MARINE & MARITIME INTELLIGENT ROBOTICS

UNIVERSITÉ DE TOULON

Marine Mechatronics Control of Underwater Robots

Practical work: Implementation of Image-Based Visual Servoing Control

Master MIR

Submitted By:

Student Name 1: **Mahmoud Aboelrayat**
ID: 22405136
Student Name 2: **Mohamed Magdy Atta**
ID: 22405169
Student Name 3: **Shahid Ahamed Hasib**
ID: 22405158
Student Name 4: **Mohamed Alsisi**
ID: 22405161

Submitted To

Prof. Dune Claire Prof. Vincent HUGEL

Universite de Toulon

2024/2025

May 11, 2025

Contents

1 Introduction

IBVS (Image-Based Visual Servoing) is a control strategy used in robotics where the robot's movements are directly controlled based on the error between the current image (from a camera) and a target image or position. The primary idea is to minimize the difference between the target object's image or position and its actual position in the camera view. This approach allows the robot to perform tasks like object tracking grasping, and different manipulation tasks solely using visual information from the camera.

In this practical work, we implemented visual servoing control to allow a BlueROV to track a buoy within an underwater environment.

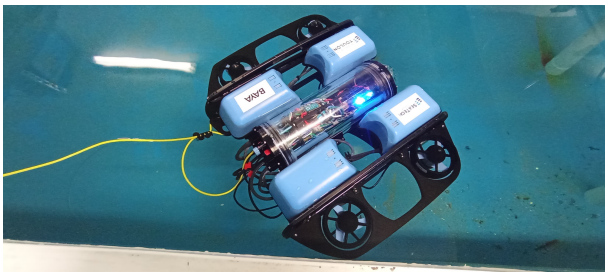


Figure 1: BlueROV

The control diagram is illustrated below:

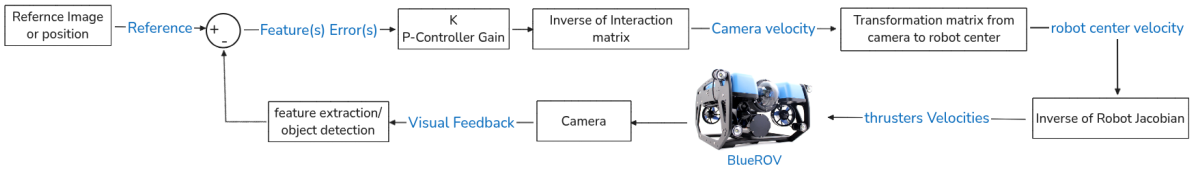


Figure 2: BlueROV IBVS Control Diagram

As shown in the diagram, an IBVS loop is executed in which a desired image feature is compared to the current feature extracted from the camera image, producing an error in image space. That error is scaled by a proportional gain and fed through the inverse of the interaction matrix (the image Jacobian) to compute the six-degree-of-freedom camera-frame velocity that will drive the feature toward its goal. A transformation then converts that camera twist into an equivalent body-frame velocity for the BlueROV, and the inverse of the vehicle's Jacobian allocates those desired surge, sway, heave and rotational velocities to individual thruster commands. The thrusters actuate, the BlueROV and camera move, new images are captured, and the loop repeats until the buoy's image error is driven to zero.

In addition, we implemented a visual servoing system for a Franka Panda robot arm in a PyBullet simulation environment to track a ball.

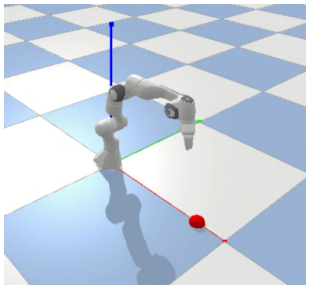


Figure 3: Franka Panda simulation

The control diagram is illustrated below:

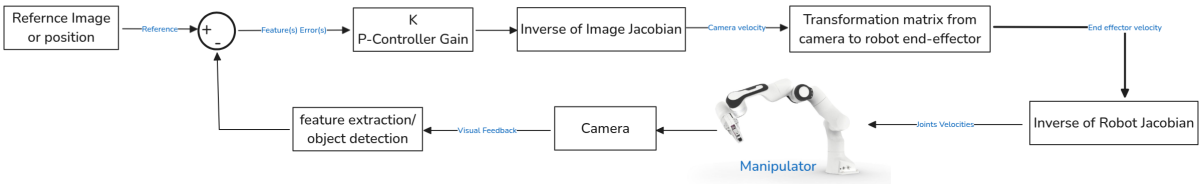


Figure 4: Simulation IBVS Control Diagram

2 Image Processing

2.1 Buoy Tracking

The accuracy of our image processing algorithm is based on the robustness of the buoy segmentation. The color and hue–saturation–value of the buoy region play an important role here. To make sure we capture this correctly, we implemented an interactive sampling routine that allow the user to observe these values and edit them manually to adapt to any buoy color and any water condition and have a robust lower- and upper-HSV bounds for buoy segmentation. The following picture shows how the output will be after the user start clicking on the buoy region to capture the hsv values.

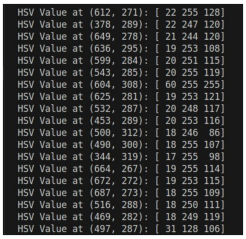


Figure 5: Capturing the Buoy HSV values using the mouse clicking routine

After setting up the HSV bounds, we move to the main algorithm which starts by converting the image to HSV and apply the manually defined lower/upper threshold to isolate the buoy's color, producing a binary mask. Nonzero mask pixels are extracted and used to crop the original image, which is converted to grayscale, Gaussian-blurred, and re-thresholded to clean noise. We then find all external contours, select the largest by area as the buoy, and fit a minimum-area rectangle around it. From that rectangle we take its longer side in pixels as the buoy width. Assuming a known physical width

and a calibrated focal length, we apply the pinhole-camera depth equation to estimate the x-distance:

$$d = \frac{W_{\text{real}}\, f}{W_{\text{pix}}}$$

In addition, the centroid of that largest contour becomes our "current_point" in image coordinates, which is then converted to meters. We used the image moments to obtain these centroids for the servo control. The following figure shows the results. If multiple buoys exist, we pick the nearest one whose distance is smallest. The following figure shows the results.

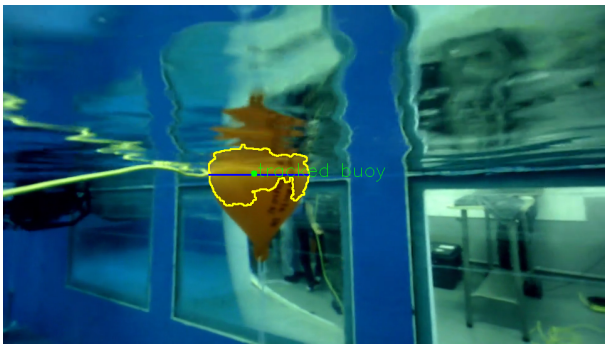


Figure 6: Image processing results

In our OpenCV-based python implementation we first register a mouse callback on the display window using `cv2.setMouseCallback("image", click_detect)`, where `click_detect` sets a global flag `get_hsv` and records the click coordinates `(mouseX,mouseY)` when the user presses the left button.

In each frame we convert the image to HSV via `cv2.cvtColor(image,cv2.COLOR_BGR2HSV)` and compute a binary mask with `cv2.inRange(hsv,lower_range,upper_range)`.

Nonzero pixels are located via `cv2.findNonZero`, and used to crop the original image (`cv2.bitwise_and`). The cropped image is converted to grayscale, blurred (`cv2.GaussianBlur`), and thresholded (`cv2.threshold`) to yield clean blobs.

We extract all external contours (`cv2.findContours`), select the largest by area, draw it, and fit a minimum-area rectangle (`cv2.minAreaRect`). From that rectangle we take the longer side in pixels as `buoy_width_px` and compute the distance in x-axis.

Finally, we compute image moments (`cv2.moments`) to obtain the centroid (c_X, c_Y), convert it to meters via the function `convertOnePoint2meter()`, and—if multiple candidates appear—select the one with minimal Euclidean distance. The resulting `current_point` and estimated depth are fed directly into the visual-servo controller.

2.2 Aruca Markers

To constrain all 6-DoF of the robot, a board with six ArUco markers was used. By tracking the relative positions of any three non-collinear markers on the board, the robot's pose (position and orientation) can be uniquely determined, as there is only one rigid transformation that aligns those three points between frames. However, detecting the markers underwater can be challenging due to factors such as the robot's movement, water distortions, and the small apparent size of the markers when the robot is positioned far from the board. To address those challenges a the received video was processed to enhance the visability and detectability of the markers. First scaling factor is applied

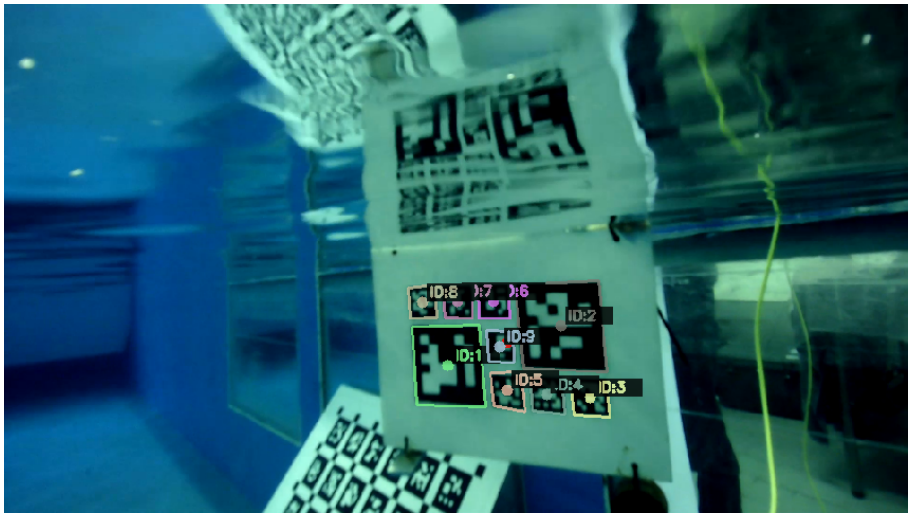


Figure 7: Aruca markres detection underwater.

to upscale the input image, effectively increasing the apparent size of the markers and making them easier to detect at longer distances. Additionally, a 3×3 sharpening kernel is used to enhance edge contrast and local features, which helps compensate for underwater blurring and refraction artifacts. Following this, the video frames were converted to grayscale to simplify processing and eliminate unnecessary color information, since the ArUco detection algorithm relies only on the contrast between the black and white patterns of the markers. Then the aruca library in OpenCV was used to preform the detection as shown in Fig. 7.

To ensure stable control output in cases where some markers are lost, an algorithm was developed to reconstruct the positions of all markers using at least three reliably detected ones. This approach allows the system to consistently compute the control commands using the interaction matrix of the same three reference markers, resulting in smoother and more stable outputs. The chosen markers for this purpose are IDs 1 and 2 — selected for their high detection reliability—and marker 7, which forms a well-conditioned triangulation with them. The individual interaction matrices for these three markers are computed and then vertically stacked to form a combined 6×6 interaction matrix. This matrix is used to derive the robot's velocity commands, following the visual servoing control logic detailed in the next section.

3 Visual Servoing

During working with the BlueROV in the pracctical sessions, we used the 2×6 *interaction matrix* (image Jacobian) that relates camera translational/angular velocities $v = [v_x, v_y, v_z, \omega_x, \omega_y, \omega_z]^\top$ to image–plane velocities $\dot{y}, \dot{z}$. Our goal was first to control the 2D image–plane of a point feature (y, x):

$$L_{2D} = \begin{bmatrix} -\frac{f_x}{Z} & 0 & \frac{f_x x}{Z} & \frac{f_x xy}{Z} & -f_x\left(1+\frac{x^2}{Z}\right) & f_x y \\ 0 & -\frac{f_y}{Z} & \frac{f_y y}{Z} & f_y\left(1+\frac{y^2}{Z}\right) & -\frac{f_y xy}{Z} & -f_y x \end{bmatrix}$$

Next, we also control the distance along z-axis which is the BlueROV surge motion in our case $\dot{z}$ by augmenting a third row in the interaction matrix to add a constraint to

the BlueROV surge motion depending on a desired distance between the buoy and the BlueROV :

$$L_{3D} \;=\; \begin{bmatrix} -\dfrac{f_x}{Z} & 0 & \dfrac{f_x\,x}{Z} & f_x\,x\,y & -\,f_x\big(1+x^2\big) & f_x\,y \\ 0 & -\dfrac{f_y}{Z} & \dfrac{f_y\,y}{Z} & f_y\big(1+y^2\big) & -\,f_y\,x\,y & -\,f_y\,x \\ 0 & 0 & \dfrac{f_x}{Z} & 0 & 0 & 0 \end{bmatrix}$$

The new full 3×6 Jacobian maps the 6-DOF camera velocities into pixel–space motions in x, y, z.

The error is equal to $e = [\ y_d - y,\ x_d - x\ ]^\top$ in the case of 2D control and $e = [\,x_d - x,\ y_d - y,\ z_d - z\,]^\top$ in the case of 3D control. The error is the difference between desired and current features positions.

Next, we start finding for the velocity that minimizes the image-space error. First, we compute the pseudo-inverse of the non-square interaction matrix L:

$$L^+ \;=\; \text{PseudoInverse}(L)\,,$$

where L^+ is the 2×6 or 3×6 matrix mapping image-space velocities back into the camera-frame velocities. We then apply a proportional control law in image-space by multiplying this pseudo-inverse by the negative error vector and the gain K_p:

$$v_{\text{cam}} \;=\; -\,K_p\,L^+\,e\,.$$

Here v_{cam} contains the three linear and three angular velocity components which, when executed by the camera/ROV, will drive the feature error e toward zero.

Finally, we transform this twist into the ROV body frame via the spatial mapping

$$v_{\text{rov}} = \begin{bmatrix} R & [t]_\times\,R \\ 0 & R \end{bmatrix} v_{\text{cam}},$$

where R and t are the rotation and translation from camera to center of the robot.

After that we publish the result is clipped between -0.5 m/s and 0.5 m/s to ensure a safe actuation and published as a ROS Twist message after this the speed are converted to pwm using a linear function

$$pwm = v * 400 + 1500$$

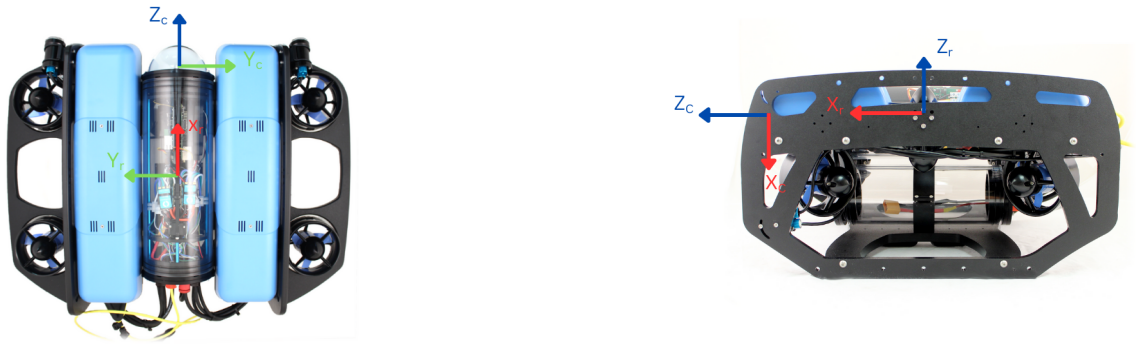
and sent to the thrusters to move the robot.

From 8 we can notice that the rotation matrix between the robot frame and the cmaera frame can be expressed as:

$$R = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

and from the fact that the electronics cylinder in the Bluerov2 is 30 cm length we can assume that the translation vector is:

$$t = \big[0, 0, -0.2\big]$$



(a) (b)

Figure 8: Robot and camera frames (a) top view, (b) side viw.

4 Autonomous Behavior

In order to operate the BlueROV autonomously without human intervention, we integrated buoy search and tracking into a simple two-state machine with states `SEARCH` and `TRACK`. In `SEARCH` mode the vehicle continuously rotates by a fixed yaw command until a buoy appears in the image; as soon as `current_point` becomes non-`None`, the state transitions to `TRACK`. In `TRACK` mode the IBVS loop described earlier is executed: the buoy's image centroid and desired point are overlaid on the frame, the current and desired feature vectors are published, the 6-DOF camera twist is computed and transformed into body-frame velocities, and the resulting thrust commands are sent. If the buoy is ever lost (`current_point` returns to `None`), the machine returns to `SEARCH` as below.

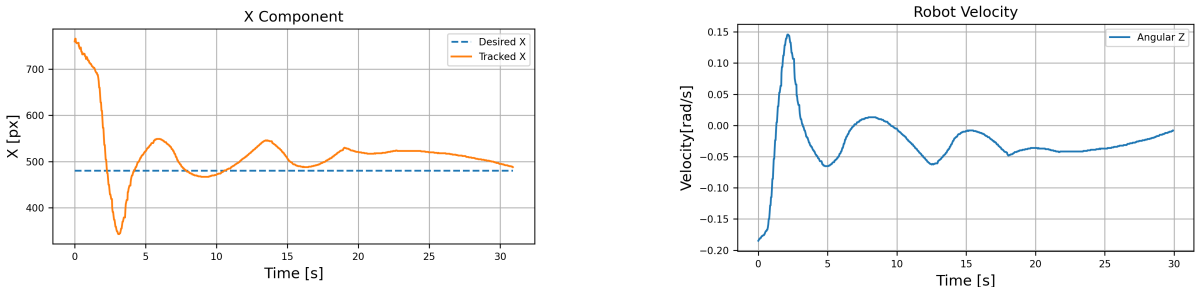
```
# In process_frame():
if current_point exists:
    # TRACK state: draw and publish valid feature data
    publish_data(current_point, buoy_width_px, distance)
    publish_desired_data(desired_point)
else:
    # SEARCH state: publish negative values to trigger yaw scan
    publish_data([-1.0, -1.0, -1.0, -1.0])
    publish_desired_data([-1.0, -1.0, -1.0, -1.0])

# In color_video_tracking_callback():
if xp < 0:
    # SEARCH behavior: small constant yaw to continue scanning
    Camera_correction_yaw = small_constant_yaw()
else:
    # TRACK behavior: compute IBVS law
    error = calculate_error(desired_point, current_point)
    L = calculate_interaction_matrix()
    L_inv = pseudo_inverse(L)
    v_cam_6d = calculate_camera_velocity(kp, L_inv, error)
    v_rov_6d = transform_to_vehicle_frame(v_cam_6d)
    v_rov_6d = saturate_velocity(v_rov_6d)
    publish_twist(v_rov_6d)
```

5 Results

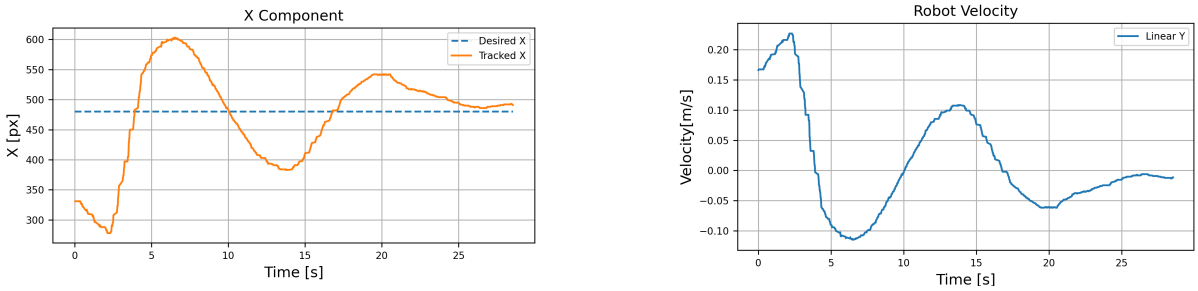
5.1 BlueROV Prictical Results

Two experiments were conducted to evaluate visual servoing performance with the BlueROV. The first experiment focused on controlling one degree of freedom (DoF) at a time using a simplified interaction matrix derived from two image lines, without incorporating any distance tracking. This approach aimed to isolate and assess the effect of individual motions—such as sway or yaw—on image-based control. However, results revealed that the robot exhibited unintended movement along the surge (x) axis when performing sway or yaw actions, indicating coupling between the degrees of freedom that the simplified model could not account for.



(a) (b)

Figure 9: Buoy visual serving using only yaw (a) shows the tracked point in camera frame and (b) is the robot's yaw speed.



(a) (b)

Figure 10: Buoy visual serving using only sway (a) shows the tracked point in camera frame and (b) is the robot's sway speed.

Consequently, a second experiment was designed to address this issue by simultaneously controlling both yaw and surge in order to maintain a constant distance from a buoy. This experiment demonstrated improved stability but also showed the ability of the algorithm to reject distributes shown in the videos in the appendix. The disturbances are represented in the plot by the sudden peaks in the tracked point position that required a high yaw and surge speeds. However, we can still notice that the robot is taking aggressive reactions especially on the surge which means the pid of this experiment still needs more tuning.

For all the experiments the vertical component was not controlled because the buoy was on the surface of the water and the robot cannot take any safe action to minimize this error.

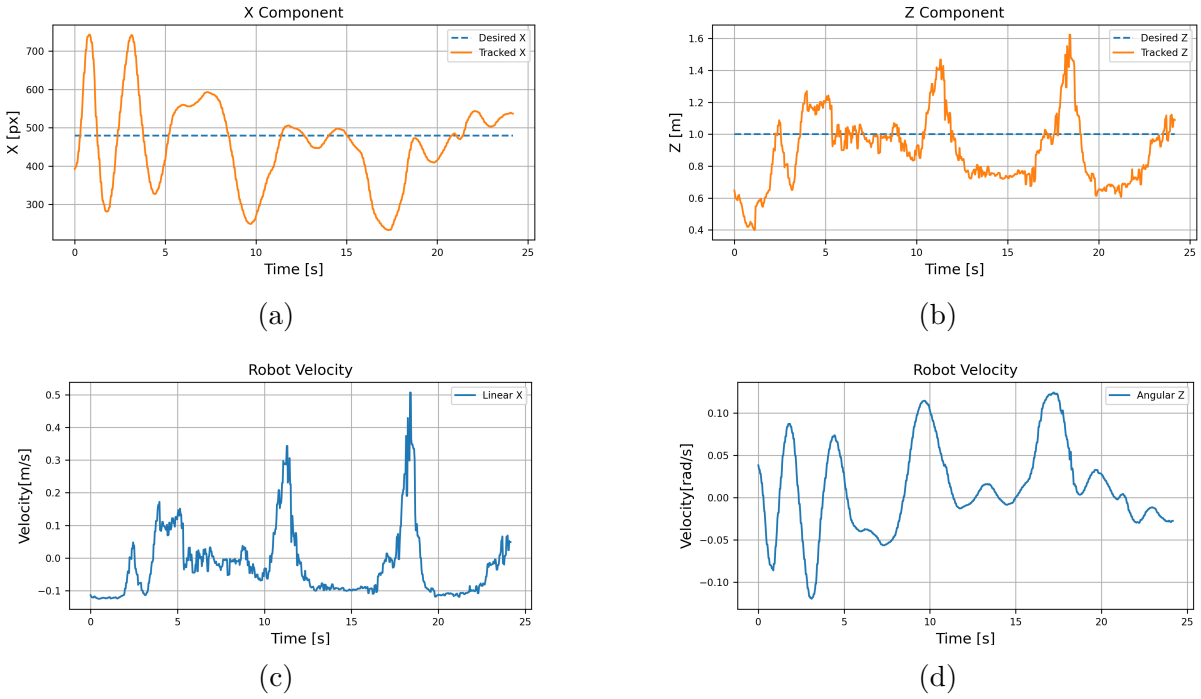


Figure 11: Buoy visual serving using yaw and surge (a) and (b) shows the tracked point in camera frame and (c), (d) are the robot's speeds.

5.2 Simulation Results

We implemented a Visual Servoing system for a Franka Panda robot in a PyBullet simulation environment. The robot tracks and moves toward a red ball using the Image-Based Visual Servoing (IBVS) control method. The system detects the ball in the camera feed, estimates its position in 3D space, and uses the visual error (displacement in image space) to control the robot's motion and align it with the object. We tested the tracking algorithm by applying different error amounts (different speeds for the target object) and the robot managed to reduce the error at all trials. The plotted simulation results is shown below.

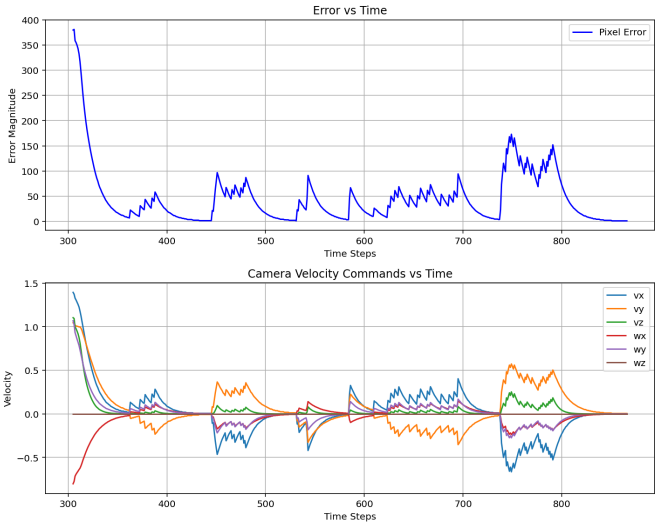


Figure 12: IBVS Simulation Results

6 Appendix

We uploaded all the code we developed for both BlueROV practicals and arm simulation to a github repository:

Github Link