PAPER • **OPEN ACCESS**

Software Architecture Design of ASV Rybitwa: Development of an Autonomous Surface Vehicle for Dynamic Navigation and Task Execution

To cite this article: Igor Rusiecki *et al* 2024 *J. Phys.: Conf. Ser.* **2867** 012031

Journal of Physics: Conference Series **2867** (2024) 012031 doi:10.1088/1742-6596/2867/1/012031

Software Architecture Design of ASV Rybitwa: Development of an Autonomous Surface Vehicle for Dynamic Navigation and Task Execution

Igor Rusiecki[1], Tomasz Ujazdowski[2], Jakub Wilk[2], Patryk Sobolewski[2,3], Serhii Pyskovatskyi[3], Marcel Skierkowski[3], Tomasz Lisowski[3], and Wiktor Sieklicki[1]

[1]Faculty of Mechanical Engineering and Ship Technology, Gdańsk Tech, Gdańsk, Poland

[2]Faculty of Electrical and Control Engineering, Gdańsk Tech, Gdańsk, Poland

[3]Faculty of Electronics, Telecommunications and Informatics, Gdańsk Tech, Gdańsk, Poland

E-mail: igor.rusiecki@simle.pl

Abstract.

This paper introduces the software design of an Autonomous Surface Vehicle (ASV) named ASV Rybitwa. It was designed as an easily transportable platform capable of autonomous navigation and performing tasks during the RoboBoat 2024 competition, with emphasis on its modularity, generality and extendibility. The paper presents a software architecture for a small autonomous surface vehicle using novel tools that provide easy system integration and expansion. To meet such requirements, a generic control box connecting an autopilot (PX4) and a onboard computer (NVIDIA Jetson Nano) was developed so as to host the Robot Operating System 2 (ROS2) and to interact with sensors and actuators. Computer vision from three Oak-D cameras, equipped with AI algorithm support (YOLOv8), was used for navigation and obstacle avoidance. The decision-making system was designed using behavioural trees and computer vision, allowing the vehicle's capabilities to be adapted to the needs of the current tasks and mission. In addition, an Omni X propulsion system was proposed, providing full holonomy and enhancing dynamic positioning and navigation capabilities in complex navigational conditions. In addition this paper describes lessons learned from ASV Rybitwa's real-world testing during the aforementioned competition.

1 Introduction

In the field of maritime operations, the development of autonomous surface vehicles (ASVs) represents a significant advance in marine technology capabilities. Their utility is evident during long-duration missions that would be physically challenging for onboard crew or during missions that would involve a high risk to the crew's life and health - so-called "three D" missions which stands for "Dull, Dirty and Dangerous" [1,2]. To support the development of this type of vehicles, many studies are being carried out on the subject of autonomous movement, shape and control systems [3–6]. Among these pioneering ASVs is the Rybitwa (ang. *Tern*) (Figure 1), an autonomous surface vessel designed for dynamic navigation and precision tasking.

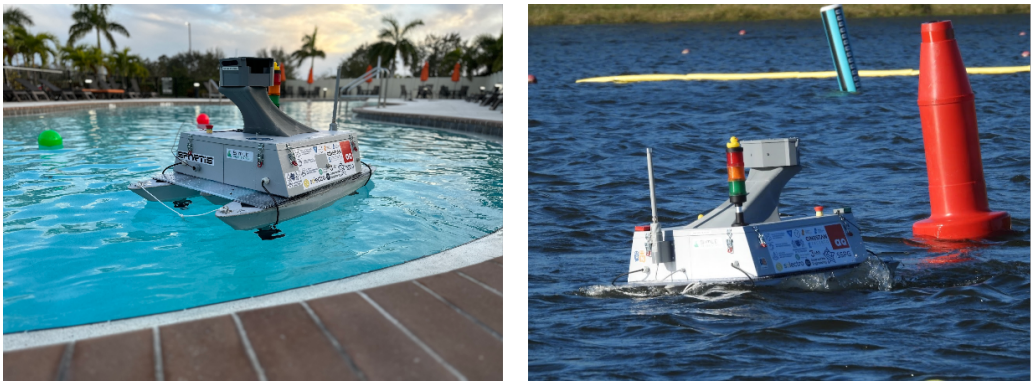

Figure 1: The ASV Rybitwa during the RoboBoat 2024 competition

In recent years, significant advancements have been mage in the development of autonomous maritime vessels. Topics that are most important to researchers include: manoeuvrability, collision avoidance, target identification and motion planning [7]. The authors [8] present the use of reinforcement learning in ASV control, but the application presented is limited to cruising around a single island. An interesting approach to waypoint-tracking control in ASV is presented in the research paper [9], simulating patrol paths, but this is limited to theoretical considerations only. In [10] short-term obstacle avoidance by use of a 3-D point cloud for obstacle tracking for ASV is presented. However, this approach is sensitive to pitch and roll and requires data filtering using an orientation sensor. Yet, some areas of research remain relatively unexplored. For example, the application of ROS2 [11], the use of camera-only computer vision on maritime vessels and the development of position controllers for surface vessels equipped with omnidirectional propulsion system. This paper aims to contribute to those areas.

At the core of Rybitwa's functionality is its software architecture, designed to aggregate sensors, actuators and decision-making algorithms. This research paper examines the software design aspects and their key role in enabling autonomous behaviour and ensuring mission success. Designing software architectures for autonomous systems is a multifaceted project that requires a delicate balance between robustness, adaptability and performance. In the case of the ASV Rybitwa, this challenge is compounded by the maritime environment's dynamic and often unpredictable nature. Therefore, the software architecture must not only facilitate autonomous navigation but also enable the vehicle to perceive, interpret and respond to its environment in real-time. This paper aims to provide a comprehensive understanding of the architecture of the ASV Rybitwa software, starting with an overview of its basic components and their interconnections. Subsequently, the paper contributes a detailed look into the specific algorithms and techniques employed for perception, localisation, path planning and control. Furthermore, it discusses the integration of advanced functions such as obstacle avoidance, collision prediction and adaptive decision-making, which are essential for autonomous navigation in complex environments.

Furthermore, this study examines the methodologies and principles guiding the development and validation of the ASV Rybitwa software architecture, particularly emphasizing the significance of simulation, testing and iterative refinement in ensuring reliability and safety. It contributes insights that can inform future advances in ASV technology.

The designed platform employs principles of robotics, artificial intelligence and marine engineering to create a system that is prepared for challenging harbour operations. This research paper presents a detailed analysis of the design and functionality of the autonomous surface vehicle (ASV), with the objective of highlighting the inner workings of this innovative technology and paving the way for further advancements in the field of marine robotics.

The paper is organised as follows. The problem is formulated in Section 2. Software architecture is described in Section 3. The setup, methods and experimental results are described in Section 4. Section 5 concludes the paper.

2 Problem formulation

This section briefly describes the design and system requirements that ASV had to fulfil to perform tasks during RoboBoat 2024. Section 2.1 describes the requirements themselves while Section 2.2 focuses on the mathematical formulation of the operating system.

2.1 Design Requirements

The vehicle was designed to perform tasks during the RoboBoat 2024 competition [12]. Requirements of the competition call for the performance of advanced manoeuvring tasks in an environment that simulates real-world challenges faced by the maritime industry like coastal surveillance, port security and various oceanographic operations. Coastal surveillance tasks require the vehicle to patrol designated areas and identify objects located there. Oceanographic operations require the vehicle to collect data by keeping the vehicle in the designated area and providing stability for measuring instruments.

Based on the requirements of the competition, software architecture must provide the following capabilities: recognition of the mission objective, definition of mission subtasks as well as routing and efficient navigation along the planned trajectory. The designed architecture must also ensure a fast and secure exchange of information between the decision-making system (DMS) and the vehicle control unit (VCU). In addition, an operator control station (OCS) is also included in the design, which allows monitoring the mission's status and remote control in an emergency.

2.2 System Description

The system was delineated below utilizing the standard notation for formulating problems in automation systems [4].

Suppose $\mathbb{R}^n$ is to represent a vector space over a field of real numbers $\mathbb{R}$ with usual addition $(+)$, scalar multiplication $(\cdot)$, Cartesian product $(\times)$, and $\mathbb{T} \subset \mathbb{R}$ denotes an open set. Taking a set of elements from a positive part of an integer field, $\{n_\mathrm{x}, n_\mathrm{u}, n_\mathrm{d}, n_\mathrm{y}, n_\mathrm{c}\} \subset \mathbb{Z}_+$, enables one to construct a n-tuple of vector valued functions $\mathbb{T} \to \mathbb{R}^n$ such that $\forall t \in \mathbb{T}$ it follows that $(\boldsymbol{x}(t), \overline{\boldsymbol{u}}(t), \boldsymbol{d}(t), \boldsymbol{y}(t), \boldsymbol{c}(t)) \in (\mathbb{X}_\mathrm{x}, \mathbb{X}_{\overline{\mathrm{u}}}, \mathbb{X}_\mathrm{d}, \mathbb{X}_\mathrm{y}, \mathbb{X}_\mathrm{c}) \subset (\mathbb{R}^{n_\mathrm{x}}, \mathbb{R}^{n_\mathrm{u}}, \mathbb{R}^{n_\mathrm{d}}, \mathbb{R}^{n_\mathrm{y}}, \mathbb{R}^{n_\mathrm{c}})$. In particular, t is to denote time.

$$\Sigma_\mathrm{ASV} : \begin{cases} \mathbb{X}_\mathrm{x} \times \mathbb{X}_{\overline{\mathrm{u}}} \times \mathbb{X}_\mathrm{d} \to \mathbb{R}^{n_\mathrm{x}} \\ \mathbb{X}_\mathrm{x} \times \mathbb{X}_{\overline{\mathrm{u}}} \times \mathbb{X}_\mathrm{d} \to \mathbb{X}_\mathrm{y} \\ \mathbb{X}_\mathrm{x} \times \mathbb{X}_{\overline{\mathrm{u}}} \times \mathbb{X}_\mathrm{d} \to \mathbb{X}_\mathrm{c} \end{cases}, \tag{1}$$

where $\mathbb{X}_\mathrm{x}$, $\mathbb{X}_{\overline{\mathrm{u}}}$, $\mathbb{X}_\mathrm{d}$, $\mathbb{X}_\mathrm{y}$, $\mathbb{X}_\mathrm{c}$ are used to denote operating regions in the state, control and disturbance input, measured and controlled output spaces, respectively.

The measurement system providing Global Positioning System (GPS), inertial measurement unit (IMU), and computer vision (Section 3.1) measurements is given by:

$$\Sigma_\mathrm{M} : \mathbb{X}_\mathrm{y} \times \mathbb{X}_\mathrm{d} \to \mathbb{X}_{\mathrm{y_m}}. \tag{2}$$

The control signals affect the ASV through an actuators system, consisting of a multi-propeller system as:

$$\Sigma_\mathrm{A} : \mathbb{X}_\mathrm{u} \to \mathbb{X}_{\overline{\mathrm{u}}}, \tag{3}$$

where $\mathbb{X}_{\overline{\mathrm{u}}} \subseteq \mathbb{X}_\mathrm{u} \ni \boldsymbol{u}(t)$, $\forall t$.

The multi-propeller system is under the control of the VCU Σ_VCU, which, $\forall t$, maps measurements $(\boldsymbol{y}_\mathrm{m}(t) \in \mathbb{X}_{\mathrm{y_m}} \subset \mathbb{X}_\mathrm{y})$ and reference trajectories $(\mathbb{X}_\mathrm{r})$ into control signals:

$$\mathbb{X}_\mathrm{u} \leftarrow \Sigma_\mathrm{A} \circ \Sigma_\mathrm{VCU} \circ \Sigma_\mathrm{M} \left[\mathbb{X}_{\mathrm{y_m}} \times \mathbb{X}_\mathrm{r}\right], \tag{4}$$

where $\circ$ denotes a composition operator.

The role of the DMS is to assign mission and submission objectives $(\boldsymbol{r}_\mathrm{MO})$ to ASV under supervision based on mission status reports $(\boldsymbol{r}_\mathrm{MR} \in \mathbb{X}_\mathrm{MR} \subset \mathbb{X}_\mathrm{y})$. The reference trajectory to be realized by the Σ_VCU is generated by a sophisticated system Σ_DMU, based on measurements and mission objectives, and described as:

$$\Sigma_\mathrm{DMS} : \boldsymbol{r}_\mathrm{MO} \times \mathbb{X}_\mathrm{MR} \mapsto \mathbb{X}_\mathrm{r}, \tag{5}$$

where: $\mathbb{X}_\mathrm{r}$ is a set of admissible mission objectives, that enables considered surface vehicle unit (Σ_ASV) to carry out an autonomous execution of prescribed mission objectives.

Therefore, it can be indicated that the control system is divided into a higher control layer for strategy development and a lower control layer for trajectory execution and communication with peripherals.

Within the higher control layer (Σ_DMS) the Behavioural Trees (BT) [13] are used to make decisions based on mission status and mission objectives. The lower control layer (Σ_VCU), uses PX4 [14] autopilot firmware.

3 Autonomous Surface Vehicle Design

Software architecture is designed for modularity and clear separation between components depending on their purpose. This attitude helped us create a robust, easy-to-maintain and flexible system. We combine the image data processed by the neural network and GPS signal for situational awareness. Processing the data for situational awareness is described in Section 3.1. A DMS running on a companion computer was used for high-level control, as detailed in Section 3.2. Low-level task execution is carried out through VCU described in Section 3.2. These two main components communicate using uXRCE-DDS [21] middleware over UDP. The feasibility of the control provided by the autonomous systems is ensured by the actuator system (Σ_A) described in Section 3.4.

3.1 Computer Vision

The computer vision block of the ASV, presented in this paper, can be divided into two segments: software and hardware. The basis of our hardware is an OAK-D stereo camera [15], which simultaneously conducts neural network computations and provides depth and colour information. Depth information is collected from two stereo cameras, while colour information is collected from a 4K camera at the centre. This central camera is occupied by DepthAI with a vision processing unit, which is designed to perform machine learning algorithms. This module has an application programming interface (API) in Python that allows Python programming solutions for computer vision. APIs allow user to create their own pipelines from pre-defined nodes. Nodes allow image processing, callback sending, neural network creation, and even customised function definition. Such node is a defined procedure, which allows the transmission of object detection and distance data to the onboard computer in the required format.

The software side of the computer vision solution presented in this paper is based on YOLOv8 [16]. This algorithm, as one of the latest versions of the YOLO family, proposes improvements in fast recognition, accuracy, and efficacy in comparison with YOLOv7 (see Figure 2). YOLOv8 consists of a convolutional neural network (CNN) that breaks the image into a grid and detects objects as bounding boxes. Computation resources were reduced with the use of the YOLOv8n version instead of the YOLOv8s version since the 'n' (nano) version has fewer trainable parameters and was pre-trained on a common objects in context (COCO) dataset. YOLOv8n has 3,2 million parameters, while YOLOv8s has 11,2 million [17]. Camera limitations and the requirement for real-time work were why the lightweight version was chosen. The YOLOv8n model has been trained using diverse data that was gathered during RoboBoat 2023, simulated data, and data collected during tests for RoboBoat 2024. All of this data has been labelled using the labelImg program [18]. This program supports the YOLO labelling format.

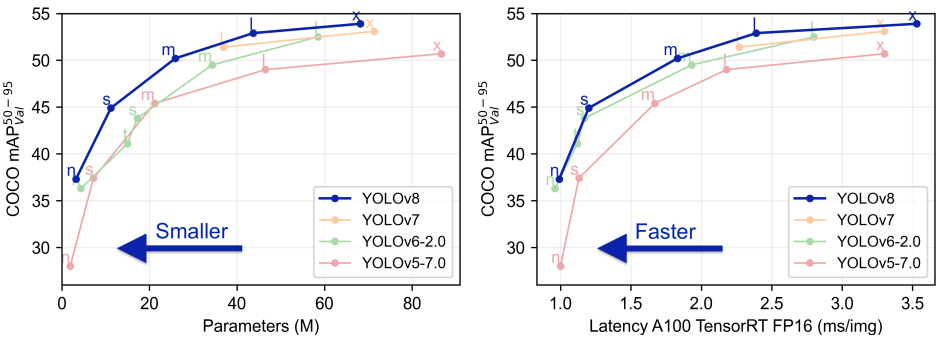

Figure 2: YOLOv8 comparison plot to previous version [17]

3.2 Vehicle Control Unit

Pixhawk 5X flight controller with PX4 firmware running on it was used as a VCU on our ASV. This solution was chosen because of wide variety of supported vehicle types as well as flexibility in creating controllers for custom ones. PX4 supports uXRCE-DDS middleware that allows uORB messages. uORB is an asynchronous publish/subscribe messaging API used for inter-thread and inter-process communication. The use of such middleware allows the messages to be published and subscribed on a ROS2 running on a companion computer as if they were ROS2 topics. Such a solution enables easy communication between these two units and smooth control of the vehicle accordingly to the currently executed task.

As the PX4 firmware is made out of modules, writing a custom controller requires implementing only parts specific to the designed vehicle. Everything else can be handled with the use of pre-made modules.

Commonly used modules such as a PID controller have ready-to-use interfaces. One of the greatest advantages of PX4 is its control allocator module. This module transforms thrust and torque setpoints into a value of the amount of power that needs to be delivered to each motor based on its position and orientation. All of the above allowed us to focus on implementing higher-level position control rather than low-level transformations.

3.3 Decision Making System

By determining its own position and the position of detected objects, the ASV calculated a safe trajectory that avoids obstacles and keeps the ASV within the navigation channel designated by red and green buoys. In addition, it dynamically determined the next navigation points. All of that is thanks to implementing the decision-making system in the behavioral trees. The behavioral trees also controlled the current stage of the mission, based on which decisions were made about the next actions. Figure 3 shows the data flow diagram.

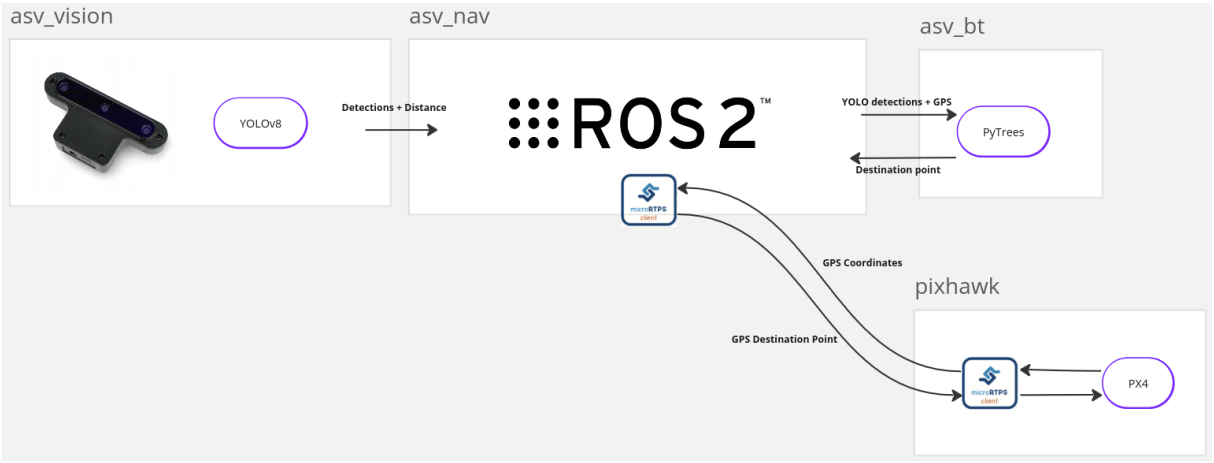

Figure 3: The data flow diagram

BTs offer a highly effective framework for designing and implementing decision-making systems in complex environments. Their modular structure allows for clear organization of tasks and behaviors, making them straightforward to implement and update. BTs enable easy testing and debugging, as individual components can be developed and validated independently. Additionally, the hierarchical nature of BTs simplifies the integration of new tasks, ensuring the scalability and adaptability of the system. However, one limitation of BTs is that their efficiency can degrade with the increasing complexity of the tree, potentially leading to longer decision times and higher computational.

For instance (see Figure 4) decision trees guide the system's actions based on the detected objects and their positions relative to the ASV. These decision trees outline the sequential execution of tasks, ensuring that each task is completed before moving on to the next. Additionally, the modularity of tasks within the system tree allows for easy updates, testing, and debugging, as well as the integration of new tasks as required.

The system's flow is structured using PyTree's library [19], which facilitates the design and implementation of behavior trees. These behavior trees enable seamless interaction between computer vision data and decision-making processes, ensuring efficient navigation and task execution in maritime environments.

3.4 Propulsion and Mobility

Propulsion is provided by four T200 Thrusters from Blue Robotics Inc. Since ASV Rybitwa is a catamaran, two thrusters are mounted on each floater. One at the bow and one at the stern, as shown in Figure 5. Each thruster is angled at 45 degrees to the centre line of the ASV. Each triangle represents the thrust vector of the propeller when it is working "forwards". When the ASV moves forwards, front propellers work "backwards". As a result, the ASV can generate a resultant thrust vector in every direction without the need to rotate the vehicle. It also allows the vehicle to spin around its axis. In addition, such configuration provides the vehicle with dynamic positioning capabilities.

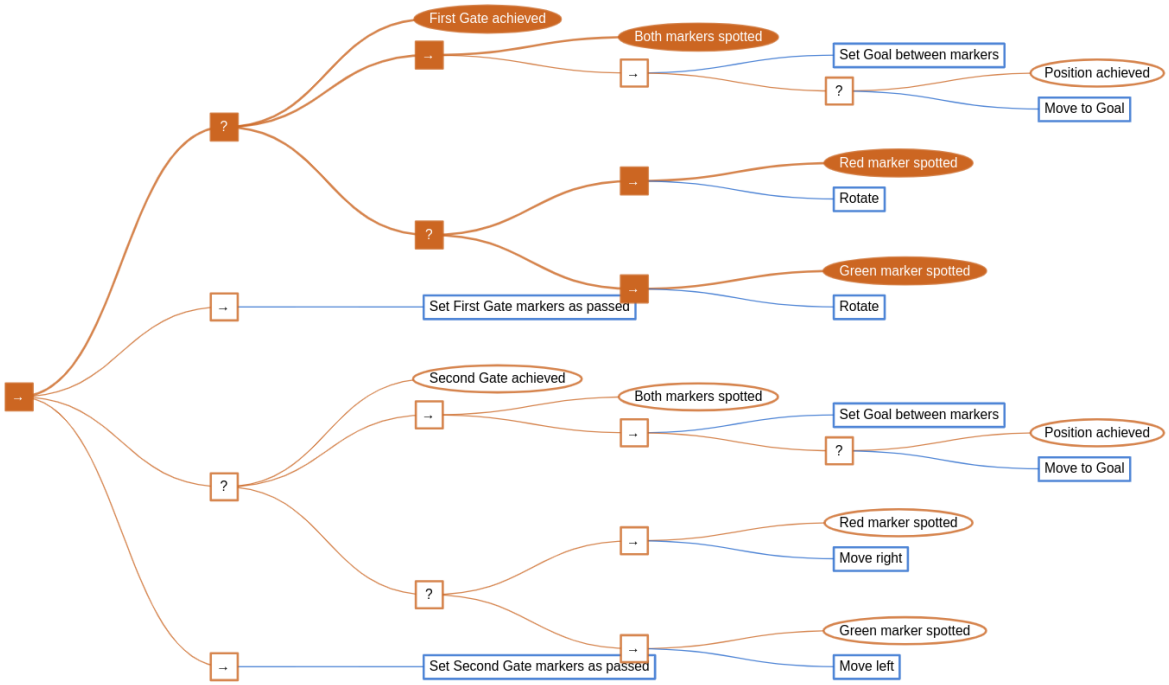

Figure 4: Diagram of the Behavioral Tree for a navigation task

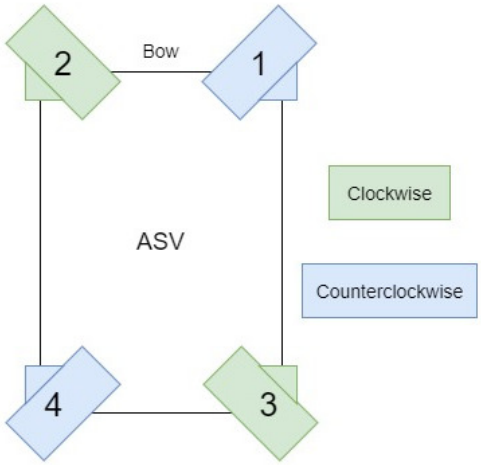

Figure 5: Thuster configuration on ASV Rybitwa

Each T200 Thruster can generate up to 5.25 kgf of thrust when operating at 16 V [20]. Thrust tests conducted during the competition indicate that the ASV's maximum bollard pull was 12,07 kgf.

Four "Basic ESCs" from BlueRobotics are used as interfaces between the Vehicle Control Unit and Thrusters. Each of them can provide 30A of maximum constant current to the thruster. This current value is controlled using a Pulse Width Modulation (PWM) signal. Since Pixhawk can output PWM signals, the control value from the software controller can be seamlessly transferred to the appropriate thruster.

4 Results & Discussion

This section presents the results obtained. Section 4.1 presents the experiment setup and a description of the conditions under which it was carried out, while Section 4.2 describes the equipment and methods used. Finally, Section 4.3 presents the results obtained.

4.1 Testing Environment and Setup

The ASV has been primarily tested during RoboBoat 2024 competition. The competition takes place annually in Sarasota County, FL. Obstacle courses are located on an artificial, freshwater lake in Nathan Benderson Park. In 2024 the competition took place in February. Air temperatures varied between 8 degrees Celsius in the early morning hours and 20 degrees Celsius in the afternoon with strong winds regularly reaching 4 in Beaufort scale. Since the lake is approximately 2 500 metres long and 600 metres wide, waves would reach a height of maximally 20-25 centimetres. RoboBoat organisers built two identical obstacle courses and a third one that was a mirrored copy of the first two courses. Each course consisted of tasks that an ASV needed to complete fully autonomously. An example of an obstacle course layout during the competition can be seen in Figure 6. An image of an actual course layout can be found in Figure 7.

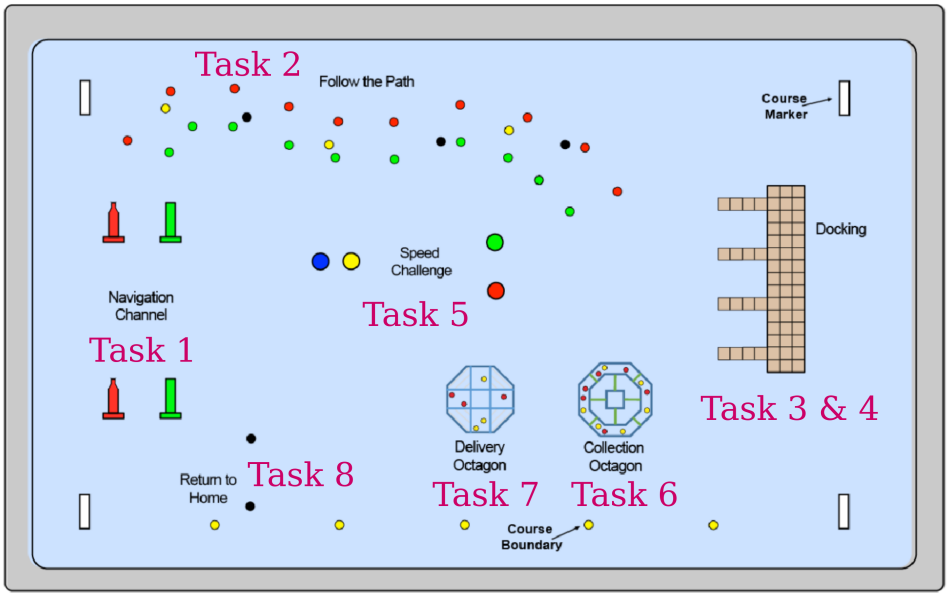

Figure 6: Obstacle courser, RoboBoat 2024 competition map [12]

4.2 Methods and Tools

ASV Rybitwa consists of components shown in Table 1.

The experiments conducted aimed to evaluate the performance of the ASV equipped with Jetson Nano, OAK-D, Pixhawk 5X, focusing on navigation, obstacle avoidance, and task execution.

Metrics and criteria were established to assess the ASV's performance. These included:

- Navigation Accuracy: Measured by the deviation from designated waypoints.
- Obstacle Avoidance: Assessed based on the number of successfully avoided obstacles.
- Task Completion Rate: Percentage of completed tasks within the designated time frame.

The ASV demonstrated satisfactory performance across the evaluated metrics. Navigation accuracy remained within acceptable limits, with deviations attributed to environmental factors such as wind and currents. Obstacle avoidance capabilities were effective, with the ASV successfully navigating around detected obstacles in the test environment.

Figure 7: Obstacle courser, RoboBoat 2024 competition photo

Software	
Component	Description
ROS2	The primary framework for robot software integration.
OpenCV	Open-source computer vision and machine learning software library used for image processing tasks.
Micro-ROS	Robotics middleware framework used for microcontroller communication.
Py Trees	Python library for implementing behavior trees.
Ubuntu 20.04	Operating System
Hardware	
Component	Description
OAK-D Camera	Provides essential depth and object detection capabilities.
Jetson Nano	Powers the onboard processing for vision and navigation algorithms.
Pixhawk 5X	Serves as the flight controller, managing the drone's movement and stability.

Table 1: Software and Hardware components with descriptions

In Figure 8 shows the course of the route with notations from the. RC mode control. The most interesting is the curve starting from coordinates (-80,-40) and extending to (-40,-80), this is the section during which the ASV correctly avoided obstacles and navigated between buoys, as well as determined the points of navigation and pursued them, based on these results, the correctness and precision of the control was assessed

However, further analysis is required to determine the impact of environmental factors on system performance.

4.3 Experiments and Discussion

To minimize noise and errors while learning the object detection model, data was carefully prepared and filtered, mixing data collected directly from the camera, from simulations, and taken manually with the help of the camera yielding around 4,300 images, with over 11,000 total labels. The hyperparameter tuning of the YOLO v8n model was adjusted based on the validation results and the author's knowledge

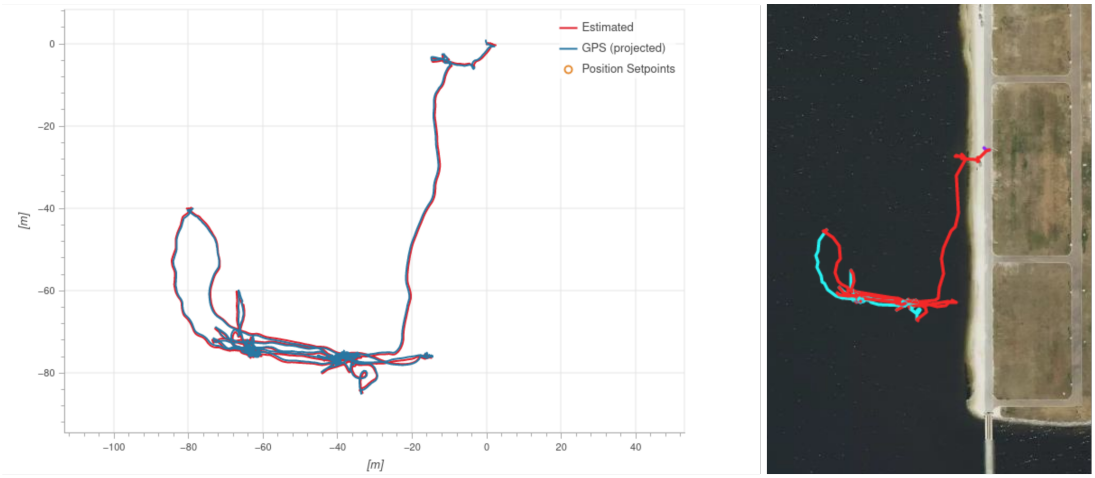

Figure 8: Example of ASV navigation trajectory during experimental trials.

together with the use of the "Hyperparameter tuning" functionality from Ultralytics [17] thus obtaining satisfactory results as shown in Figure 9.

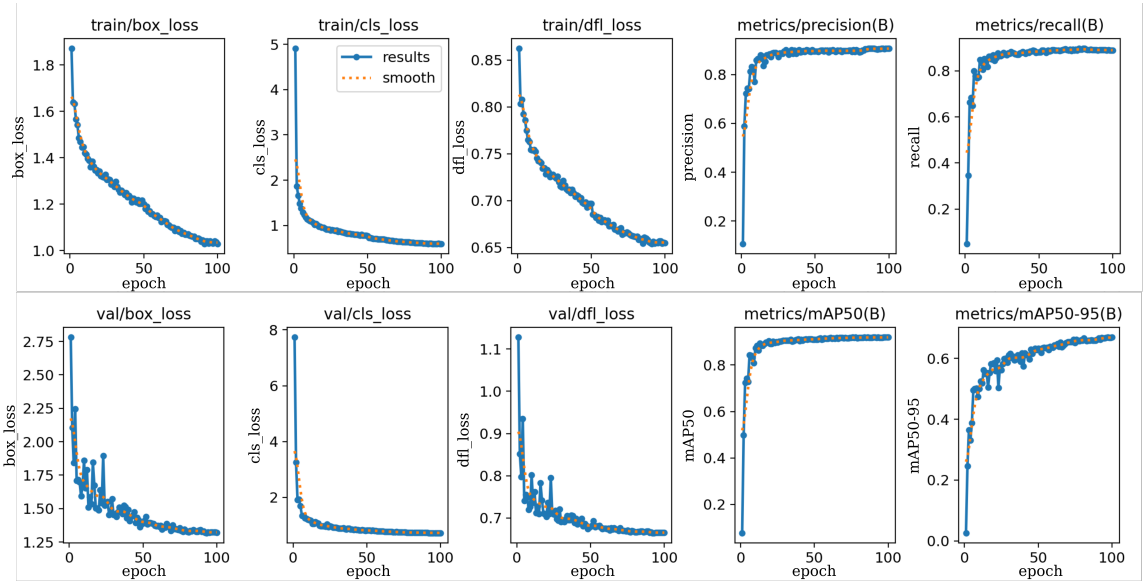

Figure 9: Training results of the neural network YOLO v8 over a period of 100 epochs

In this figure, box_loss denotes the bounding box regression loss, which measures the error in the predicted bounding box coordinates and dimensions compared to the ground truth. A lower box_loss means that the predicted bounding boxes are more accurate. Next, cls_loss means the classification loss, which measures the error in the predicted class probabilities for each object in the image compared to the ground truth. A lower cls_loss means that the model is more accurately predicting the class of the objects. The notation dfl_loss refers to the deformable convolution layer loss, a new addition to the YOLO architecture in YOLOv8. This loss measures the error in the deformable convolution layers, which are designed to improve the model's ability to detect objects with various scales and aspect ratios. A lower dfl_loss indicates that the model is better at handling object deformations and variations in appearance [16]

In figure 9 the "train" plots present results in a training part of learning our neural network, and the "val" plots are for a validation part, (B) refers to the "best" performance metric achieved by the model during the training process. The metrics: precision, recall, mAP50, and mAP50-95, served as essential tools for evaluating a model's performance in object detection. Precision quantifies the ratio of true positive detections to all positive predictions, thereby reflecting the model's effectiveness in minimizing false positives. In contrast, Recall measures the ratio of true positives to all actual instances, indicating the model's thoroughness in identifying all relevant objects. In both metrics, we achieved values close to 1 which we considered to be good results. mAP50 means the mean average precision calculated at an intersection over a union (IoU) threshold of 0.50. achieved values above 0.8 we considered very good. mAP50-95: Mean Average Precision averaged over (IoU) thresholds from 50% to 95% (at 5% steps), providing a more comprehensive view of the model's performance across different levels of detection difficulty. [17] Values above 0.5 were considered an excellent result for the training process.

As demonstrated in Figure 9, loss decreases exponentially in epoch scale for train and validation datasets. The proportion of true positives increases as shown in metrics/precision. In most cases, the proportion is above 0.9, which is a decent result. For labels that are rare in datasets like "duck" the proportion is 1.00 due to the distinct form of the labelled object and the lack of similar ones in the dataset.

However, the "background" prediction has zero cases that are correctly identified. While other labels represent a single object like a red buoy, the "background" label was used to denote every object, which may look similar to aforementioned labels. This fact is confirmed by a relatively high percentage of confidence for buoys and circles. This means that the label does not fulfill its role, as all examples of "background" labels were classified wrongly.

The prevalence of red and green buyos allowed to reach high confidence in prediction for these labels. Likewise, there are few instances of red circle, which may explain why the proportion is at 0.88 in Figure 9. Figure 10 shows results from the validation set. Real-time predictions from OAK-D cameras during RoboBoat competition with information about objects' class and their position coordination relative to the position of the ASV are shown in Figure 11.

Figure 10: YOLO v8 predictions showing on validation dataset during training process

5 Conclusions

This paper presents the ASV Rybitwa from the software design side of the autonomous unit. Design requirements and a detailed description of the system are presented. The paper focused on the presentation of the software architecture with an identification of the applied solutions available on the commercial scene and in literature, which have been tested under real-world conditions during the RoboBoat 2024 competition. The results presented demonstrate the effectiveness of the applied solutions but also show that there is an ongoing need for work to develop more reliable system capable of handling more complex tasks.

Future directions of development over ASV should focus on decision-making algorithms that allow response under uncertainty while focusing on providing explainable decisions.

Environmental conditions significantly influenced the ASV's performance during experimental trials. High wind speeds and strong currents posed challenges to navigation accuracy and obstacle avoidance.

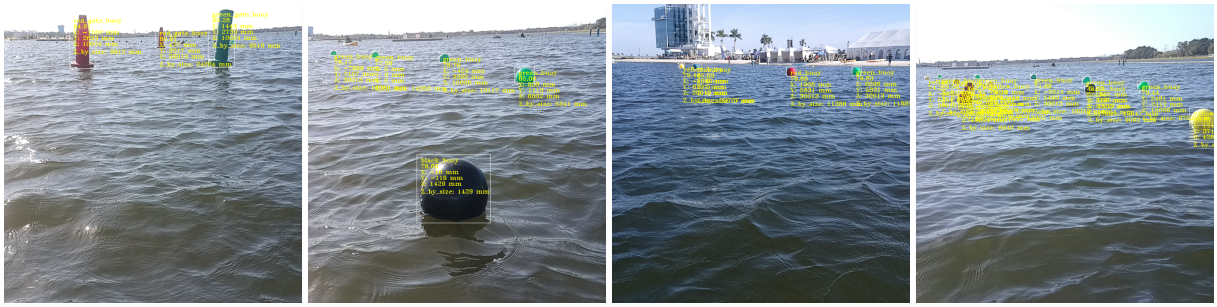

Figure 11: Performance of computer vision during experiments

Future iterations of the ASV system should incorporate robust control algorithms to adapt to varying environmental conditions and optimise performance. Overall, the experiments provided valuable insights into the ASV's capabilities and identified areas for improvement to enhance its performance in real-world maritime environments.

During RoboBoat 2024 ASV Rybitwa ended up in 7th place in the Autonomy Challenge out of 18 vehicles that participated in the competition. The vehicle performed more than well during tests and Qualifying Round. It excelled in areas that used to pose a significant challenge for our team during previous competition in 2023. These areas include: behavioural trees' ability to switch between different tasks, ASV-OCS data link range and reliability, hull shape, ground segment. Unfortunately during finals Rybitwa suffered a loss of a propeller which prevented it from scoring more points. Accurate range estimation also proved to be challenging at distances greater than approx. 5,5 metres. During our best run we managed to pass 9/10 gates with buoys (90%) and 2/2 (100%) gates with high marker buoys.

Acknowledgements
Norbert Szulc, M.Sc. for his insight, advice, support during tests and ongoing help with VCU's development and troubleshooting, F.D.C. Willard for long and contributing discussions.

References

[1] Poduval, D. R., & Rajalakshmy, P. (2022). A review paper on autonomous mobile robots. In AIP Conference Proceedings (Vol. 2670, No. 1). AIP Publishing.

[2] O'Rourke, Ronald, (2023). Navy Large Unmanned Surface and Undersea Vehicles: Background and Issues for Congress. Congressional Research Service. R45757. (Access: 21.05.2024) `https://sgp.fas.org/crs/weapons/R45757.pdf`

[3] Abrougui, H., Nejim, S., & Dallagi, H. (2023). Nonlinear Low-level Controller Design for an Autonomous Surface Vehicle. In *2023 IEEE International Conference on Advanced Systems and Emergent Technologies (IC_ASET)* (pp. 1–6). IEEE.

[4] Zubowicz, T., Armiński, K., Witkowska, A., & Śmierzchalski, R. (2019). Marine autonomous surface ship-control system configuration. *IFAC-PapersOnLine*, 52(8), 409–415. Elsevier.

[5] Johnston, P., & Poole, M. (2017). Marine surveillance capabilities of the AutoNaut wave-propelled unmanned surface vessel (USV). In *OCEANS 2017-Aberdeen* (pp. 1–46). IEEE.

[6] Molina-Molina, J. C., Salhaoui, M., Guerrero-González, A., & Arioua, M. (2021). Autonomous marine robot based on AI recognition for permanent surveillance in marine protected areas. *Sensors*, 21(8), 2664. MDPI.

[7] Xu, Haitong & Moreira, Lúcia & Guedes Soares, Carlos. (2023). Maritime Autonomous Vessels. Journal of Marine Science and Engineering. 11. 168. 10.3390/jmse11010168.

[8] Wang, N., Gao, Y., Yang, C., & Zhang, X. (2022). Reinforcement learning-based finite-time tracking control of an unknown unmanned surface vehicle with input constraints. Neurocomputing, 484, 26-37.

[9] Wang, N., & Karimi, H. R. (2019). Successive Waypoints Tracking of an Underactuated Surface Vehicle. IEEE Transactions on Industrial Informatics, 1–1. doi:10.1109/tii.2019.2922823

[10] Muhovič, J., Bovcon, B., Kristan, M., & Perš, J. (2019). Obstacle tracking for unmanned surface vessels using 3-D point cloud. IEEE Journal of Oceanic Engineering, 45(3), 786-798.

[11] Stanford Artificial Intelligence Laboratory et al. (2018). Robotic Operating System. Retrieved from `https://www.ros.org`

[12] RoboBoat 2024 - International student competition in robotics boats (Date of access: 21.05.2024) `https://roboboat.org/programs/2024/`

[13] Ögren, P., & Sprague, C. I. (2022). Behavior trees in robot control systems. Annual Review of Control, Robotics, and Autonomous Systems, 5, 81-107.

[14] DroneCode Foundation - PX4 Autopilot (Date of access: 11.08.2024) `https://px4.io/`.

[15] DepthAI - OAK-D Hardware Documentation 1.0.0 (Date of access: 11.08.2024) `http://docs.luxonis.com/projects/hardware/en/latest/pages/BW1098OAK.html`

[16] Reis, D., Kupec, J., Hong, J., & Daoudi, A. (2023). Real-time flying object detection with YOLOv8. arXiv preprint arXiv:2305.09972.

[17] Ultralytics - YOLOv8 documentation (Date of access: 11.08.2024) `https://github.com/ultralytics/ultralytics`

[18] Label Studio - LabelImg image annotation tool repository (Date of access: 11.08.2024) `https://github.com/HumanSignal/labelImg`

[19] splintered-reality - PyTrees library (Date of access: 11.08.2024) `https://py-trees.readthedocs.io/en/devel/index.html`

[20] T200 Thruster - Technical details (Date of access: 11.08.2024) `https://bluerobotics.com/store/thrusters/t100-t200-thrusters/t200-thruster-r2-rp/`

[21] DroneCode Foundation - uXRCE-DDS (PX4-ROS 2/DDS Bridge) (Date of access: 11.08.2024) `https://docs.px4.io/main/en/middleware/uxrce_dds.html`