Technical Design Report: Software Development for Autonomous Underwater Vehicle (AUV) [SAUVC 2024]

Mrinmoy Kumar Das, Sri Harshavardhan Reddy Deverapalli, Aditya Rachakonda, and Aashita Parimi
Faculty Advisor: Prof. Sebastian Uppapalli

Mahindra University, École Centrale School of Engineering

Abstract. The software team of AUV-MU has developed Makara-2, an advanced and more capable successor to Black Pearl. While retaining a similar design philosophy, Makara-2 introduces significant improvements in reliability, endurance, cable management, computational power, aesthetics, and cognitive capabilities. This year, our primary focus is on enhancing the vehicle's vision systems, AI capabilities—such as object detection and obstacle avoidance—and control systems.

Keywords: Deep-learning · Object Detection · YOLO · Stereo Imaging · Image processing · Edge Computing.

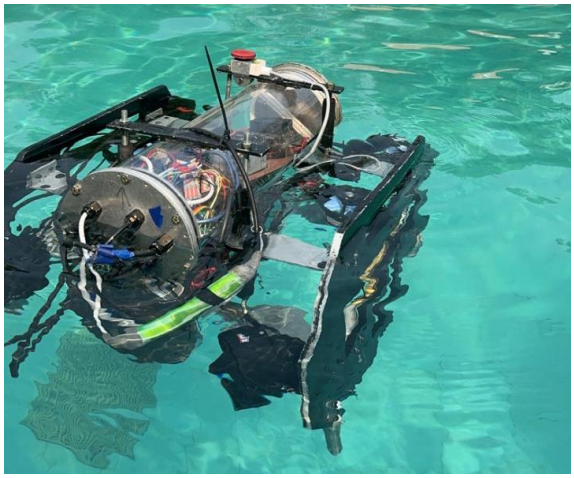

Fig. 1. Makara-2, the latest AUV developed by AUV-MU team.

1 Introduction

The Singapore Autonomous Underwater Challenge (SAUVC) 2024 consists of four main tasks: AUV navigation, visual identification, acoustic localization, and robotic manipulation. The team approached the competition with a major focus on maximizing accuracy and minimizing errors.

To achieve this, we made significant improvements to the software subsystem. Key advancements include optimization of the AI code to reduce operating temperatures, enhancing AI capabilities to run at 40 FPS, and implementing an auto-shutdown feature in case of extreme thermal throttling. Additionally, the previous object-detection model required manual supervision and various parameters for each retraining, which made the process inefficient. We have now streamlined this by developing an end-to-end pipeline, where only the dataset link and accuracy threshold are needed to automate training and evaluation.

2 Software subsystem

The software subsystem is implemented as a single stack with different packages representing various modules like vision, artificial intelligence module, and controls.

2.1 Vision

The vision system consists of two camera components: the ZED 2i (front camera) and two single cameras (bottom camera). Using the front camera feed, the vehicle is capable of efficient navigation and obstacle avoidance. The input from the ZED 2i camera and the depth map generated are shown below. Additionally, a comparison table between the two cameras is provided below.

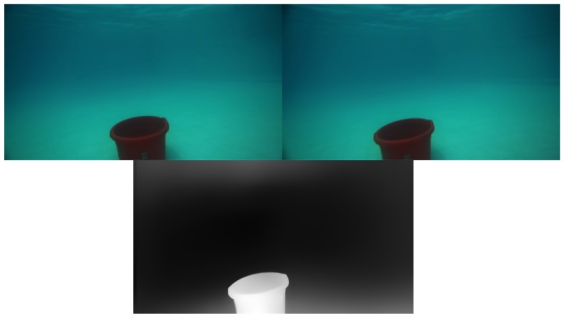

Fig. 2. Input image: Front camera feed from the ZED 2i showing the live view used for navigation and obstacle detection. **Output image**: Depth map generated, illustrating the environment's 3D structure.

Table 1. Comparison between ZED 2i and Waveshare IMX219-77 cameras

Feature	**ZED 2i**	**Waveshare IMX219-77**
Manufacturer	Stereolabs	Waveshare
Field of View (FOV)	Horizontal: 120°	Horizontal: 77°
Stereo Depth Sensing	Yes, supports stereo depth mapping	Can be achieved using two single cameras
Frame Rate	Up to 100 FPS (varies with resolution)	Up to 60 FPS at 1280 x 720
Connectivity	USB 3.0	CSI (Camera Serial Interface)

2.2 Artificial Intelligence Module

The AI module runs on the onboard computer, which is a Jetson Xavier. Upon receiving power and booting up, the navigation script initiates, commanding the vehicle to keep surging until an object is detected. Based on the location of the detected object, the appropriate commands are sent to the Arduino to control movement. The movement commands are discussed in detail in the next section. This section is divided into three parts: (a) Data collection, (b) Model selection and training, and (c) Model evaluation.

Data Collection and Dataset formation: The initial data was captured in the university pool, and these images were then augmented using various techniques such as flipping, shearing, adding noise and blur. The dataset was further combined with other datasets obtained from Roboflow. This resulted in an overall dataset size of approximately 9,000 images. Some sample images from the dataset are shown below.

Fig. 3. Images of gate, blue drum, red drum from dataset.

Model selection and Training: To train the ML-based vision subsystem for object detection and obstacle avoidance, we implemented a pipeline that automatically detects tasks in the camera feeds and draws bounding boxes around them. This pipeline has significantly reduced the time required compared to the previous manual labeling process. The initial object classes included: Gate, Ball, Blue Drum, Red Drum, Blue Flare, Yellow Flare, and Red Flare. To improve accuracy, we reduced the classes to: Gate, Drum, Flare, and Ball, and employed traditional computer vision techniques to identify the color of the drums and flares.

For enhanced accuracy, we adopted a multi-modal approach where one model was exclusively used for gate detection, another model for drum and flare detection, and a third model for ball detection. This approach simplified the task into single-class and binary-class classification problems, further boosting accuracy. We explored various object detection architectures, including YOLO V5n, YOLO V7, and YOLO V8n.

To train these models, we implemented an end-to-end pipeline that takes the dataset and model architecture as inputs and continuously trains the model using different hyperparameters until the accuracy exceeds 80%. The flowchart for the training pipeline is shown below.

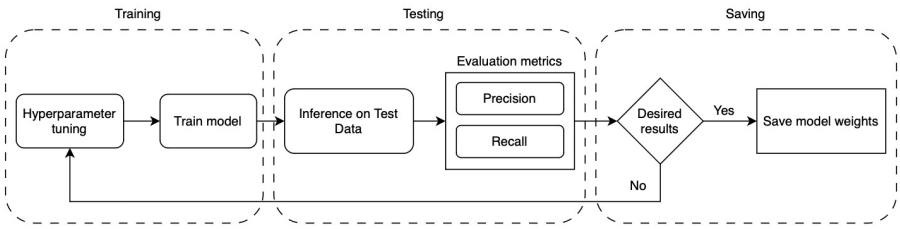

Fig. 4. Flowchart representing end-to-end pipeline for object detection model training.

Model evaluation: The criteria used to evaluate the accuracy of the models are mainly: **Precision** and **Recall**. Precision is defined as true positive estimation over whole estimation, whereas Recall only provides positive estimation. The following equations represent how the above parameters are computed:

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

Here, TP and FP stand for True Positive and False positive, while TN and FN stand for True negative and False negative respectively.

Table 2. Comparison between different Object Detection Architectures for binary-class object detection

Architecture	**Precision**	**Recall**	**FPS**
Yolo V5n	89%	28%	38
Yolo V7	83%	32%	34
Yolo V8n	94%	23%	42

2.3 Controls

The control system of the AUV was designed to manage its movements with precision and responsiveness. Initially, we developed the basic control logic, which involved creating a set of fundamental movement commands such as surge (forward and backward), lateral (side-to-side), and vertical (up and down) motions.

The movement commands are given based on the location of the vehicle and environmental feedback from the object detection and vision module. These commands are then communicated to the AUV's hardware components through the Arduino Mega, which serves as the intermediary, translating high-level instructions into precise thruster controls. This approach ensures that the vehicle can autonomously make informed decisions about its navigation and obstacle avoidance.

Table 3. Control Commands for Vehicle Movement

Control Command	**Description**	**Thruster Action**
Surge (Forward)	Move the vehicle forward	Front thrusters push water backward
Surge (Backward)	Move the vehicle backward	Front thrusters push water forward
Yaw (Turn Left)	Rotate the vehicle counter-clockwise	Left thrusters push water to the right
Yaw (Turn Right)	Rotate the vehicle clockwise	Right thrusters push water to the left
Sway (Left)	Move the vehicle to the left	Thrusters push water to the right sideways
Sway (Right)	Move the vehicle to the right	Thrusters push water to the left sideways
Heave (Up)	Ascend vertically	Vertical thrusters push water downward
Heave (Down)	Descend vertically	Vertical thrusters push water upward

3 Implementation

The system processes the input image captured by the camera, performing object detection to identify and localize the target object. Once the object is detected, a bounding box is drawn around it, followed by the generation of a depth map to estimate the distance to the object. The detected object's position relative to the center of the frame is analyzed to determine the appropriate movement commands. For example: if the object is located to the right of the center, a command is sent to yaw in a clockwise direction until alignment is achieved (represented by the black and red lines). Once the alignment is complete, the AUV is commanded to surge forward towards the object. The below figure represents the working of the software subsystem.

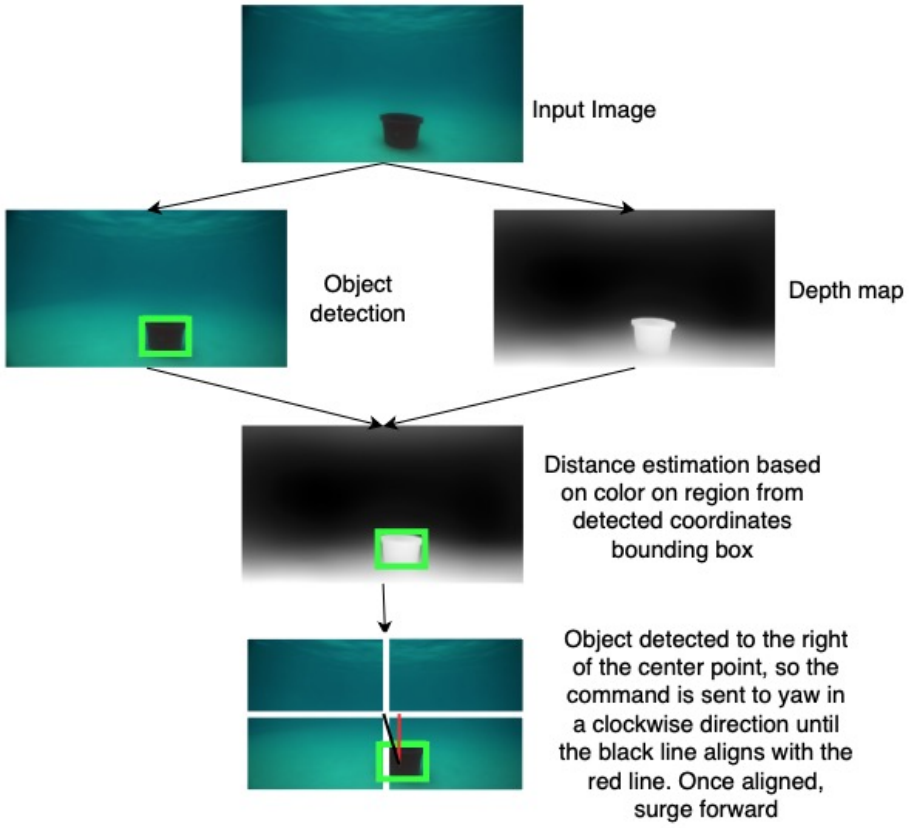

Fig. 5. Workings of software subsystem

This implementation combines computer vision techniques along with control logic to ensure efficient navigation and obstacle avoidance.

4 Conclusion

Through rigorous design, analysis, and testing by the software team, the new AUV software subsystem now has the flexibility to incorporate new AI models without requiring significant developmental changes. This will allow the team to focus more on system testing and controller fine-tuning in the future. Key advancements, such as the development of end-to-end pipelines and auto-labeling, provide a reliable and efficient interface for AI development and testing. Makara-2 presents a valuable opportunity for the team to further advance research in underwater and autonomous robotics.

5 Acknowledgement

We would like to express our sincere gratitude to the Vice-Chancellor for providing financial support and facilitating our participation in SAUVC 2024. We are also grateful to the Supercomputer Lab at Mahindra University for granting us access to GPUs, which was essential for training our deep-learning models. Special thanks go to our vendors, Blue Robotics, ElectronicsComp for their timely assistance and technical support throughout the project.