RoboBoat: Camera-LiDAR Fusion for autonomous boat navigation

SOHYUN YOO, YUVANAND SARAVANAN, MIA KHATTAR,
SHIRLEY BIAN, and BELLA JEONG, University of Califonria, San Diego

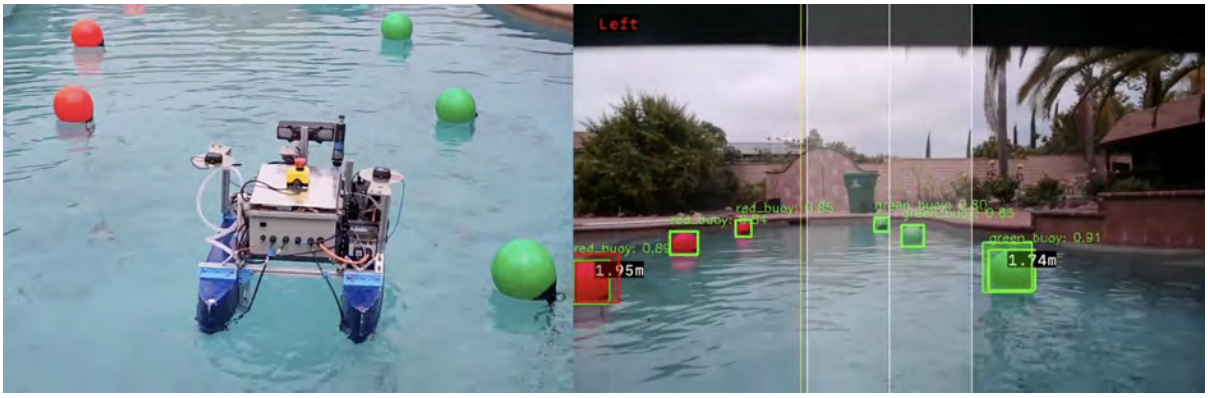


Fig. 1. World view of the test environment (left) and visualized navigation system in the camera's POV. The boat takes the midpoint (yellow) of closest red and green buoy (labeled with distance) to determine the steering command (Top left of right).

1 Abstract

The RoboBoat Sensor Fusion project aims to demonstrate that depth estimation using LiDAR, on top of accurate object detection, enhances autonomous surface vehicle navigation in dynamic water environments. The annual RoboBoat competition challenges robotic boats to navigate through a trajectory using computer vision and path planning. To meet these demands, Team Inspiration's RoboBoat is equipped with a long-range camera and a LiDAR sensor. This project demonstrates the fusion of RGB camera data with LiDAR depth data and a new path planning algorithm that takes advantage of the new depth information. As a result, the boat can follow paths outlined by buoys much more precisely, significantly reducing deviations and improving overall navigational accuracy.

2 Introduction

The annual RoboBoat competition brings together robotics teams from universities around the world to test their autonomous surface vehicles (ASVs) in real-world maritime operations. One of the challenges, the Follow the Path task (Figure 3a), requires ASVs to navigate a dynamic aquatic environment using computer vision, path planning, and obstacle avoidance. Pairs of red and green buoys form checkpoints that each ASV must clear throughout a long course using careful maneuvering. Black buoys are placed along the path as general obstacles, while yellow buoys are "duck sightings" that the boat must count as it observes them. In general, the boats must keep track of the targets while staying on the path and avoiding collisions. In this work, we consider only the red and green buoys for navigation.

To address these challenges, we explored sensor fusion as a means of improving perception and reliability. Our vehicle, Team Inspiration's RoboBoat, initially had a camera-only implementation of a navigation algorithm. This system proved unreliable under harsh lighting conditions and was ultimately abandoned. While the OAK-D LR camera on our boat can run an object detection model and accurately identify objects, its performance degrades in bad lighting, and it also cannot accurately provide depth information. LiDAR, on its own, can accurately

measure distances to objects, but struggles with object recognition because point clouds are sometimes sparse. Camera-LiDAR sensor fusion combines the best of both worlds: RGB data from the long-range camera and depth data from the LiDAR sensor. This project implements camera-LiDAR sensor fusion and evaluates its performance through multiple water tests.

The main contributions of this report are: (1) the methodology for training a fast and reliable buoy detection model, (2) the process of projecting and filtering LiDAR depth data onto the camera view, and (3) the implementation of a path planning algorithm that takes advantage of the distances of each object. For the buoy detection model, we compare ROS2 architectures for running inference on different hardware platforms. For sensor fusion, we discuss how we project LiDAR points onto the 2D image plane and filter out noise that causes inaccuracies. Finally, for navigation, we demonstrate some approaches to using depth information to calculate the trajectory of the boat and how we send the corresponding motor commands, as well as how we have tuned the thrusters during our tests.

3 Related Works

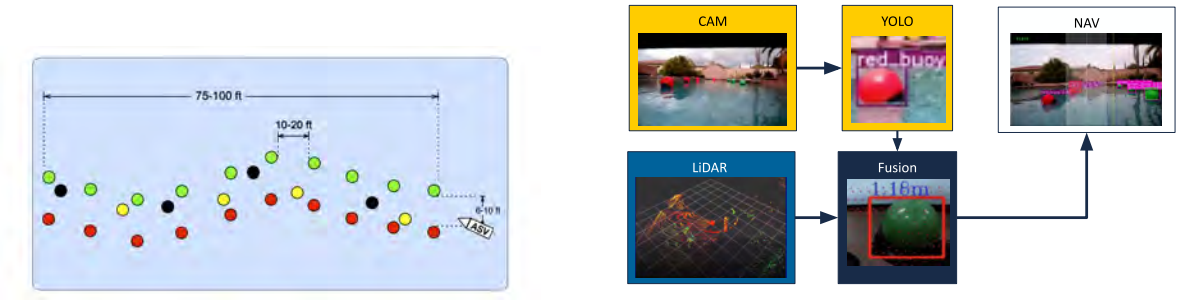
3.1 Object Detection

Employing machine learning models for object detection has become a standard approach in the field of computer vision in recent years. The YOLO (You Only Look Once) family of models [5] has become foundational in real-time object detection accredited to its balance of speed and accuracy. YOLO formulates object detection as a single regression problem, predicting bounding boxes and class probabilities directly from full images in a single forward pass. Subsequent versions, including YOLOv3, YOLOv4, and the recent YOLOv5 and YOLOv8 by Ultralytics, introduced improvements in backbone networks, anchor boxes, and training strategies to enhance performance on both low-power devices and large-scale systems. In our work, we adopt the lightweight YOLOv8 model to perform buoy detection onboard a Jetson device, leveraging its efficiency to meet the real-time constraints of autonomous navigation on water.

3.2 Sensor Fusion

Sensor fusion is crucial in autonomous driving systems due to the limitations of individual hardware sensor components. GPS may fail in various subterranean geographic situations (e.g., tunnels and caves). IMU measurements can be corrupted by noise and bias. The camera's view may be obstructed by glare and physical objects. LiDAR may fail in structureless environments (e.g., an open lake) [4]. Multi-sensor fusion mitigates the shortcomings of individual sensors, thus providing the vehicle with a more robust and reliable navigation system [4]. The most prevalent sensors used in multi-sensor fusion are visual sensors (e.g., camera), LiDAR, and IMU [4]. Our team explored working with all three sensor types this quarter, beginning with camera and LiDAR.

Visual-LiDAR fusion algorithms form navigation decisions by combining input from both the camera and LiDAR sensors. Our team committed to implementing this form of sensor fusion in the RoboBoat this quarter. Visual-LiDAR fusion integrates 2D visual information from the camera with the depth information from the LiDAR to construct a 3D representation of the vehicle's environment. These fusion algorithms may be classified as either loosely-coupled or tightly-coupled [4]. Loosely-coupled algorithms are simple, extendable, and require low computation power, while tightly-coupled algorithms offer improved accuracy and robustness [4]. One tightly-coupled approach is to use V-LOAM to aid in estimating the current camera pose[1]. However, this system depends on various external factors, so our team did not adopt this method [1][6]. Loosely-coupled algorithm approaches include registering a depth map with camera poses and point cloud data, or projecting LiDAR points



(a) Follow the Path task from RoboBoat 2024.

(b) Overview of our pipeline.

Fig. 3

onto the vehicle's camera view [3][9]. Our group opted to take an approach similar to the latter, due to compatibility with the existing RoboBoat software architecture and previous sensor fusion prototypes. Having successfully completed our initial goal, we began prototyping the integration of inertial components into the navigation system.

LiDAR-visual-inertial multi-sensor incorporates additional input sources: camera, LiDAR, and IMU or GPS. Although this system represents one of the overarching goals of the RoboBoat team, it was not among the deliverables our team committed to this quarter. Despite this, we began prototyping the system to support future teams' development efforts. This category of sensor fusion is particularly robust, as it integrates depth information from the LiDAR, visual data from the camera, and location data from the IMU or GPS. A loosely-coupled approach includes methods such as VIL-SLAM, which combines stereo cameras with LiDAR and IMU input [8]. In contrast, tightly-coupled approaches employ sequential, multilayer processing pipelines which predict motion using IMU measurements and visual–inertial odometry [2]. A hybrid approach merges both strategies to construct a pipeline which integrates IMU odometry, visual–inertial odometry, and LiDAR–inertial odometry [7]. Future RoboBoat teams may adopt this hybrid method due to its compatibility with the vehicle's existing software architecture and the prototype developed by our team. For the prototype, we adopted Fast-LIO (Fast LiDAR-Inertial Odometry) package based on Fast-livo2 [10], which uses LiDAR and IMU for odometry.

3.3 Previous RoboBoat system

The prior CSE RoboBoat team's pipeline solely relied on the camera input. Our team selectively adapt the previous team's features and extend on the work to improve navigation and minimize latency.

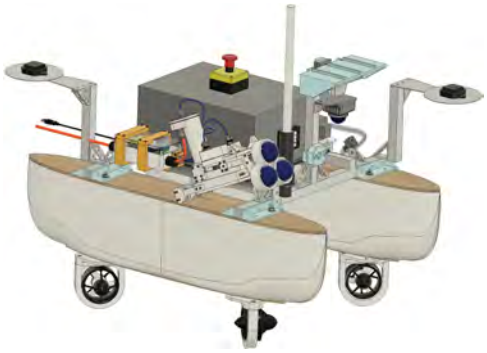


Fig. 2. Overview of the boat.

4 Technical Materials

4.1 Overview

Figure 3b shows an overview of our pipeline. For sensors we have a camera receiving RGB frames and LiDAR which retrieves the point cloud data. With the RGB frames, we extract the bounding boxes of the buoys using an object detection model. Then we project the LiDAR points on camera view to extract the depth data from the bounding boxes. Finally, we use the xyz information of buoys for navigation.

4.2 Hardware Setup

Figure 2 shows rendered 3D model of the boat. Our boat is equipped with four T200 thrusters with PWM values from 1300 to 1700. At the front, we mounted a Livox Mid-360 LiDAR facing downward to capture rich data from the front-bottom area, where buoys are typically located. On top of the boat, we have the OAK-D LR camera at an elevated position to minimize water splashes. The electrical box of the boat contains electrical speed controllers (ESCs) for the motors, along with Arduino. The main single-board computer is a Jetson Xavier, which processes the data from the LiDAR and camera.

4.3 Object Detection Model

Model Training

To train the buoy detection model, we used a dataset of 15000 buoy images by MHSeals on Roboflow, as well as a GitHub repository by MHSeals that contains a script to train the model. The dataset consists of images of many different colored buoys from previous competitions. We trained a YOLOv8 model using the training script, which generated a PyTorch (.pt) model which we could use to run inference on camera frames on the Jetson.

Latency Optimization

In our initial two ROS2 architectures, the Jetson received raw data from the camera, and inference would be run on the Jetson's GPU, as shown in the leftmost architecture from Figure 4a. Our ROS2 workspace used Python 3.10 and Ubuntu 22.04. To enable the use of CUDA cores for inference, we manually built PyTorch from source with CUDA 11.4 support to enable GPU acceleration inside a Docker container. Furthermore, to receive a stream of camera frames in a ROS2 node, we used Luxonis's DepthAI API to configure the camera pipeline for video output.

The following snippet shows how we linked the RGB camera output to the video stream.

```
cam_rgb.video.link(xout_video.input)
```

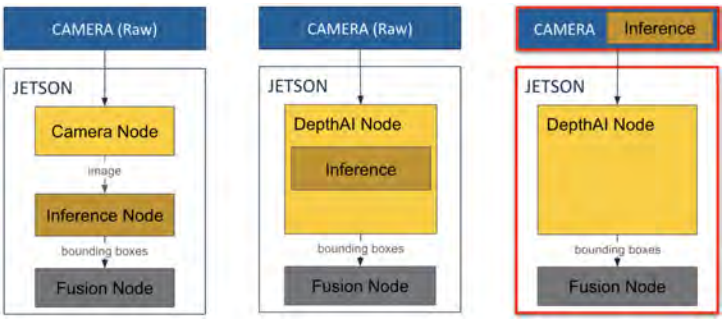
During processing, each frame is passed through the model.

```
results = self.model.predict(frame, verbose=False, device="cuda")[0]
```

Then the object class ID and bounding box coordinates are extracted from each detection for a frame, and are published to a ROS2 topic that the sensor fusion node subscribes to.

```
for box in results.boxes:
    x1, y1, x2, y2 = map(float, box.xyxy[0].tolist())
    conf = float(box.conf[0])
    class_id = int(box.cls[0])
    # add extracted items to detection_array
self.bbox_publisher.publish(detection_array) # publish bounding boxes
```

During our water tests, we took rosbag recordings that included a topic to the annotated frames and bounding box arrays around visible buoys. From the initial setup with ROS2 node subscribing to the camera frame, we observed that the publish rate of bounding boxes was significantly low. The frame rate was often much lower than 10 frames per second (FPS) even with the GPU usage for inference. The next attempt was to adapt DepthAI pipeline which allows more direct access to the camera frame. This approach led to notable improvements in latency (see Figure 4b). We also discovered that the model could run on the OAK-D LR camera itself. We converted the model to a special format (.blob) for the camera using Luxonis's DepthAI Tools, then created a similar DepthAI pipeline to our previous iteration, resulting in an architecture that looks like the rightmost diagram in Figure 4a. The main difference is that, this time, the camera outputs detections (as well as raw frames for visualization purposes)



(a) Variants in architecture for the object detection system.

(b) Comparison in latency (offset labeled with yellow.) Optimized architecture (bottom) introduces lower latency than the initial choice (top).

Fig. 4

rather than only raw frames. The modified DepthAI node simply extracts the bounding boxes from the camera detections. It then annotates the raw camera frames (again, for visualization) and publishes the bounding boxes. With this new implementation, the bounding boxes were being published at a stable 13-15 FPS, giving us a much smoother stream of bounding boxes per frame and a much smoother video for visualization. Now that inference is offloaded to the camera, the Jetson is able to devote more of its resources to sensor fusion.
Note that we used a vanilla PyTorch model and did not try TensorRT due to time limitations. Using a TensorRT model could improve the performance of inference on the Jetson.

4.4 Sensor Fusion

LiDAR Point Projection

In order to combine the LiDAR points and the images we receive from the camera, we need to map the point clouds from the 3D LiDAR space onto the 2D image space received from the camera. For this task, we need to perform matrix multiplication as shown in the equation.

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Fig. 5. Projected LiDAR points on image. Points colored from closest (red) to furthest (blue)

We obtain the camera intrinsic matrix by calculating its optical center (f_x, f_y) and scaling factors (c_x, c_y) via camera calibration. The extrinsic properties involve camera rotation $(r_i j)$ and translation (t_k), which we obtain by performing camera and LiDAR calibration using the MATLAB Camera and LiDAR calibration tool. This requires us to collect 20 pairs of image and LiDAR point cloud data with a checkerboard at different positions in frame. After obtaining the intrinsic and extrinsic matrices, we perform projection on each of the LiDAR points with their x, y, z coordinates to obtain their 2D coordinates on the image (u, v). The final calibration results were within a range of 0.03m and 6 degrees. An example of the result for LiDAR point projection is shown in Figure 5.

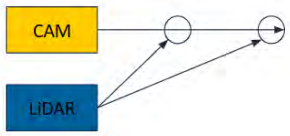


Fig. 6. Physical displacement of camera and LiDAR sensor - causing missing or overlapped mapping of LiDAR points on image

We had to collect multiple groups of paired data in order to minimize some projection errors that affected depth estimation. During

calibration, we observed noise from LiDAR points that are supposed to be out of frame, and inconsistent LiDAR point mapping where either some area on the image is not mapped by any points or is mapped by LiDAR points with very different depth information. We fixed the former through filtering points so that we do not project LiDAR points up to the POV of the camera. We believe that the latter was caused by the physical positions of the sensors, as shown in Figure 6. Due to the structure of the sensors, it is very difficult to completely eliminate this error even in other arrangements of the sensors on the boat. Thus, we decided to solve this issue in the software by selecting LiDAR points for depth estimation. This will be explained in more detail in the next subsection.

Depth Extraction from Object Detection
For each projected LiDAR point, we can obtain its relative depth from the boat by calculating the Euclidean distance in the LiDAR space with the LiDAR sensor being the origin. Combining these with the YOLO object detection model, we are able to estimate the relative distance of the detected buoys to the boat through the depth information given by the LiDAR points in the bounding boxes of the recognized buoys. We first tried averaging the depth of all the LiDAR points in a bounding box to obtain the buoy's depth. However, due to the inconsistent LiDAR mapping mentioned in the previous subsection, this approach was greatly affected by the LiDAR points that are mapped incorrectly. We resorted to using the closest point in the bounding box, which gives us much more accurate distances when estimating the buoys' distances.

4.5 Path Planning

Our path planning algorithm evolved significantly throughout the project as we identified major limitations in the original design and progressively enhanced it with depth-based logic.

Baseline: Camera-Only Navigation
Initially, following the previous CSE Roboboat team's approach, the navigation relied entirely on visual detection from the camera. The object detection model detected red and green buoys in the frame, and navigation decisions were made by calculating the midpoint between their horizontal (X-axis) positions. If the midpoint was left or right of center, the boat would steer accordingly (ex: if near center, the boat would go straight). To smooth out noisy detections, a short history of buoy positions was maintained using Python's `deque`. However, the algorithm had two major limitations. First, it always used the most recent single buoy detection as a reference, without verifying whether the buoy was spatially relevant. Second, it operated under the assumption that all detected buoys were valid, which often led to instability during turns and unreliable path-following behavior, especially in cluttered scenes.

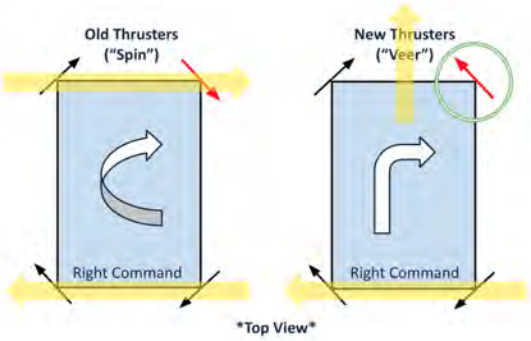
Our Approach: Depth-Aware Navigation
To address prior limitations, we implemented a depth-aware navigation algorithm using LiDAR and camera data. Our ROS2 node, `lidar_camera_projection_node`, which is the sensor fusion node explained in the previous section. From all detected buoys, the node selects the closest valid red and green buoy based on 3D depth, and publishes their center coordinates and depths to the `/centroids` topic. This introduces a key difference with the previous approach, which relied on the most recent detection without considering spatial reliability. By using the closest buoys, our path planning module forms more accurate midpoints and generates smoother steering commands, reducing instability and avoiding misinterpretation of off-angle or outdated detections. In conclusion, by shifting from a 2D top-down heuristic to 3D-aware approach using depth data, our updated path planning module addressed key limitations of the previous system. Rather than relying on Y-axis sorting, we

selected the closest valid red and green buoys using 3D Euclidean distance, allowing for more accurate midpoints and preventing over-steering. Figure 7 shows a visualized comparison between the two algorithms.

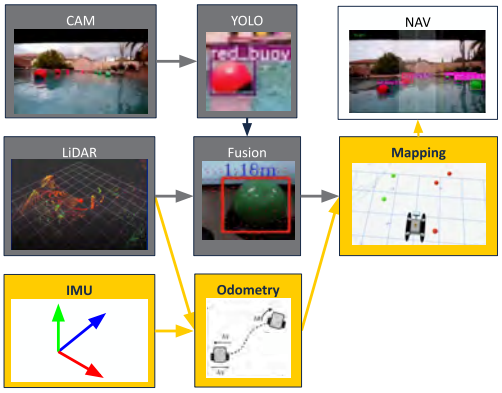
Fig. 7. Left: Initial Camera-based Path Planning Right: Sensor Fusion-based Path Planning

Motor Command Adjustment: "Spin" to "Veer"

In our initial setup, left and right turning commands used a spin-based configuration where the top and bottom thrusters applied opposing lateral forces. For example, in a right turn (see left in Figure 8a), the top thrusters produce rightward movement, while the bottom thrusters result in leftward movement. These opposing forces canceled forward motion and created in-place rotation, which we refer to as a "spin". This method caused the boat to over-rotate and drift off course during turns, especially when the boat needed to adjust quickly after passing a buoy gate. To improve this, we transitioned to a "veer" strategy. Instead of generating opposing spin forces, we modified one of the top thruster vectors (shown as green circled thruster in Figure 8a) to point diagonally forward. This way, the front two thruster vectors combined into a forward-pointing vector, guiding the boat smoothly into a turn while maintaining forward momentum.



(a) Right Command: Spin vs. Veer Approach



(b) Extended Pipeline with localization and mapping. New features labeled with yellow.

Fig. 8

Fig. 9. Visualized prototype of mapping (left) with camera view (right).

4.6 Localization & Mapping System

Looking ahead, we began prototyping a localization and mapping system with IMU sensor at the end of the project. The main motivation was to develop a more robust navigation system by storing information over time, unlike the existing system, which only stores the latest 20 detections. Persistently storing and updating obstacle information is particularly valuable when sensors experience temporary failure or noise. This also allows the robot to know obstacles behind it passed, which is not possible when relying solely on the front vision.

There are several ways to implement this functionality; our method simply extends the original pipeline for scalability and reusability. The extended system, illustrated in Figure 8b, incorporates IMU sensor and utilizes the Fast-LIO package to obtain the odometry. The detected buoys are projected into 3D space and used to update the map via Extended Kalman Filter. This allows the robot to navigate using a map rather than relying solely on live, potentially unstable detections. While we successfully implemented and visualized the prototype to support future integration, due to time constraints, this system was not fully tested for navigation performance.

5 Milestones

Refer to Table 1 and 2 for our progress in milestones. The milestones are labeled as: Completed , Delayed , Added , Deleted . We decided to drop the counting algorithm and larger water test to focus on the current system rather than going beyond the scope. Most of the delay was due to the need for in-person water testing.

6 Conclusion

6.1 Accomplishment

In this report, we presented the design and integration of our buoy detection model, sensor fusion algorithm, and path planning algorithm to improve the robustness of the RoboBoat's autonomous navigation system. By training a reliable buoy detection model and deploying it directly on the camera, mapping LiDAR points to 2D buoy detection boxes, and modifying and fine-tuning the path planning algorithm and thruster power with the new LiDAR depth information, we significantly improved the system's perception and decision-making capabilities.

6.2 Future Work

As can be seen from inaccurate mapping from Figure 9, there are several potential improvements that could be made with localization and mapping. As the back-projection stage relies on both odometry and the depth data from the fusion node, any noise in these inputs can significantly degrade the mapping accuracy. Approaches to mitigate this issue include: (1) Tuning the EKF covariance values in the mapping node (2) Improving odometry by integrating additional sensors like GPS, which has already been setup for use (3) Enhancing the robustness of LiDAR projection. Future work should also explore a LiDAR-only backup navigation system to improve fault tolerance, which would involve training a LiDAR-based object detection model.

Week	Milestone	Description	Assignee
Week 3	Environment Setup	Set up the object detection model using the Roboflow dataset	Yuvanand
		SSH into Jetson and run ROS2 camera / LIDAR nodes Collect sample data from camera	
		SSH into Jetson and run LiDAR nodes Collect sample data from LiDAR	Sohyun, Mia
	Literature Review	Read at least 2 papers on sensor fusion Find example GitHub repos and provide analysis	Shirley, Bella
Week 4	Water Test	Set up the test environment and report any challenges	Mia, Bella
		Record a video footage of the water test	Yuvanand
		Collect sample data for the fusion model	Sohyun, Shirley
		Test previous year's code and report performance	Mia, Bella
	System Refinement	Reflect on previous water test and provide suggestions to refine the system	Shirley, Bella
	Prototype Sensor Fusion	Record any fundamental preprocessing or post-processing stages for the respective sensors Provide visualization of projected LiDAR points	Sohyun, Shirley
		Build a ROS2 sensor fusion node with collected data	
		Download open source dataset and demonstrate fusion algorithm	
Deliverable: Recorded video of a baseline test & sample dataset			
Week 5	Presentation Prep	Collect and format visuals & deliverables collected Draft slides & scripts for the Oral Project Update	Mia, Bella
	ML Refinement	Setup a Docker environment for GPU acceleration Retrain custom YOLOv8 model	Yuvanand
	Synchronization	Develop a pipeline for real-time data processing	Sohyun, Shirley
		Integrate the sensor fusion prototypes into online testing environment, evaluate frame-level performance	
Assignment Due (May 6): Oral Project Update			
Week 6	Milestone Report	Draft Milestone Report	All
	Water Test (Buffer Slot)	Set up the network with the router	Yuvanand
		Conduct water test using the code from the previous work Evaluate previous motor control methods	Yuvanand, Mia, Bella
		Report model performance and evaluate detection accuracy Collect data and analyze the reliability of fusion node	Shirley, Sohyun
Deliverable: MVP with visualization of Sensor Fusion results and performance analysis			
Assignment Due (May 14): Milestone Report			
Week 7	System Refinement & Optimization	Recalibrate the extrinsics for revised camera placement Merge all the nodes/features into a single package	Shirley, Sohyun
		Analyze failure cases and refine the Path Planning system to integrate fusion data	Mia, Bella
		Compare latency & performance between the blob and pt model	Yuvanand
	Localization & Mapping	Setup Foxglove visualization for GPS & Mapping Setup GPS driver and build ROS2 GPS node	Sohyun
	Add-ons	Implement counting algorithm for detected buoys	

Table 1. Milestones from Week 3 to Week 7.

Week	Milestone	Description	Assignee
Week 8	Water Test	Run full stack with live data recording Evaluate detection/navigation ability of the system	All
	Optimization	Setup a detection node using blob file	Yuvanand
		Conduct failure case analysis on the previous water test	Mia, Bella
		Integrate Triton AI's fusion system into our pipeline Develop different variants of the pipeline for testing	Shirley, Sohyun
	Localization & Mapping	Measure GPS Error in open-field & water test environment Setup & Test LiDAR-IMU Odometry using Fast-LIO	Sohyun
Week 9	Assignment Hand-ins	Provide Documentation (README) on GitHub Design Project Webpresence	Yuvanand, Sohyun
		Draft final presentation slides	All
	Mapping	Prototype ROS2 mapping node with visualization	Sohyun
	Water Test	Continue with any unfinished testing from previous week Record video clips for the final project video	Yuvanand, Sohyun, Mia
		Test the implementation in a larger water body that mimics the competition better	
Deliverable: Recorded video of the final test & documented GitHub repository			
Assignment Due (June 3): Project Webpresence			
Week 10	Assignment Hand-ins	Work on Final Project Video & Final Report	All
Assignment Due (June 9): Final Project Video			
Assignment Due (June 13): Final Report			

Table 2. Milestones from Week 8 to Week 10.

7 Acknowledgments

We would like to thank our sponsors Team Inspiration and TritonAI for their support. We would also like to thank Colin Szeto, Alex Szeto, Nida Firdaws, and Professor Jack Silberman for their continued help and support this quarter.

References

[1] Ji Zhang, S. S. Visual-lidar odometry and mapping: Low-drift, robust, and fast. *IEEE international conference on robotics and automation (ICRA)* (2015).

[2] Ji Zhang, S. S. Laser–visual–inertial odometry and mapping with high robustness and low drift. *Journal of field robotics* (2018).

[3] Ji Zhang, Michael Kaess, S. S. A real-time method for depth enhanced visual odometry. *Autonomous Robots* (2017).

[4] Jun Zhu, Hongyi Li, T. Z. Camera, lidar, and imu based multi-sensor fusion slam: A survey. *Tsinghua Science and Technology* (2023).

[5] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 779–788.

[6] Shi-Sheng Huang, Ze-Yu Ma, T.-J. M. H. F. S.-M. H. Lidar-monocular visual odometry using point and line features. *IEEE international conference on robotics and automation (ICRA)* (2020).

[7] Shibo Zhao, Hengrui Zhang, P. W. L. N. S. S. Super odometry: Imu-centric lidar-visual-inertial estimator for challenging environments. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2021).

[8] Weizhao Shao, Srinivasan Vijayarangan, C. L. G. K. Stereo visual inertial lidar simultaneous localization and mapping. *IEEE/RSJ international conference on intelligent robots and systems (IROS)* (2019).

[9] Young-Sik Shin, Yeong Sang Park, A. K. Dvl-slam: Sparse depth enhanced direct visual-lidar slam. *Autonomous Robots* (2020).

[10] Zheng, C., Xu, W., Zou, Z., Hua, T., Yuan, C., He, D., Zhou, B., Liu, Z., Lin, J., Zhu, F., et al. Fast-livo2: Fast, direct lidar-inertial-visual odometry. *IEEE Transactions on Robotics* (2024).