v_p^d \ \omega^d \ v_v^d]^T$, which denotes the desired velocity commands to be sent to the low-level controller. In the combined simulation environment, this step is performed inside MATLAB, and the velocity commands are sent to the ROS/Gazebo package using ROS topics. The low-level speed controller takes the error signal between the reference speed signals and the actual speed of the ROV and produces a thruster speed signal, $u_T = [u_r \ u_l \ u_v]$ (right, left and vertical thruster's values in revolutions per minute), using a high-gain PID control policy. This controller plug-in sends commands to the vehicle dynamics plug-in which integrates the ROV parameters with the Gazebo's physics engine. The output of the *Dynamics* block is sent to the *Kinematics* block, which produces the kinematic parameters of the SAGA ROV in the world-fixed reference frame. Using another ROS topic, we send these parameters to the Motion Controller to close the outer-loop. This block diagram runs until the ROV reaches the desired goal point.

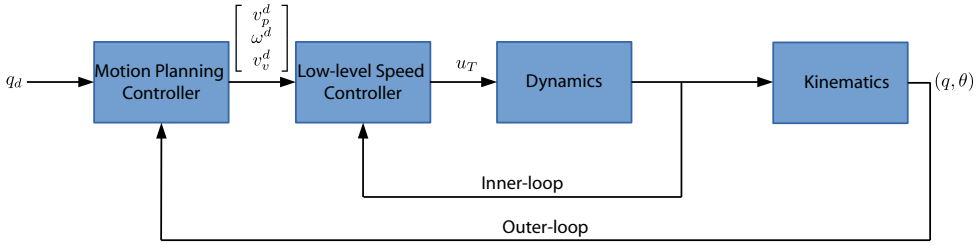


Figure 4.8: Illustration of the control loop for motion planning and navigation of the SAGA model. There are two loops inside this closed-loop system structure. The inner-loop actively regulates the error signal between the desired and actual velocities with respect to the body-fixed reference frame using a fast PID controller. The *Dynamics* and *Low-Level Speed Controller* blocks correspond to the *plant* and *controller* blocks of this inner-loop, respectively. Both blocks are realized inside the ROS/Gazebo package, and reference velocity inputs are received from MATLAB. The outer-loop is responsible for the navigation of the ROV to the desired goal state in the underwater environment. The *control* block of the outer-loop is the non-linear *Motion Planning Controller*, which basically executes our new random sequential composition algorithm. This component is realized in MATLAB and the communication between other blocks running in the ROS/Gazebo package is achieved using ROS topics. The *plant* of the outer-loop is the combined closed-loop system that includes the *Low-Level Speed Controller*, *Dynamics*, and *Kinematics* blocks along with *tracking*.

CHAPTER 5

RESULTS

5.1 2D Kinematic Simulations in MATLAB

To illustrate the important principles and properties of our new motion planning algorithm, we performed several MATLAB simulations. The equations of motion of the simplified ROV model were integrated using MATLAB's *ode45* solver. We illustrate the simulation results in Figure 5.1. The simulation environment is composed of circular obstacles with a diameter of $1.5\,\mathrm{m}$ and a circular boundary with a diameter of $25\,\mathrm{m}$ for the sake of simplicity and clarity.

We performed simulations using two different initial and goal configuration sets ($(q_{start}, q_{goal})_1 = ([-15\ 5]^T, [5\ -5]^T)$ and $(q_{start}, q_{goal})_2 = ([15\ 15]^T, [-15\ -15]^T)$). The motion planning algorithm generated the random funnels for these two sets in approximately $0.2s$ and $0.8s$, respectively. These computation times are very promising for real-time ROV/AUV applications.

In both configurations, after generating the funnel trees, two different paths were generated using different angular velocity gains. The light blue path has a higher gain that executes sharper turns than the dark blue path. We show only paths that originated from the initial configuration. Our method, however, can drive any initial condition in the environment that is covered by at least one funnel to the goal condition. It is important to note that there is no "trajectory" that we generate during the planning phase. Thus, resulting paths are the outputs of control policies that depend on the selected gains.

Figure 5.1(**b** & **d**) illustrates the 3D funnel abstractions, as in Figure 1.1 specific to

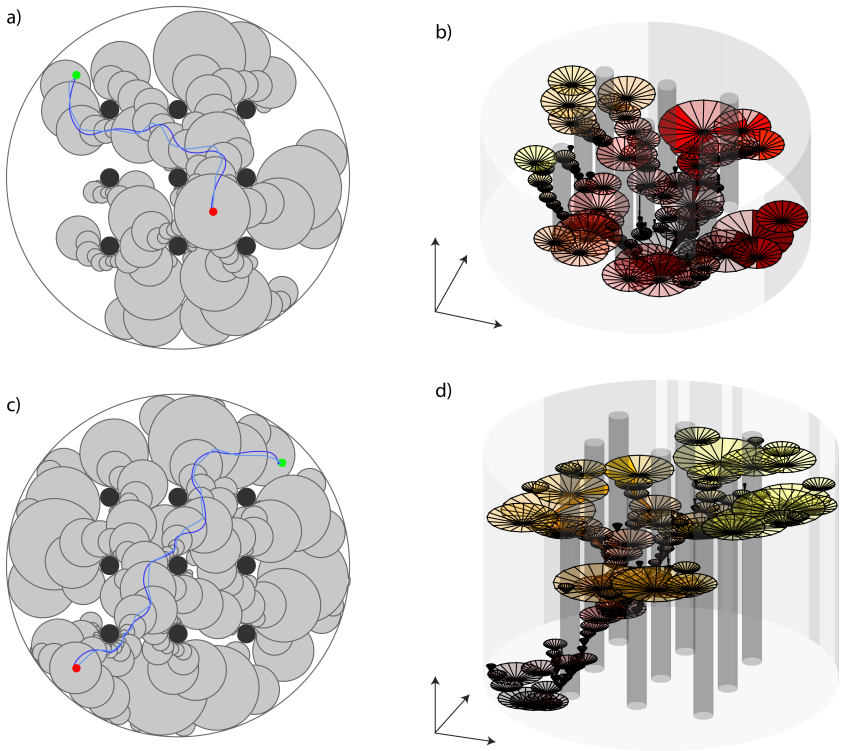


Figure 5.1: This figure illustrates the simulation results of the implementation of our motion planning algorithm in a sample environment. Upper (a & b) and bottom (c & d) rows belong to different initial and goal configurations. The **first column** (i.e., **a** & **c**) illustrates the environment, the basins of attraction of the random funnels, the initial and goal configurations (green and red markers, respectively), and two sample paths (dark and light blue). Black solid circles denote the obstacles, whereas the outer circle is the environment boundary. Light gray circles with dark grey perimeters illustrate the regions of attraction of the funnels. Each funnel control policy is responsible for navigating the initial conditions that are visible in the illustration. In other words, the sequence of circles acts like a ladder that illustrates the priority queue. The dark blue and light blue paths are the result of two different dynamic solutions and the only difference between them is in the angular velocity gain. The dark blue path has a lower gain, and thus follows a smoother path compared to the light blue path. One should note the fact that this motion planning policy can reach the goal state from any point that is covered with the grey circles, which is the main reason for the robustness. The **second column** (i.e., **b** & **d**) illustrates the metaphoric funnels in a 3D environment. The darkest red point (at the bottom of the figure) is the outlet of the *master* funnel (i.e., the global goal state). Due to the sequential composition of funnels, a metaphoric "ball" dropped inside any funnel can reach the bottom (i.e., the goal).

these simulated examples. Each inverted cone illustrates a funnel. It can be seen that each funnel except the master funnel at the bottom drops inside another funnel. By tracking this funnel tree structure, we can reach the bottom from any point that is covered by one of the funnels. The z-axis of these illustrations can be considered to be a "time" or priority queue.

In addition to spatial illustrations, in Figure 5.2, we also illustrate the temporal evolutions of the position and speed of the robot with respect to time. It can be seen that robot reaches the goal position in approximately 80 s. One can also observe from the speed plots (i.e., main control effort) there are spikes in the speed of the robot. This is an expected result because, when the robot enters a new funnel, feedback control policy changes based on sequential composition strategy.

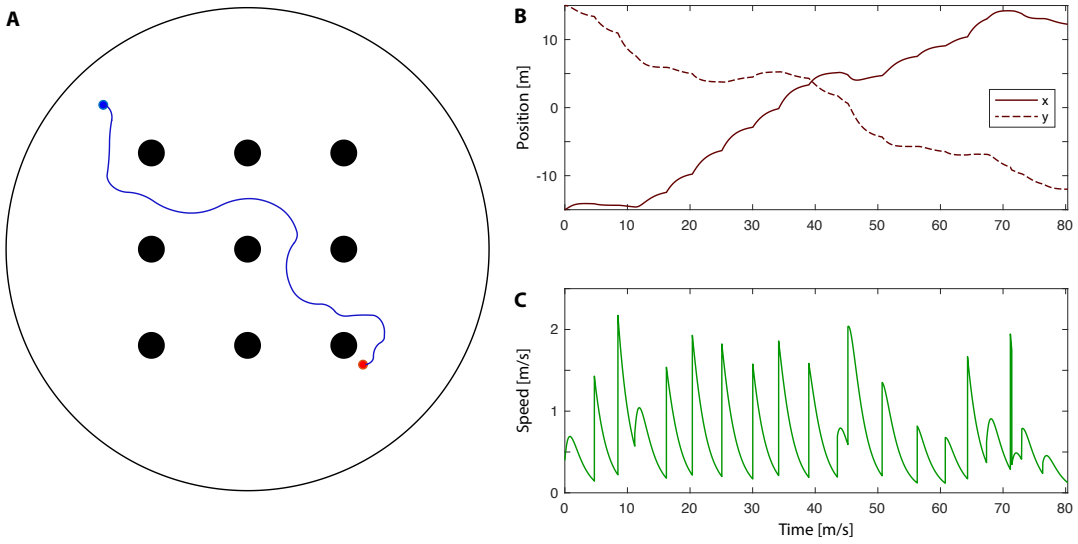


Figure 5.2: This figure illustrates the evolution of the Cartesian position and the speed of the robot with respect to time. (**A**) This figure shows the path of the robot inside the tested environment. (**B**) This figure illustrates the time dependent trajectories of the horizontal and vertical position of the robot. (**C**) This figure illustrates the temporal evolution of the speed of the robot (i.e., the main control effort) with respect to time.

We also tested the navigation method under a relatively extreme environmental uncertainty. In these simulations, we added an "environmental" (constant) velocity un-

certainty with respect to the world-fixed reference frame to simulate possible water current uncertainty. These results are illustrated in Figure 5.3. In these simulations, the speed of the added uncertainty is equal to one-half of the maximum speed of the ROV. We applied the uncertainty in two different directions specifically, along $u_1 = [1\ 0]^T$ and $u_{-1} = [-1\ 0]^T$. These simulations use the same environment and initial state/goal position configurations used in Figure 5.1. The algorithm generated the random funnels in these simulations in approximately $0.1s$ and $0.4s$, respectively.

The results show that the method is robust in the face of the applied uncertainties, since in all cases the ROV robustly converges to the goal position. One of the important features of our method is that based on the structure of the funnels and the behavior of the noise and uncertainties in the system, the ROV may follow a significantly different path. This is an expected property, because under such highly uncertain conditions trying to follow a pre-defined open-loop path may be infeasible and sometimes impossible. Instead, our method robustly and reactively finds a different solution which is one of the important features of the idea of sequential composition of controllers [16].

In addition to these simulation studies, we also compared our method to classical RRT in terms of computational complexity and sparsity. Figure 5.4 illustrates an example comparative result in the sample environment with same initial and goal configurations. In this example, our approach reduces both the computation time and the number of nodes in the resultant tree. However, to ensure fair comparisons, we performed 1.000 Monte Carlo simulations using the same environment and the same initial and goal configuration set. In these Monte Carlo simulation tests, we observed that for the original RRT the average computation time was 3.4 s and the average number of nodes was 971. For our method, by contrast, the average computation time and node count were 0.1 s and 134 respectively. It is clear that for our test environment our method both reduces the computational complexity and enhances sparsity.

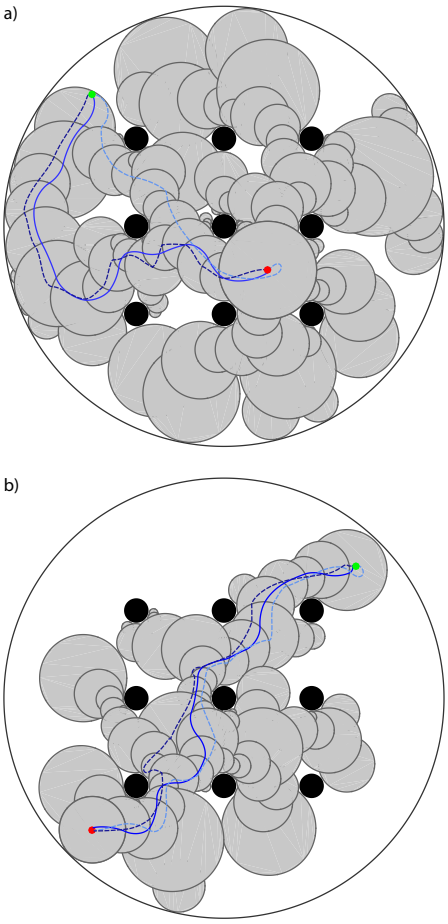


Figure 5.3: This figure illustrates the simulation results of the implementation of our motion planning algorithm in the same sample environment under a simulated environmental uncertainty. Upper (a) and bottom (b) figures display different initial and goal configurations, and these configurations match those tested in Figure 5.1. In this figure, we illustrate three different paths (solid blue, dashed dark blue, and dashed light blue), where each path belongs to a different uncertainty condition. The solid blue path represents the case in which there is no uncertainty. The dashed dark blue and light blue paths belong to the cases where the directions of the added velocity uncertainty are equal to $u_1 = [1\ 0]^T$ and $u_{-1} = [-1\ 0]^T$, respectively. We can observe that all paths converge to the goal position whether there is uncertainty or not. Clearly, in the case of added uncertainty, the ROV follows a different path. Since our method does not rely on tracking a trajectory, it can be seen that based on the structure of the funnel tree and the nature of the uncertainty, the ROV can follow substantially different paths. For example, in Figure 5.3(a), the light blue path is structurally different than other paths, whereas in Figure 5.3(b), the dark blue path follows a different funnel branch in the middle of the navigation.

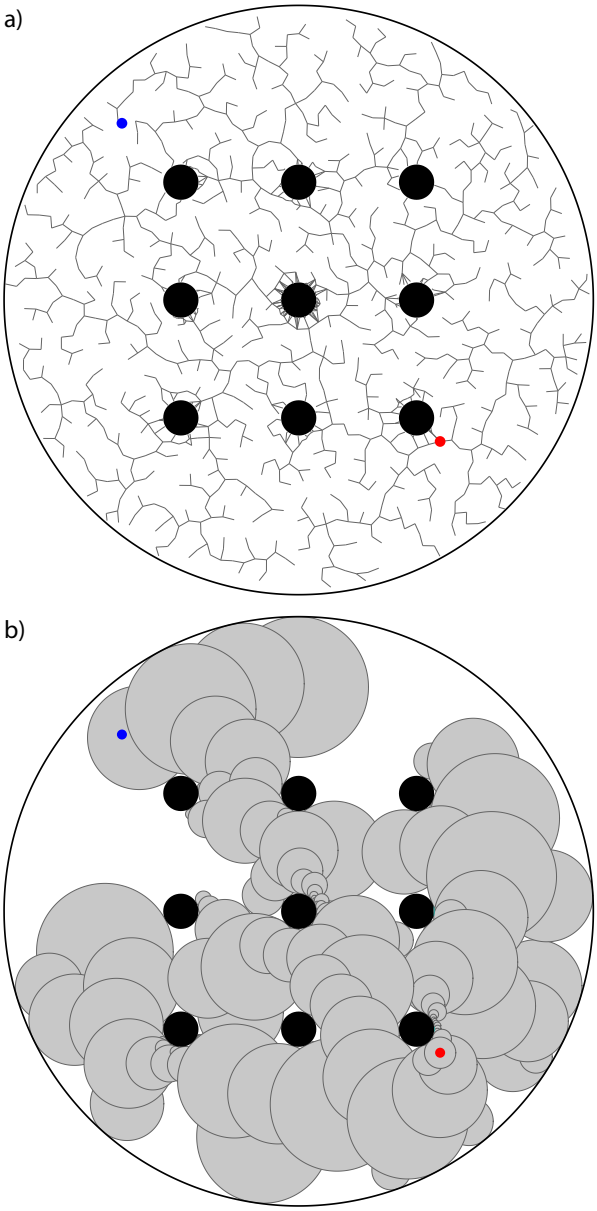


Figure 5.4: This figure illustrates the comparative simulation results of the implementation of our motion planning algorithm and the original RRT algorithm in the same sample environment. The upper (a) and bottom (b) rows represent the results of the original RRT algorithm and our method, respectively. In these results, the RRT method finds a solution (a path between the start and goal configurations) in 2.2 s and with 1.127 nodes. On the other hand, our method finds a solution in 0.11 s and with just 150 nodes. It this example, both computation time and sparsity are substantially enhanced with our method.

5.1.1 Performance Comparisons

This section provides the test results for different methods in different environments. The proposed methods are probabilistic methods, so we performed Monte Carlo tests to evaluate the performance of the algorithms. For each environment, the methods were tested for the same area with the same configuration of obstacles, but at each iteration the start and goal configurations were changed. An important parameter in the algorithm is the probability of sampling the goal node. This parameter tends the tree toward the goal node, which may prevent coverage of the map. If the probability of sampling the goal node is high, it means that the goal point is sampled frequently as a new node candidate. We tested the methods in three different environments with different number of obstacles. The number of obstacles increases in consecutive scenarios.

5.1.1.1 Scenario 1

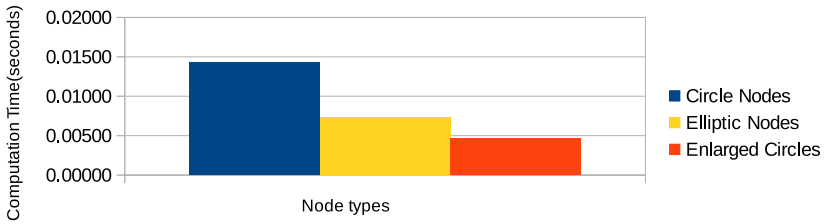
In this scenario we constructed an environment with a few number obstacles. In Figure 5.6, three methods with same initial and goal configurations are given. As can be seen, the node shapes and numbers differ for different node models.

Table 5.1 lists the results from the compared methods. All tests were performed 1.000 times on the same map but with different start and goal configurations.

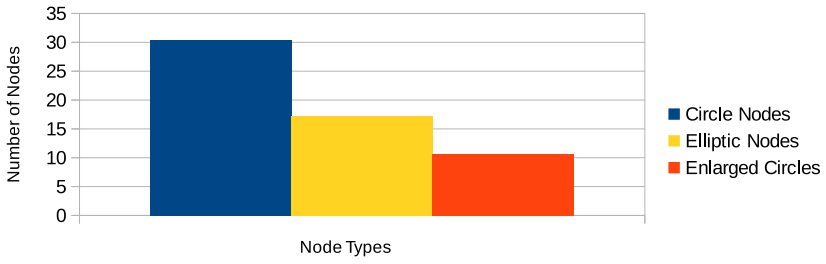
As can be readily seen from Table 5.1, if the probability of sampling the goal node is set to 1, the performance of both circular nodes and elliptic nodes decreases drastically. Setting the probability to 0 also decreases the performance. Figure 5.7 compares the computation performance of three node generation methods, and Figure 5.8 compares the numbers of generated nodes. In Figure 5.5, the average computation performances and node counts are given. Here, the outliers in the probability of sampling the goal node –0 and 1– are discarded. The sparsity enhancement methods, elliptical and circular enlargement, decreases the computation time and number of nodes.

Table 5.1: Performance results of different methods for scenario 1.

Probability	Time (seconds)			Number of Nodes		
	Circle Nodes	Elliptic Nodes	Enlarged Circles	Circle Nodes	Elliptic Nodes	Enlarged Circles
0	0.08798	0.06787	0.04384	79	46	41
0.1	0.01060	0.00525	0.00749	30	17	17
0.2	0.00961	0.00373	0.00569	26	14	13
0.3	0.00738	0.00338	0.00539	23	13	12
0.4	0.00798	0.00461	0.00442	23	13	10
0.5	0.00876	0.00354	0.00437	24	13	10
0.6	0.00956	0.00508	0.00409	24	15	9
0.7	0.01078	0.00645	0.00361	28	16	9
0.8	0.02213	0.00984	0.00332	38	19	8
0.9	0.04205	0.02430	0.00322	57	33	8
1	0.50448	1.42686	0.321	342	400	52

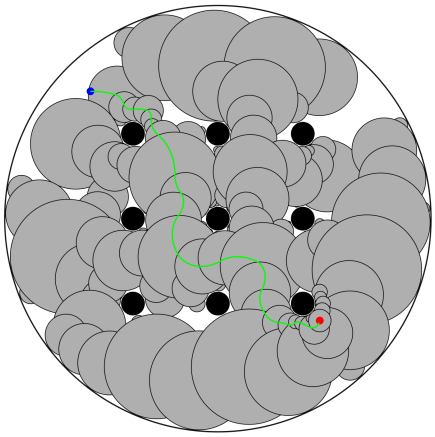


(a)

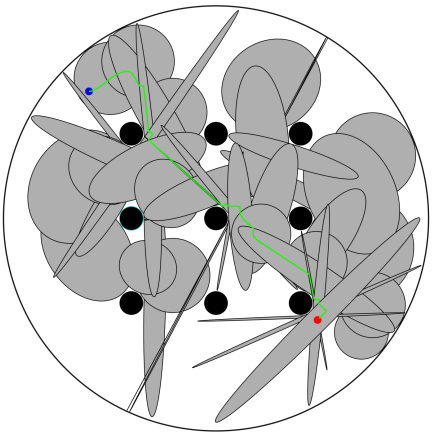


(b)

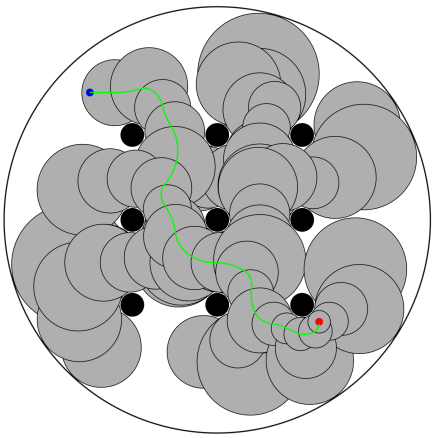
Figure 5.5: Comparison of computational performance and node count by method for scenario 1.



(a)



(b)



(c)

Figure 5.6: This figure illustrates the node styles and numbers of different node shapes for scenario 1. In (a), the algorithm uses standard circular nodes (Section 2.2). In (b), the algorithm uses elliptical nodes (Section 2.4.1). In (c), the algorithm uses enlarged circular nodes (Section 2.4.2).

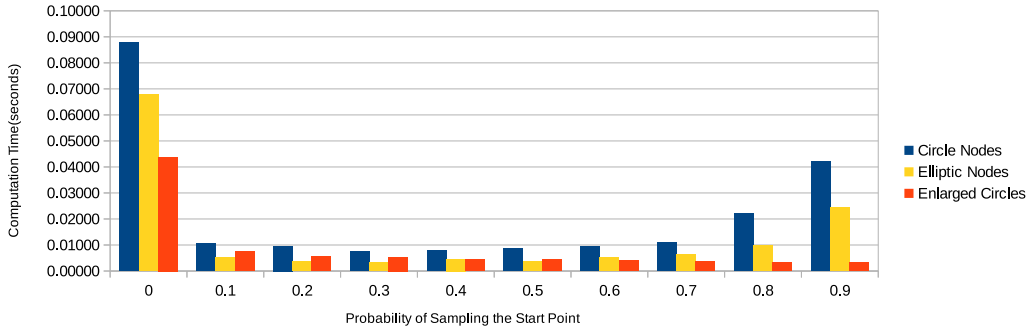


Figure 5.7: Computation performance comparison of circular, elliptic and enlarged circular nodes for scenario 1.

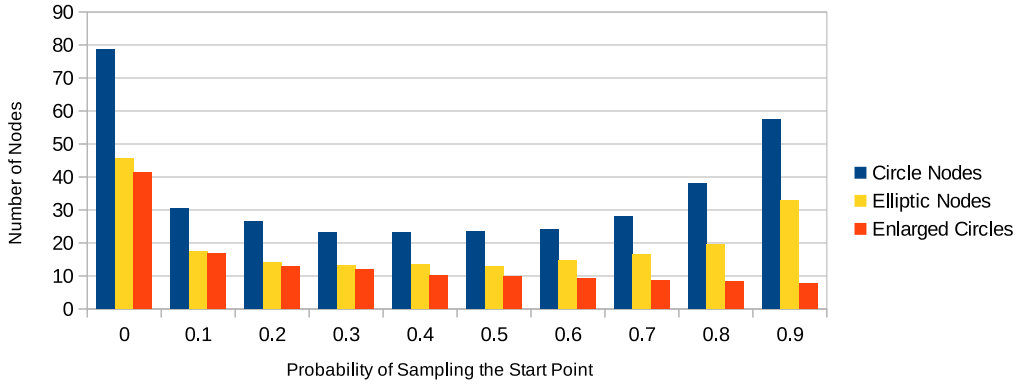


Figure 5.8: Node number comparison of circular, elliptic and enlarged circular nodes for scenario 1.

5.1.1.2 Scenario 2

In this scenario we constructed an environment with a moderate number obstacles. In Figure 5.11, three methods with same initial and goal configurations are given. As can be seen, the node shapes and numbers differ for different node models.

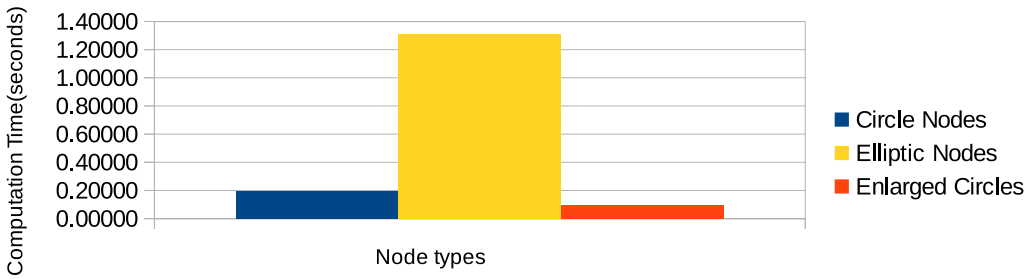
Table 5.2 lists the results from the compared methods. All tests were performed 1.000 times on the same map but with different start and goal configurations.

As can be readily seen from Table 5.2, if the probability of sampling the goal node is set to 1, the performance of both circular nodes and elliptic nodes decreases drasti-

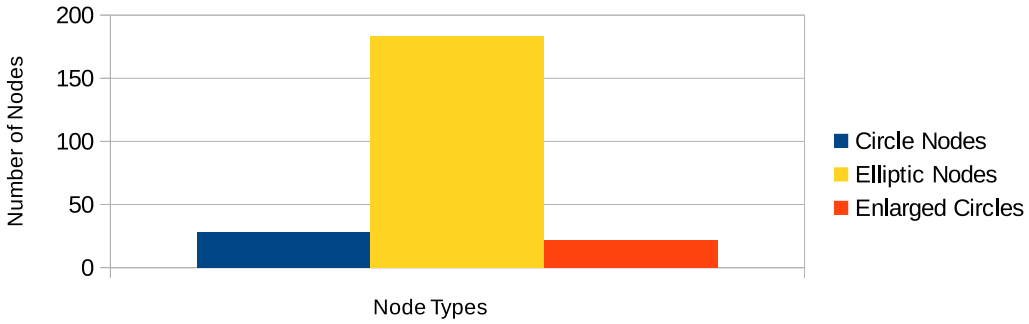
cally. Setting the probability to 0 also decreases the performance. Figure 5.10 compares the computation performance of three node generation methods, and Figure 5.12 compares the numbers of generated nodes. In Figure 5.9, the average computation performances and node counts are given. Here, the outliers in the probability of sampling the goal node –0 and 1– are discarded. The sparsity enhancement methods, elliptical and circular enlargement, decreases the computation time and number of nodes.

Table 5.2: Performance results of different methods for scenario 2.

Probability	**Time (seconds)**			**Number of Nodes**		
	Circle Nodes	**Elliptic Nodes**	**Enlarged Circles**	**Circle Nodes**	**Elliptic Nodes**	**Enlarged Circles**
0	0.08510	0.15399	0.04223	44	86	31
0.1	0.06551	0.09882	0.05212	30	64	25
0.2	0.05434	0.15049	0.04531	26	76	22
0.3	0.07999	0.33842	0.05645	23	107	24
0.4	0.07805	0.55547	0.05179	24	171	20
0.5	0.14273	0.56870	0.06325	30	153	19
0.6	0.14727	0.64839	0.07951	27	157	20
0.7	0.23469	1.33188	0.11047	32	240	22
0.8	0.29518	3.48141	0.13055	26	325	19
0.9	0.66765	4.62356	0.27694	31	359	21



(a)



(b)

Figure 5.9: Comparison of computational performance and node count by method for scenario 2.

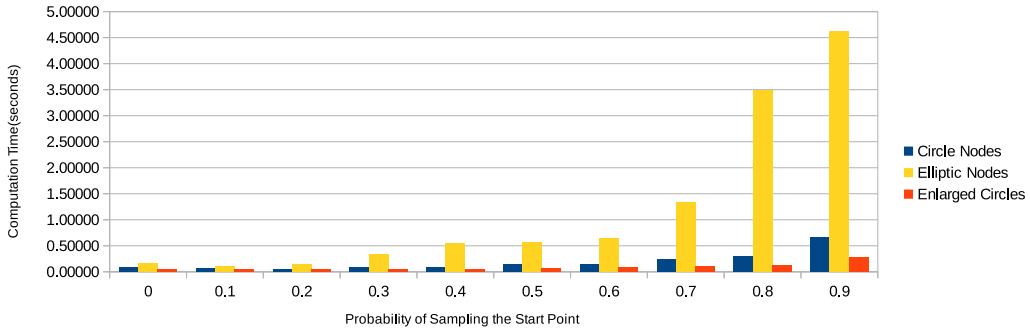
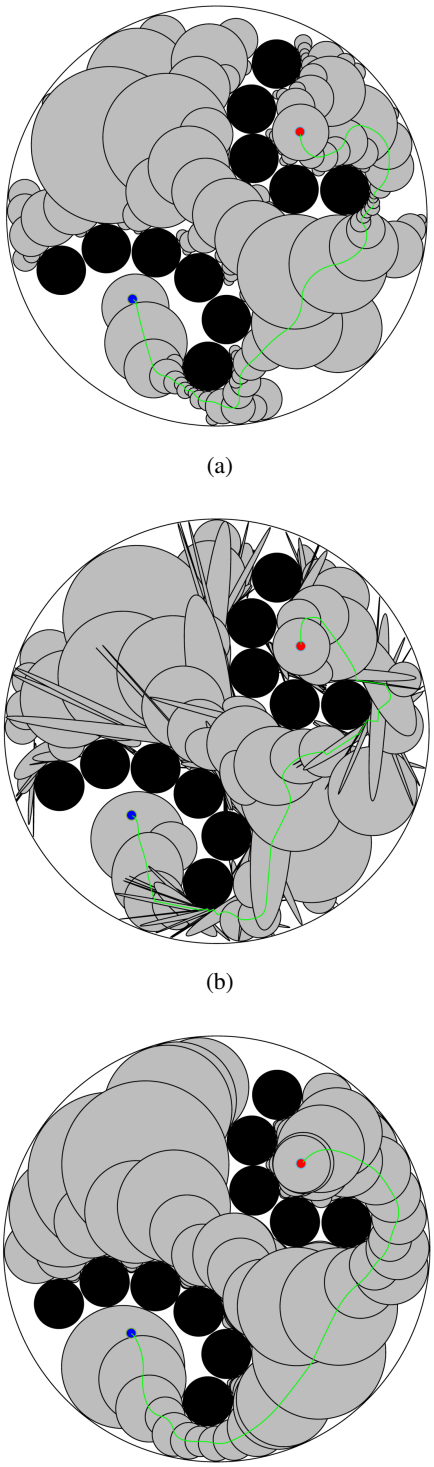


Figure 5.10: Computation performance comparison of circular, elliptic and enlarged circular nodes for scenario 2.



(a)

(b)

(c)

Figure 5.11: This figure illustrates the node styles and numbers of different node shapes for scenario 2. In (a), the algorithm uses standard circular nodes (Section 2.2). In (b), the algorithm uses elliptical nodes (Section 2.4.1). In (c), the algorithm uses enlarged circular nodes (Section 2.4.2).

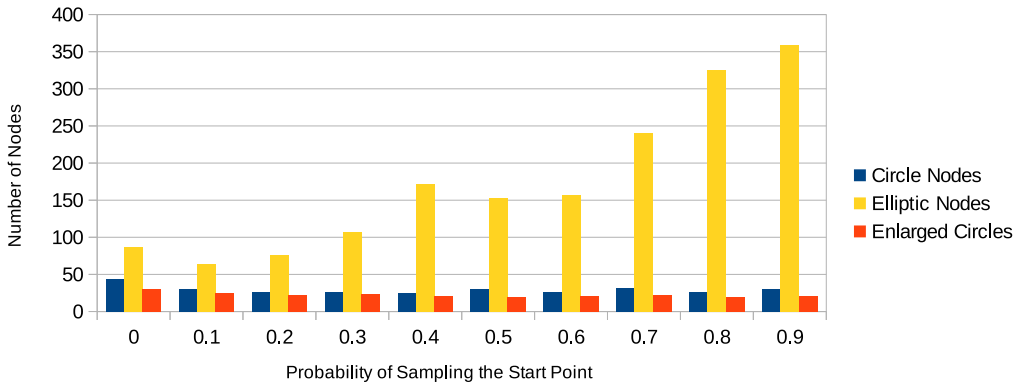


Figure 5.12: Node number comparison of circular, elliptic and enlarged circular nodes for scenario 2.

As one can see from the performance comparison figures, the performance of the elliptical enlargement method degrades in this scenario. The potential reason of this degradation is, it is hard for elliptical shapes to cover narrow passages such as the space between an obstacle and boundary. Thus, this method generates a lot of elliptical funnels to cover those small regions. In elliptical enlargement method, the funnels are stretched only in one direction and coverage of narrow regions can only be obtained with so many funnels.

5.1.1.3 Scenario 3

In this scenario we constructed an environment with great number obstacles. In Figure 5.15, three methods with same initial and goal configurations are given. As can be seen, the node shapes and numbers differ for different node models.

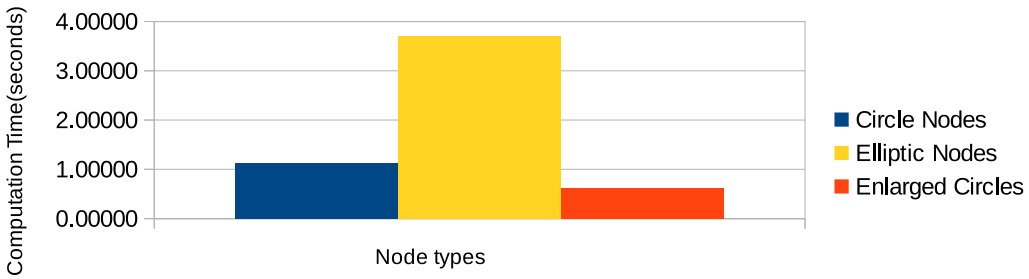
Table 5.3 lists the results from the compared methods. All tests were performed 1.000 times on the same map but with different start and goal configurations.

As can be readily seen from Table 5.3, if the probability of sampling the goal node is set to 1, the performance of both circular nodes and elliptic nodes decreases drastically. Setting the probability to 0 also decreases the performance. Figure 5.14 compares the computation performance of three node generation methods, and Figure

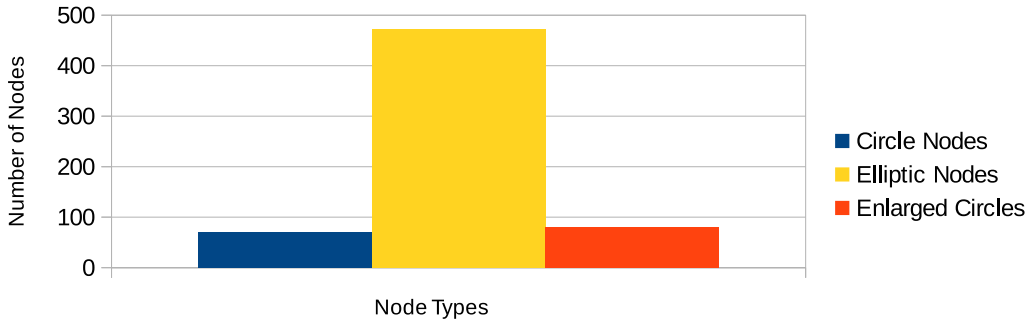
5.16 compares the numbers of generated nodes. In Figure 5.13, the average computation performances and node counts are given. Here, the outliers in the probability of sampling the goal node –0 and 1– are discarded. The sparsity enhancement methods, elliptical and circular enlargement, decreases the computation time and number of nodes.

Table 5.3: Performance results of different methods for scenario 3.

Probability	**Time (seconds)**			**Number of Nodes**		
	Circle Nodes	**Elliptic Nodes**	**Enlarged Circles**	**Circle Nodes**	**Elliptic Nodes**	**Enlarged Circles**
0	0.39462	1.25341	0.18056	99	268	92
0.1	0.34531	0.91296	0.17515	75	241	80
0.2	0.33227	1.29547	0.22814	70	271	79
0.3	0.44360	2.15521	0.21168	70	370	71
0.4	0.52551	2.09334	0.30338	70	325	79
0.5	0.57983	2.46551	0.43366	70	390	91
0.6	0.81922	3.72874	0.53514	60	464	86
0.7	0.96941	4.29960	0.79843	61	573	87
0.8	1.71260	5.87946	1.20366	69	701	83
0.9	4.44837	10.55910	1.71435	84	915	68



(a)



(b)

Figure 5.13: Comparison of computational performance and node count by method for scenario 3.

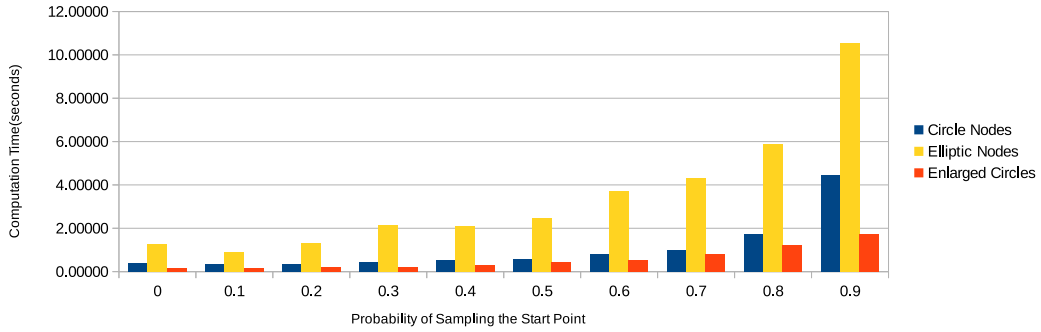
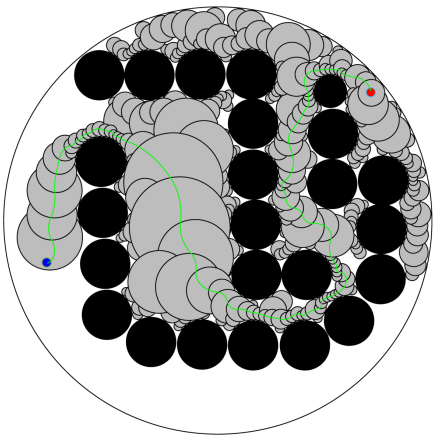
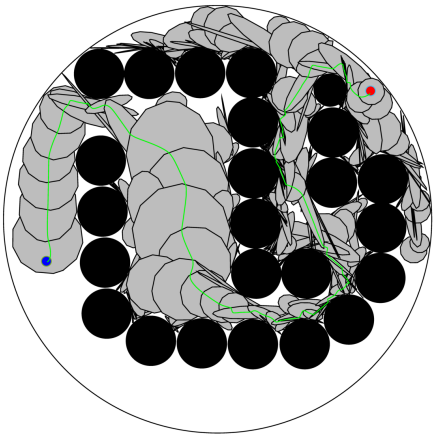


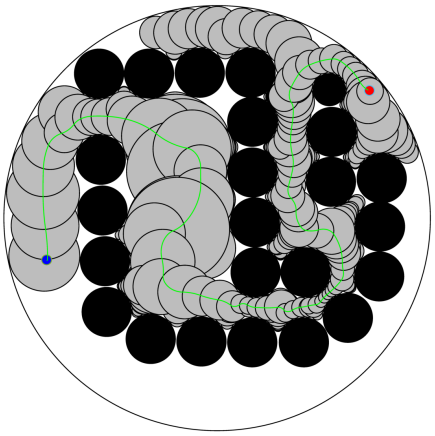
Figure 5.14: Computation performance comparison of circular, elliptic and enlarged circular nodes for scenario 3.



(a)



(b)



(c)

Figure 5.15: This figure illustrates the node styles and numbers of different node shapes for scenario 3. In (a), the algorithm uses standard circular nodes (Section 2.2). In (b), the algorithm uses elliptical nodes (Section 2.4.1). In (c), the algorithm uses enlarged circular nodes (Section 2.4.2).

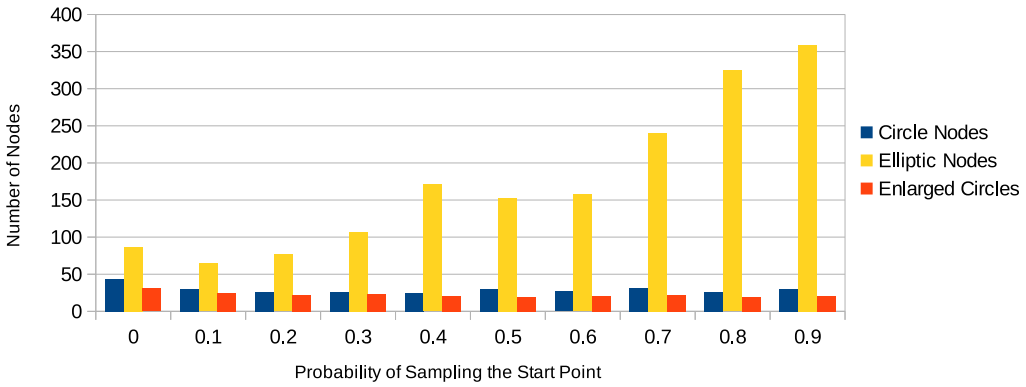


Figure 5.16: Node number comparison of circular, elliptic and enlarged circular nodes for scenario 3.

As discussed above, the performance of elliptical enlargement method degrades in this scenario too. The reason of this degradation is, again, related with the structure of obstacles.

As can be seen from the figures, the enlarged circles method gives the best performance in all scenarios. We used this method for the 3D simulation.

5.2 3D Dynamic Simulation Results

To test the robustness and full practical applicability of our approach, we performed simulated tests using the simulation infrastructure built using MATLAB and the ROS/ Gazebo package.

The first results are illustrated in Figure 5.17. The whole simulation package works based on following process. The 3D environment is first created inside MATLAB. The black cylinders in Figure 5.17 correspond to the obstacles in the environment. It can be seen that the height of some obstacles rises above sea level (as would be the case with wind turbines), while others do not. We then run our motion planning algorithm to generate sparse random funnel trees until the basis of attraction of one of the funnels covers the initial condition. We illustrate the hypothetical funnels using transparent green cylinders in Figure 5.17. After this, based on the kinematic 3D

model and the associated control policy (explained in Section 2.5), we perform a MATLAB simulation using *ode45* to find the path shown in the solid red line. At this point, our custom MATLAB code communicates with the UUV simulator and re-creates the same environment inside the fully dynamic UUV simulation package. This environment can be seen in Figure 5.18.

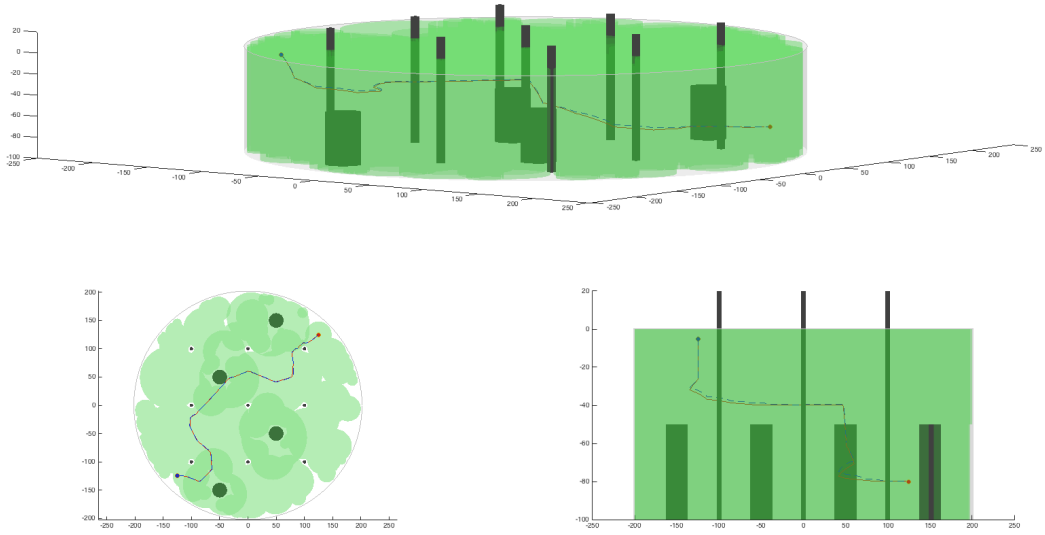


Figure 5.17: This figure illustrates the 3D simulation results of the implementation of our motion planning algorithm in a sample 3D environment. This figures includes both the kinematic simulation performed only inside MATLAB and the fully dynamic simulation package that runs on ROS/Gazebo and MATLAB. The top figure is the 3D (isometric) view of the environment. The bottom left and bottom right figures correspond to the top and side views, respectively. The black solid cylinders are the obstacles, whereas the outer cylinder is the imaginary boundary that we chose based on the tethering capabilities. The green cylinders illustrate the basins of attraction of funnels. They are most clear in the top view. The solid red path is the path generated by the kinematic model in MATLAB, whereas the dashed blue path represents the dynamic 3D simulation. The motion controller can reach to the goal state in both cases.

After the creation of the environment, the SAGA ROV and its initial conditions are initialized which triggers the closed-loop control and simulation physics. The Gazebo/ROS UUV simulator receives the reference velocity commands from the motion

controller running in MATLAB and executes the physics. The motion controller receives the states of the SAGA ROV from the Gazebo/ROS simulator and based on the funnel structure, generates sets of velocity commands. This closed-loop real-time simulation continues until the SAGA ROV reaches the goal point. The resultant 3D path is plotted with blue dashed lines in Figure 5.17. It can be seen that the motion controller is robust when subject to the uncertainties that arise from the unmodeled full 6-DOF dynamics of the SAGA ROV. The two paths are close to each other, with only small gaps in several regions. Note that the real-time dynamic simulation does not try to track the trajectory in red.

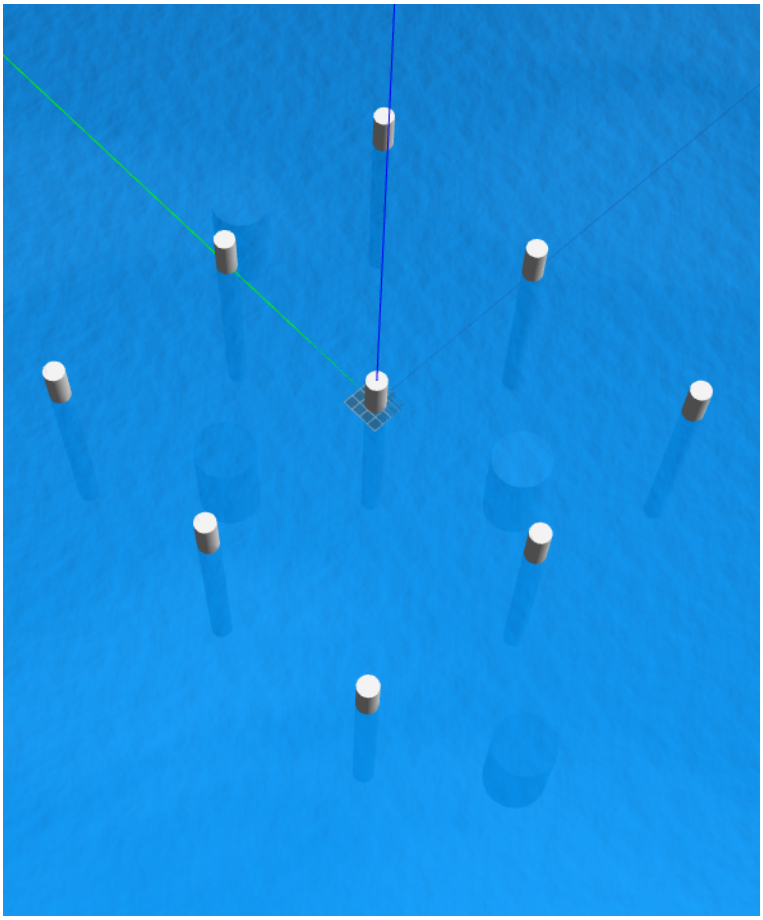


Figure 5.18: Snapshot of the Gazebo environment used for 3D simulations.

We also tested the controller under severe environmental uncertainty in the 3D dynamical simulation package. These results are illustrated in Figure 5.19. In a specific prismatic region (shown in red in Figure 5.19), we applied a water current in the di-

rection of $[1\ 0\ 0]^T$ with a strength of $1\ m/s$, with maximum forward speed in this simulation set to $2\ m/s$. It is clear that the gap between the path found by MATLAB based on the kinematic model and the path found by the dynamic simulation with external disturbances is larger inside the red region. However, due to the robustness of the feedback-based motion planning, the ROV reached the goal position successfully.

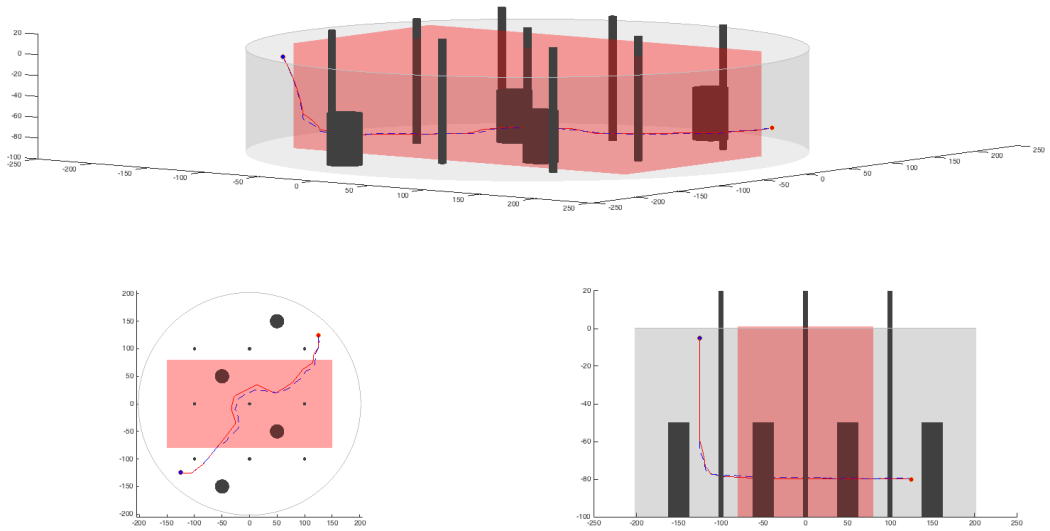


Figure 5.19: This figure illustrates the 3D simulation results with a specific environmental disturbance (water current). The environment presented in Figure 5.17 is also used here. Note that we have not included the illustration of funnels. However, the funnels and undisturbed paths are different compared to Figure 5.17, due to the probabilistic nature of our method. Even though the gaps between the paths are higher, the method robustly finds a path and reaches to the goal.

5.2.1 Performance Comparisons

In this section, the test results for different methods for the 3D environment are given. The proposed methods are probabilistic, so we performed Monte Carlo testing to assess the performance of the algorithms. All tests used the same area with same configuration of obstacles, but at each iteration the start and goal configurations were changed. An important parameter in the algorithm is the probability of sampling the goal node. This parameter tends the tree toward the goal node, which may prevent the coverage of the map. If the probability of the sampling goal node is high, it means

that the goal point is sampled frequently as a new node candidate.

Table 5.4 lists the results from different methods. All the tests were performed 1.000 times on the same map but with different start and goal configurations.

Table 5.4: Performance results of different methods in 3-D environment

Probability	**Time**		**Number of Nodes**	
	Cylinder Nodes	**Enlarged Cylinders**	**Cylinder Nodes**	**Enlarged Cylinders**
0	0.205888	0.066909	122	64
0.1	0.015793	0.010836	44	26
0.2	0.010887	0.006781	36	20
0.3	0.009072	0.00507	31	16
0.4	0.008493	0.004018	28	14
0.5	0.008453	0.003475	26	12
0.6	0.01122	0.003005	25	11
0.7	0.009971	0.002659	23	10
0.8	0.014667	0.002436	21	9
0.9	0.013953	0.002226	18	9
1	0.001235	0.001963	10	8

CHAPTER 6

CONCLUSION

The major goal of this study was to develop a new robust and computationally feasible motion planning method for a class of UUV systems. Motion and path planning is a very mature field, and there are many solutions for the problem of motion and path planning for robotic systems.

As a basis for our work, we combined different planning approaches: sampling-based planning and sequential composition. Our goal was simply to obtain an effective planning solution for a class of UUV systems by taking advantage of both planning methods and attempting to eliminate the disadvantages of each.

The keywords that we need for understanding the proposed method are robustness and computational cost. Generally, path planning studies rely on generating "open-loop" trajectories, which are not practically valuable. Open-loop trajectories generated with such planners are tracked using sets of control laws. However, this may not be a very robust solution in all cases, especially when the disturbances and uncertainties are great. In such cases, feedback control can reduce uncertainty and increase the robustness of engineering solutions. For these reasons, the idea of sequential composition of feedback-based dynamic behaviors for robotic planning proposed in the literature. RRT and other sampling-based methods are widely popular because they are relatively easy to implement and they provide fast and effective solutions from the perspective of computational costs.

As mentioned before, there are different studies which synthesized sequential composition with random sampling-based methods. However, there are some important differences and key contributions of our study, especially considering our applica-

tion domain of underwater systems. The other studies mainly attempted to develop motion control solutions for highly dynamic systems–that is, where second-order dynamics are critical. As we discussed, however, for underwater environments and the class of UUV vehicles that we consider, first-order dynamics dominate behavior. We suspect that for underwater environments, external uncertainties are more severe than possible internal dynamical uncertainties and effects. Given this main application domain difference, previously developed methods that rely on heavy optimization and numerical techniques are computationally very costly for our application.

Based on our operating conditions and specific constraints, we first proposed the basic principles of our motion planning algorithm. Indeed, this planner can be applied to a wide variety of systems with some modifications (such as the shape of funnels or different control polices). The inputs for our motion planner are the environment (with obstacles), the goal position, and the initial configuration(s). The algorithm starts by constructing a funnel around the goal position, with the basins of attraction defined in Cartesian space (a circle for 2D environments and a cylinder for 3D). Since the proposed high-level control policy has guaranteed stability and strictly remains inside the funnel, any initial condition inside this master funnel can be driven to the goal position.

The goal of randomly building a sequentially composed controller is building a funnel tree such that initial conditions inside wide regions can be eventually sent to the goal state by backtracking the sequentially attached funnels. Each funnel is controlled by similar control policies and the goal is always sending the robot to the next funnel. The simulation results show that the method successfully finds solutions that are robust and computationally cheap.

The high-level motion controller assumes a dynamic model that can be controlled with velocity inputs (forward, vertical, and angular). However, even if this assumption may be feasible under some conditions and constraints, a different low-level control policy is required to realize and test the feasibility. The second-order dynamics of the system bring some uncertainty but we showed that in a realistic, fully dynamic simulation, our feedback policy is very robust given such effects. We further tested the entire closed-loop control and planning policy under severe water flow uncertainty

and showed that the inherent robustness of the policy is effective in reaching the goal state.

REFERENCES

[1] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, "Sequential composition of dynamically dexterous robot behaviors," *The International Journal of Robotics Research*, vol. 18, no. 6, pp. 534–555, 1999.

[2] W. Bessa, M. Dutra, and E. Kreuzer, "Dynamic positioning of underwater robotic vehicles with thruster dynamics compensation," *International Journal of Advanced Robotic Systems*, vol. 10, no. 9, pp. 325–336, 2013.

[3] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[4] J. F. Canny, *The Complexity of Robot Motion Planning*. Cambridge, MA, USA: MIT Press, 1988.

[5] J. H. Reif, "Complexity of the mover's problem and generalizations," in *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, SFCS '79, (Washington, DC, USA), pp. 421–427, IEEE Computer Society, 1979.

[6] R. A. Brooks and T. Lozano-Perez, "A subdivision algorithm in configuration space for findpath with rotation," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 2, pp. 224–233, 1985.

[7] O. Khatib, "Real-time obstacle avoidance for robot manipulator and mobile robots," *The International Journal of Robotics Research*, vol. 5, pp. 90–98, 1986.

[8] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Transactions on robotics and automation*, vol. 8, no. 5, pp. 501–518, 1992.

[9] Y. Koren and J. Borenstein, "Potential field methods and their inherent limita-

tions for mobile robot navigation," in *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pp. 1398–1404, 1991.

[10] L. Kavraki and J.-C. Latombe, "Randomized preprocessing of configuration for fast path planning," in *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pp. 2138–2145, 1994.

[11] J. Moore, R. Cory, and R. Tedrake, "Robust post-stall perching with a simple fixed-wing glider using lqr-trees," *Bioinspiration & biomimetics*, vol. 9, no. 2, p. 025013, 2014.

[12] P. Reist, P. Preiswerk, and R. Tedrake, "Feedback-motion-planning with simulation-based lqr-trees," *The International Journal of Robotics Research*, vol. 35, no. 11, pp. 1393–1416, 2016.

[13] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[14] M. Brunner, B. Brüggemann, and D. Schulz, "Hierarchical rough terrain motion planning using an optimal sampling-based method," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 5539–5544, 2013.

[15] A. Ranganathan and S. Koenig, "Pdrrts: Integrating graph-based and cell-based planning," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3, pp. 2799–2806, 2004.

[16] O. Arslan and U. Saranli, "Reactive planning and control of planar spring–mass running on rough terrain," *IEEE Transactions on Robotics*, vol. 28, no. 3, pp. 567–579, 2012.

[17] D. C. Conner, H. Choset, and A. A. Rizzi, "Flow-through policies for hybrid controller synthesis applied to fully actuated systems," *IEEE Transactions on Robotics*, vol. 25, no. 1, pp. 136–146, 2009.

[18] A. A. Rizzi, J. Gowdy, and R. L. Hollis, "Distributed coordination in modular precision assembly systems," *The International Journal of Robotics Research*, vol. 20, no. 10, pp. 819–838, 2001.

[19] R. D. Gregg, T. Bretl, and M. W. Spong, "Asymptotically stable gait primitives for planning dynamic bipedal locomotion in three dimensions," in *Robotics and automation (ICRA), 2010 IEEE international conference on*, pp. 1695–1702, 2010.

[20] M. Mason, "The mechanics of manipulation," in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, vol. 2, pp. 544–548, 1985.

[21] D. E. Koditschek, "Task encoding: Toward a scientific paradigm for robot planning and control," *Robotics and autonomous systems*, vol. 9, no. 1-2, pp. 5–39, 1992.

[22] Y. Kuwata, J. Teo, S. Karaman, G. Fiore, E. Frazzoli, and J. How, "Motion planning in complex environments using closed-loop prediction," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, p. 7166, 2008.

[23] B. D. Luders, S. Karaman, E. Frazzoli, and J. P. How, "Bounds on tracking error using closed-loop rapidly-exploring random trees," in *American Control Conference (ACC), 2010*, pp. 5406–5412, 2010.

[24] Y. S. Silveira and P. J. Alsina, "A new robot path planning method based on probabilistic foam," in *2016 XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR)*, pp. 217–222, 2016.

[25] L. B. Pereira do Nascimento, D. da Silva Pereira, A. Sanca Sanca, K. J. Silva Eugenio, D. H. da Silva Fernandes, P. Javier Alsina, M. Valério Araujo, and M. Rabello Silva, "Safe path planning based on probabilistic foam for a lower limb active orthosis to overcoming an obstacle," in *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*, pp. 413–419, 2018.

[26] L. B. P. Nascimento, D. S. Pereira, P. J. Alsina, M. R. Silva, D. H. S. Fernandes, V. C. C. Roza, and A. S. Sanca, "Goal-biased probabilistic foam method for robot path planning," in *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pp. 199–204, 2018.

[27] R. Tedrake, "Lqr-trees: Feedback motion planning on sparse randomized trees," in *MIT Press*, 2009.

[28] L. Yang and S. M. Lavalle, "The sampling-based neighborhood graph: An approach to computing and executing feedback motion strategies," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 3, pp. 419–432, 2004.

[29] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "Lqr-trees: Feedback motion planning via sums-of-squares verification," *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 1038–1052, 2010.

[30] R. D. Christ and R. L. Wernli Sr, *The ROV manual: a user guide for remotely operated vehicles*. Butterworth-Heinemann, 2013.

[31] S. K. Kartal, E. Ege, and M. K. Leblebicioğlu, "Optimal autopilot and guidance of the rov: Saga," *IFAC-PapersOnLine*, vol. 49, no. 3, pp. 401–406, 2016.

[32] S. Kartal, K. Leblebicioglu, and E. Ege, "Experimental test of vision-based navigation and system identification of an unmanned underwater survey vehicle (saga) for the yaw motion," *Transactions of the Institute of Measurement and Control*, vol. 41, no. 8, pp. 2160–2170, 2019.

[33] M. M. M. Manhães, S. A. Scherer, M. Voss, L. R. Douat, and T. Rauschenbach, "Uuv simulator: A gazebo-based package for underwater intervention and multi-robot simulation," in *OCEANS 2016 MTS/IEEE Monterey*, pp. 1–8, 2016.

[34] T. Fossen, *Guidance and Control of Ocean Vehicles*. 1994.

[35] T. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*. 2011.

[36] G. Antonelli, T. Fossen, and D. Yoerger, "Modeling and control of underwater robots," pp. 1285–1306, 2016.

[37] O. Eidsvik and I. Schjølberg, "Determination of hydrodynamic parameters for remotely operated vehicles," vol. Volume 7: Ocean Engineering of *International Conference on Offshore Mechanics and Arctic Engineering*, p. 19–24, 2016.

[38] A. Spears, M. E. West, and T. R. Collins, "Autonomous control and simulation of the videoray pro iii vehicle using moos and ivp helm," *2013 OCEANS - San Diego*, pp. 1–10, 2013.

[39] S. Kartal, K. Leblebicioglu, and E. Ege, "Experimental test of the acoustic-based navigation and system identification of an unmanned underwater survey vehicle (saga)," *Transactions of the Institute of Measurement and Control*, vol. 40, pp. 2476–2487, 2018.

[40] D. Cook, A. Vardy, and R. Lewis, "A survey of auv and robot simulators for multi-vehicle operations," in *2014 IEEE/OES Autonomous Underwater Vehicles (AUV)*, pp. 1–8, 2014.

[41] M. Prats, J. Pérez, J. Fernandez, and P. Sanz, "An open source tool for simulation and supervision of underwater intervention missions," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2577–2582, 2012.

[42] G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan, "Modular open robots simulation engine: Morse," in *2011 IEEE International Conference on Robotics and Automation*, pp. 46 – 51, 2011.

[43] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, pp. 2149 – 2154, 2004.

[44] T. Watanabe, G. Neves, R. Cerqueira, T. Trocoli, M. Reis, S. Joyeux, and J. Albiez, "The rock-gazebo integration and a real-time auv simulation," in *2015 12th Latin American Robotics Symposium and 2015 3rd Brazilian Symposium on Robotics (LARS-SBR)*, pp. 132–138, 2015.

[45] D. Lee, G. Kim, D. Kim, H. Myung, and H.-T. Choi, "Vision-based object detection and tracking for autonomous navigation of underwater robots," *Ocean Engineering*, vol. 48, p. 59–68, 2012.

[46] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics and Autonomous Systems*, vol. 61, p. 1258–1276, 2013.

[47] D. R. Yoerger, J. G. Cooke, and J. J. E. Slotine, "The influence of thruster dynamics on underwater vehicle behavior and their incorporation into control system design," *IEEE Journal of Oceanic Engineering*, vol. 15, no. 3, pp. 167–178, 1990.

[48] W. Bessa, M. Dutra, and E. Kreuzer, "Dynamic positioning of underwater robotic vehicles with thruster dynamics compensation," *International Journal of Advanced Robotic Systems*, vol. 10, no. 9, pp. 325–336, 2013.

[49] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, *RotorS—A Modular Gazebo MAV Simulator Framework*. Springer International Publishing, 2016.

[50] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. von Stryk, "Comprehensive simulation of quadrotor uavs using ros and gazebo," in *Proceedings of the Third International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, SIMPAR'12, pp. 400–411, Springer-Verlag, 2012.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name	EGE, Emre
Nationality	Turkish (TC)
Date and Place	8 May 1982, Kocaeli
Phone	+90 312 210 00 01
E-mail	emreege@gmail.com

EDUCATION

Degree	**Institution**	**Year of Graduation**
MS	METU Electrical and Electronics Engineering	2007
BS	Ankara University Electronics Engineering	2004
High School	Fethiye Kemal Mumcu Anatolian Highschool	2000

WORK EXPERIENCE

Year	**Place**	**Enrollment**
2011-...	Desistek Robotics	Co-founder & CTO
2008-2013	METU Electrical and Electronics Engineering	Research Assistant
2006-2008	METU Electrical and Electronics Engineering	Project Assistant
2004-2006	AYESAS	Software Test Engineer

FOREIGN LANGUAGES

Advanced English

PUBLICATIONS

1. S. K. Kartal, M. K. Leblebicioğlu, and E. Ege, "Experimental Test of Vision-based Navigation and System Identification of an Unmanned Underwater Survey Vehicle (SAGA) for the Yaw Motion," Transactions of the Institute of Measurement and Control, vol.41, no.8, pp. 2160-2170, 2019.

2. E. Ege, and M. Ankarali, "Feedback Motion Planning of Unmanned Surface Vehicles via Random Sequential Composition," Transactions of the Institute of Measurement and Control, vol.41, no.12, pp. 3321-3330, 2019.

3. S. K. Kartal, E. Ege, and M. K. Leblebicioğlu, "Experimental Test of Acoustic-based Navigation and System Identification of an Unmanned Underwater Survey Vehicle (SAGA)," Transactions of the Institute of Measurement and Control, vol.40, no.8, pp. 2476-2487, 2018.

4. S. K. Kartal, E. Ege, and M. K. Leblebicioğlu, "Optimal Autopilot and Guidance of the ROV: SAGA," ScienceDirect, vol.49, no.3, pp. 401-406, 2016.

5. S. K. Kartal, M. K. Leblebicioğlu, and Emre Ege, "Bir İnsansız Sualtı Gözlem Aracının (SAGA) Akustik ve Görüntüleme Temelli Yer Tespiti ve Sistem Tanılaması," Bilim ve Teknoloji özel sayısı II, vol.89 no.2, pp. 44-56, 2016.

6. S. K. Kartal, M. K. Leblebicioğlu, and E. Ege,"Bir İnsansız Sualtı Gözlem Aracı (SAGA)'nın Sapma Hareketi Için Görüntü Temelli Yer Tespitinin Deneysel Testi ve Sistem Tanılaması," Otomatik Kontrol Ulusal Toplantısı (TOK 2017), Istanbul, 2017.

7. S. Y. Sizlayan, M. M. Ankaralı, and E. Ege, "İnsansız Sualtı Aracının ROS Kullanılarak Gazebo Simülatöründe Kontrolü," 7. Ulusal Savunma Uygulamaları Modelleme ve Simülasyon Konferansı (USMOS), Ankara, 2017.

8. S. K. Kartal, M. K. Leblebicioğlu, and E. Ege, "Bir İnsansız Sualtı Gözlem Aracı (SAGA) Yer Tespitinin Deneysel Testi ve Sistem Tanılaması," Otomatik Kontrol Ulusal Toplantısı (TOK), Eskişehir, 2016.

9. S. K. Kartal, M. K. Leblebicioğlu, and E. Ege, “Optimal Autopilot and Guidance of ROV: SAGA,” 14 th IFAC Symposium a Control in Transportation Systems (CTS), Istanbul, Turkey, 2016.

10. S. K. Kartal, M. K. Leblebicioğlu, and E. Ege, “Bir İnsansız Sualtı Gözlem Aracının (SAGA) Akustik ve Görüntüleme-Temelli Yer Tespiti ve Sistem Tanılaması,” 6. Ulusal Savunma Uygulamaları, Modelleme ve Simülasyon Konferansı (USMOS), 2015.

11. S. K. Kartal, M. K. Leblebicioğlu, and E. Ege, “Bir İnsansız Su Altı Gözlem Aracının Görüntü Temelli Yer Tespiti ve Sistem Tanılaması,” Sinyal İşleme ve İletişim Uygulamaları Kurultayı (SİU), 2015.

12. S. K. Kartal, M. K. Leblebicioğlu, and E. Ege, "Bir İnsansız Sualtı Gözlem Aracının Akustik Temelli Yer Tespiti ve Sistem Tanılaması," Otomatik Kontrol Ulusal Toplantısı (TOK), 2015.

13. Ç.H. Doğan, F. Y. Cevher, E. Ege, and K. Leblebicioğlu, “Taşınabilir Bir İnsansız Sualtı Aracının Mekanik Tasarımı,” Ulusal Savunma Uygulamaları Modelleme ve Simülasyon Konferansı(USMOS), Ankara, 2013.

14. E. Ege, G.K. Gültekin, and A. Saranlı, “A Statistical Predictive Model for Legged Robot Motion: Application to Pose Estimation,” Int. Conf. on Applied and Computational Mathematics (ICACM), Ankara, 2012.

15. E. Ege, and A. Saranlı, “Performance Comparison of Target Tracking Algorithms in Underwater Environment,” IEEE 16. Signal Processing and Communication Applications Conference, 2008.

16. E. Ege, and A. Saranlı, “A New Approach to Increase Performance of Rapidly-Exploring Random Trees (RRT) in Mobile Robotics,” IEEE 14. Signal Processing and Communication Applications Conference, Antalya, 2006.