A Generic Guidance Navigation and Control Framework For Marine Vehicles

Emir Cem Gezer

University of Rhode Island, emircem.gezer@gmail.com

A GENERIC GUIDANCE NAVIGATION AND CONTROL FRAMEWORK FOR MARINE VEHICLES

BY

EMİR CEM GEZER

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

OCEAN ENGINEERING

UNIVERSITY OF RHODE ISLAND

2022

MASTER OF SCIENCE THESIS

OF

EMİR CEM GEZER

APPROVED:

Thesis Committee:

Major Professor	Mingxi Zhou
	Christopher Roman
	Paolo Stegagno
	Nasser H. Zawia
	DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2022

ABSTRACT

Generic open-source software frameworks are significantly valuable for robotics research and development. The emergence of low-cost Autonomous Underwater Vehicles (AUVs) and Autonomous Surface Vehicles (ASVs) is speeding up because of the increased availability of consumer-grade parts such as actuators, pressure housings, and single-board computers. With the intention of providing off-the-shelf AUV/ASV control, guidance, and operation solution to kick-start future marine vehicle projects, this thesis presents a new open-source framework called Robot Operating System Marine Vehicle Packages (ROS-MVP). MVP is tightly integrated with the ROS infrastructure and the state-of-the-art ROS packages and can be easily configured for different marine robots. The framework consists of two main components: a low-level vehicle controller, and a mission controller with a plugin-based behavior interface. MVP-Control works by running a control allocation with a quadratic programming solver. MVP mission controller orchestrates the behaviors using a finite state machine based process called MVP-Helm. MVP mission controller is packaged with common behaviors such as path following, depth tracking, periodic surfacing, etc. This thesis presents the details of the ROS-MVP framework design, integration, and field test results for an AUV in Narragansett Bay, Rhode Island.

ACKNOWLEDGMENTS

First and foremost, I want to thank my advisor Mingxi Zhou for giving me the opportunity and the energy for my academic growth and teaching me things that I have never thought of. None of the research presented in this thesis would be possible if my peers in the Smart Ocean System laboratory weren't there. So Lin Zhao, Raymond Turrisi, and William McConnell thank you guys so much. I must add, I'm so grateful for the existence of the coffee machine in the lab.

For their help in the recovery operations for retrieving the lost equipment from the muddy bottom of the pond, I want to thank Brian Caccioppoli, Kristopher Krasnosky, David Casagrande, and the diving crew Anya Hanson, Alexa Runyan, and Myles Wagner. And a moment of silence for Brian's phone that fell to the lake whilst capturing the recovery operation.

Lastly, I'd want to express my gratitude to the members of my thesis committee, Christopher Roman, Paolo Stegagno, and Chengzhi Yuan, for their guidance and feedback on the thesis.

DEDICATION

To my friends and family who have cared for me, had memories with me, and carried me to this day.

TABLE OF CONTENTS

ABSTRACT ii

ACKNOWLEDGMENTS iii

DEDICATION iv

TABLE OF CONTENTS v

LIST OF FIGURES vii

LIST OF TABLES ix

CHAPTER

1 Introduction 1

1.1 Background 1

1.2 ASVs and AUVs 2

1.3 Related Work 5

1.4 Summary of Contribution 7

List of References 8

2 Marine Vehicle Packages (MVP) 12

2.1 Overview of the framework 12

2.2 Low-level controller - MVP-Control 14

2.2.1 Control Law 15

2.2.2 Control Allocation 15

2.2.3 Configuration 17

2.3 Mission controller - MVP Helm 21

Page

2.3.1 Finite State Machine 23

2.3.2 Behaviors 24

2.4 MVP Behaviors 28

2.4.1 Path Following 28

2.4.2 Waypoint Tracking 33

2.4.3 Depth Tracking 34

2.4.4 Periodic Surfacing 35

2.4.5 Timer Behavior 36

2.5 Robot Localization Integration 36

List of References 40

3 Simulation Validation 42

3.1 Stonefish Simulation Setup 44

3.1.1 Integration 44

3.1.2 Configuration 45

3.2 Results 45

List of References 52

4 Field Test 54

4.1 Overall System Architecture 54

4.2 Experiment Setup 56

4.3 Results 57

5 Conclusion and Future Work 65

List of References 66

BIBLIOGRAPHY 67

LIST OF FIGURES

Figure		Page
1	Roboat in the Amsterdam Waterways	3
2	Saildrone surface vehicle	4
3	Hugin 1000 AUV	5
4	MVP System Diagram	13
5	MVP-Control flow diagram	14
6	Example TF Tree for MVP-Control	19
7	Transform Tree for ALPHA AUV visualized in RViZ.	19
8	HVP-Helm flow diagram	22
9	Example Finite State Machine for MVP-Helm.	24
10	Line of Sight guidance algorithm variables	30
11	Periodic surfacing behavior internal stages.	35
12	Robot Localization data flow diagram.	38
13	Robot Localization common TF tree.	39
14	ALPHA AUV 3D Model	43
15	Stonefish simulator and RViZ	44
16	Heading control results from simulation.	46
17	Heading control error.	46
18	Surge control results from simulation.	47
19	Surge control error in the simulation.	47
20	Vehicle trajectory during multi-DOF control tests in simulation.	48

Figure **Page**

21 MVP-Controller results in Stonefish simulator for controlling X, Y, Z position and pitch simultaneously. 49

22 MVP-Controller results in Stonefish simulator for controlling Z position, yaw, and pitch simultaneously. 50

23 Trajectory Tracking results in the simulation. 51

24 Cross track error of the trajectory tracking in simulation 52

25 ALPHA AUV Revision 2 in Narragansett Bay 54

26 ALPHA AUV Revision 1 - Diagram 55

27 ALPHA AUV Revision 2 - Diagram 55

28 ALPHA AUV devices diagram 56

29 Field trial results from Narragansett Bay on 2022-06-28 with ALPHA Revision 1. 59

30 Cross track error for mission presented in Fig. 29 60

31 Field trial results from Narragansett Bay on 2022-09-15 with ALPHA Revision 2. 61

32 Cross track error for mission presented in Fig. 31 62

33 Long mission result from Narragansett Bay on 2022-09-15 with ALPHA Revision 2. 63

34 Cross track error for mission presented in Fig. 33 64

LIST OF TABLES

Table		Page
1	MVP-Control ROS Node parameters.	18
2	MVP-Helm FSM Parameter Descriptions.	23
3	MVP-Helm Behavior Configuration.	28
4	Path Following behavior parameters	32
5	Depth Tracing Behavior parameters.	35
6	Periodic surfacing parameters.	36
7	Timer behavior parameters.	36
8	Common navigation sensors and their outputs.	37
9	Behavior parameters for presented results	57

CHAPTER 1

Introduction

1.1 Background

The oceans cover about 70% of the earth's surface and only a small fraction of it had been mapped [1]. It is a great resource that provides energy, food, jobs, and many more. Knowing it enables us to better maintain and manage the ocean sources, leading to a sustainable Blue Economy with an estimated worth of US$24 in 2015 [2]. We can learn more about the ocean through conducting exploration, observation, monitoring, and sampling, and thanks to improved technology, the tools for these tasks are improving and becoming more effective and efficient. Marine robotic platforms, namely Autonomous Underwater Vehicles (AUV) and Autonomous Surface Vehicles (ASV) have enabled us to reach further, dive deeper, and see what was not possible to see before.

Marine robots are versatile oceanography instruments used for ocean sampling, data collection, manipulation, inspection, and many other tasks. They are categorized by the use cases and their designs. Remotely Operated Vehicles (ROVs) are typically box-shaped underwater platforms that, as their name suggests, are controlled remotely by a pilot from a surface ship. Communication and power are transmitted with a tether cable. They are capable of focused surveys of points of interest, such as underwater archaeological sites, shipwrecks, transatlantic communication lines, etc. Autonomous Underwater Vehicles (AUVs), on the other hand, operate based on pre-programmed scripts with limited human intervention. Depending on the application, they can be in different shapes and configurations. For example, Slocum glider [3] has wings on either side and is powered by a buoyancy engine. This setup enables them to be deployed for an extended duration from weeks to months depending on the battery types [4]. In work [5], the gliders are recovered after 330 days of operation. In contrast, torpedo-shaped AUVs are agile platforms that are suitable options for seabed surveys and coastal water sampling. At the water surface, we have Autonomous Surface Vehicles (ASVs). They are great tools for seabed surveys and air-water interface studies. Some research has also been done to investigate the application of using ASVs for transportation tasks [6].

The technology advancements in semiconductors, power systems, and actuators enabled the development of more sophisticated robotic systems [7]. As a result, robots could be outfitted with power-hungry sensors, manipulators, and actuators, resulting in longer operating times. Thus, small form factor design is more feasible and achievable than before. It fundamentally has changed decision-making and opened the door for developers to design solutions for autonomy.

Open-source robotic frameworks such as ROS[8], MOOS [9], and etc played a key role in developing autonomous marine vehicles. They provided the basic building blocks for creating a robotic platform. The users of these frameworks were able to identify problems and offer solutions, and as a result, these frameworks grew steadily more dependable and approachable for new users. But marine robotics guidance, navigation, and, control (GNC) systems started to fall behind. Thus, this thesis introduces a new framework that offers GNC functionality with simple-to-maintain individual components.

1.2 ASVs and AUVs

In the context of marine robotics, Autonomous Surface Vehicles (ASVs), are robotic platforms that operate on the sea surface. They can operate independently or complementary to another vessel. For data collection in hazardous areas where crewed boats and ships are not permitted to visit, researchers at the Woods Hole Oceanographic Institution (WHOI) designed an autonomous kayak, The WHOI

Jetyak [10], which can be deployed from a small boat to carry side tasks when the mother ship conducts other tasks. In 2020, a team of researchers at Massachusetts Institute of Technology (MIT) created the Roboat II (in Fig. 1), for urban waterway uses such as trash collection and transportation of goods [6].

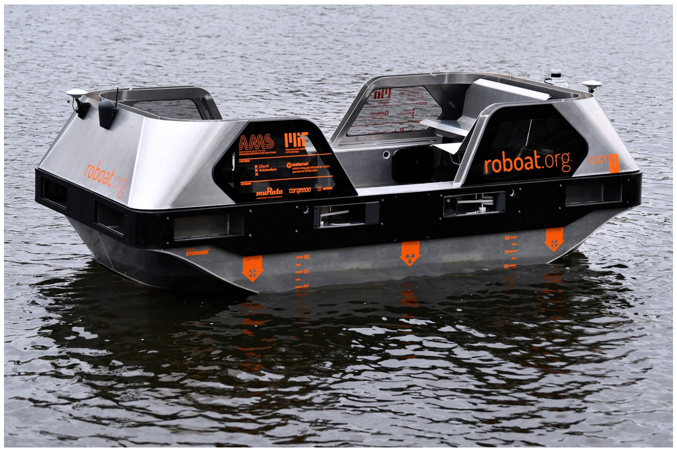


Figure 1: Roboat in the Amsterdam waterways (October 27, 2021. REUTERS/Piroschka van de Wouw)

Some ASVs are built with extended operating ranges in mind. The Wave Glider ASV uses a tethered underwater glider with a horizontal fin array to leverage ocean waves for propelling the vehicle and solar panels to harvest energy which results in a long endurance of several months. For instance, The Wave Glider ASV [11] ran continuously in open waters for 169 days. The Saildrone [12] is another long-endurance ASV that is mainly propelled using a rigid sail. (in Fig. 2). In April 2015 [13], two saildrones were deployed from Dutch Harbor, Alaska and they traveled a total of 15,525 km in 97 days.

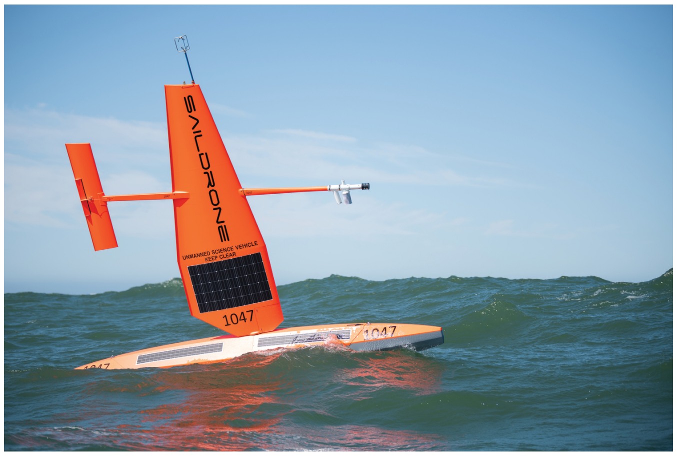


Figure 2: Saildrone moving along the wave (Image from noaa.gov, Credits Saildrone INC.)

Autonomous Underwater Vehicles (AUVs) are underwater platforms that require limited human interventions. They vary in shape, size, and propulsion system depending on the applications. They are often tasked for seabed mapping [14], mine countermeasures [15], environmental surveys [16], continuous monitoring of underwater structures or marine environments.

The very first AUV, SPURV (Self-Propelled Underwater Research Vehicle), was built in 1957 [17]. It had a carrying capacity of 45 kg, a 3 m length, a 50 cm diameter, and a 5.5 hour of endurance. SPURV was controlled by the mothership through acoustic communication. Now, Hugin 1000 AUV [18] (in Fig. 3) can offer endurance of 24 hours at a cruising speed of 5 knots with more sensors on board.

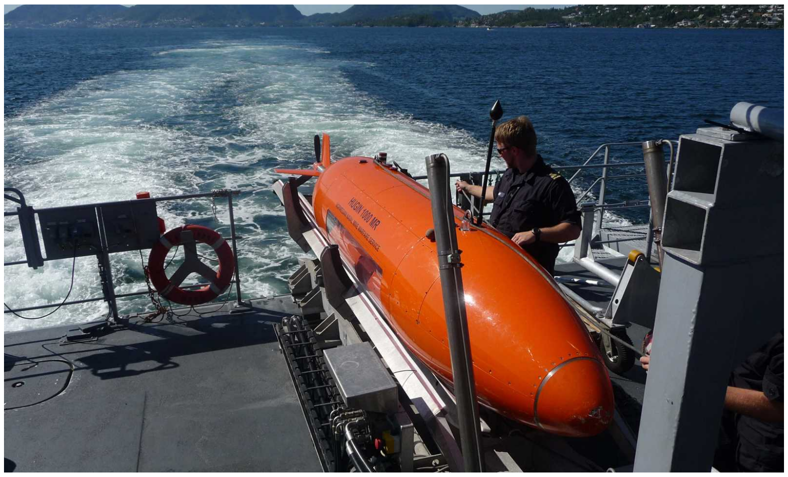


Figure 3: Hugin 1000 AUV on the ship deck before deployment (Courtesy of Kongsberg Maritime)

Recently, the emergence of low-cost AUVs is speeding up because of the increased availability of consumer-grade parts including actuators, pressure housings, and single-board computers. Increased accessibility on hardware has enabled research groups from all around the world to come up with new AUV designs for various applications [19] [20] [21].

1.3 Related Work

The software is an essential part of a marine vehicle besides the hardware. Currently, there are several existing marine vehicle guidance, navigation, and control software middlewares running on different kinds of marine robots. The MOOS-IvP [22] is the most notable one. It comprises two main components: the MOOS [9] for inter-process communication (IPC), and IvP Helm for guidance and autonomy. The MOOS-IvP framework has sensor fusion capabilities for localization through `pNav`, a navigation stack program. Users of MOOS-IvP can introduce custom software while keeping the existing packages for their ecosystem by using the Mission Oriented Operating Suite (MOOS). The work in [23] is a good example of using the MOOS-IvP, where the authors successfully customized the MOOS-IvP framework and implemented a vessel tracking application with a Bluefin-21 AUV. Moreover, in [24], an autonomy payload for Bluefin Sandshark AUV was developed using the MOOS-IvP framework. However, MOOS-IvP does not have a generic low-level vehicle controller. Furthermore, it is not readily compatible with ROS due to its fundamental differences. Therefore, advancements in the ROS ecosystem can not be reflected easily in the MOOS-IvP ecosystem, and the gap is widening between them as robotics communities lean towards ROS [25]. Although there are some efforts to bridge these two frameworks together, such as the MOOS-IvP ROS bridge[26] and ROS-IvP [27], combining two different architectures remains to be a challenging task.

COLA2 [28] is a notable ROS-based framework specialized in AUVs [8]. COLA2 has full-stack guidance, navigation, and control solutions and it currently runs two AUVs, Sparus II [29] and Girona 500 [30]. Nonetheless, COLA2 is proprietary software that is not available to adopt for different marine vehicles.

Some robotics simulation projects come with vehicle guidance, navigation, and control packages besides the MOOS-IvP and COLA2. For instance, the UUV Simulator [31] project contains a simple thruster control allocation implementation, and its successor, Project DAVE [32], has more features, such as localization stack and perception sensor simulators available. Finally, using the vehicle controller inside the UUV Simulator imposes a high coupling problem that increases the complexity of the maintenance process, and the transition from simulation and the actual operation is not streamlined.

In addition to these marine vehicle frameworks, it is important to note some of the important robotics libraries in the ROS ecosystem since they have a great impact on robotics development. To begin with, the ROS Transform Library[33], namely TF, is one of the most important libraries in the ROS ecosystem that handles coordinate frame transformations in a scalable fashion. It has an RViZ[1] plugin to visualize the transform tree and evidently makes it easy to diagnose any problem in its setup. The most frequent cause of software mistakes is failing to keep track of coordinate frame transformations. However, decoupling and distributing the coordinate frames among the robot programs is simple with TF. Consequently, the likelihood of software mistakes brought on by coordinate frame modifications is decreased. Moreover, the Robot Localization [34] package provides a localization solution for mobile robots. It combines multiple motion sensor data and fuses them into a single odometry source. It uses TF to transform sensory information and publishes the vehicle pose as a TF tree coordinate frame link. In conclusion, the libraries mentioned above and numerous more helped to modify the limitations of design and open up new avenues for creating robotics libraries.

1.4 Summary of Contribution

The main contribution of this thesis is the design of a new open-source ROS-based marine vehicle guidance, navigation, and control framework, namely Marine Vehicle Packages (MVP). MVP integrates state-of-the-art robotic software packages and commonly used control and guidance algorithms for marine vehicles into the ROS ecosystem.

ROS-MVP consists of two major components. MVP-Control enables users to easily integrate a low-level vehicle controller into their vehicles by creating configuration files other than changing the source code. On the higher level, MVP-Helm orchestrates behaviors using a finite state machine. It doesn't run a solver to decide which behavior to execute, instead, it uses a priority pool to pick desired control input by using user-defined priority levels.

ROS-MVP seeks to provide a modular, scalable, and highly customizable

[1]RViZ is a visualization tool from ROS ecosystem

framework for building GNC systems for new marine robotic platforms. MVP-Control and MVP-Helm packages are two separate packages that don't depend on each other. Therefore, MVP Mission packages can be used with any other vehicle controller software. The architecture also allows the introduction of new behaviors through the MVP behavior interface.

List of References

[1] L. Mayer, M. Jakobsson, G. Allen, B. Dorschel, R. Falconer, V. Ferrini, G. Lamarche, H. Snaith, and P. Weatherall, "The nippon foundation—gebco seabed 2030 project: The quest to see the world's oceans completely mapped by 2030," *Geosciences*, vol. 8, no. 2, p. 63, 2018.

[2] O. Hoegh-Guldberg, "Reviving the ocean economy: the case for action," 2015.

[3] O. Schofield, J. Kohut, D. Aragon, L. Creed, J. Graver, C. Haldeman, J. Kerfoot, H. Roarty, C. Jones, D. Webb, *et al.*, "Slocum gliders: Robust and ready," *Journal of Field Robotics*, vol. 24, no. 6, pp. 473–485, 2007.

[4] D. L. Rudnick, R. E. Davis, C. C. Eriksen, D. M. Fratantoni, and M. J. Perry, "Underwater gliders for ocean research," *Marine Technology Society Journal*, vol. 38, no. 2, pp. 73–84, 2004.

[5] C. Jones, B. Allsup, and C. DeCollibus, "Slocum glider: Expanding our understanding of the oceans," in *2014 Oceans - St. John's*, 2014, pp. 1–10.

[6] W. Wang, T. Shan, P. Leoni, D. Fernández-Gutiérrez, D. Meyers, C. Ratti, and D. Rus, "Roboat ii: A novel autonomous surface vessel for urban environments," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 1740–1747.

[7] X. Wang, J. Shang, Z. Luo, L. Tang, X. Zhang, and J. Li, "Reviews of power systems and environmental energy conversion for unmanned underwater vehicles," *Renewable and Sustainable Energy Reviews*, vol. 16, no. 4, pp. 1958–1970, 2012.

[8] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[9] P. M. Newman, "Moos-mission orientated operating suite," 2008.

[10] P. Kimball, J. Bailey, S. Das, R. Geyer, T. Harrison, C. Kunz, K. Manganini, K. Mankoff, K. Samuelson, T. Sayre-McCord, F. Straneo, P. Traykovski, and

H. Singh, “The whoi jetyak: An autonomous surface vehicle for oceanographic research in shallow or dangerous waters,” in *2014 IEEE/OES Autonomous Underwater Vehicles (AUV)*, 2014, pp. 1–7.

[11] T. Daniel, J. Manley, and N. Trenaman, “The wave glider: enabling a new approach to persistent ocean observation and research,” *Ocean Dynamics*, vol. 61, no. 10, pp. 1509–1520, 2011.

[12] C. W. Mordy, E. D. Cokelet, A. De Robertis, R. Jenkins, C. E. Kuhn, N. Lawrence-Slavas, C. L. Berchok, J. L. Crance, J. T. Sterling, J. N. Cross, *et al.*, “Advances in ecosystem research: Saildrone surveys of oceanography, fish, and marine mammals in the bering sea,” *Oceanography*, vol. 30, no. 2, pp. 113–115, 2017.

[13] C. Meinig, N. Lawrence-Slavas, R. Jenkins, and H. M. Tabisola, “The use of saildrones to examine spring conditions in the bering sea: Vehicle specification and mission performance,” in *OCEANS 2015 - MTS/IEEE Washington*, 2015, pp. 1–6.

[14] D. W. Caress, H. Thomas, W. J. Kirkwood, R. McEwen, R. Henthorn, D. A. Clague, C. K. Paull, J. Paduan, K. L. Maier, J. Reynolds, *et al.*, “High-resolution multibeam, sidescan, and subbottom surveys using the mbari auv d. allan b,” *Marine habitat mapping technology for Alaska*, pp. 47–69, 2008.

[15] L. Freitag, M. Grund, C. Von Alt, R. Stokey, and T. Austin, “A shallow water acoustic network for mine countermeasures operations with autonomous underwater vehicles,” *Underwater Defense Technology (UDT)*, pp. 1–6, 2005.

[16] D. R. Yoerger, A. M. Bradley, B. B. Walden, H. Singh, and R. Bachmayer, “Surveying a subsea lava flow using the autonomous benthic explorer (abe),” *International Journal of Systems Science*, vol. 29, no. 10, pp. 1031–1044, 1998.

[17] H. Widditsch, “Spurv-the first decade,” WASHINGTON UNIV SEATTLE APPLIED PHYSICS LAB, Tech. Rep., 1973.

[18] R. Marthiniussen, K. Vestgard, R. Klepaker, and N. Storkersen, “Hugin-auv concept and operational experiences to date,” in *Oceans ’04 MTS/IEEE Techno-Ocean ’04 (IEEE Cat. No.04CH37600)*, vol. 2, 2004, pp. 846–850 Vol.2.

[19] C. Edge, S. S. Enan, M. Fulton, J. Hong, J. Mo, K. Barthelemy, H. Bashaw, B. Kallevig, C. Knutson, K. Orpen, *et al.*, “Design and experiments with loco auv: A low cost open-source autonomous underwater vehicle,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 1761–1768.

[20] D. A. Duecker, N. Bauschmann, T. Hansen, E. Kreuzer, and R. Seifried, “Hippocampusx–a hydrobatic open-source micro auv for confined environments,” in *2020 IEEE/OES Autonomous Underwater Vehicles Symposium (AUV)*. IEEE, 2020, pp. 1–6.

[21] A. Griffiths, A. Dikarev, P. R. Green, B. Lennox, X. Poteau, and S. Watson, “Avexis—aqua vehicle explorer for in-situ sensing,” *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 282–287, 2016.

[22] M. R. Benjamin, H. Schmidt, P. M. Newman, and J. J. Leonard, “Nested autonomy for unmanned marine vehicles with moos-ivp,” *Journal of Field Robotics*, vol. 27, no. 6, pp. 834–875, 2010.

[23] A. Wolek, J. McMahon, B. R. Dzikowicz, and B. H. Houston, “Tracking multiple surface vessels with an autonomous underwater vehicle: Field results,” *IEEE Journal of Oceanic Engineering*, vol. 47, no. 1, pp. 32–45, 2022.

[24] O. A. Viquez, E. M. Fischell, N. R. Rypkema, and H. Schmidt, “Design of a general autonomy payload for low-cost auv r&d,” in *2016 IEEE/OES Autonomous Underwater Vehicles (AUV)*, 2016, pp. 151–155.

[25] L. Zhang, R. Merrifield, A. Deguet, and G.-Z. Yang, “Powering the world’s robots—10 years of ros,” *Science Robotics*, vol. 2, no. 11, p. eaar1868, 2017.

[26] K. DeMarco, M. E. West, and T. R. Collins, “An implementation of ros on the yellowfin autonomous underwater vehicle (auv),” in *OCEANS 2011*. IEEE, 2011, pp. 1–7.

[27] M. Snyder, J. N. Weaver, and M. J. Bays, “Ros-ivp: Porting the interval programming suite into the robot operating system for maritime autonomy,” in *OCEANS 2016 MTS/IEEE Monterey*, 2016, pp. 1–6.

[28] N. Palomeras, A. El-Fakdi, M. Carreras, and P. Ridao, “Cola2: A control architecture for auvs,” *IEEE Journal of Oceanic Engineering*, vol. 37, no. 4, pp. 695–716, 2012.

[29] M. Carreras, J. D. Hernández, E. Vidal, N. Palomeras, D. Ribas, and P. Ridao, “Sparus ii auv—a hovering vehicle for seabed inspection,” *IEEE Journal of Oceanic Engineering*, vol. 43, no. 2, pp. 344–355, 2018.

[30] D. Ribas, N. Palomeras, P. Ridao, M. Carreras, and A. Mallios, “Girona 500 auv: From survey to intervention,” *IEEE/ASME Transactions on Mechatronics*, vol. 17, no. 1, pp. 46–53, 2012.

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

[32] M. M. Zhang, W.-S. Choi, J. Herman, D. Davis, C. Vogt, M. McCarrin, Y. Vijay, D. Dutia, W. Lew, S. Peters, and B. Bingham, "Dave aquatic virtual environment: Toward a general underwater robotics simulator," in *2022 IEEE/OES Autonomous Underwater Vehicle (AUV) Symposium*, 2022, pp. 1–8.

[33] T. Foote, "tf: The transform library," in *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, ser. Open-Source Software workshop, April 2013, pp. 1–6.

[34] T. Moore and D. Stouch, "A generalized extended kalman filter implementation for the robot operating system," in *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer, July 2014.

CHAPTER 2

Marine Vehicle Packages (MVP)

2.1 Overview of the framework

The MVP framework has two main components: MVP-Control and MVP-Mission. The overview of the MVP framework is shown in Figure 4. At the low level, The MVP-Control is responsible for computing a set of actuator commands to track the desired vehicle pose (position, velocity, or orientation). It runs a control allocation method [1] with quadratic programming solver under the hood. MVP Mission is a collection of packages made up of behaviors and a state-oriented mission controller called MVP-Helm. The MVP-Helm process is in charge of providing directions using behaviors (such as waypoint following, and depth tracking) that compute the desired vehicle pose for the MVP-Control process. Since the MVP-Helm and MVP-Control communicate over ROS topics, it is possible to replace the MVP-Control with other custom solutions, e.g., a model-predictive controller or fuzzy logic controller, with the same ROS topics and services setup. In that case, the low-level controller that replaces the MVP-Control must use the same ROS message types and implement the same interface.

MVP framework requires an external navigation solution that provides odometry (vehicle pose) and related ROS Transform Tree (TF) transforms. MVP-Control reads the odometry from the navigation node, whereas MVP-Helm reads the vehicle pose from the low-level vehicle controller. Currently, to provide a navigation solution, MVP framework uses the ROS Robot Localization package [2] to obtain the vehicle odometry. Again, benefiting from the ROS framework, the Robot Localization method could be replaced with other methods, e.g., the MaRS sensor framework [3] and works presented in [4] and [5].

The following sections provide a detailed discussion of the MVP design.

SNAME 1950 notations [6] were used to describe the motion of a marine vehicle. Lastly, the MVP packages are available in the following GitHub repositories[1].

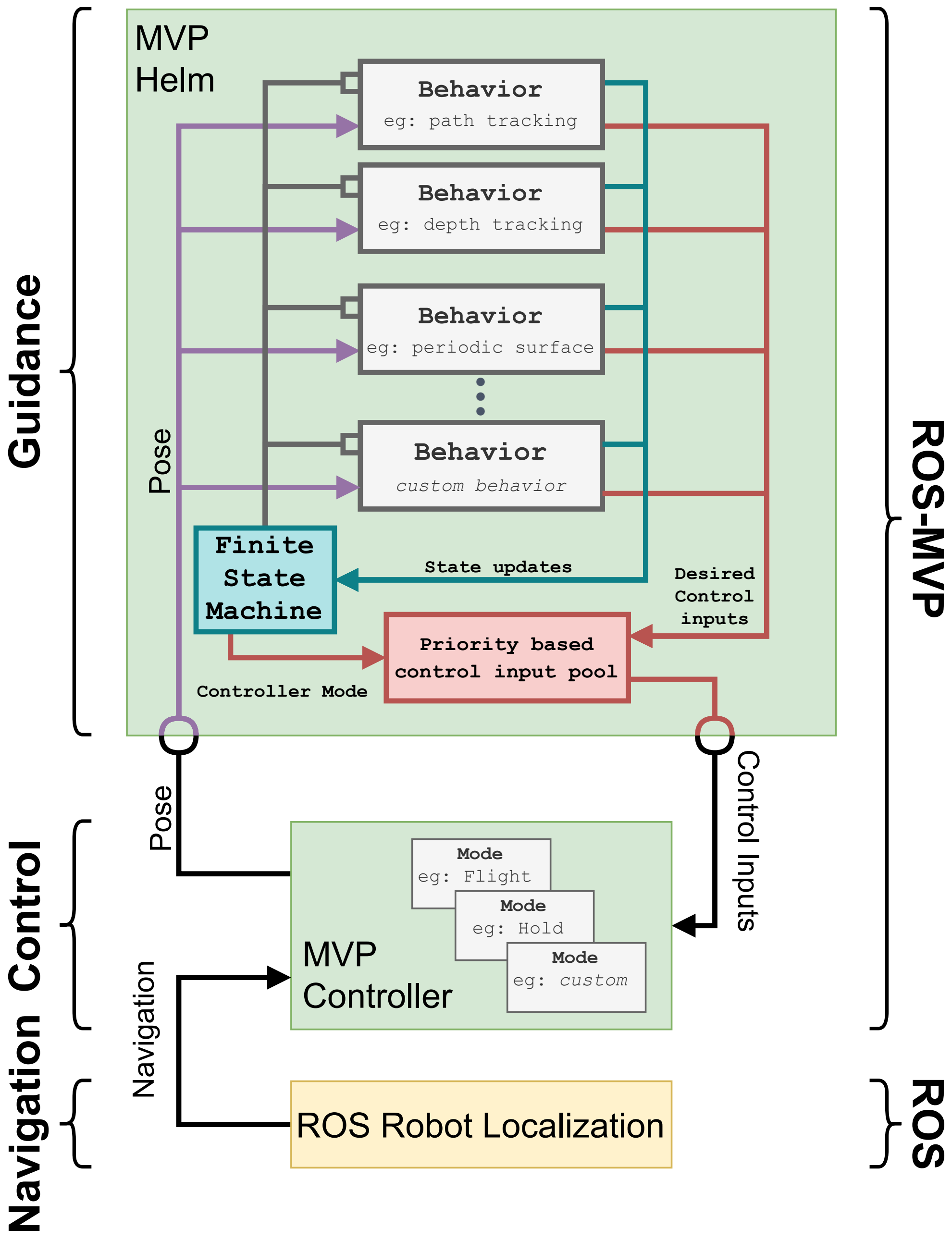


Figure 4: The system diagram of the Marine Vehicle Packages. Green blocks show ROS-MVP components, while yellow block shows the publicly available Robot Localization package from ROS repositories.

[1]MVP Packages are published under the URI Ocean Robotics GitHub page. `https://github.com/uri-ocean-robotics`

2.2 Low-level controller - MVP-Control

The MVP-Control is responsible for computing the control inputs for the actuators. MVP-Control generates actuator control commands in three steps, and the flow diagram for MVP-Control is shown in Fig.5. First, it works by computing the necessary force and torque that should act on the body frame to achieve the desired pose using a Multiple Input Multiple Output Proportional Integral Derivative (MIMO-PID) controller for each degree of freedom (DOF). Next, it feeds the target body frame force and torque to the control allocation matrix and ignores the DOFs that are not controlled in the current controller mode (configured by the user). Then, it applies the quadratic programming solver [7] to find the optimum set of forces for the thrusters to match the requested force and torque. Finally, the actual thruster command is solved based on the given thruster curves defined as polynomial functions.

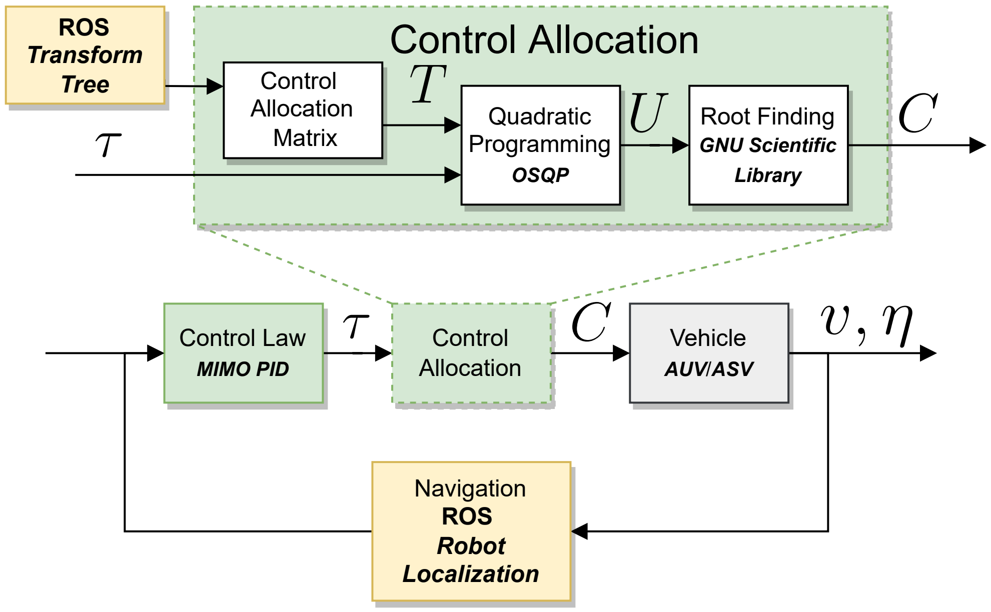


Figure 5: MVP-Control flow diagram. ROS-MVP components are shown in green and the existing ROS packages used in MVP-Control are shown in yellow.

MVP-Control is configured via a YAML[2] file where the user defines the PID gains for each control mode, thrust curve coefficients for each thruster, relevant transform tree (TF) link names, and the odometry topic source. To make the PID tuning convenient, gains can be configured dynamically on the fly using the ROS dynamic reconfigure mechanism. The propulsion systems' thrust curves are approximated using polynomials to represent the force with respect to a control command, and the data are normally available from vendors or can be obtained through thruster identification [8].

2.2.1 Control Law

The MVP-Control computes the resultant force and torque, $\boldsymbol{\tau}$, needed on the vehicle for each DOF using a MIMO-PID. The controller uses the feedback from the vehicle pose, $\boldsymbol{\eta} = [x, y, z, \phi, \theta, \psi]$ and $\boldsymbol{v} = [u, v, w, p, q, r]^\top$.

2.2.2 Control Allocation

The control allocation matrix, $\boldsymbol{T}$, maps the thrusts from individual actuators, $\boldsymbol{U}$, into the forces/torques in different DOFs either in the body frame or the earth fixed frame $\boldsymbol{\tau}$, as indicated in Eq. 1. The elements in each column in $\boldsymbol{T}$ indicate the contribution in force and torque from an actuator in different DOFs. For multiple actuators, the contributions, t_n are concatenated to form the control allocation matrix, $\boldsymbol{T} = [t_1, t_2 ..., t_n]$. In the MVP framework, the control allocation matrix can be set up manually by typing each element in the control allocation matrix or using URDF files where the program will look up the transform tree.

$$\boldsymbol{\tau} = \boldsymbol{T}\boldsymbol{U} \tag{1}$$

To find the optimal values in vector $\boldsymbol{U}$ such that the product force and torque on the vehicle, $\boldsymbol{\tau}$, we minimize the difference between the resulting values, $\boldsymbol{\tau}$, and

[2]YAML file specifications can be found on `https://yaml.org`

the required values, $\boldsymbol{\tau}^*$, from the MIMO-PID control law. Therefore, we defined the objective function (the sum of the squares of the difference), which can be expanded and simplified as shown in Eq. 2.

$$\begin{aligned} J =&(\boldsymbol{TU} - \boldsymbol{\tau}^*)^\top(\boldsymbol{TU} - \boldsymbol{\tau}^*) \\ =&\boldsymbol{U}^\top\boldsymbol{T}^\top\boldsymbol{TU} - \boldsymbol{U}^\top\boldsymbol{T}^\top\boldsymbol{\tau}^* - \boldsymbol{\tau}^{*\top}\boldsymbol{TU} + \boldsymbol{\tau}^{*\top}\boldsymbol{\tau}^* \\ =&\boldsymbol{U}^\top\boldsymbol{T}^\top\boldsymbol{TU} - 2\boldsymbol{\tau}^{*\top}\boldsymbol{TU} + \boldsymbol{\tau}^{*\top}\boldsymbol{\tau}^* \end{aligned} \tag{2}$$

The expanded objective function shares a similar format as the standard quadratic programming (QP) problem shown in Eq. 3. In the MVP-Control we applied the OSQP solver [7] to solve the QP problem defined in Eq. 4. Therefore, we obtain the thrusts for individual actuators subject to a set of defined constraints.

$$\begin{aligned} &\min_x \; J = (\frac{1}{2}\boldsymbol{x}^\top\boldsymbol{Qx} + \boldsymbol{c}^\top\boldsymbol{x}) \\ &\text{subject to } \boldsymbol{Ax} \leq \boldsymbol{b} \end{aligned} \tag{3}$$

$$\begin{aligned} &\min \underbrace{\boldsymbol{U}^\top}_{\boldsymbol{x}^\top}\underbrace{\boldsymbol{T}^\top\boldsymbol{T}}_{\frac{1}{2}\boldsymbol{Q}}\underbrace{\boldsymbol{U}}_{\boldsymbol{x}} + \underbrace{(-2\boldsymbol{\tau}^{*\top}\boldsymbol{T})}_{\boldsymbol{c}^\top}\underbrace{\boldsymbol{U}}_{\boldsymbol{x}} \\ &\text{subject to } \boldsymbol{AU} \leq \boldsymbol{b} \end{aligned} \tag{4}$$

Next, we need to obtain the control command given the thrust needed for each actuator. In MVP-Control, thrust curves are defined using polynomials where the coefficients can be estimated by applying the linear regression to the manufacturer provided thrust data at different control commands or characterized through tank tests. With the identified polynomial function, we apply the GNU Scientific Library [9] to compute the control commands, $\boldsymbol{C} = [c_1, c_2, ..., c_n]^\top$, for the required thrusts.

In the MVP-Control, the user can define different *control modes* to control different DOFs of the vehicle. For example, *flight* mode can be configured to control surge, yaw, and pitch, whereas, the *hold* mode controls X, Y, and Z position. For better performance, the user could define different PID gains for the same DOF in different modes. For instance, the PID gains for heading can be different in *flight* and *hold station* modes as the region of operation for the vehicle will be different.

2.2.3 Configuration

Information related to vehicle actuators, odometry topics, control modes, and PID gains is stored in the MVP-Control configuration file. The configuration parameters are listed in Table 1 with data type and description shown. PID parameters can be dynamically configured using the ROS Dynamic Reconfigure package.

Parameter	Type	Description
`generator_type`	`string`	Specifies the method for control allocation matrix generation.
`controller_frequency`	`number`	Controller frequency in Hertz
`control_allocation_matrix`	`complex`	Specifies the control allocation matrix. This parameter is only valid when `generator_type` is set to `user`.
`cg_link`	`string`	TF link describes the center of gravity
`tf_prefix`	`string`	the TF prefix of the robot
`world_link`	`string`	the link name for world frame
`odometry_source`	`string`	Odometry topic

`enabled`	`boolean`	Initial state of the low-level controller
`control_mode`	`complex`	Control mode definitions and PID controller gains
`thruster_ids`	`list<string>`	Arbitrary names of the actuators
`control_tf`	`complex`	TF link names of controllable actuators
`thruster_command_topics`	`complex`	Topic to publish actuator control inputs. The ROS message type is `std_msgs/Float64`
`thruster_force_topics`	`complex`	Topic to publish forces that are requested per actuator. The ROS message type is `std_msgs/Float64`
`thruster_polynomials`	`complex`	Polynomial describing the force curve of the actuator
`thruster_limits`	`complex`	Minimum and maximum forces can be requested per thruster.

Table 1: MVP-Control ROS Node parameters.

MVP-Control heavily relies on the ROS Transform Tree. The TF configuration must include an earth-fixed frame, a center-of-gravity frame, and frames for actuators. Fixed thrusters are the only supporter actuators in the current version of the MVP-Control, and for those, the X axis of the thruster frame should point in the positive force direction.

An example transforms tree with required links by MVP-Control is shown in

Fig. 6. The `odom` and `world_ned` frames are used as earth-fixed frames. The vehicle's body is represented with `base_link` frame. The transform between `odom` and `base_link` frame is provided by the navigation solution, e.g. the Robot Localization package. Lastly, thruster transforms `port_jet_link`, `starboard_jet_link`, and center of gravity transform `cg_link` are provided through URDF. Figure 7 shows a real-world example of a transform tree for ALPHA AUV [10].

$$\begin{tikzcd}
 & \text{odom} \arrow[d] \arrow[dr, bend left] & \\
 & \text{base\_link} \arrow[dl, bend right] \arrow[d] \arrow[dr, bend left] & \text{world\_ned} \\
\text{cg\_link} & \text{port\_jet\_link} & \text{starboard\_jet\_link}
\end{tikzcd}$$

Figure 6: Example transform tree with required frames by MVP-Control.

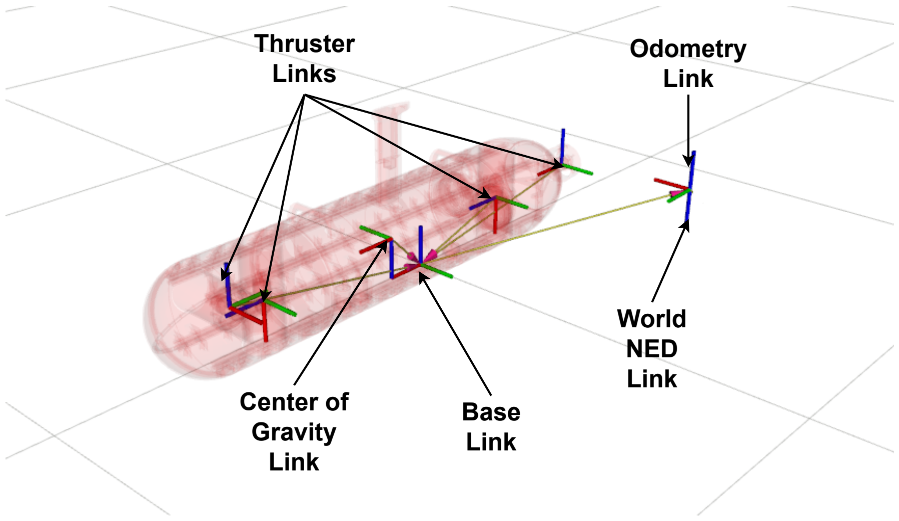


Figure 7: Transform Tree for ALPHA AUV visualized in RViZ.

During the operation, changes in the TF are permitted such as translation and rotation, and these changes in the TF tree also update the control allocation matrix. This is particularly useful when MVP-Control is used for controlling shape-reconfigurable vehicles or vehicles with movable or rotatable actuators.

Control modes are configured by `control_mode` parameter. Users can define control modes as many as needed. Each control mode contains target DOFs and PID gains for them. In Code 1, MVP-Control is configured with two control modes. The *flight* mode controls pitch, yaw, and surge, whereas the *idle* mode does not control any DOF. Having control modes without any target DOFs can be used as a safety feature. For instance, an MVP-Helm state can be configured with *idle* control mode to stop the actuators.

```
control_modes:
  flight:
    pitch: {p: 10.0, i: 3.0, d: 15.0, i_max: 20, i_min: -20}
    yaw: {p: 7.0, i: 0.5, d: 7.0, i_max: 20, i_min: -20}
    surge: {p: 10.0, i: 5.0, d: 5.0, i_max: 30, i_min: -30}
  idle: false
```

Code 1: Example control mode configuration.

MVP-Control requires several parameters about thrusters to be able to control the vehicle. These parameters inform the MVP-Control about the location of the thruster relative to the body frame, the ROS topics that the thrusters accept control inputs, the thrust-vs-command polynomial coefficients, and force constraints. A thruster is declared with `thruster_ids` parameter, and all the parameters related to the thruster are configured with its arbitrary ID as shown in Code 2.

```
thruster_ids:
  - surge
control_tf:
  surge: surge_thruster_link
thruster_command_topics:
  surge: control/thruster/surge
thruster_polynomials:
  surge: [-0.03703, 4.217, 2.84, 4.976, -0.4119, -0.8448]
thruster_limits:
  surge:
    max: 40
    min: -30
```

Code 2: Configuration example for a single actuator.

MVP-Control computes the necessary forces for each actuator and publishes them in ROS topics. These forces can be used directly by an external motor controller, or MVP-Control can solve a polynomial to generate the necessary control signal. The controller, if given polynomial coefficients of the thruster curve, will solve for the control signal given force using polynomial solver [9]. The thrust curve is configured as a polynomial, and the coefficients for the polynomial are configured as a list. The lower index in the list corresponds to the lower degree coefficient, as shown in Eq. 5.

$$\begin{gathered} f(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0 \\ \texttt{thruster\_polynomials/<thruster id>} = [a_0, a_1, ..., a_{n-1}, a_n] \end{gathered} \tag{5}$$

2.3 Mission controller - MVP Helm

The MVP Mission stack is a software suite that consists of a ROS node called MVP-Helm and behavior plugins. The MVP-Helm executes the behaviors based on a finite state machine (FSM). Each behavior is responsible for generating control inputs for the related DOFs and triggering FSM state changes. MVP Mission stack provides an abstract C++ class via the Pluginlib package in ROS middleware for behavior development, so the users can introduce new behaviors by using that library.

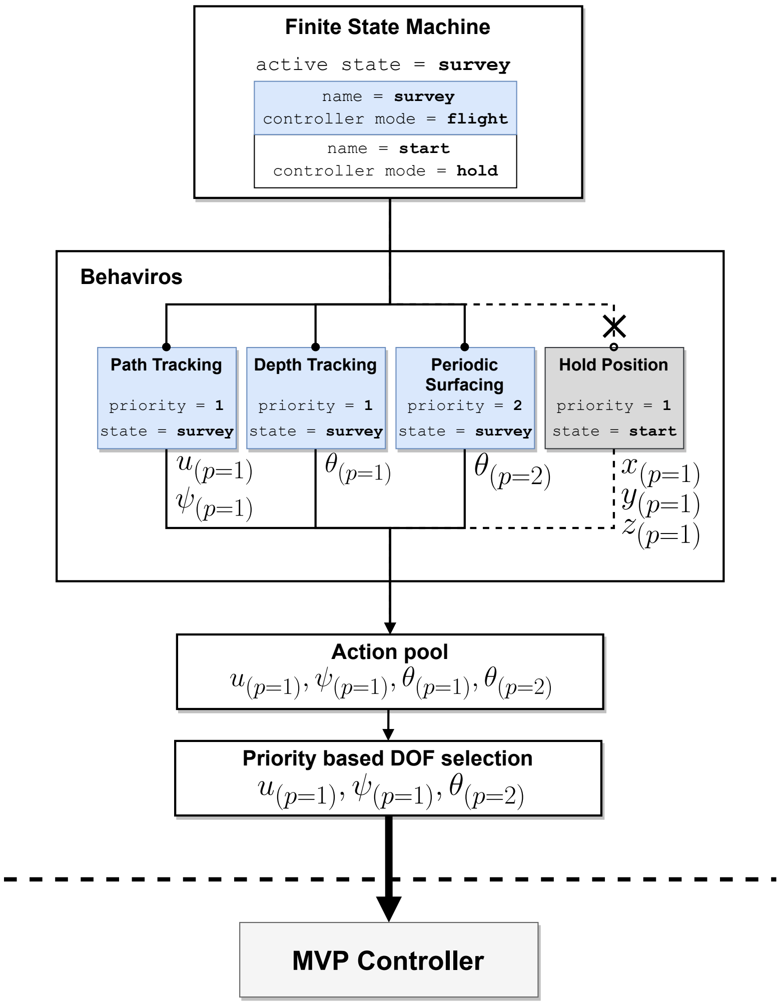


Figure 8: MVP Helm flow diagram. Priorities for desired vehicle pose requested from behaviors are shown in the letter p. Blue boxes indicate active state and executed behaviors.

Figure 8 shows an example of how the MVP-Helm node handles the control inputs from active behaviors. The MVP-Helm is configured to have two states; *survey* and *start*. The *survey* state controls the vehicle in flight mode ($M_{flight} = [u, \psi, \theta]$) while the *start* uses the controller in hold position mode ($M_{hold} = [x, y, z]$). As indicated by the green background in the FSM box, the vehicle is currently in the *survey* state where path following, depth tracking, and periodic surfacing behaviors are active. Both depth tracking and periodic surfacing behaviors control the pitch, $\boldsymbol{\theta}$. The conflicts are avoided based on the priorities defined for the behaviors in different states. For example, when the periodic surfacing behavior (with priority rank $p = 2$) is activated using a timer, the MVP-Helm will send the desired pitch angle from the periodic surfacing behavior rather than the depth tracking behavior to the MVP-Control.

2.3.1 Finite State Machine

The MVP-Helm orchestrates the behaviors using an FSM. The user defines the FSM states in the MVP-Helm configuration file. Each state contains information about vehicle controller mode and transitions to other states. State transitions are strict and if an illegal transition is requested, the MVP-Helm will fail to change its state and stay and remain in its current state. A state comprises a name, controller mode, and state transitions, as listed in Table 2.

Table 2: MVP-Helm FSM Parameter Descriptions.

Parameter	**Type**	**Description**
`name`	`string`	Name of the state.
`mode`	`string`	Controller mode for the state.
`transition`	`list<string>`	State transitions.
`initial`	`boolean`	Describes whether or not the state is the initial state.

A simple FSM with three states is given in Fig. 9. The controller mode for the given state is described with the *mode* keyword. In the example, the initial state of the FSM is the *Start* state, indicated by two circles. The configuration for the FSM shown in Fig. 9 is shown in Code 3.

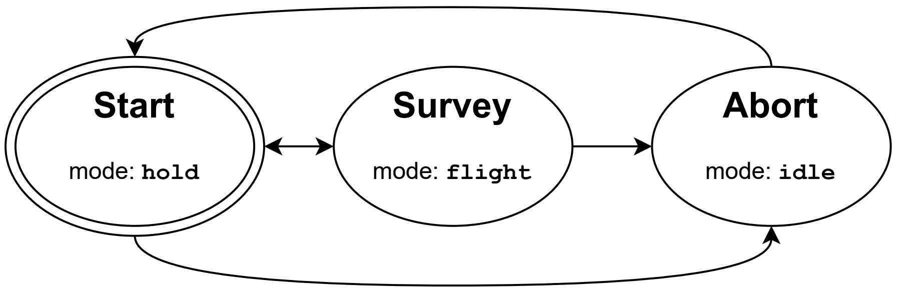


Figure 9: Example Finite State Machine for MVP-Helm.

```yaml
finite_state_machine:
  - name: start
    mode: hold
    initial: true
    transitions:
      - abort
      - survey

  - name: survey
    mode: flight
    transitions:
      - start
      - abort

  - name: abort
    mode: idle
    transitions:
      - start
```

Code 3: FMS configuration with three states, *start*, *survey*, and *abort*.

2.3.2 Behaviors

In the MVP framework, behaviors are designed to contain instructions for guidance algorithms, triggers for safety mechanisms, and executive agents for decision-making. Behaviors are designed as plugins for MVP-Helm, and they are dynamically loaded and executed in the run time. MVP-Helm informs the behaviors about the vehicle pose and low-level controller configuration and requests control inputs for their associated DOFs. After the execution, MVP-Helm collects desired control inputs for DOFs from behaviors into a priority pool and picks the DOFs with the highest priorities.

Programming Interface

Behaviors are written as plugins using PluginLib[3] library in C++, therefore all behaviors must be derived from the `BehaviorBase` class in the *behavior_interface package*. The listing 4 shows the header file for a `SimpleBehavior` class that implements only the necessary methods from `BehaviorBase` class, and these are `initialize()` and `request_set_point()` methods. First, in the `initialize()` method, a behavior is expected to declare its target DOFs, initialize ROS services and topics, and read its parameters through the ROS parameter server. Second, the `request_set_point` method, is expected to produce a set point to be handed over to the low-level vehicle controller.

[3]The PluginLib library ROS Wiki web page: `http://wiki.ros.org/pluginlib`, Accessed 2022.

```cpp
#pragma once

#include "behavior_interface/behavior_base.h"

class SimpleBehavior : public helm::BehaviorBase {
private:

    void initialize() override;

public:

    SimpleBehavior();

    bool request_set_point(
        mvp_msgs::ControlProcess *msg) override;
};
```

Code 4: The header file for a simple behavior.

In addition to the required methods, `BehaviorBase` C++ class provides two helper virtual methods that can be implemented on demand. These are `activated` and `disabled` methods. Through these methods, behaviors can detect when they are activated and disabled by the MVP-Help by implementing these functions. For instance, a path following behavior may need to append an additional track segment to the current transects after initialization or interruption. To achieve that, the behavior must know when it is activated or disabled.

The MVP-Helm reads `BehaviorBase::m_dofs` member variable to acknowledge the target DOFs of the behavior. Behaviors can trigger a state change if they are active. The behavior only needs to call `BehaviorBase::f_state_change` function in the `request_set_point` method implementation to trigger a state change. Depending on the use case, a behavior may want to control the vehicle with predefined intervals even if it is activated in the current FSM state of the MVP-Helm.

As a solution, a behavior can ask the MVP-Helm to discard its setpoint results by returning `false` in the `request_set_point` method.

The Code 5 shows implementation to the `SimpleBehavior` shown in the header file 4. Firstly, between lines 8 to 12 the `initialize` method is defined. The method sets the value for `BehaviorBase::m_dofs` to inform MVP-Helm about its target DOFs. In this instance, the behavior only controls the surge of the vehicle. Secondly, between lines 12 and 18, the `request_set_point` method is defined, in which the behavior sets desired velocity on the x axis, surge velocity, to 0. Finally, by returning `true`, the behavior instructs MVP-Helm to use the action it produced.

```
 1  #include "simple_behavior.h"
 2  #include "pluginlib/class_list_macros.h"
 3
 4  using namespace helm;
 5
 6  SimpleBehavior::SimpleBehavior() = default;
 7
 8  void SimpleBehavior::initialize() {
 9      BehaviorBase::m_dofs = decltype(m_dofs){
10          mvp_msgs::ControlMode::DOF_SURGE
11      };
12  }
13
14  bool SimpleBehavior::request_set_point(
15          mvp_msgs::ControlProcess *set_point) {
16      set_point->velocity.x = 0;
17      return true;
18  }
19
20  PLUGINLIB_EXPORT_CLASS(helm::SimpleBehavior, helm::BehaviorBase)
```

Code 5: The implementation file for a simple behavior.

Configuration

Behavior configuration parameters and MVP-Helm configuration parameters are separated with ROS namespaces. Each behavior read its own parameters from the ROS parameter server with the namespace described in MVP-Helm configuration with `name` tag. If the behavior name is *bhv00*, the private namespace for the behavior would be `helm/bhv00`. The MVP-Helm and its behaviors can be configured in a single file or separate files Behavior plugin configuration for MVP-Helm is shown in Table 3, and the template configuration is given in Code 6.

Table 3: MVP-Helm Behavior Configuration.

Parameter	**Type**	**Description**
`name`	`string`	Name of the behavior.
`plugin`	`string`	Class name of the behavior.
`states`	`list<complex>`	States and priorities of the behavior.

```
behaviors:
  - name: {{ behavior_name }}
    plugin: {{ package name }}::{{ library name }}
    states:
      - { name: {{ state name }}, priority: {{ priority }} }
```

Code 6: Configuration template for MVP Helm with placeholders.

2.4 MVP Behaviors

Currently, MVP has several behaviors implemented and are discussed in the following sections.

2.4.1 Path Following

Path following behavior is used for tracking line segments between a list of waypoints. The behavior generates surge and yaw control inputs and could be used for conducting seabed surveys where the vehicle will be programmed to follow a lawnmower pattern.

The path following behavior in MVP uses the Line of Sight (LOS) method [11]. In mobile robotics, trajectory tracking is a common control problem, and the solutions differ as the robotic applications branches out. The LOS algorithm is widely used for marine vehicles since it can compensate for side-slip and it is well-tested with marine vehicles [12].

Line of Sight Guidance

LOS algorithm works by setting the desired heading of the vessel towards a point, (x_{los}, y_{los}), placed on the path until the vessel approaches the end of the line segment as close as *acceptance radius*. The lookahead distance, Δ_h, dictates the placement of *LOS* point. The desired heading is computed with the Eq. 6 with the variables depicted in Fig. 10.

$$\Psi_d = \gamma_p + atan(\frac{-y_e}{\Delta_h}) \tag{6}$$

where;

Ψ_d	= Desired heading
γ_p	= Slope of the track
y_e	= Cross track error
Δ_h	= Look ahead distance

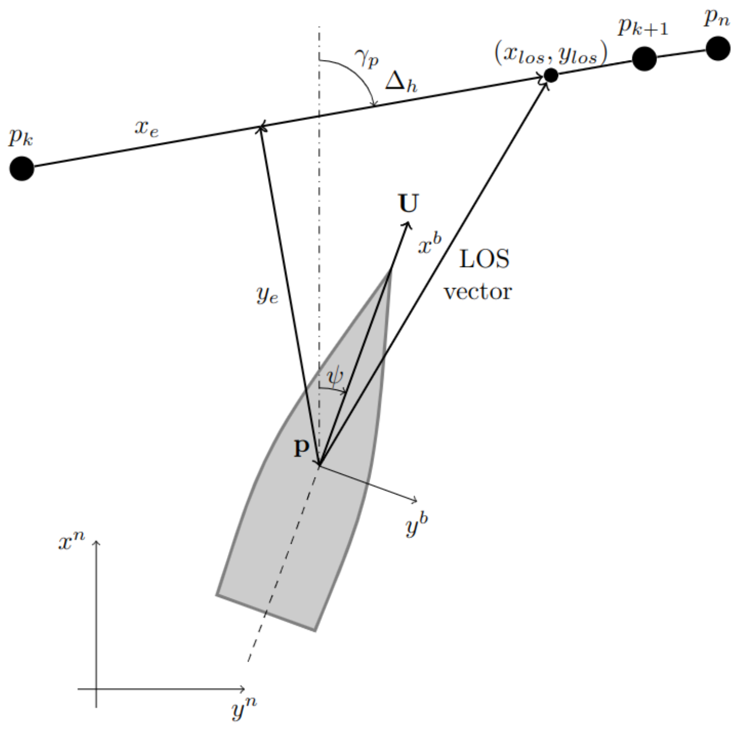


Figure 10: Line of Sight guidance algorithm variables. (Credits Lekkas, Anastasios M and Fossen, Thor I [11]).

The heading of the vessel, Ψ_d, and the course angle, X_d, may not be the same due to external disturbances or movement caused by inertia during turns. In other words, the vehicle may be *side slipping*. The relation between the heading angle and course angle can be described in the following Eq. 7.

$$X_d = \Psi_d + \beta \tag{7}$$

We can get the desired heading by taking side slip, β, into consideration as shown in Eq. 8.

$$\Psi_d = \gamma_p + atan(\frac{-y_e}{\Delta_h}) - \beta \tag{8}$$

It may be desirable to adjust the effect of the β over the heading calculation. Therefore, k_β is introduced to fine-tune the effects of the side slip correction.

$$\Psi_d = \gamma_p + atan(\frac{-y_e}{\Delta_h}) - \beta k_\beta \tag{9}$$

In some circumstances, the vehicle may fail to reach the end of the line segment within the acceptance radius. Therefore, overshoot detection is implemented by comparing along-track errors relative to the next waypoint. Calculations for along-track and cross-track errors are given in Eq. 10.

$$\begin{aligned} x_e &= (x - x_k)cos(\gamma_p) + (y - y_k)sin(\gamma_p) \\ y_e &= -(x - x_k)sin(\gamma_p) + (y - y_k)cos(\gamma_p) \end{aligned} \tag{10}$$

In addition to that, along track distance to the $(k+1)$ waypoint is defined in Equation 11.

$$x_e^k = (x - x_{k+1})cos(\gamma_p) + (y - y_{k+1})sin(\gamma_p) \tag{11}$$

We can determine the overshoot situation based on the sign of the along track error x_e and the along track distance to the target waypoint x_e^k. While the vehicle is transiting in the path segment between p_k and p_{k+1}, the along track error is $x_e > 0$ and $x_e^k < 0$. During the overshoot situation, the value of the x_e^k becomes positive. Once an overshoot is detected, the desired heading, Ψ_d, can be set to $\Psi_d + \pi$ such that the vehicle can be commanded to turn around to approach the targeted waypoint.

Implementation

Path following behavior configuration parameters is given in Table 4. This behavior is capable of continuing its task after interruption by continuing to the next line segment. Upon completing the task, it can request a state change request to MVP-Helm.

Benefiting from the ROS transform tree, users could define waypoints in different frames. This allows users to define waypoints in any frame. For example, if the vehicle has a global localization system that publishes an earth-fixed frame to the transform tree, the user can use that frame to program a path following the mission with earth-fixed coordinates. This is particularly useful when `navsat_transform_node` from ROS Robot Localization [2] package is being used for publishing earth fixed UTM frame.

Path following behavior listens for waypoint updates from the topics described with `update_topic` and `append_topic`. To receive waypoint updates, the path following behavior doesn't need to be activated.

A fail-safe method for overshoot situations is built using a timer. If the timer reaches an end, behavior requests a state change defined with `state_fail` parameter from MVP-Helm. *Overshoot* state is deactivated if the vehicle successfully enters the acceptance area or goes back to the track, in other words, if $x_e^k < 0$.

Table 4: Path Following behavior parameters

Parameter	**Type**	**Description**
`update_topic`	`string`	Topic name that behavior listens for waypoint updates.
`append_topic`	`string`	Topic name that behavior listens for additional waypoints.
`frame_id`	`string`	Frame id of the waypoints described with `waypoints` parameter.
`acceptance_radius`	`number`	Acceptance radius in meters.
`lookahead_distance`	`number`	Lookahead distance, Δ_h, in meters.
`overshoot_timeout`	`number`	Overshoot timeout in seconds.
`surge_velocity`	`number`	Surge velocity output in m/s.
`beta_gain`	`number`	Beta gain.
`state_done`	`string`	State transition after successful run.
`state_fail`	`string`	State transition after failed run.
`waypoints`	`list<comples>`	List of initial waypoints.

Example configuration for path following behavior is given in Code 7. In the example configuration, the path following behavior follows the $[(0,0),(0,10),(10,10),(10,0)]$ coordinates in the `odom` frame. The desired surge velocity for this behavior is 0.70 m/s and the acceptance radius for completing a transect after reaching the end of the line is 3.0 meters. Side slip gain, k_β, is set to 1.0.

```
waypoints:
  - {x: 0, y: 0}
  - {x: 0, y: 10}
  - {x: 10, y: 10}
  - {x: 10, y: 0}
acceptance_radius: 3.0 # In meters
frame_id: odom
surge_velocity: 0.70 # In m/s
lookahead_distance: 3.0 # In meters
beta_gain: 1.0 # Arbitrary gain
```

Code 7: Example configuration of Path Following Behavior.

2.4.2 Waypoint Tracking

This behavior is used for moving the vehicle from one waypoint to another without the requirement of staying on a straight line between two waypoints. It generates surge and yaw control inputs for the low-level vehicle controller.

The parameters for this behavior are similar to those of path following behavior except few differences. Since this behavior doesn't try to maintain a course, lookahead distance, Δ_h, beta gain, k_β, and overshoot timeout parameters are not necessary.

2.4.3 Depth Tracking

This behavior helps the underwater vehicle to maintain a certain depth. The parameters for the depth tracking behavior are given in Table 5. Depth tracking behavior controls the vehicle depth by generating control inputs for depth and pitch angle, z and θ respectively. On one hand, the behavior generates depth control input, z, directly from the desired depth, z^*, parameter. On the other hand, it computes the desired pitch angle, θ^*, by comparing desired and the current depth. This behavior can control the depth for single propeller-driven torpedo-shaped AUVs, and hover-capable AUVs and ROVs.

Equation 12 shows how the control inputs U_z and U_θ are computed. Desired depth is denoted with z^* and forward-looking distance in the vertical plane is denoted with Δ_v.

$$\begin{aligned} U_z &= z^* \\ U_\theta &= atan(\frac{z - z^*}{\Delta_v}) \end{aligned} \tag{12}$$

If the parameter `use_heave_velocity` is set to `true`, depth tracking behavior will take heave velocity into account while trying to achieve a certain depth as shown in Equation 13.

$$\begin{aligned} U_z &= z^* \\ U_\theta &= atan(\frac{z - z^*}{\Delta_v}) + atan(\frac{w}{u}) \end{aligned} \tag{13}$$

Table 5: Depth Tracing Behavior parameters.

Parameter	Type	Description
`desided_depth`	`number`	Desired depth in meters, z^*.
`forward_distance`	`number`	Forward-looking distance on vertical plane Δ_v.
`max_pitch`	`number`	Absolute pitch angle limit in radians.
`use_heave_velocity`	`boolean`	Incorporate heave velocity when computing required pitch

2.4.4 Periodic Surfacing

This behavior allows underwater vehicles to periodically surface at a user defined interval. Parameters for periodic surfacing behavior are given in Table 6. It uses a similar method as depth tracking behavior. The periodic surfacing behavior generates depth and pitch control inputs for the low-level vehicle controller.

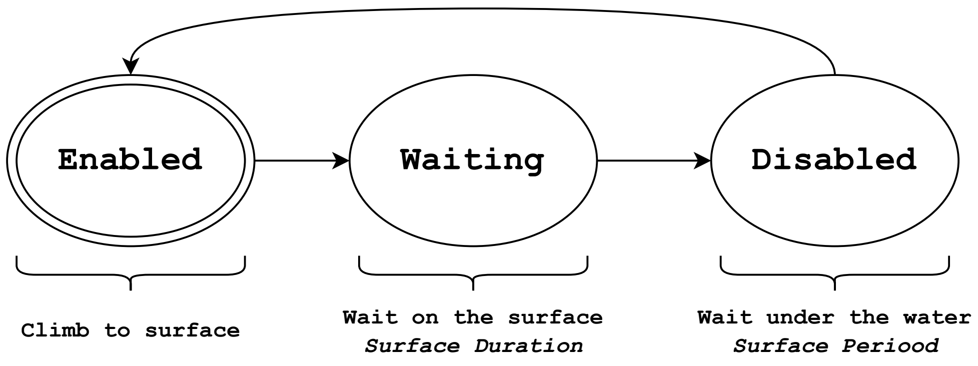


Figure 11: Periodic surfacing behavior internal stages.

The behavior will guide the vehicle to climb up to the surface. The behavior starts in the *Enabled* state and starts climbing to the surface. The *Waiting* stage will be active when the vehicle is at the surface. The behavior will remain in this stage until the time at the surface has exceeded a certain duration defined by the *surface_duration* parameter. Then the behavior enters the *disabled* stage and waits until the duration defined by `surface_period` is completed.

Table 6: Periodic surfacing parameters.

Parameter	Type	Description
`forward_distance`	`number`	Forward-looking distance on vertical plane Δ_v.
`max_pitch`	`number`	Absolute pitch angle limit in radians.
`surface_period`	`number`	Incorporate heave velocity when computing required pitch
`surface_duration`	`number`	Incorporate heave velocity when computing required pitch

2.4.5 Timer Behavior

The timer behavior enables users to implement timer-activated MVP-Helm state changes. Parameters for timer behavior are shown in Table 7. When this behavior is activated in a state, it starts the timer and waits for user defined duration. After the elapsed time has exceeded the defined duration, the timer behavior will request a state change to the FSM.

Table 7: Timer behavior parameters.

Parameter	Type	Description
`duration`	`number`	Duration in seconds.
`transition_to`	`string`	State transition request.

2.5 Robot Localization Integration

ROS Robot Localization [2] is a generic sensor fusion program developed by Charles River Analytics Inc. It contains implementations for the Extended Kalman Filter and Unscented Kalman Filter, and it reads and writes directly to the ROS transform tree.

Sensor	**Output data**
DVL	u, v, w
Pressure Sensor	z
GPS	latitude and longitude
AHRS	$\psi, \theta, \phi, \dot{\psi}, \dot{\theta}, \dot{\phi}, \dot{u}, \dot{v}, \dot{w}$

Table 8: Common navigation sensors and their outputs.

Most marine vehicles are equipped with navigation sensors such as Doppler Velocity Logger (DVL), Attitude and heading reference system (AHRS), Global Positioning System (GPS), and so on. DVL is used for measuring the velocity, AHRS is used for measuring the vehicle attitude, and GPS is used for acquiring the global position of the vehicle. For underwater vehicles, the pressure sensor is used for measuring the depth of the vehicle. Table 8 shows the common navigation sensors and their data outputs.

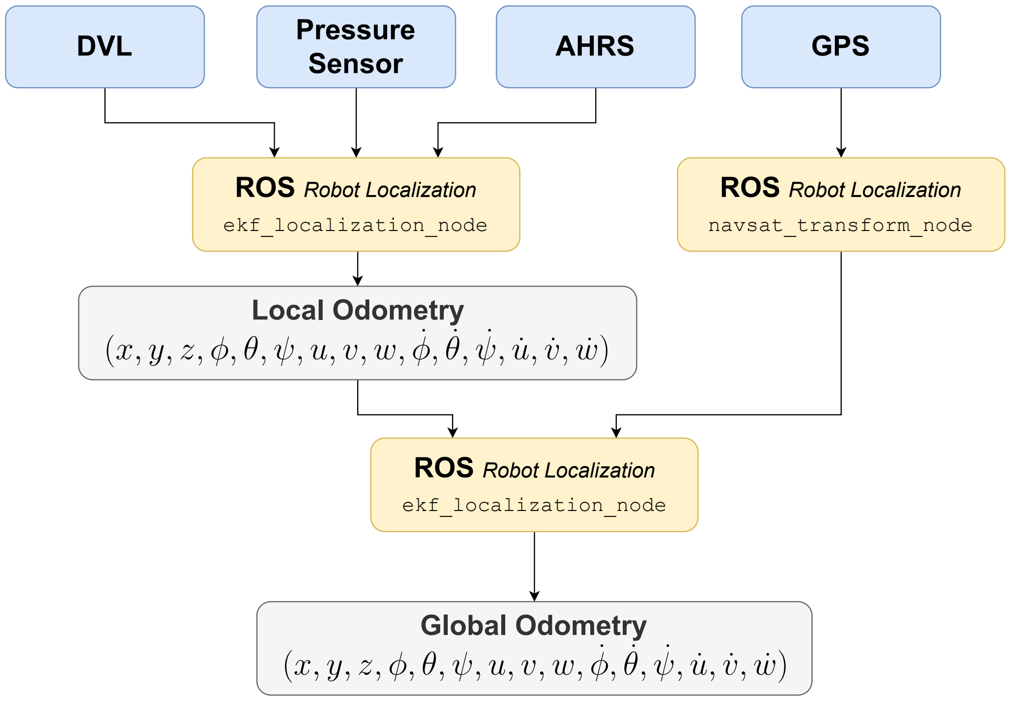

DVL
Pressure Sensor
AHRS
GPS
ROS Robot Localization
ekf_localization_node
ROS Robot Localization
navsat_transform_node
Local Odometry
$(x, y, z, \phi, \theta, \psi, u, v, w, \dot{\phi}, \dot{\theta}, \dot{\psi}, \dot{u}, \dot{v}, \dot{w})$
ROS Robot Localization
ekf_localization_node
Global Odometry
$(x, y, z, \phi, \theta, \psi, u, v, w, \dot{\phi}, \dot{\theta}, \dot{\psi}, \dot{u}, \dot{v}, \dot{w})$

Figure 12: Robot Localization data flow diagram. Existing ROS packages are shown in yellow. Data sources (essentially, sensor ROS drivers) for navigation are shown in blue. The resulting odometry information is shown in grey. They can be used for tracking waypoints in different coordinates.

In a normal operation, the data flow of the robot localization node is similar to the graph shown in Fig. 12. The robot localization node, shown in red, fuses the sensor readings from DVL, pressure sensor, and AHRS to produce odometry. This robot localization node produces odometry information relative to the wake-up position. The odometry message contains pose and velocity information of the base_link frame (attached to the vehicle) relative to the odometry frame (attached to its wake-up position). It publishes the transform between the odometry frame and the base_link frame. The `navsat_transform_node` publishes the transform between an earth-fixed frame and the odometry frame and publishes an odometry message that only contains x, y, z position. The second robot localization node, shown in yellow, fuses the continuous odometry data from the previous robot localization node with GPS odometry data from `navsat_transform_node`. The transform tree setup is often similar to that shown in Fig. 13. More information about GPS integration can be found on the robot_localization documentation page.

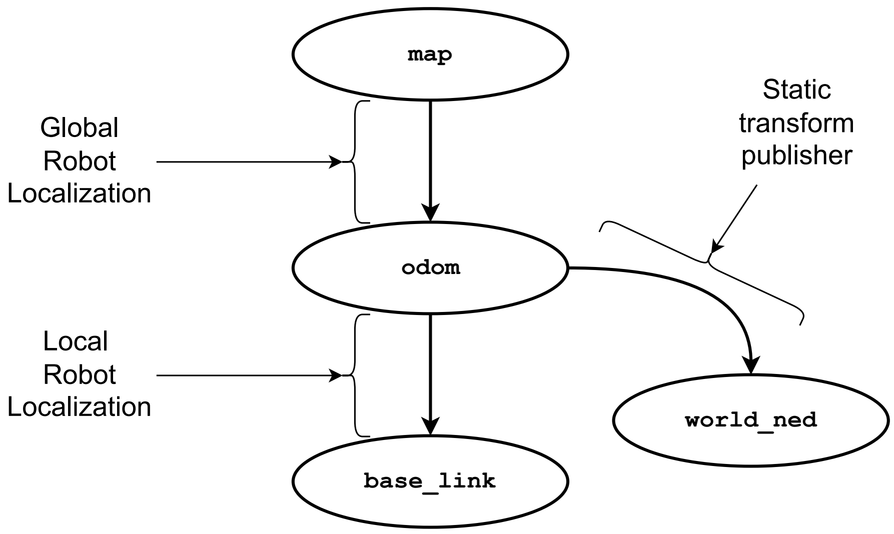


Figure 13: Robot Localization common TF tree.

Robot localization works explicitly in East North Up (ENU) coordinate system. However, it is desirable to work in North East Down (NED) coordinate system when working with marine vehicles. To achieve that a new frame called *world_ned* s published to the TF tree, as shown in Code 8.

```
<launch>
  <node
    name="world2ned"
    pkg="tf2_ros"
    type="static_transform_publisher"
    args="0.0 0.0 0.0
          1.570796327 0.0 3.141592653589793
          world world_ned"
  />
</launch>
```

Code 8: ENU to NED conversion.

List of References

[1] T. I. Fossen and T. A. Johansen, “A survey of control allocation methods for ships and underwater vehicles,” in *2006 14th Mediterranean Conference on Control and Automation*, 2006, pp. 1–6.

[2] T. Moore and D. Stouch, “A generalized extended kalman filter implementation for the robot operating system,” in *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer, July 2014.

[3] A. I. Mourikis and S. I. Roumeliotis, “A multi-state constraint kalman filter for vision-aided inertial navigation,” in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 3565–3572.

[4] C. Brommer, R. Jung, J. Steinbrener, and S. Weiss, “MaRS : A Modular and Robust Sensor-Fusion Framework,” 2020.

[5] L. Zhao, M. Zhou, and B. Loose, “Towards under-ice sensing using a portable rov,” in *OCEANS 2022: Hampton Roads*. IEEE, 2022.

[6] T. I. Fossen, “Guidance and control of ocean vehicles,” *University of Trondheim, Norway, Printed by John Wiley & Sons, Chichester, England, ISBN: 0 471 94113 1, Doctors Thesis*, 1999.

[7] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, “OSQP: an operator splitting solver for quadratic programs,” *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020. [Online]. Available: https://doi.org/10.1007/s12532-020-00179-2

[8] R. Bachmayer, L. Whitcomb, and M. Grosenbaugh, “An accurate four-quadrant nonlinear dynamical model for marine thrusters: theory and experimental validation,” *IEEE Journal of Oceanic Engineering*, vol. 25, no. 1, pp. 146–159, 2000.

[9] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth, F. Rossi, and R. Ulerich, *GNU scientific library*. Network Theory Limited, 2002.

[10] M. Zhou, E. C. Gezer, W. McConnell, and C. Yuan, “Acrobatic low-cost portable hybrid auv (alpha): System design and preliminary results,” in *OCEANS 2022: Hampton Roads*. IEEE, 2022.

[11] A. M. Lekkas and T. I. Fossen, “Line-of-sight guidance for path following of marine vehicles,” *Advanced in marine robotics*, pp. 63–92, 2013.

[12] N. Gu, D. Wang, Z. Peng, J. Wang, and Q.-L. Han, “Advances in line-of-sight guidance for path following of autonomous marine vehicles: An overview,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–17, 2022.

CHAPTER 3

Simulation Validation

Simulation environments are helpful as it reduces field trial times and costs. Depending on the requirement, the simulation environments can be simple as dynamic models [1] or very sophisticated 3-dimensional environments made with physics engines. Dynamic model simulations can be derived from nonlinear motion equations of marine vehicles [2], then customized for the target platform with matching dynamic coefficients. Physics engine-based simulations, on the other hand, aim to simulate an entire virtual world including the target robotic platform without needing nonlinear motion equations of the platform. Consequently, finding the dynamic coefficients of the target platform is not necessary. Thus, they are often generic and applicable to a variety of use cases.

Over the past decade, physics engines became extremely popular. Gazebo [3] is one popular simulator that uses a physics engine, and it goes hand in hand with ROS, the most popular robotics development middleware. It offers a great sandbox environment for robotics software development, and it has been the basis of the very popular underwater vehicle simulator, UUV Simulator [4], and Project Dave[5]. Meanwhile, there are also other emerging simulation environments. For instance, Stonefish [6] is a new simulator built based on the bulletin physic [7] and used for simulating underwater vehicles [8]. Instead of defining the hydrodynamic parameters manually by the users, Stonefish could estimate the hydrodynamic terms, e.g., added mass and hydrodynamic damping coefficient automatically based on the imported vehicle shapes. Moreover, Stonefish could simulate different water turbidity conditions which could help validate underwater computer vision algorithms.

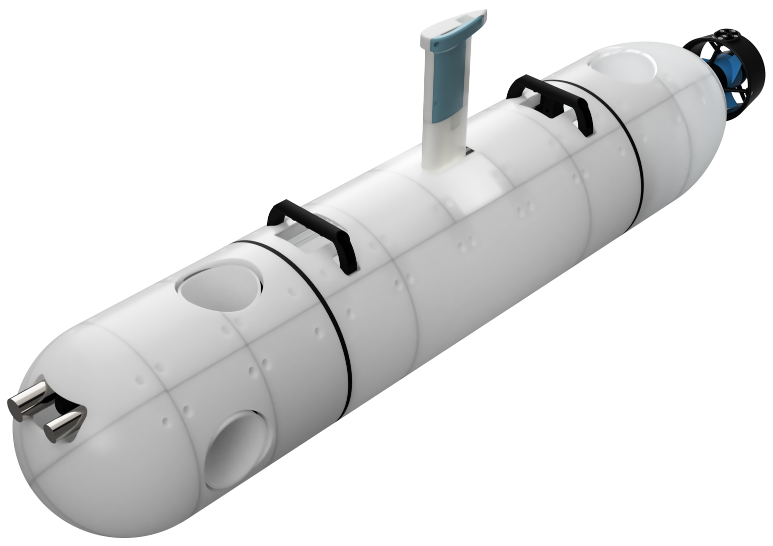


Figure 14: Newly designed ALPHA AUV from Smart Ocean Systems Lab at University of Rhode Island, Graduate School of Oceanography.

In this work, we have integrated the MVP framework with the Stonefish to simulate the operation of a newly developed AUV (shown in Fig. 14) [9].

When running the simulation environment, we mainly use two user interfaces. Figure 15 shows the Stonefish simulator user interface (on the left) and RViZ visualization tool (on the right) side by side. RViZ depicts the environment from the perspective of the simulated robot in one window, while the Stonefish user interface shows the environment from the human perspective in another window.

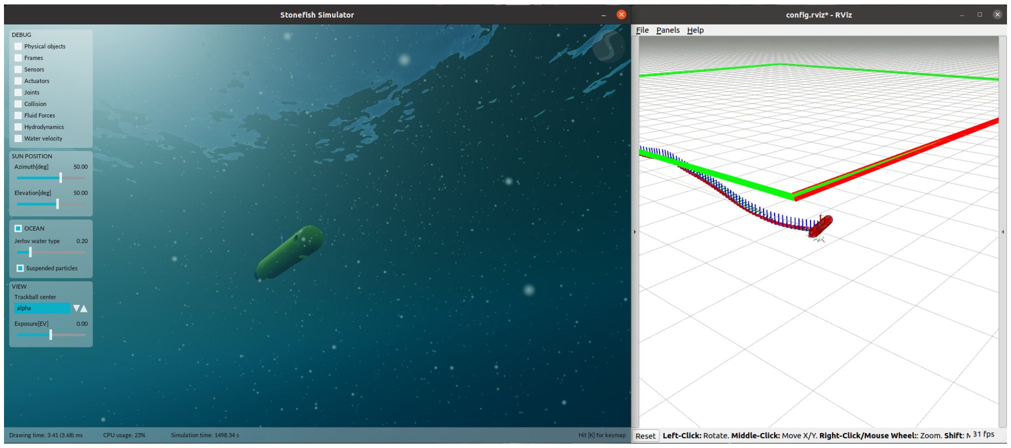


Figure 15: Stonefish simulation UI (on the left) and RViZ visualization tool (on the right) with ALPHA AUV.

3.1 Stonefish Simulation Setup

Stonefish [6] is a relatively new robotics simulator, published in 2019, developed using the Bullet physics library. It has exceptional support for underwater environment simulation compared to other open-source robotics simulators, because of its underwater environment visualization, added mass and hydrodynamic damping coefficient estimation, and current simulating capabilities. It is not tied to any robotics middleware. Instead, it is packaged as a standalone library. Thus, users could develop their own interface with the stonefish simulator. So far Stonefish has been used in several projects successfully by several research groups in Sweeden [10] and Spain [11].

3.1.1 Integration

We have modified the existing Stonefish wrapper for the ROS to work with the MVP framework. The Stonefish MVP and Stonefish ROS packages are similar, but the Stonefish MVP uses MVP ROS Messages and it uses a slightly different `scenario` parser that allows defining custom topic names.

3.1.2 Configuration

Stonefish MVP is configured through slightly customized XML scenario files. The *scenario* files contain details for the vehicle and the environment. The vehicle configuration resembles the Universal Robot Description Format (URDF)[1]. One significant difference that should be noted is that URDF uses East North Up whereas the Stonefish simulator uses North East Down (NED). Because of this, the same coordinates can not be used in both URDF and Stonefish scenario files. Thus, two separate files were created that contain the same information in two different formats and different coordinate frames, the URDF robot description, and the Stonefish scenario file.

The integration of the Robot Localization [12] was also tested with the measurements from the simulated sensors (including DVL, pressure sensor, AHRS, and GPS) from the Stonefish simulator For the physical platform, the same set of sensory information is sent to the same navigation stack as the simulated vehicle.

Different parts of the AUV are separated into different 3D model files. Two sets of 3D model files were generated. The first group was used for visualization, the second was used for physics calculations and estimations such as collisions, drag coefficients, etc. Simpler 3D models are used in order to lower the number of meshes stored in the memory.

3.2 Results

We first tested the MVP-Control in the simulation environment. During the test runs, the implemented PID control law was validated and the controller's ability to control multiple degrees of freedom is validated. The MVP mission architecture was tested following the validation of the low-level vehicle controller capability.

[1]The Universal Robot Description Format (URDF) is used for describing robot links and joints. More information can be found here: `http://wiki.ros.org/urdf`.

During the heading test, the controller is commanded to turn the vehicle to the heading of -60° and 60° with a 60 seconds period and the result is shown in Figures 16 and 17.

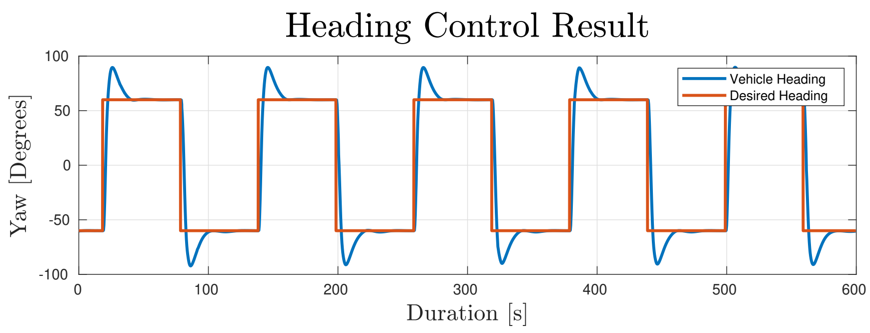


Figure 16: Heading control results from Stonefish simulator. Control inputs are shown in red, and control outputs are shown in blue.

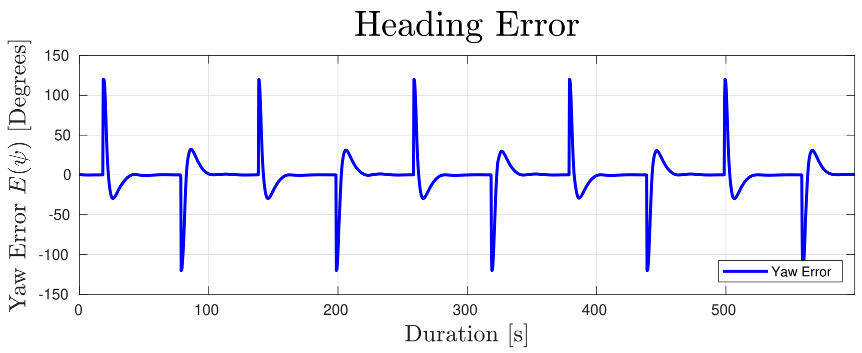


Figure 17: Heading control error.

Next, we tested the MVP-Control for surge velocity, and the result is presented in Figures 18 and 19. During this test, the vehicle is commanded to speed up to 0.5 m/s for 60 seconds and come to a complete stop for another 60 seconds.

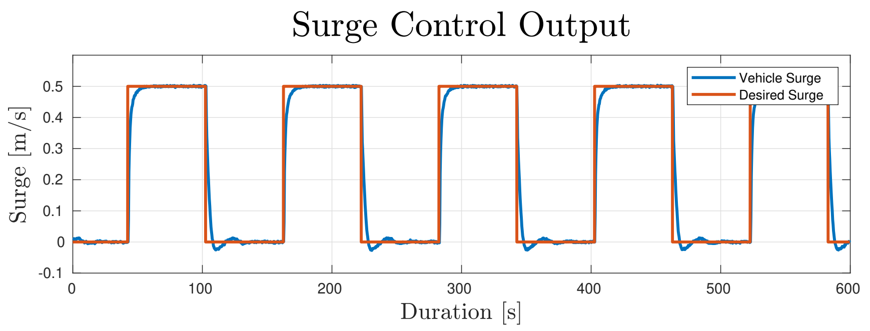


Figure 18: Surge control results from Stonefish simulator. Control inputs are shown in red, and control outputs are shown in blue.

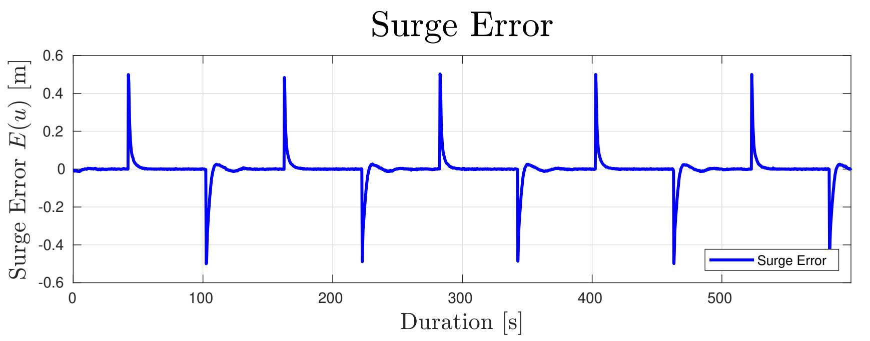


Figure 19: Surge control error in the simulation.

After we have tested the control for individual DOFs, we evaluated the MVP-Control performance for multiple DOFs. Figures 21 and 20 show the test result where the MVP-Control controls the vehicle's x, y, z position and the pitch angle. Again, we have different desired values changing back and forth. During this test requested pitch was changing between 0 and 15.0 degrees every 2 minute. The requested pose was also changing from $(x = 0, y = 0, z = 4)$ to $(x = 5.0, y = 5.0, z = 6)$.

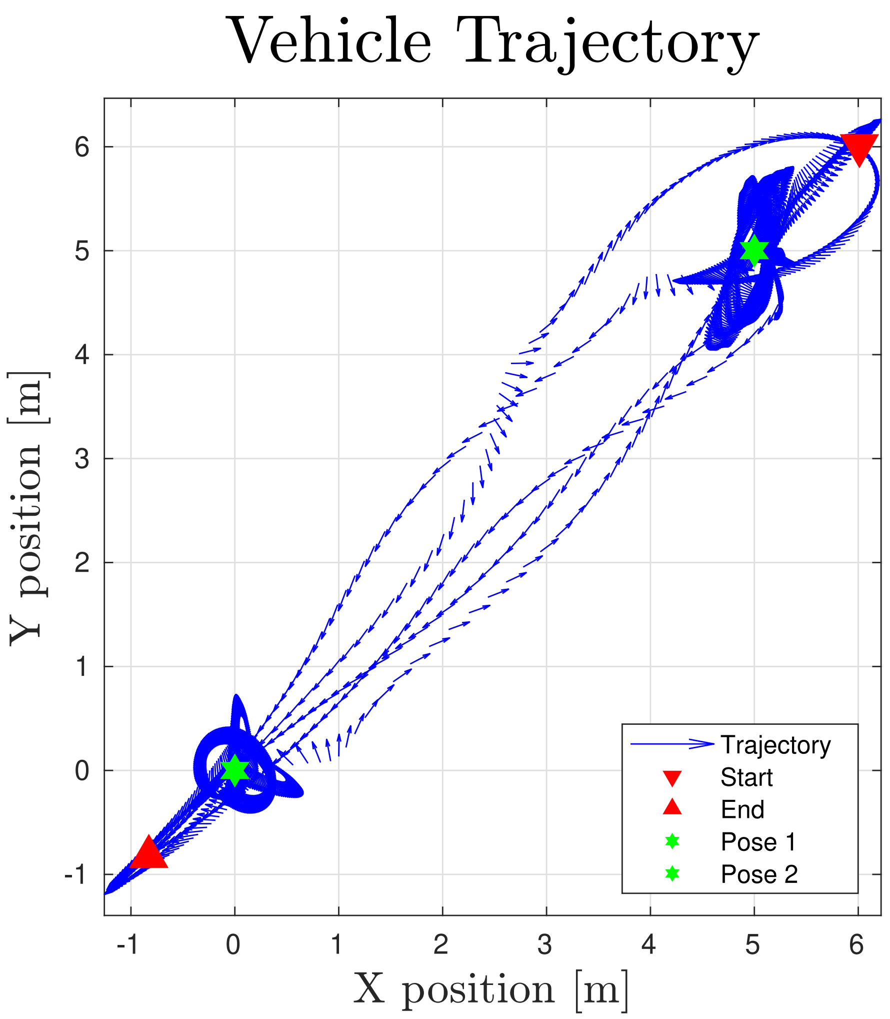
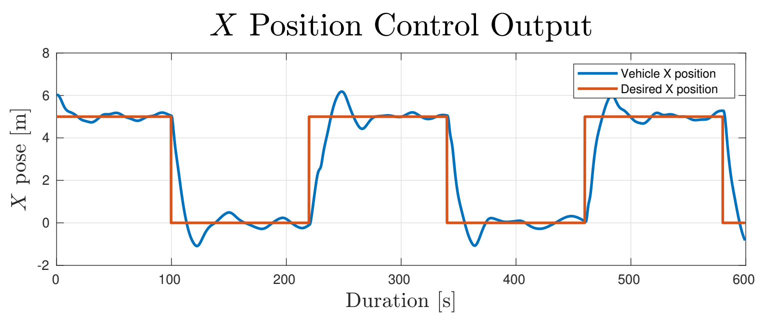
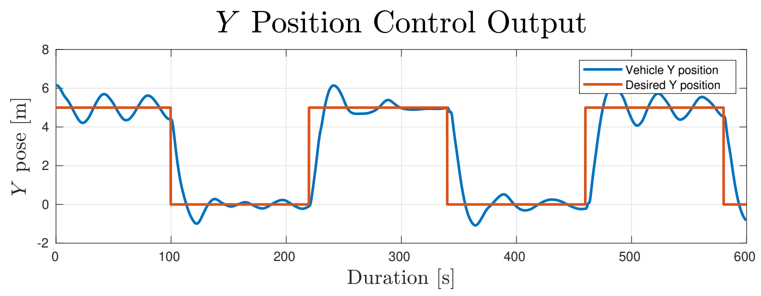


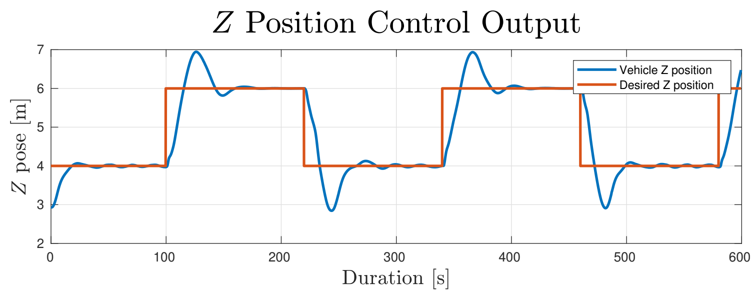
Figure 20: Vehicle trajectory during the multi-DOF control tests. During this test vehicle position, x, y, z, are the controlled states.



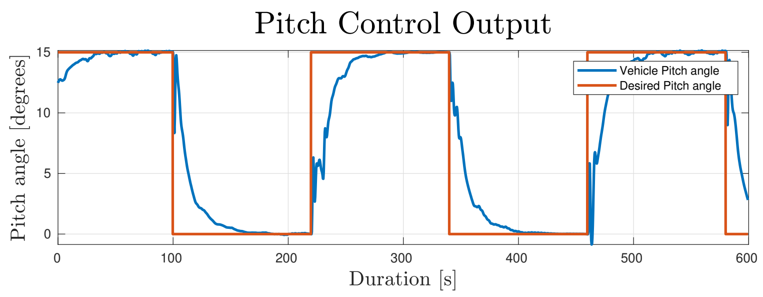
(a) Station keeping performance on X axis.



(b) Station keeping performance on Y axis.



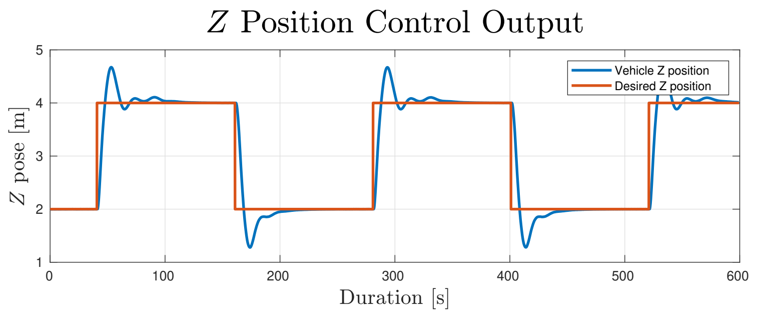
(c) Station keeping performance on Z axis.



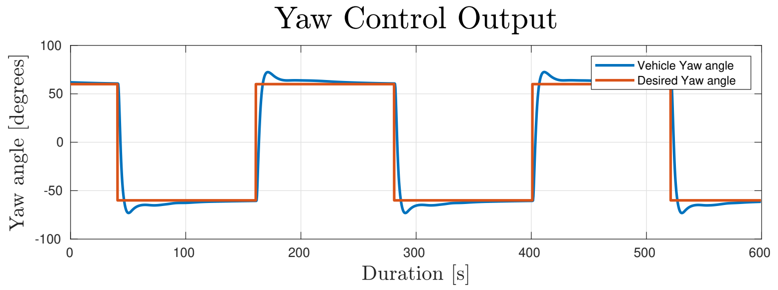
(d) Station keeping performance on pitch.

Figure 21: MVP-Controller results in Stonefish simulator for controlling X, Y, Z position and pitch simultaneously.

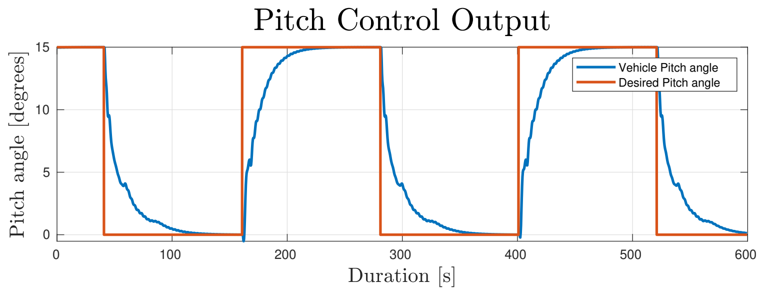
Figure 22 show the test result where the MVP-Control controls the vehicle's depth, pitch, and yaw angle. During this test, the desired pitch was changing between 0 and 15.0 degrees, yaw between −60 and 60, and depth between 2m and 4m on every 2 minute.



(a) Station keeping performance on *Z* axis



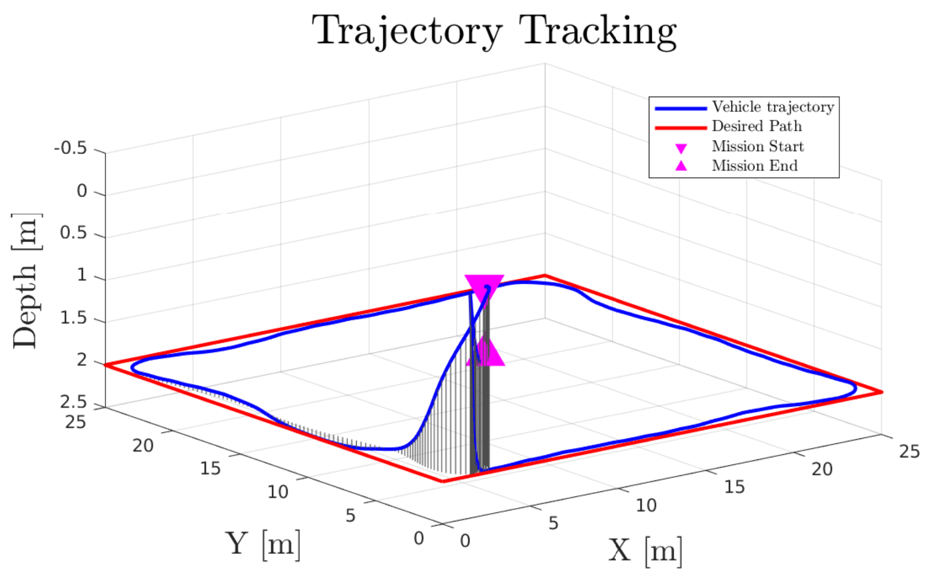
(b) Station keeping performance on Yaw rotation.



(c) Station keeping performance on Pitch rotation.

Figure 22: MVP-Controller results in Stonefish simulator for controlling Z position, yaw, and pitch simultaneously.

After we have tuned the PID gains inside the MVP-Control, we evaluated the MVP-Helm behaviors. In the result shown in Figure 23, the mission has path following and depth tracking enabled in the survey state. In Figure 23, the desired path and the depth are shown with red lines, and the actual vehicle path and trajectory are shown in blue lines. During the simulation validation, environmental disturbances such as water currents were not introduced. Therefore cross track errors were small; the root-mean-square error was 0.740016m (see Fig. 24). The segment change results in cross-track error peaks up to 3 m, as the path-following behavior changes to a new segment as soon as the vehicle approaches the end of the segment closer than the acceptance radius.



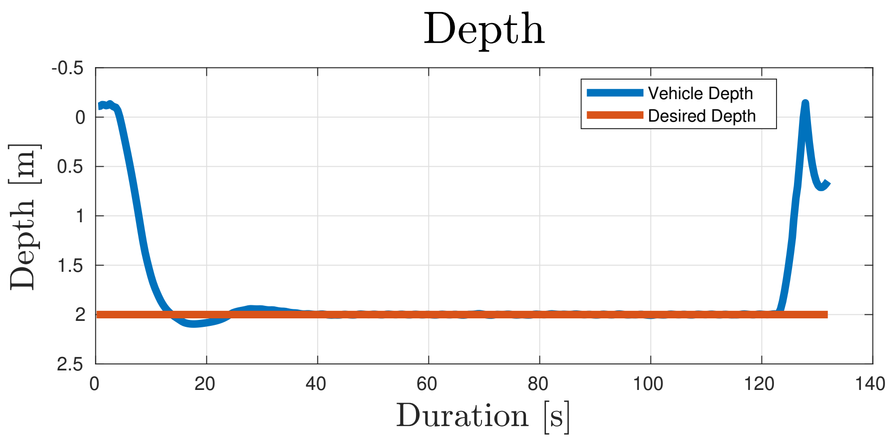


Figure 23: Trajectory Tracking mission execution in the Stonefish simulation. The graph at the top shows the 3D trajectory of the vehicle during the test. The blue color depicts the actual vehicle trajectory, while the red color shows the desired path. The graph at the bottom shows the depth tracking performance. The red line depicts the desired depth, whereas the blue shows the actual vehicle depth.

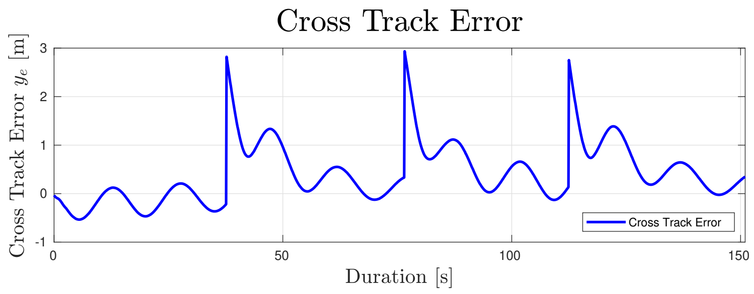


Figure 24: Cross track error for path following mission in Stonefish simulator. The cross-track error peaks indicate the line segment change in the desired path.

List of References

[1] C.-W. Chen, J.-S. Kouh, and J.-F. Tsai, “Modeling and simulation of an auv simulator with guidance system,” *IEEE Journal of Oceanic Engineering*, vol. 38, no. 2, pp. 211–225, 2013.

[2] T. I. Fossen and O.-E. Fjellstad, “Nonlinear modelling of marine vehicles in 6 degrees of freedom,” *Mathematical Modelling of Systems*, vol. 1, no. 1, pp. 17–27, 1995.

[3] N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.

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

[5] M. M. Zhang, W.-S. Choi, J. Herman, D. Davis, C. Vogt, M. McCarrin, Y. Vijay, D. Dutia, W. Lew, S. Peters, and B. Bingham, “Dave aquatic virtual environment: Toward a general underwater robotics simulator,” in *2022 IEEE/OES Autonomous Underwater Vehicle (AUV) Symposium*, 2022, pp. 1–8.

[6] P. Cieślak, “Stonefish: An advanced open-source simulation tool designed for marine robotics, with a ros interface,” in *OCEANS 2019 - Marseille*, 2019, pp. 1–6.

[7] E. Coumans, “Bullet physics simulation,” in *ACM SIGGRAPH 2015 Courses*, 2015, p. 1.

[8] R. Pi, P. Cieślak, P. Ridao, and P. J. Sanz, "Twinbot: Autonomous underwater cooperative transportation," *IEEE Access*, vol. 9, pp. 37 668–37 684, 2021.

[9] M. Zhou, E. C. Gezer, W. McConnell, and C. Yuan, "Acrobatic low-cost portable hybrid auv (alpha): System design and preliminary results," in *OCEANS 2022: Hampton Roads*. IEEE, 2022.

[10] S. Bhat, I. Torroba, Ö. Özkahraman, N. Bore, C. I. Sprague, Y. Xie, I. Stenius, J. Severholt, C. Ljung, J. Folkesson, *et al.*, "A cyber-physical system for hydrobatic auvs: system integration and field demonstration," in *2020 IEEE/OES Autonomous Underwater Vehicles Symposium (AUV)*. IEEE, 2020, pp. 1–8.

[11] J. Esteba, P. Cieślak, N. Palomeras, and P. Ridao, "Docking of non-holonomic auvs in presence of ocean currents: A comparative survey," *IEEE Access*, vol. 9, pp. 86 607–86 631, 2021.

[12] T. Moore and D. Stouch, "A generalized extended kalman filter implementation for the robot operating system," in *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer, July 2014.

CHAPTER 4

Field Test

Field trials were conducted with an AUV to validate the MVP framework. In this chapter, the electronic and software design of ALPHA AUV and the results from field tests are presented.

4.1 Overall System Architecture

ALPHA (Acrobatic Low-cost Portable Hybrid AUV), shown in Figure 25, is a highly maneuverable underwater vehicle suitable for a variety of use cases such as coastal water sampling, seabed mapping, and iceberg mapping.

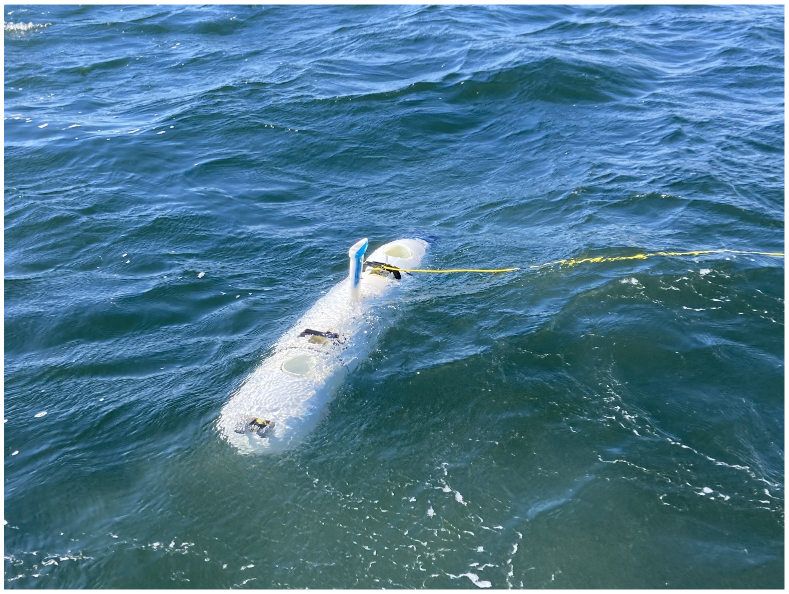


Figure 25: ALPHA AUV Revision 2 in Narragansett Bay

ALPHA is controlled using thrusters. The communication with the vehicle is established via a long-range RF (radio frequency) serial link, short-range WiFi link, and Ethernet through a single twisted pair tether. Onboard navigation sensors are GPS (Global Positioning System - U-Blox 7), DVL (Doppler Velocity Logger - *Wa-*

terlinked DVL A50), pressure sensor (*BlueRobotics Bar30*), and AHRS (Attitude and Heading Reference System - *Xsens MTI630*).

Throughout the development process, ALPHA went over two main revisions. In the first revision, ALPHA had 3 thrusters in total as shown in Figure 26. In order to increase vertical stability a second tunnel thruster was added on the rear side, and the wireless communication mast was moved to the middle of the vehicle. The system diagram for ALPHA Revision 2 is shown in Figure 27.

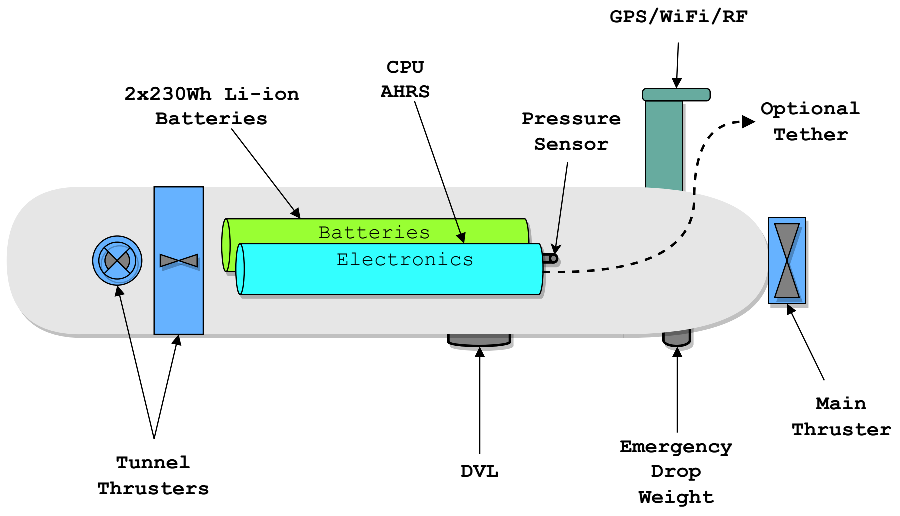


Figure 26: ALPHA AUV Revision 1 - Diagram

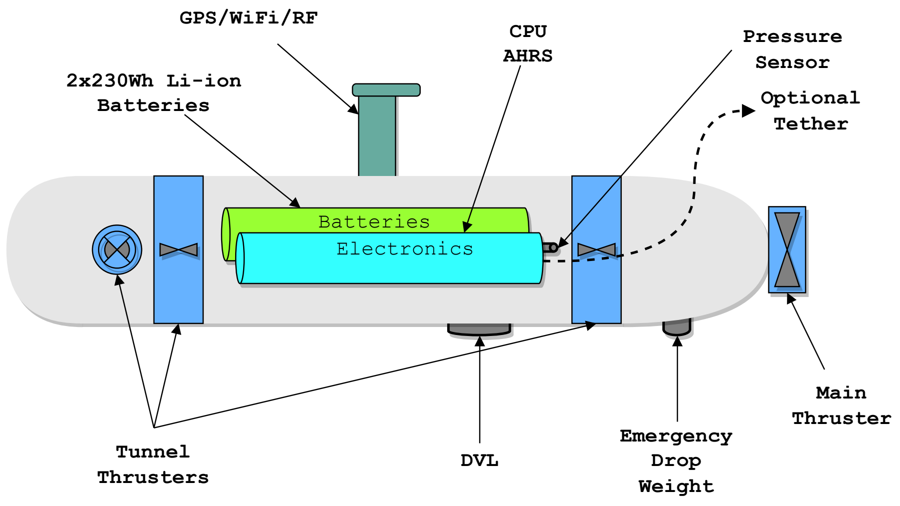


Figure 27: ALPHA AUV Revision 2 - Diagram

4.2 Experiment Setup

Figure 28 presents the system diagram of the ALPHA. To begin with, It is equipped with DVL, AHRS, GPS, and a pressure sensor for navigation, and sensor fusion for navigation was configured as same as the simulation environment. For processing, it has a Raspberry Pi 4 single board computer with 8-gigabyte memory that runs Ubuntu 20.04 LTS and ROS Noetic.

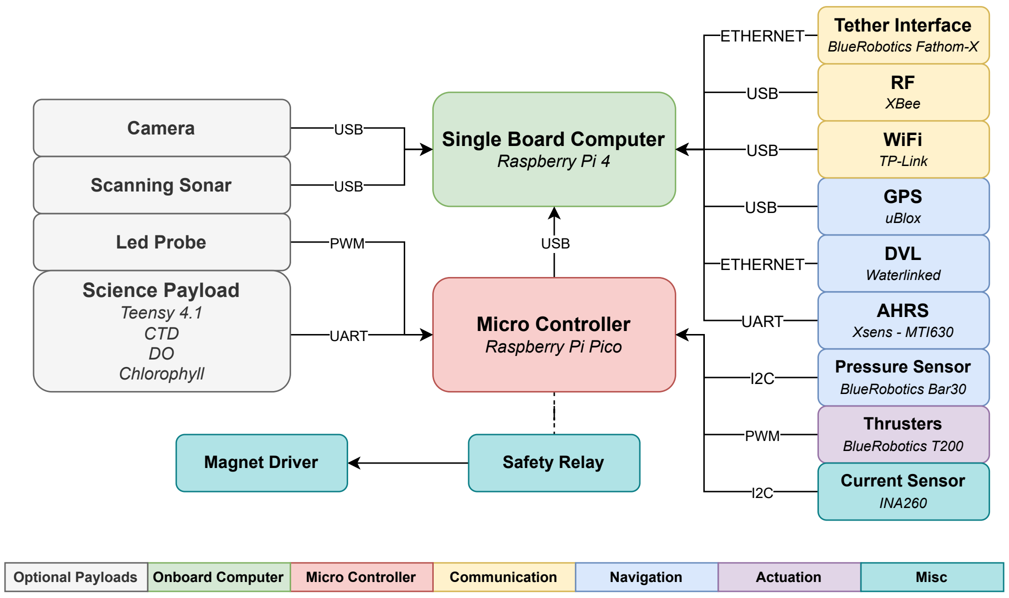


Figure 28: ALPHA AUV devices diagram. Devices that use hardware protocols such as Ethernet and USB are connected to the single board computer. Other devices that use protocols such as I2C and PWM are connected to the microcontroller. AHRS, because of the need for high-speed data transmission, is directly connected to the single board computer using UART serial protocol.

Communication with the onboard computer can be established in three different ways; WiFi (2.4 GHz), tether(single twisted pair), and RF(Radio Frequency - 900MHz). WiFi and tether link provides high-speed IPv4 (Internet Protocol version 4) connection over a short range. SSH (Secure Shell) provides access to the vehicle command line and the HTTP (Hypertext Transfer Protocol) relay server provides access to DVL's configuration interface over this link. In addition to that, an RF link provides long-range communication ranging up to 28 miles (line-of-sight) at 20 Kbps, but this link is much slower compared to a tether and WiFi connection. This link simply provides a serial protocol connection. To make use of this low-level hardware connection, `getty` program had been used. `getty` enables access to the command line through a serial channel, and unlike SSH, `getty` shell is not encrypted. Therefore, the security of the RF link is achieved natively using AES Encryption.

ALPHA is integrated with a drop weight system designed using a permanent electromagnet. It contains a separate circuit and an independent battery pack source. The MCU is responsible for drop weight circuit operation by controlling a relay. When the MCU is powered on, the relay is activated and the relay cuts the drop weight circuit's power. MCU can check for low-voltage battery state and can decide to activate the drop weight system as well.

4.3 Results

Field trials have been conducted in Narragansett Bay with both revisions of ALPHA. The presented results are from the field trials conducted in Narragansett Bay on June 28th and August 15th, 2022. During the field trials, path following, depth tracking, and periodic surfacing behaviors were used with the parameters shown in Table 9.

	June 28th	August 15th	August 15th
Vehicle Revision	`Rev 1`	`Rev 2`	`Rev 2`
Figure	Fig. 29	Fig. 31	Fig. 33
Mission Type	Square	Square	Lawn Mover
Acceptance radius	3.0m	3.0m	3.0m
LOS Lookahead distance	3.0m	5.0m	5.0m
Surge Velocity	0.7m/s	0.8m/s	0.8m/s
Surface Duration	10s	10s	10s
Surface Period	120s	60s	60s
Depth	5.0m	2.0m	2.0m
Depth tracking lookahead distance	5.0m	8.0m	8.0m

Table 9: Behavior parameters for presented results

Figures 29, 31, and 33 show the vehicle trajectory, depth tracking, and heading performance from the field trials. The vehicle trajectory is displayed in lines with color indicating the elapsed time in a mission. The planned course is depicted with black lines. The downward and upward-looking triangles show the mission start and end, respectively. The middle plots in these figures indicate the depth change during the mission. In depth plots (the middle plots in these figures), the yellow data points indicate the vehicle depth when the periodic surfacing behavior was active. For both heading (the bottom plots in these figures) and depth plots, blue indicates the current vehicle state (e.g., heading and depth) and red indicates the control input.

In the first mission (see Fig. 29) on June 28th, the vehicle (ALPHA Revision 1) was programmed to follow a 50×50m square. The path following performance was acceptable since the root-mean-square (RMS) for the cross-track errors (1.16m) was less than the vehicle length ($\sim$ 1.5m) (See Fig. 30). Some deviations were observed in the northern and the southern segments that may be caused by the transversal currents which may impact the horizontal tunnel thruster for heading keeping. In the East and West segments, the path tracking performance is better.

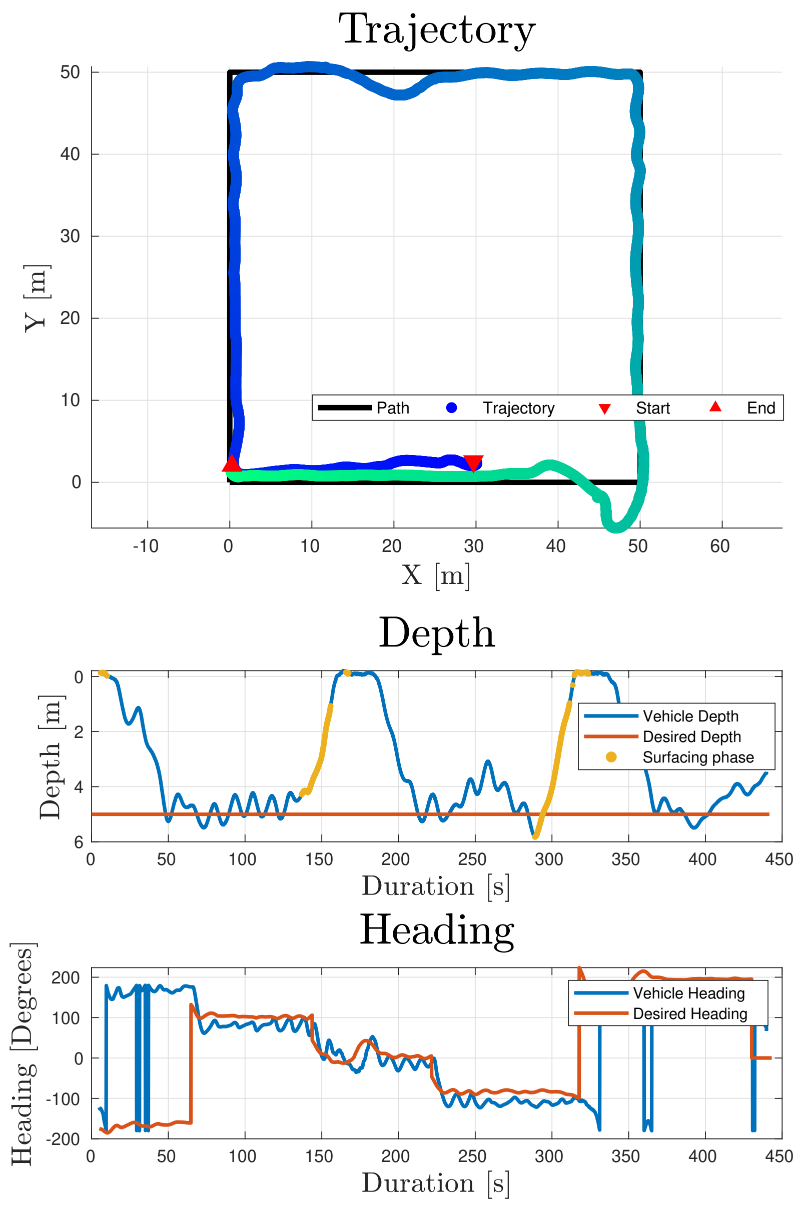


Figure 29: Field trial results from Narragansett Bay on 2022-06-28 with ALPHA Revision 1. In the trajectory plot (top), blue indicates the trajectory closer to the mission start, and green indicates the mission finish. The planned path is shown in black lines. In the depth plot (middle), the yellow data points indicate the surfacing phase, the red line indicates the desired depth, and the blue line indicates the vehicle depth. The heading plot (bottom) shows the vehicle heading in blue and the desired heading in red.

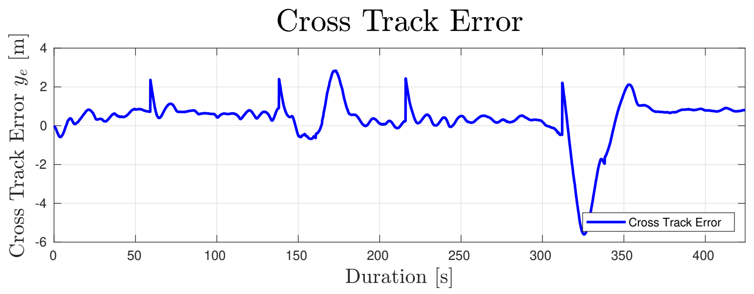


Figure 30: Cross track error for mission presented in Fig. 29, RMS = 1.161493m

On August 15, we conducted another square path mission with ALPHA revision 2 (see Fig. 31). In this mission, the vehicle was commanded to travel in a 50 × 50m square as before. The most notable improvement on revision 2 ALPHA was the depth tracking performance. Yet, some oscillation on depth control was observed in ALPHA revision 2 due to improper PID tuning. Path following performance was acceptable with cross track error RMS value of 0.75m, but some oscillations were observed on the vehicle path when following straight lines, (see Fig. 32). Comparing the heading plots in Fig. 32 to Fig. 31, we found ALPHA revision 2 has better heading maneuverability where the vehicle heading was kept close to the desired heading commanded from the path following behavior.

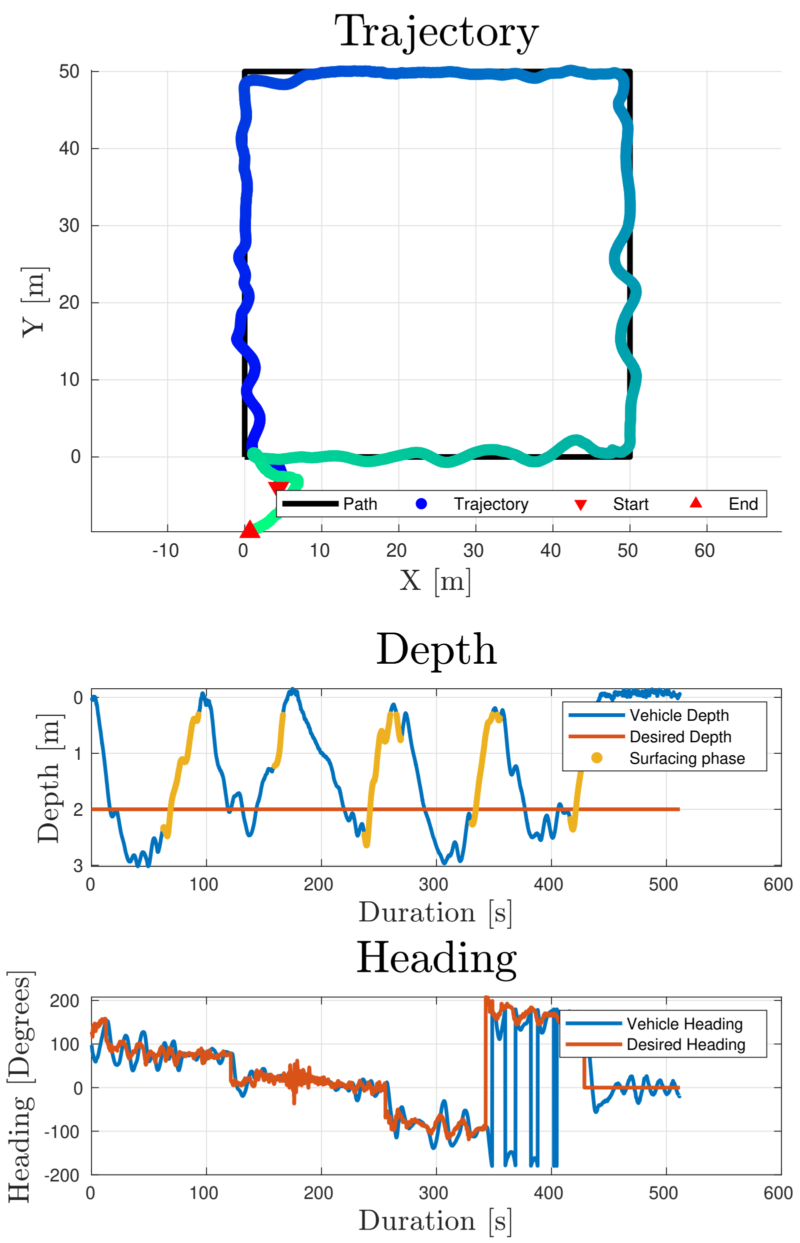


Figure 31: Field trial results from Narragansett Bay on 2022-09-15 with ALPHA Revision 2. In the trajectory plot (top), blue indicates the trajectory closer to the mission start, and green indicates the mission finish. The planned path is shown in black lines. In the depth plot (middle), the yellow data points indicate the surfacing phase, the red line indicates the desired depth, and the blue line indicates the vehicle depth. The heading plot (bottom) shows the vehicle heading in blue and the desired heading in red.

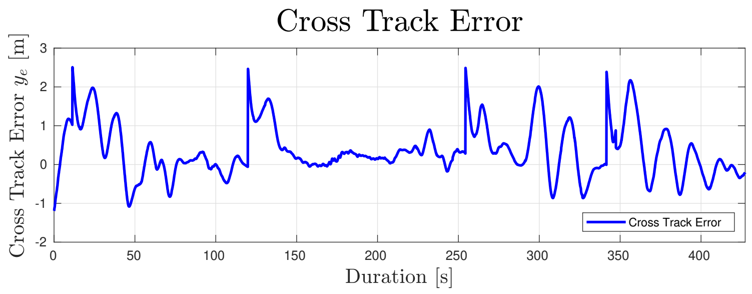


Figure 32: Cross track error for mission presented in Fig. 31, RMS = 0.749717m

Figure 33 shows vehicle trajectory, depth control, and heading control results for a lawn-mover pattern mission that was conducted with ALPHA revision 2. This mission commands the vehicle to travel on five 50-m-long transects that are 25m apart. This mission lasted approximately 15 minutes. The MVP framework successfully kept the vehicle on track with an RMS value of 0.71m (see Fig. 34 for cross track error). Since the periodical surfacing interval was programmed with a relatively short period (60 seconds), the resulting vehicle path in the vertical plane is similar to a sawtooth pattern similar to underwater gliders.

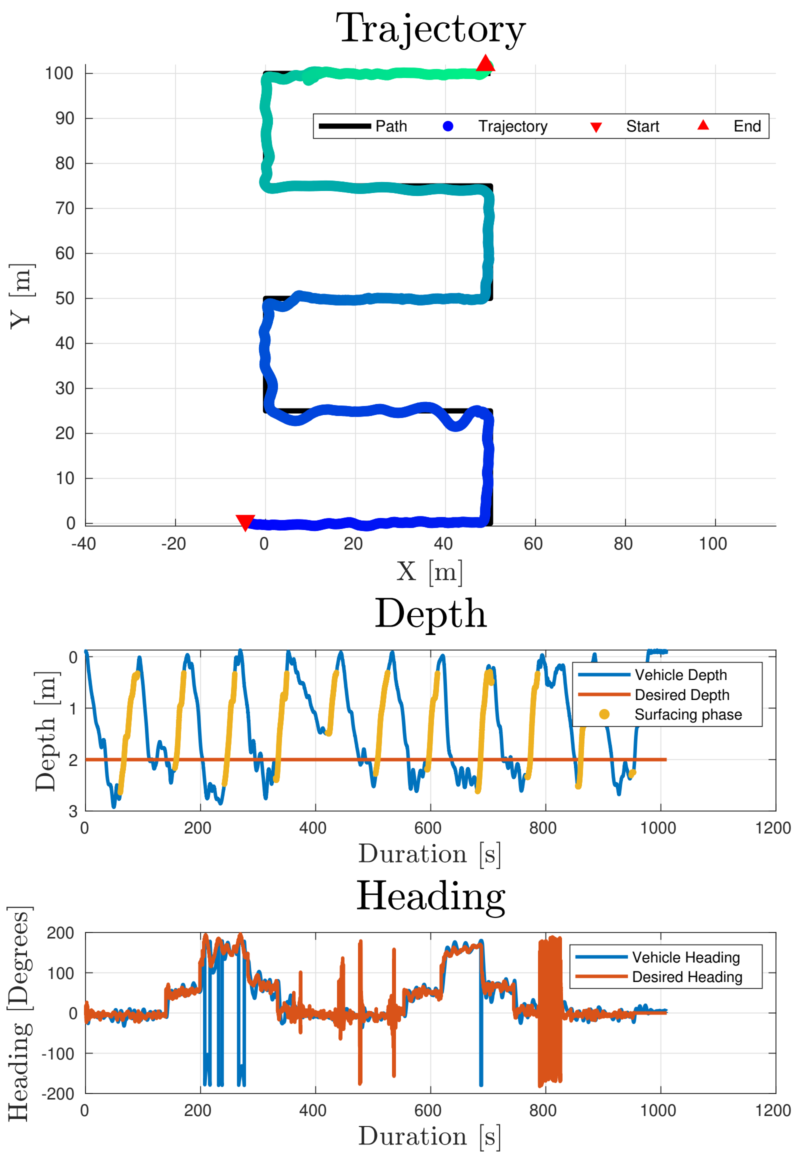


Figure 33: Field trial result from Narragansett Bay on 2022-09-15 with ALPHA Revision 2. In the trajectory plot (top), blue indicates the trajectory closer to the mission start, and green indicates the mission finish. The planned path is shown in black lines. In the depth plot (middle), the yellow data points indicate the surfacing phase, the red line indicates the desired depth, and the blue line indicates the vehicle depth. The heading plot (bottom) shows the vehicle heading in blue and the desired heading in red.

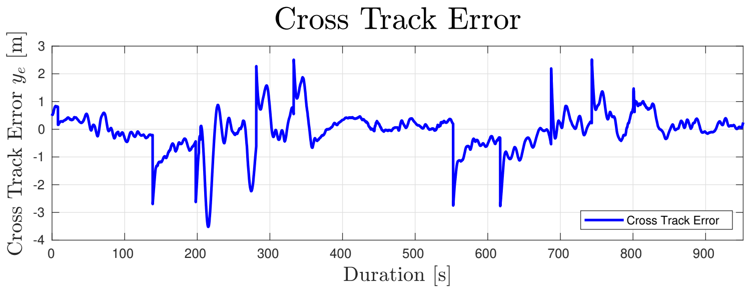


Figure 34: Cross track error for mission presented in Fig. 33, RMS = 0.706909m

CHAPTER 5

Conclusion and Future Work

In this thesis, a generic ROS-based software framework, ROS-MVP, is presented for marine robotic platforms, such as AUVs and ASVs. The introduced architecture has two main components; a low-level vehicle controller (MVP-Control), and a mission controller stack (MVP-Helm and behaviors). The first component, MVP-Control, provides generic multiple DOF vehicle controllers using MIMO-PID with QP optimization. It uses odometry information from a navigation source, such as the ROS robot localization [1]. The second component, MVP-Helm, pilots the vehicle by executing behaviors and sending their outputs to the vehicle controller. A finite state machine in the MVP-Helm decides behavior execution and a priority pool selects the actions with the highest priority from the behavior results. Finally, MVP-Behavior offers a basic interface for the construction of behavior.

MVP framework performance and functionality have been validated both in simulation and field trials with ALPHA AUV [2]. It was shown that the MVP framework is capable of controlling the ALPHA AUV in different environments. In the future, the MVP framework is planned to be further tested on the ASV platforms and AUV platforms.

The current version of the MVP architecture has several limitations. First, it currently doesn't support control surfaces such as fins and masts, and azimuth thrusters. The availability of testing platforms is the main reason leading to the limitation. However, within the development process, the Stonefish [3] simulator is found to be a sufficient sandbox environment for developing and testing such a feature. Secondly, MVP is only tested in Ubuntu 20.04 operating system and is only compatible with ROS1. As the robotics community migrates to ROS2[4], it remains to be a crucial limitation. Plans have been outlined to upgrade the MVP framework for ROS2 and the author has taken ROS2 migration into consideration during the current development.

All the source code for the MVP-Framework was made public on the source code sharing platform, github.com [1,2]. It is planned to be maintained on that platform as well to increase its longevity. In the future, these two limitations and future issues will be resolved with the open-source contributions from both the author and the users.

Overall, with these changes and further developments, we expect to provide a customizable ROS-compatible GNC framework for marine vehicles that gives a head start on future initiatives involving new marine vehicles.

List of References

[1] T. Moore and D. Stouch, "A generalized extended kalman filter implementation for the robot operating system," in *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer, July 2014.

[2] M. Zhou, E. C. Gezer, W. McConnell, and C. Yuan, "Acrobatic low-cost portable hybrid auv (alpha): System design and preliminary results," in *OCEANS 2022: Hampton Roads*. IEEE, 2022.

[3] P. Cieślak, "Stonefish: An advanced open-source simulation tool designed for marine robotics, with a ros interface," in *OCEANS 2019 - Marseille*, 2019, pp. 1–6.

[4] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074

[1]Git repository for MVP Control is hosted at `https://github.com/uri-ocean-robotics/mvp_control`.

[2]Git repository for MVP Mission is hosted at `https://github.com/uri-ocean-robotics/mvp_mission`.

BIBLIOGRAPHY

Bachmayer, R., Whitcomb, L., and Grosenbaugh, M., "An accurate four-quadrant nonlinear dynamical model for marine thrusters: theory and experimental validation," *IEEE Journal of Oceanic Engineering*, vol. 25, no. 1, pp. 146–159, 2000.

Benjamin, M. R., Schmidt, H., Newman, P. M., and Leonard, J. J., "Nested autonomy for unmanned marine vehicles with moos-ivp," *Journal of Field Robotics*, vol. 27, no. 6, pp. 834–875, 2010.

Bhat, S., Torroba, I., Özkahraman, Ö., Bore, N., Sprague, C. I., Xie, Y., Stenius, I., Severholt, J., Ljung, C., Folkesson, J., *et al.*, "A cyber-physical system for hydrobatic auvs: system integration and field demonstration," in *2020 IEEE/OES Autonomous Underwater Vehicles Symposium (AUV)*. IEEE, 2020, pp. 1–8.

Brommer, C., Jung, R., Steinbrener, J., and Weiss, S., "MaRS : A Modular and Robust Sensor-Fusion Framework," 2020.

Caress, D. W., Thomas, H., Kirkwood, W. J., McEwen, R., Henthorn, R., Clague, D. A., Paull, C. K., Paduan, J., Maier, K. L., Reynolds, J., *et al.*, "High-resolution multibeam, sidescan, and subbottom surveys using the mbari auv d. allan b," *Marine habitat mapping technology for Alaska*, pp. 47–69, 2008.

Carreras, M., Hernández, J. D., Vidal, E., Palomeras, N., Ribas, D., and Ridao, P., "Sparus ii auv—a hovering vehicle for seabed inspection," *IEEE Journal of Oceanic Engineering*, vol. 43, no. 2, pp. 344–355, 2018.

Chen, C.-W., Kouh, J.-S., and Tsai, J.-F., "Modeling and simulation of an auv simulator with guidance system," *IEEE Journal of Oceanic Engineering*, vol. 38, no. 2, pp. 211–225, 2013.

Cieślak, P., "Stonefish: An advanced open-source simulation tool designed for marine robotics, with a ros interface," in *OCEANS 2019 - Marseille*, 2019, pp. 1–6.

Coumans, E., "Bullet physics simulation," in *ACM SIGGRAPH 2015 Courses*, 2015, p. 1.

Daniel, T., Manley, J., and Trenaman, N., "The wave glider: enabling a new approach to persistent ocean observation and research," *Ocean Dynamics*, vol. 61, no. 10, pp. 1509–1520, 2011.

DeMarco, K., West, M. E., and Collins, T. R., "An implementation of ros on the yellowfin autonomous underwater vehicle (auv)," in *OCEANS 2011.* IEEE, 2011, pp. 1–7.

Duecker, D. A., Bauschmann, N., Hansen, T., Kreuzer, E., and Seifried, R., "Hippocampusx–a hydrobatic open-source micro auv for confined environments," in *2020 IEEE/OES Autonomous Underwater Vehicles Symposium (AUV).* IEEE, 2020, pp. 1–6.

Edge, C., Enan, S. S., Fulton, M., Hong, J., Mo, J., Barthelemy, K., Bashaw, H., Kallevig, B., Knutson, C., Orpen, K., *et al.*, "Design and experiments with loco auv: A low cost open-source autonomous underwater vehicle," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE, 2020, pp. 1761–1768.

Esteba, J., Cieślak, P., Palomeras, N., and Ridao, P., "Docking of non-holonomic auvs in presence of ocean currents: A comparative survey," *IEEE Access*, vol. 9, pp. 86 607–86 631, 2021.

Foote, T., "tf: The transform library," in *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, ser. Open-Source Software workshop, April 2013, pp. 1–6.

Fossen, T. I., "Guidance and control of ocean vehicles," *University of Trondheim, Norway, Printed by John Wiley & Sons, Chichester, England, ISBN: 0 471 94113 1, Doctors Thesis*, 1999.

Fossen, T. I. and Fjellstad, O.-E., "Nonlinear modelling of marine vehicles in 6 degrees of freedom," *Mathematical Modelling of Systems*, vol. 1, no. 1, pp. 17–27, 1995.

Fossen, T. I. and Johansen, T. A., "A survey of control allocation methods for ships and underwater vehicles," in *2006 14th Mediterranean Conference on Control and Automation*, 2006, pp. 1–6.

Freitag, L., Grund, M., Von Alt, C., Stokey, R., and Austin, T., "A shallow water acoustic network for mine countermeasures operations with autonomous underwater vehicles," *Underwater Defense Technology (UDT)*, pp. 1–6, 2005.

Galassi, M., Davies, J., Theiler, J., Gough, B., Jungman, G., Alken, P., Booth, M., Rossi, F., and Ulerich, R., *GNU scientific library.* Network Theory Limited, 2002.

Griffiths, A., Dikarev, A., Green, P. R., Lennox, B., Poteau, X., and Watson, S., "Avexis—aqua vehicle explorer for in-situ sensing," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 282–287, 2016.

Gu, N., Wang, D., Peng, Z., Wang, J., and Han, Q.-L., “Advances in line-of-sight guidance for path following of autonomous marine vehicles: An overview,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–17, 2022.

Hoegh-Guldberg, O., “Reviving the ocean economy: the case for action,” 2015.

Jones, C., Allsup, B., and DeCollibus, C., “Slocum glider: Expanding our understanding of the oceans,” in *2014 Oceans - St. John’s*, 2014, pp. 1–10.

Kimball, P., Bailey, J., Das, S., Geyer, R., Harrison, T., Kunz, C., Manganini, K., Mankoff, K., Samuelson, K., Sayre-McCord, T., Straneo, F., Traykovski, P., and Singh, H., “The whoi jetyak: An autonomous surface vehicle for oceanographic research in shallow or dangerous waters,” in *2014 IEEE/OES Autonomous Underwater Vehicles (AUV)*, 2014, pp. 1–7.

Koenig, N. and Howard, A., “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.

Lekkas, A. M. and Fossen, T. I., “Line-of-sight guidance for path following of marine vehicles,” *Advanced in marine robotics*, pp. 63–92, 2013.

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., and Woodall, W., “Robot operating system 2: Design, architecture, and uses in the wild,” *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074

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

Marthiniussen, R., Vestgard, K., Klepaker, R., and Storkersen, N., “Hugin-auv concept and operational experiences to date,” in *Oceans ’04 MTS/IEEE Techno-Ocean ’04 (IEEE Cat. No.04CH37600)*, vol. 2, 2004, pp. 846–850 Vol.2.

Mayer, L., Jakobsson, M., Allen, G., Dorschel, B., Falconer, R., Ferrini, V., Lamarche, G., Snaith, H., and Weatherall, P., “The nippon foundation—gebco seabed 2030 project: The quest to see the world’s oceans completely mapped by 2030,” *Geosciences*, vol. 8, no. 2, p. 63, 2018.

Meinig, C., Lawrence-Slavas, N., Jenkins, R., and Tabisola, H. M., “The use of saildrones to examine spring conditions in the bering sea: Vehicle specification and mission performance,” in *OCEANS 2015 - MTS/IEEE Washington*, 2015, pp. 1–6.

Moore, T. and Stouch, D., “A generalized extended kalman filter implementation for the robot operating system,” in *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13).* Springer, July 2014.

Mordy, C. W., Cokelet, E. D., De Robertis, A., Jenkins, R., Kuhn, C. E., Lawrence-Slavas, N., Berchok, C. L., Crance, J. L., Sterling, J. T., Cross, J. N., *et al.*, “Advances in ecosystem research: Saildrone surveys of oceanography, fish, and marine mammals in the bering sea,” *Oceanography*, vol. 30, no. 2, pp. 113–115, 2017.

Mourikis, A. I. and Roumeliotis, S. I., “A multi-state constraint kalman filter for vision-aided inertial navigation,” in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 3565–3572.

Newman, P. M., “Moos-mission orientated operating suite,” 2008.

Palomeras, N., El-Fakdi, A., Carreras, M., and Ridao, P., “Cola2: A control architecture for auvs,” *IEEE Journal of Oceanic Engineering*, vol. 37, no. 4, pp. 695–716, 2012.

Pi, R., Cieślak, P., Ridao, P., and Sanz, P. J., “Twinbot: Autonomous underwater cooperative transportation,” *IEEE Access*, vol. 9, pp. 37 668–37 684, 2021.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A. Y., *et al.*, “Ros: an open-source robot operating system,” in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

Ribas, D., Palomeras, N., Ridao, P., Carreras, M., and Mallios, A., “Girona 500 auv: From survey to intervention,” *IEEE/ASME Transactions on Mechatronics*, vol. 17, no. 1, pp. 46–53, 2012.

Rudnick, D. L., Davis, R. E., Eriksen, C. C., Fratantoni, D. M., and Perry, M. J., “Underwater gliders for ocean research,” *Marine Technology Society Journal*, vol. 38, no. 2, pp. 73–84, 2004.

Schofield, O., Kohut, J., Aragon, D., Creed, L., Graver, J., Haldeman, C., Kerfoot, J., Roarty, H., Jones, C., Webb, D., *et al.*, “Slocum gliders: Robust and ready,” *Journal of Field Robotics*, vol. 24, no. 6, pp. 473–485, 2007.

Snyder, M., Weaver, J. N., and Bays, M. J., “Ros-ivp: Porting the interval programming suite into the robot operating system for maritime autonomy,” in *OCEANS 2016 MTS/IEEE Monterey*, 2016, pp. 1–6.

Stellato, B., Banjac, G., Goulart, P., Bemporad, A., and Boyd, S., “OSQP: an operator splitting solver for quadratic programs,” *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020. [Online]. Available: https://doi.org/10.1007/s12532-020-00179-2

Viquez, O. A., Fischell, E. M., Rypkema, N. R., and Schmidt, H., “Design of a general autonomy payload for low-cost auv r&d,” in *2016 IEEE/OES Autonomous Underwater Vehicles (AUV)*, 2016, pp. 151–155.

Wang, W., Shan, T., Leoni, P., Fernández-Gutiérrez, D., Meyers, D., Ratti, C., and Rus, D., “Roboat ii: A novel autonomous surface vessel for urban environments,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 1740–1747.

Wang, X., Shang, J., Luo, Z., Tang, L., Zhang, X., and Li, J., “Reviews of power systems and environmental energy conversion for unmanned underwater vehicles,” *Renewable and Sustainable Energy Reviews*, vol. 16, no. 4, pp. 1958–1970, 2012.

Widditsch, H., “Spurv-the first decade,” WASHINGTON UNIV SEATTLE APPLIED PHYSICS LAB, Tech. Rep., 1973.

Wolek, A., McMahon, J., Dzikowicz, B. R., and Houston, B. H., “Tracking multiple surface vessels with an autonomous underwater vehicle: Field results,” *IEEE Journal of Oceanic Engineering*, vol. 47, no. 1, pp. 32–45, 2022.

Yoerger, D. R., Bradley, A. M., Walden, B. B., Singh, H., and Bachmayer, R., “Surveying a subsea lava flow using the autonomous benthic explorer (abe),” *International Journal of Systems Science*, vol. 29, no. 10, pp. 1031–1044, 1998.

Zhang, L., Merrifield, R., Deguet, A., and Yang, G.-Z., “Powering the world’s robots—10 years of ros,” *Science Robotics*, vol. 2, no. 11, p. eaar1868, 2017.

Zhang, M. M., Choi, W.-S., Herman, J., Davis, D., Vogt, C., McCarrin, M., Vijay, Y., Dutia, D., Lew, W., Peters, S., and Bingham, B., “Dave aquatic virtual environment: Toward a general underwater robotics simulator,” in *2022 IEEE/OES Autonomous Underwater Vehicle (AUV) Symposium*, 2022, pp. 1–8.

Zhao, L., Zhou, M., and Loose, B., “Towards under-ice sensing using a portable rov,” in *OCEANS 2022: Hampton Roads*. IEEE, 2022.

Zhou, M., Gezer, E. C., McConnell, W., and Yuan, C., “Acrobatic low-cost portable hybrid auv (alpha): System design and preliminary results,” in *OCEANS 2022: Hampton Roads*. IEEE, 2022.