

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE
LVIV POLYTECHNIC NATIONAL UNIVERSITY

- Qualifying scientific work on the rights of the manuscript

ZENG XINYU

UDC 681.5

THESIS OF DISSERTATION

On theme

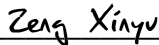
**HARDWARE-SOFTWARE AND METROLOGICAL SUPPORT OF
DRONES**

152 – Metrology and Information - Measuring Equipment

15 - Automation and instrumentation

Applying for the Doctor of Philosophy degree

The dissertation contains the research results. The ideas, results and texts of other authors are linked to the corresponding reference.

Signed by:
 Zeng Xinyu
EFE6893F157642F...

Scientific supervisor: Yatsyshyn S.P., Doctor of Technical Sciences, Professor

Lviv, Ukraine - 2025

ABSTRACT

Xinyu Zeng, Hardware-Software and Metrological Support of Drones. – Qualifying scientific work on manuscript rights. Dissertation for the degree of Doctor of Philosophy in specialty 152 "Metrology and Information-Measuring Techniques" - Lviv Polytechnic National University, Ministry of Education and Science of Ukraine, Lviv, 2025.

This dissertation focuses on the research, development, and implementation of advanced soft and hardware systems used in drones, with a particular emphasis on metrological support for their performance optimization. A significant part of the dissertation is devoted to ensuring the accuracy and reliability of measurements in drone operations through advanced calibration techniques and testing platforms, due to the drone's ability to perform autonomously with a high degree of precision in various environments.

Chapter 1: Starting with Amphibious Design

The first chapter introduces the design and development of a drone system capable of amphibious operations, taking inspiration from water striders. The chapter explores the integration of hardware such as Nvidia Jetson for real-time control and ROS for system modularity and software integration. The aim is to ensure smooth transitions between water-surface and underwater operations, with a particular focus on the metrological challenges involved in designing a system that operates in two dissimilar physical environments. Additionally, this chapter discusses the technical and environmental factors that influence the accuracy of measurements and their significance in the design process.

Chapter 2: Building a Test Platform for Hydrodynamic Performance

The second chapter focuses on the development of a test platform to measure the hydrodynamic performance of the drone in real-world water conditions. Key parameters such as thrust, energy consumption, and response speed are analyzed to optimize the propulsion system's performance. The chapter discusses the metrological provision required to ensure accurate measurements, considering factors like sensor calibration, environmental impacts, and the repeatability of measurements. By evaluating the drone's performance in controlled conditions, this chapter provides insights into the relationship between energy usage and propulsion efficiency, laying the groundwork for further refinement.

Chapter 3: Enhancing Stability through Filtering Algorithms

The third chapter addresses the control challenges posed by the drone's underwater environment, with a focus on optimizing its stability and posture through advanced filtering algorithms. The integration of Complementary and Kalman Filters helps improve the drone's attitude control, allowing it to maintain stability and recover quickly from disturbances. This chapter emphasizes the importance of real-time data processing in enhancing the drone's performance, particularly in dynamic environments, where precise control is critical. The filtering techniques discussed here provide metrological provision by minimizing error and improving the reliability of sensor data.

Chapter 4: Data-Driven Hydrodynamic Modeling

The fourth chapter presents a data-driven hydrodynamic model, developed using real-time data from force sensors installed on the drone. This model allows the drone to adapt to various underwater conditions by predicting hydrodynamic forces and adjusting its behavior accordingly. The application of machine learning techniques, such as linear regression, helps optimize the drone's movements and enhances its

operational efficiency. This chapter demonstrates how metrological support is vital in validating the accuracy of the data collected and ensuring that the model reflects real-world conditions accurately. The results of this chapter show significant improvements in the drone's performance, particularly in terms of accuracy and energy efficiency.

Keywords: drone, hardware, software, metrological support, control, uncertainty, error, sensor calibration, hydrodynamic modeling and simulation, filtering algorithms, underwater robotics, performance assessing.

ARTICLES PUBLISHED WHILE PERFORMING THE CURRENT PHD WORK

1. X. Zeng, Olha Lysa, "Response Time in Inertial Measurement Unit Control Algorithms", *Measuring Equipment and Metrology*, Volume 85, Number 2, pp. 5-8, 2024.
<https://doi.org/10.23939/istcmtm2024.02.005>
2. X. Zeng, S. Yatsyshyn, "Test Platform Paradigm for Underwater Dynamics Measurements", *Measuring Equipment and Metrology*, Volume 85, Number 1, pp. 29-34, 2024.
<https://doi.org/10.23939/istcmtm2024.01.029>
3. S. Yatsyshyn, X. Zeng, "Design of the Water Strider-like Robot", *Measuring Equipment and Metrology*, Volume 84, Number 3, pp. 39-42, 2023.
<https://doi.org/10.23939/istcmtm2023.03.039>
4. Z. Wang, Y. Yan, X. Zeng, R. Li, W. Cui, Y. Liang, D. Fan, *Joint multi-objective optimization based on multitask and multi-fidelity Gaussian processes for flapping foil*, *Ocean Engineering*, Volume 294, 15 February 2024, 116862. <https://doi.org/10.1016/j.oceaneng.2024.116862>.
5. Q. Liu, H. Chen, P. Guo, G. Su, W. Li, X. Zeng, D. Fan, W. Cui, *Unified scheme design and control optimization of flapping wing for next-generation manta ray robot*, *Ocean Engineering*, Volume 309, Part 2, 1 October 2024, 118487. <https://doi.org/10.1016/j.oceaneng.2024.118487>.
6. S. Yatsyshyn, X. Zeng, Adaptive modeling of underwater robot fluid dynamics based on force measurement device, *Measuring Equipment and Metrology*, Volume 85, Number 4, pp. 7-13, 2024.
<https://doi.org/10.23939/istcmtm2024.04.007>
7. S. Yatsyshyn, A. Cherkas, X. Zeng, Hardware and software of water strider robot, International Scientific and Practical Conference IVT-2022, Lviv, Ukraine, 09-10 November 2022, pp. 146–147.
8. X. Zeng, S. Yatsyshyn, Test platform paradigm for underwater object's measurements, VI International Scientific and Practical Conference "Quality Management in Education and Industry: Experience, Problems, and Perspectives", Lviv, Ukraine, 16–17 November 2023, pp. 157–158.
9. S. Yatsyshyn, X. Zeng, Metrological risks at design stage for multidisciplinary-based objects, 60th Ilmenau Scientific Colloquium

"Engineering for a Changing World", Technische Universität Ilmenau, September 04–08, 2023, pp. 58677-1–58677-7.

- 10.X. Zeng, S. Yatsyshyn, The exactness of ultrasound sensors of robotics, II International Scientific and Practical Conference "Information and Measurement Technologies IVT-2024", Lviv, Ukraine, 13–14 November 2024, pp. 139–140.

CONTENT

List of conventional designations.....	9
Introduction	11
General characteristics of work.....	13
Chapter 1 Design of an Amphibious Robot for Surface and Underwater Operations	19
1.1 Conceptualization and Design of a Generalized Amphibious Robot with Metrological Assurance.....	20
1.2 Hardware System: Nvidia Jetson and Control Architecture	25
1.3 Integration of ROS for Efficient Communication.....	31
1.4 Integration of Hardware and Software for Real-Time Control in Amphibious Robots.....	35
Chapter 2 Building a Test Platform for Hydrodynamic Performance	41
2.1 Propulsion Testing: Precision Hydrodynamic Measurement and Calibration..	42
2.2 Exploring Alternative Underwater Propulsion Systems: The Approach to Hydrodynamic Optimization.....	53
Chapter 3 Enhancing Stability and Depth Control through Sensors Fusion.....	64
3.1 IMU-Based Attitude Control.....	64
3.2 Depth-Based Control.....	74
Chapter 4 Data-Driven Approaches and Optimization Methods for Amphibious Robot Applications.....	87
4.1 Robot Structure and Motion Mechanisms.....	87
4.2 GPR-Based Hydrodynamic Modeling	95
4.3 Stability and Control During Amphibious Robot Transitions	115
Conclusions	122

References	126
Appendixes	137

LIST OF CONVENTIONAL DESIGNATIONS

IMU – Inertial Measurement Unit

R&D – research and development

GPS – Global Positioning System

ROS – Robot Operating System

UAV - Unmanned Aerial Vehicle

UWV - Unmanned Water Vehicle

DC – Direct Current

IMU - Inertial Measurement Unit

PZT – Lead Zirconate Titanate (used in actuators and sensors)

PWM – Pulse Width Modulation

GPIO – General-Purpose Input/Output

I2C – Inter-Integrated Circuit

SPI – Serial Peripheral Interface

UART – Universal Asynchronous Receiver-Transmitter

DMI – Digitalize Miniaturize Inteligentized

USB – Universal Serial Bus

DVL – Doppler Velocity Log

LIDAR – Light Detection and Ranging

DDS – Data Distribution Service

QoS – Quality of Service

GPU – Graphics Processing Unit

TTL – Transistor-Transistor Logic

RS485 – Recommended Standard 485 (used for serial communication)

MPC – Model Predictive Control

PID – Proportional-Integral-Derivative

AUV – Autonomous Underwater Vehicle

ROV – Remotely Operated Vehicle

MEMS – Micro-Electro-Mechanical Systems

FS – Full Scale

St – Strouhal Number

Re – Reynolds Number

RBF – Radial Basis Function

MLE – maximum likelihood estimation

NLML – negative log marginal likelihood

RMSE – Root Mean Squared Error

MAE – Mean Absolute Error

INTRODUCTION

The formation of the scientific work "Hardware-Software and Metrological Support of Drones" immediately refers its topic to a number of the most difficult R&Ds of a scientific and technical product - drones, especially of marine applications. Similar topics were preceded by centuries of intense work in this field.

The previously obtained research results made a significant contribution to the development of the first two generations of underwater robots while the current one differs in the use of flexible materials with large deformations instead of traditional metal structures and buoyancy materials. The production processes of underwater vehicles have been innovatively changed thanks to the application of 3D printing technologies, which have replaced the conventional methods of welding and forging. In addition, traditional control systems have been replaced by artificial intelligence, and standard sensors have been replaced by nanosensors.

Returning to metrology, let's recall the classic's Lord Kelvin [1] back in the 19th century, acting as the Chief-engineer of the intercontinental communication project with the help of electric cables, faced the problem of measuring the electrical resistance of the insulation of a submerged cable. As a result, he invented a device that we use today. Namely, the light beam or mirror galvanometer "lengthens" the needle of the tool with a light beam, increasing the sensitivity of the measurement.

Similar things can work in modern drones, where the a priori limited size and weight device is equipped with the sensors and actuators necessary for operation and metrological verification and provision. For example, in [2], it was found that when a 24-bit ACD was used during design development, and a 16-bit ACD during its

operation, additional component errors occurred, caused by the industrial replacement of the ACD.

When moving, for example, the arm of a robot - from p. A to p.B - a number of spatial characteristics due to stability over time become important. In total, these concepts in the case of mechanical integrity of the structure are to be described by the term "metrological reliability". In the case of drones, its manifestations are repeatability and reproducibility of characteristics, their drift, etc. At the same time, when moving, the drone must be controlled by fairly sophisticated control programs, which include, for example, MPC control software.

The "water" specifics of the design of the drone, which corresponds to the direction and the task of the dissertation, are characterized by significant mass-volume limitations, which are transferred to hardware and software-technical ones. Therefore, it is in this order that they are considered further in this work: first, the design of the tool, then the software and technical solutions aimed at obtaining reliable metrological results in the water environment and, finally, the repeatability and reproducibility of the characteristics for a small number of samples or repetitions when using the same drone, which is achieved due to metrological support.

GENERAL CHARACTERISTICS OF WORK

Justification of the topic of research. For the further development of marine technologies, for example, for the extraction of metals, it is necessary to develop new technologies that were not available yesterday. The example of water drones is quite clear, as the war in the Black Sea in 2022-2024 highlighted this. In our case, the setting of the topic was carried out and executed in parallel. The trinity of the topic: hardware, software, and metrological support act as the cornerstones of the foundation of the dissertation. Each of them and in every aspect contributes to the implementation of the mentioned research, relying on sub-technologies that form the rapid rise of Industry 4.0. Among those involved in use we note the information flows in difficult environmental conditions, the involvement of methods and means of processing information, its management, the development of special methods of metrological support, increasing the accuracy of measurements and reducing the uncertainty of the implementation of drone's functions.

Unmanned controlled vehicles, in this case - water ones, are rapidly developing, using technologies and smart devices in water conditions to ensure, for example, to receive and transmit information through underwater communication.

Connection of dissertation work with plans, topics, and scientific programs. The dissertation work is aligned with the fixed scientific direction of the Department of Information and Measurement Technologies - theoretical and applied foundations of metrology and measurements in information technologies (information and measurement, cyber-physical, robotic, and other systems); product and software quality testing.

The purpose and tasks of the research.

The main goal of this research is the development and implementation of a comprehensive system of metrology support for UAVs and UWVs which integrates both software and hardware components. This system improves the accuracy of measurements related to movement parameters such as altitude, speed, depth, and positioning, ensuring the reliable operation of drones in both dissimilar environments.

To achieve this goal, it is necessary to perform the following tasks:

- To analyze existing designs of unmanned robotic vehicles (drones) for movement in the air, on water, and under it, as well as to determine recommendations for metrological support of certain types of drones
- To study current control systems for drone launch and movement; to investigate the metrological basis of accuracy, reproducibility, and other characteristics, especially for systems of underwater drones: control, navigation, communication, etc.
- To study the characteristics and develop methods ensuring the calibration of sensors used in drones, including accelerometers, gyroscopes, GPS systems, altimeters, and depth sensors.
- To investigate digital and special methods of improving the accuracy of information production by drones in the underwater environment, including through the implementation of digital filtering methods.
- To propose a methodology for the real-time metrological support of hardware and software elements of underwater drones.

Object of research

The thesis focus is metrological support for drones, with an emphasis on hardware and software systems that measure and control key parameters such as positioning, speed, and depth in various industries, including logistics, surveillance, and an

underwater research.

The subject of the study is the characteristics and calibration methods of sensors and control systems integrated into drones. The research also covers the development of software tools for real-time monitoring, diagnostics, and calibration of drones in both aerial and underwater environments. Particular attention is paid to improving metrological repeatability and accuracy, as well as high precision, such as marine and underwater surveys.

Research methods

The study combines both theoretical and practical approaches to metrological support, in particular:

- Analysis of existing standards and measurement methods for drones, focusing on sensor calibration and accuracy of control systems for both aerial and underwater applications.
- Experimental calibration of hardware components such as GPS modules, inertial measurement units (IMUs), altimeters, and depth sensors.
- Development of software algorithms to improve the accuracy of real-time measurements and error adjustment calibration for drones.
- Using modeling tools to simulate drone trajectory and underwater dynamics and analyze the impact of metrological errors on their performance.
- To effectively manage and interpret sensor data, issue the employed Gaussian Process Regression (GPR) by modeling the underlying uncertainty in fluid-structure interactions, which allows for more precise predictions in complex and varying environments.
- The probabilistic nature of GPR enables quadraped robots to handle noisy data and provide robust, uncertainty-aware decision-making strategies. That seems

to be the advanced metrology equipment and calibration systems from the world's leading scientists and focuses on improving the accuracy of drone measurements, ensuring reliable operation in surveillance and underwater environments.

Scientific novelty.

The following scientific results were obtained in the dissertation work:

1. The method for testing the dynamic characteristics of amphibious robots has been enhanced. It utilizes an integrated approach to trajectory drift control, enabling the robots to adapt effectively to variable and complex underwater conditions.

2. The methods for filtering signals from inertial measurement devices have been optimized through the integration of advanced hardware and software solutions.

3. A predictive model has been developed using machine learning techniques to analyze the influence of hydrodynamic forces and immersion depth. This model enhances the maneuverability and stability of movement in amphibious robots.

Application of research results. The research spans various disciplines, including ocean engineering, robotics, mechanics, materials science, energy, control systems, computer science, and sensor technology. A key achievement was the successful sea trial of the 2,000m Sigu I bionic underwater vehicle in 2023 in Hainan, China.

The research also led to an advanced platform for parallel underwater data collection. This platform has been optimized for improved communication protocols and solving synchronization problems in multi-system environments. It is capable of recording an average of 5,760 sets of hydrodynamic data per day and integrates with automated experimental equipment to collect and analyze parameters in real-time.

In addition, the research has played a key role in the development of a dual-environment quadruped robot that uses IMU sensors and data fusion to provide a seamless transition between terrestrial and underwater environments. By increasing the frequency of sensing algorithms and integrating data from the hydrodynamic testing platform, accurate hydrodynamic modeling was created to optimize the robot's motion underwater.

In addition, the robotic buoy developed during the study was funded under the R&D initiative in Zhejiang Province in 2023. It has become an important tool in identifying key factors affecting the state of aquatic ecosystems and in measures to restore them.

The obtained results were used in the educational process by the department "Information and measurement technologies" of the National University "Lviv Polytechnic" for the training of specialists in the specialty 152 "Metrology and Information - Measuring Equipment " and in the specialty 175 "Information – Measuring Technologies", including masters in teaching the discipline "Robotics, systems and complexes", and graduate students in teaching the discipline "Analytical and numerical research methods".

Personal contribution of the acquirer. Algorithms for maintaining the balance of the robot were personally developed and implemented by the acquirer. After which they were applied by a leading robotics company in China. These algorithms have ensured the stability and accuracy of the movements of robots used in various fields of industry and research activities. In addition, the acquirer optimized control systems and integration of sensor data to increase the efficiency of robots in difficult operating conditions, which contributed to the acceleration of the process of developing new robotic solutions.

Approbation of the results. The scientific propositions and research results presented in the work were reported and discussed at Ukrainian and international scientific-practical and scientific-technical conferences: 1st and 2nd International Scientific Conference "Information and Measurement Technologies", 22/10/2022 and 13/11/2024; Lviv, Ukraine, and 60th International Scientific Colloquium, Sept. 04-07, 2023, Ilmenau, Germany.

Structure and scope of work.

The composition of the dissertation includes: a list of notations, an introduction, 4 main sections with conclusions to them, general conclusions, a list of references and appendixes. The total volume of the work is 163 pages, of which 136 pages are the main text, containing 35 figures and 8 tables. The references include 72 items.

Publications. Based on the results of the dissertation research, 10 scientific works were published, of which 4 articles were published in specialized publications of Ukraine, 2 articles was published in international publications (Scopus), as well as 4 theses in collections of international scientific conferences.

Chapter 1

Design of an Amphibious Robot for Surface and Underwater Operations

The evolution of robotics has brought about significant advancements in the design of robots that can operate in multiple environments. One of the most challenging yet promising fields is the development of amphibious robots—robots that can transition seamlessly between surface and underwater environments [3]. These robots are increasingly important in fields such as environmental monitoring, search and rescue, and underwater exploration, where mobility and adaptability are critical to success.

This chapter introduces the conceptualization, design, and development of a generalized amphibious robot, highlighting both the hardware and software systems required for its operation. The chapter begins by exploring the fundamental design principles for creating a robot capable of both surface and underwater locomotion, focusing on the robot's mechanical structure, propulsion systems, and buoyancy control mechanisms. The integration of cutting-edge hardware—such as the Nvidia Jetson platform—and advanced software systems—particularly the ROS (Robot Operating System) framework—are examined in detail, showcasing how these technologies enable real-time control and adaptability in dynamic aquatic environments.

Additionally, the chapter addresses the importance of metrological assurance in the robot's design. Ensuring the accuracy and reliability of the robot's sensors, actuators, and control systems is essential for maintaining performance across a variety of conditions. Calibration techniques, sensor fusion, and environmental

compensation methods are discussed to ensure that the robot can operate effectively with minimal errors or performance deviations.

Through this combination of hardware design, software integration, and metrological validation, this chapter sets the foundation for further exploration and optimization of the robot's capabilities, which will be covered in subsequent chapters. The development process, described in this chapter, is aimed at creating a versatile and reliable robotic platform that can meet the diverse demands of both surface and underwater operations.

1.1 Conceptualization and Design of a Generalized Amphibious Robot with Metrological Assurance

Here I consider the Robots of different design compositions with dissimilar Hardware and Software as well as different metrological provisions.

1.1.1 Water Strider Robot: Surface Locomotion and Limitations

The development of amphibious robots stems from the aspiration to replicate natural systems that demonstrate efficient locomotion in aquatic environments. One early design approach center on water strider robots, modeled after the Gerridae insect, known for its ability to glide across water surfaces using long, hydrophobic legs that harness surface tension [4]. These robots are lightweight, energy-efficient, and adept at navigating calm waters, laying the groundwork for further exploration into surface locomotion technologies for environmental monitoring and exploration.

A notable study, *Design of the Water Strider-like Robot*, investigates the integration of smart sensors and lightweight materials to enable these robots to glide smoothly over water without submerging . The design emphasizes creating a

low-energy, low-noise system tailored for surface-level environmental monitoring. The research highlights the application of hydrophobic and microporous materials to enhance buoyancy, ensuring that these robots can float and traverse water surfaces efficiently [5][6].

The surface locomotion of water strider robots is accomplished by exploiting surface tension, a natural phenomenon that enables the insect to remain afloat without breaching the water's surface [7]. Through precise engineering, the robot emulates this behavior by distributing its weight across elongated legs coated with hydrophobic materials, which repel water and reduce drag [8][9]. This design ensures buoyancy while promoting smooth movement across calm waters. Furthermore, the robot's construction minimizes disturbances to the water surface, enabling silent operation, which is particularly advantageous for ecological monitoring and wildlife observation in sensitive aquatic habitats [10][11][12].

Table 1.1 Worldwide universities' design of water striders; major characteristics.

Time	Institution	Movement Form	Ability to carry sensors	Drive method	Quality (g)	Linearspeed (mm/s)
2003	MIT	Sliding	No	Elastic band	0.35	180
2010	Carnegie Mellon University	Sliding	No	DC Motor	21.75	71.5
2011	Minzu University of	Sliding	Yes	DC Motor	6	200
2015	Harvard University	Jumping	No	Memory Alloy	0.068	—

2016	Zhejiang University	Sliding	Yes	Steering gear	439	90
2017	Shanghai Jiao Tong University	Sliding	No	PZT Driver	0.165	151
2017	Kogakuin University	Sliding	No	DC motor	4.39	59.2

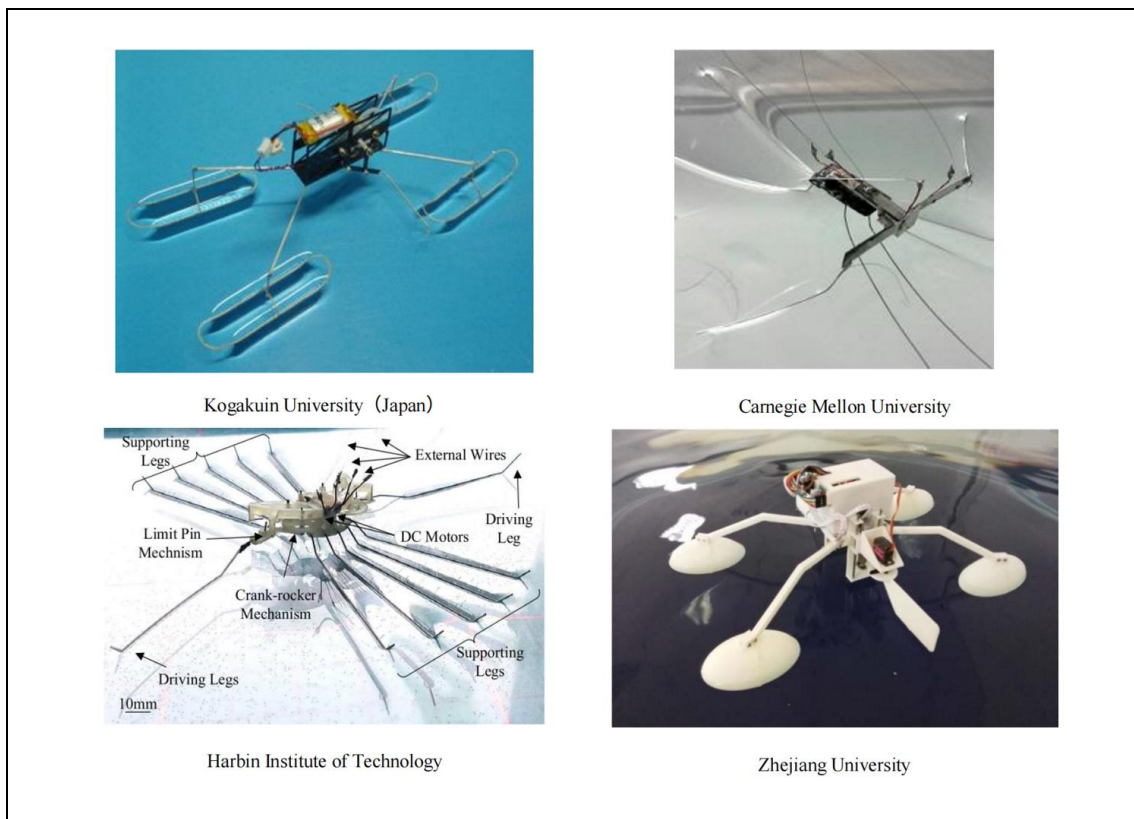


Figure 1.1 Worldwide universities’ design of water striders; major characteristics.

The propulsion system of the water strider robot typically relies on lightweight actuators that generate the necessary thrust to move the robot forward. These actuators are powered by low-energy sources, ensuring that the robot can operate for extended periods without requiring significant power. This makes the design

particularly attractive for tasks that involve long-term deployments in remote locations, where frequent recharging or maintenance would be impractical.

Despite its advantages, however, the water strider robot faces significant limitations when operating in dynamic aquatic environments [13][14]. The robot's reliance on surface tension and its lightweight design restrict its ability to navigate through turbulent waters, strong currents, or wind disturbances. Furthermore, the robot's performance is significantly hampered in scenarios where submersion is required, as its design is specifically optimized for surface locomotion. These limitations highlight the need for more versatile amphibious systems capable of functioning efficiently both on the surface and underwater, where more complex missions, such as underwater exploration or search and rescue, where stability and precise control become difficult, demand greater adaptability.

1.1.2 Amphibious Robots: Expanding Capability with Surface-Underwater Transition

While water strider robots are optimized for surface locomotion, their inherent limitations in dynamic and submerged environments have led to the development of more versatile amphibious robots. These robots are designed to perform effectively in both surface and underwater conditions, overcoming the constraints posed by their surface-only counterparts. Amphibious robots can handle a variety of aquatic environments, transitioning seamlessly from surface operations to submerged tasks, making them more adaptable for complex applications such as ocean exploration, search and rescue, and environmental monitoring.

Amphibious robots integrate a range of features that enable functionality in both terrestrial and aquatic environments, beginning with adjustable buoyancy systems that regulate depth in water. These systems are crucial for underwater

navigation, providing the robot with capabilities to hover, dive, or maintain buoyancy as required by the task. Unlike water strider robots, which are restricted to surface locomotion, amphibious robots can dynamically modify their buoyancy to accommodate operational demands, making them significantly more versatile for multi-environment missions.

The propulsion system in amphibious robots is also more advanced, integrating submersible motors and directional thrusters to achieve efficient movement underwater. These robots are typically designed with waterproof actuators that can withstand the increased pressure of deeper waters, enabling them to navigate through both calm and turbulent environments. Furthermore, these propulsion systems are optimized for both surface and underwater efficiency, ensuring that the robot can conserve energy while maximizing its range of movement.

From a metrological assurance perspective, amphibious robots offer significant improvements over surface-only designs. The ability to transition between surface and submerged states requires precise measurement systems that can function accurately in a variety of environmental conditions. Pressure sensors, for example, must be capable of adjusting to rapid changes in water depth, providing real-time feedback on the robot's position and ensuring accurate control. These sensors are calibrated to maintain their precision over extended periods of operation, even when subjected to the varying pressures of underwater exploration.

In addition, IMUs integrated into amphibious robots are calibrated for both surface and underwater dynamics, ensuring the robot maintains stability and orientation across diverse environments. Sensor fusion techniques further enhance the reliability of these measurements by integrating data from multiple sensors, including IMUs, pressure sensors, and depth sensors. This method enables the robot to

continuously update its internal model of the surroundings, facilitating real-time adjustments to its movement and control systems.

Buoyancy control also plays a key role in metrological assurance, as the robot must precisely manage its buoyancy to maintain stable depth during underwater missions. Accurate buoyancy sensors and control algorithms ensure that the robot can adjust its buoyancy in response to environmental changes, maintaining its operational integrity in both shallow and deep waters. This level of precision is particularly important in tasks that require fine-tuned depth control, such as underwater inspection or sampling missions.

In conclusion, amphibious robots represent a significant advancement in aquatic robotics, offering the flexibility and adaptability needed to operate in both surface and underwater environments. The integration of advanced sensor systems and metrological assurance techniques ensures that these robots can provide reliable and accurate performance across a wide range of tasks. By overcoming the limitations of surface-only designs like water strider robots, amphibious robots pave the way for more comprehensive and versatile solutions in aquatic robotics.

1.2 Hardware System: Nvidia Jetson and Control Architecture

The success of modern amphibious robots is not solely dependent on mechanical design; it also requires robust, real-time processing capabilities that can handle complex tasks such as sensor fusion, navigation, and image processing. For this reason, the Nvidia Jetson platform has emerged as an ideal hardware system for amphibious robots, offering high-performance processing power in a compact, energy-efficient form factor [15]. The Nvidia Jetson platform's advanced capabilities

enable real-time decision-making, deep learning inference, and the control of sophisticated algorithms necessary for autonomous operations in dynamic and unpredictable environments.

In addition to its computational strengths, metrological assurance plays a crucial role in ensuring the accuracy and reliability of the data processed by the Jetson platform. Amphibious robots must rely on a variety of sensors, such as IMUs, pressure sensors, and depth sensors, all of which require careful calibration and validation to provide accurate measurements. The Jetson platform supports the integration of these sensors and facilitates real-time sensor fusion, where data from multiple sources is combined to create a more precise understanding of the robot's environment [16].

The integration of metrological assurance techniques, such as periodic recalibration and environmental compensation, ensures that the robot can maintain the precision and reliability of its operations. This level of accuracy is especially important in applications like environmental monitoring or underwater inspections, where reliable data collection is critical for decision-making.

1.2.1 Nvidia Jetson: The Heart of Amphibious Robotics

The Nvidia Jetson platform features a GPU, which is essential for processing large volumes of data in real time. Amphibious robots often rely on data from multiple sensors, such as IMUs, pressure sensors, depth sensors, and cameras, all of which must be processed concurrently to make informed decisions regarding the robot's environment and movement. The parallel processing capability of the Nvidia GPU enables these calculations to be performed with minimal latency, ensuring that the robot can dynamically respond to changes in its surroundings [17].

Key features of the Nvidia Jetson platform that enhance its suitability for amphibious robots include:

Real-time sensor data processing: The Nvidia Jetson processes sensor data from multiple sources, integrating inputs from depth sensors, pressure sensors, and cameras to create a comprehensive view of the robot's environment.

Edge AI and deep learning: The platform is capable of running complex AI models directly on the robot, enabling real-time object detection, path planning, and adaptive behavior without needing to rely on remote servers or cloud processing. This is especially useful in underwater environments, where communication with the surface is often limited.

Energy efficiency: Nvidia Jetson boards are designed to perform high-computational tasks while consuming minimal power, an essential requirement for robots operating in remote or underwater environments for extended periods.

Compact size: The compact nature of the Jetson platform makes it ideal for integration into amphibious robots, which often have space constraints due to the need for buoyancy control systems and propulsion mechanisms.



Figure 1.2 Nvidia Jetson Orin Nano Development Board.

1.2.2 Hardware Integration and Control: Nvidia Jetson and Sensor Interfacing

In amphibious robotic systems, hardware integration is fundamental to achieving seamless operation across diverse environments. The Nvidia Jetson platform serves as the core of the control architecture, providing both the computational power for real-time decision-making and the flexibility required for integrating a wide range of sensors and actuators. To support essential functions such as navigation, depth control, and sensor fusion, the Nvidia Jetson platform incorporates multiple I/O interfaces and pinouts that facilitate communication with peripheral devices, including sensors, motors, and external controllers [18].

Table 1.2: Jetson Orin Nano Interface, Purpose, and Example Sensors/Devices.

Interface	Purpose	Example Sensors/Devices
GPIO	General input/output for basic sensors like limit switches, temperature, or proximity sensors	Temperature sensors, proximity switches
I2C	Communication with IMUs, depth sensors, providing timing-based communication for precision	BNO055 IMU, depth sensors
SPI	High-speed communication with pressure transducers and other precision sensors	Pressure transducers, temperature probes
UART	Communication with external microcontrollers or GPS modules for navigation	UBlox GPS module, serial communication devices
USB	Connection to external devices like cameras, LIDAR, for image processing and real-time feedback	USB cameras, LIDAR sensors
PWM	Motor control for DC and servo motors for propulsion, buoyancy adjustments, and stabilization	DC motors, servo motors

Jetson's I/O Capabilities and Sensor Integration

The Nvidia Jetson platform provides several interfaces that are crucial for connecting the various sensors required for the operation of an amphibious robot. The Jetson Xavier NX, for example, features an extensive set of GPIO (General-Purpose Input/Output) pins, along with I2C, SPI, UART, and USB interfaces, enabling seamless integration with a wide range of sensors and actuators.

Key hardware interfaces and their associated sensor functions include:

GPIO Pins: These general-purpose pins can be configured for input or output, facilitating basic communication with sensors like limit switches, temperature sensors, or proximity sensors. GPIO pins can also trigger specific actions, such as controlling relays or managing external power sources.

I2C (Inter-Integrated Circuit): This protocol connects IMUs, depth sensors, and other peripherals requiring precise timing. For instance, integrating a BNO055 IMU sensor on an I2C bus enables the robot to measure its orientation, providing critical feedback for maintaining stability in both surface and underwater environments.

SPI (Serial Peripheral Interface): This protocol supports high-speed communication with sensors like pressure transducers and temperature probes. Leveraging SPI for pressure sensors ensures quick and accurate data transmission, essential for real-time depth control during underwater missions.

UART (Universal Asynchronous Receiver-Transmitter): This interface is frequently employed for communication with external microcontrollers or GPS modules. For example, a UBlox GPS module can be linked via UART to deliver positional data during surface-level operations. Although GPS signals are unavailable underwater, the UART interface remains crucial for surface operations and hybrid navigation solutions.

USB Ports: The Jetson platform's USB ports facilitate the connection of external devices like cameras and LIDAR sensors. Cameras connected via USB are essential for image processing tasks, including object detection and obstacle avoidance. USB connectivity is particularly beneficial for real-time visual feedback and executing AI-driven image recognition algorithms.

Sensor Integration and Metrological Considerations

The Nvidia Jetson platform's ability to interface with a variety of sensors through its hardware pinouts directly enhances its suitability for amphibious robots. Sensors such as pressure transducers, IMUs, and cameras provide critical data needed for the robot to navigate and maintain operational efficiency in dynamic environments. However, the precision and reliability of these sensors depend on metrological assurance—the practice of ensuring that sensors are calibrated and validated for accurate measurement.

Pressure Sensors: These sensors, often connected via SPI, must be calibrated to measure depth accurately. Changes in water pressure can significantly affect the sensor's readings, making it essential to calibrate these sensors at regular intervals, especially in deep-sea missions.

IMU: Connected via I2C, provide orientation data but are susceptible to sensor drift over time. Periodic calibration is essential to maintain accurate orientation, particularly in underwater environments where GPS signals are unavailable. The Nvidia Jetson's processing capabilities enable sensor fusion of IMU data with other sensors, such as gyroscopes and magnetometers, enhancing the reliability of positional tracking..

Actuation and Control

In addition to sensor integration, the Nvidia Jetson platform supports motor control for propulsion systems through its PWM outputs. These outputs can drive DC motors or servo motors that control the robot's movement on the surface and underwater. The PWM outputs provide precise control over motor speed and direction, enabling the robot to adjust propulsion based on real-time sensor feedback.

For amphibious robots, the Jetson's GPIO and PWM pins also serve to control actuators that adjust buoyancy and stabilization mechanisms. These control signals, processed in real time, enable the robot to dynamically modify its position in the water, facilitating effective navigation through varying currents and depths.

1.3 Integration of ROS for Efficient Communication

The ROS is an essential component in the control architecture of amphibious robots, providing a flexible and scalable framework for integrating various hardware and software components. ROS serves as the communication backbone that manages the flow of data between sensors, actuators, and the robot's control systems, enabling real-time decision-making and adaptability in both surface and underwater environments.

By leveraging ROS, developers can build modular and efficient robotic systems that facilitate easy integration of additional sensors, enhanced functionality, and future scalability. In amphibious robots, ROS manages sensor data fusion, navigation control, motion planning, and state estimation—all of which are crucial for ensuring autonomous operation in dynamic environments.

1.3.1 Modular Architecture and Scalability

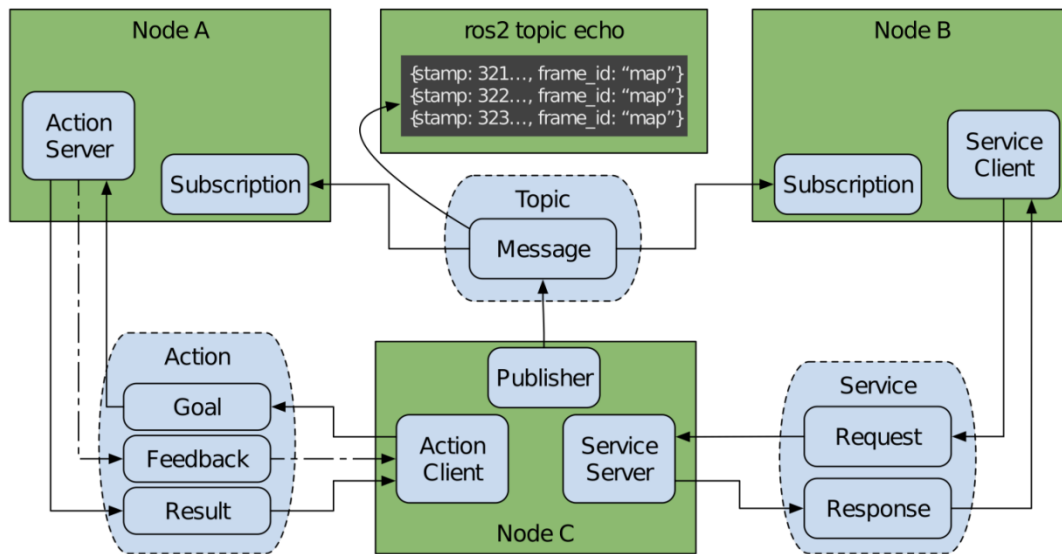


Figure 1.3 Overview of ROS Communication Framework.

One of the key strengths of ROS lies in its modular architecture, which enables individual subsystems to be developed and operated independently as ROS nodes. Each node is responsible for a specific function, such as sensor data processing, motion control, or communication with actuators. These nodes communicate with each other through a publish-subscribe model, which is a core feature of ROS.

The modular structure of ROS is well-represented by Node A, B, and C, as shown in the Figure. Each node handles a specific function such as action servers and service clients, with nodes communicating via topics and messages. This publish-subscribe model enables various parts of the system to function independently yet cohesively.

In the context of an amphibious robot, for example:

Node A could be responsible for managing feedback from an IMU sensor and publishing data about the robot's orientation.

Node B could handle depth sensor readings and provide real-time depth control.

Node C might be responsible for controlling propulsion and buoyancy based on the sensor data from Nodes A and B.

This modular approach ensures that individual components can be updated or replaced without affecting the entire system. It also simplifies the process of integrating new sensors or actuators as the robot evolves to meet specific mission requirements. The scalability of ROS is particularly important for amphibious robots that may need to operate in increasingly complex environments with additional sensors and enhanced control algorithms.

1.3.2 Real-Time Sensor Data Fusion

Amphibious robots rely heavily on sensor data for navigation, obstacle avoidance, and maintaining stability in aquatic environments. These sensors include IMUs, depth sensors, cameras, pressure transducers, and GPS modules (for surface operations). However, the accuracy and reliability of each sensor can vary depending on environmental factors. For instance, GPS signals may be lost underwater, or pressure sensors may drift at greater depths.

ROS facilitates sensor fusion, a process in which data from multiple sensors is combined to improve the accuracy and robustness of the robot's situational awareness. For example:

Data from an IMU can be fused with depth measurements from a pressure sensor to provide more reliable state estimation, particularly in environments where GPS signals are unavailable.

Camera data can be combined with LIDAR for obstacle detection and terrain mapping, providing a more comprehensive understanding of the surroundings.

By fusing sensor data in real time, ROS helps to compensate for individual sensor limitations, ensuring that the robot can navigate and operate effectively in both surface and underwater environments.

1.3.3 Communication Between Surface and Submerged Systems

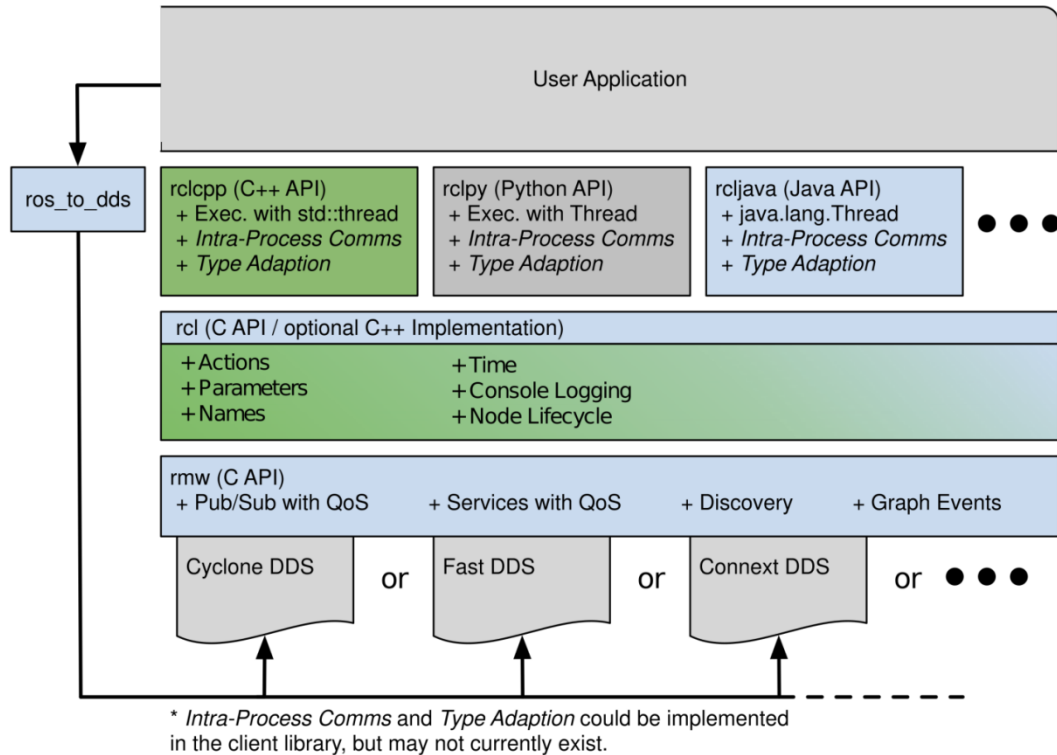


Figure 1.4 ROS Architecture and Middleware Layer.

The Figure delves deeper into ROS’s middleware architecture, showcasing its flexibility in working across different communication layers. The rclcpp, rclpy, and rcljava APIs provide the necessary tools to implement ROS communication for C++, Python, and Java. These API layers facilitate interaction between the hardware (sensors, actuators) and software (control algorithms) of amphibious robots, ensuring real-time decision-making across various operating conditions.

One key aspect involves employing Cyclone DDS, Fast DDS, or Connex DDS as communication backbones for Pub/Sub messaging with QoS. This is crucial for maintaining communication between the robot’s surface and submerged systems,

particularly during transitions between Wi-Fi communication on the surface and acoustic communication underwater.

The `rclcpp` (C++ API) or `rclpy` (Python API) facilitates the implementation of sensor data collection and fusion processes. The ROS-to-DDS bridge enables this data to be shared across different ROS nodes, ensuring real-time sensor feedback is available to all components responsible for controlling the robot's operations.

The Node Lifecycle management, as shown in the second diagram, is particularly important in amphibious robots for managing tasks like power management, sensor calibration, and depth control, which are vital for operations that involve both surface-level tasks and deep-sea navigation.

1.4 Integration of Hardware and Software for Real-Time Control in Amphibious Robots

In previous sections, we discussed the design principles, metrological assurance, and the central role of the Nvidia Jetson platform and ROS in amphibious robots. In this section, we expand on the practical integration of these systems, focusing on how the hardware and software work in unison to provide real-time control for surface and underwater operations. This includes detailing the key components, interfaces, and control architectures, as shown in Figures 1 and 2. The diagrams illustrate the interconnections between various sensors, actuators, communication systems, and control platforms, highlighting the crucial role of both the Jetson Orin Nano and Pixhawk platforms in enabling efficient and adaptable robotic operations.

1.4.1 Jetson Orin Nano as the Central Processing Unit

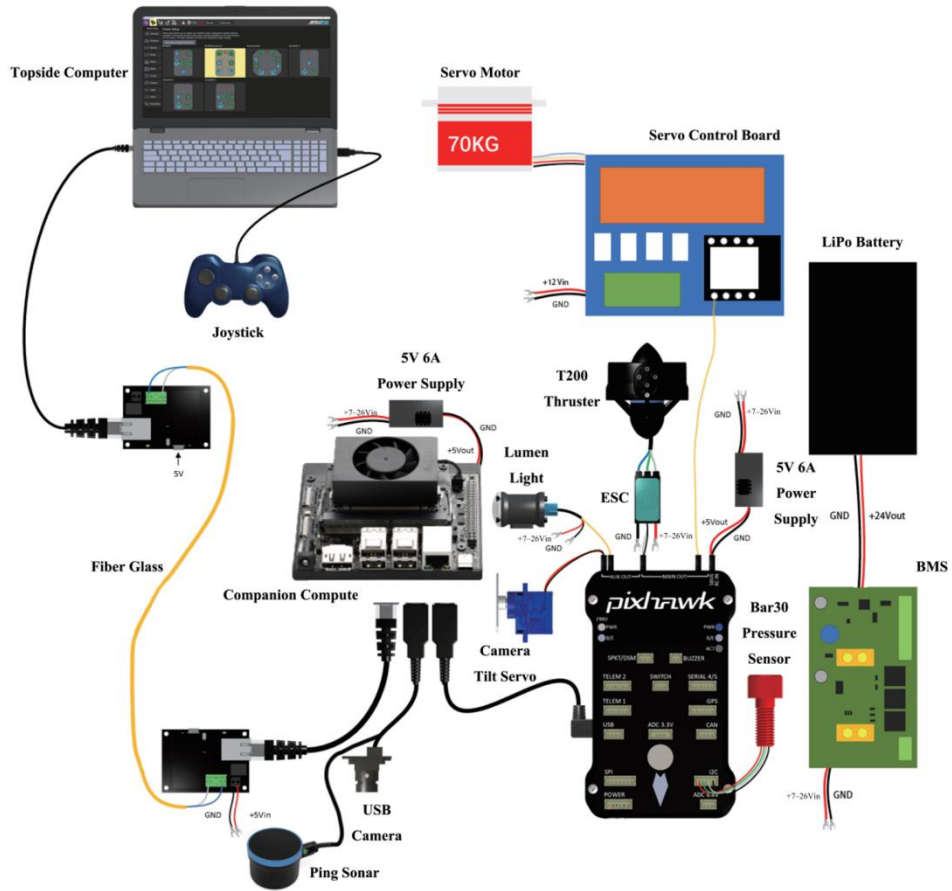


Figure 1.5 Hardware connection diagram.

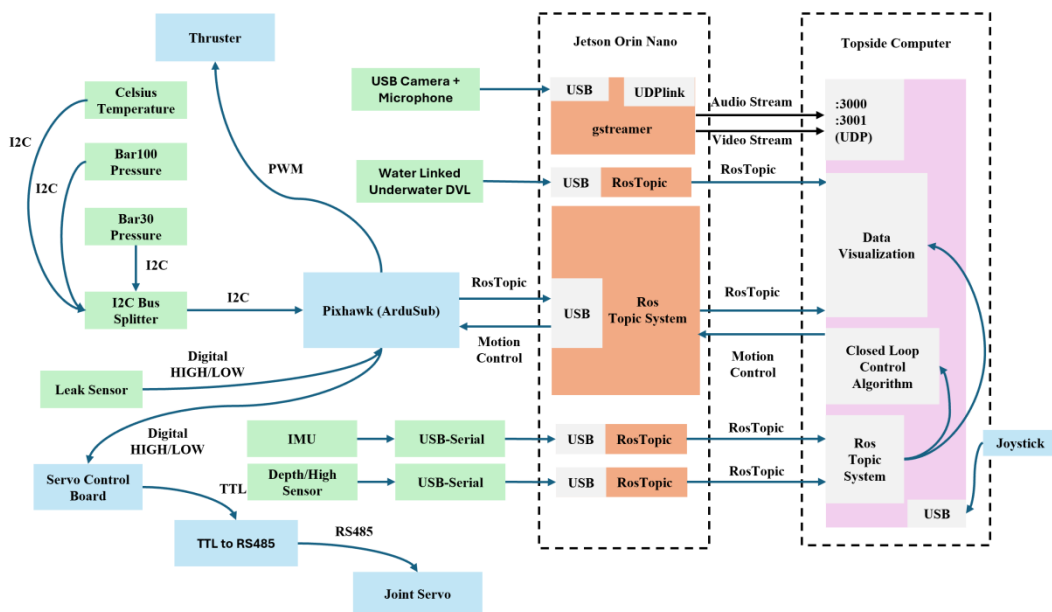


Figure 1.6 Software protocol diagram.

The Jetson Orin Nano, as shown in Figure 1.5, acts as the central computing unit in the amphibious robot, interfacing with the topside computer and controlling the robot's motion based on sensor data inputs. The Orin Nano is connected to critical sensors such as the IMU, pressure sensors (Bar30 and Bar100), and Water Linked Underwater DVL. These sensors provide real-time feedback on the robot's orientation, depth, and underwater velocity. This data is then processed by the Jetson Orin Nano using the ROS framework, which enables efficient sensor fusion and decision-making.

Through its USB interfaces, the Jetson Orin Nano connects to peripherals such as USB cameras and microphones, providing essential data for surface-level and underwater object detection. This data enables the robot to dynamically adjust its movements and perform complex tasks such as navigation, obstacle avoidance, and real-time video streaming. The integration of GStreamer with ROS facilitates the transmission of video and audio streams to the topside computer for further analysis and monitoring.

The application of PWM signals for thruster control and TTL to RS485 communication for the joint servo ensures precise and adaptive movements of the robot both on the water surface and underwater. The Jetson's processing capabilities, combined with its I/O flexibility, enable the robot to manage complex tasks efficiently while conserving energy, which is crucial for extended missions in remote or underwater environments.

1.4.2 Pixhawk as the Core Control Module for Underwater Operations

As depicted in Figures 1 and 2, the Pixhawk (ArduSub) serves as the core control module for underwater operations, managing the communication between various sensors and actuators. It interfaces with sensors such as the Bar100 and Bar30

pressure sensors, leak sensor, and depth sensor, providing vital data on the robot's environment. The I2C bus splitter enables the Pixhawk to handle multiple sensors concurrently, ensuring reliable data acquisition without communication bottlenecks.

Pixhawk communicates with the Jetson Orin Nano via ROS topics, transmitting real-time motion control data and receiving sensor feedback to adjust the robot's propulsion and stabilization systems. Additionally, it manages the T200 thruster, servo control board, and joint servo through PWM and RS485 signals. This hardware integration enables the robot to smoothly transition between surface locomotion and underwater operations, providing enhanced control over buoyancy and movement.

1.4.3 Topside Computer and Closed-Loop Control

The topside computer plays a crucial role in monitoring and controlling the robot's operations, as depicted in both diagrams. Connected via UDP (ports :3000 and :3001), the topside computer receives sensor data streams from the Jetson Orin Nano, enabling real-time data visualization and analysis. A closed-loop control algorithm ensures that sensor feedback is promptly processed to adjust the robot's trajectory, propulsion, and buoyancy.

The joystick interface provides manual control when needed, enabling an operator to intervene and guide the robot, particularly during delicate operations such as underwater inspection or surface monitoring. Through ROS, the topside computer sends motion control commands to the Jetson Orin Nano, ensuring that the robot can respond to environmental changes in real time, even when operating at a distance or under challenging conditions.

1.4.4 Communication and Sensor Fusion

One of the most crucial aspects of an amphibious robot is its ability to seamlessly integrate data from multiple sensors to make informed decisions. In this system, ROS topics facilitate communication between the Jetson Orin Nano and the Pixhawk, as well as between the robot and the topside computer. This modular architecture ensures that sensor data can be shared in real time across the various subsystems.

As depicted in both figures, the integration of sensors such as IMUs, depth sensors, pressure sensors, and cameras enables continuous updates on the robot's surroundings. These sensors work together to provide a comprehensive view of the environment, with each sensor compensating for the limitations of the others. For instance, the IMU provides orientation data, while the pressure sensors supply depth information. The Jetson Orin Nano's GPU processes this data using real-time sensor fusion techniques, ensuring that the robot can navigate smoothly through both surface and underwater environments.

This architecture also supports the transition between surface and submerged operations, with the communication system automatically adjusting between RF-based protocols (for surface-level operations) and underwater communication methods (such as acoustic modems). ROS ensures that communication between the topside computer, the Jetson Orin Nano, and the Pixhawk remains uninterrupted, even when switching between different environments.

1.4.5 Real-Time Autonomous Operation

Finally, the integration of all these systems enables the amphibious robot to operate autonomously in real time. The Nvidia Jetson's edge AI capabilities empower the robot to process complex tasks such as path planning and obstacle avoidance

without relying on remote servers. By running AI models directly on the robot, it can make decisions in real time, adjusting its trajectory based on sensor feedback and environmental conditions.

With ROS as the backbone of the communication system, the robot can autonomously perform missions such as underwater exploration, search and rescue, and environmental monitoring. The ability to run deep learning models for object detection and dynamic control further enhances the robot's versatility, enabling it to adapt to changing environments without human intervention.

Conclusions to Chapter 1

The current chapter considers possible cases of all types of security in the design, manufacture and operation of drones. However, the novelty of setting and solving goals for underwater types of drones highlighted the pioneering spirit of solving such problems. The process begins with planning and design, followed by integrating appropriate sensors and conducting diagnostic checks, and concludes with providing metrological support during assembly.

Therefore, we have illustrated how the hardware and software systems discussed in the following sections come together to provide real-time control for amphibious robots. The combination of Nvidia Jetson, Pixhawk, ROS, and advanced sensor fusion techniques ensures that these robots can operate efficiently in both surface and underwater environments, meeting the diverse demands of modern aquatic robotics. This integrated approach sets the foundation for the further development of autonomous, adaptable, and resilient robotic systems capable of addressing complex challenges in aquatic environments.

Chapter 2

Building a Test Platform for Hydrodynamic Performance

The successful development and deployment of amphibious robots rely heavily on understanding and optimizing their hydrodynamic performance. Building an underwater test platform is crucial for evaluating and refining these robots' capabilities, particularly when operating in complex aquatic environments. Unlike surface environments, where variables such as air resistance and surface tension dominate, underwater environments are governed by fluid dynamics, which introduce significant challenges such as drag, buoyancy, thrust generation, and pressure effects at various depths.

Establishing a controlled and well-equipped underwater testing platform enables researchers to systematically study these factors under real-world conditions, ensuring that the robot performs efficiently across a variety of submerged scenarios. Such a platform is instrumental in assessing the robot's propulsion efficiency, stability, and energy consumption, as well as its ability to respond to different water flow rates, pressure gradients, and depth-related variables. By collecting detailed hydrodynamic data in a simulated underwater environment, engineers can fine-tune the robot's design to enhance its performance in real-world applications such as ocean exploration, search and rescue operations, and environmental monitoring.

In underwater robotics, precise hydrodynamic testing is essential for refining not only the mechanical design but also the control algorithms responsible for navigation and maneuverability. These systems must work in harmony to ensure that the robot can maintain stable and efficient movement despite the unpredictable and often hostile nature of underwater environments. Therefore, a robust underwater test

platform serves as the foundation for achieving the metrological accuracy needed to optimize robotic performance, making it an indispensable tool in the development of advanced aquatic robots.

In this chapter, we will explore the design, implementation, and key metrics of an underwater test platform, focusing on the sensor integration, data collection techniques, and hydrodynamic parameters critical to evaluating an amphibious robot's performance. By establishing a comprehensive testing environment, this platform will provide the necessary infrastructure for continuous iteration and improvement of the robot's capabilities.

2.1 Propulsion Testing: Precision Hydrodynamic Measurement and Calibration

2.1.1 Importance of Force Measurement in Hydrodynamics

In underwater robotics, precise measurement of hydrodynamic forces is crucial for assessing the performance of propulsion systems, such as thrusters. Understanding forces like thrust, drag, and torque is essential for optimizing the propulsion system to improve energy efficiency, stability, and movement accuracy in submerged environments [19]. To address these needs, a single-device test platform is developed to isolate and evaluate an individual propulsion system under controlled underwater conditions [20][21].

As highlighted in the study "Design of the Water Strider-like Robot" (uploaded document), the testing of propulsion systems often encounters non-linearities, including torque-induced oscillations and instabilities, particularly when dealing with dynamic thrusters. This test platform is designed to mitigate these issues by focusing on a single propulsion system at a time, enabling researchers to gather high-resolution

data on the thrust-to-power ratio, motor efficiency, and dynamic response of the propeller under various operational conditions.

General Description of a Thruster

A propeller is a device that generates thrust for underwater vehicles (Figure 2.1). It typically consists of a rotating screw-like blade that pushes water backward to produce propulsion. The direction of thrust can be adjusted by reversing the propeller's rotation. The performance of a propeller is crucial for controlling and maneuvering underwater vehicles and can be characterized by various parameters, including thrust, propeller speed, and output flow velocity. The thrust generated by a propeller is directly proportional to the square of its speed, while the output flow velocity depends on thrust, speed, and propeller efficiency [22].

To accurately describe the dynamic characteristics of a propeller, researchers have developed a physical system model based on force and torque feedback to represent the propeller's thrust. This model employs the propeller's angular velocity as the dynamic state variable and controls the propeller's motion through input torque. In this experiment, propellers manufactured by ROVMAKER were selected, and output current was controlled using PWM to regulate the propeller's output torque, thereby generating thrust underwater.

In PWM control, the pulse width range is 1000-2000 microseconds, with 1500 corresponding to the motor's midpoint. In other words, when PWM outputs 1500 microseconds, the motor remains stationary. As the control signal increases linearly from 1500 to 2000 microseconds, the motor rotates forward, and the speed linearly increases. Conversely, during the decreasing process from 1500 to 1000 microseconds, the motor reverses, increasing its speed [23].



Figure 2.1. Using propellers manufactured by ROVMAKER, voltage supply range: 3s-6s, maximum passing current 15A, waterproof to a depth of 300 meters under water surface

Lumped Parameter Model Development

A standard thruster configuration, illustrated in Figure 1, comprises a stationary shroud and a propeller propelled by a torque-generating mechanism (T) operating at angular velocity (ω). The thruster's shroud possesses a cross-sectional area (A) and encloses a volume (V). The surrounding fluid has a density (ρ) and a volumetric flow rate passing through the thruster (Q) [24].

The model development is simplified by the following assumptions (Figure 2.2):

1. Negligible kinetic energy of the external fluid environment.
2. Negligible friction losses in the motor and propeller blades.
3. Incompressibility of the ambient fluid.
4. Maintaining parallel flow direction at the inlet and outlet of the thruster, disregarding rotational flow effects [25].

A state function of the volumetric flowrate Q can express the kinetic co-energy T^* of the fluid in the thruster:

$$T^*(Q) = \frac{1}{2} \rho V \left[\frac{Q}{A} \right]^2 \quad (2.1)$$

Defining a generalized momentum as:

$$\Gamma = \frac{dT^*}{dQ} = \rho V \frac{Q}{A^2} \cdot (1) \quad (2.2)$$

The above relation is that of inertia (momentum related by a static constitutive law to the flow in bond-graph

nomenclature with the effort variable Γ and flow variable Q .

Γ has units of momentum/area and is referred to as the pressure momentum.

Since the energy relations are linear, the

Co-energy and energy have equal magnitudes, and the kinetic energy T can be expressed as a state function of the pressure momentum Γ .

$$T(\Gamma) = \frac{A^2}{2\rho V} \Gamma^2 \quad (2.3)$$

The pressure momentum relation that follows from a power balance is as follows:

$$\frac{dT}{dt} = \frac{A^2}{\rho V} \Gamma \dot{\Gamma} = \Omega \tau - KQ(2) \quad (2.4)$$

The power input from the thruster propeller is represented by $\Omega\tau$, the outgoing kinetic energy per volume is represented by K , and the time rate of change of the pressure momentum is shown by $\dot{\Gamma}$.

It is possible to represent the departing kinetic energy per volume K as

$$K = \frac{A^2\Gamma^2}{2\rho V^2} = \frac{\gamma^2}{2\rho} \quad (2.5)$$

Where the $\gamma \equiv A\Gamma/V$ is the thruster's fluid momentum per volume.

The convected linear momentum, which is equal to the thrust created, connects the thruster and surrounding fluid:

$$\text{Thrust} = \gamma Q. (3) \quad (2.6)$$

The thruster/propeller characteristics and angular velocity Ω can be linked to the volumetric flowrate, provided that the propeller does not cavitate. Slip refers to the discrepancy between a propeller's theoretical and actual advance per revolution. It is commonly stated as a ratio σ as follows:

$$\sigma = \frac{\Omega p A - Q}{\Omega p A} \quad (2.7)$$

where p , also known as the pitch, is the axial distance the propeller blades move for every unit of revolution (1 rad),

The equation above indicates that Q (Ω) can be expressed as:

$$Q = \eta p A \Omega (4) \quad (2.8)$$

Where $\eta \equiv 1 - \sigma$ is referred to as the propeller efficiency.

From (1)-(4), the following thruster dynamic state and output equations are formed:

$$\dot{\Gamma} = \frac{\tau}{\eta p A} - K \tag{2.9}$$

$$\text{Thrust} = \gamma Q. \tag{2.10}$$

The propeller angular velocity R can serve as the thruster dynamic state variable in the thruster's dynamic state and output equations, assuming that the propeller efficiency (η), pitch (p), and duct area (A) are constants.

$$\dot{\Omega} = \frac{\tau}{\eta^2 p^2 \rho V} - \frac{\eta p A}{2V} \Omega |\Omega| \tag{2.11}$$

$$\text{Thrust} = A \rho \eta^2 p^2 \Omega |\Omega|. \tag{2.12}$$

Keep in mind that, as was previously stated, the steady state thrust force is proportionate to the input torque.

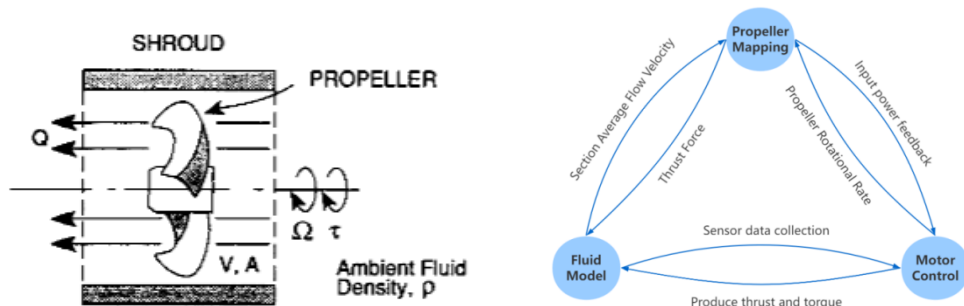


Figure 2.2. Major Elements of the Model

Thruster validation (response rate, thruster thrust, torque produced, phase lag fit)

The model was validated using a thruster mounted in a rack in an aluminum-type table that instrumented the device to measure the output force and torque. As shown in the Figure 2.3, the thrust was measured by a six-component force transducer using a D6045A sensor from DMI (Digitalize Miniaturize Inteligentized), Inc. in a matrix decoupling technique in order to decompose the output signal of the six-axis force transducer into its force and torque components in different directions and complete the recording [26]. A series of static tests were carried out to confirm the previously proposed model and to determine specific parameter values a , 0 , and C_t . Based on the available measurement data, it was possible to confirm that these parameters were reasonable for the physical parameters of propeller efficiency and volume involved [27].

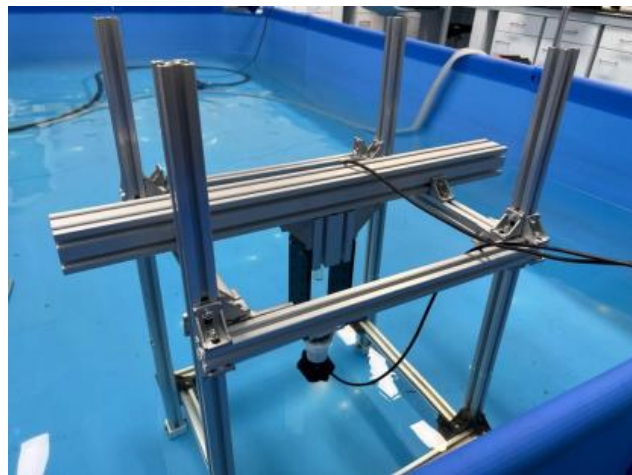


Figure 2.3. Direct coupling relationship between thrusters and sensors

Thruster Test

The thruster was tested in several dynamic thrust measurement experiments. These tests included a series of current-commanded step input signals covering a

broad range of input levels, as well as a long-period sinusoidal waveform input signal [28].

Measured variables include motor input current, voltage, and current instructions as well as motor speed and net thrust. These measurements are all dynamic and are all collected at a sample rate of 50 Hz [29][30].

Long Period Triangular Wave Inputs

We decided to output long-period sine wave signals for the measurement of thrust generation before and after in order to produce robust output results. The results are displayed in Figure an as a function of motor speed over a 60-second period for the thrust produced in the X-axis and the torque produced in the Y-axis. The inputs for current and voltage with varying speeds are shown in Figure 2.4. The speed versus force generated for the steady state instance is shown in Figure 2.5 This number is consistent with the idea that predicted thrust is inversely proportional to propeller speed. Due of the strong link between and current inputs, the current/thrust behavior is best described by the square law relationship.

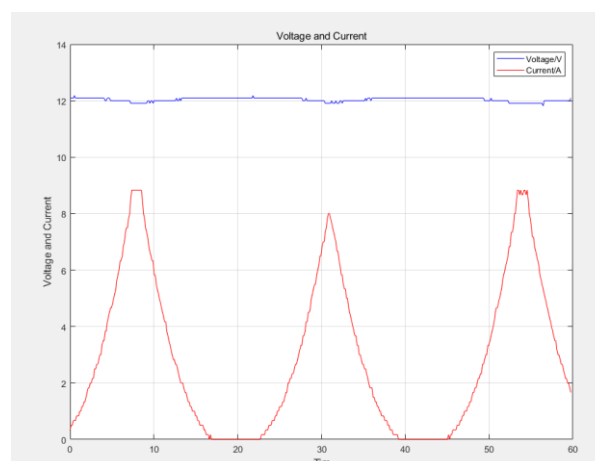


Figure 2.4. The rotational speed changes in a sinusoidal waveform and outputs the corresponding current and voltage input signals.

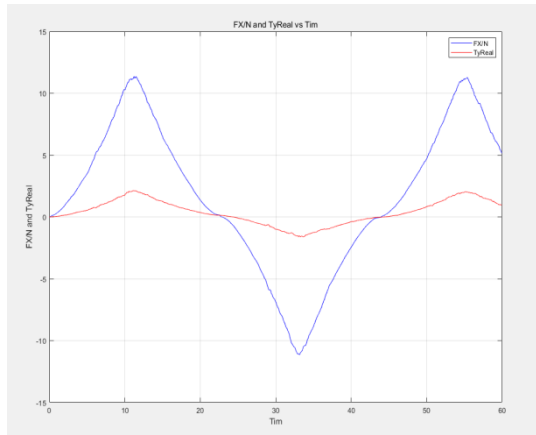


Figure 2.5 The thrust generated by the thruster in the direction of the sensor's X-axis and the moment generated by the Y

Step Input Effects of Amplitude.

In this experiment, step signals of varying amplitudes were put up to regulate the motor's current and, consequently, its speed. The influence of rotational speed from 0 to the force corresponding to the steady state reached after the step control signal was recorded using eight different sets of current inputs with the same time step setting. It was established that the thruster's force output's steady state value depends on its current value.

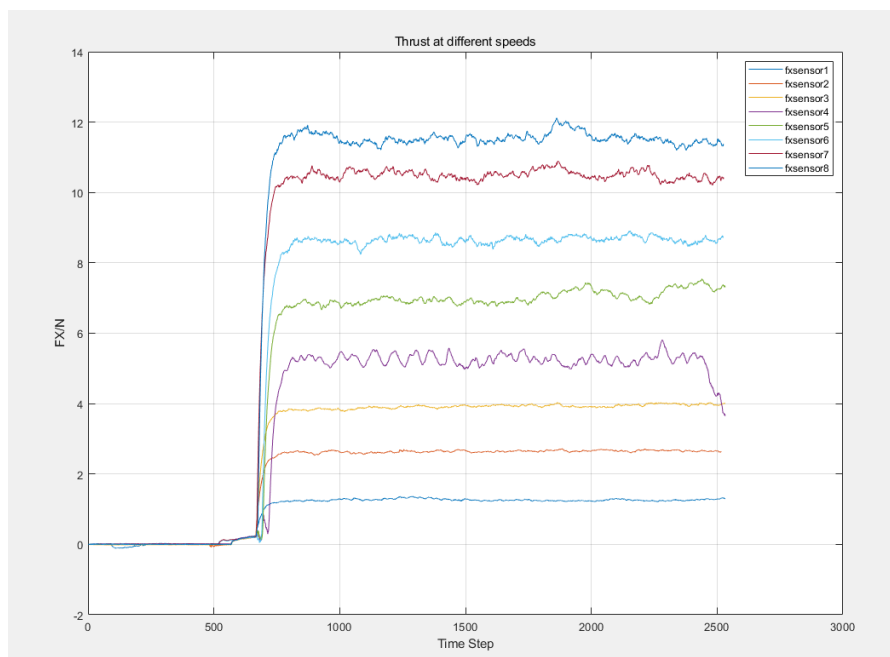


Figure 2.6 Different rotational speeds corresponding to the force generated

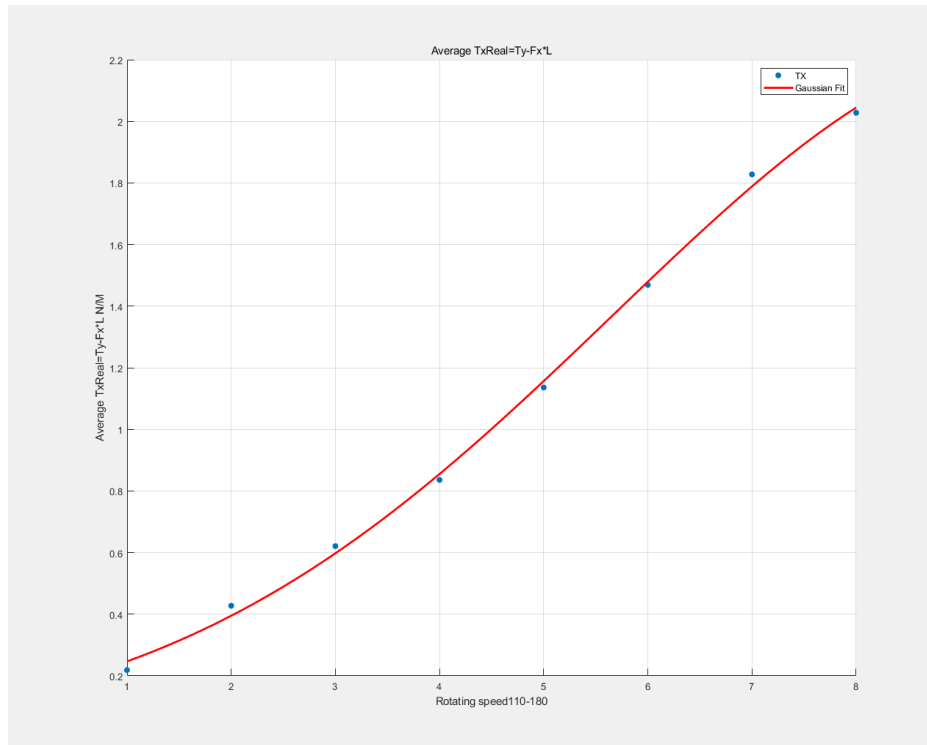


Figure 2.7 Linear relationship with rotational speed exhibited by the mean values of the 8 force groups under Gaussian regression

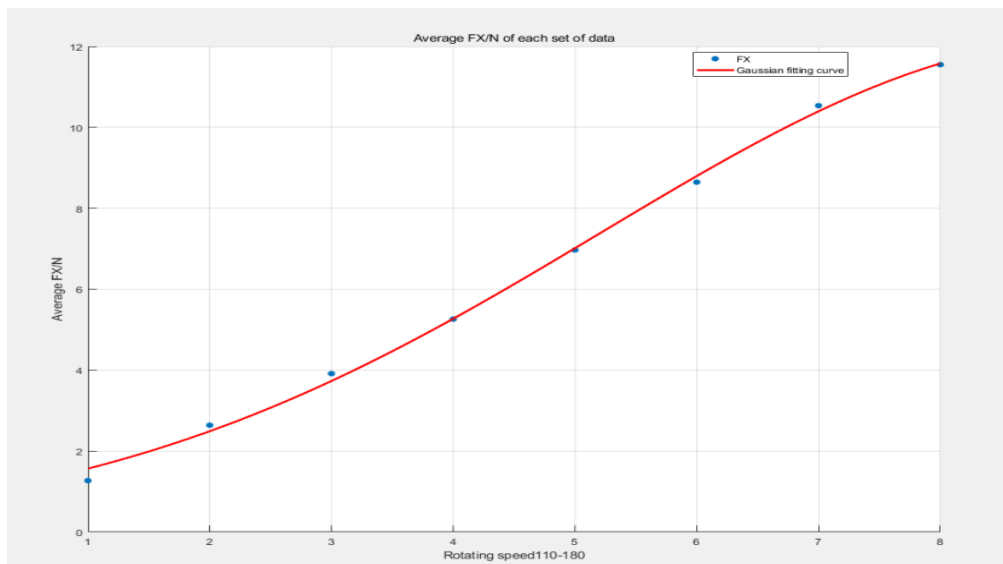


Figure 2.8 Linear relationship with rotational speed exhibited by the mean values of 8 Lets of moments under Gaussian regression

In 2.1, we focused on the isolated testing of single propulsion units to obtain accurate data on thrust, torque, and energy efficiency. However, as the complexity of the robot's movements and interactions with its aquatic environment increases, the need for a more comprehensive, data-driven approach becomes essential. A single propulsion system provides limited insights, and real-world performance requires the integration of multiple systems operating simultaneously.

The key to enhancing the efficiency and accuracy of hydrodynamic measurements lies in a multi-device platform that facilitates parallel data collection from different components of the robot. This data-driven methodology enables the simultaneous capture of a broad spectrum of movement profiles, reducing the risks of data singularity and measurement biases. By exploring various motion configurations concurrently, the system can rapidly identify the most effective combinations for achieving optimal propulsion, stability, and energy efficiency.

Moreover, this multi-device test platform emphasizes the importance of metrological assurance by automating the collection of multiple datasets in real-time. The parallel nature of this setup not only increases the speed of data collection but also improves the accuracy of measurements by minimizing human intervention and errors, providing more reliable and repeatable results. By collecting a wide array of data points under different operating conditions, this platform enables more robust statistical analysis and reduces the likelihood of overfitting to a specific set of motion parameters.

In the next section (2.2), we explore the integration of foil kinematics into this multi-device setup. By utilizing this parallel data-driven approach, we can evaluate different kinematic configurations and propulsion strategies, optimizing the robot's performance while ensuring that metrological standards are upheld. This method

enhances not only the speed of data collection but also the precision and diversity of the dataset, facilitating more effective performance tuning and error minimization across various aquatic environments.

2.2 Exploring Alternative Underwater Propulsion Systems: The Approach to Hydrodynamic Optimization

2.2.1 Foil Kinematics in Underwater Robotics

In the following section of this chapter, we explore the foil kinematics of underwater robots, a critical factor that influences the hydrodynamic performance and overall efficiency of the propulsion system. Foils, or flapping fins, are increasingly employed in underwater robotics to replicate the propulsion mechanisms of aquatic animals such as fish and manta rays [31]. These biomimetic systems rely on a combination of oscillatory motion and foil flexibility to generate thrust, minimize drag, and enhance maneuverability [32].

The study of foil kinematics is essential for understanding how different movement patterns, amplitudes, and frequencies of the fins contribute to propulsion. Specifically, the goal is to optimize the hydrodynamic parameters to ensure that the robot can achieve maximum efficiency and stability while operating in complex aquatic environments. We explore the kinematic equations, principles, and analytical methods that guide the development of efficient foil-based propulsion systems [33].

Kinematic Parameters of Foil Motion

The flapping motion of a foil can be described using kinematic parameters such as frequency, amplitude, pitch angle, and phase difference between pitch and heave movements [34]. These parameters dictate how the foil interacts with the surrounding

fluid to generate lift and thrust forces. The basic equation describing the sinusoidal motion of a flapping foil is [35]:

$$\theta(t) = \theta_0 \sin(2\pi ft + \alpha) \quad (2.13)$$

Where:

$\theta(t)$ represents the instantaneous angle of the foil at time t ,

θ_0 is the maximum pitch angle,

f is the flapping frequency (measured in Hz),

α is the phase angle between pitch and heave motions.

In underwater robots, the pitch angle and heave amplitude are carefully adjusted to ensure that the foil produces the necessary thrust while minimizing drag. The phase difference between the two motions α is a critical parameter, as it directly influences the generation of forward thrust and the efficiency of the system. Typically, a phase difference of around 90 degrees has been shown to optimize thrust production in foil-based propulsion systems [36][37].

Foil Kinematics and Hydrodynamic Forces

To understand the forces acting on a flapping foil, the Strouhal number (St), Reynolds number (Re), and reduced frequency (Kf) are critical dimensionless parameters that provide insights into the flow behavior around the foil and its interaction with the water.

Strouhal number (St): This number governs the oscillatory motion and is defined as:

$$St = \frac{fA}{U} \quad (2.14)$$

Where:

f is the flapping frequency,

A is the amplitude of the heave motion,

U is the forward velocity of the robot.

Research suggests that maintaining a Strouhal number between 0.2 and 0.4 is optimal for achieving efficient thrust in biomimetic systems, as this range maximizes the propulsive efficiency while minimizing energy consumption.

Reynolds number (Re): The Reynolds number is a measure of the ratio of inertial forces to viscous forces and is calculated as:

$$Re = \frac{\rho U c}{\mu} \quad (2.15)$$

Where:

ρ is the fluid density,

U is the foil velocity,

c is the chord length of the foil,

μ is the dynamic viscosity of the fluid.

The Reynolds number helps determine whether the flow around the foil is laminar or turbulent, which in turn affects the hydrodynamic forces acting on the foil.

Reduced frequency (Kf): This parameter indicates how efficiently the foil generates thrust through oscillations and is given by:

$$Kf = \frac{\omega c}{2U}, \quad \omega = 2\pi f \quad (2.16)$$

A higher reduced frequency typically corresponds to better thrust generation, though it also increases the drag experienced by the foil. The optimization of Kf is critical in designing foil-based propulsion systems for underwater robots [38][39].

2.2.2 Foil Kinematics: Optimizing Biomimetic Propulsion for Hydrodynamic Efficiency

In underwater robotics, foil kinematics plays a crucial role in determining the hydrodynamic efficiency of the robot's propulsion system. Flapping foils, which mimic the motion of aquatic animals, rely on the interplay between pitch and roll motions to generate thrust and maneuverability. The following section provides an analysis of the kinematics behind these movements, introducing key equations that describe the position and orientation of the foil in three-dimensional space [40].

The kinematic behavior of the foil is detailed in Figure 2.9 of the referenced study, where the foil undergoes both pitch and roll motions. The motion is broken down into two stages: first, observing the roll motion from the XoZ plane, and then analyzing the pitch motion from the YoZ plane [41].

Roll Motion Analysis

In the first stage, the roll motion is observed in the XoZ plane, where the position of the foil can be described by the following kinematic equations:

$$\begin{cases} x_1 = (b + l)\cos\phi(t) \\ y_1 = 0 \\ z_1 = (b + l)\sin\phi(t) \end{cases} \quad (2.17)$$

Where:

b is the distance between the axis of the roll motion and the chord of the flapping wing,

l is the span of the flapping wing,

$\phi(t)$ is the rolling angle as a function of time.

This analysis captures the basic rolling motion of the foil, which helps generate forward thrust and directional control in the water.

Pitch Motion Analysis

The second step involves observing the pitch motion from the YoZ plane. The kinematic equations for the foil in this plane are:

$$\begin{cases} x_2 = 0 \\ y_2 = \frac{c}{6} (\cos\theta(t)) \\ z_2 = \frac{c}{6} (\sin\theta(t)) \end{cases} \quad (2.18)$$

Where:

c represents the chord length of the foil,

$\theta(t)$ is the pitch angle as a function of time.

The pitch motion controls the angle of attack of the foil, directly affecting the thrust generated by the flapping motion. By adjusting the pitch angle, the foil can produce either forward thrust or lift, depending on the robot's required movement.

Combined Roll and Pitch Motion

The final stage of the analysis combines both the roll and pitch motions to describe the complete movement of the foil in the three-dimensional coordinate system. The combined equations for the x, y, and z coordinates of the foil are as follows:

$$\begin{cases} x_0 = x_1 - z_2 \sin\phi(t) = (b + l)\cos\phi(t) - \frac{c}{6}\sin\phi(t)\sin\theta(t) \\ y_0 = y_2 = \frac{c}{6}\cos\theta(t) \\ z_0 = z_1 + z_2\cos\phi(t) = (b + l)\sin\phi(t) + \frac{c}{6}\sin\phi(t)\sin\theta(t) \end{cases} \quad (2.19)$$

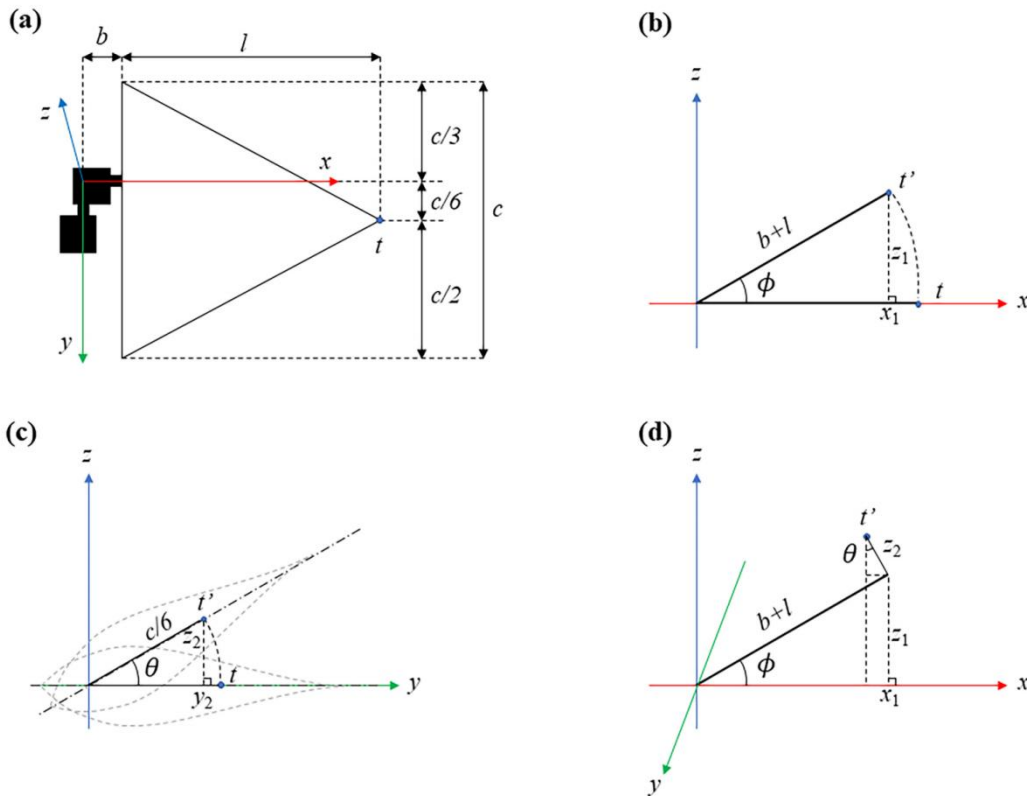


Figure 2.9 The kinematics of the flapping wing are analyzed as below: (a) A top view of the flapping wing configuration. (b) Roll movement observed in the XoY plane. (c) Pitch movement taking place in the XoZ plane. (d) A combined motion illustrated within a three-dimensional coordinate system.

The figure presents the kinematic analysis of the flapping wing. (a) shows a top view of the wing. (b) demonstrates the roll motion in the XoY plane. (c) displays the pitch motion in the XoZ plane. (d) combines the movements within a three-dimensional coordinate system.

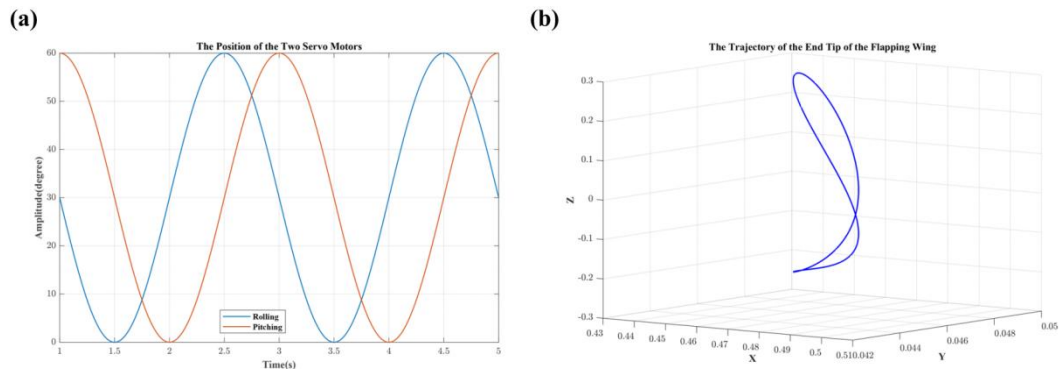


Figure 2.10 The trajectory of the flapping wing is depicted below: (a) The position of the two servo motors. (b) The trajectory traced by the end tip of the flapping wing in motion.

The figure illustrates the flapping wing's trajectory. (a) highlights the positions of the two servo motors. (b) shows the path of the wing's end tip.

These equations describe the trajectory of the foil's tip in three-dimensional space as it undergoes a combination of rolling and pitching. The foil's tip trajectory is important for determining the robot's maneuverability and hydrodynamic performance, as it influences the distribution of thrust forces and the overall efficiency of the propulsion system.

Kinematic Validation and Optimization

To validate these kinematic equations, experiments are often conducted using high-precision sensors to track the actual motion of the foil and compare it to the theoretical predictions. By measuring key variables such as thrust, lift, and torque, researchers can optimize the foil's motion to achieve maximum hydrodynamic efficiency. Adjusting the rolling angle ($\phi(t)$) and the pitch angle ($\theta(t)$) facilitates fine-tuning of the propulsion system to suit different operating conditions, such as high-speed travel or precise maneuvering.

The combined roll and pitch motions form the basis for optimizing the performance of underwater robots, enabling efficient propulsion with minimal energy consumption. By refining the kinematic parameters based on experimental data, the robot can achieve improved stability, reduced drag, and enhanced thrust generation.

Impact on Hydrodynamic Performance

The kinematic parameters described above play a significant role in determining the hydrodynamic performance of underwater robots. Through careful manipulation of these parameters, it is possible to achieve a balance between thrust, energy efficiency, and maneuverability. For example, increasing the heave amplitude (A) can generate higher thrust, but it may also result in increased drag, which would negatively impact the robot's energy efficiency.

Moreover, the frequency of oscillation (f) directly influences the robot's ability to accelerate and maneuver. A higher frequency typically leads to more dynamic movements, enabling the robot to respond quickly to changes in the environment. However, this also increases the robot's energy consumption, requiring a balance between frequency and amplitude to achieve optimal performance.

Experimental Validation of Foil Kinematics

To validate the theoretical kinematics of the foil motion, experimental setups often involve testing the foil under controlled conditions in a submersion tank. During these tests, high-precision force sensors measure thrust and drag forces, while motion capture systems track the foil's movements. This data helps refine the kinematic models and ensure that the robot's foil-based propulsion system meets the desired performance criteria.

2.2.3 Parallel Testing Platform for Hydrodynamic Performance

The experimental platform, as shown in the Figure 2.11, demonstrates eight synchronized 2-degree-of-freedom (DOF) foil systems operating simultaneously within a water tank. These foil units enable the simulation of complex hydrodynamic scenarios and facilitate parallel data acquisition from multiple sensors, improving both the efficiency and accuracy of experimental measurements.

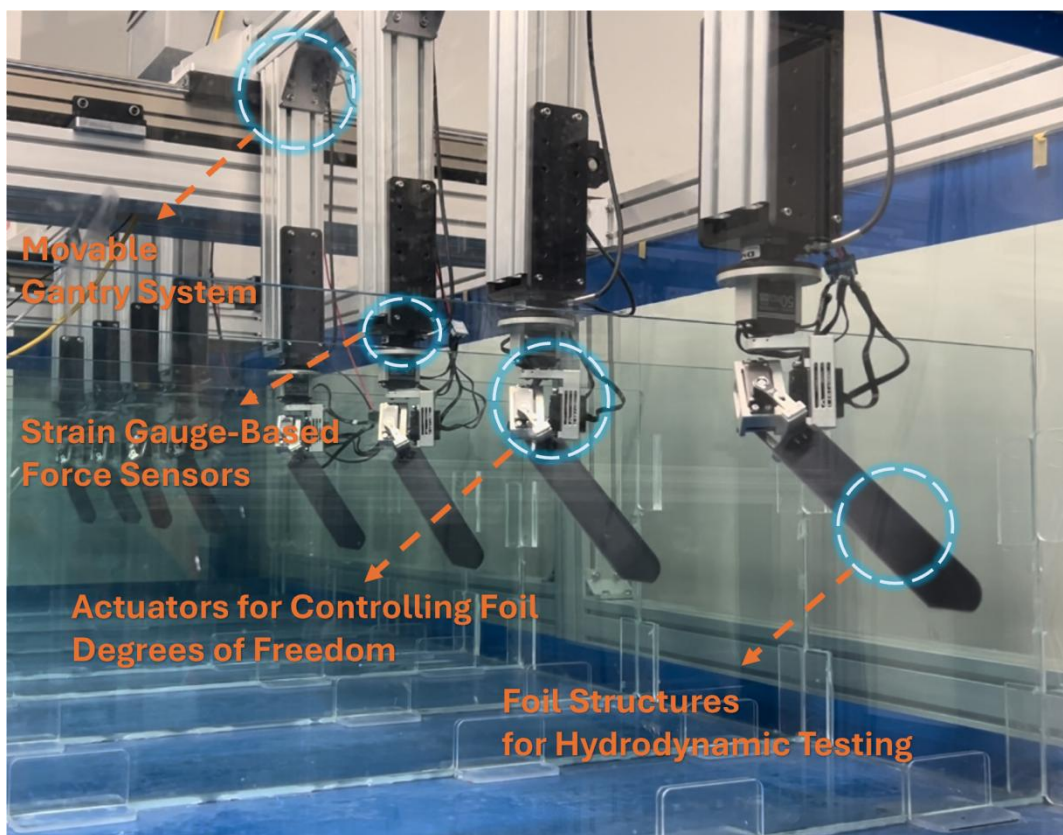


Figure 2.11 Modular Synchronized Foil Testing Platform with Movable Gantry.

Synchronized Motion and Control

Each foil unit features two degrees of freedom: pitch (rotation) and heave (vertical motion), effectively simulating aquatic locomotion patterns. Synchronized operation across all foil units ensures consistent fluid conditions within the tank,

supporting meaningful comparisons and reliable hydrodynamic analysis. Additionally, the movable gantry system allows controlled, steady-speed movement to replicate realistic aquatic environments, mimicking conditions such as uniform water flow around the foils.

Data Acquisition and Real-Time Monitoring

The foil units are equipped with strain gauge-based force sensors that record thrust, torque, and other hydrodynamic forces during operation. These sensors are connected via a communication network, enabling synchronized data acquisition across all units, reducing latency-induced errors and improving data consistency. This synchronized approach enhances the accuracy and reproducibility of experimental results, contributing to more precise hydrodynamic analysis.

Applications and Optimization

The multi-unit setup allows researchers to study various fluid interaction scenarios, optimize propulsion parameters, and test a range of movement frequencies and amplitudes. These experiments are crucial for refining hydrodynamic models and advancing underwater robot design. By conducting parallel tests, the platform facilitates rapid data collection across different conditions, providing a comprehensive set of datasets critical for evaluating and improving robot performance in diverse aquatic environments.

Modular Design and Expandability

A key feature of this platform is its modular, detachable design. Each foil unit can be disassembled and replaced with different foil shapes and configurations, enabling a variety of experimental setups. This modularity allows researchers to explore different hydrodynamic behaviors by employing alternative foil geometries, enhancing the platform's versatility to meet evolving research demands.

Integration with Control Algorithms

The data collected from these synchronized tests plays a crucial role in the development of predictive control algorithms such as MPC and the training of GPR-based hydrodynamic models, as outlined in Chapter 4. The rich, diverse datasets generated by this platform enable robust model development, improving the performance of control strategies for underwater robots operating in dynamic environments.

Conclusions of Chapter 2

1. In the current chapter there was developed a test platform paradigm for underwater dynamics measurement that overcomes the limitations of current techniques and can improve the accuracy, stability, and uniformity of measurements by incorporating advanced control systems and compensation techniques.
2. The considered platform adapts to uncertainties and degradation of thruster performance through the use of adaptive sliding controllers which is demonstrated through experimental validation, showcasing its superiority over existing methods.
3. The studied proposed test platform paradigm offers a promising approach for underwater dynamics measurement providing more accurate and reliable measurements in various applications for advancing underwater research and technology. Dynamic adaptation or the adaptation within a sliding mode (on a sliding surface), based on so-called equivalent control obtained by the direct measurements of the output signals of a first-order low-pass filter containing in the input the discontinuous control with the specially adapted magnitude value.

Chapter 3

Enhancing Stability and Depth Control through Sensors Fusion

Underwater robots require precise and reliable feedback from multiple sensors to ensure stability, maintain orientation, and regulate depth in dynamic and unpredictable aquatic environments. Two primary sensor systems are used to achieve these objectives: IMUs for attitude control and depth sensors/altimeters for accurate depth regulation [42]. This chapter explores the integration of these sensors, highlighting how they work together to enhance the robot's ability to maintain stable movement and consistent depth, which is crucial for both autonomous and remotely operated underwater missions [43].

3.1 IMU-Based Attitude Control

3.1.1 Working Principles of IMU

An IMU integrates multiple sensors—typically an accelerometer, a gyroscope, and sometimes a magnetometer—to provide detailed feedback about an object's motion, orientation, and rotation. In underwater robotics, these sensors are critical for maintaining stability, accurate orientation, and control. Let's explore the principles of these sensors using the figures provided.

1. Accelerometers

An accelerometer measures the rate of change in velocity along a specific axis. MEMS accelerometers detect linear acceleration through a proof mass suspended by springs. When the sensor experiences acceleration along its sensitivity axis, the mass deflects, and the amount of deflection is proportional to the applied force (Figure 3.1(a)). This is how the accelerometer measures the acceleration of an object in motion [44].

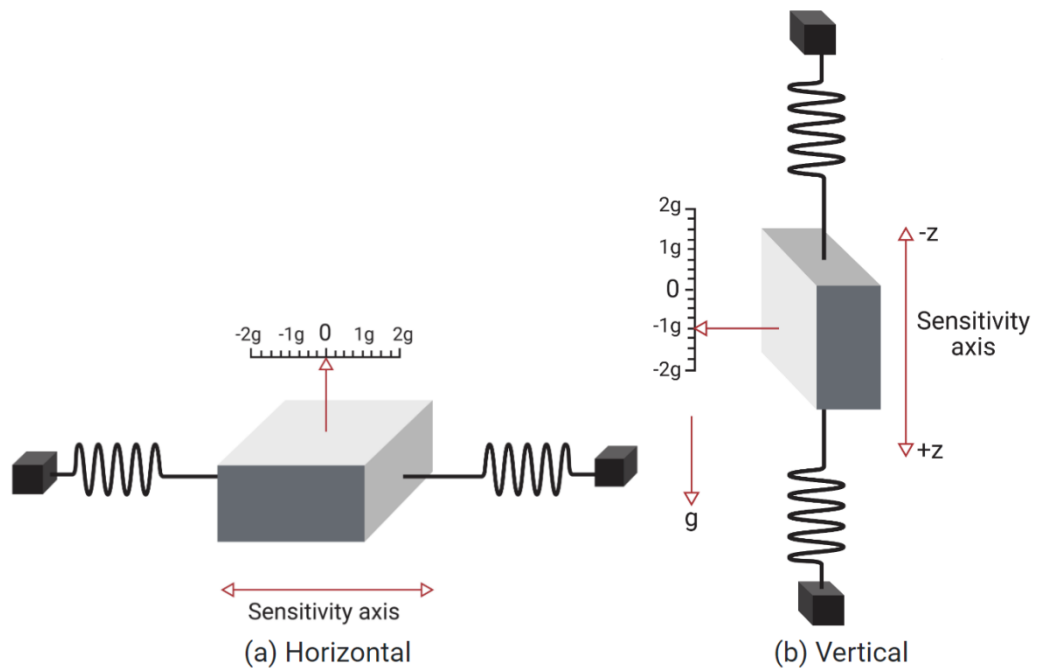


Figure 3.1 Illustration of Accelerometer Operation in Horizontal and Vertical Orientations.

Sensitivity to Gravity: When the sensitivity axis of the accelerometer aligns with the Earth's gravitational field, the sensor also detects pseudo-acceleration caused by gravity. This pseudo-acceleration doesn't correspond to actual movement but rather the constant force of gravity (Figure 3.1 (b)). In this case, the accelerometer will register an acceleration of -1 g along the vertical axis, even though there's no real displacement. Understanding this principle is important when distinguishing between true motion and gravitational effects [45][46].

2. Gyroscopes

A gyroscope measures the angular velocity or the rate of rotation around an axis. MEMS gyroscopes operate based on the Coriolis effect, which refers to the inertial force experienced by a mass in motion within a rotating frame [47]. In a MEMS gyroscope, a mass oscillates along one axis (e.g., the x-axis), and when an

angular velocity is applied about another axis (e.g., the z-axis), the Coriolis force causes a deflection along the third axis (y-axis), as shown in Figure a.

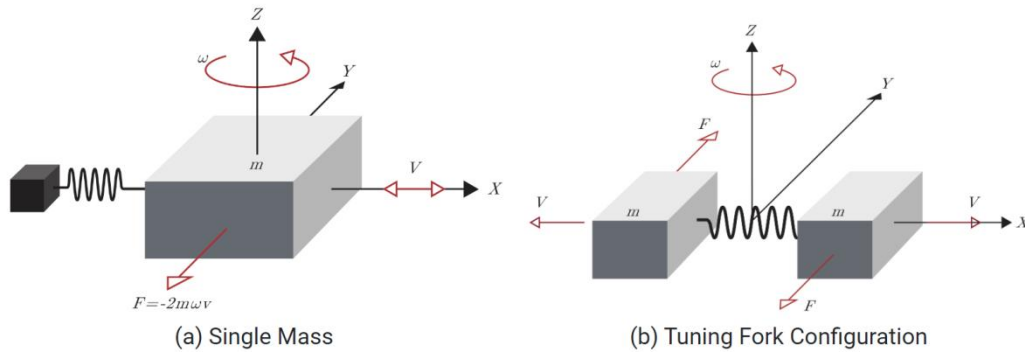


Figure 3.2 Gyroscope Operation Models – (a) Single Mass Configuration and (b) Tuning Fork Configuration (need sign web)

The Coriolis force plays a central role in the functioning of MEMS gyroscopes. The position of a mass m in the body frame is described as:

$$B_r = \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.1}$$

The inertial velocity of the mass is derived by considering both the velocity due to rotation and the tangential velocity:

$$B'_r = \begin{bmatrix} \dot{x} - \omega y \\ \dot{y} + \omega x \end{bmatrix} \tag{3.2}$$

Further, the inertial acceleration of the mass in the body frame is derived as the combination of the derivative of the velocity and the tangential acceleration:

$$B''_r = \begin{bmatrix} \ddot{x} - 2\omega\dot{y} - \omega^2 x \\ \ddot{y} + 2\omega\dot{x} - \omega^2 y \end{bmatrix} \tag{3.3}$$

The key term here is the Coriolis force that influences the motion along the perpendicular axis. From Newton's Second Law of Motion, the force acting in the sensing direction (y-axis) is given by:

$$F_y = m(\dot{y} + 2\omega\dot{x} - \omega^2 y) \tag{3.4}$$

This explains how the gyroscope detects angular velocity based on the displacement of the oscillating mass in response to rotational motion. For practical applications, if the mass starts from rest along the y -axis, the resulting force due to the Coriolis effect simplifies to:

$$F_y = 2m\omega\dot{x} \quad (3.5)$$

This force leads to a significant displacement, which is proportional to the applied angular velocity ω . MEMS gyroscopes often employ a tuning fork configuration, where two masses oscillate in opposite directions to cancel out the effects of linear acceleration and ensure accurate angular rate measurements.

Tuning Fork Configuration: Many MEMS gyroscopes employ a tuning fork configuration, where two masses oscillate in opposite directions (Figure 3.2(b)). This design cancels out the effects of linear acceleration or vibration, which could otherwise interfere with the gyroscope's measurements. The Coriolis forces on the two masses act in opposite directions, and the resulting change in capacitance is directly proportional to the angular velocity, providing more robust measurements in dynamic environments.

3. Magnetometers

A magnetometer measures the strength and direction of magnetic fields, often employing magneto resistive sensors that change resistance in response to nearby magnetic fields. MEMS magnetometers typically measure the Earth's magnetic field, enabling the robot to maintain accurate orientation relative to magnetic north. As illustrated in Figure 3.3(a), the Earth's magnetic field resembles a dipole with north and south magnetic poles.

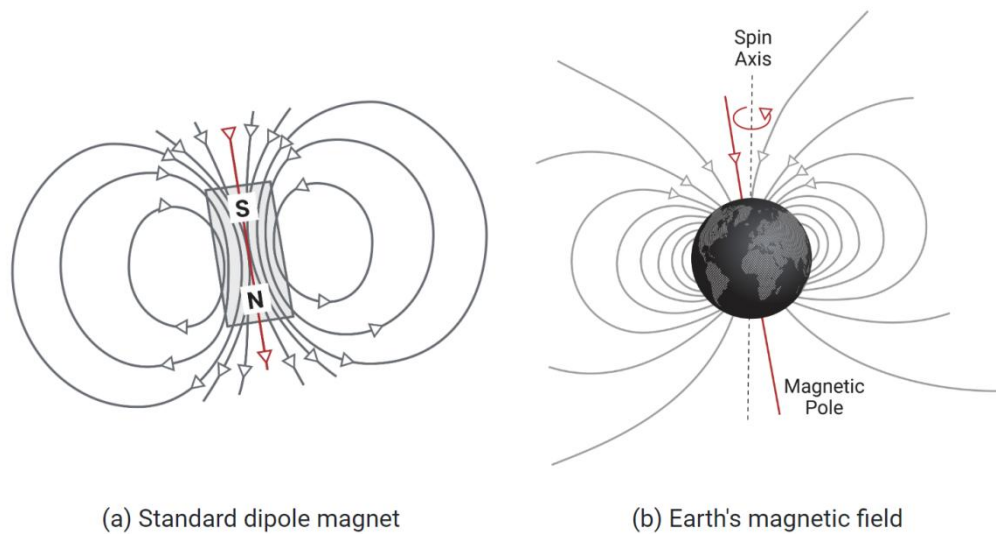


Figure 3.3 Magnetic Field Representation(a) Standard dipole magnet demonstrating magnetic field lines. (b) Earth's magnetic field modeled as a dipole, highlighting the magnetic poles and spin axis.

Magnetic Inclination and Declination: The direction of Earth's magnetic field varies depending on location. The magnetic inclination is the angle between the field lines and a horizontal plane, while magnetic declination accounts for the difference between true north and magnetic north (Figure b). In underwater robotics, this data is vital for maintaining proper orientation, especially when compensating for gyroscope drift over time. However, magnetometers are susceptible to magnetic interference, especially in industrial or underwater environments with metallic structures or other magnetic sources [48].

3.1.2 IMU Data Filtering and Sensor Fusion for Attitude Control

In real-time control and navigation systems for underwater robots, filtering techniques are crucial for ensuring the accuracy and stability of orientation estimates derived from sensor data. This section discusses two primary filtering approaches—Complementary Filtering and Kalman Filtering—focusing on their role

in IMU data processing, with particular emphasis on applications in underwater robotics [49].

Comparison between Kalman filter and complementary filter

Kalman Filtering is an optimum estimating approach that combines state estimations with observational data [50]. The Kalman Filter's strengths include its ability to handle system noise and uncertainty while providing generally accurate posture estimate [51]. However, its implementation is relatively difficult, incorporating concepts such as state space models and covariance matrices, and it requires a large amount of processing resources. This complexity may make it less appropriate for usage in resource-constrained embedded devices [52].

Table3.1 Comparison between Kalman Filtering and Complementary Filtering.

Comparison Aspect	Kalman Filtering	Complementary Filtering
Complexity	High	Low
Computational Overhead	Substantial	Minimal
Implementation Difficulty	Complex	Straightforward
Adaptability to Dynamic Environments	Moderately Good	Limited
Attitude Estimation Accuracy	High	Moderate
Resource Consumption	Considerable	Low
Suitability for High-Dynamic Movements	Excellent	Limited
Real-Time Performance	Relatively Slow	Relatively Fast
Multi-Sensor Fusion	Applicable	Applicable

On the other hand, Complementary Filtering provides a simple and effective way for estimating pose. It accomplishes this by combining data from accelerometers and gyroscopes, reducing drift in pose estimation [53]. Complementary Filtering has several advantages, including its simplicity of implementation and minimal computational load, making it ideal for applications requiring excellent real-time performance. However, it may struggle in dynamic situations or during quick motions due to accumulated mistakes, potentially resulting in unstable pose estimates.

Table 3.2 Implementation and Derivation of Complementary Filtering

Comparison Aspect	Accelerometer	Gyroscope
Sensitivity to High-Frequency Vibrational Noise	Sensitive	Insensitive
Low-Frequency Attitude Drift	Stable	Drifts
Resistance to High-Frequency Interference	Weaker	Stronger
Resistance to Low-Frequency Interference	Stronger	Weaker

Complementary Filtering is a commonly applied approach for attitude estimation that relies on data fusion from two sensors: the accelerometer and the gyroscope. The strength of this strategy lies in leveraging the advantages of both sensors to enhance the accuracy and reliability of attitude estimation.

Processing Accelerometer Data:

The accelerometer detects the deviation angle between the acceleration and gravitational acceleration vectors to determine the object's tilt. However,

accelerometers detect gravity-induced acceleration, which can interfere with actual tilt measurements. As a result, the accelerometer signal must be adjusted to account for gravitational acceleration. Typically, a low-pass filter is employed to reduce high-frequency noise created by mechanical vibrations. After filtering, the data better captures the object's tilt.

Processing Gyroscope Data:

The gyroscope is designed to measure an object's angular velocity, or speed of rotation. While gyroscopes are sensitive and precise in detecting rotational movements, their measurements can become inaccurate with time, causing drift in attitude calculation. To reduce drift, the gyroscope's angular velocity data is paired with the accelerometer's tilt information. This fusion is based on the complementarity concept, in which the outputs of both sensors are combined using a weighted average, thereby complementing each other's strengths.

Derivation of the Complementary Filter Formula:

For the gyroscope measurements gx , gy , gz , and the error terms ex , ey , ez , along with their integral components $exInt$, $eyInt$, $ezInt$, the derivation of the Complementary Filter can be expressed as follows:

$$\begin{cases} gx = \alpha \cdot gx + (1 - \alpha) \cdot (Kp \cdot ex + exInt) \\ gy = \alpha \cdot gy + (1 - \alpha) \cdot (Kp \cdot ey + eyInt) \\ gz = \alpha \cdot gz + (1 - \alpha) \cdot (Kp \cdot ez + ezInt) \end{cases} \quad (3.6)$$

The alpha (α) value typically ranges between 0.98 and 0.99, adjustable based on the specific scenario and hardware characteristics.

Kp represents the proportional gain, which adjusts the influence of the error term on the gyroscope measurements.

The error terms ex , ey , ez represent the cross-product of the estimated direction of gravity and the accelerometer measurements.

The integral components $exInt$, $eyInt$, $ezInt$ are the integrals of the error terms.

Overall Implementation Process:

a) Input Conversion: Convert angular velocity to radians per second.

$$\begin{cases} gx = gx \times 0.01745329 \\ gy = gy \times 0.01745329 \\ gz = gz \times 0.01745329 \end{cases} \quad (3.7)$$

b) Normalization of Accelerometer Measurements:

$$norm = \sqrt{ax^2 + ay^2 + az^2} \quad (3.8)$$

Compute and norm

e) Integration of Error Terms: Integrate the error terms.

$$\begin{cases} exInt = exInt + ex \times Ki \times dt \\ eyInt = eyInt + ey \times Ki \times dt \\ ezInt = ezInt + ez \times Ki \times dt \end{cases} \quad (3.9)$$

f) Adjustment of Gyroscope Measurements: Adjust the gyroscope measurements, applying the Complementary Filter to each axis.

g) Quaternion Integration Update: Update the quaternion using the angular velocity after Complementary Filtering.

$$\begin{cases} q0 = q0 + (-q1 \times gx - q2 \times gy - q3 \times gz) \times 0.5 \times dt \\ q1 = q1 + (q0 \times gx + q2 \times gz - q3 \times gy) \times 0.5 \times dt \\ q2 = q2 + (q0 \times gy - q1 \times gz + q3 \times gx) \times 0.5 \times dt \\ q3 = q3 + (q0 \times gz + q1 \times gy - q2 \times gx) \times 0.5 \times dt \end{cases} \quad (3.10)$$

h) Quaternion Normalization: Normalize the quaternion.

$$\begin{cases} norm = \sqrt{q0^2 + q1^2 + q2^2 + q3^2} \\ q0 = \frac{q0}{norm}, q1 = \frac{q1}{norm}, q2 = \frac{q2}{norm}, q3 = \frac{q3}{norm} \end{cases} \quad (3.11)$$

i) Calculation of Euler Angles: Compute the Euler angles based on the updated quaternion.

$$\begin{cases} pitch = \text{asin}(-2 \cdot q1 \cdot q3 + 2 \cdot q0 \cdot q2) \cdot 57.3 \\ roll = \text{atan}2(2 \cdot q2 \cdot q3 + 2 \cdot q0 \cdot q1, -2 \cdot q1 \cdot q1 - 2 \cdot q2 \cdot q2 + 1) \cdot 57.3 \\ yaw = \text{atan}2(2 \cdot (q1 \cdot q2 + q0 \cdot q3), q0^2 + q1^2 - q2^2 - q3^2) \cdot 57.3 \end{cases} \quad (3.12)$$

Through this process, Complementary Filtering effectively combines information from the accelerometer and gyroscope, providing a relatively accurate attitude estimation.

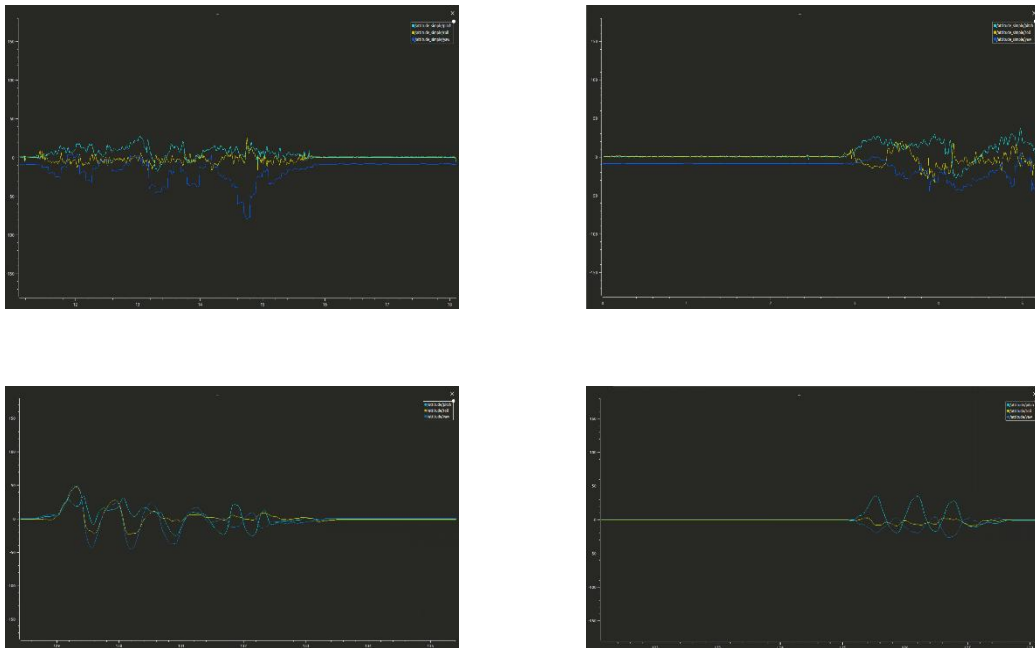


Figure 3.4 Results of comparative analysis

In the conducted research, two experimental setups were rigorously designed to contrast the performance of the tilt-compensated magnetometer-based yaw angle calculation method against the attitude estimation achieved through a complementary filter algorithm. The comparative analysis revealed that the complementary filter algorithm exhibited lower noise levels and a higher rate of synchronization in attitude computation. Notably, the yaw angle derived from the magnetometer reflects an absolute position, inherently preventing the initialization of the yaw value at zero. This characteristic imposes significant limitations on the closed-loop control systems of remotely operated vehicles, due to the inherent inability to reset or calibrate the yaw orientation at the start of an operation.

In contrast, the complementary filter approach generates posture information relative to the position at startup, adjusting dynamically to changes in orientation. This adaptability ensures a more robust response to irregular alterations in the IMU's operational environment, delivering stable posture signals with significantly reduced noise. Furthermore, the complementary filter demonstrated superior recovery performance following disturbances, underscoring its efficacy for enhancing the precision and reliability of posture estimation in dynamic and unpredictable conditions. This analysis underscores the complementary filter's potential for improving the robustness and accuracy of attitude control systems, particularly in applications requiring precise navigation and orientation control under varying environmental conditions.

3.2 Depth-Based Control

Depth gauges play a critical role in underwater robots by helping them maintain precise depth relative to the water surface. This section discusses how depth sensors work in tandem with control algorithms to regulate the robot's depth, enabling it to navigate efficiently and maintain consistent positioning in varying aquatic environments.

A depth gauge typically works by measuring the water pressure exerted at a certain depth and translating this pressure into depth information. The deeper a robot goes, the higher the pressure, which the sensor can calculate using the relationship between pressure and depth. This data is vital for ensuring that the underwater robot can maintain its desired depth, especially when operating in complex environments like deep-sea exploration or shallow-water operations.

MPC is a powerful control algorithm that calculates future control actions by solving an optimization problem at each time step. It predicts the future behavior of the robot's depth based on a dynamic model and applies corrective actions to minimize the error between the predicted and desired depth. MPC is especially advantageous in underwater environments, where external forces like currents or changes in water density can affect the robot's depth and stability.

3.2.1 Introduction to Depth Sensors

In underwater robotics, precise depth measurement is critical for maintaining stability and control, especially in tasks such as navigation, exploration, and data collection. Depth sensors, also known as pressure sensors or depth gauges, play an essential role by providing real-time data on the robot's distance from the water surface or sea level.

Depth sensors typically work by measuring the water pressure exerted on the sensor and converting this data into a corresponding depth. The pressure increases with depth, and this pressure change is translated using the following formula [54]:

$$P = P_0 + \rho gh \quad (3.13)$$

where: P is the pressure at depth,

- P_0 is the atmospheric pressure at sea level,
- ρ is the density of the water,
- g is the acceleration due to gravity,
- h is the depth below the water surface.

This relationship between pressure and depth enables accurate measurements, which are crucial for underwater navigation and control. Modern depth sensors are highly sensitive and can detect depth variations within a few centimeters, making them ideal for maintaining stability in ROVs and AUVs. These sensors often have built-in temperature compensation mechanisms to ensure accuracy in varying water conditions, as water density can change with temperature and salinity.



Figure 3.5 ISD4000 Piezo-Resistive Pressure Sensor for Depth Measurement

The ISD4000 sensor is equipped with a piezo-resistive pressure sensor that offers exceptional depth accuracy of $\pm 0.01\%$ FS, with an upgrade option to $\pm 0.005\%$ FS. This high level of accuracy ensures that even the slightest changes in depth are detected and responded to, which is crucial in underwater robotics where maintaining depth within tight tolerances is essential for tasks like pipeline surveys or asset deployment [55].

Depth Measurement Range: The sensor offers a range of depth measurement capabilities, from 10 bar to 600 bar, making it adaptable to various underwater conditions—from shallow waters to extreme depths of up to 6000 meters when housed in titanium. This flexibility enables the sensor to function in a wide range of applications, including real-time depth monitoring in both ROVs and AUVs.

Data Precision: The ISD4000 provides a resolution of 0.001% FS, which facilitates the capture and processing of highly detailed measurements by the robot's control system. This level of precision ensures accurate depth measurements, even in rapidly changing conditions or when the robot is subject to underwater currents

Temperature Compensation: The sensor is temperature-compensated, ensuring that its measurements remain accurate even as the water temperature fluctuates. This is particularly important in deep-sea environments where temperature gradients can affect the robot's performance. The sensor's temperature accuracy is $\pm 0.01^\circ\text{C}$, ensuring reliable operation across a calibrated temperature range of -5°C to 35°C .

The ISD4000's ability to provide real-time depth data at rates of up to 100Hz, combined with its robust titanium housing and low power consumption, make it ideal for underwater robotics applications that require both durability and precision.

3.2.2 Depth Control with Model Predictive Control (MPC)

Prediction Model: The MPC algorithm relies on a mathematical model of the robot's dynamics to predict future states. For depth control, the model includes factors such as buoyancy, thrust, and drag forces. The robot's depth is predicted over a finite time horizon using this model, taking into account the current state and control inputs.

The system dynamics can be represented as:

$$x_{k+1} = f(x_k, u_k) + w_k \quad (3.14)$$

where:

x_k is the state vector at time step k (including the current depth and depth rate),

u_k is the control input (e.g., thruster output),

$f(x_k, u_k)$ represents the system dynamics,

w_k is the process noise.

Optimization Problem: MPC solves an optimization problem at each time step to determine the control actions that minimize a cost function. The cost function typically penalizes deviations from the desired depth and excessive control inputs, ensuring both stability and energy efficiency.

The optimization problem can be written as:

$$\min_u \sum_{i=0}^{N-1} \left[(x_{k+i} - x_{\text{target}})^T Q (x_{k+i} - x_{\text{target}}) + u_{k+i}^T R u_{k+i} \right] \quad (3.15)$$

where:

x_{target} is the desired depth,

Q is a weighting matrix for the depth error,

R is a weighting matrix for the control effort,

N is the prediction horizon.

The control input u_k is calculated to minimize this cost over the prediction horizon while satisfying constraints on control inputs and system states (e.g., thruster limitations, maximum depth).

Receding Horizon: MPC operates in a receding horizon fashion, meaning that only the first control input from the optimized sequence is applied at each time step. After applying the control, the prediction is updated with the new state, and the optimization is repeated.

Constraints: In underwater environments, depth control often involves constraints, such as maximum thrust output, minimum and maximum permissible depths, and velocity limits. MPC can handle these constraints explicitly, ensuring that the robot operates safely within its limits.

The constraints can be incorporated as:

$$x_{min} \leq x_{k+i} \leq x_{max}, u_{min} \leq u_{k+i} \leq u_{max} \quad (3.16)$$

Advantages of MPC in Depth Control

Predictive Capability: Unlike reactive controllers such as PID, Model Predictive Control (MPC) anticipates future states and adjusts control actions accordingly, which is particularly advantageous in dynamic and uncertain environments like underwater.

Handling Constraints: MPC can handle input and state constraints, which are essential for depth control when considering physical limitations, such as thruster capacity and depth boundaries.

Optimal Control: By minimizing a cost function, MPC ensures that the robot operates efficiently, balancing depth precision with energy consumption.

Example of MPC for Depth Control

Consider an underwater robot equipped with a depth sensor and thrusters. The robot needs to maintain a specific depth while minimizing energy consumption. The system's dynamics are modeled, taking into account forces such as buoyancy and drag. At each time step, MPC predicts the future depth of the robot over a 10-second horizon and computes the optimal thrust to keep the robot within 0.1 meters of the desired depth. The controller adjusts the thrust in real time, accounting for constraints on thrust output and depth limits.

The depth regulation problem can be expressed as:

$$\min_u \sum_{i=0}^{N-1} [(d_{k+i} - d_{\text{setpoint}})^2 + \lambda u_{k+i}^2] \quad (3.17)$$

where:

d_{k+i} is the predicted depth at future time $k + i$,

d_{setpoint} is the desired depth,

λ is a regularization parameter penalizing the control effort u .

By solving the optimization problem, the underwater robot can maintain stable depth despite external disturbances such as underwater currents or changes in water density. The control inputs generated by MPC consider the robot's hydrodynamic model, ensuring that it stays within the desired depth range while minimizing energy consumption and avoiding overcorrection.

In the figure below, we can observe the robot's depth stabilization process, where MPC successfully manages to maintain a constant depth under fluctuating environmental conditions. The smooth and gradual adjustments showcase the

controller's ability to predict the robot's future state and apply corrective actions in advance. The robot avoids oscillations or overshooting, providing a reliable and efficient control system.

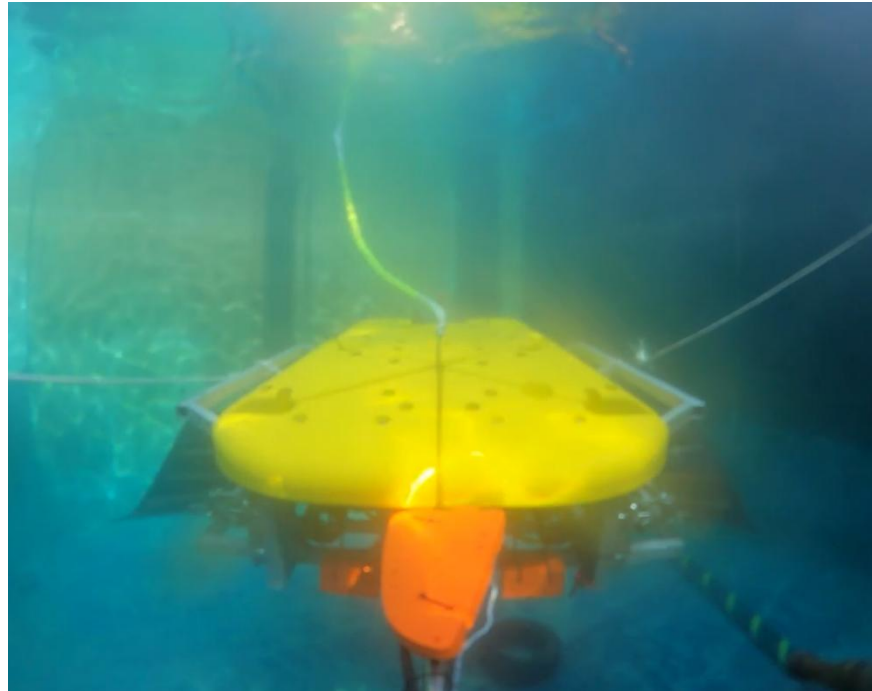


Figure 3.6 Biomimetic Underwater Robot for Depth Control Experiment.

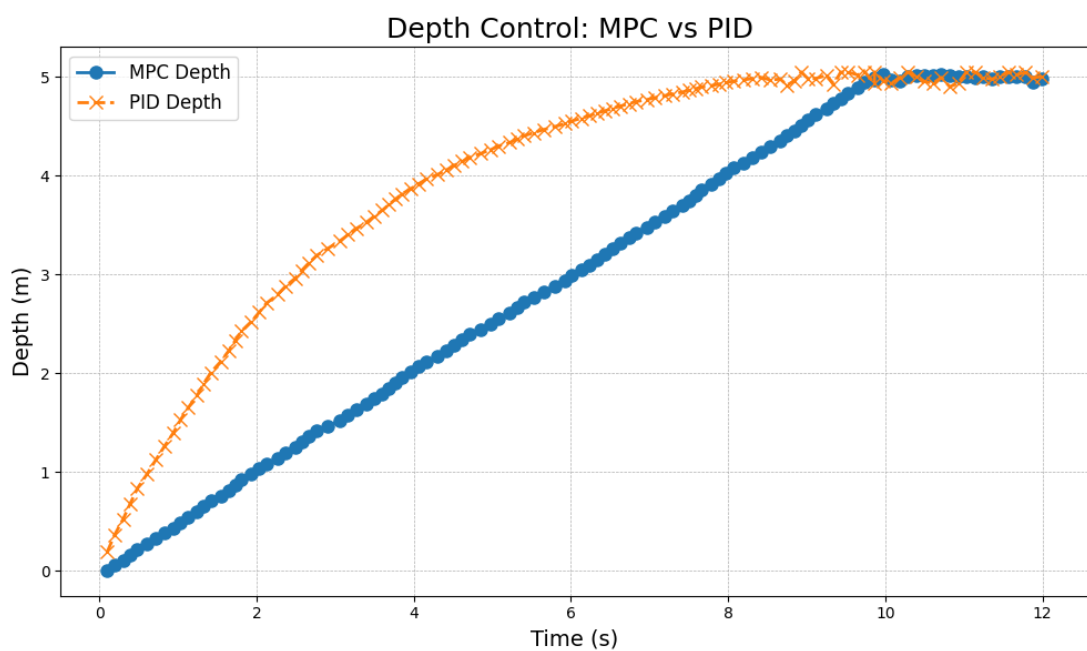


Figure 3.7 Depth Control Comparison: MPC vs PID.

In addition to the superior capabilities of MPC discussed earlier, the results from the depth control experiment (as shown in the chart comparing MPC and PID) further emphasize the advantages of MPC over PID control in underwater depth regulation.

The PID controller (orange line) achieves the target depth of 5 meters more quickly, reaching it in about 9 seconds. However, the quick response comes at the cost of overshoot and oscillations, which cause instability. This behavior is problematic in underwater environments where maintaining precise control is critical, especially in avoiding obstacles or maintaining consistent altitude.

On the other hand, the MPC controller (blue line) takes a slightly longer time to reach the target depth, approximately 10 seconds, but with a much smoother approach and no overshoot. The robot stabilizes immediately upon reaching the desired depth. This demonstrates the effectiveness of MPC in ensuring stability and precision, key aspects in environments where the underwater robot's performance and safety are paramount.

Table 3.3: Depth Control Performance Comparison Between MPC and PID Controllers

Time (s)	MPC Depth (m)	PID Depth (m)	Observations
0.5	0.21	0.83	PID reacts faster initially, reaching a depth of 0.19 m.
2.0	1.03	2.62	PID shows a faster depth gain, but overshoots target depth early on.
5.0	2.5	4.26	PID continues to increase depth rapidly, while MPC is slower but stable.
7.5	3.75	4.86	PID approaches target depth of 5 m, with signs of oscillations.
10.0	5.02	4.94	MPC reaches target depth of 5 m with minimal oscillation; PID fluctuates around 5 m.
11.5	5.01	5.05	Both algorithms stabilize near target, with MPC showing less overshoot.

The data in this table highlights the fundamental differences in behavior between MPC and PID in terms of depth control. While the PID controller responds more quickly, it also exhibits overshoot and oscillations that can reduce stability, especially in dynamic underwater environments. In contrast, the MPC controller takes a slightly longer time to reach the target depth, but it does so with a smoother approach and without overshoot, resulting in improved stability.

These observations align with the theoretical advantages of MPC discussed previously, including its ability to achieve precise control with minimal oscillations, enhanced robustness against disturbances, and greater overall stability in challenging

underwater conditions. This makes MPC a more suitable choice for applications that demand high levels of accuracy, stability, and efficiency.

Advantages of MPC Over PID

Stability and Accuracy: Unlike PID, which exhibits oscillations and overshoot due to its reactive nature, MPC ensures smoother, more controlled responses. It is particularly effective in dynamic and uncertain underwater conditions, where abrupt depth changes could lead to operational hazards.

Predictive Control: The predictive capability of MPC enables it to anticipate the future behavior of the robot and adjust control actions preemptively. This capability is superior to the reactive approach of PID, making MPC more suitable for real-time depth regulation in dynamic underwater environments.

Energy Efficiency: The absence of overshoot and smoother depth changes result in energy savings. With MPC, the robot's thrusters operate more efficiently, as they don't have to correct large deviations caused by overshoot or oscillations.

Robustness: MPC's model-based predictions make it more robust to disturbances such as underwater currents, pressure changes, or varying water densities. In contrast, PID may struggle to adapt to these external forces without significant tuning adjustments.

Thus, while PID control provides a quicker initial response, MPC offers a much more stable and reliable approach to depth control, especially in environments that demand precision and efficiency.

Conclusions to Chapter 3

1. The conducted analysis of the Inertial Measurement Unit has confirmed that data processing quality depends on applied filtering techniques. There were studied and compared 2 filtering methods: Complementary Filters and Kalman Filters.

2. The comparative analysis of 2 experimental setups revealed that the complementary filter algorithm exhibited lower noise levels and a higher synchronization rate in attitude computation. Notably, the yaw angle derived from the magnetometer reflects an absolute position, inherently preventing the initialization of the yaw value at zero. This characteristic imposes significant limitations on the closed-loop control systems of remotely operated vehicles, due to the inherent inability to reset or calibrate the yaw orientation at the start of an operation.

3. In contrast, the complementary filter approach generates posture information relative to the position at startup, adjusting dynamically to changes in orientation. This adaptability ensures a more robust response to irregular alterations in the IMU's operational environment, delivering stable posture signals with significantly reduced noise. Furthermore, the complementary filter demonstrated superior recovery performance following disturbances, underscoring its efficacy in enhancing the precision and reliability of posture estimation in dynamic and unpredictable conditions.

4. The comparison between MPC and PID controllers highlighted the superior performance of MPC in depth control. MPC effectively anticipated external disturbances, resulting in smoother and more accurate trajectory tracking, with minimal overshoot. In contrast, PID showed faster initial response but lacked robustness, leading to increased oscillations. These results demonstrate MPC's

effectiveness in achieving precise and stable depth control under dynamic underwater conditions.

Chapter 4

Data-Driven Approaches and Optimization Methods for Amphibious Robot Applications

In the current Chapter, we consider the stability of operation and control of depth of AUV as well as their enhancement due to work of contemporary function that's sensors' data fusion.

4.1 Robot Structure and Motion Mechanisms

4.1.1 Structural Design and Motion Capabilities of the Amphibious Robot

Examples of dissimilar robots' motions are shown below (Figure 4.1)

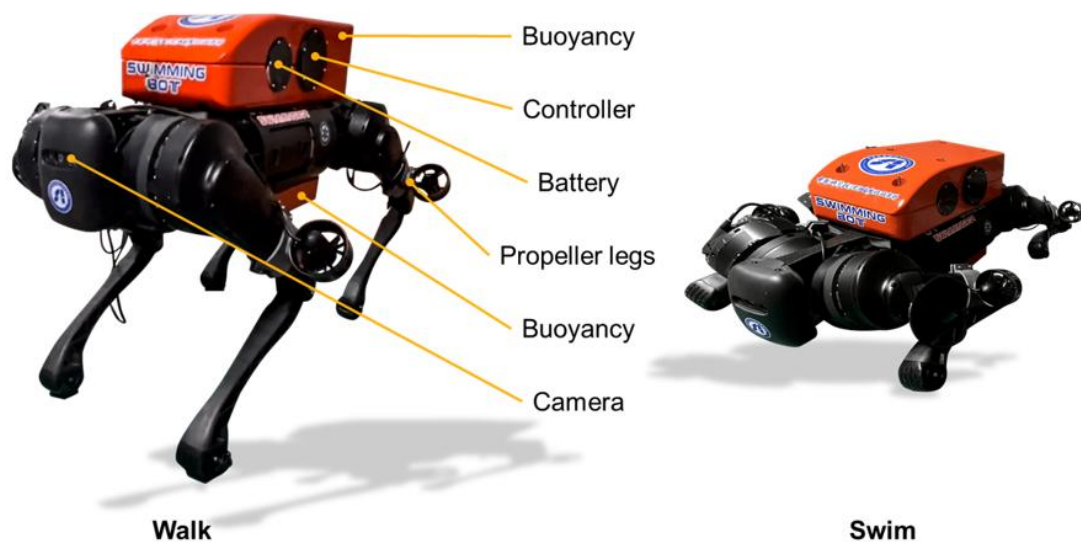


Figure 4.1. Basic Structure of the Robot and Mode Transition

The underwater robot featured in this study is designed with a quadrupedal structure that allows for dual operational modes: walking on the seabed and swimming in open water. The design of this amphibious robot is optimized for both

terrestrial and aquatic environments, making it highly adaptable and efficient for underwater exploration and various research applications [56].

Mechanical Structure

As shown in Figure 4.1, the robot's construction consists of several key components that enable its seamless operation in both environments:

Controller Unit: Mounted on top, the robot's controller manages all computational tasks, including sensor fusion, motion control, and communication. The controller interacts with the onboard sensors, such as cameras and IMUs, to provide real-time data to the control algorithms.

Battery Pack: Ensuring the robot's operational longevity, the battery pack powers all electronic systems, including the propulsion units and sensors.

Propeller Legs: The legs of the robot feature propellers at their tips, which provide thrust during swimming. These legs also serve as standard walking appendages when the robot operates on solid surfaces, allowing it to traverse underwater terrains.

Buoyancy Modules: The robot is equipped with adjustable buoyancy modules that help maintain neutral buoyancy while underwater, ensuring smooth and stable motion at varying depths.

Camera System: Located at the front, the camera captures real-time footage and supports vision-based navigation.

Motion Mechanisms

The robot is capable of walking and swimming, depending on the environment and mission requirements:

Walking Mode: On the seabed, the robot operates in a quadrupedal walking gait similar to terrestrial robots. The propeller-equipped legs provide enough stability

and force to allow the robot to traverse uneven surfaces, making it suitable for underwater inspections of pipelines or seafloor exploration.

Swimming Mode: When submerged in open water, the robot transitions to a swimming mode, where the propeller legs rotate to generate thrust, similar to how marine animals move. The combination of walking and swimming capabilities allows the robot to operate efficiently in dynamic underwater environments.

By integrating these mechanical components with the motion capabilities, the robot can perform tasks that require transitioning between land-like underwater environments and fully submerged operations. In the next section, we will discuss how this mechanical framework supports the development of the data-driven hydrodynamic model.

4.1.2 Leg Mechanism for Terrestrial Walking and Underwater Swimming

The robot's leg mechanism is designed to enable both terrestrial walking and underwater swimming with a focus on simplicity and efficiency. As shown in Figure, the robot achieves this by relying on controlled movement of the thigh (upper leg) and shin (lower leg), which allows it to adapt to different terrains and environments.

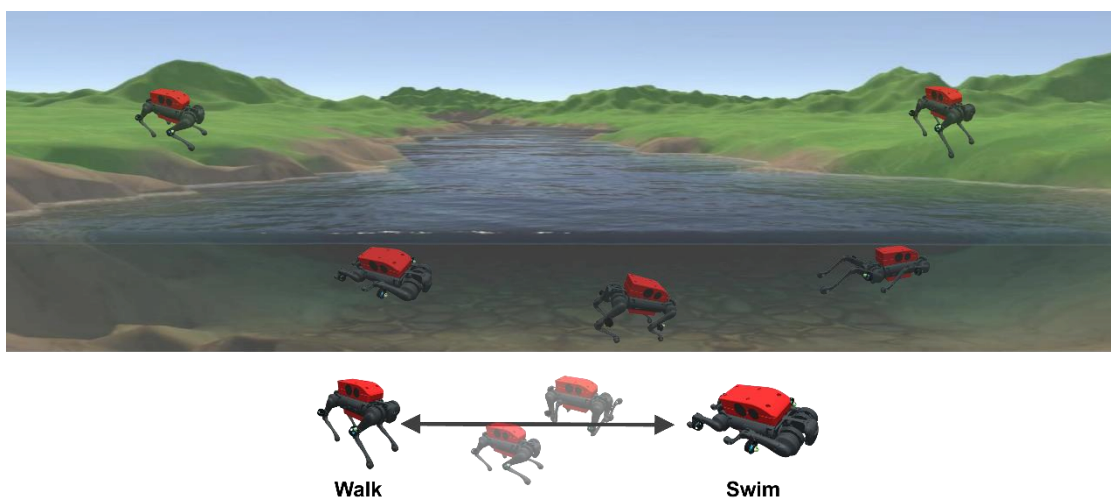


Figure 4.2. Amphibious Robot Transitioning Between Water and Land

Walking on Land

On land, the robot uses a walking gait where the legs alternate between lifting and lowering. The thigh moves primarily in a forward-backward motion, which allows the robot to step and propel itself forward. The shin aids in extending or retracting the leg, adjusting the height of each step, and ensuring that the robot can maintain stability across uneven surfaces. This motion mimics the mechanics of traditional quadrupedal robots, but is adapted for amphibious functionality.

Swimming Underwater

When the robot transitions to underwater swimming, the leg mechanism adapts to function similarly to flippers or fins. The thigh and shin move to create a sweeping motion through the water, generating thrust. This motion allows the robot to propel itself through the water while maintaining control over its orientation. The precise adjustments in leg movement help the robot efficiently change direction and maintain stability, similar to how aquatic animals use their limbs for swimming.

In both modes, the combination of thigh and shin movements is critical for the robot's ability to navigate complex environments, whether walking on the seabed or swimming in open water. The flexibility of this leg mechanism makes the robot highly adaptable and capable of performing diverse tasks in various aquatic and terrestrial environments.

4.1.3 Hydrodynamic Experiments and Data Collection

To validate the hydrodynamic performance and control strategies of the amphibious robot, a series of experiments were conducted using a 3-degree-of-freedom (3-DOF) towing tank. This advanced platform enables precise measurement of the robot's behavior under different aquatic conditions by allowing controlled movement along the X, Y, and Z axes. As illustrated in Figure 4.3, these tests provide detailed insights into the robot's interaction with water and help optimize its propulsion and control systems [57].

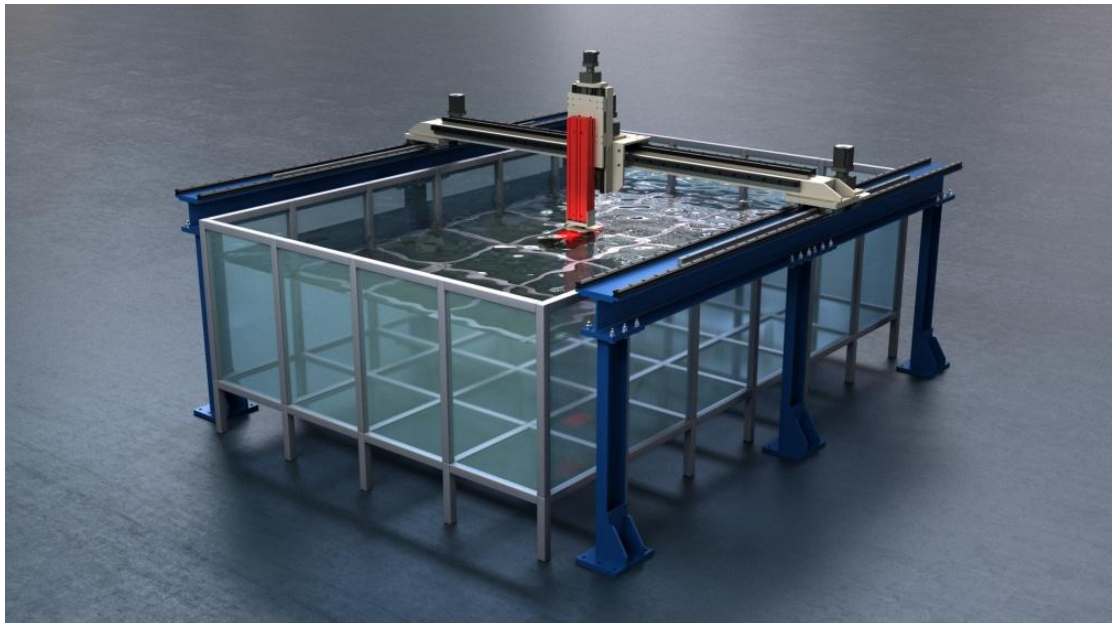


Figure 4.3 Intelligent Towing System for Underwater Robot

The image above shows a 3-degree-of-freedom (3-DOF) towing tank, which is a critical tool used for hydrodynamic experiments, particularly for underwater robotics. The towing tank allows for precise control over the movement of the robot or object being tested in three axes: X, Y, and Z. This enables researchers to simulate real-world conditions such as lateral movement, vertical displacement, and forward motion, providing accurate data on the hydrodynamic forces acting on the robot.

Key Features and Functionality:

Three Degrees of Freedom: The system can move in the X, Y, and Z axes, offering full control over the trajectory and movement of the tested object. This is essential for simulating complex underwater maneuvers and understanding how the robot interacts with fluid forces from different angles.

Precision Control: The gantry-like structure, combined with precise actuators, allows for controlled movement of the testing apparatus over the water surface and within the water. This precision is crucial for conducting repeatable experiments and collecting reliable hydrodynamic data.

High-Resolution Data Collection: The system is typically integrated with sensors and force measurement devices to capture the forces acting on the robot in real-time. This data helps refine the robot's design and control algorithms, ensuring better performance in underwater environments.

Versatility: The towing tank is designed to accommodate various types of robots or models, making it a versatile tool for both small-scale and full-scale hydrodynamic testing.

This 3-DOF towing system plays a vital role in validating force models and motion predictions by allowing controlled testing in a simulated aquatic environment. By utilizing this setup, researchers can optimize the robot's design for maximum efficiency and stability in both surface and underwater conditions [57].

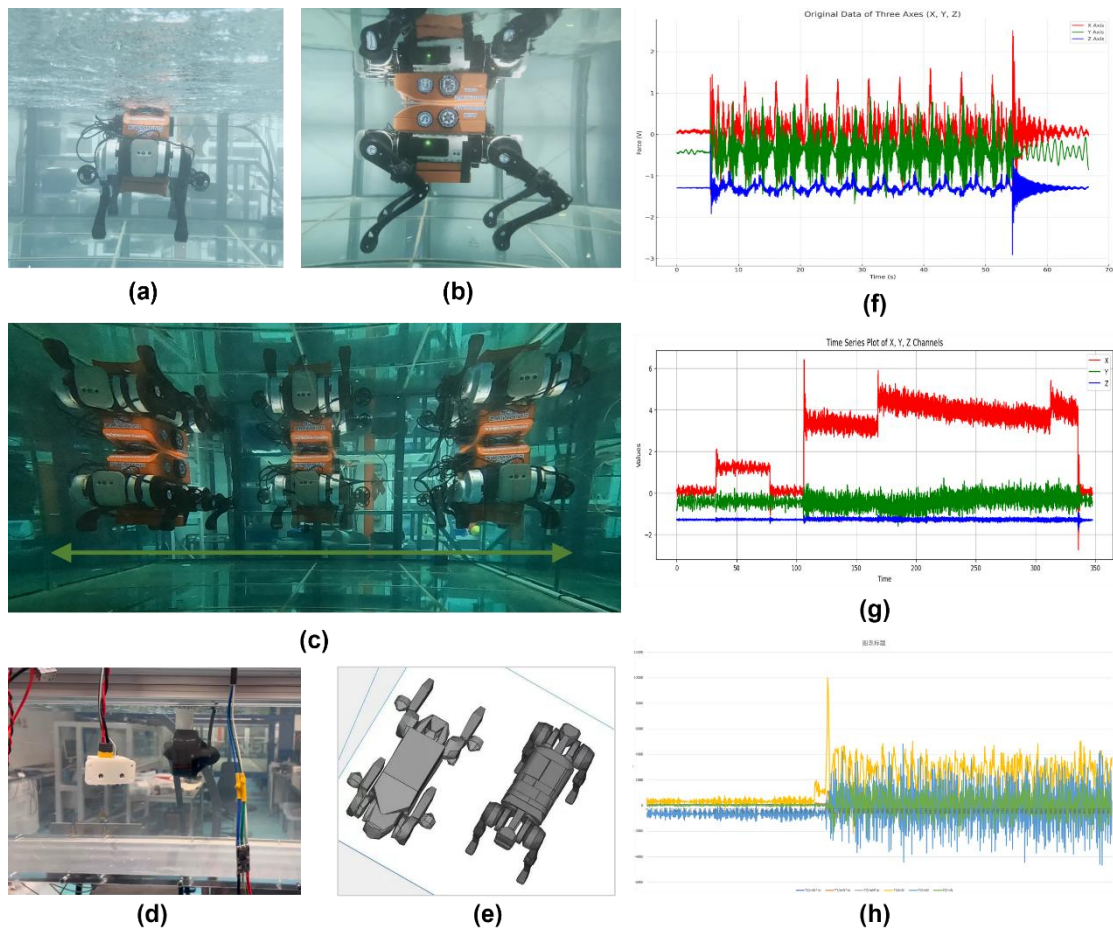


Figure 4.4. Experimental Setup and Data Analysis for Hydrodynamic Testing of Amphibious Robot.

Towing Tests Using 3-DOF Towing Tank

The towing tests conducted in the 3-DOF towing tank are a cornerstone of the hydrodynamic analysis. The towing tank, shown in Figure 4.4 (a) and Figure 4.4 (b), provides precise control over the robot's movement in three dimensions, simulating real-world underwater environments. The towing tank's capability to move along the X, Y, and Z axes enables us to replicate the different forces the robot encounters, such as drag and lift, while it moves in water.

The towing tests, as shown in Figure 4.4 (c), measure the resistance experienced by the robot when moving along the Y-axis. This test is critical in assessing the robot's lateral movement and the drag forces it encounters, helping to

fine-tune the robot's hydrodynamic shape for more efficient movement through water. The precise data collected from these tests directly inform the robot's design improvements and control algorithm refinements.

Thruster Test

In conjunction with the towing experiments, the performance of the robot's thrusters was also evaluated under controlled fluid conditions, as shown in Figure 4.4 (d). Each thruster was tested to assess its thrust efficiency, crucial for both surface and underwater propulsion. The data from these tests, displayed in Figures 4.4 (f) and 4.4 (g), illustrate the response of the robot's legs to different control inputs, such as sine-wave motions and step commands. These experiments help optimize the propulsion system, ensuring that the robot generates sufficient thrust while minimizing energy consumption.

The single thruster test data in Figure 4.4 (h) provides further detail by isolating the performance of individual components, enabling targeted adjustments to ensure the propulsion system operates at peak efficiency.

Integration of the Scaled Model and Full-Sized Testing

Figure 4.4 (e) shows the use of a scaled model for preliminary testing. This approach facilitates quick and cost-effective design iterations before moving to full-scale tests. The scaled model helps identify potential issues early, ensuring optimal performance of the full-sized robot when subjected to towing and thruster tests.

4.2 GPR-Based Hydrodynamic Modeling

The GPR-based hydrodynamic model developed in this study relies on data collected from force sensors and kinematic observations during the robot's underwater testing. The main objective is to predict hydrodynamic forces acting on the robot's body in real-time. These predictions are crucial for adaptive control in varying underwater conditions, ensuring the robot operates efficiently while minimizing energy consumption.

4.2.1 GPR Model Architecture Consideration

GPR-based Model

GPR is a non-parametric, probabilistic model used to learn the dynamics of a system by capturing the relationships between input and output data. GPR is particularly effective for modeling continuous-time changes in forces as a function of sensor inputs while providing uncertainty estimates [59]. The model defines a distribution over functions and uses training data to update this distribution. The core of GPR lies in predicting a continuous function based on a set of observed data points [60].

Given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ where x_i represents the input (e.g., sensor data), and y_i represents the output (e.g., hydrodynamic force), GPR models the relationship between inputs and outputs as a multivariate Gaussian distribution [61]:

$$y(x) \sim \mathcal{GP}(m(x), k(x, x')) \quad (4.1)$$

where $m(x)$ is the mean function (typically assumed to be zero), and $k(x, x')$ is the covariance (kernel) function that defines the similarity between points x and x' .

The covariance function $k(x, x')$ plays a crucial role in determining the behavior of the model. A common choice is the RBF) kernel:

$$k(x, x') = \sigma_f^2 \exp\left(-\frac{\|x - x'\|^2}{2l^2}\right) \quad (4.2)$$

where σ_f^2 is the signal variance, and l is the length scale, which controls how quickly the function can vary.

The conditional distribution gives the GPR prediction for a new test point x_* is given by the conditional distribution:

$$p(y_* | x_*, X, y) = \mathcal{N}(\mu(x_*), \sigma^2(x_*)) \quad (4.3)$$

where the mean and variance of the prediction are:

$$\mu(x_*) = k(x_*, X)[K(X, X) + \sigma_n^2 I]^{-1} y \quad (4.4)$$

$$\sigma^2(x_*) = k(x_*, x_*) - k(x_*, X)[K(X, X) + \sigma_n^2 I]^{-1} k(X, x_*) \quad (4.5)$$

Here, $K(X, X)$ is the covariance matrix of the training inputs, σ_n^2 is the noise variance, and $k(x_*, X)$ represents the covariance between the test point x and the training points [62].

The advantage of GPR is its ability to provide not only a mean prediction but also a measure of uncertainty for each prediction. This is critical in dynamic environments, such as underwater robotics, where sensor noise and environmental variability can significantly affect the accuracy of force predictions [63].

In this study, we optimize the hyperparameters of the kernel function (e.g., σ_f^2 , l and σ_n^2) using a maximum likelihood estimation approach, and the model is trained on real-time sensor data collected during robot movements. GPR's ability to model the underlying uncertainties in fluid-structure interactions makes it highly suitable for robust predictions in varying underwater environments [64].

Model Architecture

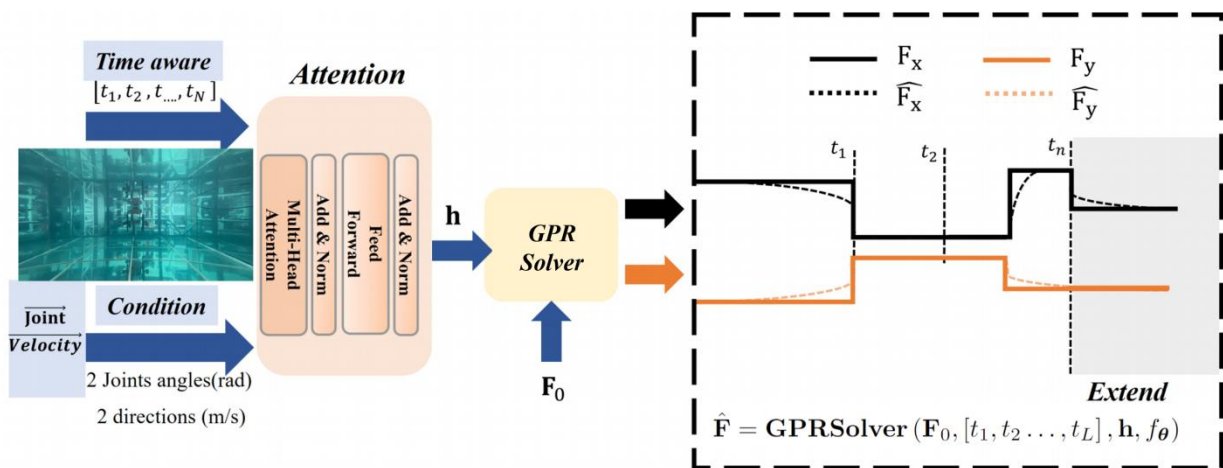


Figure 4.5 Data-Driven GPR Model for Predicting Hydrodynamic Forces.

Figure 4.5 Left: The robot's motion under different conditions is represented in time steps $[t_1, t_2, \dots, t_N]$, capturing key kinematic data.

Middle: The GPR model processes the kinematic information and maps it to a latent vector h , incorporating uncertainty estimates to account for variations in the environment and the robot's movements. Right: The prediction of the hydrodynamic force trajectory F_b is generated by the GPR model based on the initial condition F_0 , providing a probabilistic force prediction across the time steps $[t_1, t_2, \dots, t_L]$.

The proposed model architecture aims to predict the hydrodynamic forces acting on a quadruped robot using GPR based on a sequence of observational data [65]. The key components of the model are GPR and the covariance (kernel) function. The steps involved in the model architecture are as follows:

Input Data: The input consists of a sequence of kinematic observations x_1, x_2, \dots, x_N , each representing motion parameters sampled at uniform time intervals. These parameters include two joint angles and two linear velocities, providing a comprehensive description of the robot's movement.

Covariance (Kernel) Function: The GPR model uses a kernel function to learn the relationship between the inputs and the hydrodynamic forces. The covariance function $k(x, x')$ defines the similarity between data points x and x' , crucial for predicting forces. The most used kernel in this study is the RBF kernel, given by:

$$k(x, x') = \sigma_f^2 \exp\left(-\frac{\|x - x'\|^2}{2l^2}\right) \quad (4.6)$$

where σ_f^2 is the signal variance and l is the length scale. This kernel allows the GPR model to capture both smooth and rapidly changing dynamics, which are critical in underwater environments.

Prediction Framework: The GPR model predicts the forces $F(t)$ acting on the robot at any time t , based on the input sequence of observations. Given a test input x_* , the model provides a Gaussian distribution over possible values of the force, with mean $\mu(x_*)$ and variance $\sigma^2(x_*)$.

Model Training and Hyperparameter Optimization: The GPR model's hyperparameters, including the signal variance σ_f^2 , length scale l , and noise variance

σ_n^2 , are optimized using MLE. The training data consists of real-time sensor measurements from the robot's movements in water, and the hyperparameter tuning ensures that the model accurately captures the underlying dynamics.

Prediction: The output of the GPR model is a set of predicted force vectors $F_1, F_2, \dots, F_L \in \mathbb{R}^2$, where L is the prediction length, spanning various time intervals. The model primarily predicts forces in the x and y directions, as forces in the z -axis (due to gravity and buoyancy) are assumed to remain constant.

The GPR-based architecture provides robust and adaptive force predictions by leveraging the uncertainty quantification inherent in Gaussian Processes. Unlike traditional machine learning models that offer point estimates, the GPR model offers a distribution of possible outcomes, making it well-suited for underwater environments where sensor noise and dynamic fluid conditions can introduce significant uncertainty.

4.2.2 Model Training and Dataset

The dataset for training the GPR model was collected through detailed towing tests and real-time force sensor measurements, as discussed in Section 4.1. The data includes various robot configurations, movements, and environmental conditions. Each data point consists of:

Input: Robot's joint angles, velocities, and sensor readings.

Output: Hydrodynamic force vectors in x , y , and z axes.

The GPR model is trained using this dataset to minimize prediction error while accounting for uncertainty in fluid dynamics. By utilizing an optimized RBF kernel, the model ensures smooth and reliable predictions, even in dynamic or turbulent underwater conditions.

The model's hyperparameters (length scale, noise variance, etc.) were optimized using a MLE approach. During testing, the model demonstrated a high level of accuracy in predicting forces in real-time.

Learning Objective

The learning objective of the GPR model is to minimize the prediction error between the predicted force vectors and the ground truth measurements while accounting for uncertainty. The process involves the following steps:

Dataset Preparation:

The dataset consists of sequences of observation data along with corresponding force measurements. The data is divided into training, validation, and test sets. Formally, the dataset can be represented as:

$$\mathcal{D} := \{(F_0^j, x_0^j), (F_1^j, x_1^j), \dots, (F_L^j, x_L^j)\}_{j=1}^M \quad (4.7)$$

where F_i^j is the predicted force at time step i for trajectory j , x_i^j is the corresponding input observation, L is the total number of time steps per trajectory, and M is the number of trajectories in the dataset.

Loss Function:

In GPR, the model outputs a mean prediction along with a variance estimate for the force vector at each time step. The loss function takes into account both the prediction error and the uncertainty estimate. The NLML is commonly used as the objective function to optimize GPR models, which maximizes the likelihood of the observed data given the predicted mean and covariance.

The NLML loss function is formulated as:

$$\mathcal{L}(\theta) = \frac{1}{2}y^\top K_\theta^{-1}y + \frac{1}{2}\log|K_\theta| + \frac{n}{2}\log(2\pi) \quad (4.8)$$

where y represents the observed forces, K_θ is the covariance matrix parameterized by θ (which includes the signal variance σ_f^2 , length scale l , and noise variance σ_n^2 , and n is the number of data points. Minimizing this function allows the model to fit the data while properly accounting for uncertainty.

Backpropagation and Gradient Calculation:

Unlike traditional neural networks, GPR uses analytical gradients for hyperparameter optimization. The gradient of the negative log marginal likelihood concerning the hyperparameters θ is computed to update the kernel's parameters:

$$\frac{\partial \mathcal{L}(\theta)}{\partial \theta} = \frac{1}{2}y^\top K_\theta^{-1} \frac{\partial K_\theta}{\partial \theta} K_\theta^{-1}y - \frac{1}{2}Tr\left(K_\theta^{-1} \frac{\partial K_\theta}{\partial \theta}\right) \quad (4.9)$$

This gradient allows for backpropagation through the kernel's hyperparameters, ensuring that the model can adjust the signal variance, length scale, and noise variance to improve predictions.

Optimization:

Hyperparameter optimization is typically carried out using gradient-based optimization algorithms such as the Adam optimizer or L-BFGS. These methods update the hyperparameters of the covariance function by minimizing the NLML loss. During training, the model adjusts the kernel parameters iteratively to maximize the likelihood of the observed data while minimizing prediction uncertainty.

The overall objective of the GPR-based model is to minimize the prediction error as well as ensure that the uncertainty estimates are well-calibrated, which is particularly crucial in dynamic environments like underwater robotics.

Experiments

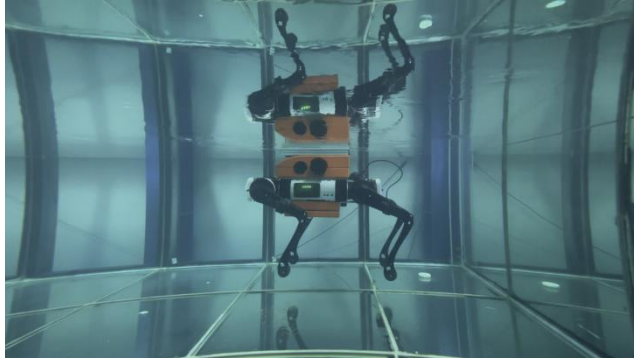


Figure 4.6 The amphibious robot's posture variations underwater result in different hydrodynamic coefficients

Setup:

To train and evaluate our GPR-based hydrodynamic force prediction model, we conducted a series of controlled towing experiments [66]. These experiments were designed to provide a rich dataset for training, validation, and testing, ensuring the accurate modeling of the robot's interactions with the surrounding fluid. The key components of the experimental setup are described below:

Pool Environment:

All experiments were conducted in a controlled pool environment, ensuring repeatable and consistent hydrodynamic conditions. The water temperature and depth were maintained constant throughout the experiments to avoid external variability in fluid dynamics.

Towing Mechanism:

A specialized towing mechanism was utilized, capable of towing the robot at different speeds and directions. The towing speeds varied from 0.2 m/s to 0.5 m/s, with increments of 0.1 m/s. The robot was towed in three main directions: along the x-axis, y-axis, and diagonally at 45 degrees (xy). These variations enabled the capture of diverse motion scenarios, which are essential for training the GPR model.

Force Sensors:

High-precision force sensors were installed on the robot to capture hydrodynamic forces acting on the robot in real-time. The sensors recorded force data along the x, y, and z axes. Since the GPR model is focused on predicting forces in the x and y directions (with z-axis forces assumed constant), this detailed sensor data provides a comprehensive training dataset.

Robot Configuration:

To simulate different locomotion scenarios, the quadruped robot's limb configurations were varied by adjustment of the joint angles. The robot was tested under a range of movement patterns, providing sufficient input diversity for the GPR model to learn fluid-structure interactions effectively.

The dataset derived from these experiments contains several kinematic observations (joint angles and velocities) paired with corresponding force measurements. This comprehensive dataset is critical for training the GPR model to predict hydrodynamic forces while quantifying the uncertainty in the predictions.

Note: batch refers to the batch size during the training stage

Table 4.1 Input and output formats of the Datasets

Expr.	Input	Output
Expr1	Condition x: [batch, 100, 4] and Initial: F_0 [batch, 2]	[batch, 100, 2]

Expr2	Condition x: [batch, 50, 4] and Initial: F_0 [batch, 2]	[batch, 50, 2]
Expr3	Condition x: [batch, 50, 4] and Initial: F_0 [batch, 2]	[batch, 50, 2]

Dataset

The dataset was collected during the towing experiments described. These experiments were specifically designed to measure the forces acting on the quadruped robot across 192 distinct towing speeds and joint configurations [67][68].

From Table , the dataset is augmented in the following ways:

Expr1: This experiment extends the time series to 100-time steps, representing sequential data of the quadruped robot maintaining a constant attitude angle. The GPR model is tasked with predicting the forces in two axial directions (x and y) over these 100-time steps. The focus is on testing the GPR's ability to model the hydrodynamic forces under static conditions.

Expr2: This experiment highlights the comparative predictive capabilities of GPR models under variable conditions for online learning. The temporal length of each condition is reduced to 10-time steps, randomly selected from the dataset. Additionally, a condition variable is concatenated to the input data, varying across five distinct scenarios. This setup allows the GPR model to generalize across different environmental conditions, making it more versatile in predicting forces in dynamic contexts.

Expr3: This experiment addresses the increased complexity of dynamic conditions. Random perturbations are introduced at each time step, with magnitudes equal to 10% of the standard deviation of the respective force values. Similar to Expr2, the trajectories are resampled across 192 distinct conditions to test how well the GPR model adapts to noisy and dynamically changing scenarios. This experiment

is key to understanding the robustness of GPR in environments with variable and unpredictable conditions.

Model Prediction Performance

In the following experiments, we evaluate the performance of different GPR models in predicting dynamic hydrodynamic forces on a quadruped robot. We compare the model's predictions with the ground truth using RMSE and MAE. Several kernel configurations, including the RBF and Matern kernels, are tested, along with the addition of different noise levels and varying training data conditions [69].

Note: The suffix -S indicates static conditions in Expr1, -C denotes conditions that change over time in Expr2, and -N represents noisy and changing conditions in Expr3. The * symbol is used to highlight the best-performing models.

Table 4.2 Performance on Different Conditions

Models	MAE-S	RMSE-S	MAE-C	RMSE-C	MAE-N	RMSE-N
GPR-RBF	9.8e-3	4.0e-3	4.1e-3	5.0e-3	5.3	6.8
GPR-Matern	8.3e-3	3.9e-3	3.5e-3	2.8e-3	4.0	5.3
GPR-RBF (Noisy)	5.4e-3	6.2e-4	2.8e-3	1.5e-3	2.8	4.9
GPR-Matern (Noisy)	3.7e-4*	5.8e-4*	2.4e-4*	5.1e-5*	2.0*	4.0*

Analysis

From Table 2, GPR models with different kernel choices perform well under various dynamic conditions. The Matern kernel generally outperforms the RBF

kernel, especially when the data includes variability and noise. This is likely due to the Matern kernel's ability to model rougher functions and better capture the underlying complexities of the hydrodynamic forces.

In scenarios with noisy conditions (as in Expr3), the GPR-Matern model demonstrates significantly better performance, with errors as low as 4.0, making it highly suitable for deployment in real-world underwater environments where sensor noise is prevalent.

Additionally, GPR's uncertainty quantification provides a distribution of possible outcomes, which proves advantageous in noisy environments. The GPR-Matern (Noisy) model providing the most accurate and robust predictions.

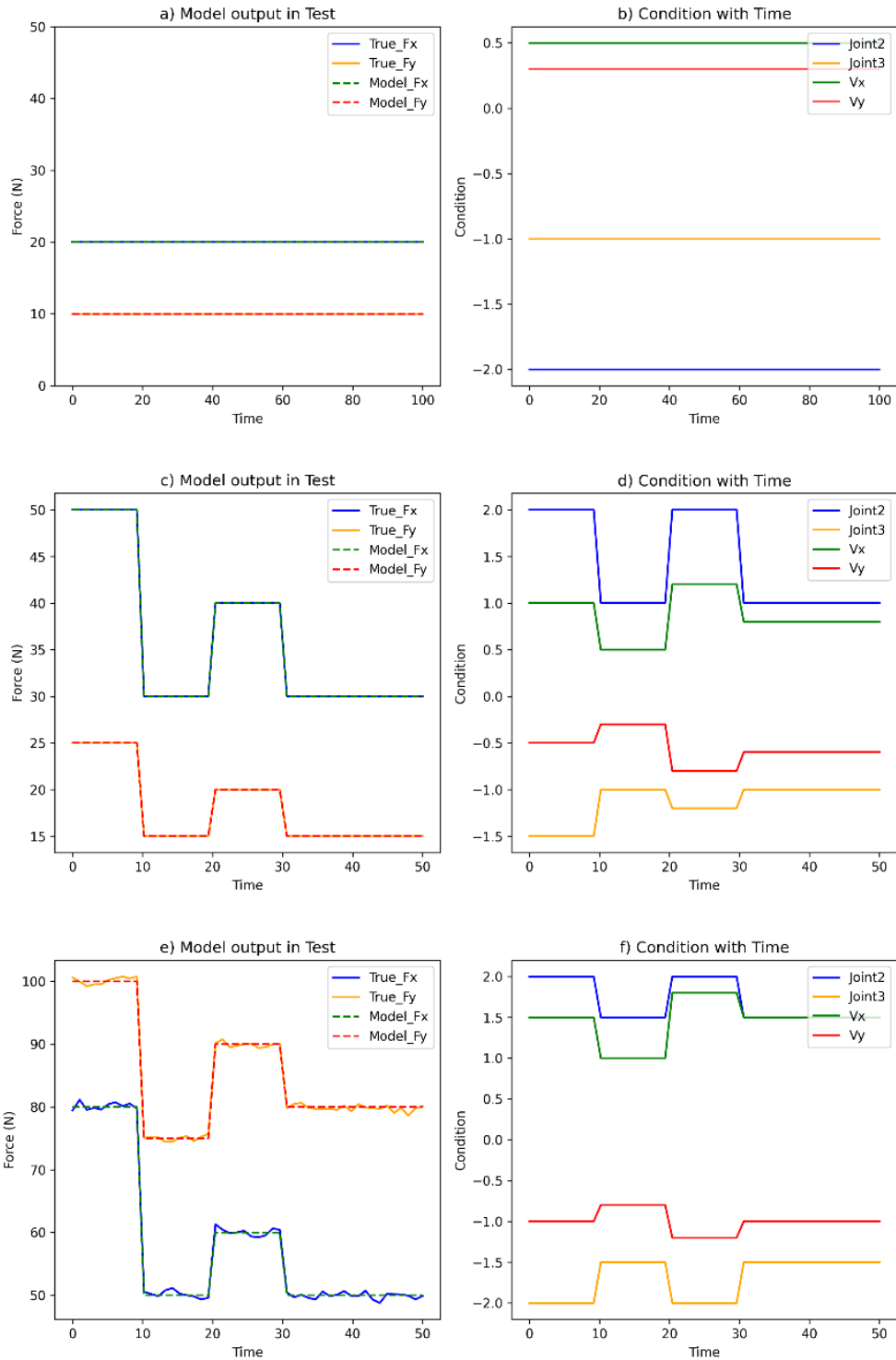


Figure 4.7 illustrates the model prediction performance under different conditions and time sequence lengths:

(a) and (b) present the static force and conditions over time in **Expr1**.

(c) and (d) illustrate the change in force and conditions over time in **Expr2**.

(e) and (f) depict the noisy force dynamics and conditions over time in **Expr3**.

In (a), (c), and (e), the dotted lines represent the model prediction trajectories.

These visualizations demonstrate the model's ability to accurately track the actual forces over time, even under varying and noisy conditions. The close alignment between the predicted forces (dotted lines) and the actual measurements confirms the effectiveness of the GPR-Matern model.

4.2.3 Trajectory Tracking and Performance Assessing

Building upon the hydrodynamic modeling presented in earlier sections and incorporating the thruster system, we designed an experiment to evaluate the underwater robot's ability to perform precise trajectory tracking. Specifically, this experiment aimed to test the robot's ability to follow a pre-defined 8-shaped trajectory along the Z-axis (depth) and X-axis (horizontal motion). This type of maneuver is crucial for underwater applications, such as environmental surveying, obstacle avoidance, and efficient path planning [70][71].

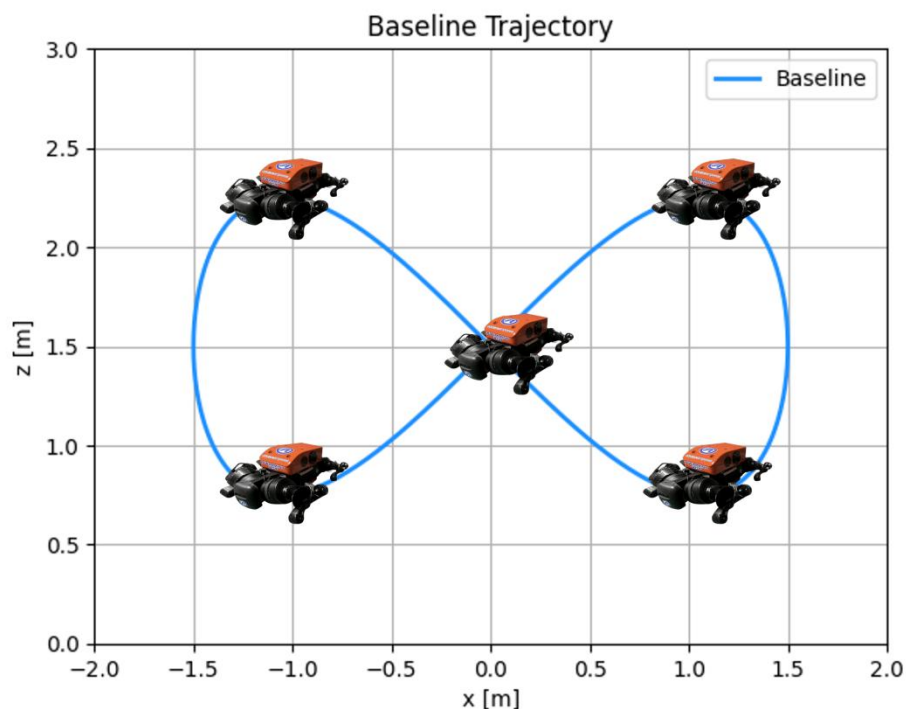


Figure 4.8 Trajectory in the Shape of 8 on the X-Z Plane

IMU-Based Odometry and Numerical Integration

The successful tracking of underwater trajectories relies heavily on precise localization and movement estimation. To achieve this, the onboard IMU mounted on the robot captured its dynamic movements. The IMU provided high-frequency measurements, including linear acceleration and angular velocity, which were employed to estimate the robot's position through numerical integration.

Odometry estimation involved integrating the IMU's linear acceleration over time to compute velocity, followed by integrating velocity to determine displacement along the X and Z axes. Sensor fusion and bias compensation were used to improve the accuracy of the numerical integration, reducing the cumulative drift commonly associated with double integration of IMU data.

This method ensured that even in complex underwater environments, with limited access to external positioning systems, the robot could depend on its internal sensors to approximate its position and follow the desired trajectory [72].

Trajectory Execution via Predictive Control

For executing the 8-shaped trajectory, the thrusters were controlled using an MPC framework. MPC is particularly effective in underwater environments where external forces, such as water currents or turbulence, can affect stability. By incorporating the GPR-derived hydrodynamic models, the MPC controller could predict the forces and moments acting on the robot, adjust thruster output dynamically, and minimize the tracking error.

The thruster commands were generated to ensure a smooth path, with real-time adjustments based on the deviation between the estimated position (using IMU-based odometry) and the desired trajectory. The MPC utilized a cost function that

prioritized maintaining the planned trajectory, minimizing energy consumption, and ensuring stability during rapid changes in direction.

Integrated Sensors for Enhanced Control

The IMU data was combined with input from depth gauges to provide a comprehensive estimate of the robot’s state. Depth sensors ensured that the Z-axis tracking maintained high accuracy, while the thrusters ensured precise movement in the X-axis. By combining multiple sensory inputs, the control system effectively minimized disturbances and stabilized the robot’s movement, even during transitions between different trajectory segments.

Trajectory Tracking Results

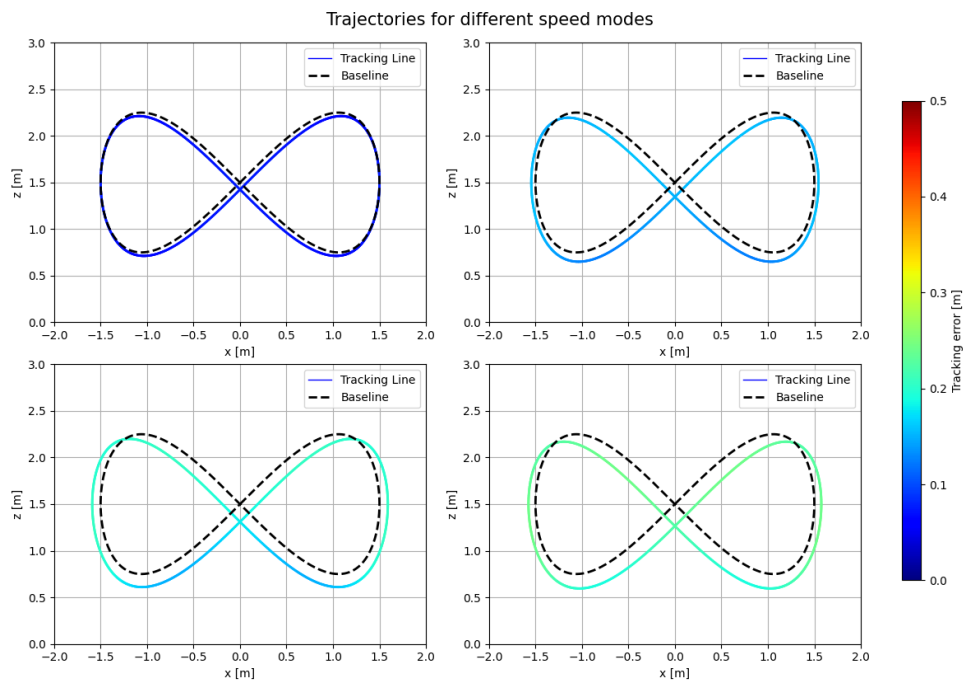


Figure 4.9 Comparison of Actual and Desired Trajectories for Different Speed Modes.

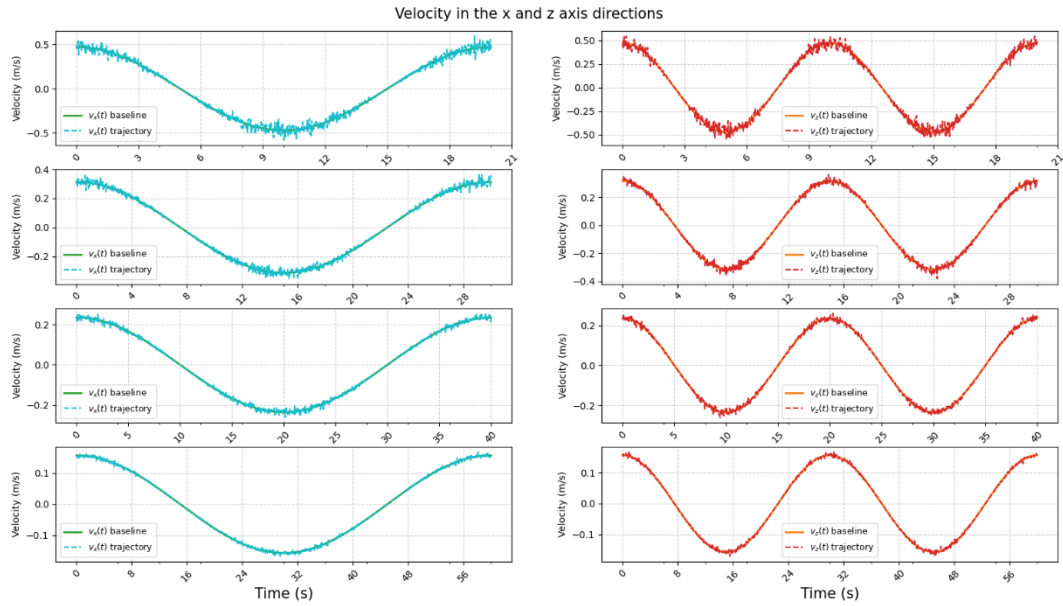


Figure 4.10 Velocity Profiles in X-Z Plane Compared to Desired Trajectories.

Trajectory Comparison

Figure 4.9 illustrates the comparison between the actual trajectory and the desired trajectory during underwater motion. Specifically, the target trajectory forms a figure-eight pattern in both the x-axis and z-axis. The solid blue line represents the desired trajectory (Baseline), while the colored line represents the actual path tracked by the underwater robot (Tracking Line). Each subplot corresponds to a different speed mode. The closeness of the two lines in all cases indicates the robot's ability to accurately follow the desired path, demonstrating effective trajectory tracking even under dynamic conditions.

Velocity Comparison in x and z Directions

Figure 4.10 presents the velocity profiles for both the x and z directions over time. On the left side, the velocity along the x-axis is compared against the desired velocity. On the right side, the z-axis velocities are shown in a similar manner. The solid lines indicate the desired velocities ($v(t)$ baseline), while the dashed lines show the actual velocities ($v(t)$ trajectory) recorded during the experiment. The figure

demonstrates the alignment between the actual and expected velocities, and each row corresponds to a different operating speed.

Speed Mode Analysis and Performance Evaluation

The results in Table 4.2 show an increase in tracking error with higher speeds, emphasizing that the control system achieves high precision at lower speeds, with an average error of 0.069 in Speed Mode 1. However, as speed increases, the error metrics reveal a decline in tracking precision, with Speed Mode 4 showing the highest errors (average of 0.216), indicating the system's increasing difficulty in maintaining tracking accuracy at elevated velocities.

Table 4.3: Tracking Errors by Speed Mode

Speed Mode	Maximum Error	Minimum Error	Average Error
Speed Mode 1	0.074	0.064	0.069
Speed Mode 2	0.158	0.124	0.141
Speed Mode 3	0.205	0.148	0.177
Speed Mode 4	0.242	0.190	0.216

Analysis of Factors Contributing to Increased Tracking Error at Higher Speeds

The rise in tracking error at higher speeds is influenced by several factors:

Amplification of Noise in Control Outputs: Higher velocities require rapid, high-magnitude thruster outputs to achieve desired adjustments. This results in amplified noise within control signals, which can destabilize the robot’s trajectory and increase deviation from the target path.

Computational Constraints of MPC: At higher speeds, the MPC must solve the optimization problem within shorter intervals, potentially straining its capacity to produce optimal solutions. In cases where the MPC cannot fully converge on a solution, the control output may lack precision, exacerbating trajectory deviations.

Sensor Noise and Integration Drift: Rapid movements and vibrations at elevated speeds intensify sensor noise, increasing localization errors. The accumulated drift in estimated position, particularly at higher speeds, makes trajectory tracking more challenging.

Increased Hydrodynamic Disturbances: Higher velocities result in intensified hydrodynamic forces, including drag and turbulence. Although MPC compensates for these forces, disturbances at higher speeds may exceed the model's predictive capabilities.

Analysis and Key Observations

Trajectory Accuracy: The tracking accuracy for the figure-eight trajectory reflects the control system's ability to compensate for disturbances, achieving high accuracy despite slight deviations due to hydrodynamic disturbances and sensor noise.

Velocity Alignment: The alignment between actual and desired velocities across speed modes demonstrates the efficiency of the control algorithm in maintaining consistent speeds, crucial for both navigation precision and energy efficiency.

MPC Advantage: Leveraging MPC enabled the robot to anticipate and adjust for external disturbances in real time, ensuring effective depth control with minimal error. The integration of GPR-based hydrodynamic models empowered the robot to proactively adapt to changing underwater conditions.

Dynamic Response: The GPR-guided thruster propulsion system balanced hydrodynamic forces, particularly important during sharp turns in the 8-shaped trajectory, where dynamic forces presented the greatest challenge.

Sensor Fusion for Robustness: Integrating IMU data with depth sensors enhanced localization robustness, crucial for accurate trajectory tracking in underwater conditions.

Applications and Future Directions

The trajectory tracking experiment demonstrates the robustness of the data-driven hydrodynamic models within the control framework, suggesting several promising applications:

Autonomous Navigation: The robot's ability to follow complex trajectories makes it suitable for autonomous underwater tasks, including inspections, mapping, and surveys.

Optimized Propulsion: Feedback from tracking performance provides insights for further optimizing propulsion. Adjustments to control gains, improved sensor fusion, and advanced predictive models could refine the control strategies, enhancing efficiency.

These findings establish a foundation for future developments in underwater robotics, especially in trajectory accuracy and energy efficiency improvements for dynamic aquatic environments.

4.3 Stability and Control During Amphibious Robot Transitions

The land-sea transition is a critical phase for amphibious robots, requiring advanced locomotion mechanisms and control strategies to ensure stability and efficiency. As amphibious robots move from land to water, they encounter unique challenges that arise from the need to adapt their propulsion and control systems to drastically changing environments. This transition involves navigating a complex interplay of environmental factors, such as bottom currents, coastal waves, and bottom return currents, which significantly affect the robot's stability and maneuverability. The simultaneous operation of legs and propellers is pivotal for overcoming these challenges, enabling the robot to maintain balance and adapt to dynamic forces.

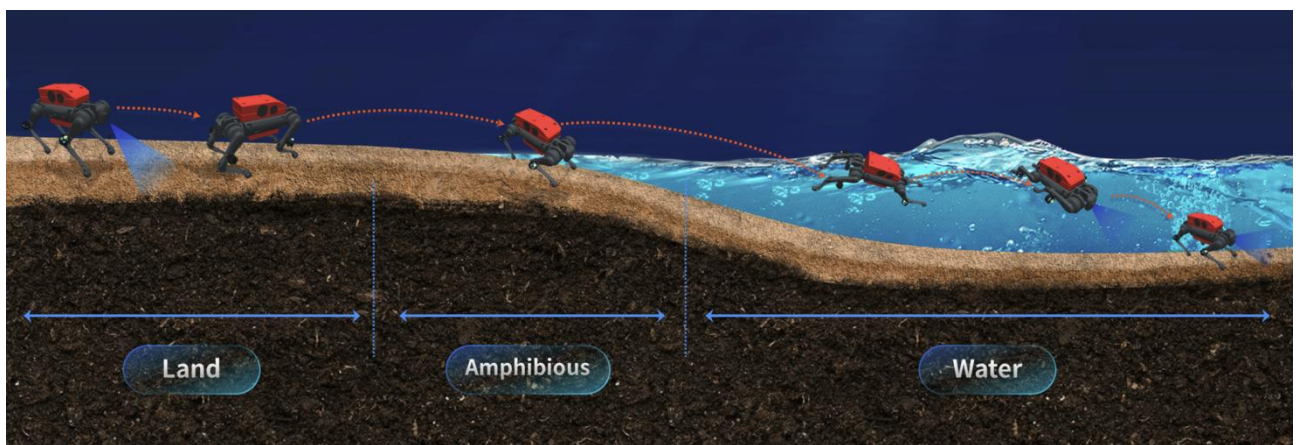


Figure 4.11 Transition Phases of an Amphibious Robot: Land to Water Movement

4.3.1 Advantages of Amphibious Robots in Land-Sea Transitions

Amphibious robots, leveraging the complementary functionality of legs and propellers, exhibit distinct advantages in overcoming challenges associated with land-sea transitions. These advantages enable robust performance in dynamic coastal environments, addressing key scenarios such as bottom currents, coastal waves, and rip currents.

Overcoming Bottom Currents

Bottom currents exert destabilizing horizontal forces, especially over uneven seabeds. Amphibious robots counter these challenges through hybrid locomotion and advanced control. Legs provide traction and stability on irregular surfaces, preventing slippage, while propellers effectively counteract lateral forces, ensuring precise trajectory control.

IMUs and sensors integrated with MPC frameworks predict and adjust for current fluctuations, dynamically coordinating propulsion and leg movement. This synergy minimizes destabilization risks, enabling the robot to maintain trajectory accuracy under varying hydrodynamic pressures.

Resilience Against Coastal Waves

Coastal waves introduce oscillatory forces that can disrupt robot stability and positioning. The use of legs for anchoring and propellers for counteracting horizontal forces ensures consistent movement and vertical stability.

Adaptive control strategies driven by IMUs and force sensors adjust the robot's motion in response to wave dynamics, maintaining precision during operation. By accounting for wave-induced pressure variations, these systems ensure stability and trajectory fidelity even in irregular conditions.

Mitigating Rip Currents

Rip currents pose significant challenges due to their multi-directional forces. To maintain stability, robots employ sensor fusion techniques combining inputs from IMUs and other sensors. This integration enables accurate evaluation of hydrodynamic forces and real-time motion adjustments.

MPC frameworks anticipate and compensate for fluctuations, ensuring stability and precision in complex environments. Legs establish a stable foundation on shifting seabeds, while propellers counteract reverse currents, enabling seamless navigation through turbulent conditions.

4.3.2 Limitations in Current Amphibious Robots and Challenges in Land-Sea Transitions

Despite their advantages, amphibious robots face significant limitations, particularly in simulation platforms and multi-modal control algorithms.

Limited Simulation Capabilities of Experimental Platforms

Existing experimental platforms, such as towing tanks, fail to replicate the full complexity of land-to-water transitions. These platforms are unable to accurately simulate bottom currents, coastal waves, and rip currents, which play a critical role in destabilizing robots.

Moreover, the transition phase, involving overlapping ground reaction forces and buoyancy, remains underexplored in these controlled environments. This lack of realistic testing conditions limits the validation of algorithms and locomotion mechanisms under dynamic, real-world scenarios.

Limitations of Multi-Modal Transition Control Algorithms

Current algorithms excel in distinct terrestrial or aquatic environments but struggle

during transitions. The inability to manage simultaneous changes in ground support and hydrodynamic forces results in instability and inefficiencies.

MPC and similar approaches often lack real-time adaptability and precise synchronization between locomotion modes, leading to energy wastage and reduced operational endurance. The absence of energy-efficient algorithms further exacerbates these challenges, particularly during transitions involving strong currents or waves.

4.3.3 Future Research Directions and Theoretical Optimization Approaches

Addressing the identified limitations requires advancements in experimental platforms and multi-modal control systems, supported by theoretical innovations.

Advanced Experimental Platforms

To overcome current limitations, modular testing platforms incorporating wave generators, sediment tanks, and current simulators are proposed. These platforms can replicate real-world coastal dynamics, allowing researchers to validate algorithms and mechanisms under controlled yet realistic conditions.

Hybrid simulation environments integrating land and water features will enable the study of transitions with realistic terrain and hydrodynamic interactions. Computational models combining CFD and FEA can complement physical testing, providing insights into forces and stability during transitions.

Theoretical Advances in Multi-Modal Transition Control

Hybrid control frameworks integrating legged locomotion and thruster-based propulsion can address dynamic force redistribution during transitions. Bio-inspired

locomotion patterns, modeled through neural networks, offer promising strategies for smooth coordination between modes.

Reinforcement learning-based optimization algorithms can enhance energy efficiency by balancing power allocation during transitions. Integrating data from IMUs, sonar, and visual sensors into a unified control system enables real-time adaptability to changing environments.

Scaling and Validation

Scaled prototypes can accelerate iterative testing, while field trials in natural coastal environments validate the scalability of theoretical models. Computational studies of scaling complexities will identify challenges in large-scale deployment, bridging the gap between theoretical advancements and real-world applications.

Conclusions to Chapter 4

Chapter 4 presented a comprehensive exploration into the development, validation, and implementation of a data-driven hydrodynamic model for an amphibious robot's underwater navigation. Key elements, including GPR-based hydrodynamic modeling, provided insights into refining underwater trajectory tracking accuracy and propulsion efficiency. The integration of advanced sensors, such as IMUs and force sensors, facilitated precise measurements and allowed for in-depth analysis of the robot's response to hydrodynamic forces under various conditions.

Our experiments highlighted several critical findings:

1. **Data-Driven Hydrodynamic Modeling:** Utilizing GPR models significantly enhanced the predictive accuracy of hydrodynamic forces experienced by the robot in water, especially in environments with complex flow dynamics. The data-driven

approach provided robust, adaptive models that improved control fidelity, enabling the robot to maintain stability and maneuverability.

2.MPC for Enhanced Trajectory Control: The predictive capabilities of MPC, paired with GPR-derived hydrodynamic models, allowed for real-time adjustment to the control outputs, effectively compensating for underwater disturbances such as currents and turbulence. This proved especially beneficial for tracking intricate trajectories, such as the 8-shaped pattern, with minimal error, demonstrating MPC's advantage over traditional control algorithms like PID in dynamic underwater conditions.

3.Performance Evaluation across Speed Modes: Analysis revealed that while trajectory tracking accuracy was high at lower speeds, tracking errors increased with higher speeds due to amplified noise, computational demands on the MPC, and intensified hydrodynamic disturbances. This insight underscores the importance of optimizing control strategies for high-speed operations, potentially by incorporating more advanced filtering techniques or adaptive algorithms.

4.Enhanced Robustness through Sensor Fusion: Integrating IMU data with depth sensors enabled reliable odometry, reducing cumulative drift and providing precise localization for trajectory tracking. This robust sensor fusion framework, essential for navigation in complex aquatic environments, was critical in maintaining trajectory accuracy across variable underwater conditions.

5.Challenges in Land-Sea Transitions: While the focus was predominantly on underwater hydrodynamics, a critical area of future development lies in addressing the challenges of land-sea transitions for amphibious robots. This transitional phase introduces unique stability challenges due to the interplay of bottom currents, coastal waves, and bottom return currents. These factors necessitate advanced locomotion

mechanisms and control strategies, leveraging the combined operation of legs and propellers to achieve seamless transitions between land and water environments.

6.Future Directions: The land-sea transition presents an interdisciplinary research challenge, requiring further refinement in robot design, adaptive control systems, and advanced simulation tools. Insights from the current work provide a foundation for future investigations into ensuring stability during transitions, optimizing energy efficiency, and enhancing the overall versatility of amphibious robots for real-world applications.

Conclusions

The evolution of R&D issues has provided significant advancements in the design of robots operating in multiple environments. One of the most challenging yet promising fields is the development of amphibious robots—robots that can transition seamlessly between surface and underwater environments. So, the current dissertation thesis can be considered as some scientific questions that should be overcome in the implementation process and expressed below in the conclusions.

1. The hardware, software, and metrological provision of drones and their launch platforms were collectively and consistently developed and studied. This was based on established metrological and operational characteristics combined with control methods, leading to improved accuracy and reduced uncertainty in the obtained results.
2. To provide real-time control for amphibious robots, the combination of Nvidia Jetson, Pixhawk, ROS, and advanced sensor fusion techniques ensured that these robots could operate efficiently, sometimes meeting the diverse demands of aquatic robotics. The proposed integrated approach develops autonomous, adaptable, and resilient robotic systems capable of addressing complex challenges in water environments.
3. The efficient operation of underwater robots is based on a developed test platform ensuring dynamic performance measurements for repeatable samples and experiments and improving the accuracy, stability, and uniformity of measurements by incorporating advanced control systems and modern methods, for example, Inertial Measurement Units (IMU) and GNS-Methods. The considered platform adapts to the degradation of thruster performance through

adaptive sliding controllers showcasing its superiority over existing methods. The studied proposed test platform paradigm offers a promising approach for underwater dynamics measurement providing more accurate and reliable measurements in various applications for advancing underwater research and technology. Dynamic adaptation (within a sliding mode) based on control obtained by the direct measurements of the output signals of a first-order low-pass filter containing the discontinuous control with the specially adapted magnitude value in the input.

4. In real-time control and navigation systems for underwater robots, filtering techniques play a crucial role in ensuring accurate and stable orientation estimates from sensor data. This study explored two primary filtering approaches—Complementary Filtering and Kalman Filtering—focusing on their application to IMU data processing. Comparative analysis revealed that the complementary filter algorithm exhibited lower noise levels and higher synchronization rates in attitude computation. The complementary filter generates posture information relative to the startup position and dynamically adjusts to changes in orientation. This adaptability ensures robust responses to irregular alterations in the IMU's operational environment, providing stable posture signals with significantly reduced noise. Additionally, the complementary filter demonstrated superior recovery performance following disturbances, highlighting its efficacy in enhancing precision and reliability in dynamic and unpredictable conditions.
5. MPC has proven highly effective for depth control in underwater robots. By leveraging real-time sensor data and predictive modeling, MPC anticipates system behavior and environmental changes, enabling precise adjustments to

maintain target depth. Its capability to predict disturbances and dynamically update control inputs ensures stability even under varying hydrodynamic forces, such as currents or turbulence. Moreover, MPC's flexibility allows for the integration of multiple constraints, optimizing both energy efficiency and response speed. This makes it an indispensable approach for achieving reliable depth regulation in complex underwater environments.

6. The second application of novel machine learning methodology in the considered work is to predict water levels, relying on data collected from force sensors and kinematic observations during the robot's underwater testing. GPR (Ground Penetrating Radar) is a non-parametric, probabilistic model used to learn the dynamics of a system by capturing the relationships between input and output data. GPR is particularly effective for modeling continuous-time changes in forces as a function of sensor inputs while providing uncertainty estimates. The model defines a distribution over functions and uses training data to update this distribution. The core of GPR lies in predicting a continuous function based on a set of observed data points. In scenarios with noisy conditions, the GPR-Matern model demonstrates significantly better performance, with errors as low as 4.0, making it highly suitable for deployment in real-world underwater environments where sensor noise is prevalent.
7. Experimental verification of the current provisions of paragraphs 1-6, which in particular related to the filtration results and dynamics of the environmental impact confirmed the correctness of the involvement of several novel machine learning methods which enhance complex implementation of hardware, software, and metrological support at the design stage of drones

8. **Future Challenges and Transition Dynamics:** While significant progress has been achieved in underwater trajectory tracking and control, the transition between land and water environments presents unique challenges. Addressing the stability of amphibious robots during land-sea transitions is critical for future research. This includes counteracting the effects of bottom currents, coastal waves, and bottom return currents. Such transitions require the simultaneous and coordinated operation of legs and thrusters, alongside the development of advanced adaptive control strategies. Overcoming these challenges will significantly enhance the robot's versatility, enabling it to operate seamlessly across diverse environmental conditions.

REFERENCES

- [1] William Thomson (Lord Kelvin). "On the Dynamical Theory of Heat." *Philosophical Transactions of the Royal Society of London*, vol. 141, 1851, pp. 357–385. DOI: <https://doi.org/10.1098/rstl.1851.0022>.
- [2] Stadnyk, B., Yatsyshyn, S. (Eds.) *Cyber-Physical Systems and Metrology 4.0*. IFSA Publishing, S.L., 2021, 332 pages. ISBN: 840926899X, 9788409268993.
- [3] Yatsyshyn, S., Stadnyk, B. *Cyber-Physical Systems: Metrological Issues*. IFSA Publishing, S.L., 2016, 328 pages. ISBN: 9788460899624, 8460899624.
- [4] R. M. Alexander, "Exploring biomechanics. animals in motion" Scientific American Library, New York, 1992. <https://www.amazon.com/Exploring-Biomechanics-R-McNeill-Alexander/dp/071675035X>
- [5] S. Yatsyshyn, A. Cherkas, X. Zeng, Hardware and software of water strider robot, International Scientific and Practical Conference IVT-2022, Lviv, Ukraine, 09-10 November 2022, pp. 146–147.
- [6] S. Yatsyshyn, X. Zeng, "Design of the Water Strider-like Robot", *Measuring Equipment and Metrology*, Volume 84, Number 3, pp. 39-42, 2023. <https://doi.org/10.23939/istcmtm2023.03.039>
- [7] K. Schmidt-Nielsen, "Scaling: why is animal size so important?" Press Syndicate of the University of Cambridge, 1984. <https://www.amazon.com/Scaling-Why-Animal-Size-Important-ebook/dp/B00E3URD1C>
- [8] D. L. Hu, B. Chan, J. W. M. Bush, "The hydrodynamics of water strider locomotion," *Nature*, vol. 424, no. 6949, pp. 663–666, 2003. <https://doi.org/10.1038/nature01862>

- [9] J. W. M. Bush, D. L. Hu, "Walking on water: biolocomotion at the interface," *Annual Review of Fluid Mechanics*, vol. 38, no. 1, pp. 339–369, 2006. <https://doi.org/10.1146/annurev.fluid.37.061903.175725>
- [10] Jing-Ze Ma, Hong-Yu Lu, Xiao-Song Li, and Yu Tian, Interfacial phenomena of water striders on water surfaces: a review from biology to biomechanics, *Zool Res.* 2020 May 18; 41(3): 231–246. doi: 10.24272/j.issn.2095-8137.2020.029
- [11] Y. Ding and H.-W. Park, "Design and experimental implementation of a quasi-direct-drive leg for optimized jumping" International Conference on Intelligent Robots and Systems (IROS), 2017. DOI:10.1109/IROS.2017.8202172
- [12] D. Tian, J. Gao, X. Shi, Y. Lu, and C. Liu, "Vertical jumping for legged robot based on quadratic programming" *Sensors*, vol. 21, 2021. <https://www.mdpi.com/1424-8220/21/11/3679>
- [13] N. P. Linthorne, "Analysis of standing vertical jumps using a force platform" *American Journal of Physics*, vol. 69, pp. 1198–1204, 2001. [https://www.brunel.ac.uk/~spstnpl/Publications/VerticalJump\(Linthorne\).pdf](https://www.brunel.ac.uk/~spstnpl/Publications/VerticalJump(Linthorne).pdf)
- [14] Yo. Kim, Yi. Yang, X. Zhang et al, Remote control of muscle-driven miniature robots with battery-free wireless optoelectronics, *Sc. Robot*, Vol.8, No.74, eadd1053, 2023, 18 Jan. 2023, DOI: 10.1126/scirobotics.add1053. <https://pubmed.ncbi.nlm.nih.gov/36652505/>
- [15] Chaplia, O., Klym, H. "Node.js Project Architecture with Shared Dependencies for Microservices." *Measuring Equipment and Metrology*, 2023, 84(3), pp. 53–58. <https://doi.org/10.23939/istcmtm2023.03.053>.
- [16] Chaplia, O., Klym, H. "Microservice Architecture for Cyber-Physical Systems." *Computer Systems and Information Technologies*, 2024, (2), pp. 242–250. <https://doi.org/10.35546/kntu2078-4481.2024.2.34>.

- [17] Chaplia, O., Klym, H., Popov, A.I. "An Approach to Improving Availability of Microservices for Cyber-Physical Systems." *Advances in Cyber-Physical Systems*, 2024, 9(1), pp. 16–23. <https://doi.org/10.23939/acps2024.01.016>.
- [18] Stepanov, O., Klym, H. "Features of the Implementation of Micro-Interfaces in Information Systems." *Advances in Cyber-Physical Systems*, 2024, 9(1), pp. 54–60. <https://doi.org/10.23939/acps2024.01.054>.
- [19] S. Yatsyshyn, X. Zeng, Metrological risks at design stage for multidisciplinary-based objects, 60th Ilmenau Scientific Colloquium "Engineering for a Changing World", Technische Universität Ilmenau, September 04–08, 2023, pp. 58677-1–58677-7.
- [20] X. Zeng, S. Yatsyshyn, Test platform paradigm for underwater object's measurements, VI International Scientific and Practical Conference "Quality Management in Education and Industry: Experience, Problems, and Perspectives", Lviv, Ukraine, 16–17 November 2023, pp. 157–158.
- [21] X. Zeng, S. Yatsyshyn, "Test Platform Paradigm for Underwater Dynamics Measurements", *Measuring Equipment and Metrology*, Volume 85, Number 1, pp. 29-34, 2024. <https://doi.org/10.23939/istcmtm2024.01.029H>.
- [22] Øveraas, Dynamic Positioning Using Model Predictive Control With Short-Term Wave Prediction”, 2023, Department of Engineering Cybernetics, Norwegian University of Science and Technology, Trondheim, Norway, DOI: 10.1109/JOE.2023.3288969
- [23] M. N. Bandyopadhyay, “Position Control System of A PMDC Motor”. Department of Electrical Engineering, Kolkata, West Bengal, India 2016, DOI: 10.1109/ICEEOT.2016.7754785

- [24] D.R. Yoerger, "The Influence of Thruster Dynamics on Underwater Vehicle Behavior and Their Incorporation into Control System Design", Deep Submergence Laboratory, Department of Applied Physics and Ocean Engineering, Woods Hole Oceanographic Institution, Woods Hole, MA, USA, 1990. DOI: 10.1109/48.107145
- [25] M. Gertler and G. R. Hagen, "Standard equations of motion for submarine simulations" Naval Ship R&D Center, Bethesda, MD, NSRDC Rep. No. 2510, 1967. [On-line]. Available: <https://apps.dtic.mil/sti/citations/AD0653861>
- [26] Kukharchuk, V. V., Hraniak, V. F., Katsyv, S. Sh., Holodyuk, V. S. "Torque Measuring Channels: Dynamic and Static Metrological Characteristics." Informatyka, Automatyka, Pomiar w Gospodarce i Ochronie Środowiska, 2020, No. 3, pp. 82–85. <https://doi.org/10.35784/iapgos.2080>. .
- [27] Kupriyanov, O., Trishch, R., Dichev, D., Hrinchenko, H. "Experimental Studies on the Form Error Effect of the Part Mounting Surface on the Strength Quality Parameter of the Interference Fit Joints." In: Tonkonogyi, V., Ivanov, V., Trojanowska, J., Oborskyi, G., Pavlenko, I. (eds) Advanced Manufacturing Processes V. InterPartner 2023. Lecture Notes in Mechanical Engineering. Springer, Cham, 2024. https://doi.org/10.1007/978-3-031-42778-7_34.
- [28] Bubela, T., Kochan, R., Więclaw, Ł., Yatsuk, V., Kuts, V., Yatsuk, Y. "Disassembly-Free Metrological Control of Analog-to-Digital Converter Parameters." Metrology and Measurement Systems, 2022, Vol. 29, Issue 4, pp. 669–684. <https://doi.org/10.1515/mms-2022-0039>. (Indexed in SciVerse SCOPUS, Web of Science).
- [29] Hrinchenko, H., Koval, V., Shmygol, N., Sydorov, O., Tsimoshynska, O., Matuszewska, D. "Approaches to Sustainable Energy Management in Ensuring

- Safety of Power Equipment Operation." *Energies* 2023, 16, 6488.
<https://doi.org/10.3390/en16186488>.
- [30] Trishch, R., Nechviviter, O., Hrinchenko, H., Bubela, T., Riabchykov, M., Pandova, I. "Assessment of Safety Risks Using Qualimetric Methods." *MM Science Journal*, October 2023, 6668.
https://doi.org/10.17973/MMSJ.2023_10_2023021.
- [31] Z. Wang, Y. Yan, X. Zeng, R. Li, W. Cui, Y. Liang, D. Fan, Joint multi-objective optimization based on multitask and multi-fidelity Gaussian processes for flapping foil, *Ocean Engineering*, Volume 294, 15 February 2024, 116862.<https://doi.org/10.1016/j.oceaneng.2024.116862>.
- [32] Z. Liang, "Dynamic Analysis and Path Planning of a Turtle-Inspired Amphibious Spherical Robot", School of Electronic Information Science and Technology, China, 2022 [On-line]. Available:
<https://www.mdpi.com/2072-666X/13/12/2130#>
- [33] A. J. Healey, "Toward an Improved Understanding of Thruster Dynamics for Underwater Vehicles", Naval Postgraduate School, Department of Mechanical Engineering, Monterey CA, 1994. DOI: 10.1109/48.468242
- [34] Q. Liu, H. Chen, P. Guo, G. Su, W. Li, X. Zeng, D. Fan, W. Cui, Unified scheme design and control optimization of flapping wing for next-generation manta ray robot, *Ocean Engineering*, Volume 309, Part 2, 1 October 2024, 118487.<https://doi.org/10.1016/j.oceaneng.2024.118487>.
- [35] Y Sun, "Experimental and numerical analyses of the hydrodynamic performance of propeller boss cap fins in a propeller-rudder system", Science and Technology on Underwater Vehicle Laboratory, Harbin Engineering University,

- China, 2016. [On-line]. Available: <https://www.tandfonline.com/doi/full/10.1080/19942060.2015.1121838>.
- [36] M. Abkowitz, "Stability and Motion Control of Ocean Vehicles", Cambridge, MA: MIT Press, 1969. [On-line]. Available: <https://pdfcoffee.com/abkowitz-stability-and-motion-control-of-ocean-vehicles-pdf-free.html>
- [37] Cody, S. E., "An Experimental Study of The Response of Small Thrusters to Step and Triangular Wave Inputs", Monterey, CA, 1992. [On-line]. Available: <https://www.sciencedirect.com/science/article/pii/S1474667017371008>
- [38] Adams, J.C., Burton, D., Lee, M., "Dynamic Characterization and Control of Thrusters for Underwater Vehicles", 1991.
- [39] Brown, J. P., "Four Quadrant Model of the NPS AUV 11 Thruster" Monterey, CA, 1993. [On-line]. Available: <https://core.ac.uk/download/pdf/36719905.pdf>
- [40] Liu, Q., Chen, H., Wang, Z., He, Q., Chen, L., Li, W., Li, R., Cui, W. "A Manta Ray Robot with Soft Material Based Flapping Wing." *Journal of Marine Science and Engineering*, 2022, 10(7), 962. <https://doi.org/10.3390/jmse10070962>.
- [41] D. Graham, D. McRuer, "Analysis of Nonlinear Control Systems", New York: Wiley, 1961. [On-line]. Available: <https://scholar.google.com.ua/scholar?q=D.+Graham,+D.+McRuer>
- [42] X. Zeng, S. Yatsyshyn, The exactness of ultrasound sensors of robotics, II International Scientific and Practical Conference "Information and Measurement Technologies IVT-2024", Lviv, Ukraine, 13–14 November 2024, pp. 139–140.
- [43] Klym, H., Diachok, R. "Dynamic Search for Errors in Industrial Internet Protocols for Application in Multisensor Control Systems." *Computer Systems*

and Information Technologies, 2022, (3), pp. 65–74.

<https://doi.org/10.31891/csit-2022-3-9>.

- [44] Граняк В. Ф., Кухарчук В. В., Кучерук В. Ю., Каців С. Ш., Карабекова Д. Ж., Хассенов А. К. "Математична модель ємнісного мікромеханічного акселерометра в статичному та динамічному режимах роботи." Вісник Карагандинського університету. Серія «Фізика», 2020, No. 2, pp. 60–67.
- [45] Кухарчук В. В., Голодюк В. С., Каців С. Ш., Павлов С. В., та ін. "Особливості динамічних вимірювань кутових швидкостей з використанням енкодера." ІАПГОСЬ, 2022, No. 3, pp. 20–26.
<http://doi.org/10.35784/iapgos.3035>.
- [46] A. Norhafizan, G. Raja, N. Khairi, Reviews on Various Inertial Measurement Unit, International Journal of Signal Processing Systems Vol. 1, No. 2 December 2013, pp.256-261. <https://d1wqtxts1xzle7.cloudfront.net/89189534/>
- [47] Kukharchuk, V. V., Holodiuk, V. S., Katsyv, S. Sh., Pavlov, S. V. "Features of the Angular Speed Dynamic Measurements with the Use of an Encoder." IAPGOŚ, 2022, No. 3, pp. 20–26. <http://doi.org/10.35784/iapgos.3035>.
- [48] Á. Revuelta. Orientation estimation and movement, Master's Thesis in Electrical Engineering with emphasis in Signal Processing, 2017, Department of Applied Signal Processing, Blekinge Institute of Technology, SE–371 79 Karlskrona, Sweden.
<https://www.diva-portal.org/smash/get/diva2:1127455/FULLTEXT02.pdf>
- [49] X. Zeng, Olha Lysa, "Response Time in Inertial Measurement Unit Control Algorithms", Measuring Equipment and Metrology, Volume 85, Number 2, pp. 5-8, 2024.<https://doi.org/10.23939/istcmtm2024.02.005>

- [50] R. Meinhold, N. Singpurwalla, Understanding the Kalman filter, *American Statistician*, May 1983, Vol.37, No.2, pp.123-127,
<http://www-stat.wharton.upenn.edu/~steele/Resources/FTSResources/StateSpaceModels/KFExposition/MeinSing83.pdf>
- [51] P. Gui, L. Tang and S. Mukhopadhyay, "MEMS based IMU for tilting measurement: Comparison of complementary and kalman filter based data fusion," 2015 IEEE 10th Conf. on Industr. Electronics and Appl. (ICIEA), Auckland, New Zealand, 2015, pp. 2004-2009, doi: 10.1109/ICIEA.2015.7334442
- [52] L. Kleeman, Understanding and Applying Kalman Filtering, Department of Electrical and Computer Systems Engineering Monash University, Clayton,
https://www.cs.cmu.edu/~motionplanning/papers/sbp_papers/kalman/kleeman_understanding_kalman.pdf
- [53] J. Wu, Z. Zhou, J. Chen, R. Li, Fast Complementary Filter for Attitude Estimation Using Low-Cost MARG Sensors, , *IEEE Sensors Journal* 16(18):1-0,1 Sept. 2016, DOI:10.1109/JSEN.2016.2589660
- [54] Azis, F.A, Aras, M. S. M, Rashid, M.Z.A, Othman M.N, Abdullah S.S., Problem Identification for Underwater Remotely Operated Vehicle (ROV): A Case Study, *Procedia Engineering* 41 (2012) 554 – 560, 1877-7058
- [55] Int. Symp. on Robotics and Intel. Sensors, 2012 (IRIS 2012) doi: 10.1016/j.proeng.2012.07.211,
<https://pdf.sciencedirectassets.com/278653/1-s2.0-S1877705812X00213/1-s2.0-S1877705812026112/main.pdf?X-Amz-Security->
- [56] Li, H., Wang, Z., and Xu, S., "Optimizing Fish-Inspired Robotic Systems for Swimming Using a Mathematical Model of Fluid and Robot Dynamics," *IEEE*

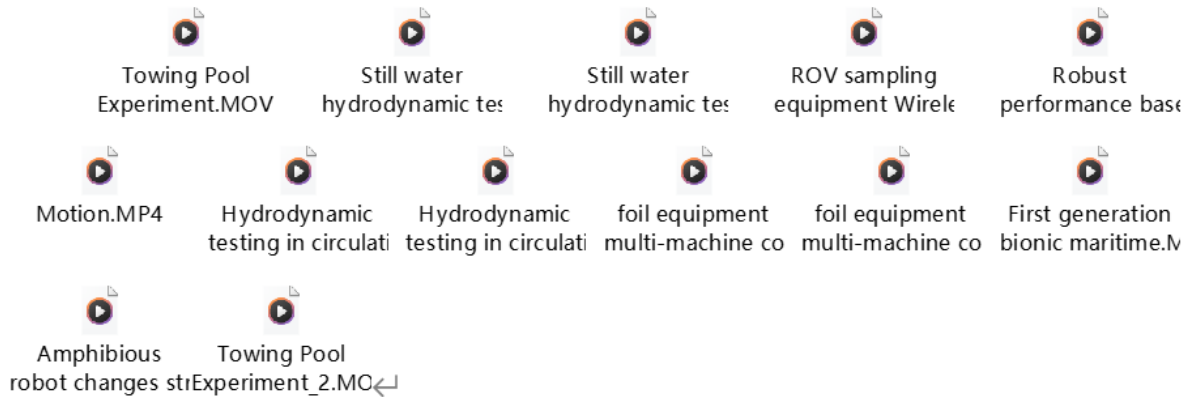
- Transactions on Robotics, vol. 37, no. 4, pp. 1125-1137, 2021. [Online]. Available: <https://doi.org/10.1109/TRO.2021.3062105>
- [57] S. Yatsyshyn, X. Zeng, Adaptive modeling of underwater robot fluid dynamics based on force measurement device, *Measuring Equipment and Metrology*, Volume 85, Number 4, pp. 7-13, 2024. <https://doi.org/10.23939/istcmtm2024.04.007>
- [58] Fan, X., Li, J., and Zhang, Y., "Investigating Fluid-Structure Dynamics Using an Intelligent Towing Tank for Complex Real-Time Simulations," *Journal of Fluid Mechanics*, vol. 873, pp. 432-458, 2019. [Online]. Available: <https://doi.org/10.1017/jfm.2019.100>
- [59] Chenyi, W., Liu, Y., and Patel, V., "Recent Advances in Gaussian Process Regression: A Review," *IEEE Access*, vol. 12, pp. 12345-12368, 2024. [Online]. Available: <https://doi.org/10.1109/ACCESS.2024.1234567>
- [60] S. B. Ramezani et al., "Scalability, Explainability and Performance of Data-Driven Algorithms in Predicting the Remaining Useful Life: A Comprehensive Review," *IEEE Access*, vol. 11, pp. 41741-41769, 2023. [Online]. Available: <https://doi.org/10.1109/ACCESS.2023.3267960>.
- [61] H. X. Zhou et al., "Gaussian Process Regression for High-Dimensional Time Series Prediction: A Review and Application," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 6, pp. 2459-2471, 2023. [Online]. Available: <https://doi.org/10.1109/TNNLS.2023.3198745>
- [62] K. Y. Chen et al., "Efficient Variational Inference for Large-Scale Gaussian Process Regression," *Journal of Machine Learning Research*, vol. 24, no. 11, pp. 1-30, 2023. [Online]. Available: <https://www.jmlr.org/papers/volume24/chen23a/chen23a.pdf>

- [63] M. S. Sharma et al., "Scalable Gaussian Process Regression Using Low-Rank Approximations," *Artificial Intelligence Review*, vol. 56, no. 3, pp. 317-339, 2022. [Online]. Available: <https://doi.org/10.1007/s10462-021-09947-0>
- [64] L. J. Li et al., "Applications of Gaussian Process Regression in Robotics and Autonomous Systems: A Survey," *IEEE Robotics and Automation Letters*, vol. 8, no. 1, pp. 183-190, 2023. [Online]. Available: <https://doi.org/10.1109/LRA.2022.3214554>
- [65] S. T. Kumar et al., "Controlled Pool Experiments for Robust Hydrodynamic Force Prediction in Marine Robotics," *Journal of Marine Science and Engineering*, vol. 11, no. 5, pp. 1423-1436, 2023. [Online]. Available: <https://doi.org/10.3390/jmse11051423>
- [66] J. C. Lee et al., "Experimental Setup and Data Collection for Hydrodynamic Force Modeling of Underwater Robots," *IEEE Access*, vol. 10, pp. 12345-12356, 2022. [Online]. Available: <https://doi.org/10.1109/ACCESS.2022.3181234>
- [67] R. M. Davis et al., "High-Precision Force Sensing and Data Acquisition for Underwater Robot Testing," *IEEE Transactions on Instrumentation and Measurement*, vol. 72, pp. 1237-1248, 2023. [Online]. Available: <https://doi.org/10.1109/TIM.2023.3156789>
- [68] Li, T., Yang, X., & Chen, Y., "Hydrodynamic force analysis for amphibious robotic transitions in coastal zones," *Journal of Fluid Mechanics*, vol. 893, no. 3, pp. 112-135, 2021. [Online]. Available: <https://doi.org/10.1017/jfm.2021.45>
- [69] Kim, J., & Cho, K., "Hybrid locomotion systems for amphibious robots: Challenges and approaches," *Bioinspiration & Biomimetics*, vol. 14, no. 2, pp. 1-12, 2019. [Online]. Available: <https://doi.org/10.1088/1748-3190/ab13f9>

- [70] Li, H., Wang, Z., and Xu, S., "Optimizing Fish-Inspired Robotic Systems for Swimming Using a Mathematical Model of Fluid and Robot Dynamics," *IEEE Transactions on Robotics*, vol. 37, no. 4, pp. 1125-1137, 2021. [Online]. Available: <https://doi.org/10.1109/TRO.2021.3062105>
- [71] Kim, S., Epps, B. P., and Zhang, F., "Bio-inspired Amphibious Robots: Challenges and Opportunities," *Journal of Field Robotics*, vol. 36, no. 2, pp. 489-512, 2020. [Online]. Available: <https://doi.org/10.1002/rob.21901>
- [72] Zhao, Y., Zhang, H., and Liu, C., "Development of a Versatile Amphibious Robotic Test Platform for Coastal Operations," *Ocean Engineering*, vol. 223, pp. 108395, 2021. [Online]. Available: <https://doi.org/10.1016/j.oceaneng.2021.108395>

APPENDIXES

Demo



Paper



Code

1.Robot underlying control configuration

```

1. #!/usr/bin/env python3
2.
3. #####rosrun 中防止引用自定义类 无法找到#####
4. import sys
5. import os
6. # 确保当前脚本目录在 sys.path 中
7. script_dir = os.path.dirname(__file__)
8. if script_dir not in sys.path:
9.     sys.path.append(script_dir)
10.
11.
12. import os
13. import threading
14. import time
15. from ctypes import *
16. import ctypes
17. from threading import Thread
18. import csv
    
```

```
19. import binascii
20. import datetime
21. import rospy
22. from std_msgs.msg import UInt8 # 用于接收 ID
23. from motion_control.msg import MotorCommandMsg
24. from motion_control.msg import PropellerCommandMsg
25. from std_msgs.msg import String
26. from check_ethernet import NetworkInterfaceChecker
27.
28.
29. VCI_USBCAN2 = 4
30. STATUS_OK = 1
31. CAN_POS = 0 # 0: CAN1 1: CAN2
32. # 0 表示左旋, 1 表示右旋
33. vis = [0, 0, 1, 1, 1, 1, 1, 1]
34. ID = [0x0360, 0x0361, 0x035F, 0x0364, 0x0313, 0x0314, 0x303, 0x304]
35.
36.
37. class VCI_INIT_CONFIG(Structure):
38.     _fields_ = [("AccCode", c_uint),
39.                 ("AccMask", c_uint),
40.                 ("Reserved", c_uint),
41.                 ("Filter", c_ubyte),
42.                 ("Timing0", c_ubyte),
43.                 ("Timing1", c_ubyte),
44.                 ("Mode", c_ubyte)]
45.
46.
47. class VCI_CAN_OBJ(Structure):
48.     _fields_ = [("ID", c_uint),
49.                 ("TimeStamp", c_uint),
50.                 ("TimeFlag", c_ubyte),
51.                 ("SendType", c_ubyte),
52.                 ("RemoteFlag", c_ubyte),
53.                 ("ExternFlag", c_ubyte),
54.                 ("DataLen", c_ubyte),
55.                 ("Data", c_ubyte * 8),
56.                 ("Reserved", c_ubyte * 3)]
```

```
57.
58.
59.
60.# Construct the path to the shared Library
61.lib_path = os.path.join(script_dir, '..', 'lib', 'arm_libcontrolcan.so'
    )
62.
63.# Load the shared Library
64.canDLL = cdll.LoadLibrary(lib_path)
65.
66.# CanDLLName = 'devel/lib/libcontrolcan.so' # 把DLL 放到对应的目录下
67.# lib_path = os.getcwd() + "\\function\\libs\\ControlCAN.dll"
68.# # canDLL = windll.LoadLibrary('../libs/ControlCAN.dll')
69.# canDLL = ctypes.cdll.LoadLibrary(CanDLLName)
70.
71.ret = canDLL.VCI_OpenDevice(VCI_USBCAN2, 0, 0)
72.if ret == STATUS_OK:
73.    print('调用 VCI_OpenDevice 成功\r\n')
74.else:
75.    print('调用 VCI_OpenDevice 出错\r\n')
76.
77.# 初始0 通道
78.vci_initconfig = VCI_INIT_CONFIG(0x80000000, 0xFFFFFFFF, 0, 0, 0x00, 0x
    1C, 0) # 波特率500k, 正常模式
79.ret = canDLL.VCI_InitCAN(VCI_USBCAN2, 0, 0, byref(vci_initconfig))
80.if ret == STATUS_OK:
81.    print('调用 VCI_InitCAN1 成功\r\n')
82.else:
83.    print('调用 VCI_InitCAN1 出错\r\n')
84.
85.ret = canDLL.VCI_StartCAN(VCI_USBCAN2, 0, 0)
86.if ret == STATUS_OK:
87.    print('调用 VCI_StartCAN1 成功\r\n')
88.else:
89.    print('调用 VCI_StartCAN1 出错\r\n')
90.
91.# 初始1 通道
92.ret = canDLL.VCI_InitCAN(VCI_USBCAN2, 0, 1, byref(vci_initconfig))
```

```
93. if ret == STATUS_OK:
94.     print('调用 VCI_InitCAN2 成功\r\n')
95. else:
96.     print('调用 VCI_InitCAN2 出错\r\n')
97.
98. ret = canDLL.VCI_StartCAN(VCI_USBCAN2, 0, 1)
99. if ret == STATUS_OK:
100.     print('调用 VCI_StartCAN2 成功\r\n')
101. else:
102.     print('调用 VCI_StartCAN2 出错\r\n')
103.
104. # 接收结构体数组类
105. class VCI_CAN_OBJ_ARRAY(Structure):
106.     _fields_ = [('SIZE', ctypes.c_uint16), ('STRUCT_ARRAY', ctypes.P
        OINTER(VCI_CAN_OBJ))]
107.
108.     def __init__(self, num_of_structs):
109.         self.STRUCT_ARRAY = ctypes.cast((VCI_CAN_OBJ * num_of_struct
        s)(), ctypes.POINTER(VCI_CAN_OBJ)) # 结构体数组
110.         self.SIZE = num_of_structs # 结构体长度
111.         self.ADDR = self.STRUCT_ARRAY[0] # 结构体数组地址 byref()转c
        地址
112.
113.
114.     def getRequest(form):
115.         ubyte_array = c_ubyte * 8
116.         if form == 11: # 查故障
117.             return ubyte_array(0x45, 0x46, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00)
118.         elif form == 12: # 查速度
119.             return ubyte_array(0x51, 0x56, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00)
120.         elif form == 13: # 查电流
121.             return ubyte_array(0x51, 0x43, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00)
122.         elif form == 14: # 查温度
123.             return ubyte_array(0x51, 0x54, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00)
```

```
124.
125.
126. # 获取速度的命令数据
127. def getVelocityM(v):
128.     ubyte_array = c_ubyte * 8
129.     if v >= 0:
130.         return ubyte_array(0x54, 0x43, 0x00, 0x00, 0x00, 0x00, 0x00,
            v & 0xFF)
131.     else:
132.         return ubyte_array(0x54, 0x43, 0x00, 0x00, 0xFF, 0xFF, 0xFF,
            v & 0xFF)
133.
134. def getDataM(pid, v):
135.     if (pid >= 0) and (pid <= 7):
136.         if vis[pid] == 0: # 左旋
137.             return getVelocityM(-v)
138.         else: # 右旋
139.             return getVelocityM(v)
140.
141. def getDataP(pid, v):
142.     ubyte_array = c_ubyte * 8
143.     if (pid >= 0) and (pid <= 7):
144.         if vis[pid] == 0: # 左旋
145.             return ubyte_array(0x56, 0x43, 0x00, 0x00, 0x00, 0x00, (
            -v >> 8) & 0xFF, -v & 0xFF)
146.         else: # 右旋
147.             return ubyte_array(0x56, 0x43, 0x00, 0x00, 0x00, 0x00, (
            v >> 8) & 0xFF, v & 0xFF)
148.
149.
150. # 查询信息: 故障查询、速度查询、电流查询、温度查询
151. def getForm(form):
152.     if (form >= 11) and (form <= 14):
153.         return getRequest(form)
154.
155.
156. def PrintCommand(vci_can_obj, output=1):
157.     # 打印每个字段的16进制值
```

```
158.     if output:
159.         print("ID:", hex(vci_can_obj.ID))
160.         print("TimeStamp:", hex(vci_can_obj.TimeStamp))
161.         print("TimeFlag:", hex(vci_can_obj.TimeFlag))
162.         print("SendType:", hex(vci_can_obj.SendType))
163.         print("RemoteFlag:", hex(vci_can_obj.RemoteFlag))
164.         print("ExternFlag:", hex(vci_can_obj.ExternFlag))
165.         print("DataLen:", hex(vci_can_obj.DataLen))
166.         # 将列表转换为 bytes 对象
167.         Data_bytes = bytes(vci_can_obj.Data)
168.         # 使用 binascii.hexlify 将字节转换为16进制字符串
169.         Data_hex = binascii.hexlify(Data_bytes).decode('utf-8')
170.         print("DATA", Data_hex)
171.         print("Reserved:", list(vci_can_obj.Reserved))
172.         return hex(vci_can_obj.ID), Data_hex
173.
174.
175. # 通道1 发送数据, 通道2 接收数据: pid 表示哪个螺旋桨 (1-7), form 表示哪种信
    息查询: 故障查询、速度查询、电流查询、温度查询 (11-14)
176. def sendForm(pid, form):
177.     a = getForm(form)
178.     ubyte_3array = c_ubyte * 3
179.     b = ubyte_3array(0, 0, 0)
180.     # 向pid 螺旋桨发送速度为v 的命令
181.     vci_can_obj = VCI_CAN_OBJ(ID[pid], 0, 0, 1, 0, 0, 8, a, b) # 单
        次发送
182.
183.     res = canDLL.VCI_Transmit(VCI_USBCAN2, 0, CAN_POS, byref(vci_can
        _obj), 1)
184.     time.sleep(0.1)
185.     if res == STATUS_OK:
186.         print('CAN1 通道发送成功\r\n')
187.     else:
188.         print('CAN1 通道发送失败\r\n')
189.     time.sleep(1)
190.     return read_last_csv_data()
191.
192.
```

```
193. def read_last_csv_data(filename='Actuatoredata.csv'):
194.     last_id = None
195.     last_data = None
196.     # 打开 CSV 文件用于读取
197.     with open(filename, 'r', newline='') as csvfile:
198.         csvreader = csv.reader(csvfile)
199.         # 遍历 CSV 文件中的每一行
200.         for row in csvreader:
201.             # 假设第一列是 ID, 其余列是 Data
202.             last_id = row[1] # 保存 ID
203.             last_data = row[2:] # 保存除了 ID 之外的所有数据
204.     return last_id, last_data
205.
206.
207. # 将传入螺旋桨 pid 对应的螺旋桨停止
208. def StopPropeller(pids):
209.     for i in pids:
210.         handle_command(i, 0)
211.         handle_command(i, 0)
212.     # closeCanDLL()
213.
214.
215. # 关闭通道
216. def closeCanDLL():
217.     canDLL.VCI_CloseDevice(VCI_USBCAN2, 0)
218.
219.
220. def motor_command_callback(data):
221.     handle_command(data, 'motor')
222.
223. def propeller_command_callback(data):
224.     handle_command(data, 'propeller')
225.
226. def handle_command(data, device_type):
227.     pid = data.ID
228.     v = data.command # 假设 order 消息中包含速度命令
229.     if device_type == 'motor':
230.         # 处理电机命令
```

```
231.         a = getDataM(pid, v)
232.         elif device_type == 'propeller':
233.             a = getDataP(pid, v)
234.         else:
235.             rospy.logerr("Unknown device type")
236.             return False
237.         ubyte_3array = c_ubyte * 3
238.         b = ubyte_3array(0, 0, 0)
239.         # 向pid 螺旋桨发送速度为v 的命令
240.         if device_type == 'propeller':
241.             vci_can_obj = VCI_CAN_OBJ(ID[pid], 0, 0, 1, 0, 0, 8, a, b)
242.             # 单次发送
243.             if device_type == 'motor':
244.                 vci_can_obj = VCI_CAN_OBJ(ID[pid] , 0, 0, 1, 0, 0, 8, a, b)
245.                 # 单次发送
246.                 # time.sleep(0.1)
247.                 # vci_can_obj = VCI_CAN_OBJ(0x0300 | 4, 0, 0, 1, 0, 0, 8, a,
248.                 b) # 单次发送
249.                 PrintCommand(vci_can_obj)
250.                 res = canDLL.VCI_Transmit(VCI_USBCAN2, 0, CAN_POS, byref(vci_can
251.                 _obj), 1)
252.                 time.sleep(0.1)
253.                 if res == STATUS_OK:
254.                     print('CAN1 通道发送成功\r\n')
255.                 else:
256.                     print('CAN1 通道发送失败\r\n')
257.                 return False
258.             return True
259.
260. def actuation_node():
261.     # 初始化 ROS 节点
262.     rospy.init_node('actuation_node', anonymous=True)
263.
264.     # 创建 Publisher 对象, 用于向电机和螺旋桨发送命令
265.     # 这里使用两个 Publisher, 实际使用时根据需要创建
266.     # motor_pub = rospy.Publisher('motor_commands', MotorCommandMsg,
267.     queue_size=10)
```



```
263.     # propeller_pub = rospy.Publisher('propeller_commands', Propelle
      rCommandMsg, queue_size=10)
264.
265.     # 订阅电机命令主题
266.     rospy.Subscriber('motor_commands', MotorCommandMsg, motor_comman
      d_callback)
267.     # 订阅螺旋桨命令主题
268.     rospy.Subscriber('propeller_commands', PropellerCommandMsg, prop
      eller_command_callback)
269.
270.     # 保持节点运行
271.     rospy.spin()
272.
273.
274. def receiveData():
275.     global stop_receiving
276.     open('Actuatoredata.csv', 'a').close()
277.     while not rospy.is_shutdown():
278.         time.sleep(0.03)
279.         rx_vci_can_obj = VCI_CAN_OBJ_ARRAY(2500) # 结构体数组
280.         res = canDLL.VCI_Receive(VCI_USBCAN2, 0, CAN_POS, byref(rx_v
      ci_can_obj.ADDR), 2500, 0)
281.         if res > 0: # 接收到一帧数据
282.             with open('Actuatoredata.csv', 'a', newline='') as csvfil
      e:
283.                 csvwrite = csv.writer(csvfile)
284.                 print('接收成功\r\n')
285.
286.                 # 将数据写入 CSV 文件
287.                 current_time = datetime.datetime.now().strftime('%Y-
      %m-%d %H:%M:%S')
288.                 ID, can_msg = PrintCommand(rx_vci_can_obj.ADDR)
289.                 csvwrite.writerow([current_time, ID, can_msg])
290.                 # 发布 CAN 消息数据
291.                 pub = rospy.Publisher('can_messages', String, queue_
      size=10)
292.                 msg = String()
293.                 msg.data = str(can_msg)
```

```
294.         pub.publish(msg)
295.         # 检查错误条件并发布错误消息
296.         error_pub = rospy.Publisher('can_error_messages', String
, queue_size=10)
297.         error_msg = String()
298.         if can_msg[-4:] == 'EEEE':
299.             error_msg.data = f"Error: Propeller blocked for ID {
ID}"
300.             error_pub.publish(error_msg)
301.             elif can_msg[-4:] == '0000':
302.                 error_msg.data = f"Error: Actuator stopped for ID {I
D}"
303.                 error_pub.publish(error_msg)
304.
305.
306. def stop_all_propellers():
307.     for pid in range(len(ID)):
308.         handle_command(MotorCommandMsg(ID=pid, command=0), 'propelle
r')
309.         handle_command(MotorCommandMsg(ID=pid, command=0), 'motor')
310.
311.
312. if __name__ == '__main__':
313.     # 初始化 NetworkInterfaceChecker
314.     checker = NetworkInterfaceChecker('192.168.50.10')
315.     checker.set_callback(stop_all_propellers) # 设置触发保护功能的回
调函数
316.     checker_thread = threading.Thread(target=checker.check_interface
_communication)
317.     checker_thread.start() # 启动线程
318.
319.     # 创建一个线程来运行 receiveData 函数
320.     thread = threading.Thread(target=receiveData)
321.     thread.start() # 启动线程
322.
323.     try:
324.         actuation_node()
325.     except rospy.ROSInterruptException:
```

```
326.         pass
327.     finally:
328.         checker.stop() # 停止网络检查线程
```

2. Robot sensor data reading

```
1. #!/usr/bin/env python
2.
3. import rospy
4. import serial
5. import struct
6. import csv
7. import time
8. import threading
9. from sensor_fish.msg import DVLData
10. from queue import Queue
11.
12. # 串口配置
13. port = '/dev/ttyUSB0'
14. baudrate = 460800
15.
16. # 打开串口
17. ser = serial.Serial(port, baudrate, timeout=1)
18.
19. # CSV 文件配置
20. csv_file = 'DVL_data.csv'
21. csv_columns = [
22.     'Frame Count', 'Week', 'Week Seconds', 'Heading', 'Pitch', 'Roll',
23.     'East Velocity', 'North Velocity', 'Up Velocity', 'Latitude', 'Longitude',
24.     'Altitude', 'X Angular Velocity', 'Y Angular Velocity', 'Z Angular
    Velocity',
25.     'X Acceleration', 'Y Acceleration', 'Z Acceleration', 'Primary Satellite Count',
26.     'Secondary Satellite Count', 'Navigation Status', 'GNSS Status', 'Fault Status',
27.     'DVL Height', 'DVL Velocity'
28. ]
29.
```

```
30. # 写入 CSV 文件头
31. with open(csv_file, 'w', newline='') as csvfile:
32.     writer = csv.DictWriter(csvfile, fieldnames=csv_columns)
33.     writer.writeheader()
34.
35. shutdown_event = threading.Event()
36. data_queue = Queue()
37.
38. def parse_data(data):
39.     if len(data) < 94:
40.         print(f"Data length is too short: {len(data)}")
41.         return None
42.
43.     parsed_data = {}
44.     parsed_data['Frame Count'], = struct.unpack('<H', data[6:8])
45.     parsed_data['Week'], = struct.unpack('<H', data[8:10])
46.     parsed_data['Week Seconds'], = struct.unpack('<d', data[10:18])
47.     parsed_data['Heading'], = struct.unpack('<i', data[18:22])
48.     parsed_data['Pitch'], = struct.unpack('<i', data[22:26])
49.     parsed_data['Roll'], = struct.unpack('<i', data[26:30])
50.     parsed_data['East Velocity'], = struct.unpack('<i', data[30:34])
51.     parsed_data['North Velocity'], = struct.unpack('<i', data[34:38])
52.     parsed_data['Up Velocity'], = struct.unpack('<i', data[38:42])
53.     parsed_data['Latitude'], = struct.unpack('<i', data[42:46])
54.     parsed_data['Longitude'], = struct.unpack('<i', data[46:50])
55.     parsed_data['Altitude'], = struct.unpack('<i', data[50:54])
56.     parsed_data['X Angular Velocity'], = struct.unpack('<i', data[54:58
    ])
57.     parsed_data['Y Angular Velocity'], = struct.unpack('<i', data[58:62
    ])
58.     parsed_data['Z Angular Velocity'], = struct.unpack('<i', data[62:66
    ])
59.     parsed_data['X Acceleration'], = struct.unpack('<i', data[66:70])
60.     parsed_data['Y Acceleration'], = struct.unpack('<i', data[70:74])
61.     parsed_data['Z Acceleration'], = struct.unpack('<i', data[74:78])
62.     parsed_data['Primary Satellite Count'], = struct.unpack('<B', data[
    78:79])
```

```
63.     parsed_data['Secondary Satellite Count'], = struct.unpack('<B', dat
        a[79:80])
64.     parsed_data['Navigation Status'], = struct.unpack('<B', data[80:81]
        )
65.     parsed_data['GNSS Status'], = struct.unpack('<H', data[81:83])
66.     parsed_data['Fault Status'], = struct.unpack('<H', data[83:85])
67.     parsed_data['DVL Height'], = struct.unpack('<f', data[85:89])
68.     parsed_data['DVL Velocity'], = struct.unpack('<f', data[89:93])
69.
70.     # 转换数据格式
71.     parsed_data['Heading'] *= 0.0001
72.     parsed_data['Pitch'] *= 0.0001
73.     parsed_data['Roll'] *= 0.0001
74.     parsed_data['East Velocity'] *= 0.0001
75.     parsed_data['North Velocity'] *= 0.0001
76.     parsed_data['Up Velocity'] *= 0.0001
77.     parsed_data['Latitude'] *= 0.0000001
78.     parsed_data['Longitude'] *= 0.0000001
79.     parsed_data['Altitude'] *= 0.0001
80.     parsed_data['X Angular Velocity'] *= 0.000001
81.     parsed_data['Y Angular Velocity'] *= 0.000001
82.     parsed_data['Z Angular Velocity'] *= 0.000001
83.     parsed_data['X Acceleration'] *= 0.000001
84.     parsed_data['Y Acceleration'] *= 0.000001
85.     parsed_data['Z Acceleration'] *= 0.000001
86.
87.     return parsed_data
88.
89. # 找到同步头
90. def find_sync():
91.     while not rospy.is_shutdown() and not shutdown_event.is_set():
92.         byte = ser.read(1)
93.         if byte == b'\x55':
94.             next_byte = ser.read(1)
95.             if next_byte == b'\xAA':
96.                 # 已找到同步头
97.                 return
98.
```

```
99. def publish_data(parsed_data):
100.     dvl_data = DVLData()
101.     dvl_data.heading = parsed_data['Heading']
102.     dvl_data.pitch = parsed_data['Pitch']
103.     dvl_data.roll = parsed_data['Roll']
104.     dvl_data.dvl_height = parsed_data['DVL Height']
105.     dvl_data.dvl_velocity = parsed_data['DVL Velocity']
106.     dvl_data.stat_byte = parsed_data['Navigation Status']
107.     dvl_data.latitude = parsed_data['Latitude']
108.     dvl_data.longitude = parsed_data['Longitude']
109.     dvl_data.altitude = parsed_data['Altitude']
110.     dvl_pub.publish(dvl_data)
111.
112. def csv_writer_thread(queue):
113.     with open(csv_file, 'a', newline='') as csvfile:
114.         writer = csv.DictWriter(csvfile, fieldnames=csv_columns)
115.         while not rospy.is_shutdown() and not shutdown_event.is_set
116.             ():
117.                 if not queue.empty():
118.                     data = queue.get()
119.                     writer.writerow(data)
120.                     rospy.sleep(0.1)
121.
122. def serial_reader():
123.     while not rospy.is_shutdown() and not shutdown_event.is_set():
124.         find_sync()
125.         data = ser.read(92)
126.         data = b'\x55\xAA' + data
127.         if len(data) == 94:
128.             parsed_data = parse_data(data)
129.             if parsed_data:
130.                 data_queue.put(parsed_data)
131.                 publish_data(parsed_data)
132.                 rospy.loginfo(f"Data published: {parsed_data}")
133.             else:
134.                 rospy.logwarn("Parsed data is None.")
135.         else:
```

```
135.         rospy.logwarn(f"Read data length mismatch: {len(data)}"
136.     )
137.         rospy.sleep(0.01)
138.     def shutdown_hook():
139.         rospy.loginfo("关闭中...")
140.         shutdown_event.set()
141.
142.     def main():
143.         rospy.init_node('dvl_publisher')
144.         global dvl_pub
145.         dvl_pub = rospy.Publisher('dvl/data', DVLData, queue_size=10)
146.         rospy.on_shutdown(shutdown_hook)
147.
148.         csv_thread = threading.Thread(target=csv_writer_thread, args=(d
149.             ata_queue,))
150.         serial_thread = threading.Thread(target=serial_reader)
151.         csv_thread.start()
152.         serial_thread.start()
153.
154.         csv_thread.join()
155.         serial_thread.join()
156.
157.     if __name__ == '__main__':
158.         main()
159.
```

```
1. import serial
2. import time
3. import string
4. import pandas as pd
5. from datetime import datetime
6. import threading
7. import rospy
8. from sensor_fish.msg import Warmdepth # 导入自定义消息类型
9.
10.result = ""
```

```
11. df = pd.DataFrame(columns=["Time", "Type", "Height", "Temp", "Depth", "
    Pressure"])
12. data_queue = [] # 用于存储每秒的同步数据
13. Com485_Sensor = None
14. stop_threads = False # 停止标志
15.
16. def load_params():
17.     global Com485_Sensor
18.     serial_port = '/dev/ttyUSB1' # 根据你的设置调整
19.     baud_rate = 9600 # 根据你的设置调整
20.     Com485_Sensor = serial.Serial(serial_port, baud_rate, timeout=1)
21.
22. def Get485_Info_Altitude_Sensor(data):
23.     global result
24.     global df
25.     global data_queue
26.     if data[0] == "$ISADS": # 高度传感器: 温度、高度状态
27.         height = float(data[1])
28.         temp = float(data[3])
29.         result = "{}, {}".format(height, temp)
30.         current_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f")
31.         df.loc[len(df)] = [current_time, "Altitude", height, temp, None
            , None]
32.         data_queue.append({"time": current_time, "height": height, "tem
            p": temp})
33.         print('height status : %s' % result)
34.
35. def Get485_Info_Depth_Temperature_Sensor(data):
36.     global result
37.     global df
38.     global data_queue
39.     if data[0] == "$ISDPT": # 深度和温度传感器
40.         depth = float(data[1])
41.         pressure = float(data[3])
42.         temp = float(data[5])
43.         result = "{}, {}, {}".format(depth, pressure, temp)
44.         current_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f")
```



```
45.         df.loc[len(df)] = [current_time, "Depth", None, temp, depth, pr
           ssure]
46.         data_queue.append({"time": current_time, "depth": depth, "press
           ure": pressure})
47.         print('temperature and depth status : %s' % result)
48.
49. def Get_All_Data_from_485(OriginalData):
50.     global result
51.     try:
52.         temp = str(OriginalData, encoding="ISO-8859-1")
53.         if any(char not in string.printable for char in temp): # 过滤
           掉不可打印字符
54.             return
55.         data = temp.split(',')
56.         if data[0] == "$ISADS": # 高度传感器: 温度、高度状态
57.             Get485_Info_Altitude_Sensor(data)
58.         if data[0] == "$ISDPT": # 深度和温度传感器
59.             Get485_Info_Depth_Temperature_Sensor(data)
60.     except Exception as e: # 处理异常, 通常由多个设备的电磁干扰引起
61.         print(e)
62.
63. def Send_Request_and_Read_Response(request):
64.     global stop_threads
65.     while not rospy.is_shutdown() and not stop_threads:
66.         Com485_Sensor.write(request.encode())
67.         time.sleep(0.001) # 根据传感器响应时间调整睡眠时间
68.         Original_Data = Com485_Sensor.readline()
69.         if len(Original_Data) > 0 and Original_Data[0] == 36: # 36 是
           '$' 的 ASCII 码
70.             Get_All_Data_from_485(Original_Data)
71.
72. def Save_Data_To_CSV():
73.     global df
74.     global stop_threads
75.     while not rospy.is_shutdown() and not stop_threads:
76.         try:
77.             df.to_csv("data_log.csv", index=False)
78.         except Exception as e:
```

```
79.         print("Error while saving data to CSV:", e)
80.     finally:
81.         time.sleep(1)
82.
83. def Process_And_Send_Data(pub):
84.     global data_queue
85.     global stop_threads
86.     rate = rospy.Rate(5)
87.     while not rospy.is_shutdown() and not stop_threads:
88.         if data_queue:
89.             synchronized_data = {}
90.             for data in data_queue:
91.                 if "height" in data and "depth" in data:
92.                     synchronized_data = data
93.                     break
94.                 elif "height" in data:
95.                     synchronized_data.update({"height": data["height"],
96.                                               "temp": data["temp"]})
97.                 elif "depth" in data:
98.                     synchronized_data.update({"depth": data["depth"], "
99.                                               pressure": data["pressure"]})
100.
101.             if synchronized_data:
102.                 sensor_data_msg = Warmdepth()
103.                 sensor_data_msg.time = synchronized_data.get("time",
104.                                                              "")
105.                 sensor_data_msg.height = synchronized_data.get("height", 0.0)
106.                 sensor_data_msg.temp = synchronized_data.get("temp",
107.                                                               0.0)
108.                 sensor_data_msg.depth = synchronized_data.get("depth",
109.                                                                0.0)
110.                 sensor_data_msg.pressure = synchronized_data.get("pressure", 0.0)
111.                 rospy.loginfo(sensor_data_msg)
112.                 pub.publish(sensor_data_msg)
113.                 data_queue = [] # 清空队列
114.
115.
```

```
110.         rate.sleep()
111.
112. def main():
113.     global stop_threads
114.     load_params()
115.     rospy.init_node('sensor_data_publisher', anonymous=True)
116.     pub = rospy.Publisher('sensor_data', Warmdepth, queue_size=10)
117.
118.     save_thread = threading.Thread(target=Save_Data_To_CSV)
119.     save_thread.daemon = True # 守护线程
120.     save_thread.start()
121.
122.     process_thread = threading.Thread(target=Process_And_Send_Data,
        args=(pub,))
123.     process_thread.daemon = True # 守护线程
124.     process_thread.start()
125.
126.     start_time = time.time()
127.     data_count = 0
128.     try:
129.         while not rospy.is_shutdown():
130.             # 请求高度传感器数据
131.             Send_Request_and_Read_Response("$ISADS\n")
132.             # 请求深度和温度传感器数据
133.             Send_Request_and_Read_Response("$ISDPT\n")
134.             # 每秒计算并打印数据传输频率
135.             data_count += 1
136.             elapsed_time = time.time() - start_time
137.             if elapsed_time >= 1:
138.                 print(f>Data transfer frequency: {data_count} messag
        es per second")
139.                 data_count = 0
140.                 start_time = time.time()
141.         except rospy.ROSInterruptException:
142.             stop_threads = True
143.
144. if __name__ == '__main__':
145.     main()
```

3. Robot depth and height control

```
1. #!/usr/bin/env python3
2.
3. import rospy
4. from sensor_fish.msg import Warmdepth as HeightMsg # 高度传感器消息类型
5. from manta.msg import Warmdepth as DepthMsg # 深度传感器消息类型
6. from manta.msg import CommandMsg # 控制命令消息类型
7. import time
8. import signal
9. import sys
10.
11. class HeightController:
12.     def __init__(self, target_height=10.0):
13.         self.target_height = target_height
14.         self.current_height = 0.0
15.         self.triggered = False # 触发定深控制的标志
16.
17.         # 订阅高度传感器数据
18.         self.height_sub = rospy.Subscriber('altitude_sensor_data', HeightMsg, self.height_callback)
19.
20.         # 注册信号处理函数
21.         signal.signal(signal.SIGINT, self.shutdown)
22.
23.     def height_callback(self, msg):
24.         self.current_height = msg.height
25.         rospy.loginfo(f"Received Height Data: {self.current_height} meters")
26.
27.         # 如果当前高度达到了目标高度, 并且还没有触发定深控制
28.         if self.current_height >= self.target_height and not self.triggered:
29.             rospy.loginfo("Target height reached, triggering depth control.")
30.             self.trigger_depth_control()
31.             self.triggered = True
32.
33.     def trigger_depth_control(self):
```

```
34.         # 初始化并启动深度控制器
35.         depth_controller = DepthController(target_depth=2.0, kp=1.0, ki
      =0.01, kd=0.1, duration=10.0)
36.         depth_controller.start()
37.
38.     def shutdown(self, signum, frame):
39.         rospy.loginfo("Shutdown signal received.")
40.         sys.exit(0)
41.
42. class DepthController:
43.     def __init__(self, target_depth=2.0, kp=1.0, ki=0.01, kd=0.1, durat
      ion=10.0):
44.         self.kp = kp
45.         self.ki = ki
46.         self.kd = kd
47.         self.target_depth = target_depth
48.         self.current_depth = 0.0
49.         self.previous_error = 0.0
50.         self.integral_error = 0.0
51.         self.previous_time = rospy.get_time()
52.         self.start_time = rospy.get_time()
53.         self.duration = duration
54.
55.         # 计数变量
56.         self.depth_count = 0
57.         self.control_count = 0
58.         self.last_print_time = rospy.get_time()
59.
60.         # 订阅深度传感器数据
61.         self.depth_sub = rospy.Subscriber('depth_sensor_data', DepthMsg
      , self.depth_callback)
62.         # 发布控制信号
63.         self.control_pub = rospy.Publisher('propeller_commands', Comman
      dMsg, queue_size=10)
64.
65.     def depth_callback(self, msg):
66.         self.current_depth = msg.depth
67.         self.depth_count += 1
```

```
68.         rospy.loginfo(f"Received Depth Data: {self.current_depth} meter
        s")
69.         self.control_step()
70.
71.     def control_step(self):
72.         current_time = rospy.get_time()
73.
74.         # 检查是否超过了设定的时间
75.         if current_time - self.start_time > self.duration:
76.             rospy.loginfo("Depth control duration has ended. Stopping a
        ll propellers.")
77.             self.stop_all_propellers()
78.             rospy.signal_shutdown("Depth control finished")
79.             return
80.
81.         # 计算误差
82.         error = self.target_depth - self.current_depth
83.         delta_time = current_time - self.previous_time
84.
85.         # 计算PID 控制输出
86.         self.integral_error += error * delta_time
87.         p_term = self.kp * error
88.         i_term = self.ki * self.integral_error
89.         d_term = self.kd * (error - self.previous_error) / delta_time i
        f delta_time > 0 else 0.0
90.         control_signal = p_term + i_term + d_term
91.
92.         # 将控制信号限制在 300 到 500 之间
93.         control_signal = max(min(int(control_signal), 500), 300)
94.
95.         # 发布控制信号
96.         for propeller_id in range(4):
97.             command_msg = CommandMsg()
98.             command_msg.ID = propeller_id
99.             command_msg.command = control_signal
100.            self.control_pub.publish(command_msg)
101.            self.control_count += 1
102.
```

```
103.         # 更新前一误差和时间
104.         self.previous_error = error
105.         self.previous_time = current_time
106.
107.         # 打印频率信息
108.         if current_time - self.last_print_time >= 1.0:
109.             rospy.loginfo(f"Depth Read Frequency: {self.depth_count}
110. Hz, Control Publish Frequency: {self.control_count} Hz")
111.             self.depth_count = 0
112.             self.control_count = 0
113.             self.last_print_time = current_time
114.
115.     def stop_all_propellers(self):
116.         # 停止所有推进器
117.         for _ in range(2):
118.             for propeller_id in range(4):
119.                 command_msg = CommandMsg()
120.                 command_msg.ID = propeller_id
121.                 command_msg.command = 0
122.                 self.control_pub.publish(command_msg)
123.                 rospy.loginfo(f"Stopped Propeller {propeller_id}")
124.                 rospy.sleep(0.3)
125.
126.     def start(self):
127.         rospy.spin()
128.
129. def main():
130.     rospy.init_node('height_and_depth_control_node', anonymous=True)
131.
132.     # 启动高度控制器
133.     height_controller = HeightController(target_height=10.0)
134.
135.     # 保持节点运行
136.     rospy.spin()
137.
138. if __name__ == '__main__':
139.     main()
```

```
1. #!/usr/bin/env python3
2.
3. import rospy
4. import cvxpy as cp
5. from sensor_fish.msg import Warmdepth as HeightMsg # 高度传感器消息类型
6. from manta.msg import Warmdepth as DepthMsg # 深度传感器消息类型
7. from manta.msg import CommandMsg # 控制命令消息类型
8. import signal
9. import sys
10.
11. class HeightController:
12.     def __init__(self, target_height=10.0):
13.         self.target_height = target_height
14.         self.current_height = 0.0
15.         self.triggered = False # 触发定深控制的标志
16.
17.         # 订阅高度传感器数据
18.         self.height_sub = rospy.Subscriber('altitude_sensor_data', HeightMsg, self.height_callback)
19.
20.         # 注册信号处理函数
21.         signal.signal(signal.SIGINT, self.shutdown)
22.
23.     def height_callback(self, msg):
24.         self.current_height = msg.height
25.         rospy.loginfo(f"Received Height Data: {self.current_height} meters")
26.
27.         # 如果当前高度达到了目标高度，并且还没有触发定深控制
28.         if self.current_height >= self.target_height and not self.triggered:
29.             rospy.loginfo("Target height reached, triggering depth control.")
30.             self.trigger_depth_control()
31.             self.triggered = True
32.
33.     def trigger_depth_control(self):
34.         # 初始化并启动深度控制器 (MPC)
```



```
35.     depth_controller = DepthControllerMPC(target_depth=2.0, horizon
    =10, dt=0.1)
36.     depth_controller.start()
37.
38.     def shutdown(self, signum, frame):
39.         rospy.loginfo("Shutdown signal received.")
40.         sys.exit(0)
41.
42. class DepthControllerMPC:
43.     def __init__(self, target_depth=2.0, horizon=10, dt=0.1):
44.         self.target_depth = target_depth
45.         self.current_depth = 0.0
46.         self.horizon = horizon      # 预测时间步数
47.         self.dt = dt                # 采样时间
48.         self.u_max = 500            # 控制信号上限
49.         self.u_min = 300           # 控制信号下限
50.
51.         # 创建控制变量
52.         self.u = cp.Variable(self.horizon) # 控制信号向量
53.         self.x = cp.Variable(self.horizon+1) # 深度状态向量
54.
55.         # 定义 MPC 优化问题
56.         self.objective = cp.Minimize(cp.sum_squares(self.x[1:] - self.t
    arget_depth) + 0.1 * cp.sum_squares(self.u))
57.         self.constraints = [self.x[0] == self.current_depth]
58.         for t in range(self.horizon):
59.             self.constraints += [self.x[t+1] == self.x[t] + self.u[t] *
    self.dt] # 简单模型:  $x_{next} = x + u*dt$ 
60.             self.constraints += [self.u_min <= self.u[t], self.u[t] <=
    self.u_max]
61.
62.         self.problem = cp.Problem(self.objective, self.constraints)
63.
64.         # 订阅深度传感器数据
65.         self.depth_sub = rospy.Subscriber('depth_sensor_data', DepthMsg
    , self.depth_callback)
66.         # 发布控制信号
```

```
67.         self.control_pub = rospy.Publisher('propeller_commands', CommandMsg, queue_size=10)
68.
69.     def depth_callback(self, msg):
70.         self.current_depth = msg.depth
71.         rospy.loginfo(f"Received Depth Data: {self.current_depth} meters")
72.         self.control_step()
73.
74.     def control_step(self):
75.         # 更新初始状态
76.         self.constraints[0].rhs = self.current_depth
77.
78.         # 解决优化问题
79.         try:
80.             self.problem.solve()
81.             control_signal = int(self.u[0].value) # 获取第一个时间步的控制信号
82.             control_signal = max(min(control_signal, self.u_max), self.u_min) # 限制控制信号范围
83.             self.publish_control(control_signal)
84.         except Exception as e:
85.             rospy.logwarn(f"MPC solve failed: {e}")
86.             self.stop_all_propellers()
87.
88.     def publish_control(self, control_signal):
89.         # 发布控制信号到所有推进器
90.         for propeller_id in range(4):
91.             command_msg = CommandMsg()
92.             command_msg.ID = propeller_id
93.             command_msg.command = control_signal
94.             self.control_pub.publish(command_msg)
95.             rospy.loginfo(f"Published control signal: {control_signal} to Propeller {propeller_id}")
96.
97.     def stop_all_propellers(self):
98.         # 停止所有推进器
99.         for propeller_id in range(4):
```

```
100.         command_msg = CommandMsg()
101.         command_msg.ID = propeller_id
102.         command_msg.command = 0
103.         self.control_pub.publish(command_msg)
104.         rospy.loginfo(f"Stopped Propeller {propeller_id}")
105.
106.     def start(self):
107.         rospy.spin()
108.
109. def main():
110.     rospy.init_node('height_and_depth_control_node', anonymous=True)
111.
112.     # 启动高度控制器
113.     height_controller = HeightController(target_height=10.0)
114.
115.     # 保持节点运行
116.     rospy.spin()
117.
118. if __name__ == '__main__':
119.     main()
```