Practical Work - Control of Underwater Robots

Sahil Abdullayev, Samantha Connolly, Alexander Endresen, Hyejoo Kwon

Course: Marine Mechatronics

Teacher: Vincent Hugel, Vincent Creuze

Submitted on March 11, 2025

MIR MARINE & MARITIME INTELLIGENT ROBOTICS

UNIVERSITÉ DE TOULON

1. Introduction

Through this practical, our team will build a simple visual servoing system for a BlueROV by implementing proportional controllers. We aim to control the position of the robot, calculating the error of desired point and current point. Our code finds the buoy in the image, computes its center, and then drives the robot so that the buoy stays in the middle of the view and at a fixed size. These practicals will require both knowledge in the systems required to operate the BlueROV along with the physical capabilities and dynamics.

2. Practical Work

2.1. System Setup

- Camera Input: We subscribe to the ROS2 camera topic and convert each frame to an OpenCV image.
- HSV Thresholding: We turn the image into HSV color space and apply a color mask. This mask recognizes orange pixels. We tune the mask with trackbars or a mouse click.

2.2. Feature Detection and Tracking

We find the buoy in each camera frame by looking for its color. When we find it, we calculate the buoy's center position and its size. We use these values to know how far the buoy is from the image center.

2.3. Proportional Control

We created a basic controller to move the robot smoothly towards the buoy. The controller uses three simple errors: horizontal position, vertical position, and the size of the buoy. Based on these errors, we tell the robot how to move forward, turn, or tilt.

Error definitions:

- err_x = cx - (image_width/2) → positive if buoy is to the right.
- err_y = cy - (image_height/2) → positive if buoy is below center.
- err_r = desired_radius_px - radius_px → positive if buoy is smaller (farther away).

Control gains: We chose kp_x, kp_y, kp_z by simple testing so that motions are smooth and not too fast.

- angular.z = -kp_x * err_x moves the ROV's yaw to center horizontally.
- angular.y = -kp_y * err_y tilts the ROV's pitch up or down.
- linear.x = kp_z * err_r moves the ROV forward or backward to adjust distance.

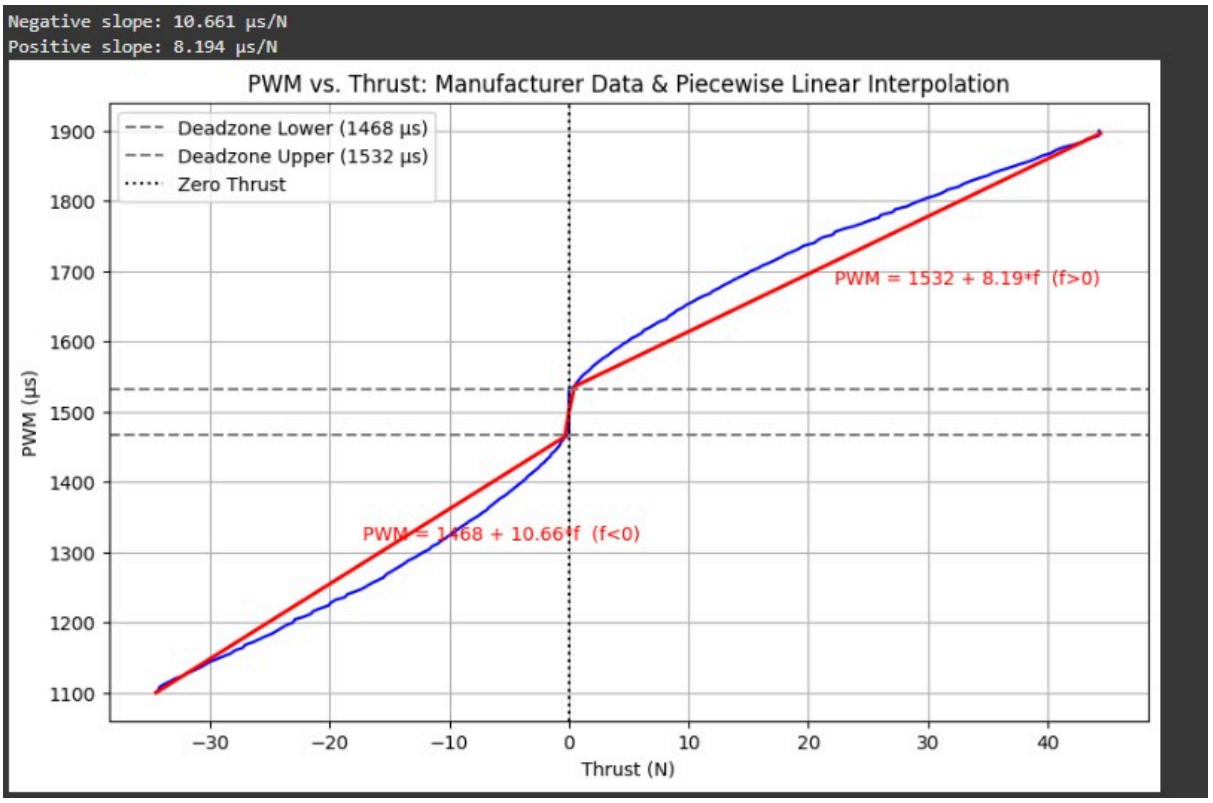
Publish: We send this Twist on /cmd_vel each frame.

2.4.

-

3. Practical Work

3.1. Computing a thruster's control input depending on the desired thrust



3.2. Experimental validation of the thruster's model

To validate the model, we estimated the ROV's floatability by attaching a 1 L water bottle and measuring the force needed to keep the robot neutrally buoyant. The corresponding PWM values were computed and tested on the actual thrusters.

3.3. Implementing a proportional controller for the depth

To control the depth of the Blue ROV 2, we implemented a Proportional Controller (P) which adjusts the thrust force based on the error between the desired depth and the actual measured depth. The control law is given by:

$$f_z = K_p(z_{des} - z)$$

where f_z is the force applied by the thrusters, K_p is the proportional gain, z_{des} is the target depth, and z is the current depth measurement. The idea is simple:

- If the ROV is too deep, it applies upward thrust.

- If the ROV is too high, it applies downward thrust.

The P controller was implemented by reading the depth sensor data, calculating the depth error, and applying a proportional gain K_p to determine the necessary thrust force. This force was then converted into a PWM signal and sent to the thrusters to adjust the ROV's position.

During testing, we carefully adjusted the K_p value. A small K_p made the ROV slow to react, while a large K_p caused oscillations. After gradual tuning, we found a good balance between stability and response time.

Overall, the P controller provided a simple but effective way to control depth. But, improvements were still needed for better precision and stability.

Kp values tested: 12.5, 10, 15 —- 12.5 best

3.4. Implementing a depth controller with floatability compensation

After implementing the Proportional Controller (P) for depth control, we noticed that the ROV was not able to maintain the desired depth accurately. The main reason for that was its natural buoyancy, which caused it to float upwards even when the controller was active. Since the proportional controller only reacts to the error between the desired and actual depth, it could not fully compensate for this constant force. As a result, a steady-state error remained, meaning the ROV was always slightly off from the target depth.

To address this issue, we modified the depth controller by adding a floatability compensation term. This additional term accounts for the constant buoyancy force and ensures that the ROV receives enough thrust to remain at the desired depth without continuously drifting upwards. The updated formula became:

$$\tau_z = K_p (z_{des} - z) + f_{buoyancy}$$

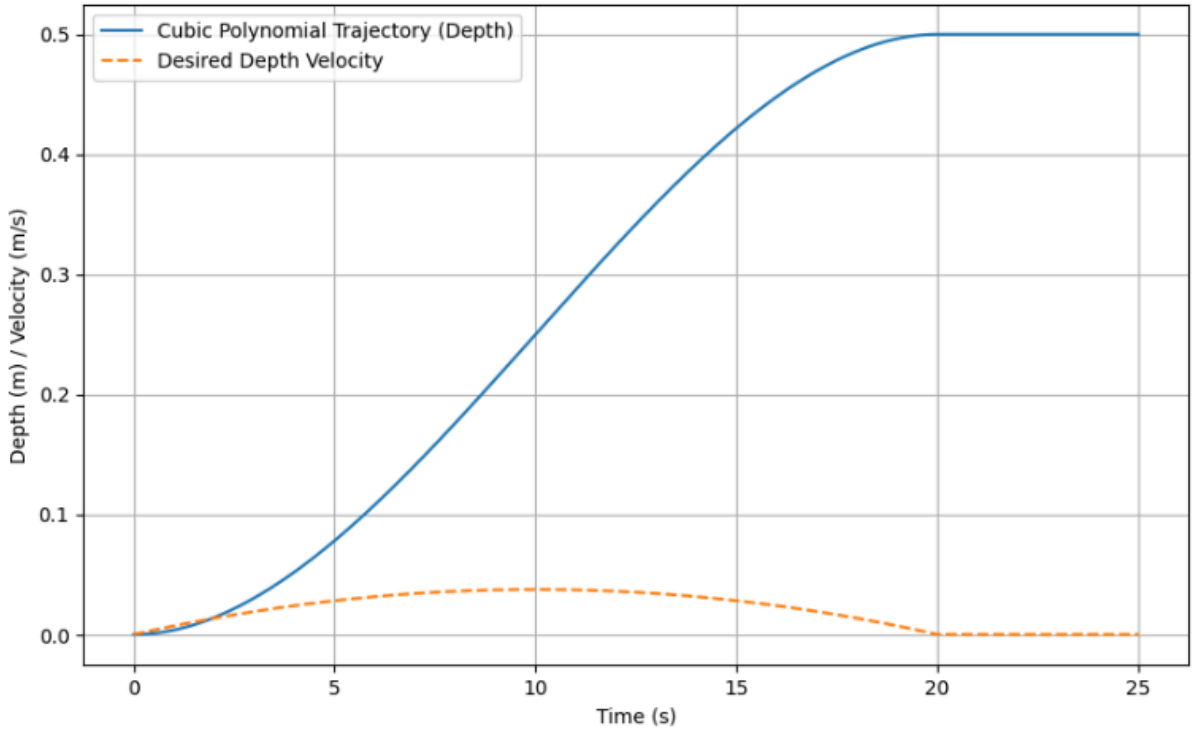
Where K_p is the proportional gain, z_{des} is the desired depth, z is the current depth measurement, and $f_{buoyancy}$ is a constant force added to counteract the natural buoyancy of the ROV.

With the floatability compensation in place, we saw a significant reduction in steady-state error. The ROV was now able to maintain its depth much more accurately, without continuously floating upwards due to buoyancy. This adjustment also allowed us to use a smaller K_p value, which made the control system more stable and reduced the chances of oscillations.

Adding floatability compensation was an important step in refining our depth control strategy, making the system more robust and reliable for underwater operation.

Kp values tested: 8, 6

3.5. Creating a trajectory compatible with the ROV's dynamics



3.6. Implementing the polynomial trajectory

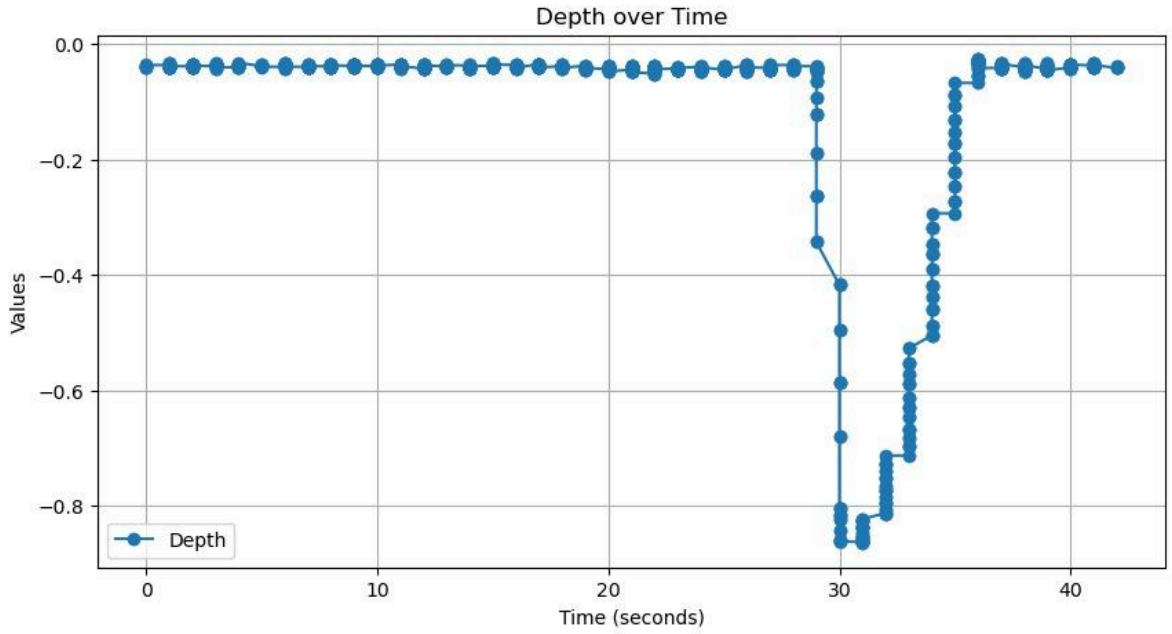
To improve the smoothness of the ROV's depth control, we implemented a polynomial trajectory instead of using a sudden step input for the desired depth. A step input causes abrupt changes in the setpoint, which can lead to overshooting and oscillations due to the physical limitations of the ROV. By using a cubic polynomial, we ensured a gradual transition between depth levels, making the control system more stable and predictable.

The equation used was:

$$z_{desired(t)} = z_{init} + a_2 * t^2 + a_3 * t^3$$

where the coefficients a_2 and a_3 were calculated to ensure smooth movement from the initial depth (z_{init}) to the desired depth ($z_{desired(t)}$) within a set time.

We first determined the starting depth from the ROV's sensor and set the target depth. We then calculated the trajectory using the polynomial equation and updated the desired depth in real time. The velocity reference was also computed as the derivative of the polynomial function to provide a smooth speed transition.



During testing, the polynomial trajectory significantly reduced oscillations and improved overall depth control. The ROV followed the desired depth more smoothly, with less overshoot compared to a direct step input. This approach also helped in reducing the workload of the PID controller, as the system no longer had to react to sudden depth changes.

Overall, implementing a polynomial trajectory made the depth control more natural, stable, and efficient, ensuring the ROV transitioned smoothly between depth levels while maintaining precise control.

3.7. Proportional Integral controller (PI) for the depth

While adding floatability compensation to the proportional controller improved depth control, we still observed a small steady-state error. This was because the proportional controller alone only reacts to the current error but does not account for accumulated errors over time. As a result, even with floatability compensation, the ROV still struggled to perfectly reach and maintain to desired depth under varying conditions.

To eliminate this error, we introduced an Integral term, leading to a possible implementation of a Proportional-Integral Controller (PI). The PI controller not only responds to the current error but also accumulates past errors to ensure that the ROV reaches and stays at the exact target depth. The control law is given by:

$$\tau_z = K_p(z_{des} - z) + K_i \int (z_{des} - z)dt + f_{buoyancy}$$

Where $K_p(z_{des} - z)$ is the proportional term that reacts to the current depth error.
$K_i \int (z_{des} - z)dt$ is the integral term that accounts for accumulated errors over time.
$f_{buoyancy}$ is the floatability compensation force.

An implementation of the PI controller could improve the depth control performance of the ROV. Unlike the proportional controller, which leaves a steady-state error, the addition of the Integral term allows the system to eliminate this error over time. The ROV was able to reach and maintain the desired depth with much higher accuracy.

During testing, we observed that the response was more stable and precise, even when external disturbances were introduced. The integral term helps correct small deviations that the proportional controller alone could not compensate for. However, tuning the integral gain K_i requires careful adjustment. If K_i was too high, the ROV exhibited oscillations and instability. If too low, the system can take longer to correct depth errors. By gradually increasing K_i after tuning K_p, we found an optimal balance between error correction speed and stability.

The Proportional-Integral (PI) controller proved to be more effective than the basic Proportional (P) controller, especially for maintaining depth in real underwater conditions.

3.8. Is the PI robust toward external disturbances?

To test the robustness of the PI controller against external disturbances, we introduced a sudden change in buoyancy by attaching a 1L plastic bottle to the top of the ROV. This increased the upward force acting on the vehicle, simulating an external disturbance.

This may show that while the PI controller was able to compensate for steady-state errors, it struggled to fully counteract sudden disturbances. Initially, the ROV's depth deviated significantly when the disturbance was introduced, and although the integral term worked to correct this, the response was slow. The system can take time to stabilize, and in some cases, small oscillations may appear before settling.

This behaviour highlighted a key limitation of the PI controller like while it effectively eliminates steady-state errors