FEEDBACK MOTION PLANNING OF A NOVEL FULLY ACTUATED UNMANNED SURFACE VEHICLE VIA SEQUENTIAL COMPOSITION OF RANDOM ELLIPTICAL FUNNELS

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

BY

OĞUZ ÖZDEMİR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

DECEMBER 2022

Approval of the thesis:

FEEDBACK MOTION PLANNING OF A NOVEL FULLY ACTUATED UNMANNED SURFACE VEHICLE VIA SEQUENTIAL COMPOSITION OF RANDOM ELLIPTICAL FUNNELS

submitted by **OĞUZ ÖZDEMİR** in partial fulfillment of the requirements for the degree of **Master of Science 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** ______________

Assoc. Prof. Dr. Mustafa Mert Ankaralı
Supervisor, **Electrical and Electronics Engineering, METU** ______________

Examining Committee Members:

Prof. Dr. Afşar Saranlı
Electrical and Electronics Engineering, METU ______________

Assoc. Prof. Dr. Mustafa Mert Ankaralı
Electrical and Electronics Engineering, METU ______________

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

Prof. Dr. Klaus Werner Schmidt
Electrical and Electronics Engineering, METU ______________

Assoc. Prof. Dr. İsmail Uyanık
Electrical and Electronics Engineering, Hacettepe University ______________

Date: 27.12.2022

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: Oğuz Özdemir

Signature :

ABSTRACT

FEEDBACK MOTION PLANNING OF A NOVEL FULLY ACTUATED UNMANNED SURFACE VEHICLE VIA SEQUENTIAL COMPOSITION OF RANDOM ELLIPTICAL FUNNELS

Özdemir, Oğuz
M.S., Department of Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Mustafa Mert Ankaralı

December 2022, 81 pages

This thesis proposes and analyzes a motion planning and control schema for unmanned surface vehicles that fuses sampling-based approaches' probabilistic completeness with closed-loop approaches' robustness. The proposed schema is based on the sequential composition of elliptical funnels, and it consists of two stages: tree generation and motion control. For validation of the approach, we carried out experiments using both simulation and physical setup besides the mathematical analysis. In order to have a common interface for both the simulations and the physical setup and to reduce duplication of work done, we implemented the approach as a ROS (Robot Operating System) node that can interface both similarly. Our results show that the proposed method handles the disturbances with minimal disruptions in the stability of the system. Furthermore, elliptic funnels improve the sparsity of the tree compared to the circular ones, thus, resulting in fewer mode changes.

Keywords: Motion and Path Planning, Motion Control, Sequential Composition, Unmanned Surface Vehicles

ÖZ

TAM TAHRİKLİ İNSANSIZ YÜZEY ARACININ RASSAL ELİPTİK HUNİLERİN SIRALI BİLEŞİMİ İLE GERİ BESLEMELİ HAREKET PLANLAMASI

Özdemir, Oğuz
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Mustafa Mert Ankaralı

Aralık 2022 , 81 sayfa

Tez kapsamında, insansız su üstü araçları için, örneklemeye dayalı yaklaşımların olasılıksal bütünlüğünü kapalı döngü yaklaşımların gürbüzlüğü ile birleştiren bir hareket planlama ve kontrol şeması önerilmiş ve incelenmiştir. Eliptik hunilerin sıralı bileşimine dayandırdığımız çözüm önerimiz, ağaç oluşturma ve hareket kontrolü olmak üzere iki aşamadan oluşmaktadır. Çalışmamızda, çözümümüzün geçerliliğinin gösterimi için matematiksel analizin yanı sıra hem simülasyon hem de fiziksel sistem kullanılarak deneyler gerçekleştirilmiştir. Önerilen yaklaşım, simülasyon ve fiziksel sistem testlerinde ortak arayüz kullanabilmek ve gereken iş tekrarını azaltmak için, her ikisini de benzer şekilde arayüzleyebilen bir ROS (Robot İşletim Sistemi) düğümü olarak uygulanmıştır. Sonuçlarımız, önerilen yöntemin bozucu etkenleri sistem kararlılığında minimum bozulma ile ele aldığını göstermektedir. Ayrıca, eliptik hunilerin, dairesel olanlara kıyasla ağacın seyrekliğini iyileştirerek, hareket kontrolü aşamasında daha az mod değişikliğiyle sonuçlandığı gözlenmiştir.

Anahtar Kelimeler: Hareket ve Yol Planlama, Hareket Kontrolü, Sıralı Ardışık Bileşim, İnsansız Suüstü Araçları

To my family

ACKNOWLEDGMENTS

First, I would like to express my gratitude to my supervisor Mustafa Mert Ankaralı for his support, guidance, and valuable feedback. His timely feedbacks were what filled me with motivation and energy to execute challenging tasks. He has been a great contributor to both my engineering career and research.

I want to thank Mustafa Kılınç, Ferhat Gölbol, and Atakan Durmaz for their contribution to the development of the experimental platform, which is an essential part of this thesis. I want to thank Mustafa Kılınç especially for sharing the joy and hardships during the physical experiments.

I would like to thank my colleagues at Kuartis, especially Balkar Erdoğan and Berker Loğoğlu, for their support even during my stressful times. I also thank Murat Kumru and Hilal Köksal for their genuine feedback in writing and presenting this thesis.

I would like to thank my dear friends, Onurcan Yılmaz, Uğur Açıkgöz, Burak Sevsay, Mustafa Kılınç, Sami Alperen Akgün, Seyit Yiğit Sızlayan, Ferhat Gölbol, Ramazan Akdoğan and Alperen Yaman; who were there for me when I needed them the most. I would not be able to cope with the intensive workload without them.

I am thankful to TUBITAK, The National Scientific and Technological Research Council of Turkey, for supporting me with M.S Studies scholarship. Additionally, this thesis is partially supported via TUBITAK 1001 program through projects 118E195 (initial phases) and 122E249 (final phases).

Last but not least, none of this would have been possible without my family's unconditional love and trust. I am thankful to my mother, Emriye Özdemir; my father, Ahmet Özdemir; my brother Şaban Özdemir; and my sisters, İlknur Özdemir and Çiğdem Özdemir. I can hardly express my gratitude for their love and trust in me.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vi

ACKNOWLEDGMENTS ix

TABLE OF CONTENTS x

LIST OF TABLES xiii

LIST OF FIGURES xiv

LIST OF ABBREVIATIONS xxiii

CHAPTERS

1 INTRODUCTION 1

  1.1 Motivation of the Study 1

  1.2 Literature Review 2

  1.3 Contributions 3

  1.4 The Outline of the Thesis 4

2 BACKGROUND 5

  2.1 Sampling-based Motion Planning 5

    2.1.1 Rapidly-exploring Random Trees 6

  2.2 Sequential Composition of Controllers 7

    2.2.1 Sequential Composition of Circular Funnels 9

3 METHODOLOGY 11
3.1 Tree Generation 11
3.2 Motion Control 14
3.2.1 Control Policy in the case of Circular Funnels 21
3.2.2 Stability Analysis 22
3.2.3 Practicality Changes 24
4 IMPLEMENTATION 29
4.1 Framework 29
4.1.1 ROS Concepts 29
4.2 System's Software Components 30
4.2.1 sft_server 30
4.2.2 mavros 33
5 EXPERIMENTAL SETUP 35
5.1 Experimental platform 35
5.2 Communication Channel 39
5.3 State Estimation Infrastructure 40
5.3.1 State Estimation Experiments 43
5.4 Operation Modes 46
5.4.1 Guided Control 47
5.5 Simulation Setups 51
5.5.1 Ideal Kinematic Unicycle Simulation 51
5.5.2 SITL Simulation 51
5.5.2.1 Model Used in the SITL Simulations 52

6 RESULTS 55
6.1 Motion Control Results 56
6.1.1 Single-Funnel Results 56
6.1.2 Multi-Funnel Results 60
6.2 Performance Comparison With the Circular Funnel Approach 67
6.2.1 Tree Generation Results 67
6.2.2 Motion Control Results 70
7 CONCLUSION AND FUTURE WORK 75
7.1 Conclusion 75
7.2 Future Work 76
REFERENCES 77

LIST OF TABLES

TABLES

Table 4.1 Request message of the */create_sft_latlon service*. */create_sft* service's request is identical to this, except that points are defined directly in cartesian space rather than geographic. 32

Table 4.2 Used mavros plugins and their function. 34

Table 5.1 Physical properties of the experimental platform. 38

Table 5.2 States of the EKF running on the ArduSub. Gyroscope bias represents the angular bias resulting from the integration of the gyroscope measurements, and the accelerometer bias represents the velocity bias likewise. 41

Table 5.3 FCU parameter updates regarding guided control. 49

Table 5.4 In the SITL setup, the following parameters are used for modeling the dynamics of the robot. 54

Table 6.1 Parameters used in tree generation tests, which are described in Section 3.1. 68

Table 6.2 Statistics from the tree generation executions run on scenario 1. . . . 68

Table 6.3 Statistics from the tree generation executions run on scenario 2. . . . 68

Table 6.4 Controller parameters used in performance tests. 71

LIST OF FIGURES

FIGURES

Figure 2.1 RRT expansion illustration for a holonomic point robot. q_{new} is found by applying an input to the robot in the direction of q_{rand} from q_{near}. 7

Figure 2.2 Single funnel and sequential composition of funnels illustrations gotten from [1]. Active funnel changes when the agent enters the basin of a higher-priority funnel. 8

Figure 2.3 Single complex funnel and sequential composition of simpler funnels illustrations gotten from [1]. In both of these cases, the agent is guided to the same goal configuration. 8

Figure 2.4 Funnel definition used in [2]. The cyan dashed curve is the scaled circle defined by the current distance ρ to the center. The red ray shows the reference direction determined by the ϕ angle, which is the direction directly toward the center, and α is the angle between the robot's orientation and the reference direction. 10

Figure 2.5 Example circular funnel tree gotten from [2]. The red marker shows the goal configuration whereas the green one shows the start configuration. In this figure, two example paths are also shown that are results from experiments that use different k_α. 10

Figure 3.1 Block diagram of the proposed schema. Ellipse tree block determines the active funnel from the plant's state q and outputs active funnel's state q_{funnel}. Then controller uses q and q_{funnel} to calculate reference forward velocity v and yaw rate w for the plant to move toward the active funnel's center. 11

Figure 3.2 Calculating next funnels position for $\eta = \frac{\|q_{center}q_{new}\|}{\|q_{center}q_{closest}\|} = 0.8$ and 0.6. First, a circular funnel is created that centers q_{new} and it is expanded elliptically in the direction that allows further expansion using the environmental information. 13

Figure 3.3 Tree generation illustration. In the steps from (a) to (d), the goal funnel's creation and expansion are illustrated. In the following steps, tree expansion is illustrated where new funnels are created referencing the cyan q_{tmp}'s with $\eta = 0.8$. In this realization, newly added funnels stayed circular since they have no direction that they can expand elliptically. 13

Figure 3.4 Unicycle model and reference frames. W and B represent world-fixed and base-fixed frames accordingly. Here v represents forward (i.e. surge) velocity, whereas ω represents the angular velocity. 14

Figure 3.5 Funnel definition. M is a world-fixed frame that defines the relation between funnels, whereas W is the frame centered on the active funnel's center. The cyan dashed curve is the scaled ellipse defined by the current *distance* to W calculated using (3.5) and the red dashed circle is the auxiliary circle of the scaled ellipse. The red ray shows the reference direction determined by the ϕ angle, and α is the angle between the robot's orientation and the reference direction. For circular funnels (i.e. elliptic funnels with a=1), the red ray passes through the funnel's center. 15

Figure 3.6 Change in ψ with regards to itself after α's convergence to 0. In this figure, the controller gains $k_v = 0.2$ and $k_\alpha = 2$ used and ρ assumed constant at 10. Note that the constant ρ assumption made here does not alter the convergence behavior since it only scales up and down the graph. For the case $a = 1$, the ψ term vanishes from the angular controller; thus, ψ remains constant. 22

Figure 3.7 Convergence of the simulated agent from 228 start configurations each with 19 starting position and 12 starting headings with approximately $30°$ apart at each position in funnels with $a = 1$ and $a = 6$. 23

Figure 3.8 Paths followed by the simulated USV with and without command saturation. In the case where command saturation is enabled (labeled WSat), gains $k_v = 0.2$ and $k_\alpha = 2$ are used and the resultant mission duration was 66 seconds. In the case where command saturation is disabled (labeled WoSat), gains $k_v = 0.03$ and $k_\alpha = 0.5$ are used and the resultant mission duration was 149 seconds. 25

Figure 3.9 Executed forward velocity and angular velocity commands; and the states ρ and α plotted against time. Discrete jumps on the ρ mark entries to the funnels. In this experiment, reference saturation is disabled and lower-valued gains are utilized to keep the commands in the operating range of the USV. In this execution, gains $k_v = 0.03$ and $k_\alpha = 0.5$ are used and the resultant mission duration was 149 seconds. . 26

Figure 3.10 Executed forward velocity and angular velocity commands; and the states ρ and α plotted against time. Discrete jumps on the ρ and α mark entries to the next funnels. In this experiment, reference saturation is enabled and the magnitude of the forward velocity and angular velocity commands are saturated at $0.8\,m/s$ and $0.4\,rad/s$ respectively. In this execution, gains $k_v = 0.2$ and $k_\alpha = 2$ are used and the resultant mission duration was 66 seconds. 27

Figure 4.1 Graph showing the interactions between the executables *sft_server*, *odom_creator* and *mavros*. Names inside the ellipses show the executables, whereas names in the rectangular boxes show the ROS topics and namespaces. 31

Figure 4.2 Screenshot of RViZ, showing the visualization elements published by the *sft_server*. In this screenshot, black polygons show the virtual obstacles and green markers show the funnels. 33

Figure 5.1 The platform used in the experiments. It consists of the main body, catamaran floaters on both sides and the communication assembly at the top. 35

Figure 5.2 Connection diagram of the experimental platform's and ground station's main components. 36

Figure 5.3 Connection diagram of the BlueROV2 and ground station's main components. The $FXTI$ box connected to the ground station contains the interface board and an Ethernet to USB converter, allowing Ethernet communication with the ROV from the USB port. On the FXTI's product page, the manufacturer mentions that the choice of USB is to enable both powering and communication from a single cable. 37

Figure 5.4 Motor configuration used in the experimental platform. In the illustration, the red triangle shows the forward direction, blue thrusters represent clockwise propellers and green thrusters represent counter-clockwise propellers. 38

Figure 5.5 Frame content of MAVLink v1.0 messages. *STX* byte of 0xFE marks the start of a v1.0 frame. *LEN* byte shows the length of the payload. *SEQ* is used for detecting packet losses by being incremented for each message sent. *SYSID* contains the identifier for the vehicle/system sending the message, allowing multiple vehicles/systems to coexist in the same network. *COMPID* encodes the type of the system sending the message. *MSGID* describes the message type; the payload should be decoded according to the message definition indicated by the *MSGID*. 39

Figure 5.6 Frame content of MAVLink v2.0 messages. *STX* byte of 0xFD marks the start of a v2.0 frame. Unlike MAVLink v1.0, *MSGID* is represented by 3 bytes rather than a single byte enabling far more types of messages. The incompatibility/compatibility flags indicate whether some features need to be considered when unpacking the payload. The optional signature enables authentication and increases security. 39

Figure 5.7 State estimation and control architecture used in ArduSub stack when the robot is in guided mode. The high-level control commands v and ω that are sent via MAVLink are forwarded to the guided controller by GCS_Mavlink. Sensor drivers collect raw sensor data, convert it to standard units, and store it in buffers. 42

Figure 5.8 State estimation, local position results in the stationary scenarios. 43

Figure 5.9 Local position results in the multi-turn rotation scenarios. 44

Figure 5.10 Local position results in the rectangular motion with rotation scenarios. 44

Figure 5.11 Euler angle results in the stationary scenarios. 45

Figure 5.12 Euler angle results in the multi-turn rotation scenarios. 45

Figure 5.13 Euler angle results in the rectangular motion with rotation scenarios. 46

Figure 5.14 Position control architecture in guided mode. 47

Figure 5.15 Attitude (orientation) control architecture and control allocation stage in guided mode. 47

Figure 5.16 Forward velocity response of the USV on the SITL simulation that replicates a real multi-funnel experiment with default FCU parameters. 49

Figure 5.17 Forward velocity response of the USV on the SITL simulation with updated FCU parameters. 50

Figure 5.18 Forward velocity response of the real platform with updated FCU parameters. 50

Figure 5.19 The thrust curve used in the SITL setup. There exists a dead band with a width of 50 μs at 1500 μs and ESC (Electronic Speed Controller, thruster's driver) command saturates above 1900 μs. In this model, magnitude of the thrust increases quadratically to the maximum. The thrust curve is symmetric around 1500 μs. 53

Figure 6.1 Planning arena defined over the dam lake in METU. The orange-edged polygons represent the virtual obstacles that are defined on the map to make the scenarios more challenging. 55

Figure 6.2 Controller's responses with different controller parameters on a funnel with $a = 2$ on kinematic simulation. All trajectories begin near the boundary and travel to the funnel's center. The difference between the four controllers is their k_α, while they share the same k_v. The first argument of Funnel in the legend shows the r of the ellipse, whereas the second shows the a of it. 56

Figure 6.3 Angular velocity ω and local state ρ on real experiments when $k_\alpha = 4.0$ is used starting from two distinct points. Trajectories starting from these points are also given on the left part of the figure. Trajectories labeled with 'Sim' postfix are results from the repeated executions with the ideal kinematic unicycle simulation. 57

Figure 6.4 Change in ρ when $k_\alpha = 2.0$ is used on multiple setups. Note that there existed discrete jumps on the $\dot{\rho}$ of the real results due to the use of finite difference with the ZOH'ed ρ measurements, that's why we present the moving median filtered (with a window of 0.6 seconds) version of it. 58

Figure 6.5 Angular velocity response of the USV on the lake with controller gains $k_v = 0.2$ and varying k_α with controlled initial conditions. The radius of the minimum enclosing circle for the initial positions of these recordings is less than 0.4 meters, and initial headings are within a window of $11.4°$. 58

Figure 6.6 Controllers' responses on a single funnel with different initial conditions and k_α. Experiments carried out at the lake and their results are reproduced both in SITL and kinematic simulations using the same initial conditions. Initial twists observed in some of the trajectories occur when USV's heading is outward. It rotates the robot's heading inward without increasing ρ. 59

Figure 6.7 Ground station, starting points, and goal point used in multi-funnel experiments. 60

Figure 6.8 Used funnel tree realization MF2 with simulated trajectories from arbitrary start locations. The red circle shows the goal position where the USV would reach from any other funnel. 61

Figure 6.9 Multi-funnel experiment results of the USV with $k_v = 0.2$ and $k_\alpha = 4$. The recording starts from the MF2-pt1 given in Figure 6.7. Since the local states α and ρ are defined with respect to the active funnel's center, discrete jumps exist between funnels. These discrete jumps on the local states are marked with red dashed vertical lines. Global states shown in (b) are defined relative to the right-handed frame (ENU) shown in the experiment illustration, where the x, y, and z axes are represented with red, green, and blue, respectively. 62

Figure 6.10 Multi-funnel experiment results of the USV with $k_v = 0.2$ and $k_\alpha = 2$. The recording starts from the MF1-pt1 given in Figure 6.7. Since the local states α and ρ are defined with respect to the active funnel's center, discrete jumps exist between funnels. These discrete jumps on the local states are marked with red dashed vertical lines. Global states shown in (b) are defined relative to the right-handed frame (ENU) shown in the experiment illustration, where the x, y, and z axes are represented with red, green, and blue, respectively. 63

Figure 6.11 Angular velocity responses of the experimental platform on the MF1 execution given in Figure 6.10 with varying k_α. The radius of the minimum enclosing circle for the initial positions of these recordings is less than 0.69 meters, and initial headings are within a window of $8.1°$. . 64

Figure 6.12 Path resulted from the multi-funnel experiment made with the USV on top of the used funnel tree (MF1). The scenario is repeated in SITL and SIM setups using the same initial conditions in the real experiment. The test starts from the MF1-pt1 given in Figure 6.7 and gains $k_v = 0.2$ and $k_\alpha = 2$ are used in all three realizations. A video replay of this physical experiment can be accessed at `https://youtu.be/xzBtKxv8xjM`. 65

Figure 6.13 Path resulted from the multi-funnel experiment made with the USV on top of the used funnel tree (MF2). The scenario is repeated in SITL and SIM setups using the same initial conditions in the real experiment. The test starts from the MF2-pt1 given in Figure 6.7 and gains $k_v = 0.2$ and $k_\alpha = 2$ are used in all three realizations. A video replay of this physical experiment can be accessed at `https://youtu.be/R1eT_o0o5LE`. 66

Figure 6.14 Planning scenarios defined over the lake in METU, with example circular and elliptical tree realizations on top. In the scenarios, red points show the goal positions whereas the blue points show the starting positions. 67

Figure 6.15 Histogram plots of the 50 000 trees generated each for elliptic and circular funnels on scenario 1. For ellipse and circle strategies, mean node counts are 148.20 and 219.20; and the mean start node's depths are 12.32 and 16.09 respectively. 69

Figure 6.16 Histogram plots of the 50 000 trees generated each for elliptic and circular funnels on scenario 2. For ellipse and circle strategies, mean node counts are 45.88 and 77.74; and the mean start node's depths are 18.51 and 35.19 respectively. 70

Figure 6.17 Histogram plots of the path length and average absolute yaw rate metrics for elliptic and circular funnel cases on scenario 1. For ellipse and circle strategies, mean path lengths are 76.54 and 73.20 meters; and mean average absolute yaw rates are 0.0795 and 0.0891 rad/s respectively. 71

Figure 6.18 Histogram plots of the path length and average absolute yaw rate metrics for elliptic and circular funnel cases on scenario 2. For ellipse and circle strategies, mean path lengths are 211.55 and 216.37 meters; and mean average absolute yaw rates are 0.0433 and 0.0594 rad/s respectively. The multi-peak distribution of the path length in scenario 2 is due to scenario 2 having two distinct homotopy classes of paths with comparable probabilities. 72

Figure 6.19 Paths and angular velocity responses from a real experiment carried on the damn lake in the Yalincak province of METU. Gains $k_v = 0.2$ and $k_\alpha = 2$ are used in all realizations. 73

Figure 6.20 Histogram plots of the mission duration and average speed metrics for elliptic and circular funnel cases on scenario 1. For ellipse and circle strategies, mean mission durations are 97.54 and 94.74 seconds; and mean average speeds are 0.7847 and 0.7728 m/s respectively. 74

Figure 6.21 Histogram plots of the mission duration and average speed metrics for elliptic and circular funnel cases on scenario 2. For ellipse and circle strategies, mean mission durations are 265.83 and 274.03 seconds; and mean average speeds are 0.7958 and 0.7896 m/s respectively. 74

LIST OF ABBREVIATIONS

2D	2 Dimensional
3D	3 Dimensional
AHRS	Attitude and Heading Reference System
BPL	Broadband over Power Lines
CSV	Comma Separated Value
EMCIP	European Marine Casualty Information Platform
EMSA	European Maritime Safety Agency
ENU	East, North, Up
ESC	Electronic Speed Controller
FCU	Flight Control Unit
GCS	Ground Control Station
GPS	Global Positioning System
GTE	Geometric Tools Engine
MAVLink	Micro Air Vehicle Link
METU	Middle East Technical University
MF1	Multi-Funnel Case 1
MF2	Multi-Funnel Case 2
NED	North, East, Down
RC	Radio Control
RCT	Random Circular Tree
ROS	Robot Operating System
ROV	Remotely Operated Vehicle
RTK	Real-Time Kinematic
SITL	Software in the Loop

CHAPTER 1

INTRODUCTION

1.1 Motivation of the Study

The demand for the creation of unmanned surface vehicles (USVs) and autonomous surface vehicles (ASVs) has increased in tandem with the rise in global interest in industrial, scientific, and military uses in both oceans and shallow waters [3]. ASVs are surface vessels that are capable of decision-making and autonomous operation without the aid of human guidance, navigation, or control independent from the number of passengers or crew members on board [4]. USVs on the other hand, are vehicles with no crew or passenger; that can be operated autonomously, semi-autonomously or remotely. Due to their common attachments, these terms can be used to describe the same surface vehicle.

The majority of currently used USVs are restricted to experimental platforms and primarily consist of small-scale USVs with limited autonomy, endurance, payloads, and power outputs [5]. However, increased autonomy for regular ships would also positively affect operational performance and safety. The European Maritime Safety Agency (EMSA) reports that more than 43% of all the 15481 recorded incidents and casualties between 2014-2020 in the European Marine Casualty Information Platform (EMCIP) with ships are of navigational casualties in its Annual Overview of Marine Casualties and Incidents [6]. In the same report, it is also stated that 60.6% of the investigated marine casualties in EMCIP are related to human action. From these figures, we can say that reducing human action from the operations of these vehicles can decrease accidents and reduce the frequency and impact of human mistakes. These promising benefits from the increased autonomy in surface vehicles and the need for a mature solution in this area make autonomy-increasing research areas such as motion control and path planning of surface vehicles an active research topic. For these reasons, in this thesis, we demonstrate our work on the feedback motion planning of a USV that uses the sequential composition of random elliptical funnels method.

1.2 Literature Review

Motion control and path planning are fundamental problems in robotics, where each tries to solve the problem: how to drive the robot to a target location without colliding with obstacles. In the literature, different approaches exist that propose solutions to these problems with different assumptions, limitations, and resource requirements concerning environmental representation and motion models.

The open-loop approach to this problem is to create an open-loop trajectory to be tracked by a closed-loop control strategy. Sampling-based planners such as PRM [7], RRT [8, 9], and EST [10] are well-known examples of open-loop planners. Later in the trajectory tracking stage, generated online path trackers follow a collision-free and possibly smooth path [11] [12]. Generally speaking, sampling-based planners' trajectories are not required to be smooth and may require an additional smoothing layer before the path-following stage.

The closed-loop approach to the path planning and motion control problem requires the creation of a control policy, which would bring the agent to the goal configuration from any valid initial configuration. This approach ensures that the agent would reach the goal configuration even if it diverges from the optimal trajectory due to the disturbances. The most common realizations of the closed-loop approach are based on potential functions, with control policies based on gradient descent [13]. There also exist planners based on navigation functions, which are potential functions with single stable local minima at the goal configuration [14]. Navigation function methods are immune to the local-minima problem. However, their application is limited to simple environment representations such as sphere worlds and star worlds [15], with limited real-world use. The use of harmonic functions for global navigation is another local-minima-free alternative [16] [17], but the required processing power for numerical solutions in these approaches is enormous and not suitable for real-time onboard computing. Since creating a single lightweight control policy over the whole configuration space that would forward the agent to the goal configuration while avoiding collisions is a challenging (and mostly impossible) task, hybrid approaches also exist.

Hybrid methods utilize the open-loop approach's simplicity with the closed-loop approach's robustness. Sampling-based neighborhood graph constructs a graph whose deployments probabilistically cover the configuration space in its initialization phase [18]. The constructed graph is later used to find the paths between the goal deployment and any other deployment, and local navigation functions are applied to each deployment. This approach enables the use of compositions of navigation functions in rather complex environments. Work presented in [19] follows a similar schema, but it uses Model Predictive Control (MPC) for the motion control inside the deployments. Schemas presented in [20] and [2], on the other hand, use the ideas given in [1] and [9] to create a funnel tree with a lightweight control policy defined over the funnels. Another example of a hybrid method that creates a collision-free feedback law over given cell decompositions is proposed in [21].

1.3 Contributions

The main contribution of this thesis is the development and hardware verification of the partial feedback linearized controller that guarantees safe operation inside elliptic funnels. This thesis proposes a new trajectory-free, sampling-based feedback motion planning scheme that utilizes the developed controller at its core. Thanks to the 2D polygonal map representation it is compatible with, the proposed scheme can work in real-world workspaces in which unmanned surface vehicles operate without being over-conservative.

We implemented the proposed algorithms and controller using C++ and ROS (Robot Operating System) and tested them on two different simulation setups besides the experimental platform. With clear abstraction between tree generation and motion control stages combined with the utilized language's performance, we could make extensive testing, accelerated by the open source tools and libraries.

1.4 The Outline of the Thesis

In this study, a feedback motion planning approach is proposed, deployed, and examined. In Chapter 1, we gave a brief introduction to existing motion control topologies and presented examples of them. In Chapter 2, the works and concepts crucial for a clear understanding of our approach are described in detail. Chapter 3 presents the proposed algorithm in detail, containing both the graph generation and motion control stages. Details of the proposed algorithm's implementation and architecture are presented in Chapter 4. Later in Chapter 5, we exhibit the platform on which our experiments were conducted, its low-level control architecture, and its state estimation infrastructure. This chapter also presents the results from the investigations related to the experimental platform. Chapter 6 presents the results from both the simulations and real-world experiments. The experiments in this chapter are divided into multiple sections that focus on different aspects of our approach. Finally, Chapter 7 concludes the thesis and lays out the field's future directions.

CHAPTER 2

BACKGROUND

In this thesis, we present a method that combines elements from two different approaches: Rapidly-exploring Random Trees (RRT) [22] and sequential composition of controllers [1]. In order to provide a clear and comprehensive understanding of our proposed method, we will briefly review the key concepts and ideas behind RRT and sequential composition in this chapter.

2.1 Sampling-based Motion Planning

Sampling-based planners are a type of motion planning algorithm that does not explicitly construct a representation of the obstacles in the configuration space. This makes them computationally efficient compared to other complete motion planning algorithms [23]. Instead of explicitly representing the obstacles, sampling-based planners use a collision-checking procedure to determine whether a particular configuration of the robot would be in collision with any obstacles. These planners have limited access to the configuration space, and efficient collision detection is crucial to their implementation and range of applicability.

Another key characteristic of sampling-based planners is their ability to achieve some level of completeness. A complete planner can provide a correct answer to a path-planning query in a bounded amount of time. However, complete planners are only practical for robots with up to three degrees of freedom due to their high combinatorial complexity [24, Chapter 7]. Sampling-based planners offer a weaker form of completeness, known as probabilistic completeness. Probabilistic completeness means that given enough time, the planner will eventually find a solution if it exists.

Sampling-based planners can be divided into two categories: multi-query and single-query. PRM is an example of a multi-query planning algorithm that involves a learning phase where collision-free configurations are sampled and connected to create a roadmap, a graph representation of the free configuration space Q_{free}. This roadmap can then be queried in the query phase to find a path between any two configurations. The separation between the learning and query phases of multi-query planning algorithms allows for offline computation in the learning phase, enabling a more complete representation of Q_{free}. In contrast, RRT and EST are examples of single-query planning algorithms. These planners prioritize speed in solving a specific path-planning query and do not focus on the exploration of the entire free configuration space.

2.1.1 Rapidly-exploring Random Trees

The core principle of the Rapidly-exploring Random Tree (RRT) algorithm is the utilization of a sampling strategy to direct the exploration of high-dimensional state spaces that are subject to both algebraic and differential constraints. As stated in [22], the algorithm samples points in the state space and "pulls" the search tree towards these sampled points, thus biasing the exploration towards previously unexplored regions of the space. The basic form of the RRT algorithm is given in Algorithm 1 and an example tree expansion is shown in Figure 2.1. The process begins with an initial configuration and iteratively builds a tree structure within the configuration space by sampling new configurations and expanding the tree through the use of the *Extend* function. The *Nearest* function is utilized to identify the closest node within the tree to a newly sampled configuration. The *Extend* function, in turn, is responsible for steering from the nearest node of the tree towards the newly sampled configuration.

Algorithm 1 Basic RRT Algorithm

```
$\mathcal{T}$.Init($q_{start}$)
for $i$ = 1 to $K$ do
    $q_{rand} \leftarrow$ RandomConf($\mathcal{C}$)
    $q_{near} \leftarrow$ Nearest($\mathcal{T}, q_{rand}$)
    Extend($\mathcal{T}, q_{near}, q_{rand}$)
end for
```

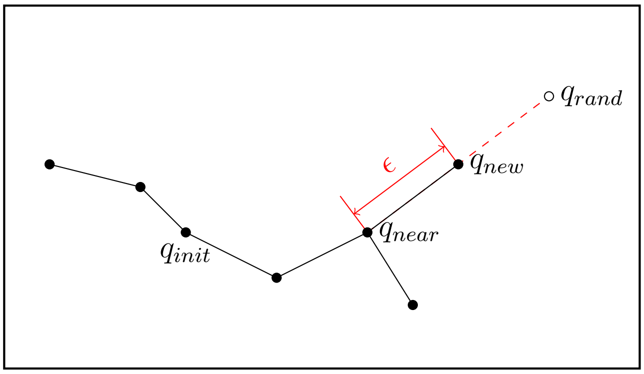


Figure 2.1: RRT expansion illustration for a holonomic point robot. q_{new} is found by applying an input to the robot in the direction of q_{rand} from q_{near}.

In the RRT algorithm, the main loop can be customized to control different aspects of the algorithm. One such aspect is the distribution of the *RandomConf*, which can be slightly biased towards the q_{goal} to improve the algorithm's convergence. However, it's important to note that biasing it too much can lead to the algorithm getting stuck in local minima, similar to potential field planners. Another aspect of the RRT algorithm to consider is the choice of the *Nearest* and *Extend* functions. These functions play an essential role in satisfying differential constraints for robots with nonholonomic constraints. By carefully selecting these functions, the algorithm can be made to handle more complex cases.

2.2 Sequential Composition of Controllers

As previously discussed, creating a single closed-loop control policy that guides the agent to the goal configuration while avoiding collisions throughout the entire configuration space is a difficult task that is, in most cases, infeasible [25]. The idea of sequential composition, introduced in [1], is that in these cases, instead of defining a single complex control policy, it is more feasible to define a sequence of simpler yet dedicated controllers that would forward the agent to the goal. The idea of funnels was first used to describe the organized, self-regulating behavior displayed by passive masses in the presence of mechanical guideways in [26]. Later, the notion of funnels was extended to feedback controllers with a basin of attraction and a stable equilibrium in [1]. Funnel and sequential composition ideas are described best by

Figure 2.2, where the metaphorical funnel is illustrated by an actual funnel-shaped structure. In the sequential composition illustration of the same figure, each funnel guides the agent to its local goal and when the agent enters the basin of a funnel with a higher priority, its controller is activated. In Figure 2.3, the sequential composition idea is illustrated in a case where obstacle-free state space is not convex and the ideal single funnel (i.e. control policy) that covers the whole obstacle-free state space is hard to define.

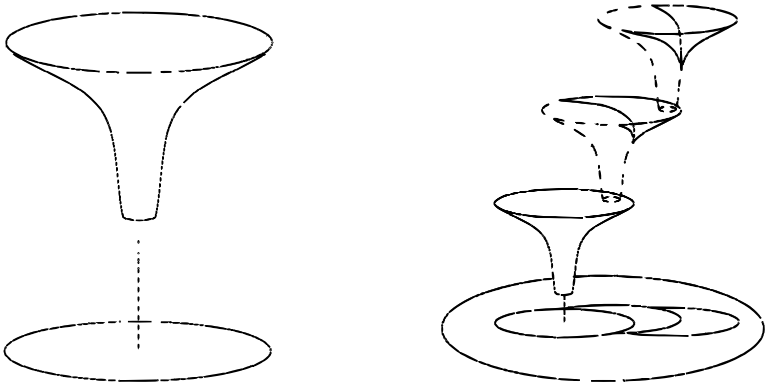


Figure 2.2: Single funnel and sequential composition of funnels illustrations gotten from [1]. Active funnel changes when the agent enters the basin of a higher-priority funnel.

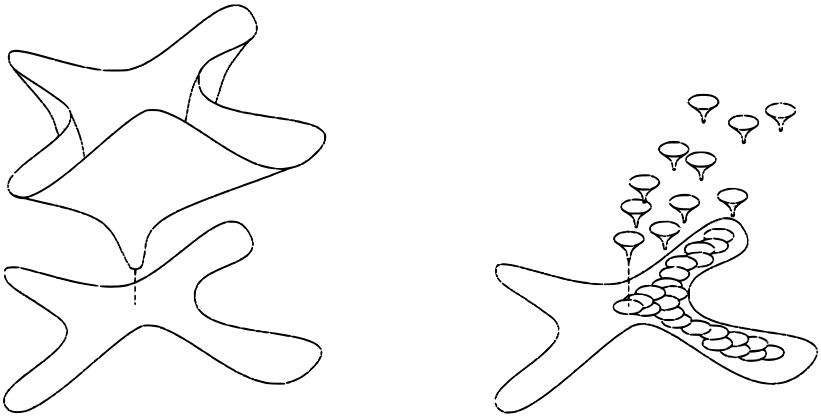


Figure 2.3: Single complex funnel and sequential composition of simpler funnels illustrations gotten from [1]. In both of these cases, the agent is guided to the same goal configuration.

The sequential composition scheme can handle small disturbances by utilizing funnels, which provide asymptotic stability for each controller and local goal. It also handles large, unexpected disturbances because it does not have a predefined state or plan. When the system's state changes to an unexpected location due to external disturbances, the controller for that specific area will activate, and the process will start again. Also, suppose the state gets to a funnel closer to the goal state or begins directly in the goal funnel. In that case, the intermediate controllers will not activate, and the system will proceed toward the goal state directly. Thus, the proposed control scheme effectively manages disturbances, provided that the said disturbances do not result in the system state departing from the union of all domains.

2.2.1 Sequential Composition of Circular Funnels

Referencing the works from [1], [22], and [18], Ege and Ankarali [2] proposed a feedback motion planning scheme for USVs with nonholonomic constraints using random sequential composition. In their work, they defined circular funnels similar to the one in Figure 2.4 where each funnel guided the unicycle-modeled agent to its center using the nonlinear control policy in (2.1). In their work, Ege and Ankarali showed that when k_α and k_ρ in (2.1) are chosen in a way that satisfies $k_\alpha > k_\rho$, then the center of the funnel is an asymptotically stable equilibrium point.

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} k_\rho \rho \cos\alpha \\ k_\alpha \alpha \end{bmatrix}. \tag{2.1}$$

In their work, the sequential composition is realized by generating a funnel tree whose root is at the goal configuration and expanding it in a fashion similar to RRT such that the new funnels have their outlets at the existing higher-priority funnels' basin. We used a very similar tree-generation algorithm in our work, with only elliptical expansion and termination condition differences. An example circular funnel tree can be seen in Figure 2.5. In their work, they also compared the sparsity of the circular funnel tree with the original RRT algorithm using Monte Carlo simulations. They observed that on average of 1000 runs, the RRT algorithm resulted in 971 nodes, whereas their algorithm resulted in 134 nodes in the map seen in Figure 2.5. This

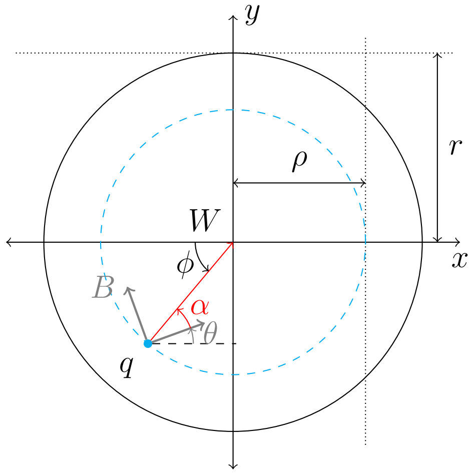


Figure 2.4: Funnel definition used in [2]. The cyan dashed curve is the scaled circle defined by the current distance ρ to the center. The red ray shows the reference direction determined by the ϕ angle, which is the direction directly toward the center, and α is the angle between the robot's orientation and the reference direction.

dramatic sparsity enhancement is a result of a single funnel covering large portions of the map, which we have inherited and improved.

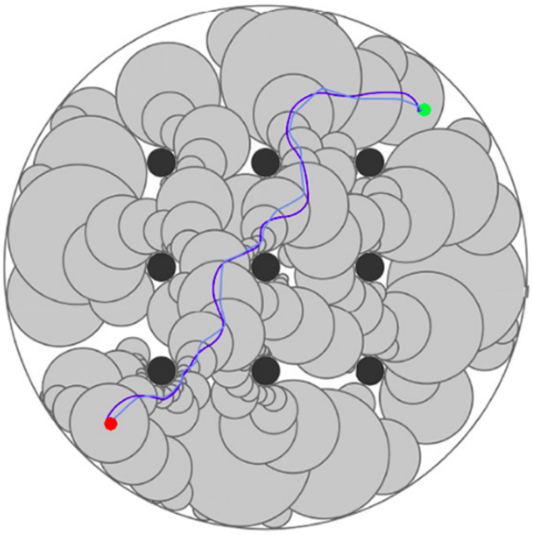


Figure 2.5: Example circular funnel tree gotten from [2]. The red marker shows the goal configuration whereas the green one shows the start configuration. In this figure, two example paths are also shown that are results from experiments that use different k_{α}.

CHAPTER 3

METHODOLOGY

The schema we propose is shown in Figure 3.1, which consists of two stages: tree generation and feedback motion control. Algorithm 2 creates an ellipse tree in the tree generation stage that starts with the goal funnel that centers the q_{goal}. Later, the ellipse tree is expanded until the termination condition is satisfied. After successfully creating the ellipse tree, the robot's motion on the active funnel is controlled via the control policy detailed in Section 3.2, which funnels any q inside the funnel boundaries to its center asymptotically without any boundary violation. Since funnels are generated on free space, this policy guarantees a collision-free motion to the next connected funnel.

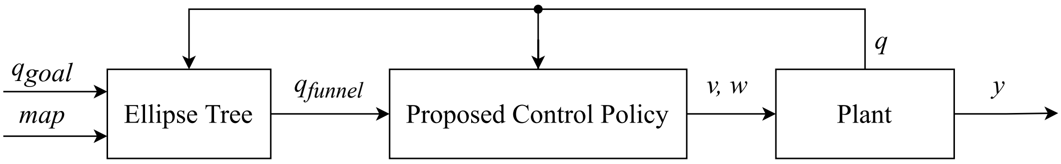


Figure 3.1: Block diagram of the proposed schema. Ellipse tree block determines the active funnel from the plant's state q and outputs active funnel's state q_{funnel}. Then controller uses q and q_{funnel} to calculate reference forward velocity v and yaw rate w for the plant to move toward the active funnel's center.

3.1 Tree Generation

In the tree generation stage, Algorithm 2 is used with inputs polygonally defined map, start position, and goal position. In the execution of the algorithm, the tree is initialized with a funnel centering q_{goal}, and the tree is expanded using random configurations until the termination condition is satisfied. In Algorithm 2, the *TerminationUnsatisfied* function has two responsibilities: checking whether q_{start} is contained in the set covered by the tree and checking whether the tree can still be extended. The approach presented in [18] is utilized to check the tree's extendability. The approach is implemented by counting successive funnel creation failures and comparing it with m_{limit} given in (3.1). The parameters P_c and β in (3.1) are user-defined parameters in the range $(0, 1)$ representing coverage confidence level and coverage fraction, respectively. This additional termination condition is added as a way to return from the algorithm on cases where there is no possible solution exists.

Algorithm 2 Ellipse-Tree Generation Algorithm

```
Input: $q_{goal}, q_{start}, map$
Output: Tree of elliptic funnels, $G$
1: $G$.Initialize($map$, $q_{goal}$)
2: while TerminationUnsatisfied($G$, $q_{start}$, $m_{limit}$) do
3:     $q_{tmp} \leftarrow$ RandomConf($map$)
4:     $V \leftarrow G$.ClosestFunnel($q_{tmp}$)
5:     $q_{new} \leftarrow V$.CalculateNextFunnelPosition($q_{tmp}$, $\eta$)
6:     $V_{new} \leftarrow G$.ConditionallyAddFunnel($map$, $q_{new}$)
7: end while
8: return $G$
```

$$m \geq \frac{\ln(1 - P_c)}{\ln \beta} - 1 = m_{limit} \, . \tag{3.1}$$

In the main body of Algorithm 2, a random configuration q_{tmp} in the map is generated, and a new funnel is conditionally added to the tree using the q_{tmp}. For executing lines 4-6, finding the closest funnel and collision checking procedures suitable for ellipses are required, which are satisfied by using distance and closest point queries from *Geometric Tools Engine* [27]. Later, q_{new} is calculated by the *CalculateNextFunnelPosition* function as in Figure 3.2. User-defined η in this function determines how much q_{new} is inside the funnel V. Lastly, the *ConditionallyAddFunnel* function creates a funnel centering q_{new}. Note that it only inserts the V_{new} to the tree G if it satisfies additional constraints defined on the size of the funnel for eliminating micro-funnels. For illustrative purposes, an example tree generation sequence is given in Figure 3.3, where the ellipse-tree is initialized and expanded.

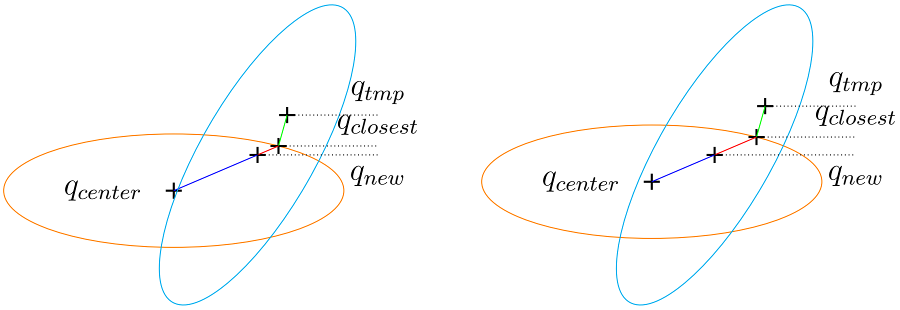
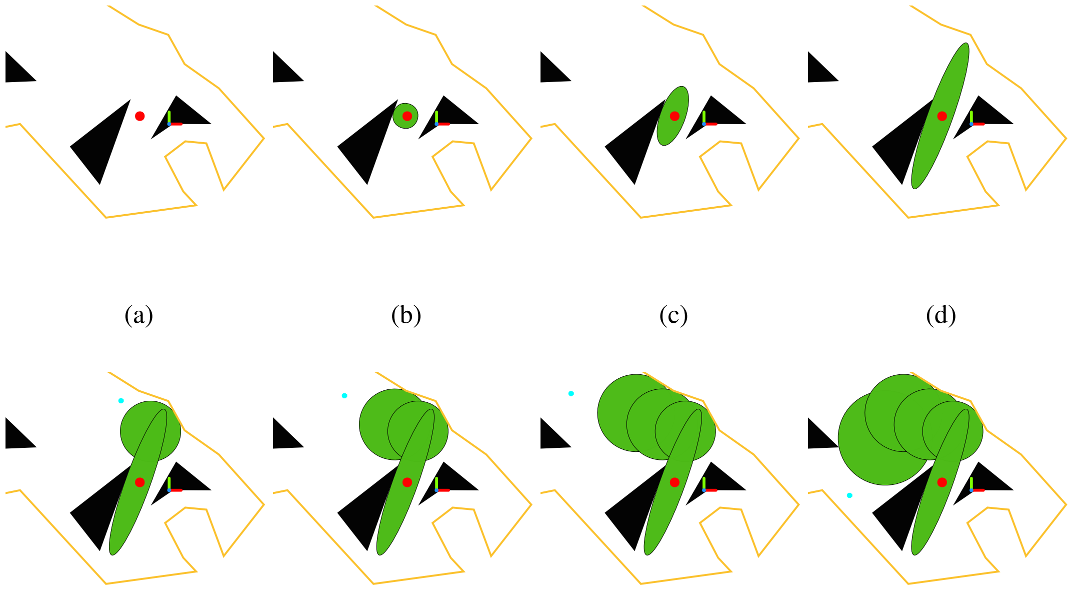


Figure 3.2: Calculating next funnels position for $\eta = \frac{\|q_{center}q_{new}\|}{\|q_{center}q_{closest}\|} = 0.8$ and 0.6. First, a circular funnel is created that centers q_{new} and it is expanded elliptically in the direction that allows further expansion using the environmental information.



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

(e) (f) (g) (h)

Figure 3.3: Tree generation illustration. In the steps from (a) to (d), the goal funnel's creation and expansion are illustrated. In the following steps, tree expansion is illustrated where new funnels are created referencing the cyan q_{tmp}'s with $\eta = 0.8$. In this realization, newly added funnels stayed circular since they have no direction that they can expand elliptically.

3.2 Motion Control

The differential drive systems mainly carry autonomous ground and water (both surface and underwater) vehicles' motion control. In high friction environments where autonomous systems operate and high bandwidth-based speed controllers are used for low-level control of these systems, the motion pattern of robotic systems in this class can be accurately captured with a unicycle model [28], [2]. State-space representation of the unicycle model can be seen in (3.2) and Figure 3.4. In this model, the state consists of the 2D position and orientation w.r.t world-fixed reference frame W. The model's inputs are forward (and backward) velocity v and yaw rate w, which are to be executed with previously mentioned high bandwidth, low-level controllers.

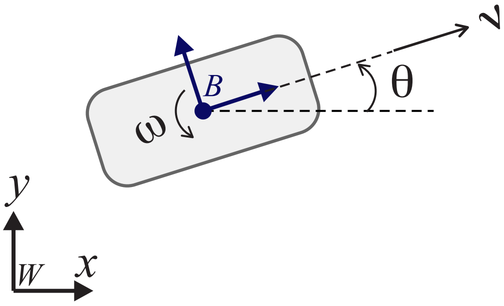


Figure 3.4: Unicycle model and reference frames. W and B represent world-fixed and base-fixed frames accordingly. Here v represents forward (i.e. surge) velocity, whereas ω represents the angular velocity.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v \\ \omega \end{bmatrix}. \tag{3.2}$$

The elliptical funnels used in our study impose different convex state-space constraints on the orbits of the robot than circular ones. For this reason, we cannot guarantee that the orbits of the model will remain in the ellipse if we utilize the previous control policies that were introduced in [28] and [2]. To be able to guarantee, a novel control algorithm is presented in this thesis.

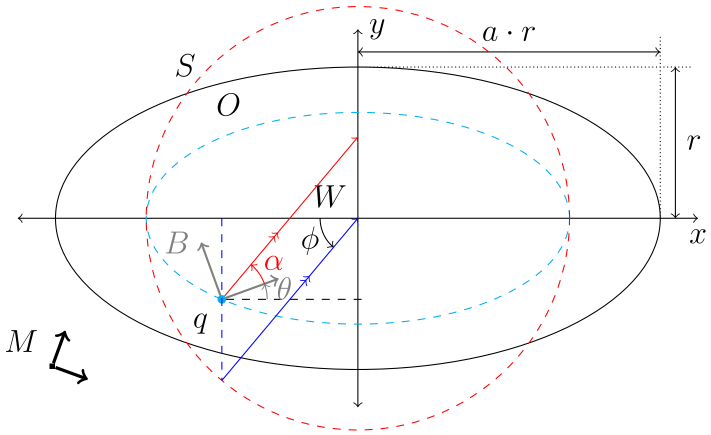


Figure 3.5: Funnel definition. M is a world-fixed frame that defines the relation between funnels, whereas W is the frame centered on the active funnel's center. The cyan dashed curve is the scaled ellipse defined by the current *distance* to W calculated using (3.5) and the red dashed circle is the auxiliary circle of the scaled ellipse. The red ray shows the reference direction determined by the ϕ angle, and α is the angle between the robot's orientation and the reference direction. For circular funnels (i.e. elliptic funnels with a=1), the red ray passes through the funnel's center.

Inspecting the elliptic region given in Figure 3.5, which has its primary axis length $2ar$ $(a > 1)$, and with the W frame defined on its center, we can define the set S formed by the points on the boundary of the elliptical region as

$$S = \left\{ q = \begin{bmatrix} x \\ y \end{bmatrix} \mid q \in R^2, \frac{x^2}{a^2} + y^2 = r^2 \right\}. \tag{3.3}$$

Similarly, we can express the "safe" set G as follows:

$$G = \left\{ q = \begin{bmatrix} x \\ y \end{bmatrix} \mid q \in R^2, \frac{x^2}{a^2} + y^2 = \rho^2 < r^2 \right\}. \tag{3.4}$$

Instead of defining the system dynamics in the Cartesian coordinate system, we will define it in a special elliptical coordinate system, which is analogous to polar coordinates in circular systems. Aicardi et al. [28] and Ege and Ankarali [2] use the Euclidean norm as the error distance (radius) because they use circular polar coordinates in their studies. Since elliptic coordinates are used in this study, and we aim to ensure that the orbits of the model stay in the defined region within the boundary ellipse, we will use a weighted norm definition suitable for elliptic geometry. In definition 3.4, it can be easily seen that the set formed for a fixed ρ defines a new ellipse within the safe area, which is a scaled-down version of the boundary ellipse with a ratio of ρ/r. In this framework, the following formula, which we extracted from (3.4) will be used as the definition of *distance*, and this variable will be the first of the state space element that we defined according to the new coordinate system.

$$\rho(x, y) = \left(\frac{x^2}{a^2} + y^2 \right)^{\frac{1}{2}} . \tag{3.5}$$

Before deducing the angular coordinates, it would be helpful to talk about the relation of this distance definition with the control algorithm. As stated earlier, the purpose of the local control policy is to drive the model asymptotically to the center of the ellipse and ensure that the orbits stay within the elliptical field at all times. While asymptotic convergence can be achieved with the following limit expression:

$$\lim_{t \to \infty} \rho(x, y) = 0 , \tag{3.6}$$

the control algorithm must meet the following condition for the robot to stay in the safe area:

$$\rho(x, y) < r , \quad \forall t \in R^+ . \tag{3.7}$$

However, the control algorithm in this thesis is developed with a more robust approach, providing the following condition, which is sufficient but not necessary, with the principle that the model never goes out of the ellipse during its movement.

$$\frac{d}{dt}\rho(x, y) \leq 0 , \quad \forall t \in R^+ . \tag{3.8}$$

If this condition is met, we will ensure that the axis lengths of the scaled ellipse surrounding the robot will never increase, thus ensuring that the robot always stays within a safe area. In the continuation of the formulation, reference frames M, W, and B, which can be seen in Figure 3.5, will be defined that will be used to describe new state variables. M reference frame is a world-fixed frame on which our positional measurements, like ones from GNSS systems, are referenced. W is a locally-fixed frame centered around the active funnel's center, and its x-axis is aligned with the major axis of the active funnel. B is a base-fixed frame attached to the robot, and its x-axis is aligned with the robot's forward direction. Next, the scaled ellipse intersecting the robot's position is created, and the dashed cyan curve visualizes it in Figure 3.5. We can define the set formed by the points on this ellipse as in (3.9), where x_B^W and y_B^W represents B frame's translation w.r.t W frame.

$$O = \left\{ q = \begin{bmatrix} x \\ y \end{bmatrix} \mid q \in R^2, \frac{x^2}{a^2} + y^2 = \left[\rho\left(x_B^W, y_B^W\right)\right]^2 \right\}. \tag{3.9}$$

We can also define ϕ as in (3.10), which shows the opposite direction of the eccentric angle of q, as shown in Figure 3.5.

$$\phi(x, y) = \operatorname{atan2}\left(-y, \frac{-x}{a}\right). \tag{3.10}$$

At this point, we aim to provide coordinate transformation by transforming the system dynamics depending on the variables (x, y) in a way that depends on the variables (ρ, ϕ) for them to be used on the controller. First, let us focus on the time derivative of the distance variable.

$$\begin{aligned}
\dot{\rho} &= \frac{d}{dt}\left(\frac{x^2}{a^2} + y^2\right)^{\frac{1}{2}} \\
&= \frac{1}{\sqrt{\frac{x^2}{a^2} + y^2}} \cdot \left(\frac{x\dot{x}}{a^2} + y\dot{y}\right) \\
&= \frac{\frac{x}{a^2}}{\sqrt{\frac{x^2}{a^2} + y^2}} \cdot \dot{x} + \frac{y}{\sqrt{\frac{x^2}{a^2} + y^2}} \cdot \dot{y} \\
&= \frac{\cos(\phi + \pi)}{a} \cdot \dot{x} + \sin(\phi + \pi) \cdot \dot{y} \\
&= -\frac{\cos\phi}{a} \cdot \dot{x} - \sin\phi \cdot \dot{y}
\end{aligned} \tag{3.11}$$

When $\dot{x}$ and $\dot{y}$ in (3.11) are replaced with the system dynamics given in (3.2), we get the following.

$$\begin{aligned}
\dot{\rho} &= -v \cdot \left[\frac{\cos\phi\cos\theta}{a} + \sin\phi\sin\theta\right] \ , \\
\dot{\rho} &= \frac{v}{2a} \cdot \left[(a-1)\cos(\phi+\theta) - (a+1)\cos(\phi-\theta)\right] \ , \\
\dot{\rho} &= -\frac{v}{2a} \cdot \left[(a+1)\cos(\phi-\theta) - (a-1)\cos(\phi+\theta)\right] \ .
\end{aligned} \tag{3.12}$$

If we were to focus on the time derivative of ϕ,

$$\begin{aligned}
\dot{\phi} &= \frac{d}{dt}\left(\operatorname{atan2}\left(-y, \frac{-x}{a}\right)\right) = \frac{a(x\dot{y} - y\dot{x})}{a^2y^2 + x^2} \ , \\
&= \frac{1}{\sqrt{\frac{x^2}{a^2} + y^2}} \cdot \left(\frac{\frac{x}{a}}{\sqrt{\frac{x^2}{a^2} + y^2}} \cdot \dot{y} - \frac{\frac{y}{a}}{\sqrt{\frac{x^2}{a^2} + y^2}} \cdot \dot{x}\right) \ , \\
&= \frac{1}{\rho} \cdot \left(\cos(\phi+\pi) \cdot \dot{y} - \frac{\sin(\phi+\pi)}{a} \cdot \dot{x}\right) \ , \\
&= \frac{1}{a\rho} \cdot \left(\sin\phi \cdot \dot{x} - a\cos\phi \cdot \dot{y}\right) \ .
\end{aligned} \tag{3.13}$$

When $\dot{x}$ and $\dot{y}$ in (3.13) are substituted with the system dynamics given in (3.2), we get the following:

$$\begin{aligned}
\dot{\phi} =& \frac{v}{a\rho} \cdot \left(\sin\phi\cos\theta - a\cos\phi\sin\theta\right) , \\
=& \frac{v}{2a\rho} \cdot \left[(\sin(\phi+\theta) + \sin(\phi-\theta)) - a \cdot (\sin(\phi+\theta) - \sin(\phi-\theta))\right] \ , \\
=& \frac{v}{2a\rho} \cdot \left[(1-a)\sin(\phi+\theta) + (1+a)\sin(\phi-\theta)\right] , \\
=& -\frac{v}{2a\rho} \cdot \left[(a-1)\sin(\phi+\theta) - (1+a)\sin(\phi-\theta)\right] .
\end{aligned} \tag{3.14}$$

As a result of these inferences, the equation of motion in the state-space form with states (ρ, ϕ, θ) and inputs (v, ω) takes the following form:

$$\begin{bmatrix} \dot{\rho} \\ \dot{\phi} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} -\frac{v}{2a} \cdot \left[(a+1)\cos(\phi-\theta) - (a-1)\cos(\phi+\theta)\right] \\ -\frac{v}{2a\rho} \cdot \left[(a-1)\sin(\phi+\theta) - (a+1)\sin(\phi-\theta)\right] \\ \omega \end{bmatrix} . \tag{3.15}$$

Before developing the controller, one more step of coordinate transformation will be performed to simplify the notation. Instead of the angular coordinates (ϕ, θ), the new angular coordinates

$$\begin{aligned} \alpha &= \phi - \theta \,, \\ \psi &= \phi + \theta \end{aligned} \tag{3.16}$$

will be used. In this context, the equations of motion in the state-space form with states (ρ, α, ψ) and inputs (v, ω) takes the form

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} -\frac{v}{2a} \cdot [(a+1)\cos\alpha - (a-1)\cos\psi] \\ -\frac{v}{2a\rho} \cdot [(a-1)\sin\psi - (a+1)\sin\alpha] - \omega \\ -\frac{v}{2a\rho} \cdot [(a-1)\sin\psi - (a+1)\sin\alpha] + \omega \end{bmatrix} . \tag{3.17}$$

The first state variable that we will consider is the distance measure ρ. It will be sufficient to satisfy the condition given in (3.7) to ensure that the robot stays in the safe area throughout its trajectory and to satisfy the limit condition given in (3.6) for the robot to converge to the center point asymptotically. To ensure that the $\dot{\rho}$ in (3.17) is always non-positive, and to cancel out a on the denominator that decreases $\dot{\rho}$ on long ellipses, v is chosen as in (3.18), where k_v is a positive constant control parameter.

$$v = k_v a\rho \cdot [(a+1)\cos\alpha - (a-1)\cos\psi] \ . \tag{3.18}$$

When control policy for the forward velocity is chosen as in the (3.18), $\dot{\rho}$ becomes

$$\dot{\rho} = -\frac{k_v \rho}{2} \cdot [(a+1)\cos\alpha - (a-1)\cos\psi]^2 \ . \tag{3.19}$$

Proposition 1. *If the initial condition of the model is within the safe area defined by the outer ellipse, it is guaranteed that the orbits always remain within bounds.*

Proposition 1 can be formulated as in the (3.20), where set G defines safe set of states as defined in (3.4).

$$(x(0), y(0)) \in G \Longrightarrow (x(t), y(t)) \in G\,, \quad \forall t \in R^+ \tag{3.20}$$

Proof. If and only if a $(x(t), y(t))$ pair is within the arena defined by G, it satisfies

$$(x(t), y(t)) \in G \Longleftrightarrow \rho(t) \leq r\,. \tag{3.21}$$

Assuming the initial state $(x(0), y(0))$ satisfies the (3.21), if we are to inspect time integral of $\dot{\rho}$,

$$\begin{aligned}
&\xi(t) = [(a+1)\cos\alpha - (a-1)\cos\psi]^2 \\
&\rho(t) = \rho(0) - \frac{k_v}{2}\int_0^t \xi(\tau)\rho(\tau)d\tau \\
&\xi(t) \geq 0 \Longrightarrow \int_0^t \xi(\tau)\rho(\tau)d\tau \geq 0 \Longrightarrow \rho(t) \leq \rho(0) \\
&\rho(t) \leq \rho(0) \Longrightarrow \rho(t) \leq r\,, \quad \forall t \in R^+ \\
&\rho(t) \leq r\,, \quad \forall t \in R^+ \Longrightarrow (x(t), y(t)) \in G\,, \quad \forall t \in R^+ \quad \square
\end{aligned} \tag{3.22}$$

Thus, we have proven the proposition and that the control algorithm will ensure that the robot's trajectory will always remain within the safe area.

The next state variable we will consider is α since α measures how much the robot's instantaneous linear velocity deviates from the target velocity. In order to linearize $\dot{\alpha}$ and stably control α, ω is chosen as

$$\omega = k_\alpha\alpha - \frac{v}{2a\rho}\cdot[(a-1)\sin\psi - (a+1)\sin\alpha]]\,. \tag{3.23}$$

In (3.23), k_α is a positive constant control parameter. When the control policy for the yaw rate given in (3.23) is also utilized, system dynamics becomes as in (3.24) and (3.25). The angular controller is mainly used for regulating the α, however after its convergence; it also pushes ψ to its stable equilibrium 0.

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \end{bmatrix} = \begin{bmatrix} -\frac{k_v\rho}{2}\cdot[(a+1)\cos\alpha - (a-1)\cos\psi]^2 \\ -k_\alpha\alpha \end{bmatrix}\,. \tag{3.24}$$

$$
\begin{aligned}
\dot{\psi} &= k_{\alpha}\alpha - \frac{v}{a\rho} \cdot \left[(a-1)\sin\psi - (a+1)\sin\alpha\right], \\
\dot{\psi} &= k_{\alpha}\alpha - k_{v} \cdot \left[(a+1)\cos\alpha - (a-1)\cos\psi\right] \\
&\qquad \cdot\left[(a-1)\sin\psi - (a+1)\sin\alpha\right]], \\
\dot{\psi} &= k_{\alpha}\alpha - \frac{k_{v}}{2} \cdot \left[2 \cdot (a^{2}-1) sin(\alpha+\psi)\right. \\
&\qquad - (a+1)^{2}\sin(2\alpha) \\
&\qquad \left. - (a-1)^{2}\sin(2\psi)\right].
\end{aligned}
\tag{3.25}
$$

3.2.1 Control Policy in the case of Circular Funnels

In the tree generation stage, generated funnels can remain a circle after elliptic expansion if they collide with the obstacles. In this subsection, we will analyze the control policy for the case where the elliptic scale a of the funnel is equal to 1 and compare it with the control policy in [2]. In this case, base states restated in (3.26) becomes as in (3.27) with derived states still as in (3.28).

$$
\begin{aligned}
\rho(x,y) &= \left(\frac{x^{2}}{a^{2}} + y^{2}\right)^{\frac{1}{2}}, \\
\phi(x,y) &= \operatorname{atan2}\left(-y, \frac{-x}{a}\right).
\end{aligned}
\tag{3.26}
$$

$$
\begin{aligned}
\rho(x,y) &= \left(x^{2} + y^{2}\right)^{\frac{1}{2}}, \\
\phi(x,y) &= \operatorname{atan2}\left(-y, -x\right).
\end{aligned}
\tag{3.27}
$$

$$
\begin{aligned}
\alpha &= \phi - \theta, \\
\psi &= \phi + \theta.
\end{aligned}
\tag{3.28}
$$

With $a = 1$, our control policy restated in (3.29) reduces to one in (3.30). Additionally, when $a = 1$; states ρ and α become the same with controlled states defined in [2], which utilizes the nonlinear control policy in (2.1).

$$
\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} k_{v} a\rho \cdot \left[(a+1)\cos\alpha - (a-1)\cos\psi\right] \\ k_{\alpha}\alpha - \frac{v}{2a\rho} \cdot \left[(a-1)\sin\psi - (a+1)\sin\alpha\right] \end{bmatrix}.
\tag{3.29}
$$

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} 2k_v \rho \cos\alpha \\ k_\alpha \alpha + \frac{v}{\rho} \sin\alpha \end{bmatrix}. \tag{3.30}$$

Inspecting the differences between control policies in (3.30) and (2.1), we can see that the control command for the forward velocity v is the same when $k_\rho = 2k_v$. The other difference is that in the proposed controller, ω has additional feedback linearizing term that linearizes $\dot{\alpha}$ in the system dynamics.

3.2.2 Stability Analysis

Inspecting the system dynamics given in (3.24), we can easily see that $\dot{\alpha}$ is only dependent on α and asymptotically stable. By design, $\dot{\rho}$ is non-increasing and certainly decreasing as α converges. The angular velocity ω's controller is designed mainly for regulating α. However, after the α's convergence, for funnels with $a > 1$, ψ term in the angular controller becomes active. It pushes the USV's heading slowly in the direction that makes ψ converge to its stable equilibrium 0. This phenomenon can also be seen in Figure 3.6, where ψ converges to 0 from both directions, except at the unstable equilibriums at $\pm\pi$.

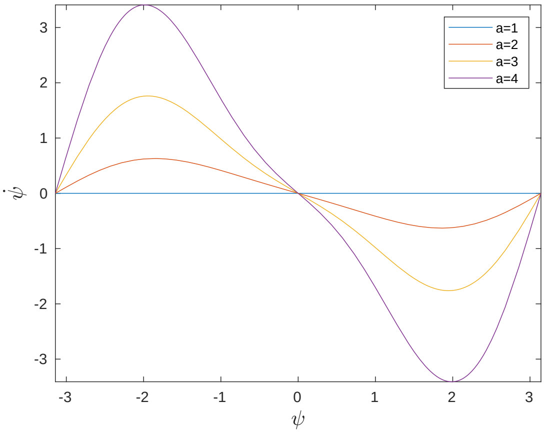
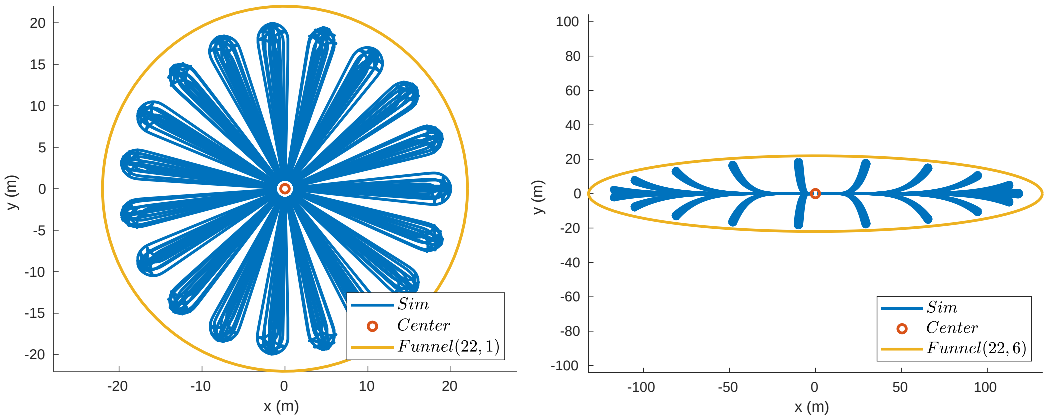


Figure 3.6: Change in ψ with regards to itself after α's convergence to 0. In this figure, the controller gains $k_v = 0.2$ and $k_\alpha = 2$ used and ρ assumed constant at 10. Note that the constant ρ assumption made here does not alter the convergence behavior since it only scales up and down the graph. For the case $a = 1$, the ψ term vanishes from the angular controller; thus, ψ remains constant.

Following the implications in (3.31), we can see that ψ converges to 2ϕ. When combined with ψ's stable equilibrium information, it is easy to conclude that the agent would favor convergence to the funnel's center from $\phi = 0$ and $\phi = \pi$ directions.

$$\begin{aligned} & \lim_{t\to\infty} \alpha = 0\,, \\ \implies & \lim_{t\to\infty} \phi - \theta = 0\,, \\ \implies & \lim_{t\to\infty} \theta = \phi\,, \\ \implies & \lim_{t\to\infty} \psi = \lim_{t\to\infty} \phi + \theta = 2\phi\,. \end{aligned} \tag{3.31}$$

To also inspect the convergence behavior experimentally, we created simulated experiments, in which the agent is started from various positions and headings inside the funnels with varying a, and all the states are recorded. Paths followed in two of these experiments can be seen in Figure 3.7. Inspecting them, we can verify that for the circular funnel case, the agent does not favor any direction of arrival, whereas, for the elliptical one, the agent tries to arrive from the directions 0 or π.



(a) Funnel with $a = 1$.

(b) Funnel with $a = 6$.

Figure 3.7: Convergence of the simulated agent from 228 start configurations each with 19 starting position and 12 starting headings with approximately $30°$ apart at each position in funnels with $a = 1$ and $a = 6$.

In the experiments made with funnels with elliptic scales $a = 2, 4, 10$, the results were consistent with the ones provided here. Funnels with lower a converge slower to the equilibrium direction, whereas funnels with higher a tend to converge to the equilibrium direction first and converge to the equilibrium position later.

3.2.3 Practicality Changes

In the implementation of the proposed controller, some improvements are made related to its usability. These changes include reference saturation and the manual handling of the arrival to the goal configuration. In this section, these changes will be presented in detail.

Considering the use of reference saturation, in Figure 3.9, we can see the output of the control policy in the scenario in Figure 3.8. Observing the operation of the controller, we can see that the robot slows down at each funnel while getting closer to its center. From a practical point of view, this behavior increases the mission duration and might not be desired. Increasing k_v is an option for decreasing mission duration, but it would result in reference velocities far out of USV's operational range when entering new funnels. In order to overcome this limitation, we implemented reference saturation to limit commanded v and ω to USV's operational range. The method we used involves limiting v and using saturated v for the calculation of linearizing terms of ω before forwarding them to the robot's low-level controllers. To allow such an approach, v in (3.23) is not substituted with its closed form but handled as it is. With the help of reference saturation, k_v can be increased to a point where USV always moves at saturation speed, excluding the termination. Thus, improving mission duration dramatically without destabilizing α. In Figure 3.10, executed control commands and resultant ρ and α are shown against time in a case where increased gains are used with reference saturation. In this experiment, mission duration is improved by more than 50%.

Another usability change is related to handling the USV's arrival to the goal configuration. Since we are using the angle to the origin while calculating the states α and ψ, and utilizing ρ at the denominator of ω's linearizing term; there exists a singularity at the origin. In regular funnels, this is not an issue since before reaching the center of the funnel, USV enters the next funnel. However, when the goal funnel is reached, there is no next funnel, and the agent can be affected by the singularity. To circumvent this singularity, we manually check arrival to the goal configuration inside the

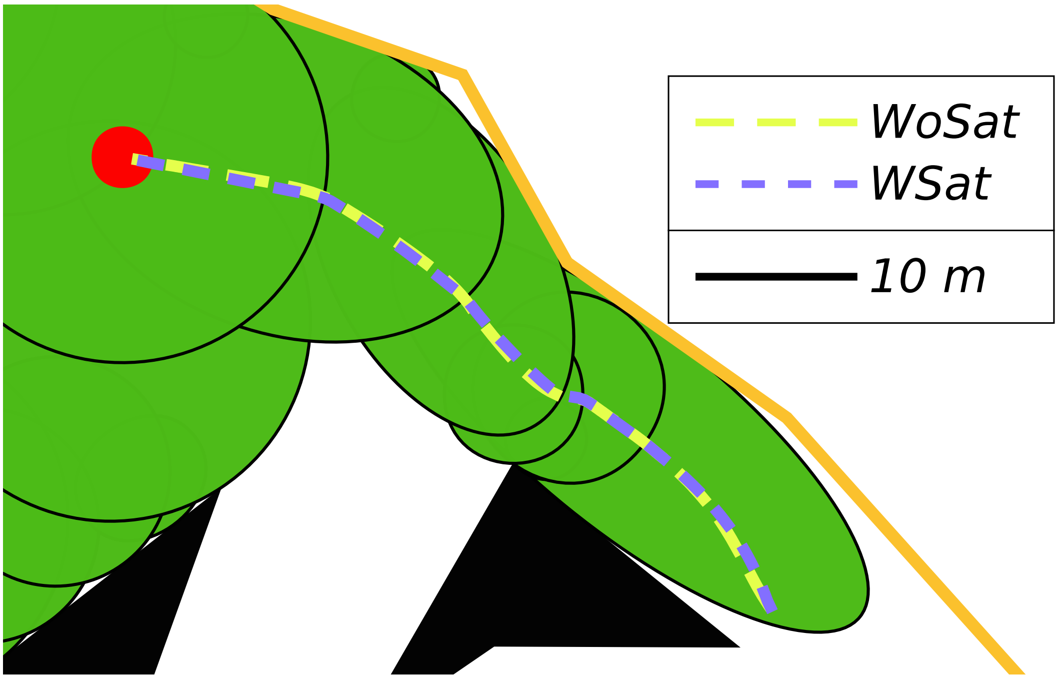
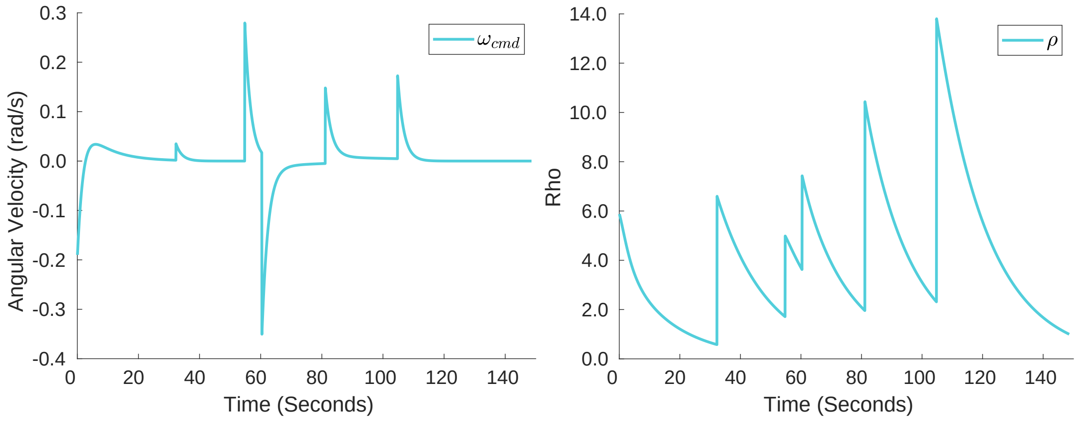


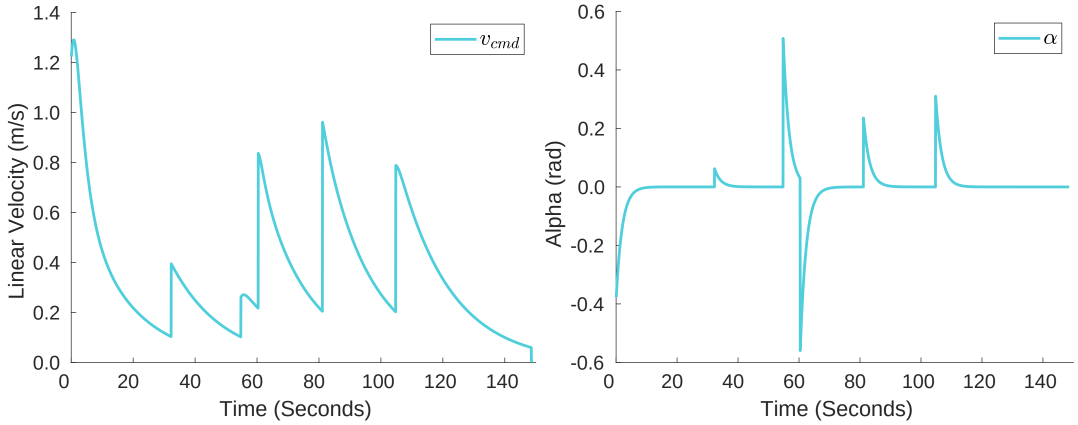
Figure 3.8: Paths followed by the simulated USV with and without command saturation. In the case where command saturation is enabled (labeled WSat), gains $k_v = 0.2$ and $k_\alpha = 2$ are used and the resultant mission duration was 66 seconds. In the case where command saturation is disabled (labeled WoSat), gains $k_v = 0.03$ and $k_\alpha = 0.5$ are used and the resultant mission duration was 149 seconds.

goal funnel by comparing ρ with a parametric arrival threshold. When ρ falls below the arrival threshold, stopping commands are sent instead of the control commands calculated by the control policy, halting the agent before it arrives at the center, thus, eluding the singularity.



(a) Angular velocity ω with respect to time.

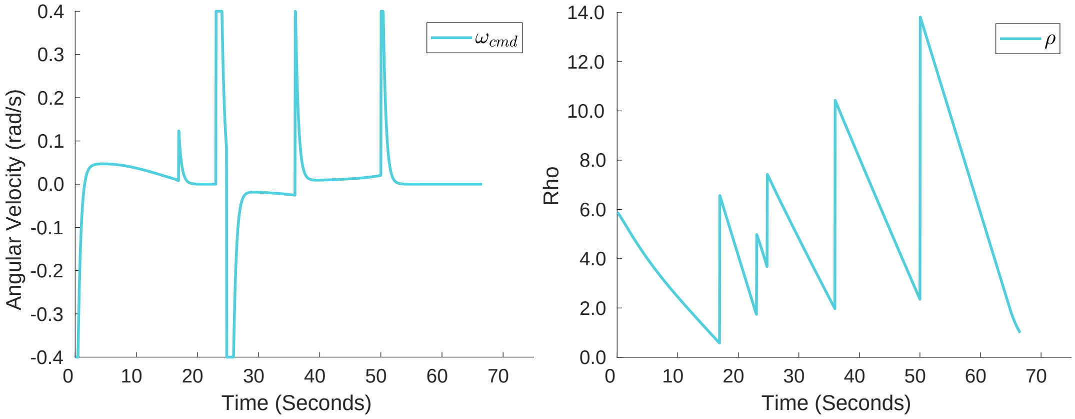
(b) USV's local state ρ with respect time.



(c) Linear Velocity v with respect to time.

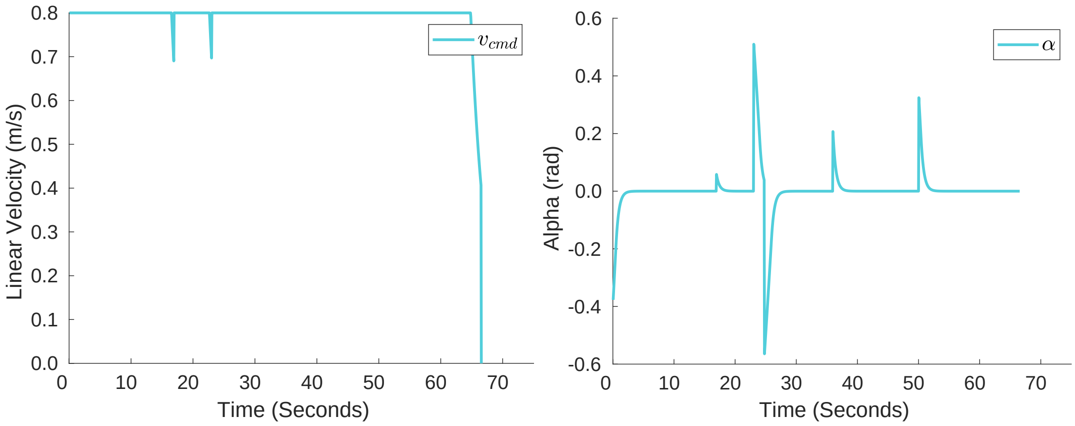
(d) USV's local state α with respect to time.

Figure 3.9: Executed forward velocity and angular velocity commands; and the states ρ and α plotted against time. Discrete jumps on the ρ mark entries to the funnels. In this experiment, reference saturation is disabled and lower-valued gains are utilized to keep the commands in the operating range of the USV. In this execution, gains $k_v = 0.03$ and $k_\alpha = 0.5$ are used and the resultant mission duration was 149 seconds.



(a) Angular velocity ω with respect to time.

(b) USV's local state ρ with respect time.



(c) Linear Velocity v with respect to time.

(d) USV's local state α with respect to time.

Figure 3.10: Executed forward velocity and angular velocity commands; and the states ρ and α plotted against time. Discrete jumps on the ρ and α mark entries to the next funnels. In this experiment, reference saturation is enabled and the magnitude of the forward velocity and angular velocity commands are saturated at $0.8\,m/s$ and $0.4\,rad/s$ respectively. In this execution, gains $k_v = 0.2$ and $k_\alpha = 2$ are used and the resultant mission duration was 66 seconds.

CHAPTER 4

IMPLEMENTATION

4.1 Framework

For implementing our motion planning and control algorithm, we used Robot Operating System (ROS) framework for the flexibility it creates, thanks to its loosely coupled architecture. This architecture in ROS is realized by publishing and subscribing to ROS messages for periodic data sharing between ROS nodes. In ROS, there also exists request/response use of ROS services for query forwarding, which is suitable for commands and planning requests. This loose coupling allows multiple executables to share responsibilities such as visualization, communication, and execution, simplifying system architecture and implementation. Additionally, ROS's extensive visualization, debugging, and data recording/replaying tools make it a perfect choice as a framework.

4.1.1 ROS Concepts

In this part, ROS concepts will be inspected more thoroughly referencing [29] and [30]. ROS Nodes are executables that perform work. A robot control system is typically made up of multiple nodes that handle multiple layers of work. They communicate/interact with one another through messages/services. A ROS message is the data structure that can be passed between nodes. Messages can be of standard primitive types, arrays of primitive types, or other types. Similarly, a ROS service is defined by two message structures, one for the request and one for the response. A serving node provides a service under a name, and a client utilizes it by sending a request message and waiting for a response.

Topic is the identifier of the message channel between nodes for a specific message. Messages are published to and subscribed from a topic. A node broadcasts a message by publishing it to a specific topic. Another node interested in that type of information will subscribe to the topic. A single topic may have numerous concurrent publishers and subscribers. Another ROS concept is the ROS master. It is the enabler of the nodes' communications. Each node communicates with the master and reports the topics it publishes and the topics it wants to subscribe to. ROS master matches publishers with subscribers and servers with clients, later for them to communicate.

The last ROS concept that we will cover is ROS Packages. Packages are the primary units for organizing software in ROS. They can contain nodes, libraries, message and service definitions, launch files, etc. Packages make it possible to reference enclosed nodes, parameter files, or any other resource relative to the package, making it easier to work with the contained.

4.2 System's Software Components

Responsibilities of the system are distributed between multiple ROS executables using the primitives mentioned in Section 4.1. This section will detail the software components that make up the system. Ellipses labeled in Figure 4.1 show the functional nodes, and the rectangular boxes show the topics exchanged between the nodes. The responsibilities of the three nodes are as follows. *Sft_server* implements our approach, *mavros* handles the communication with the USV, and *odom_creator* uses reported orientation, geographic coordinates, and the requested datum (reference geographic coordinates) to publish the USV's state in the cartesian coordinate frame defined by the datum for it to be used in *sft_server*.

4.2.1 sft_server

Sft_server is a ROS node that is a part of a ROS package named after it. Its primary responsibility is to create the funnel tree in the requested map using Algorithm 2 and periodically execute the control policy derived in Section 3.2. *Sft_server*'s primary interface with the user is via the advertised ROS services */create_sft* and

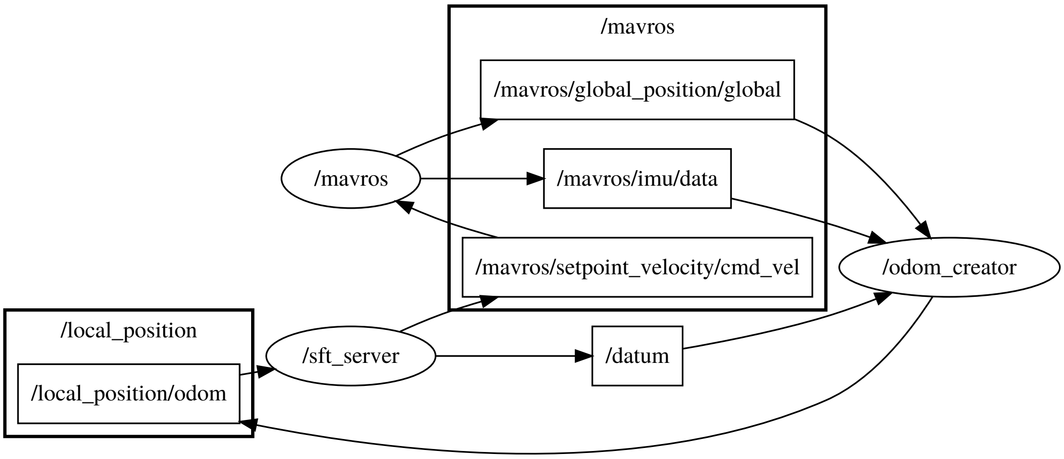


Figure 4.1: Graph showing the interactions between the executables *sft_server*, *odom_creator* and *mavros*. Names inside the ellipses show the executables, whereas names in the rectangular boxes show the ROS topics and namespaces.

/create_sft_latlon. When a */create_sft_latlon* request is received with the information in Table 4.1, *sft_server* returns the created funnel tree's CSV representation along with some statistics such as the number of nodes and the time elapsed while generating the tree. If returned CSV is saved, *sft_server* can import the funnel tree from the CSV file since it contains all information required for the tree's reproduction. The */create_sft* service works identical to the */create_sft_latlon* service except in the former, arena and obstacles are defined directly in the cartesian space. Another feature of the node is that it can also be configured to create circular funnels, enabling us to compare elliptic funnels with circular ones.

For convenience, *sft_server* subscribes to the goal updates from the RViZ (ROS visualization software) and re-generates the tree with the updated goal using the same map defined by the latest request. *Sft_server* also publishes visualization elements that illustrate the map, the tree generated on it, and the active funnel. Visualizations can be seen in Figure 4.2, showing the active funnel and ϕ angle from the current location of the USV.

Additional open-source libraries and tools besides ROS are utilized in implementing the *sft_server* node. For example, *GeographicLib* [31] is utilized for projecting geographic coordinates to the desired local cartesian reference frame, which is defined

Table 4.1: Request message of the */create_sft_latlon service*. */create_sft* service's request is identical to this, except that points are defined directly in cartesian space rather than geographic.

Type	**Signal**	**Description**
string	frame_id	Frame that the positions are defined
geographic_msgs/GeoPoint	datum	Origin that will be used for cartesian projection of the coordinates
geographic_msgs/GeoPoint	startPosition	Position of the starting point
geographic_msgs/GeoPoint	goalPosition	Position of the goal point
sft_server/GeoPolygon	arena	Arena as a non-intersecting polygon
sft_server/GeoPolygon[]	obstacles	Obstacles as a list of non-intersecting polygons
float64	eta	The η used in Algorithm 2
float64	resolution	Discretization resolution while creating visualizations, in meters
float64	minRadius	Minimum radius for a funnel to be created, in meters

by the provided datum.

The *Geometric Tools Engine* (GTE) [27] is another open-source library used in the implementation of the *sft_server* node. GTE is a collection of C++ sources that implements distance and intersection calculations between geometric primitives such as points, lines, segments, ellipses, and boxes. It also contains many more functionalities, such as curve-fitting, shape-fitting, and containment calculations. In this work, GTE is mainly used for collision checking of elliptic funnels with the obstacles and finding points' distances to the funnels in the tree generation stage of the scheme. In these operations, GTE's 3D segment and 3D ellipsoid intersection query *IntrSegment3Ellipsoid3* is used in 2D by zero-initializing their z-dimension for collision checking. For distance calculations, GTE's N-dimensional point to hyper-ellipsoid distance query *DistPointHyperellipsoid* is utilized in 2D mode. This work focuses on 2D surface operation, but the *Geometric Tools Engine*'s 3D capabilities for these

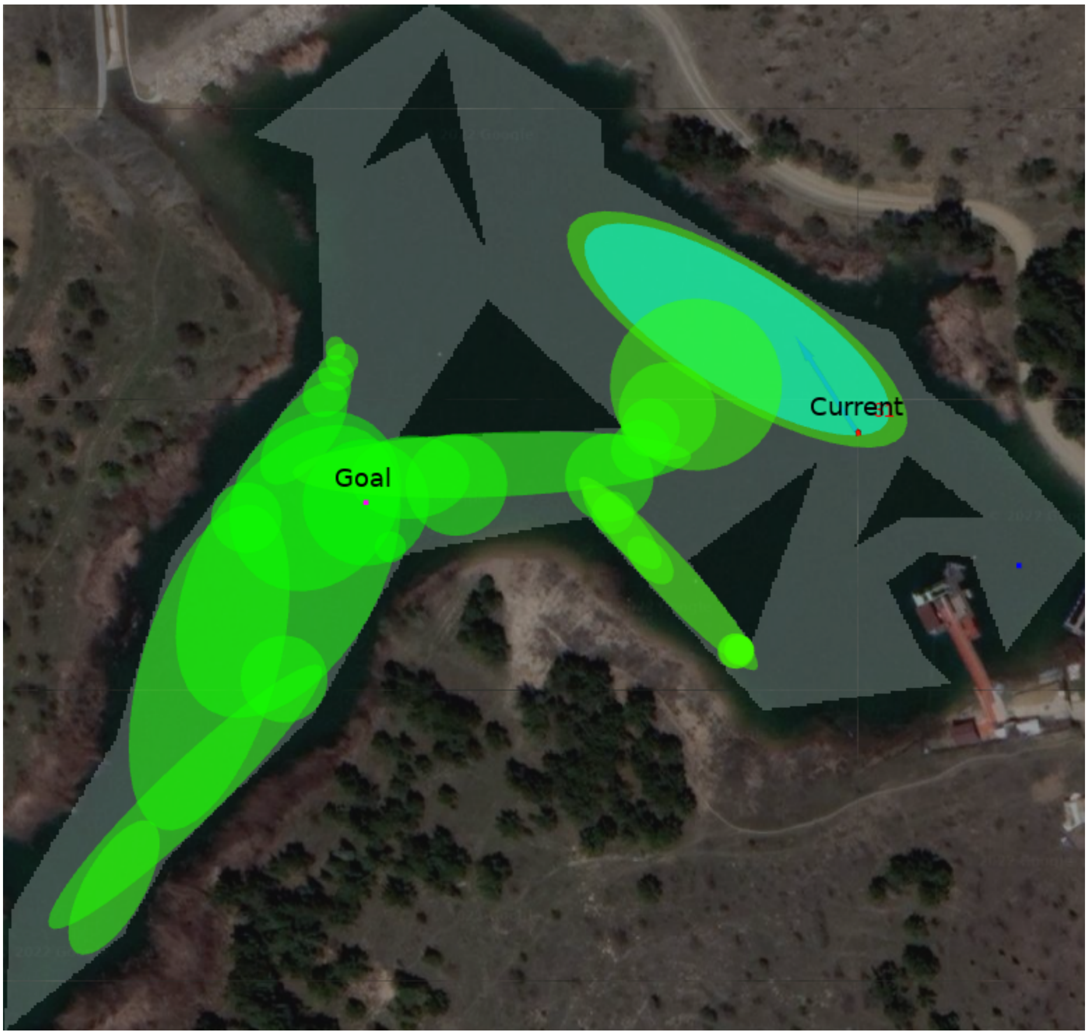


Figure 4.2: Screenshot of RViZ, showing the visualization elements published by the *sft_server*. In this screenshot, black polygons show the virtual obstacles and green markers show the funnels.

queries enable its use also for 3D map representations that can be utilized in the future.

4.2.2 mavros

MAVLink is a lightweight protocol widely used for communication with autonomous systems, including the experimental platform we worked on. More details about MAVLink are in Section 5.2. Mavros is an open-source package with a node that links ROS messages to MAVLink messages and vice versa. The mavros node is highly customizable with plugins for specific use cases and MAVLink messages. The plugins we directly use with the mavros node can be seen in Table 4.2. Using mavros, we were able to use generic ROS messages for interacting with the robot without directly sending/receiving MAVLink messages.

It should be noted that while ROS nodes and packages adhere to the ENU (East, North, Up) convention, the state estimates reported from the FCU follow the NED (North, East, Down) convention. For proper operation, conversions between NED and ENU conventions need to be made for the states reported in NED and commands sent in ENU. The mavros node simplifies the usage by handling convention-related conversions while converting MAVLink messages to ROS messages and vice versa.

Table 4.2: Used mavros plugins and their function.

Mavros Plugin	**Direction**	**Description**
global_position	MAVLink ->ROS	Reports fused position in geographic coordinates
local_position	MAVLink ->ROS	Reports velocity estimates of the USV in its body frame
imu_pub	MAVLink ->ROS	Reports fused orientation and raw IMU measurements
setpoint_velocity	ROS ->MAVLink	Forwards commanded velocity to the USV
sys_status	ROS <->MAVLink	Reports robot's debug info and forwards commands such as change mode

CHAPTER 5

EXPERIMENTAL SETUP

5.1 Experimental platform

The experimental platform we tested our algorithm on can be seen in Figure 5.1. The platform is based on the BluROV2, a tethered underwater remotely operated vehicle (ROV) supplied by BlueRobotics. BlueROV2 is a commercially available open-source ROV that employs ArduSub software and PixHawk autopilot to provide autonomous capabilities. Thanks to its open-source software and hardware, BlueROV2 is easily hackable and expandable. BlueROV2's expandability and low cost make it a good candidate as a research platform; thus, it is widely used in the academy [32–35].

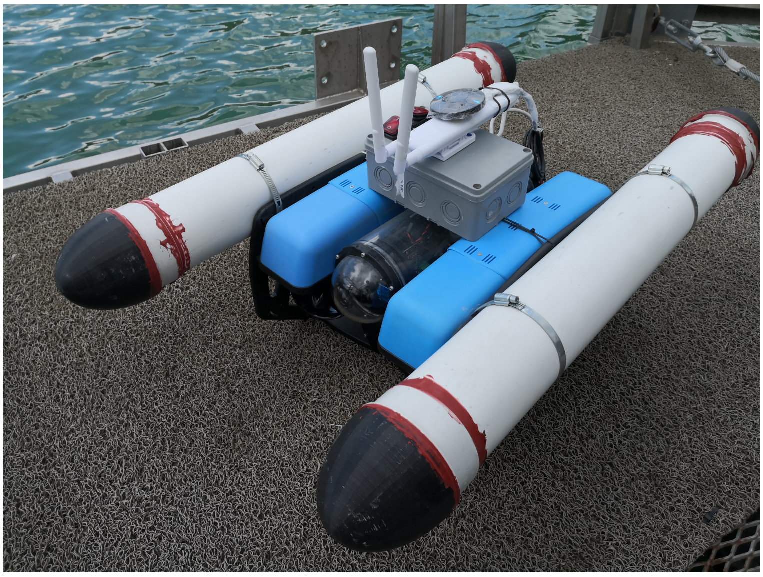


Figure 5.1: The platform used in the experiments. It consists of the main body, catamaran floaters on both sides and the communication assembly at the top.

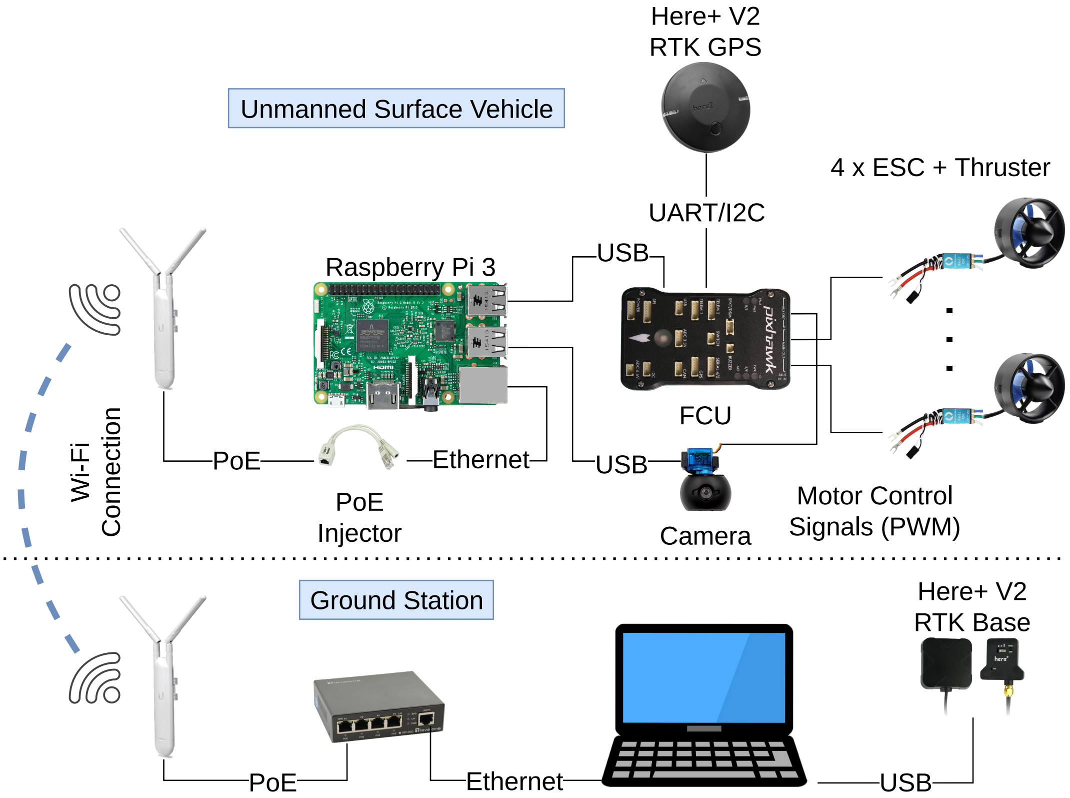

Unmanned Surface Vehicle
Here+ V2
RTK GPS
UART/I2C
4 x ESC + Thruster
Raspberry Pi 3
USB
FCU
Wi-Fi
Connection
PoE
Ethernet
USB
PoE
Injector
Camera
Motor Control
Signals (PWM)
Ground Station
Here+ V2
RTK Base
PoE
Ethernet
USB

Figure 5.2: Connection diagram of the experimental platform's and ground station's main components.

The experimental platform's components with our modifications can be seen in Figure 5.2, whereas the BlueROV2's components are as in Figure 5.3. The experimental platform's differences from the original ROV can be categorized into two for making it an unmanned surface vehicle. The first category of differences is buoyancy increasing differences, and the other category of differences is autonomous operation differences. Buoyancy-increasing differences can be summarized as follows. The vertical thrusters were removed from the platform because they were no longer needed for surface operation, and buoyancy foams were installed in their place (inside the blue volumes to the sides). With the removal of the vertical thrusters, the motor configuration of the robot became as given in Figure 5.4. Additionally, catamaran-type floaters were attached to the sides for safe surface operation. The physical properties of the robot changed as a result of these modifications and became as shown in Table 5.1.

The data link between the ground station and ROV in the original platform is realized using the MAVLink protocol via the wired Ethernet connection. Since the Fast

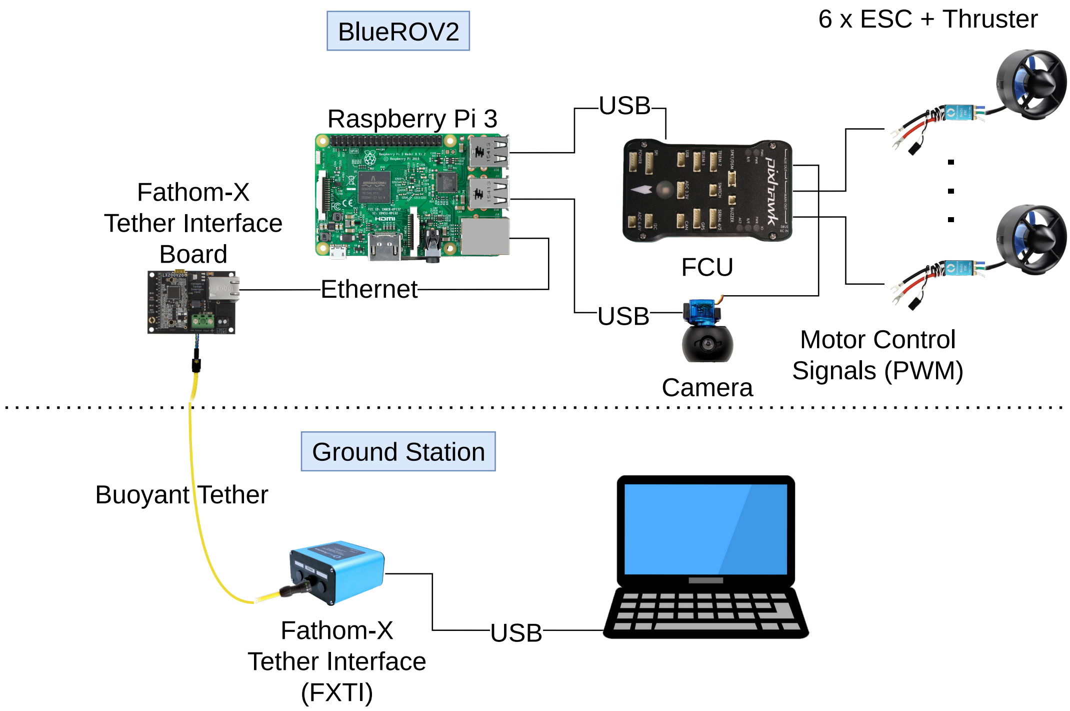

BlueROV2
6 x ESC + Thruster
Raspberry Pi 3
USB
Fathom-X
Tether Interface
Board
FCU
Ethernet
USB
Motor Control
Signals (PWM)
Camera
Ground Station
Buoyant Tether
Fathom-X
Tether Interface
(FXTI)
USB

Figure 5.3: Connection diagram of the BlueROV2 and ground station's main components. The $FXTI$ box connected to the ground station contains the interface board and an Ethernet to USB converter, allowing Ethernet communication with the ROV from the USB port. On the FXTI's product page, the manufacturer mentions that the choice of USB is to enable both powering and communication from a single cable.

Ethernet (100BASE-TX) can only support cable lengths up to 100 meters [36], which might not be enough for some underwater operations, the original platform uses interface boards to convert the physical layer. Used interface boards convert Ethernet (10/100BASE-TX) physical layer to the BPL (Broadband over Power Lines, IEEE 1901) to support cable lengths up to 2000 meters [37]. In the experimental platform, this physical link is replaced with mesh-capable wireless access points to eliminate possible tether entanglements. Used Ubiquiti UAP-AC-M Wireless modems enable network connection between the USV and GCS using WiFi (IEEE 802.11ac) and allow untethered operation up to 183 meters [38]. In our tests, we observed that operating the USV from approximately 150 meters line of sight is possible. There also exists radio telemetry solutions such as RFD900X [39] that support up to 40 km line of sight communication, which we are not using currently. However, it can be considered when farther range of operation is desired. The last difference related to autonomous operation is the addition of the RTK-capable GPS to the platform and the addition of the RTK base station to the ground station to be used in position estimation filters running on the flight control unit (FCU).

Table 5.1: Physical properties of the experimental platform.

Mass (kg)	**Width (cm)**	**Length (cm)**	**Height (cm)**
12.40	54.00	88.00	48.50

Real-Time Kinematic (RTK) is a relative positioning method that can achieve a cm-level positioning accuracy of a station by enabling the use of satellite corrections from other stations [40]. RTK methods require communication between stations to share satellite corrections. In traditional RTK, these corrections from the base stations are sent to the rover stations using radio modems [41]. The RTK-GPS used in this work consists of a base station installed to the GCS and a rover station that we connect to the experimental platform. The satellite corrections from the base station are sent to the FCU via MAVLink over the wireless link that we set.

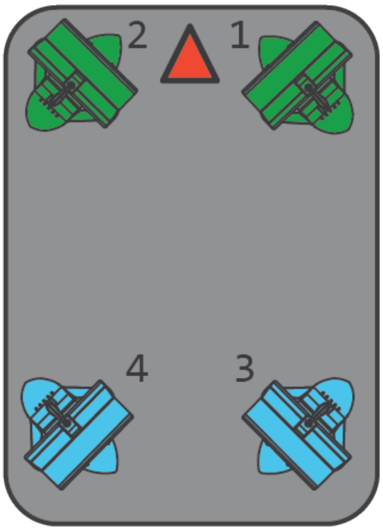


Figure 5.4: Motor configuration used in the experimental platform. In the illustration, the red triangle shows the forward direction, blue thrusters represent clockwise propellers and green thrusters represent counter-clockwise propellers.

5.2 Communication Channel

As previously mentioned in Section 5.1, the communication between the robot and the ground control station (GCS) uses MAVLink as the protocol. Known simply as MAVLink, the micro air vehicle link is a lightweight message serialization protocol for unmanned systems such as drones and ROVs [42]. The MAVLink protocol specifies the method for defining message structure and serialization at the application layer. These messages are then routed to the lower layers (transport and physical layers) for transmission to the network. There are two actively used MAVLink versions whose frame contents can be seen in Figures 5.5 and 5.6. MAVLink v1.0 was adopted around 2013, and MAVLink v2.0 was released in 2017 with some capability and security improvements over v1.0 [42]. These protocol versions can coexist in a network thanks to the different start-of-text (STX) markings used.

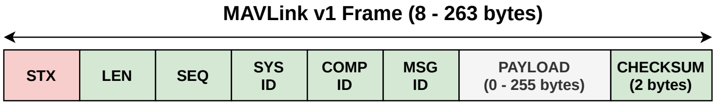

MAVLink v1 Frame (8 - 263 bytes)
STX
LEN
SEQ
SYS ID
COMP ID
MSG ID
PAYLOAD (0 - 255 bytes)
CHECKSUM (2 bytes)

Figure 5.5: Frame content of MAVLink v1.0 messages. *STX* byte of 0xFE marks the start of a v1.0 frame. *LEN* byte shows the length of the payload. *SEQ* is used for detecting packet losses by being incremented for each message sent. *SYSID* contains the identifier for the vehicle/system sending the message, allowing multiple vehicles/systems to coexist in the same network. *COMPID* encodes the type of the system sending the message. *MSGID* describes the message type; the payload should be decoded according to the message definition indicated by the *MSGID*.

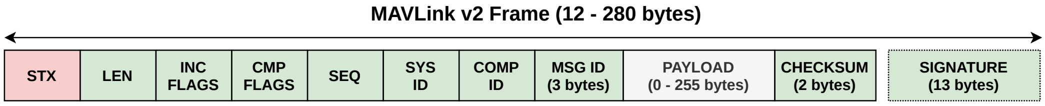

MAVLink v2 Frame (12 - 280 bytes)
STX
LEN
INC FLAGS
CMP FLAGS
SEQ
SYS ID
COMP ID
MSG ID (3 bytes)
PAYLOAD (0 - 255 bytes)
CHECKSUM (2 bytes)
SIGNATURE (13 bytes)

Figure 5.6: Frame content of MAVLink v2.0 messages. *STX* byte of 0xFD marks the start of a v2.0 frame. Unlike MAVLink v1.0, *MSGID* is represented by 3 bytes rather than a single byte enabling far more types of messages. The incompatibility/compatibility flags indicate whether some features need to be considered when unpacking the payload. The optional signature enables authentication and increases security.

Thanks to the wireless MAVLink connection between the robot and the GCS, it is possible to allocate responsibilities between them to their capabilities. For example, due to its access to the outer world, we gave the responsibility of mission control to the GCS and implemented the proposed algorithms on the GCS. It sends the calculated commands to the FCU via MAVLink, while FCU periodically publishes its state estimates and raw measurements to GCS using MAVLink.

5.3 State Estimation Infrastructure

As previously seen in Figure 5.2, the experimental platform has PixHawk as its FCU. PixHawk contains a fast M4 core with FPU as its main microprocessor, and an M0 failsafe co-processor, whose specifications are as given below [43]. The PixHawk in the experimental platform is flashed with ArduSub V4.1.0 autopilot stack [44] with only guided control changes mentioned in Section 5.4.

- 32-bit STM32F427 Cortex® M4 core with FPU:
  - 168 MHz Clock Frequency,
  - 256 KB RAM,
  - 2 MB Flash.
- 32 bit STM32F100 Cortex® M0 failsafe co-processor:
  - 24 MHz Clock Frequency,
  - 8 KB RAM,
  - 64 KB Flash.

ArduSub scheduler manages multiple processing loops with different frequencies. For example, it reads RC(Radio Control) commands in its 50 Hz loop, whereas it checks leakage in its 3 Hz loop. Thanks to its powerful main processor, ArduSub executes state estimation and control tasks in its main loop, which is executed at 400 Hz. For estimating vehicle position, velocity and angular orientation; ArduSub utilizes an Extended Kalman Filter that uses GPS, accelerometer, gyroscopes, magnetometer and barometric pressure measurements [45]. State estimation and control flow can also be seen in Figure 5.7. ArduSub's main thread regularly runs (400hz) and retrieves the most recent data accessible via methods in the driver's front end. For example, the AHRS/EKF would obtain the most recent accelerometer, gyro, and compass data from the sensor drivers' front ends to produce the most recent attitude estimate [46]. In our experimental platform, the following set of sensors exists.

- Here+ V2 RTK Rover
  - NEO-M8P GNSS module
  - Invensense® ICM-20948
- PixHawk onboard sensors [43]
  - Invensense® MPU 6000 3-axis accelerometer and gyroscope
  - ST L3GD20 3-axis 16-bit gyroscope
  - ST LSM303D 3-axis 14-bit accelerometer and magnetometer
  - MS5611 barometer

In this version (V 4.1.0) of ArduSub, the default state estimation implementation is called EKF3 [45, 47–49] and its state has the 24 elements given in Table 5.2.

Table 5.2: States of the EKF running on the ArduSub. Gyroscope bias represents the angular bias resulting from the integration of the gyroscope measurements, and the accelerometer bias represents the velocity bias likewise.

EKF States
Angular orientation (4 quaternion states)
Velocity (3 axes, m/s)
Position (3 axes, m)
Gyroscope bias (3 axes, rad)
Accelerometer bias (3 axes, m/s)
Earth magnetic field (3 axes, Gauss)
Body magnetic field (3 axes, Gauss)
Wind velocity (2 axes, m/s)

The EKF3 (and its prior implementations) uses parameters to enable or change the source of measurements for the state estimation since Ardupilot is a multi-platform library, and it is being used on air, ground, and water vehicles. For example, wind velocity states of the EKF are not active in our setup since our platform is not equipped with an airspeed sensor. We did not modify any EKF3-related parameters in our setup and used the default values for the ArduSub V4.1.0.

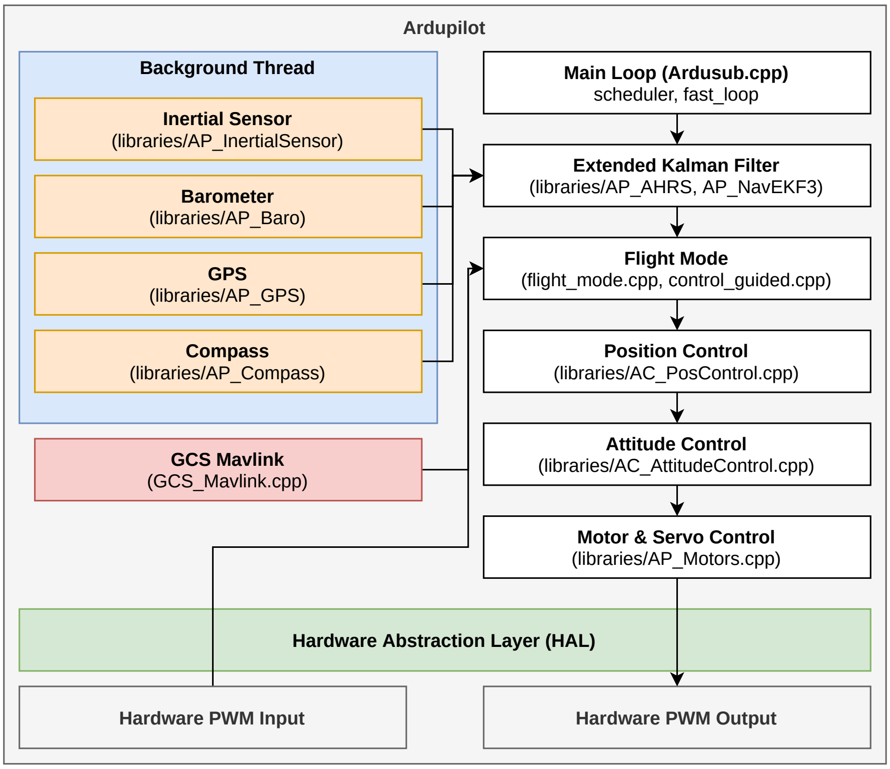

Ardupilot
Background Thread
Inertial Sensor (libraries/AP_InertialSensor)
Barometer (libraries/AP_Baro)
GPS (libraries/AP_GPS)
Compass (libraries/AP_Compass)
GCS Mavlink (GCS_Mavlink.cpp)
Main Loop (Ardusub.cpp) scheduler, fast_loop
Extended Kalman Filter (libraries/AP_AHRS, AP_NavEKF3)
Flight Mode (flight_mode.cpp, control_guided.cpp)
Position Control (libraries/AC_PosControl.cpp)
Attitude Control (libraries/AC_AttitudeControl.cpp)
Motor & Servo Control (libraries/AP_Motors.cpp)
Hardware Abstraction Layer (HAL)
Hardware PWM Input
Hardware PWM Output

Figure 5.7: State estimation and control architecture used in ArduSub stack when the robot is in guided mode. The high-level control commands v and ω that are sent via MAVLink are forwarded to the guided controller by GCS_Mavlink. Sensor drivers collect raw sensor data, convert it to standard units, and store it in buffers.

EKF3 architecture supports running multiple instances of EKF cores with different measurement sources [49], which is parametrically customizable and capable of changing its primary core depending on the error scores of the cores. Note that only the estimate of the primary core is reported by the AHRS (Attitude and Heading Reference System) navigation subsystem, and other cores are updated in the background, ready to switch anytime in case of degradation of the estimates from the primary core. In the default configuration, two EKF3 cores were running in parallel. In choosing the primary core, the error scores of the cores are calculated as the maximum of the GPS, altimeter, and magnetometer fusion scores for the sensors used in that EKF instance. The calculation of fusion scores is based on measurement innovations. In this context, lower the error score is better.

5.3.1 State Estimation Experiments

In order to observe the state estimation performance of the experimental platform, with and without RTK (Real-Time Kinematic) enabled, we executed a series of experiments at the parking lot near the A2 gate of METU. In these tests, we first marked the robot's location and heading and executed the following scenarios with RTK disabled. After each scenario, the robot is placed back in the marked position with the same heading. Experiment recordings were kept on between 7.5 - 11.0 minutes for the scenarios. Later, these scenarios were repeated after enabling RTK streaming with 180 seconds of surveying.

- Robot is let stationary.
- Robot is moved in a rectangular trajectory without much rotation.
- Robot is rotated multiple turns in both directions around the yaw axis.
- Robot is moved in a rectangular trajectory with rotations around the yaw axis.

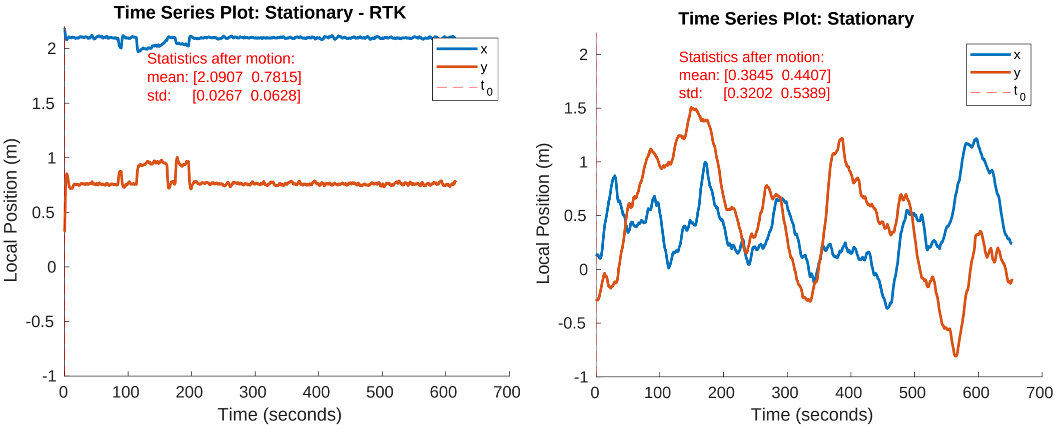


Figure 5.8: State estimation, local position results in the stationary scenarios.

Note that placing the robot in the same location with the same heading is made manually with our best effort, but our error margin is unknown However, the consistency of the mean positions in the experiments with the RTK can be seen as an indicator. In this section, a subset of the recorded results is given in Figures 5.8 to 5.13.

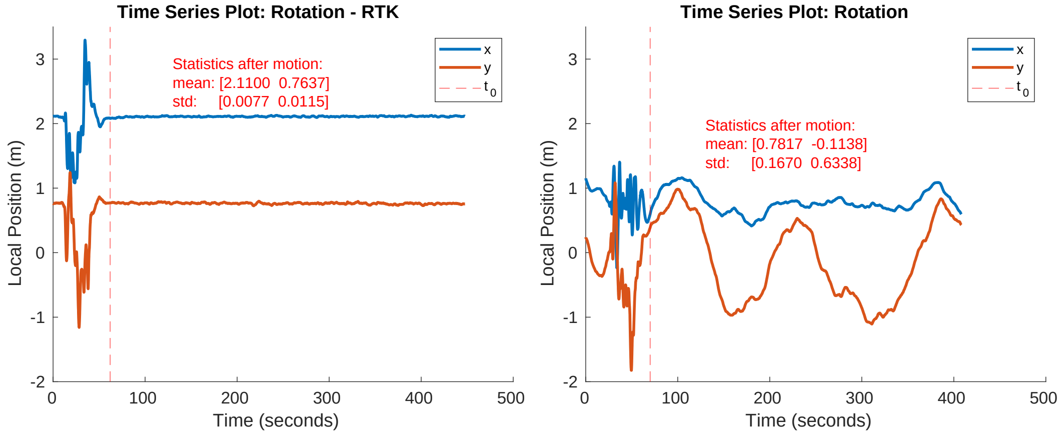


Figure 5.9: Local position results in the multi-turn rotation scenarios.

In Figures 5.8 to 5.10, local positions reported by the robot and the statistics of the x and y coordinate estimates are given. The reported local positions are relative to a frame complying with the ENU (East-North-Up) convention at the location where the robot is "armed". Note that the robot reports the local position in the NED frame, but it is converted to ENU by the mavros node mentioned in Section 4.2.2. Statistics given in the results are calculated using only the estimates after the provided t_0 instant, which marks the time point where we think the robot is let to rest.

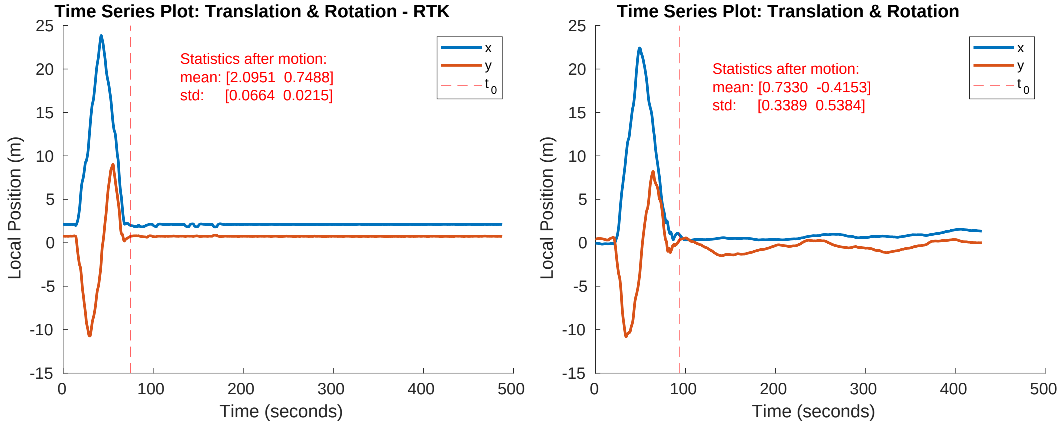
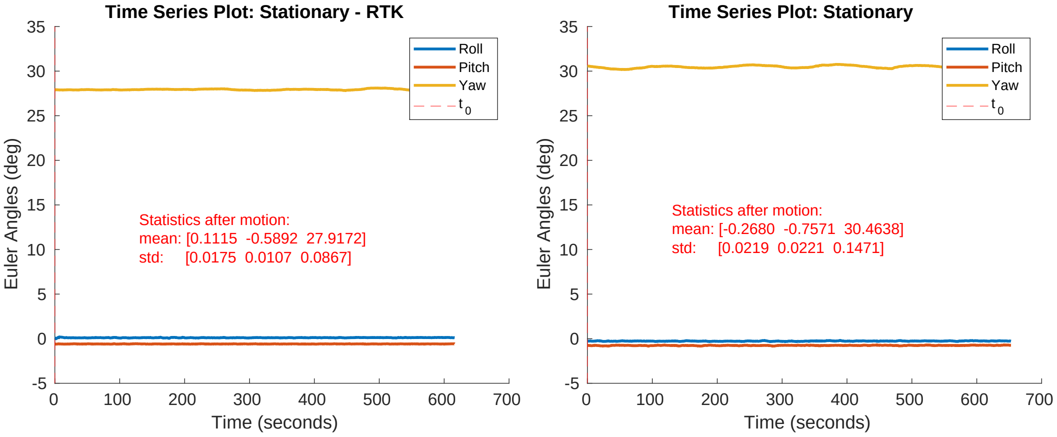


Figure 5.10: Local position results in the rectangular motion with rotation scenarios.

Inspecting the local position results, we observed that when the RTK is used, the mean of the position estimates converges to the same position (Minimum enclosing circle is with a radius of 1.65 cm) independent from the motion carried beforehand. However, when the RTK is disabled, the mean of the position varies in a circle with a radius of 46.21 cm. These results are also visible in the standard deviations given in the figures.


Time Series Plot: Stationary - RTK

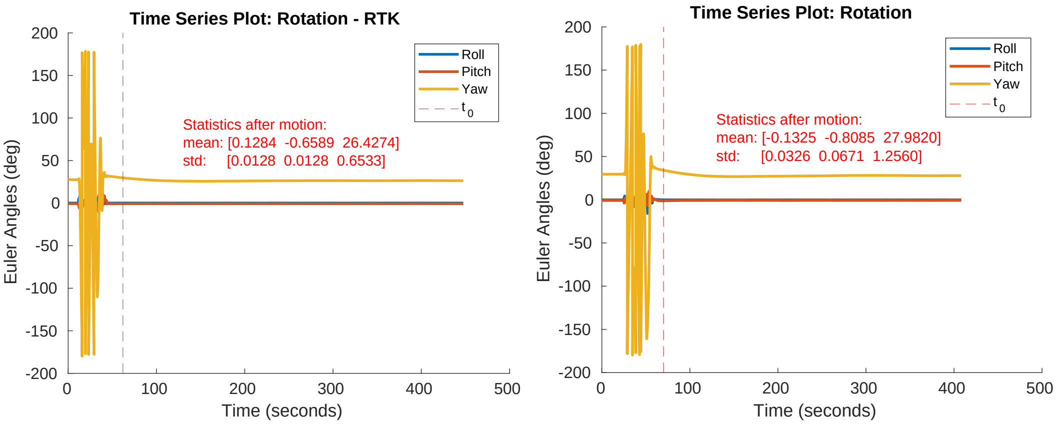
Statistics after motion:
mean: [0.1115 -0.5892 27.9172]
std: [0.0175 0.0107 0.0867]

Time Series Plot: Stationary

Statistics after motion:
mean: [-0.2680 -0.7571 30.4638]
std: [0.0219 0.0221 0.1471]

Figure 5.11: Euler angle results in the stationary scenarios.

In Figures 5.11 to 5.13, orientations reported by the robot and the statistics of them after the marked t_0 instants are given. Inspecting these results, we observed that the deviations on the yaw axis are larger than the other two axes. This is expected since the primary heading source of the robot is its magnetometers, whereas the primary sources for the other two axes are the inertial sensors.


Time Series Plot: Rotation - RTK

Statistics after motion:
mean: [0.1284 -0.6589 26.4274]
std: [0.0128 0.0128 0.6533]

Time Series Plot: Rotation

Statistics after motion:
mean: [-0.1325 -0.8085 27.9820]
std: [0.0326 0.0671 1.2560]

Figure 5.12: Euler angle results in the multi-turn rotation scenarios.

Another observation we made on the Euler angle results is that even if the RTK can only supply positional corrections, it still decreases the standard deviations on the Euler angle estimates of the EKF state estimator.

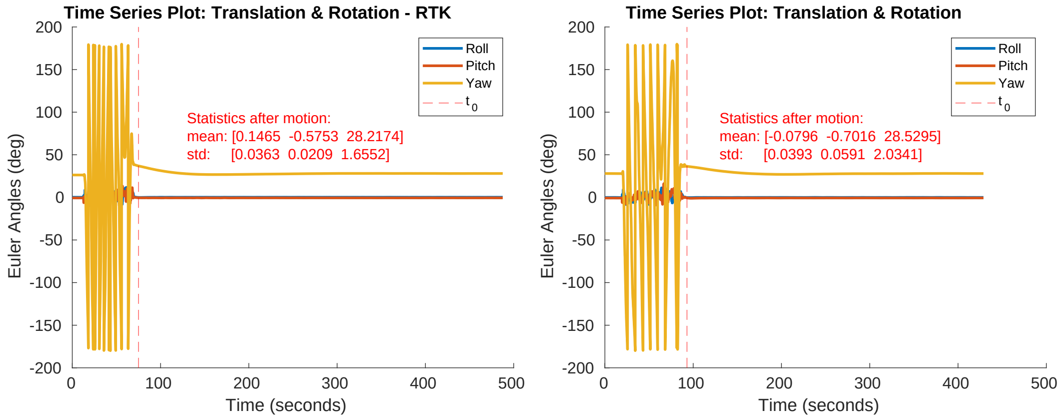


Figure 5.13: Euler angle results in the rectangular motion with rotation scenarios.

5.4 Operation Modes

The Ardusub stack has numerous operation modes, including manual, guided, stabilize, depth hold, etc. However, not all of them are usable for surface operation. The most straightforward and fully stable mode of operation is manual control, where the vehicle allocates motor control signals based on pilot input according to the motor configuration. In manual mode, it is possible to control the vehicle's thrust and torque direction with some degree of magnitude. Still, it is not possible to make it able to control the velocities, which our algorithm requires.

When judged by its interface, another possible operation mode is the guided mode. Since the MAVLink message that the guided mode is commanded (message with id 84) lets the GCS send position, velocity, acceleration, and yaw rate commands. But in practice, this interface is not fully realized (guided mode is in beta), and it was impossible to command the vehicle with raw velocity commands using the ArduSub V4.1.0 (As of September 2022, the latest stable version). In the progress, we worked on the realization of the low-level controller using both modes and chose to proceed with a patch that would enable the guided mode's velocity control.

5.4.1 Guided Control

In guided mode, different from manual mode, command channels, which are forward, lateral, throttle, roll, pitch and yaw channels, are filled by the position and orientation controllers shown in Figures 5.14 and 5.15a. Later on, filled channels are used for allocating individual motor efforts just like in the manual control mode using the motor configuration in the function summarized in Figure 5.15b.

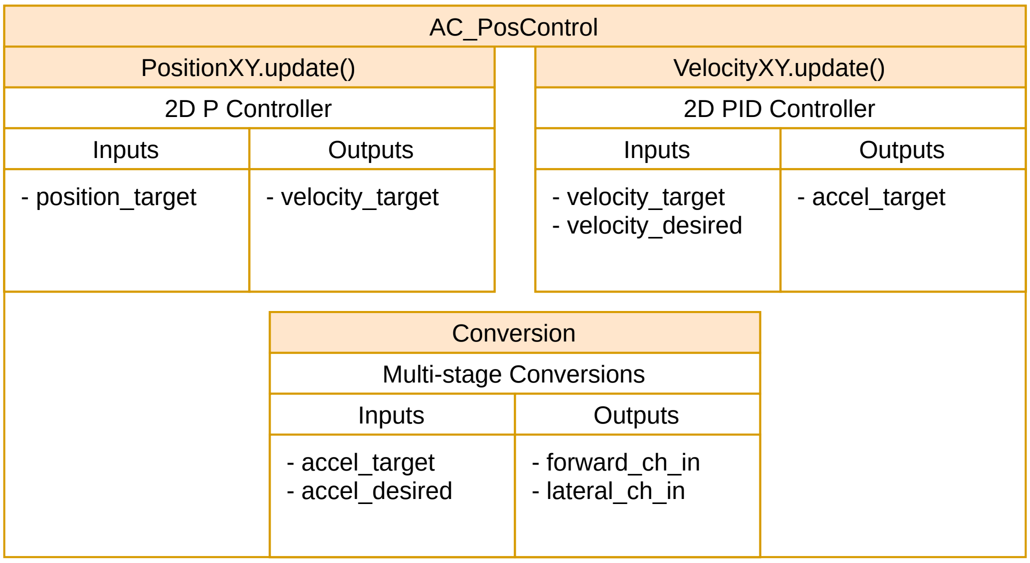
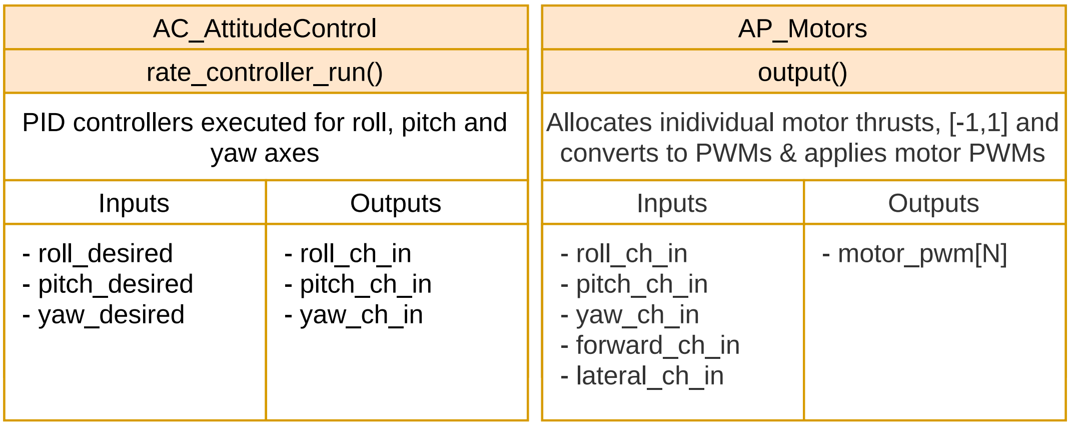

AC_PosControl
PositionXY.update()
2D P Controller
Inputs
Outputs
- position_target
- velocity_target
VelocityXY.update()
2D PID Controller
Inputs
Outputs
- velocity_target
- velocity_desired
- accel_target
Conversion
Multi-stage Conversions
Inputs
Outputs
- accel_target
- accel_desired
- forward_ch_in
- lateral_ch_in

Figure 5.14: Position control architecture in guided mode.


AC_AttitudeControl
rate_controller_run()
PID controllers executed for roll, pitch and yaw axes
Inputs
Outputs
- roll_desired
- pitch_desired
- yaw_desired
- roll_ch_in
- pitch_ch_in
- yaw_ch_in
AP_Motors
output()
Allocates inidividual motor thrusts, [-1,1] and converts to PWMs & applies motor PWMs
Inputs
Outputs
- roll_ch_in
- pitch_ch_in
- yaw_ch_in
- forward_ch_in
- lateral_ch_in
- motor_pwm[N]

(a) Attitude control architecture.

(b) Control allocation.

Figure 5.15: Attitude (orientation) control architecture and control allocation stage in guided mode.

In the control schema shown in Figure 5.14, *position_target*, *velocity_desired*, and *accel_desired* are filled with the commands from MAVLink message *#84*, whereas *velocity_target*, *accel_target* respective controller's output. In this schema, commanded velocity and accelerations are used as feed-forward terms to the following controllers, enabling intervention to the desired control layer via the previously mentioned MAVLink message. To realize the required velocity control, we only supplied the controller with *velocity_desired* and disabled position stabilization by sending the current position as the *position_target*.

In the attitude controller shown in Figure 5.15a, the commanded orientation is used for calculating required rotational velocities. They are used as references to the respective PID controllers that use gyroscope measurements as feedback signals. The yaw rate command from MAVLink message *#84* is converted to the yaw angle target using discrete integration to the current yaw angle before giving target orientation to the attitude controller. With the disabled vertical thrusters, zero desired rates on roll and pitch axes are not realized by the controller; but they are primarily stabilized passively by the buoyancy.

As previously mentioned, the guided control mode of Ardusub did not fully realize its interface, and it was impossible to command USV using velocity commands. To resolve the issue, we wrote a patch to the Ardusub stack. The written patch featured two changes, the first of which enabled forwarding yaw rate commands from the command message to the low-level controllers illustrated in Figures 5.14 and 5.15a. The other change implemented is the disablement of position stabilization when the position ignore flag was given in the command message. With these changes, it was possible to command the USV and its SITL simulation with yaw rate and forward (and backward) velocity commands.

After the patch, USV's response to the yaw rate commands was satisfactory, but its response to the forward velocity command showed some unexpected and hard-to-model patterns both in SITL and hardware experiments. A light realization of these patterns can be seen in Figure 5.16, which is the velocity response of the USV in the SITL simulation that replicates a real experiment. Some inspection of the control parameters of the FCU showed that the irregularities were derived from high gains

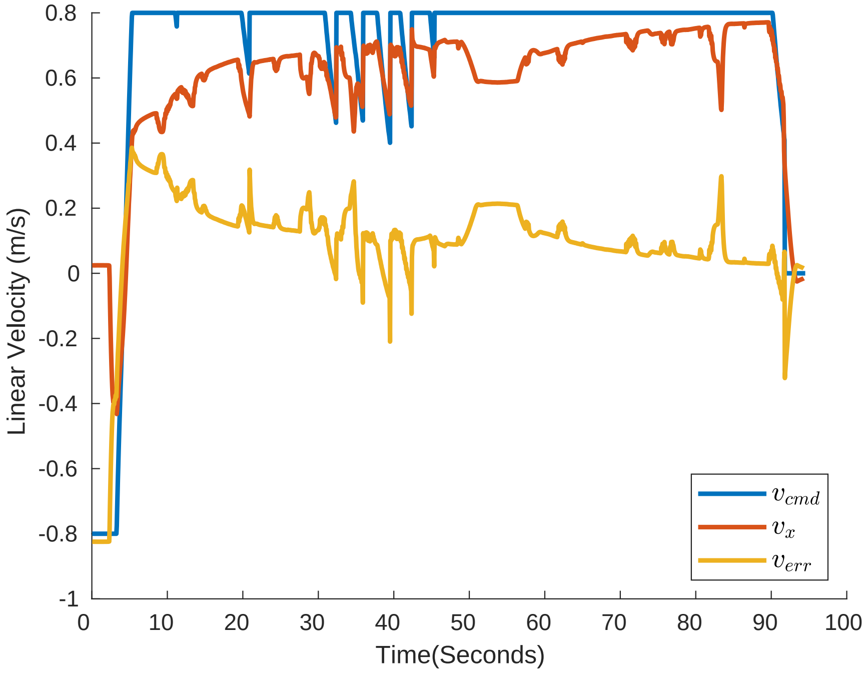


Figure 5.16: Forward velocity response of the USV on the SITL simulation that replicates a real multi-funnel experiment with default FCU parameters.

of the z-axis controller, which had no utility for a surface vehicle. After making the adjustments given in Table 5.3 to the FCU parameters, better velocity responses were gotten both in SITL and the physical platform. These results can be seen in Figure 5.17 and Figure 5.18 respectively.

Table 5.3: FCU parameter updates regarding guided control.

Parameter	**Default**	**Changed to**
PSC_ACCZ_FF	0.75	0.00
PSC_ACCZ_I	4.00	0.10
PSC_ACCZ_P	2.00	0.50
PSC_POSXY_P	2.50	2.00
PSC_VELXY_D	0.80	0.00
PSC_VELXY_FF	0.00	4.00
PSC_VELXY_I	0.50	0.80
PSC_VELXY_P	5.00	6.00
PSC_VELZ_P	8.0	0.20

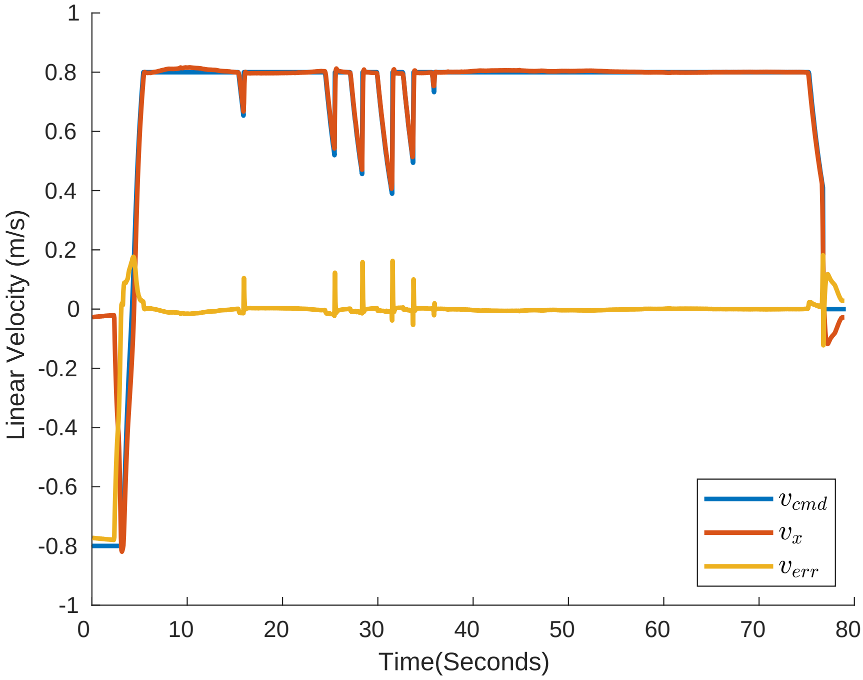


Figure 5.17: Forward velocity response of the USV on the SITL simulation with updated FCU parameters.

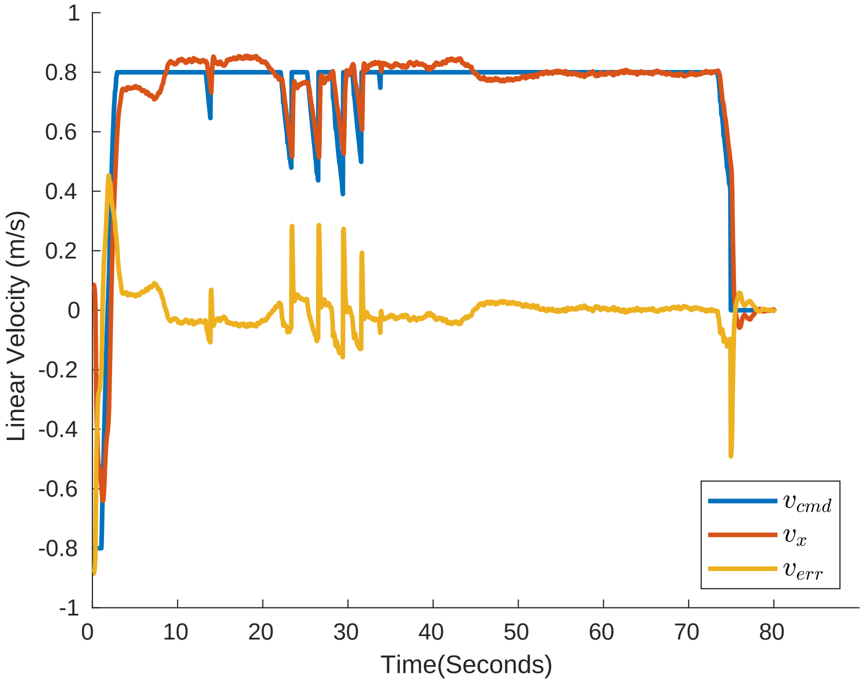


Figure 5.18: Forward velocity response of the real platform with updated FCU parameters.

5.5 Simulation Setups

For the algorithm's development and testing, two separate simulation setups were used, each of which is utilized for a different layer of the implementation. The first simulation configuration is an ideal kinematic unicycle model realization. The other simulation environment is the Software in the Loop (SITL) simulation of the Ardusub.

5.5.1 Ideal Kinematic Unicycle Simulation

Ideal kinematic simulation implemented as a software plugin that could directly use the implemented partial feedback linearized nonlinear controller transparently. Implemented model uses motion equations given in (3.2) to update its state. We utilized this simulation to evaluate the controller's response when used with different controller gains in funnels with varying specifications. The true merit of this setup is its execution can be made faster than real-time (around 1300x, depending on the computer's processing and disk-write performance), enabling automated Monte Carlo simulations of the proposed control schema. This simulated setup is used extensively for the results presented in Section 6.2.2.

5.5.2 SITL Simulation

In the software-in-the-loop approach, the system is simulated with its interfaces, and the interfacing software is run without knowing the system is being simulated. The SITL approach is frequently used because it enables a quicker, safer, and more affordable development process [50]. In our case, physical experiments required coordination with other parties using the Yalincak lake and METU Internal Services to access the lake. To increase the efficiency of our experiments, we utilized software in the loop simulation of Ardusub, the framework our experimental setup is based on, primarily for testing and verifying the interface in between. This way, we could focus on algorithmic results in the physical experiments. Additionally, since the model used in the SITL setup was a dynamic model that showed similar responses to our experimental platform, we also utilized it to evaluate the effect of the control parameters.

5.5.2.1 Model Used in the SITL Simulations

As previously stated, SITL simulations are primarily employed for the verification of the MAVLink interface between our modules and the USV. We resolved interface-related issues before the physical experiments thanks to this prior verification. Furthermore, we have yet to attempt to modify or improve the dynamical model introduced in this section. The information presented here is provided by inspecting the C++ implementation of the SITL model [44].

Upon inspecting the implementation, the state is updated periodically using PWMs calculated by the control allocation stage described in Section 5.4.1. State update is executed in two primary steps. The first step is responsible for calculating the resultant rotational and linear accelerations when the PWMs are applied to the thrusters. In the second step, a time update (position and orientation) is made using the constant acceleration motion model. In the first step, the following tasks are carried out for each thruster:

- Use commanded PWM to calculate thrust to be generated using the model shown in Figure 5.19. This thrust curve is a piecewise quadratic one with a dead band.
- Calculate the resultant linear accelerations on the forward and lateral axes when the calculated thrust is generated using Newton's second law of motion and thruster's pose.
- Calculate the resultant angular accelerations that will be generated when the calculated thrust is generated using the thruster's mounting pose and the frame's modeled moment of inertia. In the SITL setup, the frame's inertia is modeled as equivalent to a sphere with a radius of 0.2 m.
- This process is repeated for all the thrusters and resultant accelerations on the body frame are added.

The linear and angular accelerations due to the thrusters are modeled at this point in the process. However, other sources must be considered as well, such as drag forces, as they have a large impact on the motion of the USV. Furthermore, drag torques,

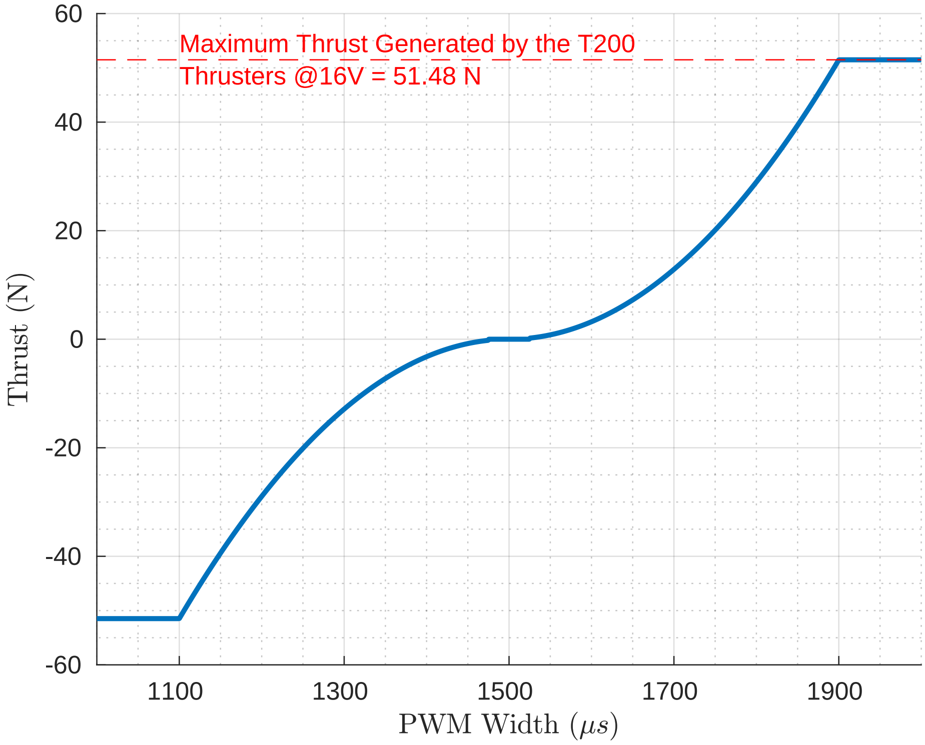


Figure 5.19: The thrust curve used in the SITL setup. There exists a dead band with a width of 50 μs at 1500 μs and ESC (Electronic Speed Controller, thruster's driver) command saturates above 1900 μs. In this model, magnitude of the thrust increases quadratically to the maximum. The thrust curve is symmetric around 1500 μs.

which are rotational counterparts of drag forces, are also an important factor. That's why drag forces and torques are also modeled in the SITL setup. Later in this step, accelerations due to these sources are calculated using Newton's second law and the vector forms of the following equations.

$$F_D = \frac{1}{2} C_l d A v^2 \,, \tag{5.1}$$

$$\tau_D = \frac{1}{2} C_r d A \omega^2 \,. \tag{5.2}$$

Later, the accelerations derived from these are superposed with the ones from the thrusters. The physical quantities and the parameters used for the calculation of the accelerations can be seen in Table 5.4.

Table 5.4: In the SITL setup, the following parameters are used for modeling the dynamics of the robot.

Name	**Symbol**		**Value**	**Unit**
Drag force	F_D		-	N
Drag torque	τ_D		-	Nm
Water Density	d		1023.6	kg/m^3
Moment of inertia	I		0.268	kgm^2
Mass	m		16.75	kg
Equivalent sphere radius	R		0.2	m
Maximum thrust of T200 @16V	T_{max}		51.48	N
Linear drag coefficient	C_l	C_{lx}	1.4	-
		C_{ly}	1.8	-
		C_{lz}	2.0	-
Angular drag coefficient	C_r	C_{rx}	1.05	-
		C_{ry}	1.05	-
		C_{rz}	1.05	-

Since the BluROV2 is originally an underwater platform, the model used in the SITL setup does not perfectly match our experimental platform. Additional lead sinkers are the most apparent difference between the SITL model and the experimental platform. Since the experimental platform does not have these lead sinkers, its mass is around 12.4 kg. Another difference is that the inertia of the model in the SITL setup is approximated as a sphere, whereas the experimental platform is not. Due to this difference, this approximation might not accurately reflect the experimental platform's inertia.

CHAPTER 6

RESULTS

The planning arena shown in Figure 6.1 was built over the lake close to the Yalincak province of METU, where we conducted our experiments with the physical setup to enable direct comparison of simulation results with real-world studies.

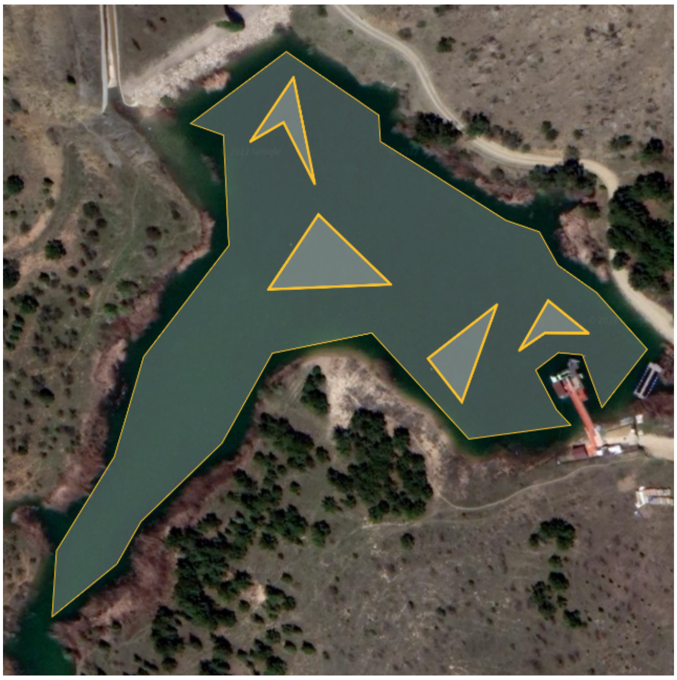


Figure 6.1: Planning arena defined over the dam lake in METU. The orange-edged polygons represent the virtual obstacles that are defined on the map to make the scenarios more challenging.

The arena is defined using geographic coordinates for consistency over the experiments and projected to a local cartesian coordinate frame using GeographicLib [31] before using it in the tree generation stage. Virtual polygonal obstacles (both convex and concave in shape) exist in the defined arena to make the arena more challenging.

6.1 Motion Control Results

In this section, motion control results are presented and the results are divided into two parts. In 6.1.1, the agent's response in single funnels with varying gains and initial conditions are presented, whereas in 6.1.2, the USV's response in complete funnel trees are shown.

6.1.1 Single-Funnel Results

Single-funnel experiments were carried out to observe changes in the trajectories followed by the agent with varying k_α ($k_v = 0.2$ is fixed since it was noted to be a viable gain). Various initial conditions from different funnel sections were employed in these tests to demonstrate the control policy's response better. In the kinematic simulations illustrated in Figure 6.2, we observed that increasing k_α results in more direct trajectories.

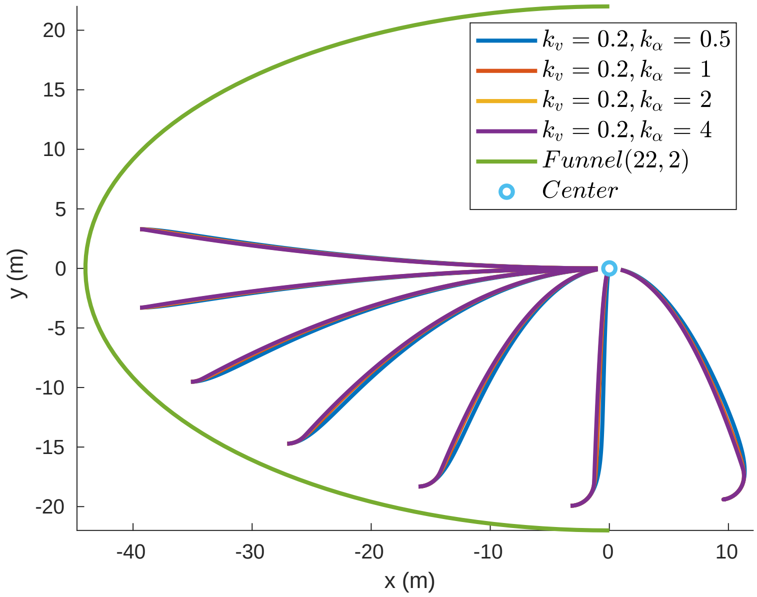


Figure 6.2: Controller's responses with different controller parameters on a funnel with $a = 2$ on kinematic simulation. All trajectories begin near the boundary and travel to the funnel's center. The difference between the four controllers is their k_α, while they share the same k_v. The first argument of Funnel in the legend shows the r of the ellipse, whereas the second shows the a of it.

From the simulation results, we were prone to use higher k_α in the physical experiments; but when used, we observed oscillations on the yaw axis of the robot. These oscillatory responses can be seen in Figure 6.3. It should be noted that the executed paths are still quite consistent with the kinematic simulation results. More single funnel results can be seen in Figure 6.6 that demonstrates the paths that the USV executed from various initial conditions also containing outward headings.

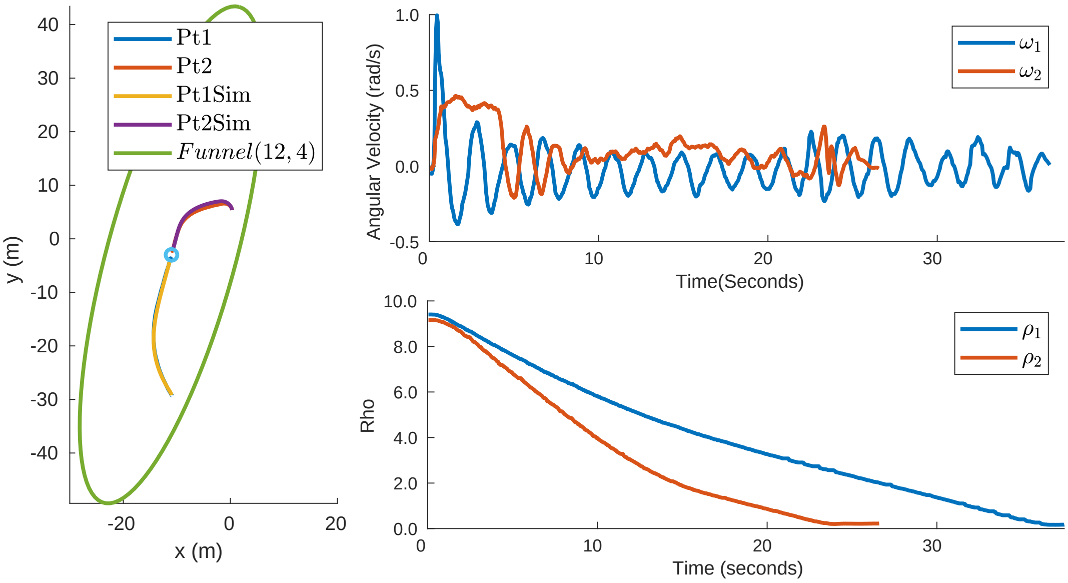


Figure 6.3: Angular velocity ω and local state ρ on real experiments when $k_\alpha = 4.0$ is used starting from two distinct points. Trajectories starting from these points are also given on the left part of the figure. Trajectories labeled with 'Sim' postfix are results from the repeated executions with the ideal kinematic unicycle simulation.

Change in ρ, which is one of the controlled variables, while being controlled by the designed partial feedback linearized controller illustrated in Figure 6.4. The tailored controller creates inputs that are supposed to result in non-increasing ρ, but it is still possible for ρ to have slight increments due to external forces, initial velocity and the inertia of the USV.

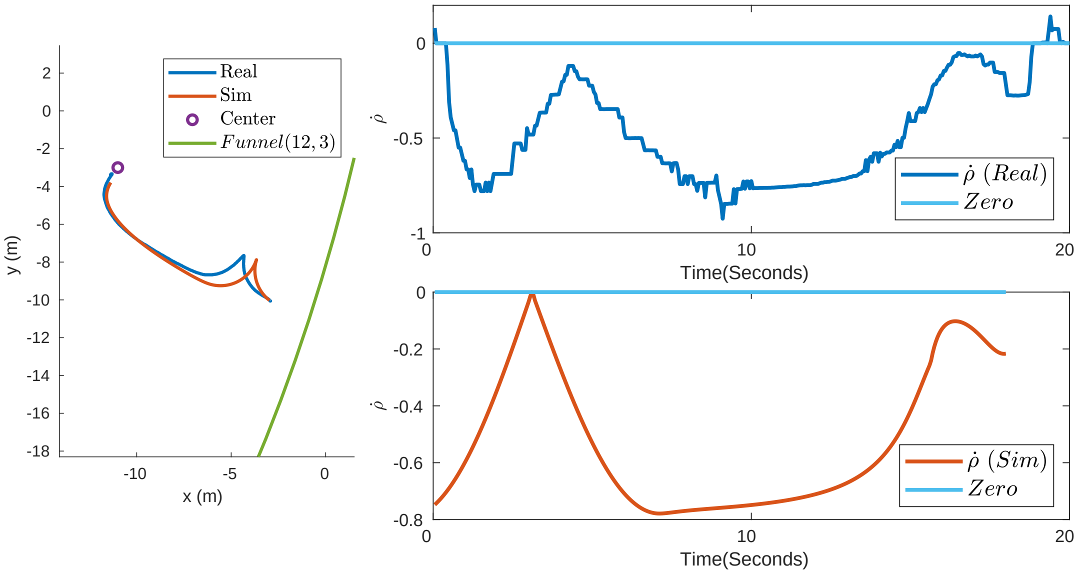


Figure 6.4: Change in ρ when $k_\alpha = 2.0$ is used on multiple setups. Note that there existed discrete jumps on the $\dot{\rho}$ of the real results due to the use of finite difference with the ZOH'ed ρ measurements, that's why we present the moving median filtered (with a window of 0.6 seconds) version of it.

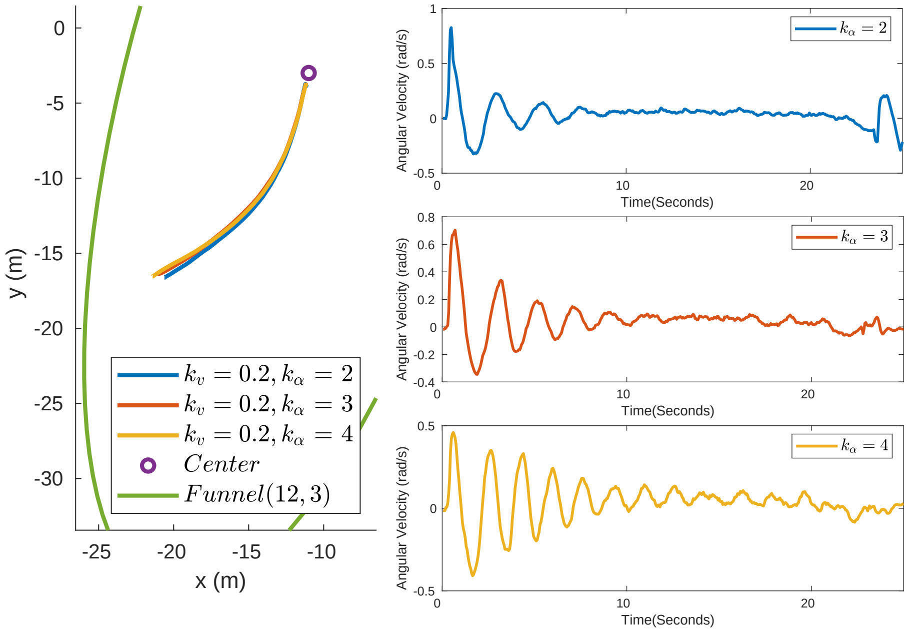
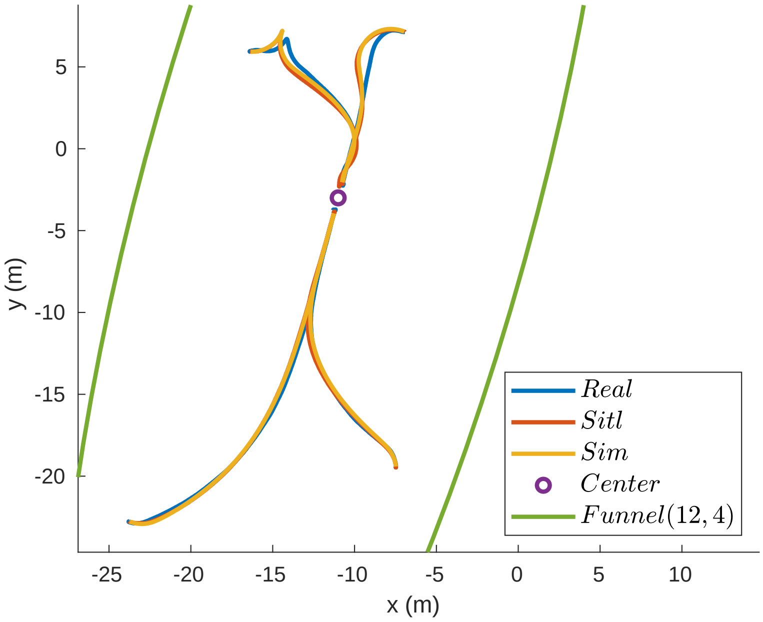
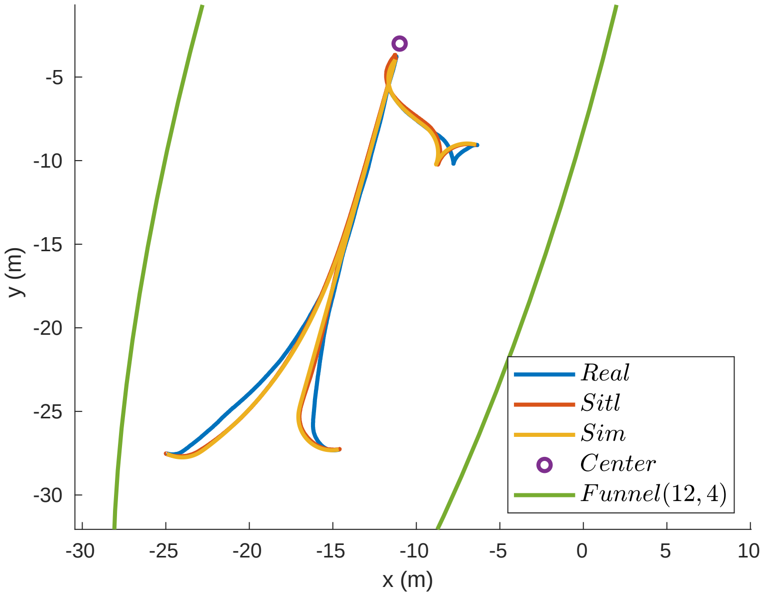


Figure 6.5: Angular velocity response of the USV on the lake with controller gains $k_v = 0.2$ and varying k_α with controlled initial conditions. The radius of the minimum enclosing circle for the initial positions of these recordings is less than 0.4 meters, and initial headings are within a window of $11.4°$.



(a) $k_v = 0.2$, $k_\alpha = 2.0$



(b) $k_v = 0.2$, $k_\alpha = 3.0$

Figure 6.6: Controllers' responses on a single funnel with different initial conditions and k_α. Experiments carried out at the lake and their results are reproduced both in SITL and kinematic simulations using the same initial conditions. Initial twists observed in some of the trajectories occur when USV's heading is outward. It rotates the robot's heading inward without increasing ρ.

6.1.2 Multi-Funnel Results

To observe the effects of the gain changes, we run the experiments recorded on multiple funnel cases on two realizations of the tree generation stage labeled MF1 and MF2. The funnel tree realization MF2 is illustrated in Figure 6.8 as an example. In these trees, two starting points were used for the observability of the changes made. These starting points can be seen in Figure 6.7.

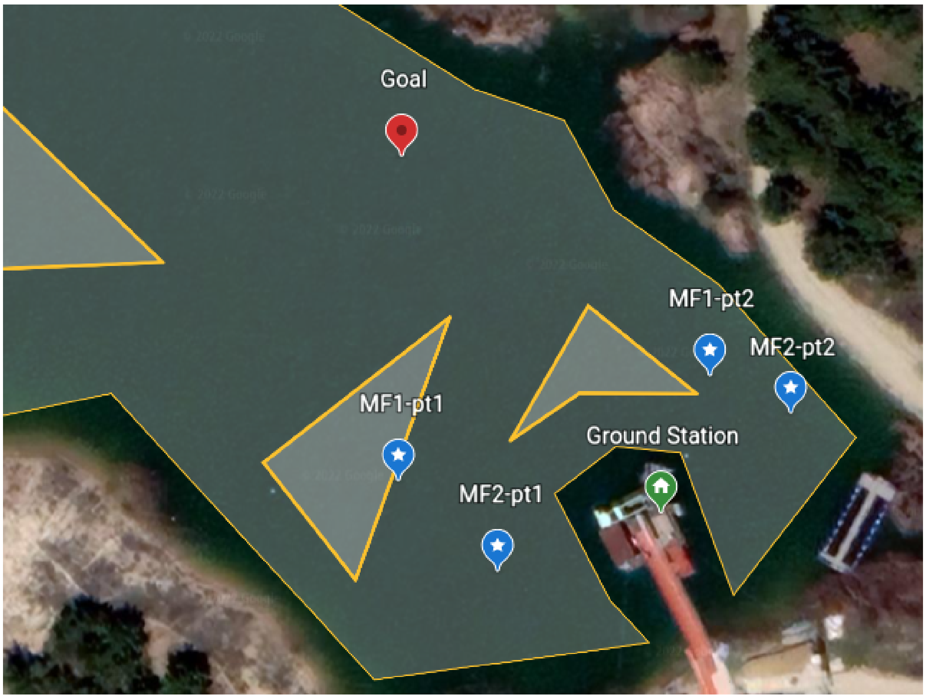


Figure 6.7: Ground station, starting points, and goal point used in multi-funnel experiments.

Results from one of the multi-funnel experiments are given in Figure 6.9, which was carried out on the funnel tree labeled MF2. This experiment was carried out on a day with heavy rain, wind, and possibly strong surface currents. Due to these conditions, the oscillations on the yaw axis resulting from the k_α being 4 are more noticeable. For a more clear comparison of gains, the angular velocity responses of three consecutive executions are given in Figure 6.11. The improvement in the yaw axis response compared to the $k_\alpha = 4$ case can be seen clearly in Figure 6.11.

In Figure 6.10, results from another multi-funnel experiment with $k_\alpha = 2$, which was executed on the MF1 tree, are illustrated. The resultant paths in this execution

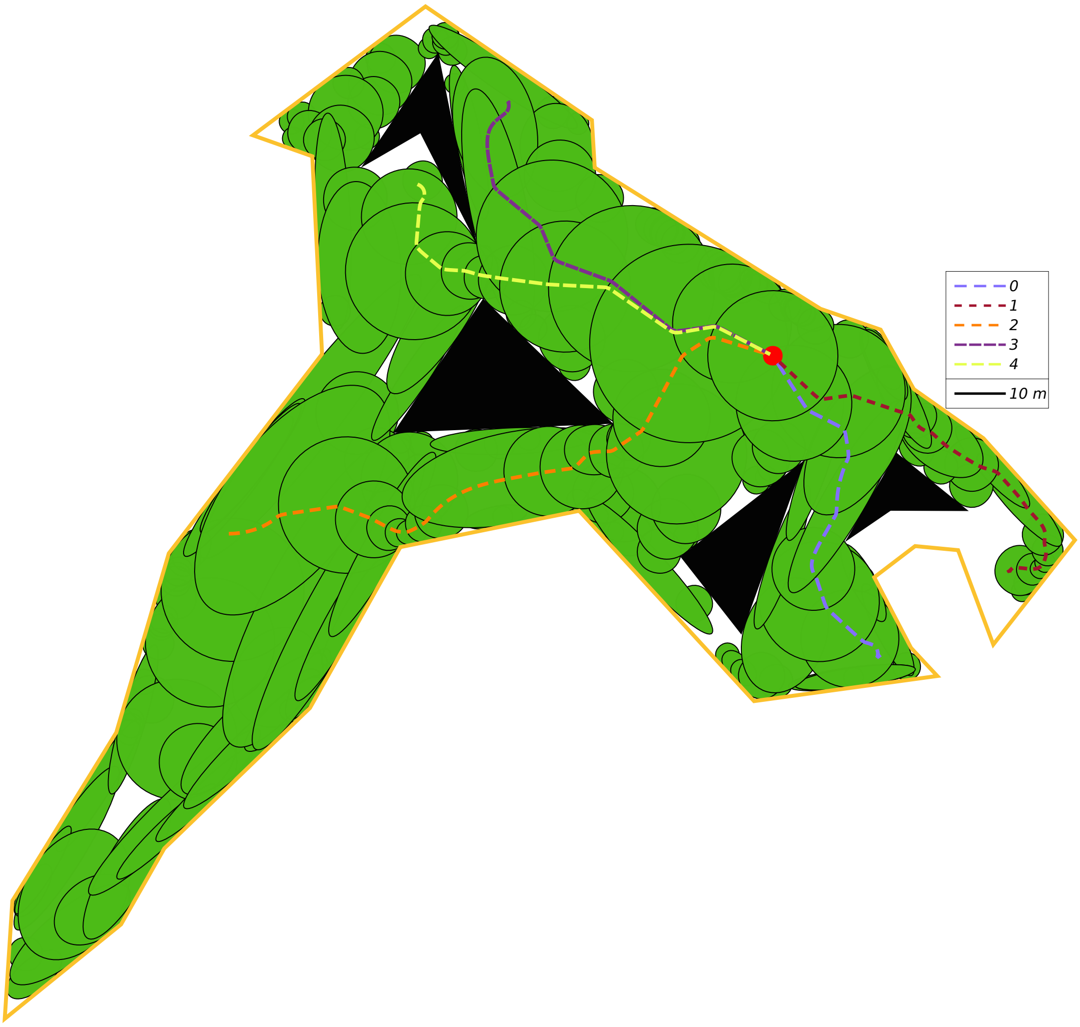
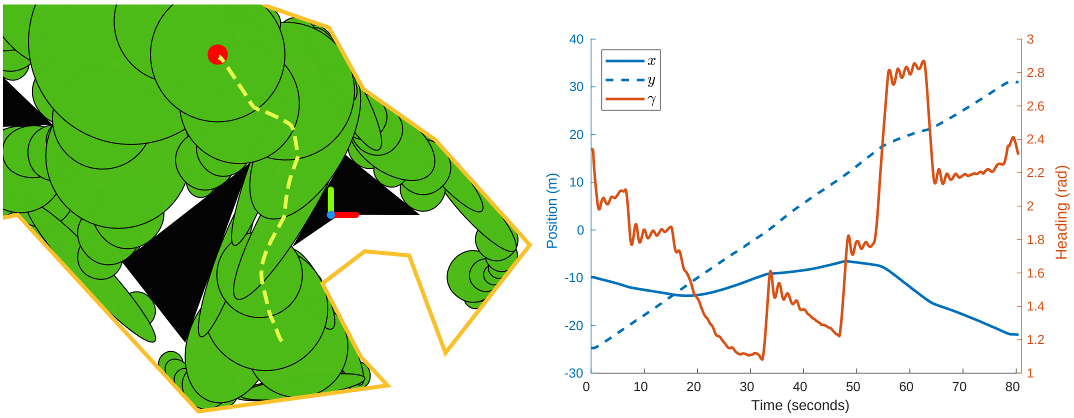


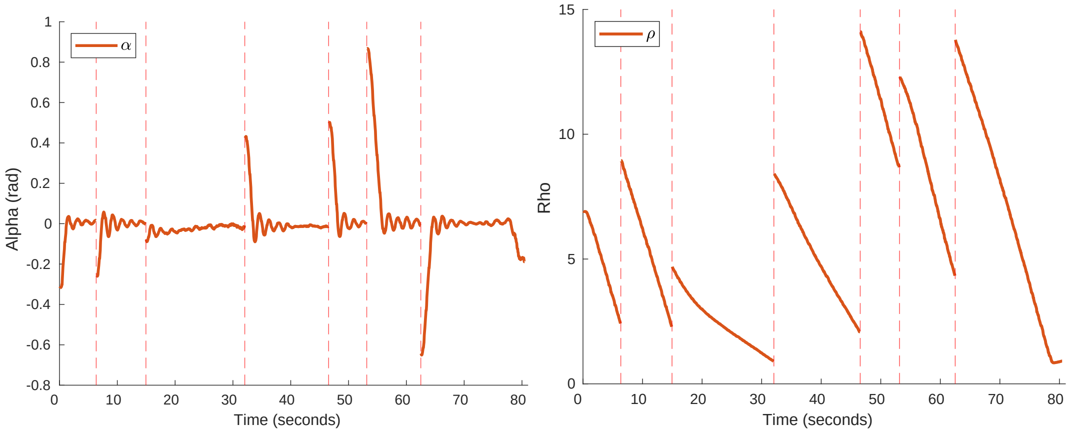
Figure 6.8: Used funnel tree realization MF2 with simulated trajectories from arbitrary start locations. The red circle shows the goal position where the USV would reach from any other funnel.

with also SITL and SIM realizations on top of the used funnel tree can be seen in Figure 6.12. In Figure 6.13, a multi-funnel execution on the MF2 tree starting from MF2-pt1 is shown. The closeness of the trajectories in real, SITL and SIM realizations in these multi-funnel experiments enables the use of SITL and SIM setups for further comparisons with the previous method using Monte Carlo experiments.



(a) Path followed by the USV in this experiment.

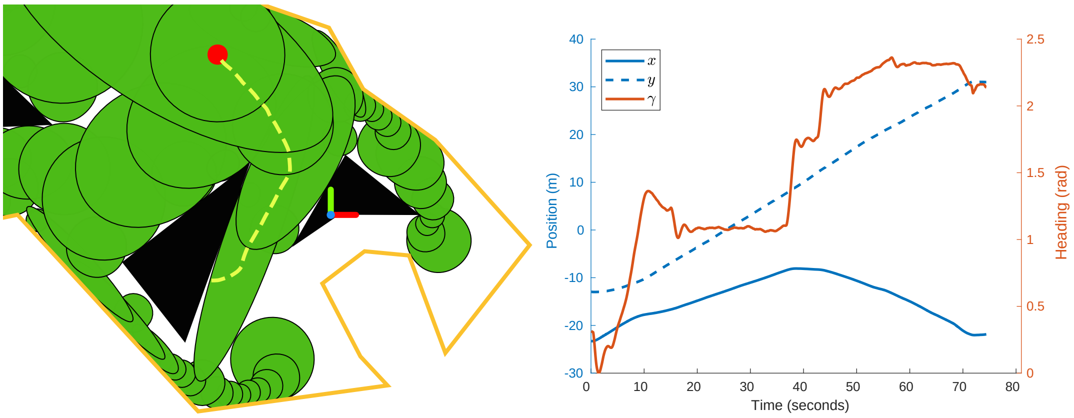
(b) USV's global states with respect to time.



(c) USV's local state α with respect to time.

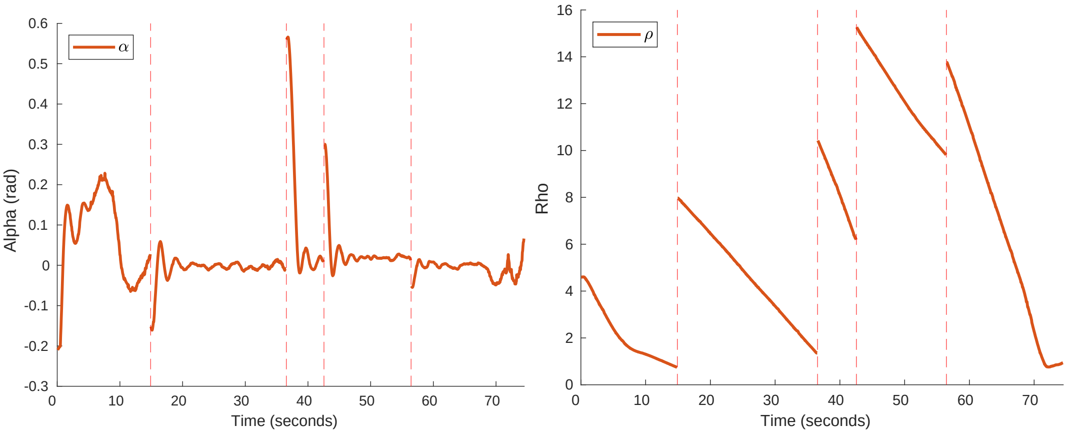
(d) USV's local state ρ with respect time.

Figure 6.9: Multi-funnel experiment results of the USV with $k_v = 0.2$ and $k_\alpha = 4$. The recording starts from the MF2-pt1 given in Figure 6.7. Since the local states α and ρ are defined with respect to the active funnel's center, discrete jumps exist between funnels. These discrete jumps on the local states are marked with red dashed vertical lines. Global states shown in (b) are defined relative to the right-handed frame (ENU) shown in the experiment illustration, where the x, y, and z axes are represented with red, green, and blue, respectively.



(a) Path followed by the USV in this experiment.

(b) USV's global states with respect to time.



(c) USV's local state α with respect to time.

(d) USV's local state ρ with respect time.

Figure 6.10: Multi-funnel experiment results of the USV with $k_v = 0.2$ and $k_\alpha = 2$. The recording starts from the MF1-pt1 given in Figure 6.7. Since the local states α and ρ are defined with respect to the active funnel's center, discrete jumps exist between funnels. These discrete jumps on the local states are marked with red dashed vertical lines. Global states shown in (b) are defined relative to the right-handed frame (ENU) shown in the experiment illustration, where the x, y, and z axes are represented with red, green, and blue, respectively.

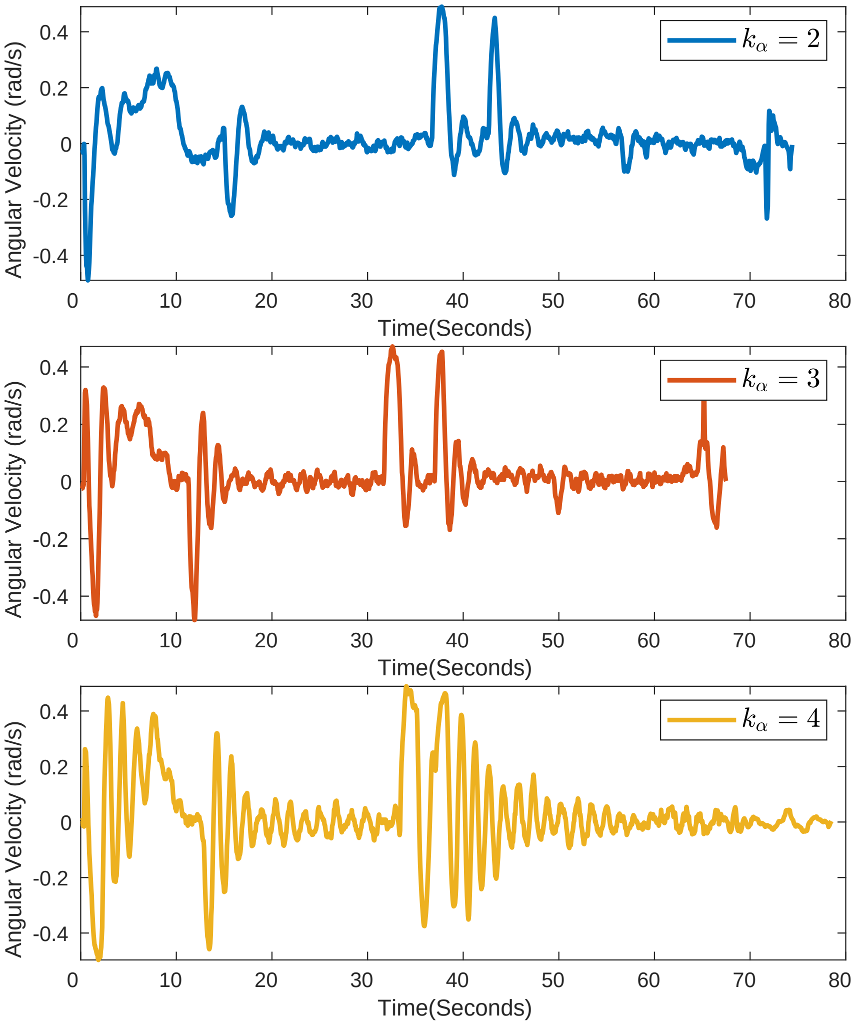


Figure 6.11: Angular velocity responses of the experimental platform on the MF1 execution given in Figure 6.10 with varying k_α. The radius of the minimum enclosing circle for the initial positions of these recordings is less than 0.69 meters, and initial headings are within a window of $8.1°$.

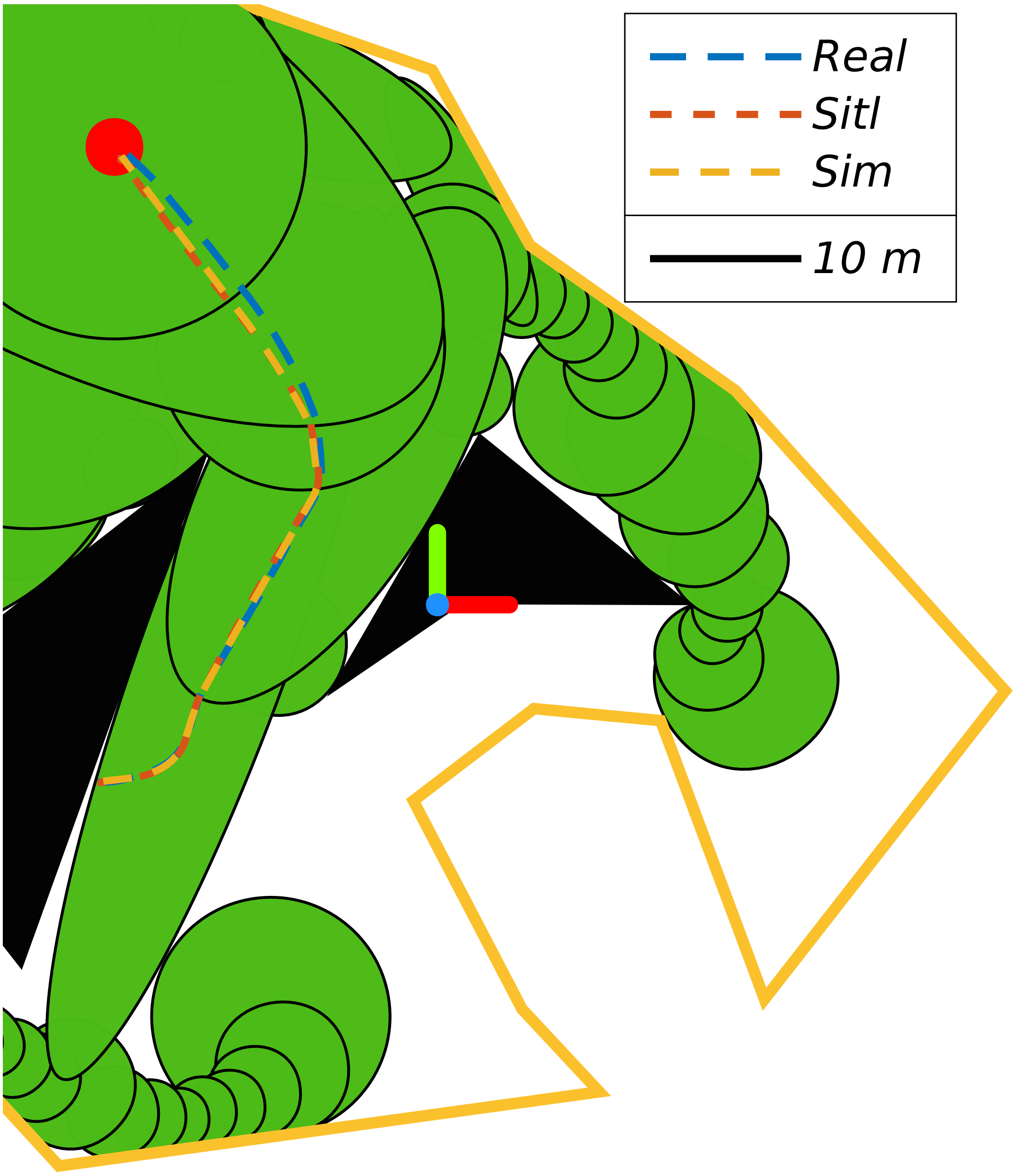


Figure 6.12: Path resulted from the multi-funnel experiment made with the USV on top of the used funnel tree (MF1). The scenario is repeated in SITL and SIM setups using the same initial conditions in the real experiment. The test starts from the MF1-pt1 given in Figure 6.7 and gains $k_v = 0.2$ and $k_\alpha = 2$ are used in all three realizations. A video replay of this physical experiment can be accessed at `https://youtu.be/xzBtKxv8xjM`.

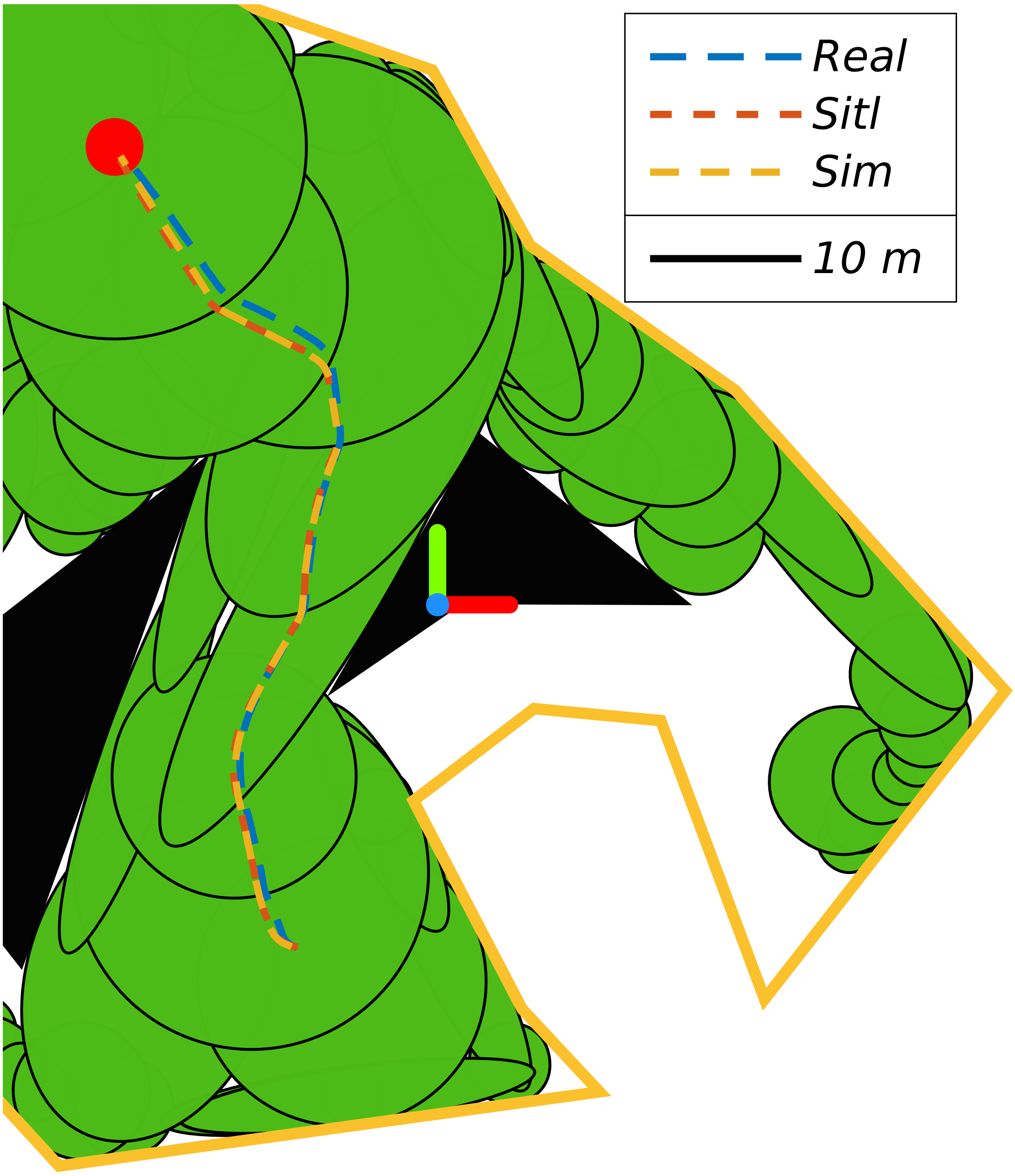
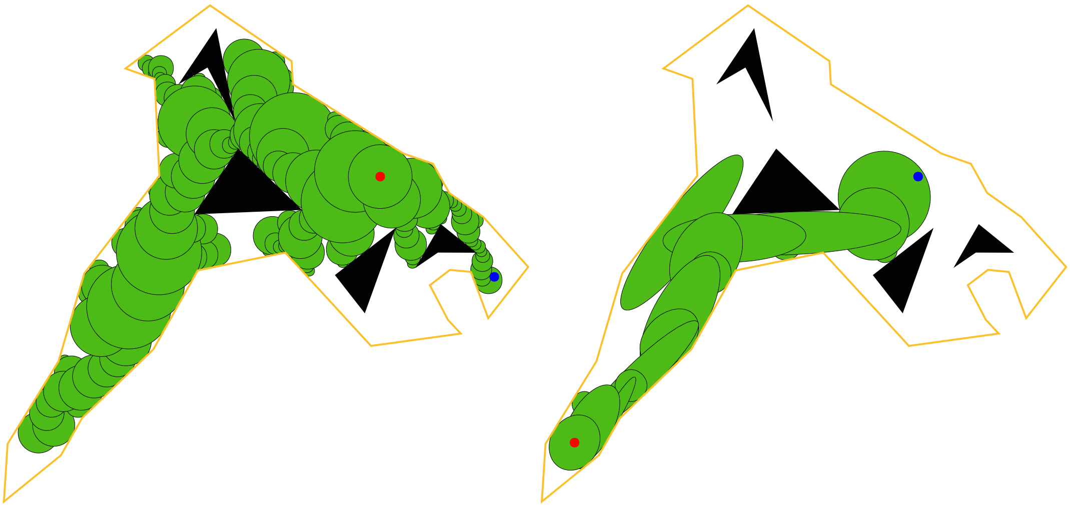


Figure 6.13: Path resulted from the multi-funnel experiment made with the USV on top of the used funnel tree (MF2). The scenario is repeated in SITL and SIM setups using the same initial conditions in the real experiment. The test starts from the MF2-pt1 given in Figure 6.7 and gains $k_v = 0.2$ and $k_\alpha = 2$ are used in all three realizations. A video replay of this physical experiment can be accessed at `https://youtu.be/R1eT_o0o5LE`.

6.2 Performance Comparison With the Circular Funnel Approach

In this section, tree generation and motion control performances of elliptical and circular funnel approaches will be compared. In order to compare both approaches, the two scenarios given in Figure 6.14 are used which use the same arena with different start and goal coordinates. Due to the difference in their start and goal positions, these two scenario results in two distinct tree expansion characteristics. For example, scenario 1 in Figure 6.14a results in trees with lower depth for the start node due to the lower physical distance between start and goal positions. Another difference between the scenarios is that the start position in scenario 2 is in the natural expansion direction of the tree with the root at the goal position, that's why scenario 2 results in fewer node counts when compared with scenario 1.



(a) Scenario 1 with a circular funnel tree on top. (b) Scenario 2 with an elliptical funnel tree on top.

Figure 6.14: Planning scenarios defined over the lake in METU, with example circular and elliptical tree realizations on top. In the scenarios, red points show the goal positions whereas the blue points show the starting positions.

6.2.1 Tree Generation Results

To illustrate the tree generation improvements of elliptical funnels over the circular funnels, executed tree generation stages with elliptical funnels and circular funnels

50 000 times in scenarios given in Figures 6.14a and 6.14b. In these total of 200 000 recorded executions, the user-defined parameters given in Table 6.1 are used.

Table 6.1: Parameters used in tree generation tests, which are described in Section 3.1.

η	**Coverage Confidence (P_c)**	**Coverage Fraction (β)**	**Minimum Radius**
0.8	0.99	0.5	2.0

Results derived from the executions on scenario 1 are given in Table 6.2 and Figure 6.15 whereas results from scenario 2 are in Figure 6.16 and Table 6.3. In both scenarios, elliptic funnels result in significantly fewer funnels while still requiring time within reasonable limits. Failure rates given in Tables 6.2 and 6.3 shows how much, in a thousand trials, the tree generation stage terminates without reaching the start position due to the probabilistic termination condition in (3.1) with the P_c and β given in Table 6.1. This small ratio for early terminations certainly does not overlay the purpose of using them, which is used as a way of terminating in cases where no possible solutions exist.

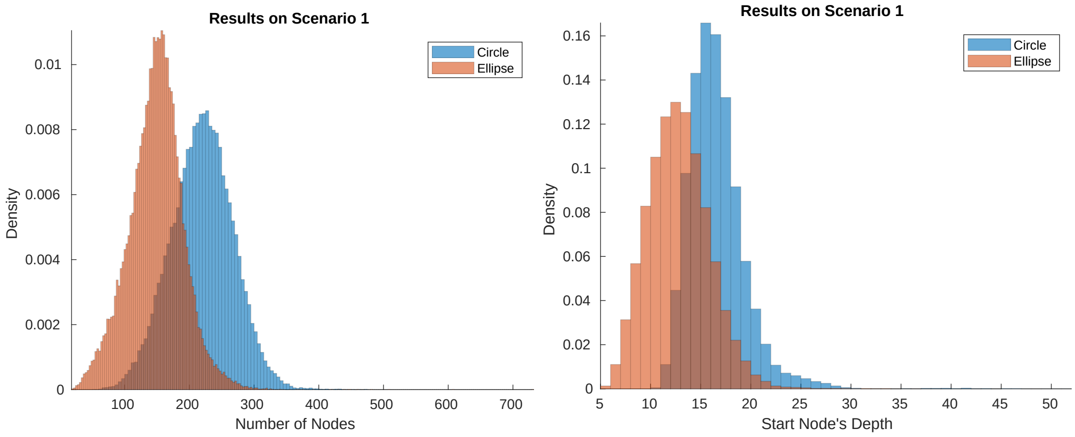
Table 6.2: Statistics from the tree generation executions run on scenario 1.

Funnel Type	**Failure Rate**	**Time Elapsed (Seconds)**		**Number of Nodes**		**Start Node's Depth**	
-	‰	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**
Elliptic	2.06	0.0984	0.1421	148.1801	42.0072	12.3174	3.0716
Circular	0.22	0.0083	0.0053	219.1985	47.7933	16.0907	2.8884

Table 6.3: Statistics from the tree generation executions run on scenario 2.

Funnel Type	**Failure Rate**	**Time Elapsed (Seconds)**		**Number of Nodes**		**Start Node's Depth**	
-	‰	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**
Elliptic	0.10	0.0068	0.0276	45.8832	23.8350	18.5060	4.0199
Circular	0.00	0.0011	0.0013	77.7369	27.1693	35.1913	5.9719

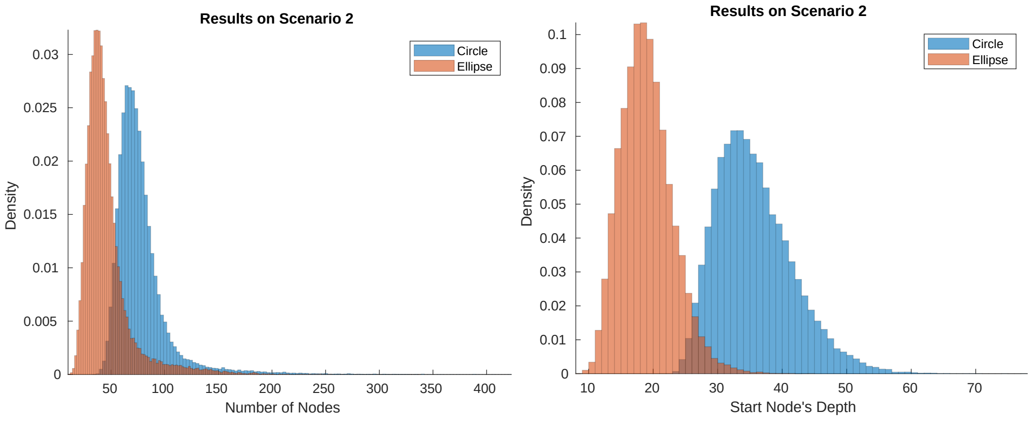
Another statistic given in Tables 6.2 and 6.3 is the start node's depth. The depth of a node shows its direct connectivity order with the goal node. For example, the depth of the goal node is 0, whereas its immediate neighbor's depths are 1. In our case, the start node's depth shows how many active controller changes will occur before reaching the goal node. Since controller changes might result in discrete jumps on the commands sent to the actuators, the lower the start node's depth, the better.



(a) Number of nodes generated on scenario 1.

(b) Start node's depth on scenario 1.

Figure 6.15: Histogram plots of the 50 000 trees generated each for elliptic and circular funnels on scenario 1. For ellipse and circle strategies, mean node counts are 148.20 and 219.20; and the mean start node's depths are 12.32 and 16.09 respectively.



(a) Number of nodes generated on scenario 2. (b) Start node's depth on scenario 2.

Figure 6.16: Histogram plots of the 50 000 trees generated each for elliptic and circular funnels on scenario 2. For ellipse and circle strategies, mean node counts are 45.88 and 77.74; and the mean start node's depths are 18.51 and 35.19 respectively.

6.2.2 Motion Control Results

In this thesis, we propose an alternative trajectory-free planning and control schema to the schema proposed in [2]. We use some performance metrics, as described below, to conduct a comparative analysis of the control stages of the two methodologies. Note that these metrics do not contain any metric that is purely related to the node generation stage, since those are already covered in Section 6.2.1.

Mission Duration: The time that passes until the USV reaches the goal position

Average Speed: The average speed of the USV during its mission

Path Length: The distance covered by the USV until it reaches the goal position

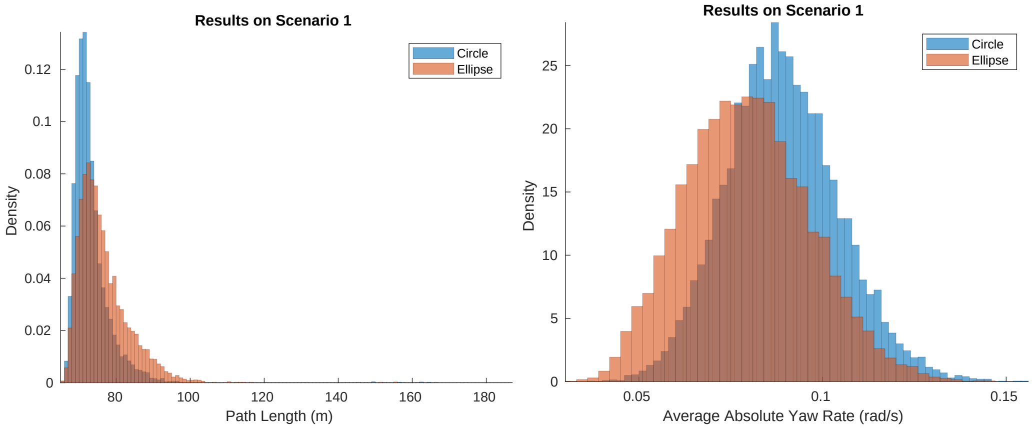
Average Absolute Yaw Rate: It is a measure of the rotation made by the USV during its mission. In order to make it independent of the mission duration, it is normalized with the mission duration as shown in (6.1). When it is considered for the same start and goal positions, the lower this metric is, the better.

$$\frac{1}{t_{end} - t_{start}} \int_{t_{start}}^{t_{end}} |\omega| \, dt \tag{6.1}$$

For performance comparison, a subset of the 200 000 trees recorded in tree generation tests given in Section 6.2.1 is used with the kinematic simulation setup; and the commands and the resultant trajectories are recorded for analysis. The test subset consists of a total of 40 000 simulations, equally divided between scenarios; and elliptical and circular funnels. For elliptical funnels, the controller parameters used in the simulations are the same as the final parameters used in the real USV. For circular funnels, the controller proposed in [2] is utilized with the equivalent parameters based on the analysis in Section 3.2.1. Used controller parameters are summarized in Table 6.4.

Table 6.4: Controller parameters used in performance tests.

Funnel	Control Policy	Period (s)	k_v | k_ρ	k_α	v_{sat} (m/s)	ω_{sat} (rad/s)
Elliptical	(3.29)	0.05	0.2	2.0	0.8	0.4
Circular	[2], (2.1)	0.05	0.4	2.0	0.8	0.4

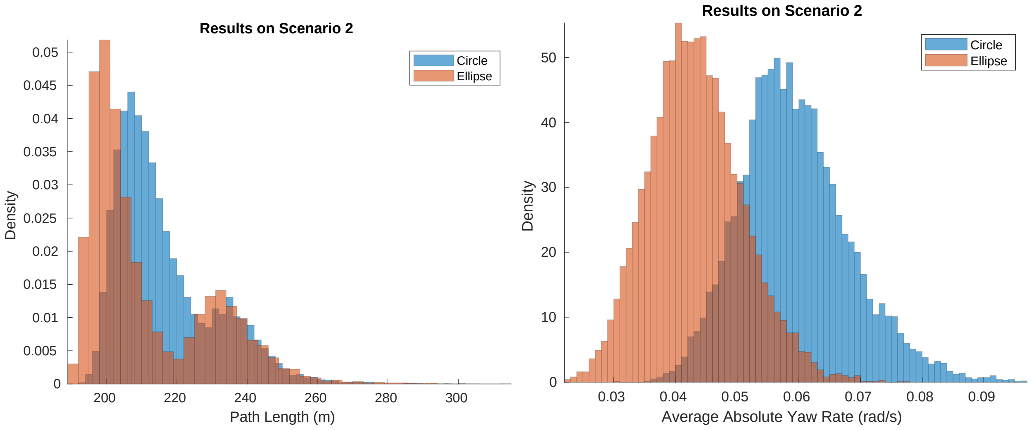


(a) Path length on scenario 1.

(b) Average absolute yaw rate on scenario 1.

Figure 6.17: Histogram plots of the path length and average absolute yaw rate metrics for elliptic and circular funnel cases on scenario 1. For ellipse and circle strategies, mean path lengths are 76.54 and 73.20 meters; and mean average absolute yaw rates are 0.0795 and 0.0891 rad/s respectively.

Inspecting the histograms in Figures 6.17, 6.18, 6.20 and 6.21, the results for path length and mission duration metrics are close, and the winner depends on the scenario. For the average speed metric, the elliptic controller has a slight advantage over the circular one for both scenarios. Lastly, we can see that the elliptical control and

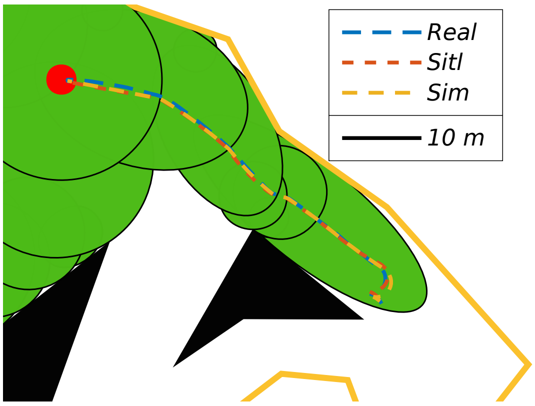


(a) Path length on scenario 2.

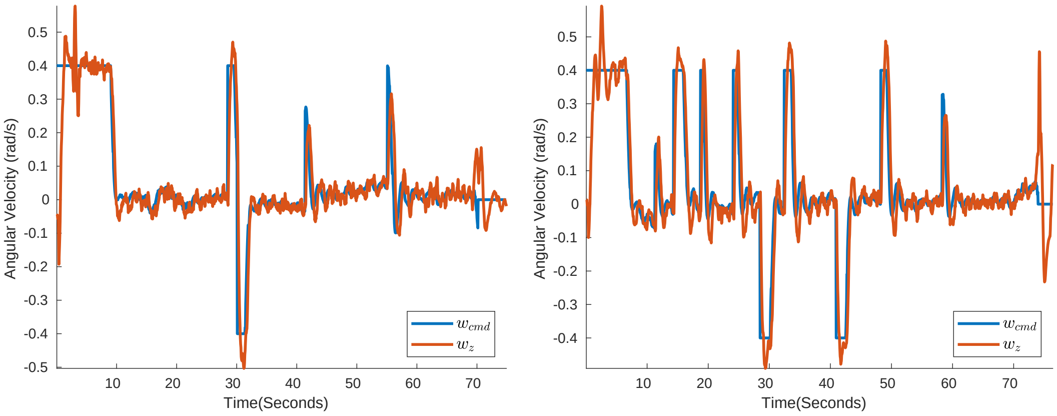
(b) Average absolute yaw rate on scenario 2.

Figure 6.18: Histogram plots of the path length and average absolute yaw rate metrics for elliptic and circular funnel cases on scenario 2. For ellipse and circle strategies, mean path lengths are 211.55 and 216.37 meters; and mean average absolute yaw rates are 0.0433 and 0.0594 rad/s respectively. The multi-peak distribution of the path length in scenario 2 is due to scenario 2 having two distinct homotopy classes of paths with comparable probabilities.

planning schema has a clear advantage on average absolute yaw rate metric independent from the scenario. This advantage can also be seen in the example realizations given in Figure 6.19, where it is easy to see the surplus yaw-rate commands given by the circular controller for the same start and goal configurations.

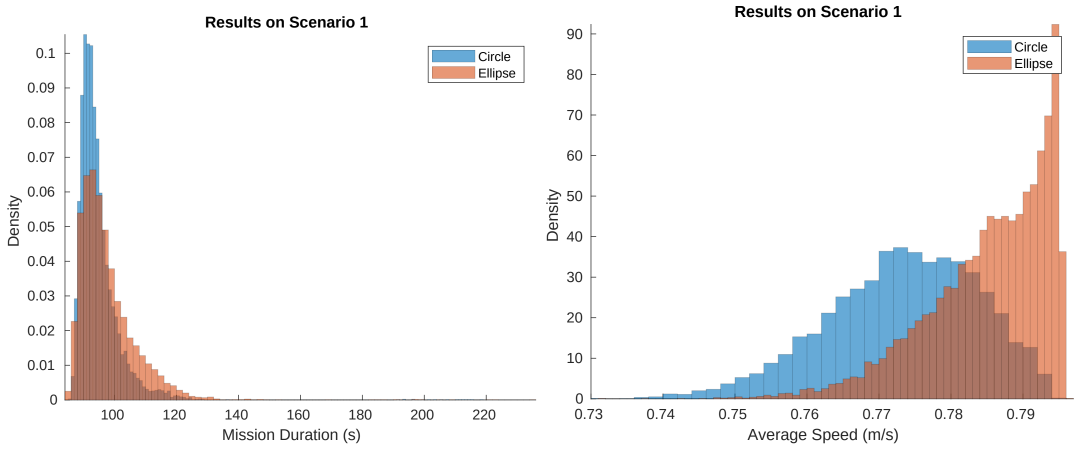


(a) An elliptic realization and followed path. (b) A circular realization and followed path.



(c) Angular velocity response on the elliptic case. (d) Angular velocity response on the circular case.

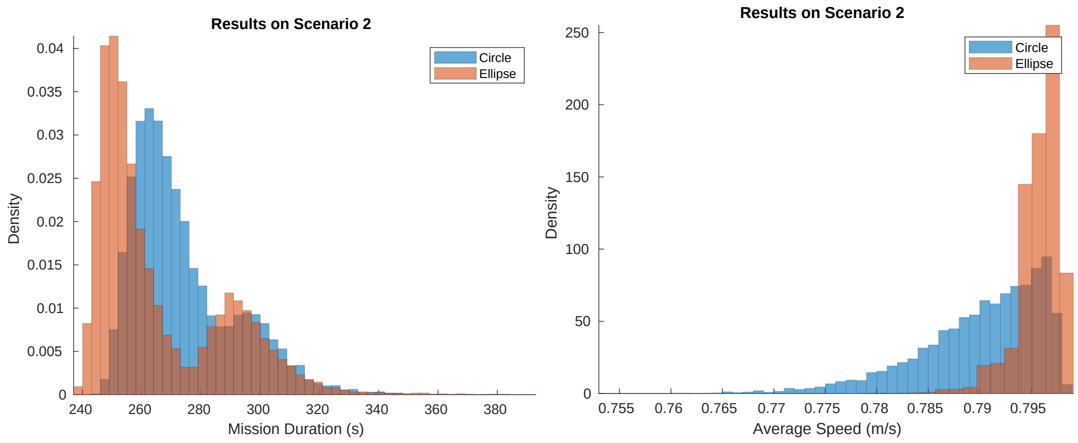
Figure 6.19: Paths and angular velocity responses from a real experiment carried on the damn lake in the Yalincak province of METU. Gains $k_v = 0.2$ and $k_\alpha = 2$ are used in all realizations.



(a) Mission duration on scenario 1.

(b) Average speed on scenario 1.

Figure 6.20: Histogram plots of the mission duration and average speed metrics for elliptic and circular funnel cases on scenario 1. For ellipse and circle strategies, mean mission durations are 97.54 and 94.74 seconds; and mean average speeds are 0.7847 and 0.7728 m/s respectively.



(a) Mission duration on scenario 2.

(b) Average speed on scenario 2.

Figure 6.21: Histogram plots of the mission duration and average speed metrics for elliptic and circular funnel cases on scenario 2. For ellipse and circle strategies, mean mission durations are 265.83 and 274.03 seconds; and mean average speeds are 0.7958 and 0.7896 m/s respectively.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

In this thesis, we proposed a motion planning and control scheme for unmanned surface vehicles with low-level forward velocity and yaw rate controllers. We demonstrated the proposed scheme's effectiveness on a novel USV and simulated versions of it. The proposed schema sequentially composes the partial feedback linearized controllers tailored for elliptic funnels in two stages: tree generation and motion control. The tree generation stage generates a tree starting from the goal position and expanding randomly in free configuration space until a specified starting position is contained within a funnel. When the initial position of the USV is connected to the goal funnel with the generated elliptic funnel tree, the motion control stage is activated, steering the USV from node to node until the goal position is reached.

The partial feedback linearized controllers tailored for each funnel ensures the USV's elliptic distance (ρ) to the active funnel is non-increasing, not permitting any collisions with the obstacles. Additionally, the stationary elliptic distance is possible only momentarily since the control on the yaw axis pushes the non-positive rate of change in the negative direction, ensuring convergence to the funnel center and sequentially to the goal position.

Our algorithm operates on 2D maps defined by polygons and can contain both convex and non-convex shapes. Using this environment representation, we can model the real-world workspaces the USV operates on without being over-conservative. The unicycle motion model we assume for the controlled agent can accurately model the motion carried for a wide range of surface vehicles. This lets us delegate the low-level controls of the USV to the onboard controllers dedicated to its maneuverability, enabling us to work with a wide range of control agents.

7.2 Future Work

In this study, the randomly generated ellipse tree from the goal position is directly used without measuring or optimizing its quality, focusing on the motion control stage. However, the fast nature of the tree generation algorithm could allow procedures such as re-planning or post-processing in the tree generation stage. In the future, it is possible to design quality metrics for the generated tree and optimize it before using it in the motion control stage, resulting in more direct paths from start to goal. Another possible solution to this problem is updating tree generation steps so that the generated funnel tree already complies with the desired quality measures.

Another point to mention is that the implemented algorithm works in a static environment representation where we assume there won't be any obstacles inside the generated funnels. This was a fair assumption on the lake that we were testing our algorithm on since no other agents were using the lake. But for a more generalized use case, handling dynamic obstacles is a must. In the future, it is possible to make the funnel tree a reactive one where the tree is updated continuously depending on the other agent's possible trajectories.

Another possible improvement is the creation of a sequential composition framework that would enable easier deployment and comparison of algorithms using the sequential composition as a basis.

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, pp. 534 – 555, 1999.

[2] E. Ege and M. 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] Z. Liu, Y. Zhang, X. Yu, and C. Yuan, “Unmanned surface vehicles: An overview of developments and challenges,” *Annual Reviews in Control*, vol. 41, pp. 71–93, 2016.

[4] A. Vagale, R. Oucheikh, R. T. Bye, O. L. Osen, and T. I. Fossen, “Path planning and collision avoidance for autonomous surface vehicles I: a review,” *Journal of Marine Science and Technology (Japan)*, vol. 26, pp. 1292–1306, 12 2021.

[5] S. Savitz, I. Blickstein, P. Buryk, R. W. Button, P. DeLuca, J. Dryden, J. Mastbaum, J. Osburg, P. Padilla, A. Potter, C. C. Price, L. Thrall, S. K. Woodward, R. J. Yardley, and J. M. Yurchak, “U.s. navy employment options for unmanned surface vehicles (USVs),” 2013.

[6] “Annual overview of marine casualties and incidents,” Technical report, European Maritime Safety Agency, 12 2021.

[7] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

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

[9] S. M. LaValle and J. James J. Kuffner, “Randomized Kinodynamic Planning,”

The International Journal of Robotics Research, vol. 20, no. 5, pp. 378–400, 2001.

[10] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2719–2726, 1997.

[11] E. Camacho and C. Bordons, *Model Predictive Control*, vol. 13. Springer, 01 2004.

[12] O. Amidi and C. Thorpe, "Integrated mobile robot control," *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 1388, pp. 504 – 523, 1991.

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

[14] D. E. Koditschek and E. Rimon, "Robot navigation functions on manifolds with boundary," *Advances in Applied Mathematics*, vol. 11, no. 4, pp. 412–442, 1990.

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

[16] C. Connolly, J. Burns, and R. Weiss, "Path planning using Laplace's equation," in *Proceedings., IEEE International Conference on Robotics and Automation*, pp. 2102–2106 vol.3, 1990.

[17] Y. Golan, S. Edelman, A. Shapiro, and E. Rimon, "Online robot navigation using continuously updated artificial temperature gradients," *IEEE Robotics and Automation Letters*, vol. 2, pp. 1280–1287, 2017.

[18] 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.

[19] O. K. Karagoz, S. Atasoy, and M. M. Ankarali, "MPC-Graph: Feedback motion planning using sparse sampling based neighborhood graph," in *2020 IEEE/RSJ*

International Conference on Intelligent Robots and Systems (IROS), pp. 6797–6802, 2020.

[20] F. Golbol, M. M. Ankarali, and A. Saranli, “RG-Trees: Trajectory-free feedback motion planning using sparse random reference governor trees,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6506–6511, 2018.

[21] S. R. Lindemann and S. M. La Valle, “Simple and efficient algorithms for computing smooth, collision-free feedback laws over given cell decompositions,” *International Journal of Robotics Research*, vol. 28, no. 5, pp. 600–621, 2009.

[22] S. Lavalle and J. Kuffner, “Rapidly-exploring random trees: Progress and prospects,” *Algorithmic and computational robotics: New directions*, 1 2000.

[23] 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.

[24] C. Howie, L. Kevin, H. Seth, K. George, B. Wolfram, K. Lydia, and T. Sebastian, *Principles of Robot Motion Theory, Algorithms, and Implementations*, vol. 7. 2001.

[25] D. E. Koditschek, “Task encoding: Toward a scientific paradigm for robot planning and control,” *Robotics and Autonomous Systems*, vol. 9, no. 1, pp. 5–39, 1992.

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

[27] D. H. Eberly, “Geometric tools.” `https://github.com/davideberly/GeometricTools/releases/tag/GTE-version-5.14`, 2021.

[28] M. Aicardi, G. Casalino, A. Bicchi, and A. Balestrino, “Closed Loop Steering of Unicycle-like Vehicles via Lyapunov Techniques,” *IEEE Robotics and Automation Magazine*, vol. 2, no. 1, pp. 27–35, 1995.

[29] Open Source Robotics Foundation, “ROS Concepts.” `http://wiki.ros.org/ROS/Concepts`, 2021.

[30] Open Source Robotics Foundation, "ROS Introduction." `http://wiki.ros.org/ROS/Introduction`, 2021.

[31] C. F. F. Karney, "Algorithms for geodesics," *Journal of Geodesy*, vol. 87, pp. 43–55, jun 2012.

[32] J. S. Willners, Y. Carreno, S. Xu, T. Łuczyński, S. Katagiri, J. Roe, Èric Pairet, Y. Petillot, and S. Wang, "Robust underwater SLAM using autonomous relocalisation," *IFAC-PapersOnLine*, vol. 54, no. 16, pp. 273–280, 2021. 13th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles CAMS 2021.

[33] S. Pedersen, J. Liniger, F. F. Sørensen, K. Schmidt, M. von Benzon, and S. S. Klemmensen, "Stabilization of a ROV in three-dimensional space using an underwater acoustic positioning system," *IFAC-PapersOnLine*, vol. 52, no. 17, pp. 117–122, 2019. 7th IFAC Symposium on System Structure and Control SSSC 2019.

[34] S. Lack, E. Rentzow, and T. Jeinsch, "Experimental parameter identification for an open-frame ROV: Comparison of towing tank tests and open water self-propelled tests," *IFAC-PapersOnLine*, vol. 52, no. 21, pp. 271–276, 2019. 12th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles CAMS 2019.

[35] E. Einarsson and A. Lipenitis, "Model predictive control for the BlueROV2, theory and implementation," Master's thesis, Aalborg University, 2020.

[36] C. Spurgeon and J. Zimmerman, *Ethernet: The Definitive Guide (2nd ed.)*. O'Reilly Media, 01 2014.

[37] RAK Wireless, *PowerLine Module, LX200V20*, 7 2014. Rev. 1.2.

[38] Ubiquiti, *UniFi AC Mesh Quick Start Guide*, 7 2014. Rev. 1.0.

[39] RFDesign, *RFD900x Datasheet*, 3 2017. Rev. 1.0.

[40] S. R. Saghravani, S. Mustapha, and S. F. Saghravani, "Accuracy comparison of RTK-GPS and automatic level for height determination in land surveying," *MASAUM Journal Of Reviews and Surveys*, vol. 1, pp. 10–13, 01 2009.

[41] T. Baybura, I. Tiryakioglu, M. A. Ugur, H. I. Solak, and S. Safak, "Examining the accuracy of network RTK and long base RTK methods with repetitive measurements," *Journal of Sensors*, p. 12, Nov 2019.

[42] A. Koubâa, A. Allouch, M. Alajlan, Y. Javed, A. Belghith, and M. Khalgui, "Micro air vehicle link (MAVlink) in a nutshell: A survey," *IEEE Access*, vol. 7, pp. 87658–87680, 2019.

[43] PX4 Community, "Pixhawk Flight Controller." `https://docs.px4.io/main/en/flight_controller/mro_pixhawk.html`, June 2022.

[44] Ardupilot Community, "Ardupilot repository." `https://github.com/ArduPilot/ardupilot/releases/tag/ArduSub-4.1.0`, June 2022.

[45] ArduPilot Dev Team, "Ardupilot Navigation, Extended Kalman Filter Overview." `https://ardupilot.org/copter/docs/common-apm-navigation-extended-kalman-filter-overview.html`, June 2022.

[46] ArduPilot Dev Team, "Sensor Drivers." `https://ardupilot.org/dev/docs/code-overview-sensor-drivers.html`, June 2022.

[47] ArduPilot Dev Team, "Extended Kalman Filter Navigation Overview and Tuning." `https://ardupilot.org/dev/docs/extended-kalman-filter.html`, June 2022.

[48] ArduPilot Dev Team, "EKF2 Estimation System." `https://ardupilot.org/dev/docs/ekf2-estimation-system.html`, June 2022.

[49] ArduPilot Dev Team, "EKF3 Affinity and Lane Switching." `https://ardupilot.org/dev/docs/common-ek3-affinity-lane-switching.html`, June 2022.

[50] M. Vignati, N. Debattisti, M. L. Bacci, and D. Tarsitano, "A software-in-the-loop simulation of vehicle control unit algorithms for a driverless railway vehicle," *Applied Sciences*, vol. 11, no. 15, 2021.