Black box Search and Rescue using Bluerov2

Underwater interventions

Pablo Daniel Candelas Pérez

Godsfavour Ugwu-Gabriel

Mukesh Sadhasivam

Universitat Jaume I

EMJMD MIR

January 16, 2026

Contents

1 Abstract

This project explores a hybrid approach to underwater robotics using a BlueROV that can operate in both teleoperated and autonomous modes. The robot navigates a pool using ArUco markers placed on the floor, locates a blackbox, and performs a simple but challenging manipulation task: attaching a carabiner to a small handle using a custom-designed gripper. The gripper was designed in Onshape, 3D-printed, and integrated with the robot for testing.

All behaviors were first developed and validated in the Stonefish simulator using ROS before being deployed on the real system. Teleoperation and supervision were handled through a user interface built with Unity and Foxglove. Rather than aiming for full autonomy, the project focuses on understanding how control can be shared between a human operator and the robot, showing how autonomous behaviors can support and simplify underwater manipulation tasks in a realistic setting.

2 Introduction

The idea behind this project is simple: take a BlueROV, and try to get it to find a blackbox and attach a carabiner to its handle. In reality, it's a lot messier. Underwater visibility is limited, currents can interfere with the robot's motion, and the manipulation task is much harder than it sounds.

To make things manageable, we use two operating modes:

- **Teleoperated**, where a human controls the robot directly
- **Autonomous**, where the robot navigates and aligns itself with the target

A custom gripper was designed in Onshape and 3D-printed specifically for this task. Everything was first tested in simulation with Stonefish to avoid unnecessary risks, and a user-friendly interface was built in Unity and Foxglove to make teleoperation smooth and intuitive.

2.1 Motivation

The idea behind this project is to understand how much of an underwater manipulation task can be automated, and when it still makes sense for a human to step in. Full autonomy underwater is difficult and expensive, but full teleoperation is tiring and error-prone.

The task — finding a blackbox and attaching a carabiner — is a perfect example. It sounds simple but becomes complicated underwater, making it an ideal challenge for testing shared control. Instead of chasing full autonomy, the project focuses on letting the robot take care of predictable steps while the operator handles the uncertain ones. Retrieving a blackbox underwater is not a trivial task, if correctly done in the future could aid understanding plane crashes, their causes and how to make flying safer.

2.2 State of the Art

Most underwater robots today are still piloted by humans. Operators watch a live camera feed and manually steer the robot, especially during precise tasks like grasping or attaching tools. Autonomous underwater manipulation remains difficult due to limited sensing, poor visibility, and the lack of GPS.

ArUco markers have become a common solution in controlled environments such as pools, because they are cheap, simple to detect, and reliable for visual localization.

This project takes these existing ideas and pushes them further by adding autonomous behaviors for specific phases, without removing the human from the loop. The approach sits between fully manual and fully autonomous operation.

2.3 Problem definition

The main problem is getting the robot to locate an object underwater and interact with it reliably. The BlueROV must:

1. Detect the ArUco markers on the pool floor
2. Use them for localization
3. Navigate autonomously toward the blackbox
4. Align with the handle
5. Attach the carabiner by pushing it into place

Doing everything manually is slow and error-prone, but making the entire pipeline autonomous would be unreliable with the available sensing. The challenge is deciding which parts the robot should handle automatically and which parts should stay under human control, while keeping the transition smooth.

2.4 Materials, workspace and software

The project was developed using a single BlueROV2 heavy vonfiguration equipped with a custom gripper and an onboard camera. The gripper was designed from scratch in Onshape and then 3D-printed, so it could be adapted specifically to pushing and attaching the carabiner onto the handle. The tests were carried out in a pool environment, with nine ArUco markers placed on the floor to help the robot understand where it is and where it needs to go.

Everything was built around ROS for control, perception, and communication between the different modules. Before touching the real robot, all the code was first tested in the Stonefish simulator to make sure the behaviors made sense and to avoid breaking

anything underwater. Unity and Foxglove were used to design the user interface, making teleoperation and monitoring more intuitive and easier to handle during experiments.

3 Methodology

3.1 Detection

One of the most important parts of the system is the ability to detect and identify the black box and its handle. To achieve this, a YOLOv8n model was trained. The motivation for using this model is based on the characteristics of the detection problem, such as:

- Simple shapes
- Color variation
- Underwater tuning required
- Only one box and one handle per frame
- Low computational power (GPU)
- Small dataset

To create the necessary dataset, different recordings were collected. The recordings were obtained with the black box positioned differently, at various times of day, and from different distances. Afterwards, the recordings were post-processed to extract a single frame every 3 seconds in order to obtain a reduced dataset that captured only significant variations in the videos. Finally, the images were annotated to complete the dataset and proceed with the training.

The training process was straightforward: 100 epochs were run with data augmentation to simulate different underwater conditions. The detection results were then validated, and an acceptable confusion matrix was obtained.

3.2 Gripper Design

The BlueROV2 was equipped with a Newton gripper attachment. Nevertheless, a new design for the gripper components was required in order to firmly grasp the carabiner used in the mission. For this reason, the team developed several design iterations to reach the final configuration.

The first design, shown in Figure 1, presented several issues: the carabiner tilted slightly forward, and the angle of contact was not optimal for the task.

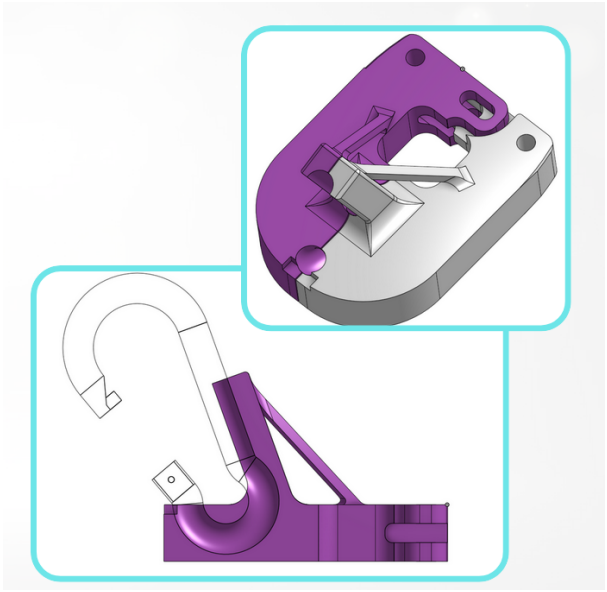

Figure 1: First design

Subsequently, a different set of gripping concepts was tested, exploring approaches that provided better visibility, reduced 3D printed material usage, and varied orientations. A collage of the tested concepts can be seen as follows.

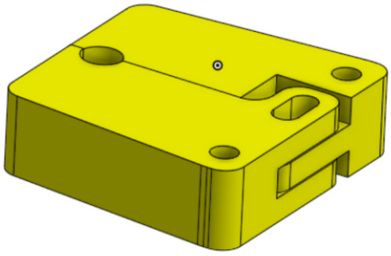

Figure 2: Gripper Model 2

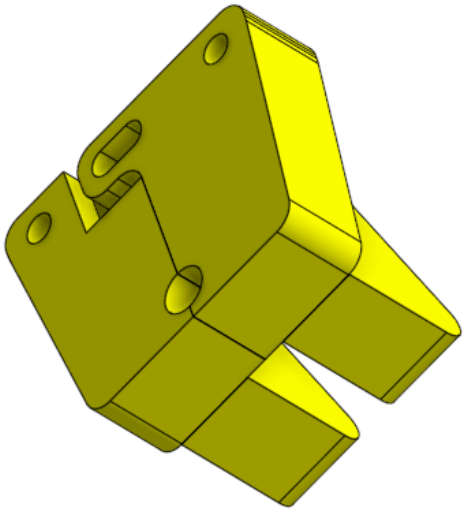

Figure 3: Gripper Model 3

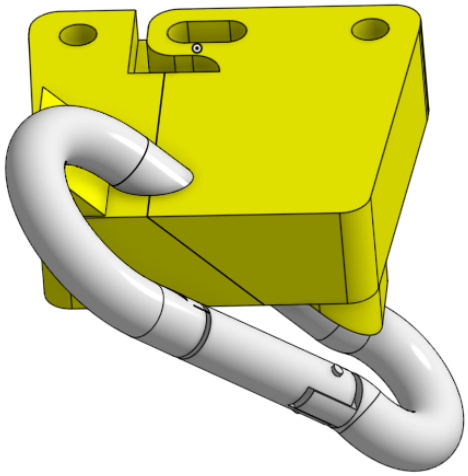

Figure 4: Gripper Model 4

However, the designs shown in previous figures still resulted in undesired rotations, as not all degrees of freedom (DoF) were properly constrained. Additionally, the original carabiner was replaced because the force required to open it was excessively high. Therefore, the team searched for more suitable alternatives (Figure 5) and ultimately selected the carabiner shown in Figure 6.

Figure 5: Carabiner Options

Figure 6: Carabiner Used

With the final carabiner selected and the lessons learned from previous concepts, two final designs were manufactured. Unfortunately, the design shown in Figure 7 broke after several uses. Fortunately, this was not the only gripper produced, and the design shown

in Figure 8 functioned properly.

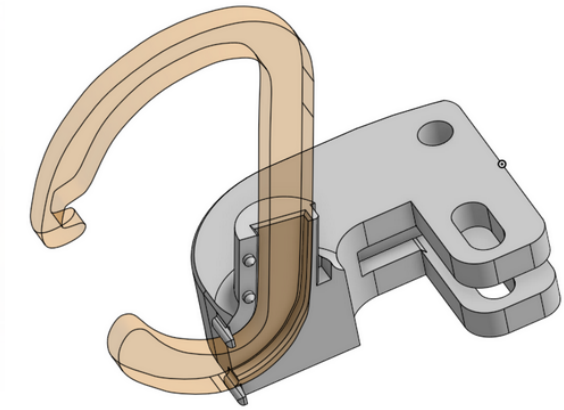

Figure 7: Gripper model 5(broke)

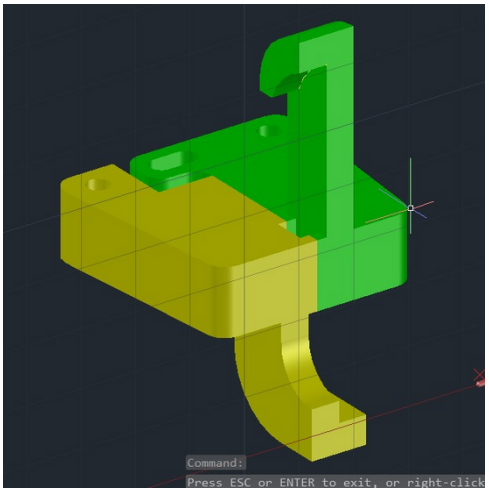

Figure 8: Gripper Model 6 (good)

3.3 Localization

To solve the localization problem of the BlueROV2 inside the water tank, a combination of sensing methods was used. The water tank floor contained a set of nine ArUco markers that were evenly spaced, with known ground-truth positions. Using the OpenCV library and the onboard camera, the position of the BlueROV2 was estimated relative to these markers.

To improve depth accuracy, data from the depth sensor was incorporated. Finally, a moving average filter was applied to the sensor outputs to obtain a smoother localization estimation and reduce jitter caused by noisy readings.

The complete algorithm operates as a sequence of coordinate transformations: from a reference corner of the pool (0, 0, 0) to each ArUco's known position, then to the camera frame, and finally to the BlueROV2 body frame. These transformations were managed as a specific TF tree, allowing later visualization in RVIZ2, Foxglove, and Unity.

3.4 Autonomous Recovery Approach

To implement autonomous recovery, the entire mission workflow was first mapped, as shown in the following figures:

The first step "Deployment and initialization" flowchart is the following:

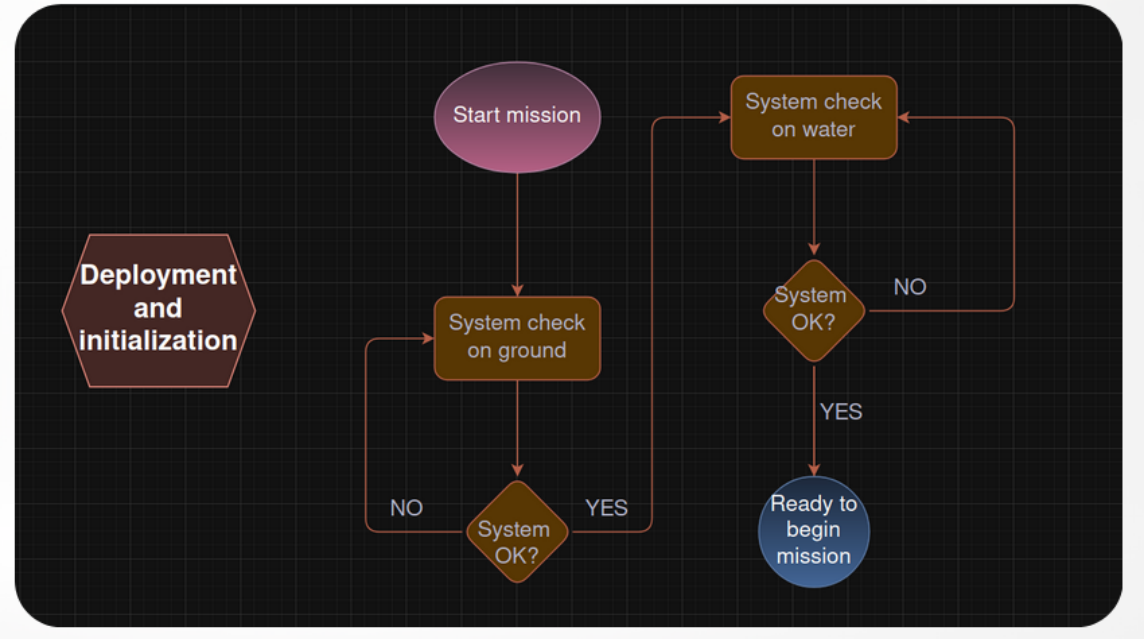

Figure 9: Deployment and Initialization flowchart

The second step "search" flowchart is the following:

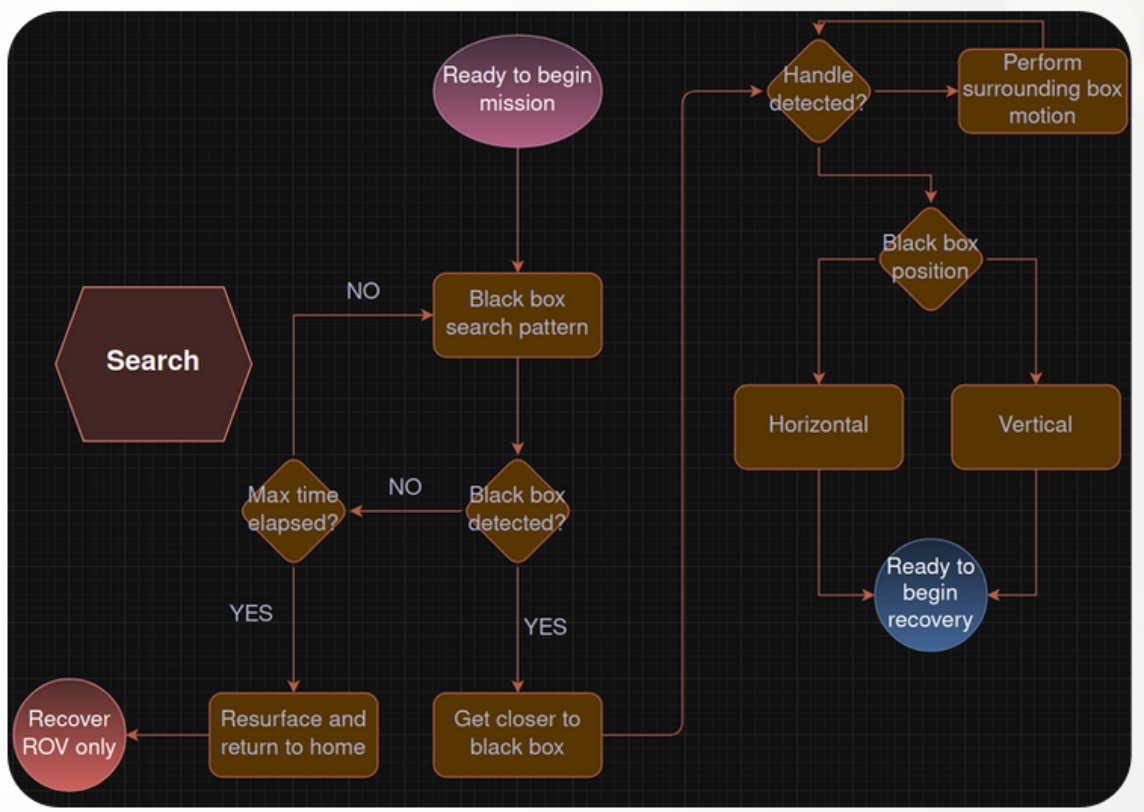

Figure 10: Search step flowcahrt

The third step "Rescue" flowchart is the following:

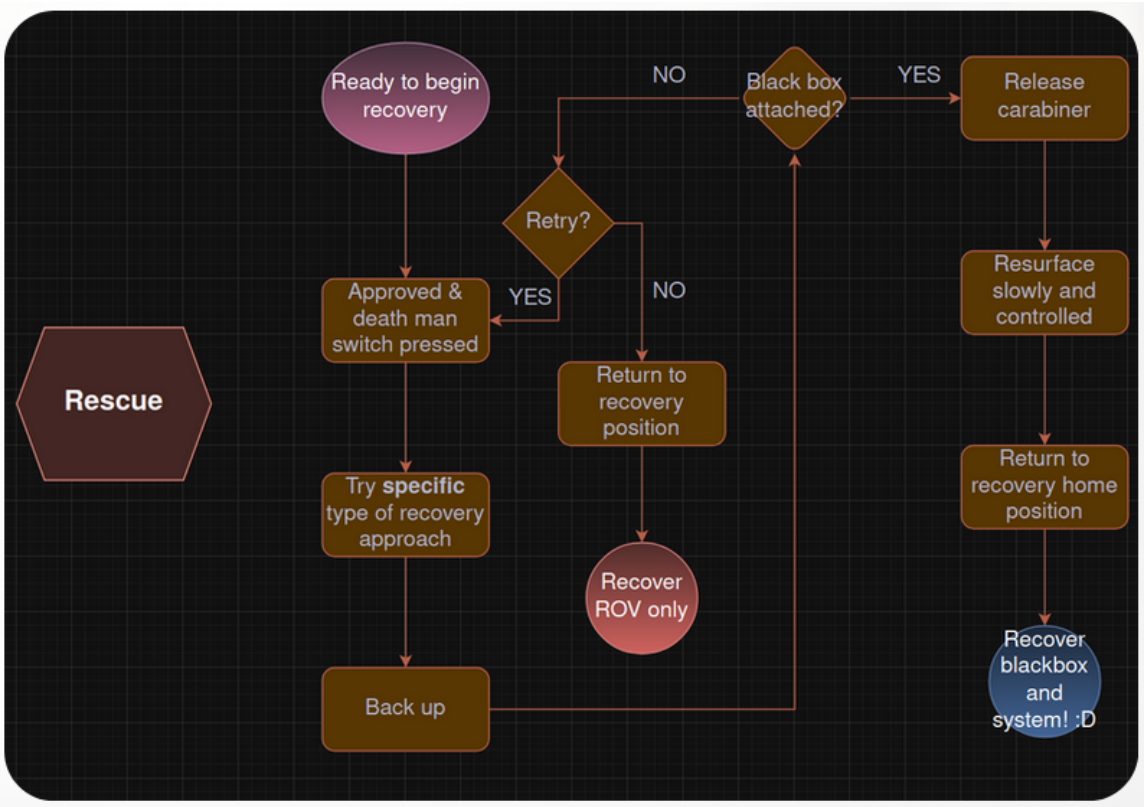

Figure 11: Rescue step general flowchart

After establishing the mission workflow, the team performed a manual execution of the mission to understand the challenges involved and to identify the sequence of actions the robot would need to perform. Based on these observations, the detailed flowchart of the recovery approach was created, as shown in Figure 12.

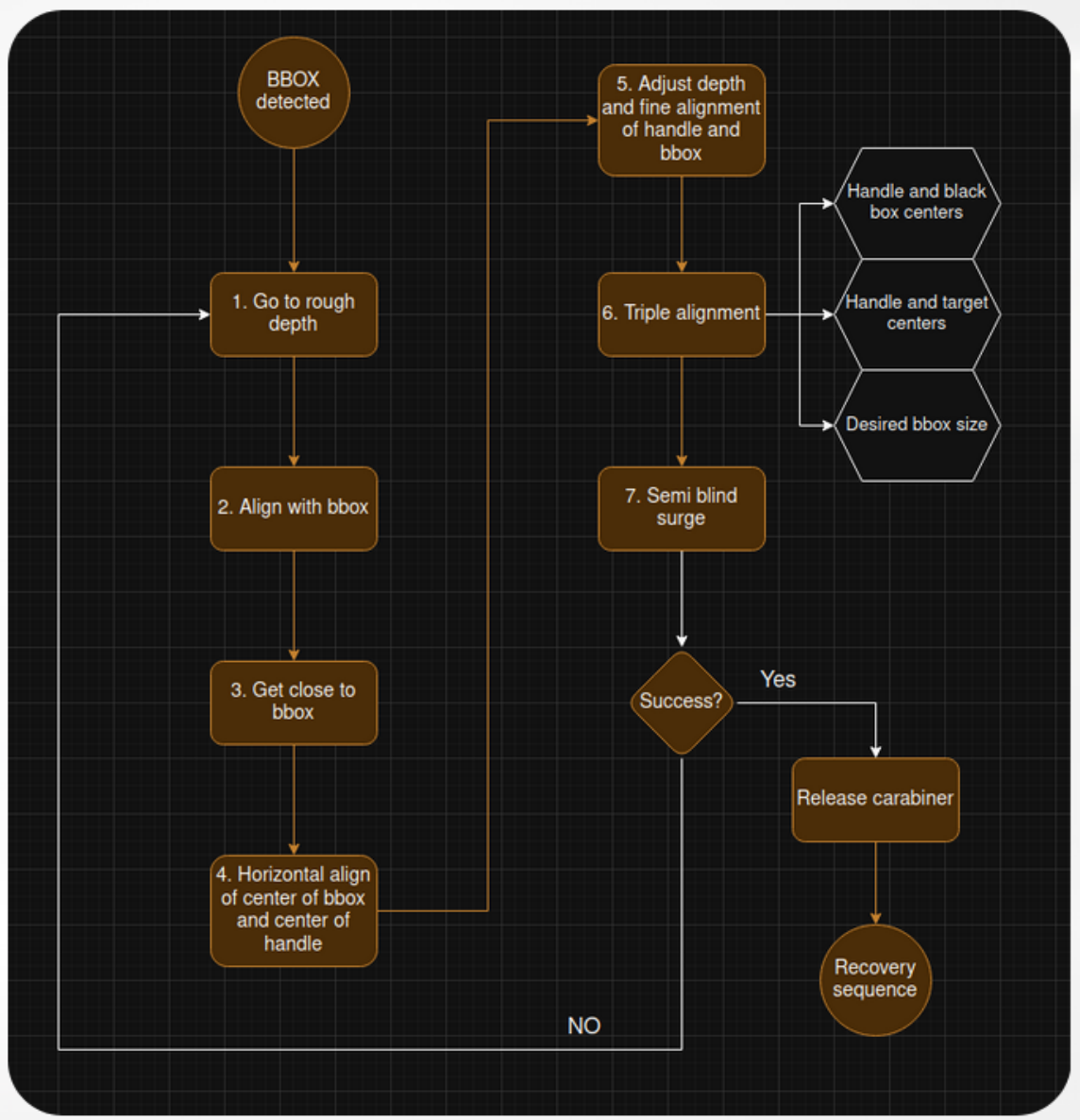

Figure 12: Rescue step general flowchart

With this information, all the ROS 2 nodes were implemented, the required operation modes (such as `depth_alt`) were added, and the PID controllers were tuned. A dedicated PID was configured for each type of motion. This enabled the robot to execute the mission as a sequence of structured steps, with specific movements and condition checks defining the transition from one step to the next.

It is important to note that the team decided to handle only one specific orientation of the black box: the configuration in which the black box lies horizontally while the handle is oriented vertically.

3.5 Thinking Out Loud — Robot Explanation

As the flowcharts from the previous subsection were already defined, only a mission follower module was required for this part. A dedicated ROS 2 node was developed to iterate through the mission steps based on the current state and sensor readings. This node also interacts with the user to verify that certain steps have been correctly completed (for example, robot initialization, correct detection of the object, and *deadman switch* validation). Finally, the node logs all relevant mission information into a text-based mission report.

3.6 Simulation

Figure 13: Stonefish environment

Before sending the BlueROV anywhere near the pool, we reproduced the whole environment inside Stonefish: the pool walls, the water properties, the lighting, and even the ArUco markers on the floor. The idea was simple — if it behaves properly in simulation, there's a good chance it won't crash into something in real life.

We started by importing the BlueROV model, adding the thruster configuration, and matching the mass and buoyancy as closely as we could to the real robot. The buoyancy tuning took more time than expected because the simulated robot was either floating away like a balloon or sinking like a brick until we found the right parameters. Once we got it hovering properly, the rest of the physics started to make sense.

Next, we tested the control side. Instead of manually piloting the robot immediately, we used the simulator to check PID tuning, attitude control, and basic maneuvering. This was super useful because we could dial in the gains without risking the robot suddenly flipping over in the pool.

After that, we integrated the vision pipeline. This gave us a closed-loop system: simulated robot → virtual camera → back to the controller.

We tested the workflow: approach, detect, and, align. Even though the simulator can't fully replicate underwater physics of contact and friction, it helped us test the timing, trajectories, and approach angles.

This helped us practice the whole mission before trying it on the real robot.

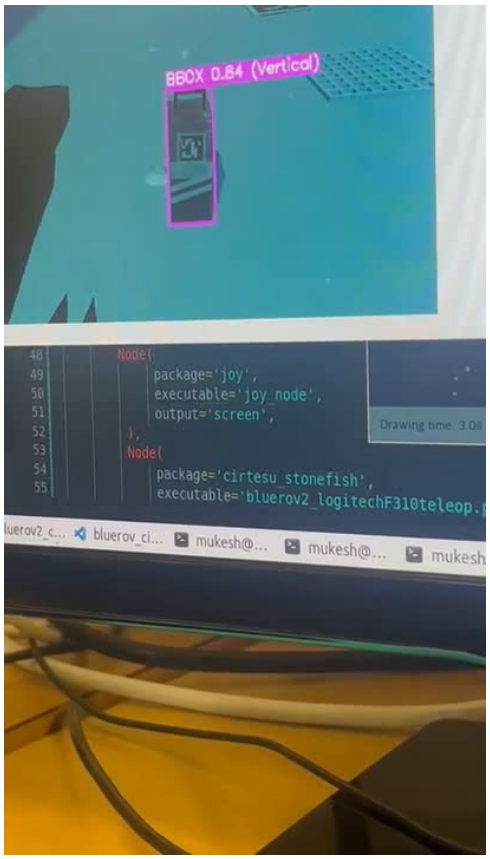

Figure 14: Blackbox and handle detection in Stonefish

3.7 GUI - Unity

Unity was used to develop the GUI for the system. In the initial stage, the GUI was built and tested using ROS bag files. These ROS bags allowed us to replay recorded data from the robot, which made it possible to visualise the robot's motion without needing the physical robot at that point. Through this setup, the GUI displayed the robot's translations and rotations in real time, and the simulated robot inside Unity moved exactly as the real robot had moved during the recording.

As the GUI developed further, additional features were added to improve usability.

Instead of relying on terminal commands, buttons were implemented within the GUI to control different functions. This made the system much more user-friendly and intuitive, especially for users who were not comfortable working directly with the terminal.

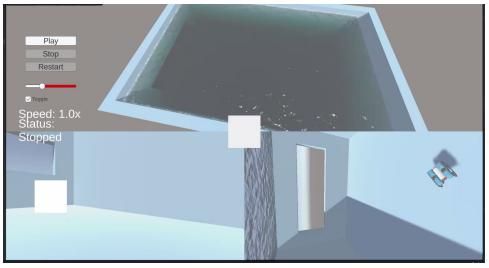

Figure 15: Unity GUI 1

After successfully testing the GUI with ROS bags, the system was then extended to work with the real robot. In this stage, live data from the robot was streamed into Unity, allowing the simulated robot to translate and rotate in response to the real robot's movements. The IMU data fed the robot's orientation into Unity, and the digital twin rotated exactly as the real robot did in the pool. The next step was handling the position tracking using **ArUco markers** on the floor of the pool. Unity received the estimated position from the vision pipeline and updated the robot's position accordingly. This worked well and demonstrated that the GUI could be used for real-time monitoring. However, some inconsistencies were observed in the motion of the simulated robot. These issues were mainly due to differences in axis definitions and alignment between the underwater robot's coordinate system and Unity's coordinate system. Because of these discrepancies, the simulated robot's orientation did not always perfectly match the real robot, highlighting the need for further calibration and alignment corrections.

3.8 GUI — Foxglove

To provide a dashboard for critical mission data and enable mission control, Foxglove was used. With this interface, the user can visualize the video stream, battery levels, depth readings, light status, and incoming messages from the robot. Additionally, Foxglove allows the user to validate specific actions occurring during the mission, including monitoring propulsion power through a dedicated gauge.

4 Results and Discussion

4.1 Detection Accuracy

Once the trained model was tested underwater, its performance aligned with expectations derived from the confusion matrix. The robot was able to detect the black box reliably from an approximate distance of 5 meters, with high confidence values (typically above 60% and, in most cases, above 85%). Handle detection proved more challenging, and the robot occasionally struggled with it at longer distances. Nevertheless, when the handle was within close range, detection accuracy was also high (above 80%). Due to the handle's black color, the robot's lights needed to be activated to ensure correct detection. Figure 16 shows a camera view from the BlueROV with the corresponding detections.

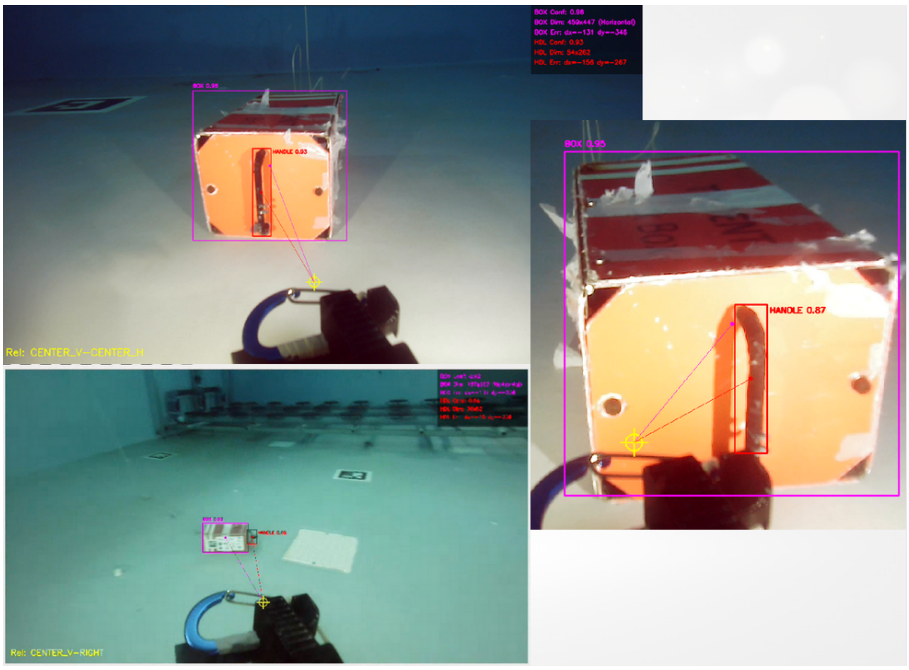

Figure 16: Yolo underwater detections

Some occasional misdetections were observed. Rarely, another BlueROV was detected as a black box. To prevent issues, only the single detection with the highest confidence score was used for decision-making. Furthermore, to minimize potential misdetections involving other robots, the team ensured that no other BlueROV was within the field of view during the autonomous mission.

4.2 Simulation

Using Stonefish proved extremely valuable, especially during the early development stages. The biggest advantage was being able to test high-risk behaviors without any real-world consequences. In simulation, the robot could collide with walls, misalign its gripper, or approach the blackbox incorrectly, and the worst outcome was simply resetting the scene.

The first major success was achieving realistic motion. Once the buoyancy and thruster parameters were tuned correctly, the simulated BlueROV behaved very similarly to how it moves in the actual pool. Its inertia, turning response, and stability felt convincing, which gave us confidence that the control algorithms would translate well to real-life conditions.

ArUco and object (Blackbox and handle) detection in the simulator also worked very well. Because the virtual environment had perfect lighting and no visual noise, the markers were detected cleanly and consistently. This provided an ideal reference to compare against real-world performance later. In the real pool, reflections, turbidity, and camera angles introduced noise, so having a clean baseline from simulation helped us understand where the errors were coming from.

However, several limitations became obvious once we switched from simulation to physical testing. Stonefish does not fully replicate underwater contact or the small mechanical quirks of real objects.

Stonefish uses a perfectly consistent coordinate frame, while the real underwater system introduces small rotational and positional offsets. As a result, the position in Unity (fed by real ArUco detections) tended to jump or drift slightly, whereas the simulated version was perfectly stable.

Despite these differences, the simulation provided three major benefits:

- **Safe debugging:** We could test failures and edge cases without risking the robot.
- **A controlled space for autonomy:** The robot could make mistakes harmlessly.
- **A clean baseline for comparison:** Ideal conditions in simulation highlighted real-world noise clearly.

Stonefish did not replace real-world testing, but it made the entire development process smoother, safer, and significantly more efficient. It allowed us to validate core ideas, tune controllers, and understand the workflow long before getting the robot wet.

4.3 Autonomous Control

The BlueROV was able to correctly perform the autonomous sequence of motions. It successfully detected the black box from a distance, descended to the appropriate depth, aligned with the box, and approached it. Afterwards, it detected the handle, centered on it, and fine-tuned its alignment in terms of depth, size, and position relative to the gripper. Finally, it engaged and attached the carabiner using a fast surge motion.

Even though the autonomous control successfully attached the carabiner, further fine tuning is required, as several improvement areas were identified:

- The depth sensor was used to reach the engagement depth. Since the depth sensor relies on pressure measurements, this approach is only suitable in controlled environments. Even in the pool, the depth had to be calibrated frequently, as small variations affected mission success.
- The yaw PID gains could be improved to achieve faster and more stable convergence to the desired orientation.
- The full approach could take up to 3 minutes depending on conditions. Ideally, this process should be completed in less than 1 minute.
- The algorithm should support multiple black box and handle orientations. Although feasible, small adjustments are needed to make the solution general and robust.

4.4 User Interfaces

4.4.1 Unity

The Unity GUI showed the robot's movements, and the digital twin followed the real robot in real time. The initial rosbag-based version was surprisingly helpful. It let us iron out bugs without needing to power up the robot or fill the pool. During development, we could fast-forward through data, rewind, pause, and test all the interface logic over and over. This made the GUI much more stable by the time we hooked it up to the real system.

The next big milestone was the live connection. The IMU integration worked pretty nicely. The rotation of the robot in Unity rotated the real robot almost immediately.

Seeing the virtual robot roll, pitch, and yaw as the real one moved in the pool was a huge step forward.

The position tracking using the ArUco markers was good but not perfect. Sometimes the robot's position would "jump" slightly in Unity even though the real robot hadn't moved that much. After some digging, we figured out that the main issue came from the difference in coordinate frames. The underwater camera has its own axes, Unity has its own axes, and the ArUco marker detection has yet another convention for orientation. When the transformations don't line up exactly, small misalignments get amplified and show up as odd behavior in the Unity visualization.

Another problem was that underwater lighting and camera angle affect how consistently the markers are detected. Depending on where the robot was in the pool, visibility changed a lot. As a result, Unity sometimes received slightly noisy pose estimates.

The GUI worked. The live model was never meant to be millimeter-accurate — just accurate enough for the operator to know where the robot was and what it was doing. For that purpose, the platform worked.

Building the GUI in layers (r9osbags → simulated robot → real robot → ArUco integration) made the whole process much smoother. Each step acted as a safety buffer before moving to the next one. And even though there were inconsistencies with the coordinate alignment, those issues can be solved with calibration, axis remapping, and better synchronization between Unity, ROS, and the camera system.

4.4.2 Foxglove

In the real practice sessions and the final demonstration, only the Foxglove UI was used. It worked correctly and displayed the expected information and values, as shown in Figure 17. Since several processes were running simultaneously and the project is still in a preliminary phase, an additional debugging video feed directly from a ROS 2 node was used. This made the image processing pipeline too heavy for the computer to handle. Therefore, during the demonstration, the video broadcast within the Foxglove UI was disabled, and only the debugging node was used for visual feedback.

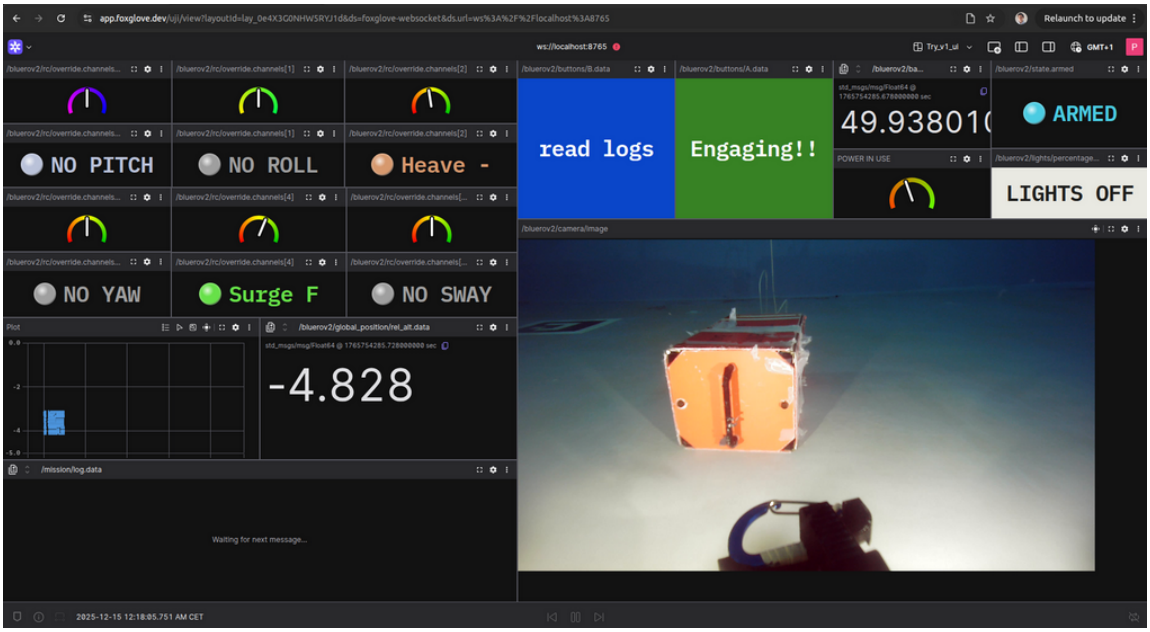

Figure 17: Foxglove Interface

The Foxglove UI provided several advantages, giving the operator a clear understanding of the robot's behavior (e.g., motion mode indicators and power gauges). Displaying the battery level, depth, and light status increased operator confidence, since the operator could verify that the robot had sufficient battery, adjust lights as needed, and avoid having to visually inspect the pool to determine the ROV's position or whether the lights were on. Overall, having a UI significantly improved mission flow and increased trust in the system.

4.5 Localization

The localization system worked well on its own; however, the processor of the workstation was overwhelmed by the number of components that needed to run simultaneously. Due to this limitation, localization could not be executed at the same time as the other nodes. In practice, localization remained in a standalone configuration. When all components were executed together, including the localization node, the video stream became laggy and noisy, and detection quality decreased dramatically as a result.

The localization produced satisfactory results: as long as the BlueROV was facing an ArUco marker, the robot could determine its position within the pool. Unity was used for visualization (please refer to the UI unity section to visualize it), and a simple TF tree display in RVIZ2 can be seen in Figure 18.

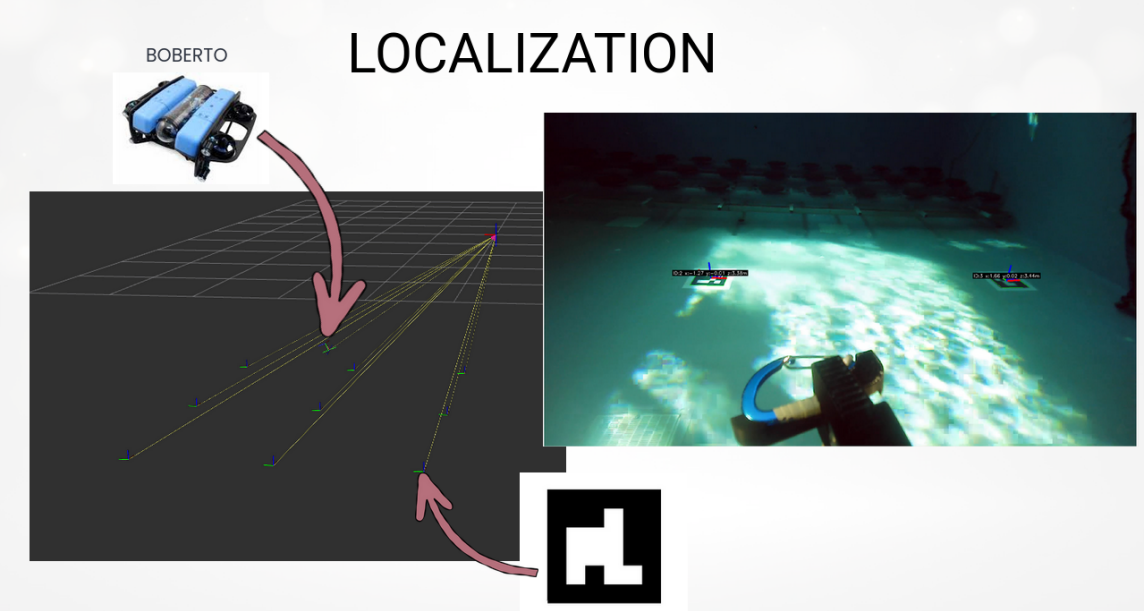

Figure 18: Localization result

While localization provided useful information regarding the global position of the robot, it relied entirely on visual detection of the markers. Therefore, when no markers were within the field of view, the robot lost positional updates and retained only the last known estimate. No reliable method was available to quantitatively measure the localization error. Ultimately, the system was useful for the operator to understand the BlueROV's global position, but for autonomous navigation it was only suitable for coarse sector-level segmentation of the environment. Fine-grained localization with sufficient accuracy for high-risk maneuvers, such as approaching the black box, was not achieved.

4.6 Overall mission

The mission was completed successfully, although the full autonomous cycle was achieved for only one orientation of the black box. The autonomous approach occasionally required a few retries, which is normal given the constantly changing conditions. The important aspect is that the system supports persistence, which is crucial in robotics. The initial deployment of the BlueROV and the final retrieval remain manually operated. During retrieval, the operator must carefully maneuver the BlueROV to the edge of the water tank. Due to the added payload, the vehicle becomes slightly unbalanced, and the most sensitive part of the procedure occurs near the walls as it approaches the surface at the end of the mission.

5 Conclusion and Future work

5.1 Conclusion

This work presented the development and integration of an autonomous underwater recovery system using a BlueROV2 platform. The project successfully demonstrated a full autonomous recovery cycle under controlled conditions for a specific black box and handle orientation. Key contributions included the implementation of underwater detection using a YOLO-based model, a custom gripper design for robust carabiner attachment, a ROS 2-based mission execution framework, a simulation enviroment in stonefish and an operator interface through Foxglove and Unity for mission visualization and monitoring.

The detection module proved capable of identifying both the black box and its handle at meaningful distances, achieving high confidence levels during close-range operation. The gripper designs evolved through iterative prototyping until a reliable configuration was achieved. The autonomous mission logic enabled the robot to navigate, align, and perform the carabiner attachment with a high level of consistency. Although localization worked as a standalone capability, computational constraints limited its integration with the full mission pipeline.

Overall, the system achieved its primary objective: autonomously attaching a carabiner to a target underwater object. The results indicate that with further refinement and generalization, such a system has potential for broader application in underwater inspection, recovery, and intervention scenarios.

5.2 Future Work

While the results are promising, several opportunities for improvement and extension have been identified. Future work may consider the following directions:

- **Computational Optimization:** Improve onboard and topside processing pipelines to enable simultaneous execution of localization, perception, and mission control without performance degradation.
- **Generalized Detection:** Extend visual detection to support multiple black box and handle orientations, varying object types, and dynamic underwater environments.

- **Robust Localization:** Incorporate marker-less localization approaches such as SLAM, inertial fusion, or acoustic positioning to avoid reliance on visual fiducials.
- **Improved Control Strategies:** Enhance control laws and PID tuning to reduce approach time and improve stability during fine manipulation tasks.
- **Real-World Deployment:** Validate the system in natural underwater environments with turbidity, currents, and reduced visibility to assess operational robustness.
- **Full Mission Automation:** Automate deployment and retrieval procedures to eliminate the remaining manual steps and achieve a fully autonomous mission pipeline.
- **Mechanical Improvements:** Further refine the gripper design for increased durability and payload stability, especially when surfacing near physical boundaries.

These developments would increase the adaptability, robustness, and autonomy of the system, ultimately enabling more complex and reliable underwater recovery missions.