MARINE & MARITIME INTELLIGENT ROBOTICS

UNIVERSITÉ DE TOULON

Implementation of a Proportional controller for Heading and Depth control in a BlueROV

Ayesha ASLAM, Worachit KETRUNGSRI, Yosef GUEVARA.

Marine Mechatronics

March 31, 2023

Contents

1 Introduction

Remotely Operated Vehicles (ROVs) have become increasingly popular for a variety of underwater applications, such as offshore oil and gas operations, defense, aquaculture, and scientific research, where they are primarily used for inspection and intervention tasks. To fulfill these roles and operate efficiently underwater, the vehicles require accurate navigation and control systems that allow them to maneuver and maintain station with minimal input from the operator. This report focuses on the implementation of a Proportional controller on the Blue ROV to improve its control system.

To develop an effective control system, we used a system identification approach that involved immersion tank testing with the use of on-board sensors for parameter estimation. This approach allowed us to estimate unknown parameters from the experimental data using the least squares algorithm. By obtaining accurate estimates of the system parameters, we were able to design a Proportional controller that met the requirements of the ROV's operation.

Accurate control of ROVs is critical for their successful operation and completion of tasks. The implementation of a Proportional controller in our system has improved the ROV's navigation and control, enabling it to maintain its position and perform tasks more efficiently. Through our experimentation and analysis, we have gained valuable insights into the behavior of our system, identified areas for improvement, and validated the effectiveness of our approach.

2 BlueROV setup

The Blue Robotics BlueROV2 is an observation class mini ROV that is capable of depths of 100 meters. It includes four thrusters of type T200. On the surface, a topside computer is likewise running Ubuntu 20.04 ROS and a game-pad controller is supported for manual operation. Communication between the ROV and the topside is made via a 300 meters long neutrally buoyant CAT5 tether cable connected at either end to a Fathom-X Tether interface board.

Figure 1: BlueROV Setup.

3 T200 Thrusters

The BlueROV2 is equipped with six T200 motors, which are controlled by Electronic Speed Controllers (ESCs). To initiate the motors, the ESCs require a "stopped signal" that lasts a few seconds and corresponds to a 1500 microseconds PWM signal. Once this initialization process is complete, the ESCs are then operated using a PWM signal ranging from 1500 to 1900 μs for forward thrust and 1100 to 1500 μs for reverse thrust. The motors remain still within the range of 1470 to 1530 μs, which is known as the "dead zone."

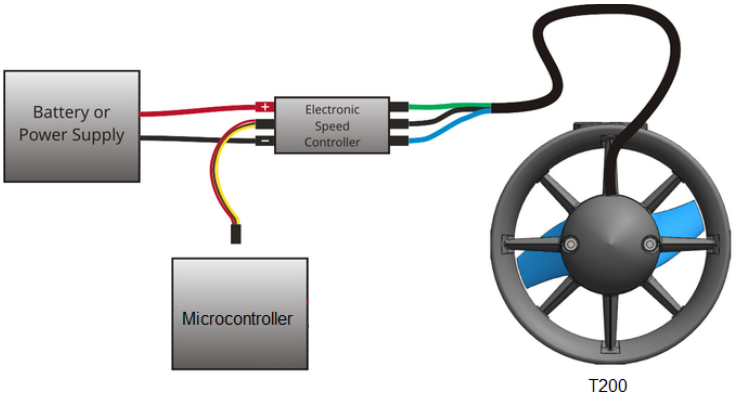


Figure 2: T200 wiring schematics

4 Practical Work 1.

4.1 PWM vs thrust relation of the thrusters

By interpolating the performance data of the T200 thrusters, it becomes possible to generate a curve that shows the relationship between the desired thrust (measured in Newtons and plotted horizontally) and the corresponding PWM signal (measured in μs and plotted vertically), with a 12 V battery serving as the reference.

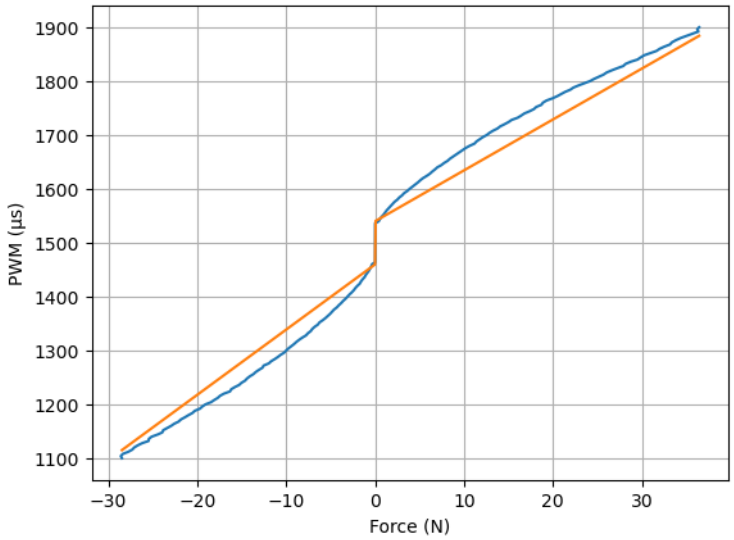


Figure 3: Lineal interpolation of the TR200 performance.

As a result, the behavior of the TR200 performance for the negative and positive thrust are given by:

$$Negative\ thrusts: \ PWM(f) = 9.447 \times f + 1568$$
$$Positive\ thrusts: \ PWM(f) = 12.110 \times f + 1433$$

```
def force2PWM(f):
        if f == 0:
                return 1500
        elif f > 0:
                return 6.792 + 1563.892*f
        else:
                return 8.725 + 1437.511*f
#############
```

Figure 4: Negative and Positive Thrust values

4.2 Experimental validation of the PWM vs thrust function

To validate the model, an empty 1.5-liter plastic bottle was attached to the top of the ROV. Using the previously developed model, a PWM value of 1603μs was determined to generate a force equivalent to 1.5 kg or 0.37 kgf per thruster. This value fell inside the manufacturer-specified range of 1592μs to 1604μs for thrusts ranging from 0.33 kgf to 0.43 kgf. The determined PWM value was then applied to the four vertical thrusters. However, due to the natural buoyancy of the ROV and the potential underestimation of the required PWM values, the bottle submerged slightly, as can be seen in the following image.

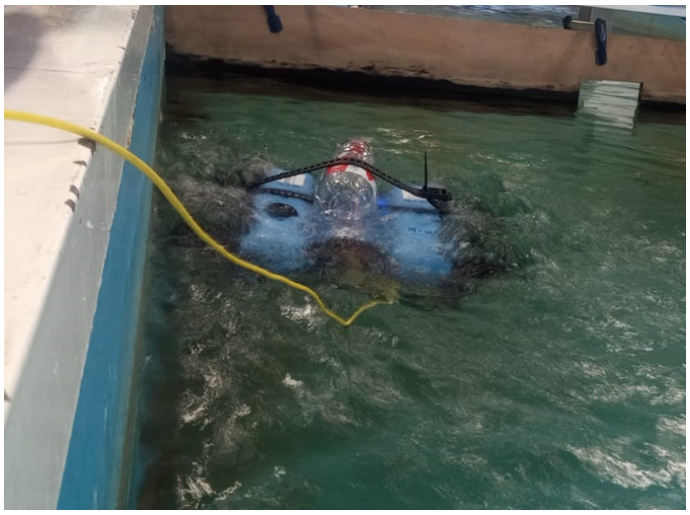


Figure 5: Validation of the PWM vs thrust function

4.3 Estimating the floatability

To achieve neutral buoyancy for the BlueROV 2 and determine the necessary additional force to counteract its buoyancy, an experiment was conducted. The experiment involved removing the bottle and filling it with water, which was then used to apply a force to submerge the BlueROV to its natural buoyancy. The experiment yielded a result where 1/3 of the bottle was submerged, indicating that a 1 kgf force was necessary to achieve neutral buoyancy. However, after we testing with the robot by using the algorithm implemented in the section 4.6 by set the $K_p = 0$ and let the robot maintain the stable position as we can see in figure 6, we arrived at the final floatability of 2.1 kgf or 20.58 N.

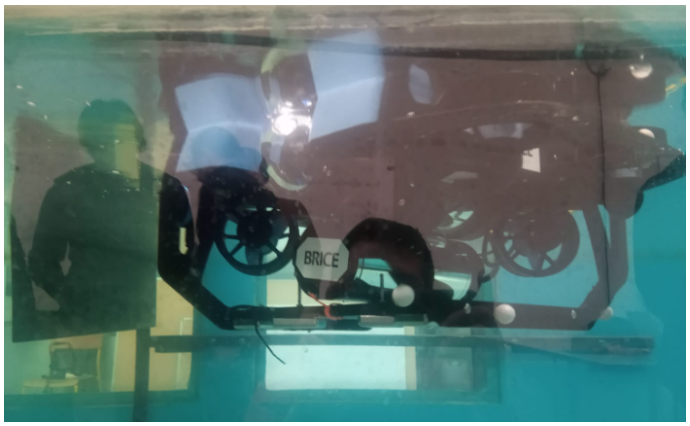
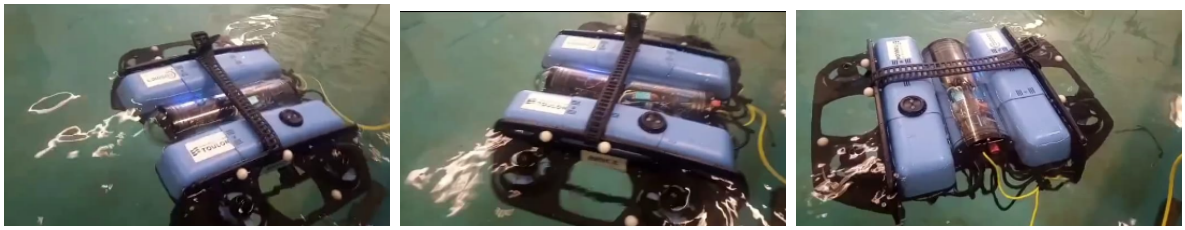


Figure 6: BlueROV in a neutral buoyancy

4.4 Implementing a Proportional controller (P) for the yaw

The goal of this experiment was on the yaw (ψ, heading) degree of freedom. By using a proportional controller, we were able to generate a torque vector $\tau = \gamma_z$ proportional to the yaw error ϵ. This was achieved by applying a torque γ_z, which is the torque created by the four horizontal thrusters.



(a) Yaw disrupted (CCW) (b) Desired yaw position (c) Yaw disrupted (CW)

Figure 7: Proportional controller (P) for the yaw Results

In order to ensure the accuracy of our results, we restricted the definition of ψ to the interval $[0; 2\pi]$ and implemented appropriate precautions in our code to address the potential impact of $(\psi_{des} - \psi)$ around $\psi = 0$. To test our controller, we employed a step input and tuned the proportional gain (K_p) to achieve the desired results. Through the analysis of the curves for yaw and thruster values, we found that starting with a small value of K_p and gradually increasing it helped us to achieve the desired position more quickly, but it also decreased the stability of the robot. As a result, when the value of K_p was set too high, the robot exhibited oscillations. To achieve the desired results, a value of $K_p = 0.1$ was used for the final gain for the proportional yaw control.

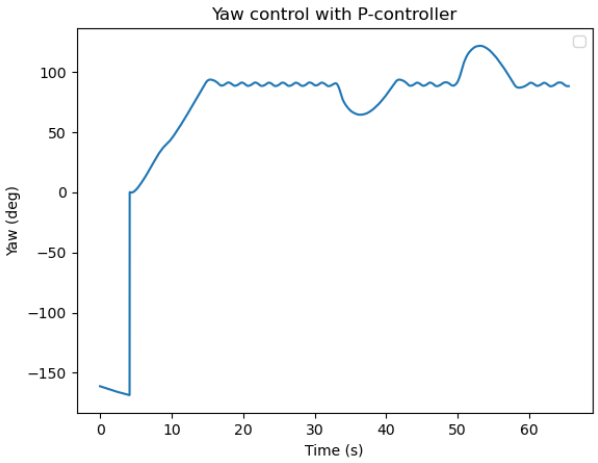


Figure 8: Yaw control with $K_p = 0.1$

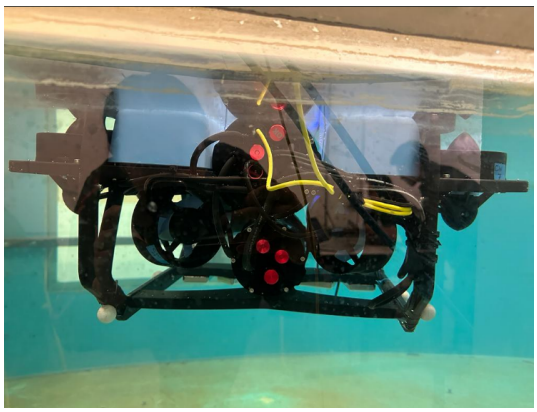
```
def pControlYaw(yaw_des, yaw):
        K_p = 0.1
        if math.abs(yaw_des - yaw) <= math.abs(2*math.pi + yaw_des - yaw)
                error = yaw_des - yaw
        else
                error = 2*math.pi*yaw_des - yaw

        return K_p*(yaw_des - yaw)
```

Figure 9: Python script Yaw angle control with Kp = 0.1

4.5 Implementing a Proportional controller for the depth

To implement a Proportional controller for the depth control of the Blue ROV. Since only the depth is being considered, the force and torque vector created by the controller is simplified to $\tau = f_z$, where f_z is the force that the vertical thrusters should produce together.By implementing the controller after 5 trials, the K_p of 50 was found, considering the residual error at steady state.



(a) Depth control with Kp = 0.5

```python
def pControlDepth(z_des, z):
        K_p = 1
        f_z = K_p*(z_des - z)
        return f_z
```

(b) Python Script for Depth control with P controller

Figure 10: Proportional control results and code.

Using the plots, an initial estimation of the K_p gain was obtained to maintain a constant depth.

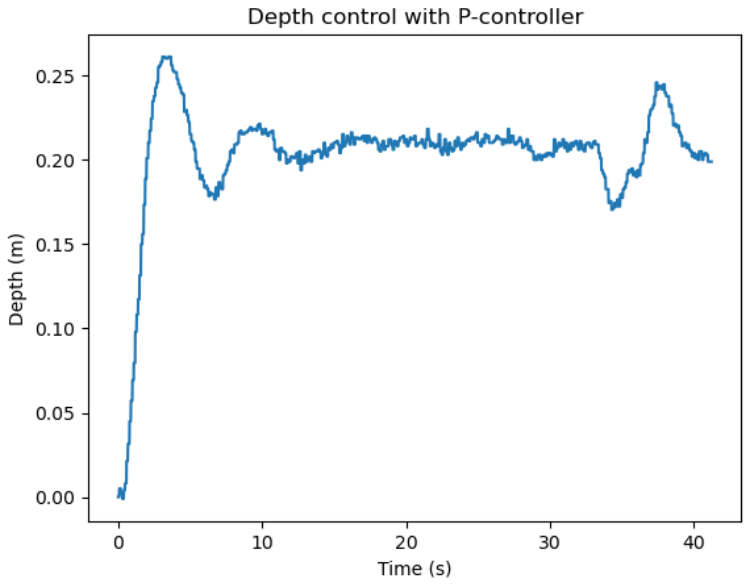
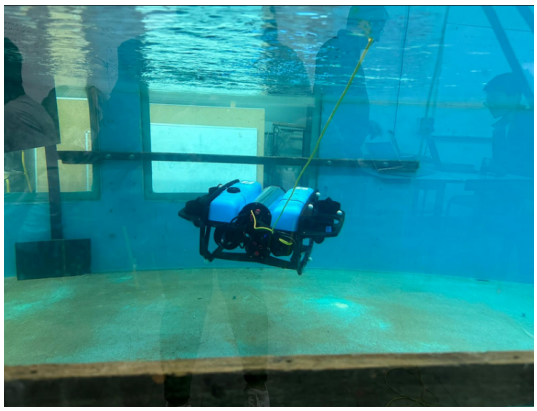


Figure 11: Depth control with P controller over the time with $z_{des} = 0.4$

In our experiment aimed to achieve a target position of $0.4m$ for the robot using our control system. However, a steady-state error was observed, potentially due to an inappropriate value of the proportional gain (K_p) in our controller. We tested different values of K_p to find the optimal gain, and also evaluated the system's ability to respond to disturbances by manually pushing the robot deeper into the water. Our results indicated that the robot was able to recover and maintain stability at around $0.2m$, indicating the effectiveness of our control system. Through analysis of potential sources of error and testing under different conditions, we were able to gain valuable insights and make improvements to enhance the system's performance.

4.6 Implementing a depth controller with floatability compensation

To implement a depth controller with floatability compensation to address the large steady state error observed during previous tests, which is caused by the ROV's floatability. A constant floatability compensation term is added to the controller to deal with this issue. The compensation term is based on the floatability measurement obtained in Step 3, and the controller is updated to $\tau_z = K_p(z_{des} - z) + f_{loatability}$.



(a) Depth controller w.floatability compensation

```
def pControlwFloatability(z_des, z, floatability):
        K_p = 1
        f_z = K_p*(z_des - z) + floatability
        return f_z
```

(b) Depth control code

Figure 12: Proportional control results and code.

Tuning the K_p gain again, result in an smaller gain than in the previous test ($K_p = 6$). Plotting and analyzing the depth and thruster curves is recommended. It should be noted that this approach works for buoyant ROVs as well as those that naturally sink, in which case the sign of g would be negative.

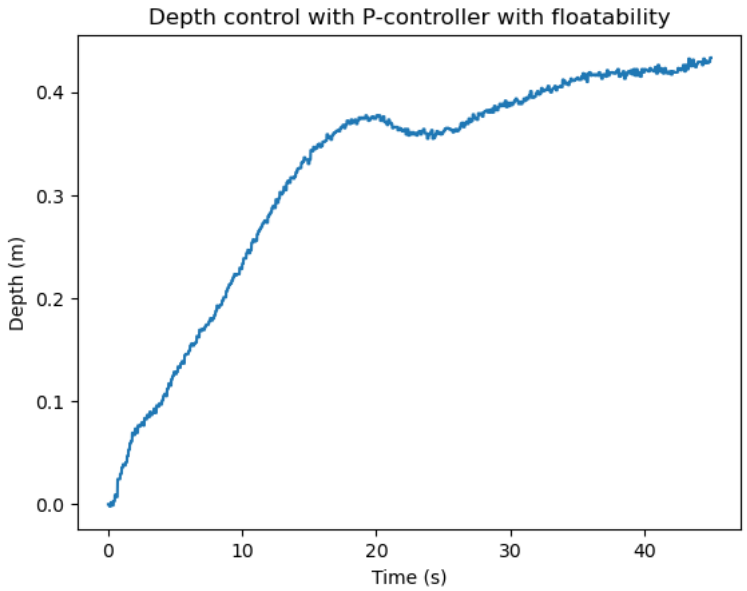


Figure 13: Depth control with P controller over the time with $z_{des} = 0.4$

As we can see from the above figure that after taken into account the floatability of the robot, the performance of the controller significantly increase as it can reach the target position at $0.4m$.

5 Practical Work 2.

5.1 Creating a trajectory compatible with the ROV's dynamics

The use of a step input as the desired trajectory can cause overshoots or oscillations in the robot's trajectory tracking due to the input not being physically feasible for the robot. This can make controller tuning more difficult. To overcome this issue and create a trajectory that is compatible with underwater vehicle dynamics, it is recommended to replace the standard step input with a simple cubic polynomial. The algorithm for creating this trajectory is provided below.

A trajectory can be defined by the following cubic polynomial:

$$\begin{gathered}\text{if } t < t_{\text{final}} \text{ then } z_{\text{desired}} = z_{\text{init}} + a_2 \cdot t^2 + a_3 \cdot t^3 \\ \text{if } t \geq t_{\text{final}} \text{ then } z_{\text{desired}} = z_{\text{final}} \\ \text{with } a_2 = \frac{3(z_{\text{final}} - z_{\text{init}})}{t_{\text{final}}^2} \\ \text{with } a_3 = \frac{-2(z_{\text{final}} - z_{\text{init}})}{t_{\text{final}}^3}\end{gathered}$$

Similarly, the desired trajectory for $\dot{z}_{\text{desired}}$, the derivative of the depth, can be generated as:

$$\begin{gathered}\text{if } t < t_{\text{final}} \text{ then } \dot{z}_{\text{desired}} = z_{\text{init}} + 2a_2 \cdot t + 3a_3 \cdot t^2 \\ \text{if } t \geq t_{\text{final}} \text{ then } \dot{z}_{\text{desired}} = 0\end{gathered}$$

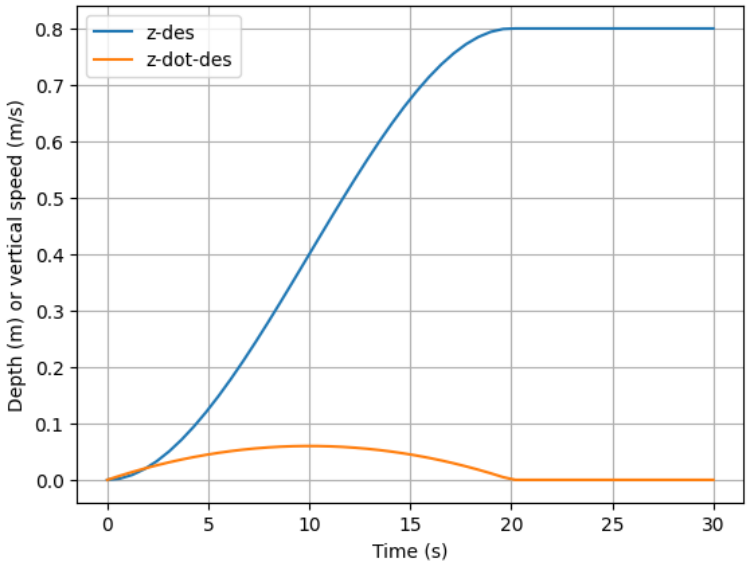


Figure 14: Depth controller with floatability compensation

5.2 Implementing the trajectory

To implement the trajectory designed, a P controller with gravity and buoyancy compensation should be implemented using the polynomial trajectory. The K_p value should be tested and tuned accordingly. To avoid integrating large initial errors, it is recommended to initiate the trajectory at the actual depth of the ROV when it is floating at the surface, accounting for the offset of the depth sensor. Finally, the resulting depth and thruster value curves should be plotted and analyzed.

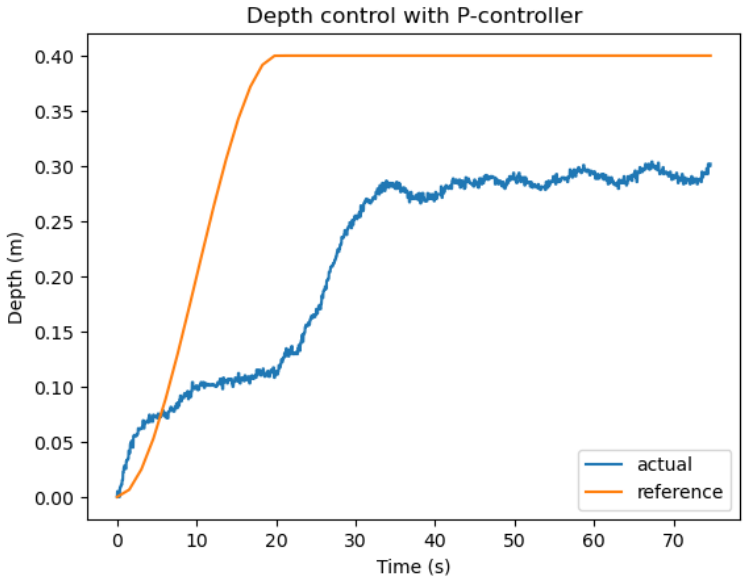


Figure 15: Depth control with P controller with floatability compensation

During the experiment, we initially set the gain of the Proportional (P) controller to the value specified in section 4.6. The resulting plot, shown in Figure 12, indicates that even when the reference trajectory of the robot was set to 0.4, the actual depth of the robot only reached around 0.3 and oscillated. This steady state error can be reduced by either increasing the value of Kp or correcting the floatability of the robot. However, a better approach to eliminate the oscillations and reduce the steady-state error is to use a Proportional Integral (PI) controller. By incorporating an integral term that integrates the error over time, the PI controller can effectively eliminate the steady-state error and provide smoother control. Therefore, implementing a PI controller would be a more effective solution to address the steady-state error observed in the P controller.

5.3 Proportional Integral controller (PI) for the depth

A Proportional Integral controller (PI) was implemented in Step 3 to eliminate the steady-state error observed during previous tests. The PI controller takes into account the integral of the error and adjusts the control output accordingly. The equation used for the controller is $\tau_z = K_p\tilde{z} + K_i \int \tilde{z}(t)dt + f_{loatability_{t_0}}$, where $\tilde{z} = (z_{des} - z)$ represents the depth error, and K_p and K_i are the proportional and integral gains, respectively. The PI controller was implemented in the code, and K_p and K_i were tuned to track the trajectory generated in Step 1. The curves of depth and thruster values were plotted and analyzed. To implement the integral term, a "sum" variable was created that was reset at the starting time of the trajectory. Finally, K_i was slowly increased after tuning K_p first to obtain optimal results.

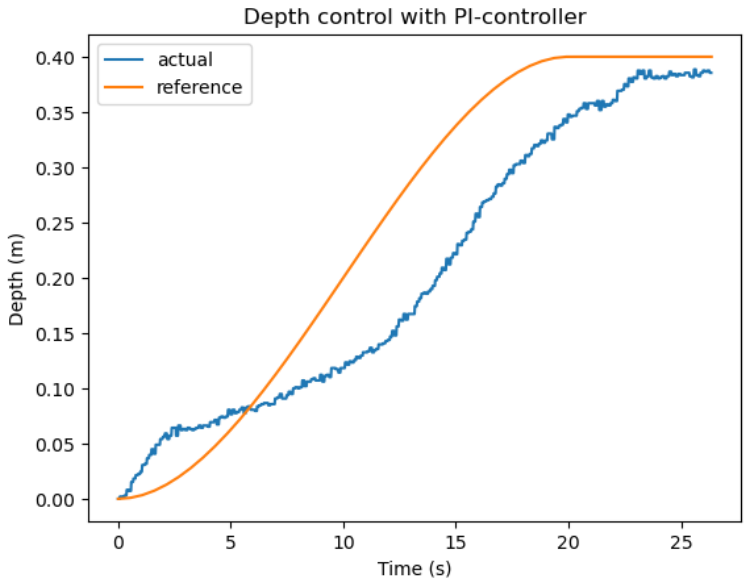


Figure 16: Depth control with PI controller with floatability compensation

During the experiment, the gain of the Proportional Integral (PI) controller was initially set to $K_p = 5$ and $K_i = 0.01$. As shown in Figure 15, the actual depth of the robot only reached around 0.3 and oscillated, even when the reference trajectory was set to 0.4. This steady-state error can be reduced by either increasing the value of K_p or correcting the floatability of the robot. However, a more effective approach to eliminate the oscillations and reduce the steady-state error is to use a PI controller. By incorporating an integral term that integrates the error over time, the PI controller can effectively eliminate the steady-state error and provide smoother control. Therefore, implementing a PI controller would be a more effective solution to address the steady-state error observed in the experiment. Further experimentation and tuning of the PI controller may be necessary to achieve optimal performance.

5.4 Is the PI robust toward external disturbances?

Without changing the gain obtained during previous tests by attaching an empty 1.5 liter plastic bottle. We then plotted the new depth and thrusters' values and observed that the blue curve, which represents the new results with the bottle attached, deviated significantly from the nominal test represented by the black curve. The dotted red curve indicates the desired depth. It is important to note that the PI or even the PID controllers used previously are not very robust controllers and may not be able to effectively deal with disturbances. There are several other controllers that exist to address this issue, including the sliding mode controller, L1 adaptive controller, and saturation-based nonlinear control, which are worth exploring further.

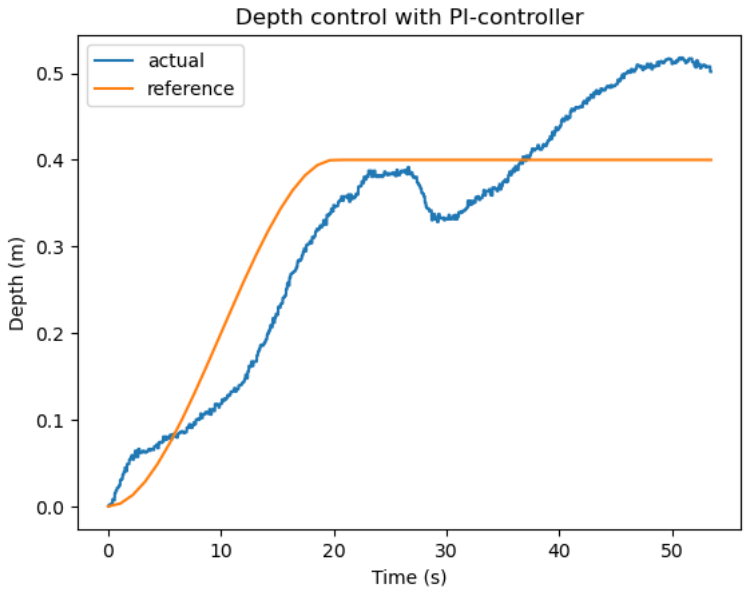


Figure 17: Depth control with PI controller with floatability compensation

In the same experiment, we try to disturb the robot as we can see that the actual position of the robot drop at $30s$, the robot manages to regain the position after that but overshoot away from the target position. The robot may take time to recover the target position after this due to the term, K_i which reduces the steady state error.

```
def piControlwFloatability(z_des, z, floatability, cum_error):
        K_p = 1
        K_i = 0.01
        error = z_des - z

        f_z = K_p*error + K_i*cum_error + floatability
        cum_error += error

        return f_z, cum_error
```

Figure 18: Python Script for Depth control with PI controller with floatability compensation

5.5 Proportional Integral controller (PI) for the yaw

To control the yaw, we implemented the trajectory generator and Proportional Integral (PI) controller in our code. We then tested our controller to track the trajectory and tuned the proportional gain (K_p) and integral gain (K_i) to achieve the desired results. To avoid the integration of large initial errors, we ensured that the trajectory started at the actual yaw angle of the ROV just before the beginning of the test. When plotting and analyzing the curves for depth and thruster values, we found that the PI controller was effective in controlling the yaw, with the controller output responding well to changes in the reference trajectory. However, tuning the gains was crucial to achieve accurate results, and it required careful observation and adjustment.

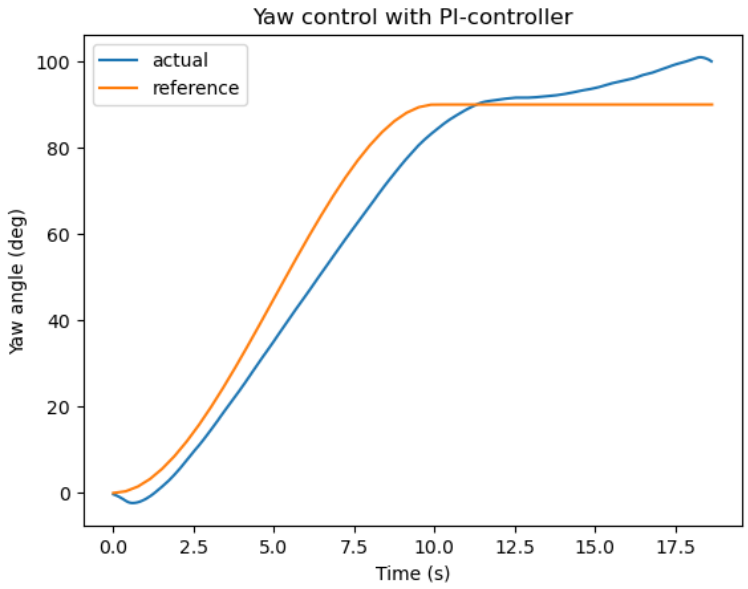


Figure 19: Depth control with PI controller with floatability compensation

```python
def piControlYaw(yaw_des, yaw, cum_error):
        K_p = 1
        K_i = 0.01
        if math.abs(yaw_des - yaw) <= math.abs(2*math.pi + yaw_des - yaw)
                error = yaw_des - yaw
        else
                error = 2*math.pi*yaw_des - yaw

        t_z = K_p*error + K_i*cum_error
        cum_error += error

        return t_z, cum_error
```

Figure 20: Python script for Depth control with PI controller with floatability compensation

Implementing the controller to control the yaw of the ROV, we determine the optimal values of the proportional gain (K_p) and integral gain (K_i) for the PI controller. After tuning them, we found that a value of $K_p = 1$ and $K_i = 0.01$ produced the desired results when the reference angle was set to 90. The controller manage to follow the trajectory and reach the desire position, however, due to incorrect value of K_i or measurement error of the sensor this may cause the integration error and make the actual trajectory slightly drift away from the target position.