RoboSub 2024 Technical Design Report

Troy High School SPEAR

Thomas Nguyen-Ta, Derek Peng, Yireh Ban, Kaleb Lee, Aidan Chen, Dylan Xiang, Joshua Kim, Bruce Deng, Yifei Zhang, Daniel Tran, Elvina Liou, Gavin Gibson, Jason Pan, Kaileo Truong, Shri Krishna Sivakumar, Landis Tien, Mateus Noronha, Ryan Zhou, Yongjing Li, Yun Long

***Abstract*— The Troy High School NJROTC RoboSub team's Autonomous Underwater Vehicle (AUV), Aura, was designed to consistently complete missions for the 2024 RoboSub competition and beyond. Newly built this year, the sub was designed to conduct the basics of autonomous movement and to be modular, for easier possible future improvements. With Simscale and Onshape, our team went through designs for the other components. Designing Aura allowed our team to integrate YOLOv8 for vision, ROS, a proportional integral derivative (PID) controller, and power distribution boards (PDBs). The use of only one AUV was primarily driven by the time and resource constraints, however, this allowed our team to focus on Aura's design and reliability.**

I. Competition Strategy

This year's competition, Logarithmic Spiral, is updated with a couple unique twists on previous years tasks. Despite this, the course remains to consist of 5 components:

(i) Enter the Pacific (Gate)
(ii) Hydrothermal Vent (Buoy)
(iii) Ocean Temperature (Bins)
(iv) Mapping (Torpedoes)
(v) Collect Samples (Octagon)

As a new team competing for the first year with an AUV, our overall approach to this competition was to tackle the most fundamental components of the course first, such as the gate and surfacing task, and then move onto tasks that were more complex in order to maximize the number of points that we could score consistently within the time allocated.

New:

A. Number of AUVs

As per rule 4.3.2 of the RoboSub 2023 Mission and Rules, each team is allowed to enter up to two vehicles. [1] While this option was considered by our team as it would decrease the time in the pool and improve task specialization, we ultimately decided against it due to the increased cost and complexity. Our team determined that investing our time and resources into one submarine would not only allow us to reduce complexity in our setup but also allow us to allocate more time towards perfecting advanced systems that would enable us to complete tasks with greater precision. As Sea++ runs more extensive trials, our team will modify our design according to the results.

B. Task Prioritization

As a first-year team, we recognized that giving each task an equal amount of time would result in our AUV not being able to properly complete any of the tasks. Therefore, during the planning process, we opted to allocate each of the tasks a different amount of time and prioritized the tasks in the following order: Destination (Gate), Start Dialing (Buoy), Location (Bins), Goa'uld Attack (Torpedoes), and Engaging Chevrons (Octagon). This was mainly determined by how equipped our sensors and tools were to focus on them.

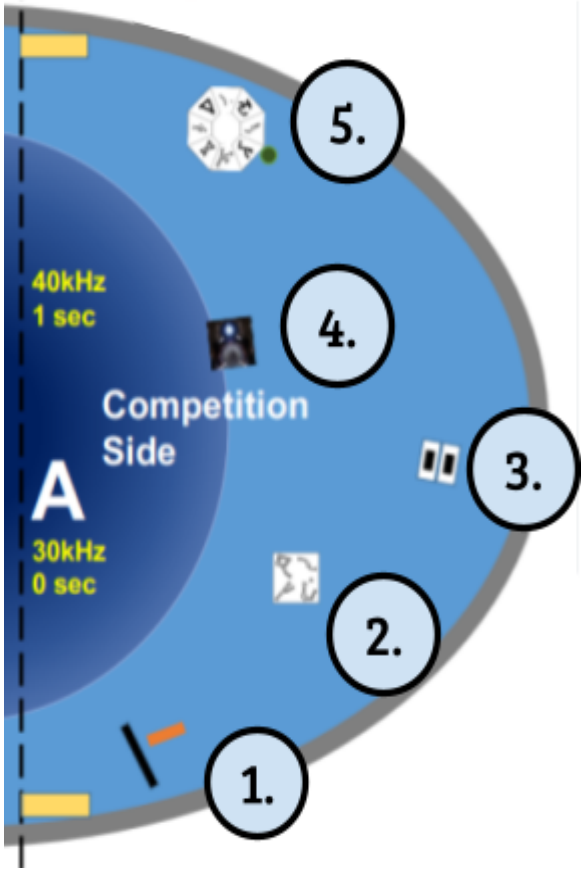

Fig. 1. The 2023 course diagram with task prioritization labeled

The order was also selected based on requirements (navigating through the gate is a required task and thus was our top priority), point values, and ease. It was also extremely efficient, as the AUV wouldn't need to repeat certain paths. The prioritization can be seen in the submarine's design and software.

II. Design Creativity

A. Overall Design (Mechatronics)

The main goal for Sea++ was to create an AUV that is reliable and modular, given our lack of manufacturing equipment. With this, our mechatronics subteam settled on using the BlueROV2 R2 frame, due to its high support for off-the-shelf components, and durable frame. Our design process was also heavily driven by the complexity our electrical and software subteams would have to endure implementing the solutions.

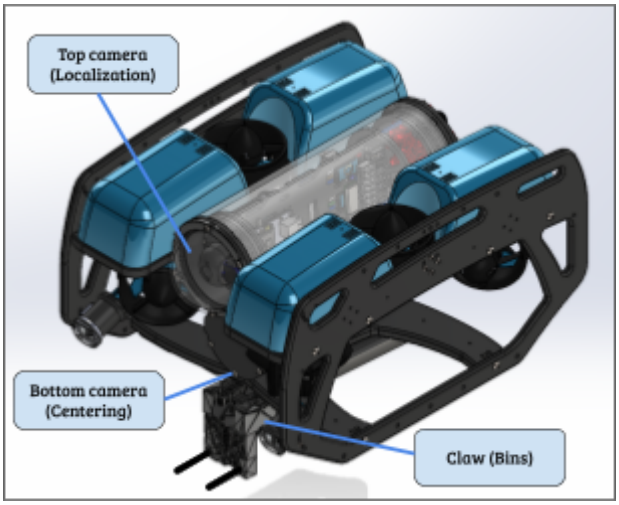

Fig. 2. Corner view of Sea++

(i) Cameras

Sea++ utilizes 2 mounted lowlight cameras with continuous video capture as its primary vision input. Both cameras have been placed perpendicular to each other to ensure Sea++'s efficient scanning of its environment. The front-facing camera is utilized for odometry and localizing the AUV's position relative to the gate and buoys. The bottom camera is mounted to detect the symbols and allow Sea++ to center itself accurately. This setup was chosen for its cost-efficiency, reliability, and simplicity.

(ii) DVL

As a backup solution for our vision system, Sea++ utilizes 2 BlueRobotics's Ping Sonar Altimeter and Echosounder for reliable object location and obstacle avoidance. The sonars are positioned to the immediate left and right of the forward facing camera, in the case our front facing camera fails or can't detect the next task. Computer vision is preferred over this method due to the versatility and resolution of the data extracted.

New: After consideration of the complexity of the use of sonars in Aura's systems and interacting with other teams, a DVL was implemented in place. Due to the inertial measurement unit having mediocre results, the DVL soon became essential as a result of its great underwater accuracy.

(iii) Kill switch

The kill switch consists of a waterproof switch which is attached on the rear of Sea++ allowing for ease of access. This switch is inline with the cables which carry power from the battery compartment to the upper compartment. This switch will kill power to the motors, as well as any other systems aboard Sea++.

(iv) Upgraded Claw

Last year, our team didn't get a chance to use our claw, but this year we were able to find

(iv) Torpedo System

Last yea

(v) Computer

Sea++'s onboard computer is an NVIDIA Jetson Nano. It runs most of Sea++'s software including the object detection algorithm, mission planning, and more. Our team chose to use an NVIDIA Jetson Nano over a Raspberry Pi due to its quicker I/O ports and better processor. Additionally, its compatibility with our software made it more straightforward to work with.

(v) Computer

Sea++'s

B. Software Overview

(i) Mission Planning

Our team narrowed down our options to either a Finite State Machine (FSM) or using a behavior tree. We ultimately chose to utilize a behavior tree, using BehaviorTree CPP with ROS for high-level decision making, due to its simplicity and flexibility. As a first-time team, its capabilities in abstraction and easy management of nodes made it a suitable choice for use. A FSM would overcomplicate our design and make scaling and further development exponentially more difficult. Additionally, its specific integration with ROS made it the optimal choice. Its ability to be configured and altered during runtime makes it easy to test and fix in a competition or testing environment.

(ii) Computer Vision

Our strategy in completing tasks, notably Start Dialing (Buoy), involved using computer vision for most navigation and decision making, especially as we did not utilize hydrophones this year. We used YOLO v4 and its resources in using Path Aggregation Networks and Cross Stage Partial Networks in conjunction with OpenCV and Tensorflow for optimization. [3] We opted to not utilize the newer versions of YOLO due to YOLO v4 being more accurate and faster for our needs. We utilized a loss function to minimize the error and used YOLO v4 to disentangle the data. In order to make it more accurate, we leveraged the extraction of the useful data. In the live environment, our computer vision uses edge extraction to only consider the edges of images that it uses to identify symbols in Start Dialing(Buoy).

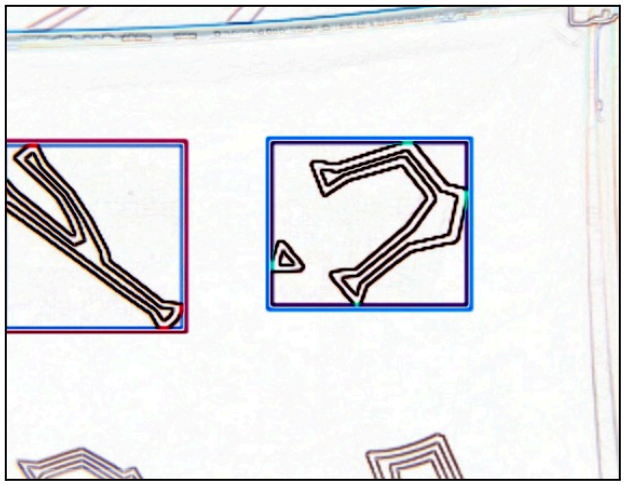

Fig. 3. Demonstration of edge extraction on symbols in Start Dialing

Though Sea++ utilizes the BlueRobotics Low-Light HD USB Camera, which is calibrated for underwater low-light conditions, we noticed glare and tint from the water impacted our computer vision performance. To combat this, we implemented Sea-Thru, an algorithm designed by Derya Akkaynak, which in turn also gave our team more flexibility for obtaining training data. [3] With minimal issues with color, our team can use training data from on land and in the simulator without the need for replicating the effect.

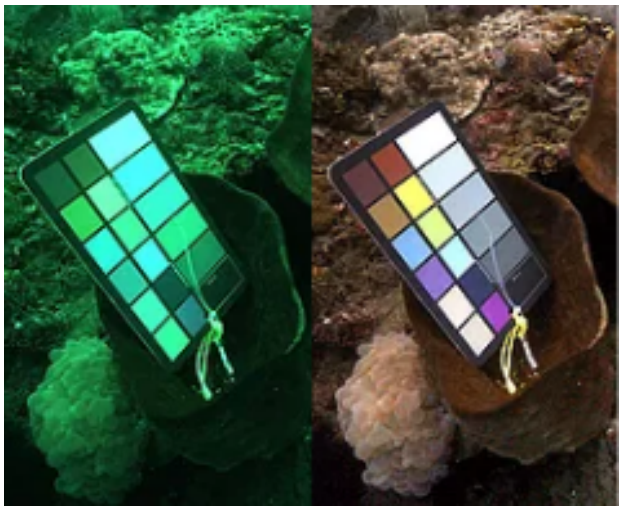

Fig. 4. Non-color corrected image on the left and image with Sea-Thru algorithm applied on the right

(iii) Architecture and Navigation

To operate efficiently and autonomously in an underwater environment, we rely on a series of processes. These processes work together in a coordinated and consistent manner, utilizing different algorithms to navigate and accomplish tasks. [4] For communication between different algorithms and parts of the system, we used the publisher/subscriber model provided by ROS, and then MAVROS to communicate over MAVLink to our Pixhawk PX4.

For navigation and localization, we used the IMU provided by the onboard Pixhawk and computer vision to identify our position within the environment. Our onboard sonar acted as a supplement, as it was not as accurate as vision localization in our research and experience. We have worked around the lack of a Doppler Velocity Log (DVL) due to the high cost and our low budget. Using proportional integral derivative (PID), Sea++ can adjust the PWM of the motor to get to the location more accurately according to the velocity collected by the IMU.

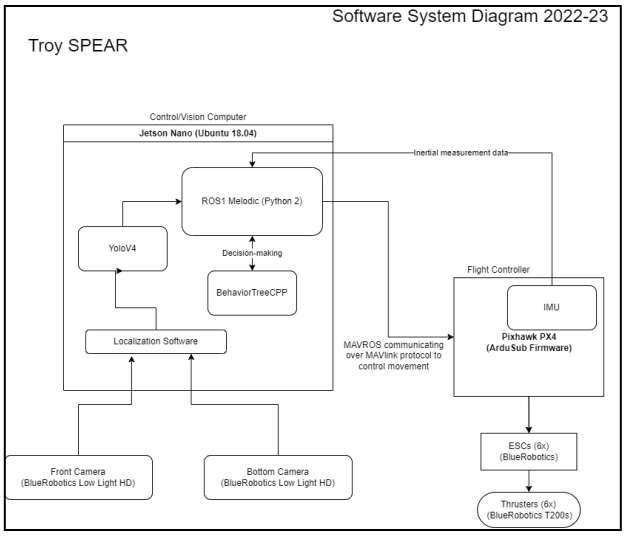

Fig. 5. Software Diagram illustrating our system organization

The steps taken by Sea++ after an object is detected are split into 3 stages:

Initial Detection - Sea moves around the area of the pool it is in until it recognizes an object significant to the current task.

Repositioning - Once the submarine finds the

object it is searching for, it repositions itself so that the object is in the frame of the forward-facing camera and on the same level as it.

Distance detection - Using localization with the known height and width of the object, Sea++ calculates its relative position to the task with the front camera.

III. Experimental Results

A. Cameras

The first aspect of Sea++ that needed to be tested was the vision algorithm. This was arguably the most integral aspect of Sea++ if it were to succeed in the competition. We needed to test if it was able to correctly identify objects and images based on the database we provided. Sea++'s vision algorithm utilizes a framework that allows for us to switch between databases in a single line of code. Thus, we are able to use different databases instantaneously to identify numerous images and objects.

Once we have completed the vision algorithm, we began the testing phase. We initially manually inputted images into the algorithm. Some of these images include an underwater shot of a pool to test as a baseline, and homemade orange markers and other pictures similar to the one used for the competition to see if the vision algorithm could successfully identify it. Once the algorithm could correctly identify all of the images, the same process was repeated for the cameras on the AUV.

For testing movement and testing without having to use the AUV, we used BlueROV2's simulator [5], powered by Gazebo. By using an emulated flight controller, Ardupilot's SITL, we were able to send commands, sending feedback from the simulator to the virtual AUV in a loop.

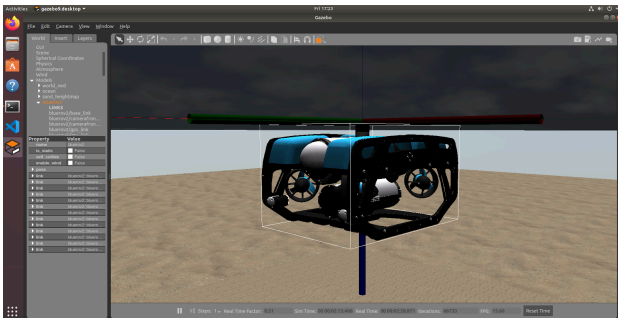

Fig. 6. BlueROV2 Simulator in Gazebo shown

For in-person testing, we used our school's pool. However, it lacked significant depth and therefore was not an accurate modeling of an actual competition environment. Therefore, more testing is required at the competition venue.

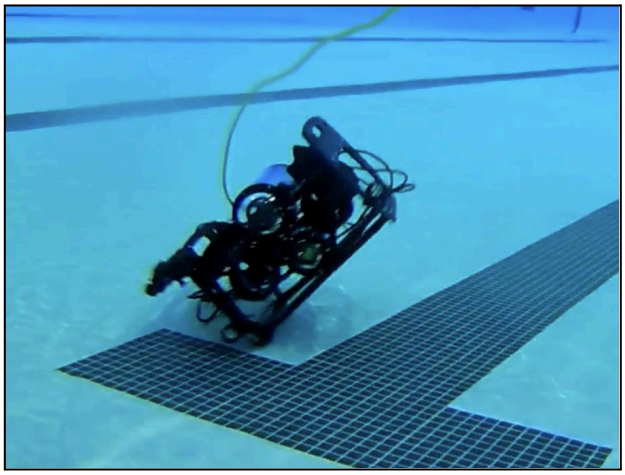

Fig. 7. Picture of Sea++'s during an underwater trial

IV. Acknowledgements

Our team could not have functioned without guidance from our mentor, Lt. Roger Fronek, and support from our generous sponsors. We would like to thank the following organizations for sponsoring our team: Troy High School NJROTC Booster Club, Raytheon Technologies, Armed Forces Communications and Electronics Association, Navy League of the United States Inland Empire Council, and the Navy League of the United States STEM Institute. Lastly, we would like to thank Team Inspiration and CSULA Robosub for helping us

better understand the competition and how to set up our AUV. Without the support of any one of our sponsors, we would not have been able to progress as far as we have.

V. References

[1] “Resources.” *RoboSub*, Available: robosub.org/resources/ (2023/06/16).

[2] Akkaynak, Derya. “Sea-Thru.” *Derya Akkaynak*, Available: www.deryaakkaynak.com/sea-thru (2023/06/16).

[3] Supeshala, Chamidu. “YOLO v4 or YOLO v5 or PP-YOLO?” *Chamidu Supeshala*, Available at: https://towardsdatascience.com/yolo-v4-or-yolo-v5-or-pp-yolo-dad8e40f7109 (2023/06/16).

[4] Iranmehr, Masoud. “YOLO v4 or YOLO v5 or PP-YOLO?” *Masoud Iranmehr*, Available at: https://pypi.org/project/mavros-python-examples (2023/06/16).

[5] “BlueROV2 ROS Simulation” *UUVControl*, Available at: https://github.com/UUVControl/bluerov2 (2023/06/16).

Appendix A: Component Specifications

Component	Vendor	Model/Type	Specs	Qty	Total Cost
Submarine	BlueRobotics	BlueROV2	Acrylic - 100m Fathom ROV Tether (ROV-ready) (1 Twisted Pair) (50m) Lumen Subsea Light (Pre-Connected Sets) (2) Newton Subsea Gripper	1	$4,050.00
Battery	Turnigy	Turnigy Graphene Panther 5000mAh 4S 75C	-	1	$81.74
CPU	Nvidia	Jetson Nano	GPU and 4 GB of RAM	1	$98.95
Camera	BlueRobotics	Low-Light HD USB Camera	-	1	$80.10
Pinger Localization	BlueRobotics	Ping Sonar Altimeter and Echosounder	BLUART USB to TTL Serial and RS485 Adapter	2	$558.00
3D Printer Filament	Polymaker	PolyMide™ PA6-CF	1.75mm 0.5kg	1	$44.99
Battery Bags	Amazon	Tenergy 2 Pack, Fire Retardant Lipo Bags	-	1	$11.99
Kill Switch	Amazon	Hmknana IP67 Waterproof Inline Cord Switch	IP67 12V-24V 20A	1	$14.99
Tether Spool	Amazon	Woods E103 E-103 Wheel	-	1	$17.63
Controller	Amazon	Logitech F310 Wired Gamepad Controller	-	1	$15.19
Torpedo Propulsion	Amazon	Crosman CO2 Cartridges	50-Count	1	$24.85
Algorithms: vision	-	-	Sea-Thru, YOLO v4 Object Detection	-	-
Algorithms: localization and mapping	-	-	-	-	-
Open source software	-	-	ROS Melodic, OpenCV, YoloV8, Tensorflow	-	-
Team size (number of people)	-	-	20 persons	-	-

Expertise ratio (hardware vs. software)	-	-	5 mech to 5 software	-	-
Testing time: simulation	-	-	14 hours	-	-
Test time: in-water	-	-	7 hours	-	-
Programming languages	-	-	Java, Python, C++	-	-