Technical Report: Robot and Sensor Architecture for Cartography

Emma CASAGRANDE
(supervisor: Loica Avanthey and Laurent Beaudoin)

Technical Report n^o02406-techrep-CASAGRANDE, June 2023

This report will cover the research I am conducting with the Underwater Robotics Laboratory of EPITA, SEAL, which focuses on the integration of the Robot Operating System, otherwise known as ROS, to build software architectures for different robots, particularly surface and underwater robots: interfacing actuators, sensors, behavior, communication and multi-robot coordination.

Ce rapport couvrira la recherche que je mène avec le laboratoire de robotique sous-marine de l'EPITA, SEAL, qui se concentre sur l'intégration du Robot Operating System, également connu sous le nom de ROS, pour construire des architectures logicielles pour différents robots, en particulier les robots de surface et sous-marins : interfaçage des actionneurs, des capteurs, du comportement, de la communication et de la coordination multi-robots.

Keywords
ROS, surface and underwater robots, software architecture

Laboratoire de Recherche de l'EPITA
14-16, rue Voltaire – FR-94276 Le Kremlin-Bicêtre CEDEX – France
Tél. +33 1 53 14 59 22 – Fax. +33 1 53 14 59 13
emma1.casagrande@epita.fr – http://www.lre.epita.fr/

Copying this document

Aknowledgments

I would like to thank my supervisors Loica AVANTHEY and Laurent BEAUDOIN for helping me and guiding me during these first several months at the LRE. Their guidance and advice has been invaluable and I am very excited to continue working with them on this project.

Contents

0.1 Introduction

0.1.1 Context

Throughout the past couple of years, there has been a significant surge in the development of underwater and surface exploration robots (Hayden). This has been a response to a number of challenges confronting our oceans. Some examples include:

- Marine Exploration
- Underwater Mapping
- Environmental Monitoring: monitoring environmental parameters such as water quality, temperature, salinity, and marine life distribution.
- Offshore Industry Activities: support various offshore industry activities such as oil and gas exploration, pipeline inspection, or underwater infrastructure maintenance.

However, despite the increasing demand and necessity for these machines, the underwater robotics community faces several obstacles that are hindering its progress:

- The natural environment: Factors like water pressure or corrosion make designing reliable and durable systems a complex task. Moreover, ensuring the robot's ability to navigate, gather data, and perform tasks accurately can be difficult due to factors like low visibility or communication difficulties.
- Cost: Autonomous Underwater Vehicles (AUVs) are generally more expensive compared to aerial or ground AUVs. Even low-cost AUVs in the market may require significant investment and time for prototyping, especially for inexperienced organizations.
- Bring-up Time: Developing a custom underwater robot often requires a significant amount of time for prototyping and testing, as well as training inexperienced organizations on how to use different tools.
- Lack of Modularity: Lack of modularity in existing AUV designs hampers flexibility in configuration across mechanical, electrical, and software designs.

0.1.2 Subject

Some significant challenges we are facing at SEAL are the effective communication with underwater robots as well as their navigation and control, especially given the limitations imposed by the environment. We need to find a way to coordinate these robots that is cost-effective, quick to implement, requires minimal learning time, and offers modular logistics that can be easily integrated into various robots (whether they be surface or underwater) with different objectives.

There are various tools and platforms tailored for underwater applications, such as ROS, Ardupilot, and MOOS. However, integrating and combining different systems and components remains challenging. For instance, Ardupilot is often combined with ROS for robot navigation, but it is not always obvious how to combine the two softwares. There is still a lack of straightforward implementations to effectively bridge these different tools, leading to significant difficulties. Indeed, our aim is to transition away from dependence on systems like Ardupilot.

The SEAL laboratory of underwater robotics at EPITA is focused on developing underwater robotic systems for terrain mapping that are modular, low-cost, and that rely exclusively on the Robot Operating System (ROS) as middleware.

0.2 State of the art

0.2.1 Robots of SEAL

The main objective of the robots at SEAL is to be able to map underwater terrains. This can later be useful to track the health of an ecosystem. For example, mapping the terrain could help monitor the growth of seagrass such as Posidonia and track the success of different strategies to protect the area.

SEAL has two main robotics systems:

- A surface robot connected to an underwater robot via a wired connection for communication. This setup can be useful for distributing the mapping zone across multiple surface robots, all guided by a fixed station on land.
- An underwater robot equipped with various cameras for underwater mapping.

Surface Robots:

- 2 motors
- GPS

Underwater Robots:

- 4 motors
- Pressure sensor
- Acceleration sensor
- Angle sensor
- GPS

The following data is used to correct the robot's trajectory:

- Robot angle
- Acceleration
- Localization on expected trajectory

The communication between the different systems and navigation are two of the main challenges we face. On one side, as mentioned beforehand, the surface robots communicate with the underwater ones through an ethernet cable, but these cables are fragile and costly to replace. On the other hand, the navigation still relies on different tools such as Ardupilot which can cause technical issues on the field, or takes up too much time to properly integrate this tool with ROS in an efficient and flexible manner.

0.2.2 The Robot Operating System (ROS)

ROS is a middleware dedicated to reliable, high-performance rapid prototyping for robotics applications (Wikipedia). It addresses several key challenges in robot construction. These include:

Pros:

- **Modularity:** Modularity is a key property that enables reuse of code, incremental design of functionality, and efficient testing. ROS allows code to be divided into different packages and nodes that perform singular actions.
- **Efficiency:** There is a variety of open-source code that is up to date with the latest hardware, which allows us to start working more quickly on more advanced functionality, seeing as there is no need to recode more basic already existing nodes. For example, the majority of sensor drivers already exist.
- **Compatibility:** ROS is operational across numerous operating systems and hardware architectures, thereby reducing costs and simplifying integration with existing robotic platforms.
- **Testing:** The tools provided by ROS allow us to record data from the robot's sensors, enabling us to develop and test the functionality of our robots in simulated environments.

Despite its advantages, several issues deter people from using ROS:

Cons:

- ROS is not a real-time operating system, which is crucial for robot control requiring high reactivity and low latency.
- There is still a significant lack of documentation and despite its extensive libraries, it has minimal applications for Autonomous Underwater Vehicles (AUVs).
- ROS is caught in a vicious cycle: the lack of documentation makes it difficult and time-consuming to adopt, leading many organizations to choose alternatives. Consequently, less documentation is developed, perpetuating the cycle.

However, the reason why we at SEAL are looking to transition to systems using only ROS is because it would simplify the maintenance and development of our robotics systems.

0.2.3 Study Case n°1 - YellowFin AUV

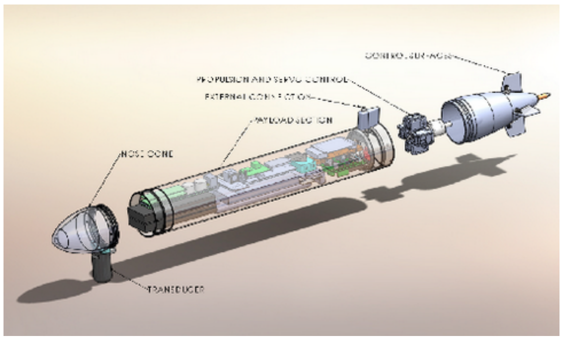


Figure 1: CAD of the GTRI Yellowfin

Objective:

The YellowFin is an autonomous underwater vehicle (AUV) developed by the Georgia Tech Research Institute (GTRI), whose purpose is to achieve mission goals independently, without requiring teleoperation, especially in environments with significant communication limitations like the ocean. (1) The GTRI Yellowfin uses the Robot Operating System (ROS) as the middleware facilitating interaction among all system components. The GTRI Yellowfin was designed for basic research into collaborative control, mine detection and countermeasures, or for scientific sampling in the ocean. It was guided on three principles:

- the use of open source software and standards,
- the use of commercial off-the-shelf (COTS) electronics,
- the use of modular software components.

(8)

Main components:

- Communication hardware
- Navigation sensors
- Low-level controller processing
- High-level controller processing

Architecture:

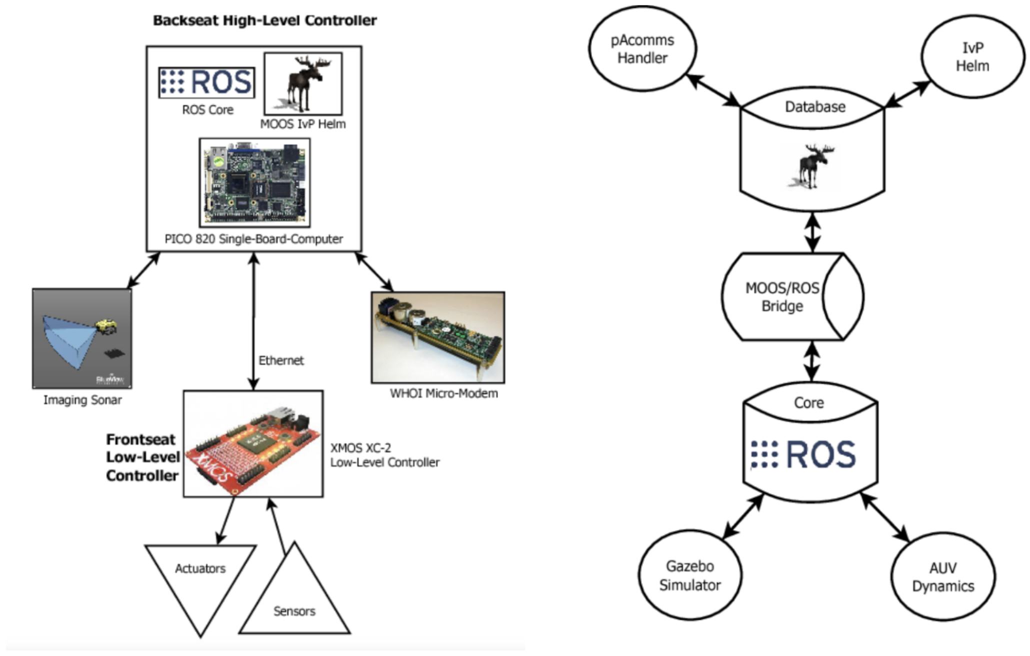


Figure 2: LEFT: Overview of Yellowfin Architecture RIGHT: Hardware-Software interaction architecture

The GTRI Yellowfin is based on a Front-seat/Back-seat paradigm, which means that they have separated vehicle autonomy from vehicle control. The front-seat is a low-level controller which maintains vehicle stability and reads and broadcasts sensor data. For example, it processes data collected from the navigational sensors, which include the inertial measurement unit (IMU), leak sensor, pressure sensor, and digital compass. The back-seat is a high-level operating system that handles completing the mission objectives with the data it received from the front-seat. (1)

Software Tools Utilized in the Yellowfin AUV

The Yellowfin AUV utilizes three different software tools: (7)

MOOS

MOOS is a middleware designed specifically for AUVs and features an onboard publish/subscribe architecture via its shared database, MOOSDB. In the Yellowfin system, mission execution of the vehicle is carried out using the MOOS-IvP suite of applications for high-level autonomy.

XMOS

The XMOS low-level controller is responsible for maintaining vehicle stability and handling direct control of the vehicle's actuators. The software handles Ethernet communication, sensor sampling, sensor fusion, motion control, and health monitoring.

ROS

ROS provides various services such as hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. Although ROS has minimal applications for AUVs thus far, extending ROS for AUVs is advantageous.

Example of ROS Usage in Yellowfin: When a ROS Camera node executes, it announces to the Master that it will publish images on the topic named "images." Subsequently, an Image Viewer ROS node informs the Master of its intention to subscribe to the topic "images." The Master then notifies both the Camera and Image Viewer nodes of each other's existence, enabling the flow of image data from the Camera node to the Image Viewer node.

Conclusion

According to the authors of the article "An overview of autonomous underwater vehicle systems and sensors at Georgia Tech", MOOS is a popular tool seeing as it is tailored specifically to AUVs, providing stricter scheduling and arbitration.

However, while MOOS excels in AUV-specific tasks and scheduling, integrating ROS into the software architecture allows the Yellowfin AUV to take advantage of ROS's broader capabilities and services. Therefore, a MOOS/ROS Bridge application has been set in place to enable the use of both ROS and MOOS applications within the same robotic platform, combining their strengths.

In conclusion, the Yellowfin uses MOOS primarily for high-level autonomous decision-making and a XMOS controller for maintaining vehicle stability and handling direct actuator control, which are critical for real-time operations and require the rigorous scheduling and arbitration provided by MOOS. ROS is used mainly for allowing better communication between the different components as well as handling the Gazebo simulation and AUV dynamics.

0.2.4 Study Case n°2 - Modularis

Objective:

The goal of the Modularis AUV design by researchers at the University of Florida is to respond to some of the main challenges we face today in the development of AUVs, such as making affordable and easily adaptable systems for the harsh underwater environment. Modularis is meant to offer a "design flexible in its configuration and able to be accommodated to different use cases". (3)

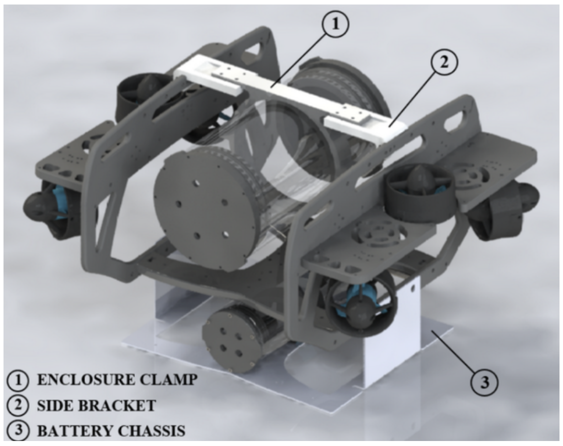


Figure 3: Assembly model of the upgraded BlueROV2

Software Design:

Modularis's software operates a dual-boot Robot Operating System (ROS) with a tethered and untethered architecture.

Tethered architecture: Uses the processing power of a land-based computer.

Untethered architecture: System relies on a Raspberry Pi for all computations.

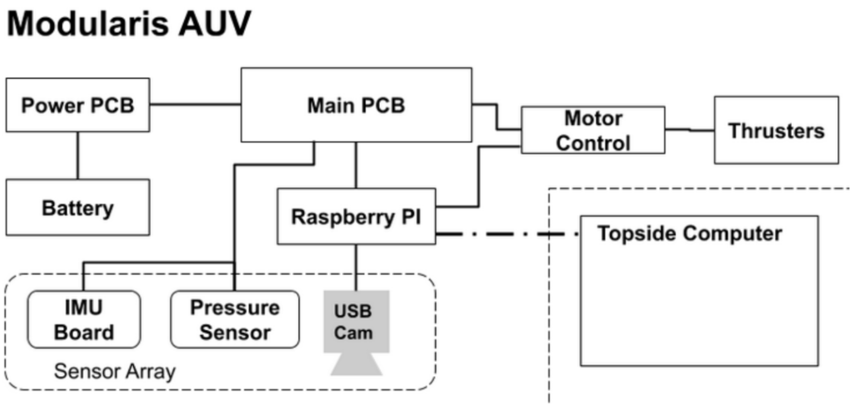


Figure 4: Diagram of Modularis's Electrical and Software Systems

Both systems use ROS because it facilitates the easy incorporation and combination of nodes. Each sensor functions as an individual node that publishes its specific data as a 'message'. This setup allows for much more flexible integration and/or swapping of new sensors and drivers into the sensor array, and consequently allows us to more easily adapt the prototype to specific tasks we want it to complete.

Main uses of ROS in Modularis:

Some examples of uses of ROS in Modularis are as follows:

- A configurable ROS interface that implements a common driver for standard USB web cameras (Open Robotics ROS2 USB-cam driver).
- Node-message configuration for tracking and sharing the data concerning the position and movement of Modularis collected by a BNO085 IMU.
- Thrusters structured as unique nodes in ROS.

Limitations:

One of the main issues with this design is the combination of both ROS 1 and ROS 2. ROS 1, the original version released in 2007, will only be officially supported through 2025, and therefore, as time progresses, the transition from ROS 1 to ROS 2 presents potential incompatibilities. On the contrary, ROS 2 has been designed to offer improved security, reliability, and support for real-time systems.

Conclusion:

Modularis is a perfect example of how ROS can increase the modularity of underwater robotic applications, enhancing the adaptability and portability of these systems. Indeed, the modular system of nodes and messages in ROS allows developers to more efficiently add and remove components since these are divided into singular nodes.

0.2.5 Study Case n°3 - Hybrid Underwater Robot System

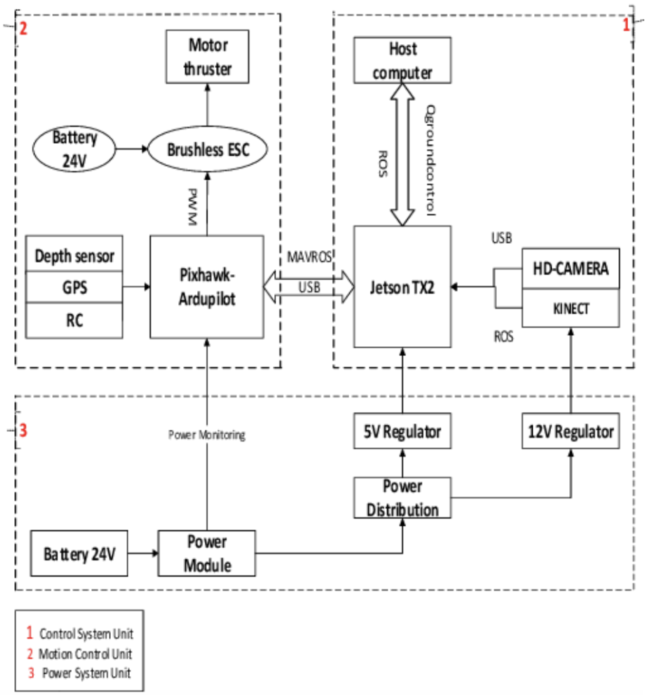


Figure 5: Hybrid hardware System Overview

Objective:

The hybrid underwater robot system is a design for AUVs proposed by Abdou Yahouza M. Sani, Tao He, Wenlong Zhao, and Tingting Yao, aiming to combine features of remotely operated underwater vehicles (ROVs) and autonomous underwater vehicles (AUVs) using both ROS and Ardupilot. The robot system can navigate while performing Simultaneous Localization and Mapping (SLAM), path-planning, and obstacle avoidance. (4)

Components of the Hybrid Underwater Robot System:

- **Control System Unit:** Connects to the host computer via USB and interfaces with sensors such as an HD camera (e.g., Kinect) through ROS.
- **Motion Control Unit:** Manages sensors (Depth sensor, GPS, RC) and interfaces with the motor thruster. The Pihawk-Ardupilot communicates with the Jetson TX2 (which manages the control system) via a ROS package that allows controlling drones via the MAVLink protocol, Mavros.
- **Power System Unit**

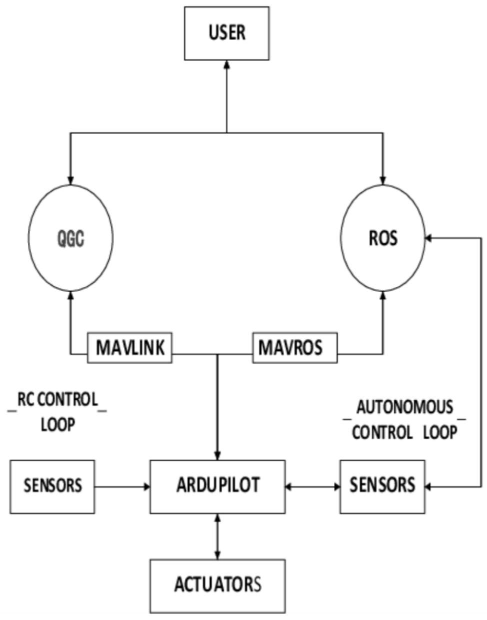


Figure 6: System Software-Hardware Interaction

A closer look at how the hardware and software interact reveals that the user interface communicates with QGroundControl (QGC) on one side and with ROS on the other. QGC provides a graphical interface for monitoring and controlling the AUV via MAVLink, while ROS nodes enable autonomous operations using MAVROS. Data is gathered and relayed through Ardupilot, which interfaces with various sensors and actuators. The autonomous control loop uses MAVROS to send commands from ROS to ArduPilot.

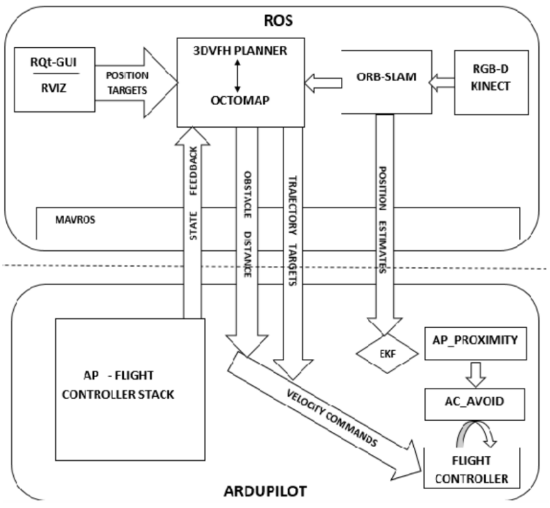


Figure 7: Software System Architecture

Software Components Division:

- **Components handled by ROS:**
  - **3DVFH Planner:** Uses obstacle data from the ORB-SLAM module to generate trajectory targets and provides information about obstacle distances.
  - **ORB-SLAM:** Processes visual data from the RGB-D Kinect to estimate position.
  - **RQT-GUI and RVIZ:** Tools for visualization and GUI interface within ROS.
- **Components handled by Ardupilot:**
  - Generated trajectory targets and velocity commands are sent to ArduPilot, which manages the flight controller and repositions the robot accordingly.

Conclusion:

In this hybrid underwater robot system, ROS serves as the principal middleware facilitating communication between various components that collect and process data. ROS handles high-level tasks such as perception (using Kinect), planning (3DVFH Planner), and visualization (RVIZ). The gathered information is then communicated to the Ardupilot flight controller, which manages low-level tasks such as reading sensor inputs (depth, GPS, RC) and controlling actuators (thrusters). This architecture provides insights into parts of an underwater robot's system that typically do not utilize ROS, highlighting potential gaps in ROS packages or libraries.

0.3 Study Case n°4 - Autonomous Sailing Boat

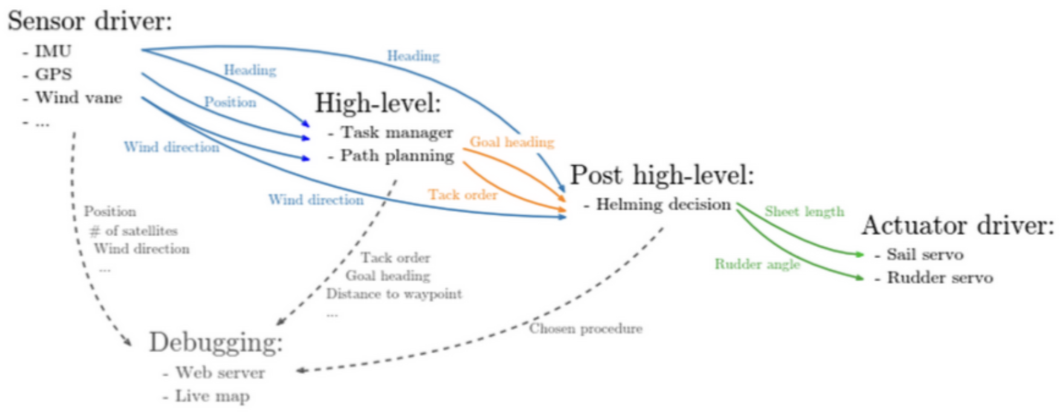


Figure 8: Figure Hardware System Overview

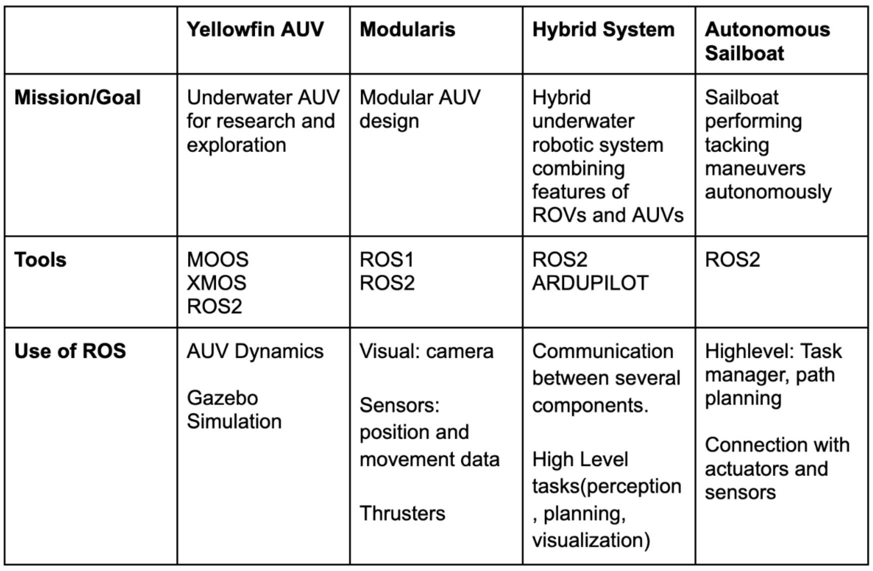
Objective

To sail upwind, sailboats have to cross the wind by performing a maneuver called tacking. The aim of the Southampton robotics team is to design an autonomous sailboat capable of performing the best tack according to various data such as weather conditions and the state of the boat. To achieve this, the team has developed a robotic architecture written in Python and assembled using ROS. (Team)

Software Architecture

The efficient use of ROS has allowed the Southampton Robotics Team to design an autonomous sailing boat that can correct its trajectory based on various factors. ROS facilitates communication between different hardware components, handling high-level tasks such as task management and planning, as well as low-level tasks like data acquisition and action control. The sailboat continuously collects environmental data, including wind conditions, navigation status, and position, which is relayed to the high-level control system responsible for task and movement planning. This system then issues commands to the different control units (pilots). Furthermore, beyond the high-level control, there is an additional stage specifying which of the four available tacking maneuvers to execute.

0.3.1 Comparison of Different Robotic Structures



	Yellowfin AUV	**Modularis**	**Hybrid System**	**Autonomous Sailboat**
Mission/Goal	Underwater AUV for research and exploration	Modular AUV design	Hybrid underwater robotic system combining features of ROVs and AUVs	Sailboat performing tacking maneuvers autonomously
Tools	MOOS XMOS ROS2	ROS1 ROS2	ROS2 ARDUPILOT	ROS2
Use of ROS	AUV Dynamics Gazebo Simulation	Visual: camera Sensors: position and movement data Thrusters	Communication between several components. High Level tasks(perception , planning, visualization)	Highlevel: Task manager, path planning Connection with actuators and sensors

Overall, we can see that all the projects concerning AUVs ended up combining ROS with another tool, confirming the principle that ROS is not fully adapted to AUV-specific needs.

ROS is evidently used in most projects for handling high-level tasks. It is not typically used for low-level tasks like reading sensor inputs (e.g., depth, GPS) and controlling actuators (e.g., thrusters).

However, there were certain projects exclusively utilizing ROS. One such project was the Modularis, a conceptual design aimed at making affordable and easily adaptable underwater systems. Despite its intent, Modularis chose to combine both ROS1 and ROS2, a decision which may not be very strategic or optimal at this time. (Venkatadri)

In contrast, a more positive example is the Autonomous Sailboat developed by the Southampton robotics team, which effectively employed ROS throughout its system.

The use of ROS in these projects highlights its importance in managing high-level tasks and integrating various components in robotic systems. By examining different strategies for integrating ROS, we can better understand its strengths and the areas requiring further development. Additionally, it becomes evident that while ROS excels in high-level task management, it often needs to be paired with other tools for low-level tasks, given that ROS is not a real-time operating system.

0.4 Actions Completed

A significant portion of my initial months at the laboratory has been dedicated to learning and mastering essential tools required for the development of robotics systems such as ROS and Gazebo. Additionally, I conducted an investigation into various existing projects within the domain of underwater robotics, whether they involve surface or underwater robots, with several key questions in mind:

0.4.1 Surface Robot Simulator

The objective was to develop a functional surface robot simulator using Gazebo, enabling testing under conditions similar to real-world deployments without physical terrain access. Key milestones included:

- Simulating the surface of a water body:
    1. Step 1: Implementing a simple still water surface without waves or currents.
    2. Step 2: Introducing wave simulations.
- Interaction with the vehicle: Incorporating buoyancy forces.
- Implementing a 3D model of the lab's surface robot.

Versions developed:

1. Version 1: Initial setup, focused on adding a simple surface representation, inspired by the Unmanned Underwater Vehicle Simulator package for Gazebo.
2. Version 2: Refinement with improved scale representation and updated color settings to denote 2D model simulation.

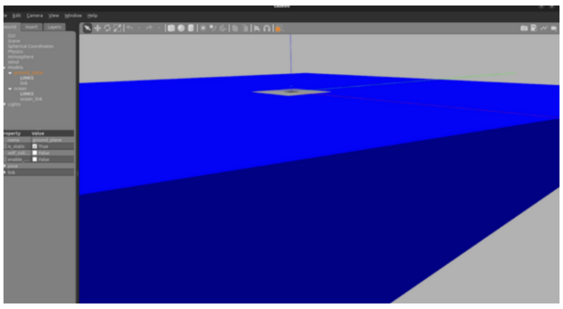


Figure 9: Simulator Version 1

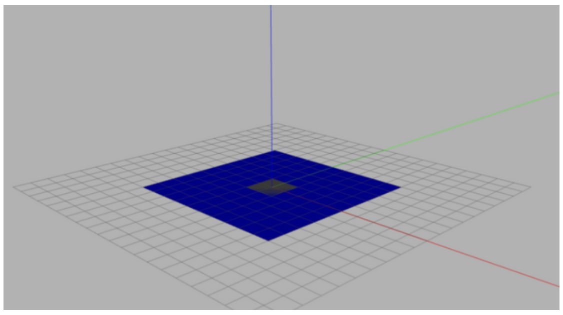


Figure 10: Simulator Version 2

0.4.2 Raspberry Pi Configuration

Currently, surface robots in the lab communicate with underwater robot systems via wired Ethernet connections, which restrict mobility and increase production costs. To address this issue, a Raspberry Pi has been configured as an access point to facilitate wireless communication among surrounding robots. Key steps included setting up the Raspberry Pi as an access point using a DHCP server.

0.4.3 Next Steps

Moving forward, one of the primary challenges in underwater robotics remains communication, as water impedes radio waves. Planned future steps include testing the maximum effective communication distance between two Raspberry Pis. To facilitate this, a "ping/pong" code has been developed:

- A publishing node sends a "ping" message along with the timestamp.
- A subscribing node responds with "pong" upon receiving the message, accompanied by the reception timestamp.

0.5 Conclusion

In conclusion, integrating the Robot Operating System (ROS) into SEAL's underwater and surface robots addresses the need for developing efficient, modular, and cost-effective solutions for underwater exploration and mapping. The research conducted over the past months underscores both the potential and challenges associated with this integration. Analysis of various projects and their ROS implementation strategies has highlighted strengths in modularity, efficiency, compatibility, and testing capabilities. However, limitations such as the lack of real-time operating system capabilities and inadequate documentation complicate ROS adoption, particularly for Autonomous Underwater Vehicles (AUVs).

Despite these challenges, the strategic adoption of ROS aims to streamline the complexity associated with integrating multiple systems like Ardupilot and MOOS. My current work has focused on developing a virtual simulator in Gazebo for surface robots and establishing Raspberry Pi configurations to enable wireless communication.

Future research will concentrate on refining these simulations, testing communication range and reliability, and fully transitioning to ROS. This progression promises to enhance the development, maintenance, adaptability, and scalability of our robotic systems.

Bibliography

[1] DeMarco, K., West, M. E., and Collins, T. R. (2011). An implementation of ros on the yellowfin autonomous underwater vehicle (auv). In *OCEANS'11 MTS/IEEE KONA*, pages 1–7. (pages 6 and 7)

[Hayden] Hayden, B. Progress in underwater robotic technology. (page 3)

[3] Herrin, B., Close, V., Berner, N., Herbert, J., Reussow, E., James, R., Woodward, C., Mindlin, J., Paez, S., Bretas, N., and Shin, J. (2024). Modularis: Modular underwater robot for rapid development and validation of autonomous systems. (page 8)

[4] Sani, A. Y. M., He, T., Zhao, W., and Yao, T. (2019). Hybrid underwater robot system based on ros. *Proceedings of the 2019 International Conference on Robotics, Intelligent Control and Artificial Intelligence*. (page 11)

[Team] Team, S. R. Southhampton autonomous sailboat github. (page 14)

[Venkatadri] Venkatadri, A. Ros 1 vs ros 2 what are the biggest differences? (page 15)

[7] West, M. E., Collins, T. R., Bogle, J. R., Melim, A., and Novitzky, M. M. (2011). An overview of autonomous underwater vehicle systems and sensors at georgia tech. (page 7)

[8] West, M. E., Novitzky, M. M., Varnell, J. P., Melim, A., Seguin, E., Toler, T. C., Collins, T., Bogle, J. R., Brad-ley, M. P., and Henshaw, A. M. (2010). 1 design and development of the yellowfin uuv for homogeneous collaborative missions. (page 6)

[Wikipedia] Wikipedia. Robot operating system. (page 5)