UNIVERSITY OF TOULON

April 1, 2024

Marine Mechatronics: Visual Servoing

By:

Sergei Chashnikov

Abhimanyu Bhowmik

Madhushree Sannigrahi

Tayyab Tahir

Ishfaq Bhat

Under the supervision of: Prof. Claire Dune & Prof. Vincent Hugel

Contents

List of Figures

1 Introduction

Visual servoing, a significant aspect of robotics control, involves the use of computer vision techniques to guide the operation of robotic systems, particularly in dynamic and visually complex environments like underwater scenarios. This fusion of vision-based sensing and robotic control facilitates precise navigation, object detection, and decision-making, crucial for underwater exploration and infrastructure inspection tasks.

In this assignment, the objective is to implement visual servoing using OpenCV by detecting an orange buoy within an underwater environment. Subsequently, we aim to implement a Proportional (P) controller using an interaction matrix to regulate various degrees of freedom of a BlueROV. Initially, the controller focuses on regulating yaw-heave, and then sway-heave. Ultimately, the controller aims to oversee all five degrees of motion, except for pitch. Through this exercise, we explore the practical application of computer vision in underwater robotics and the efficacy of proportional control mechanisms in manoeuvring autonomous vehicles in dynamic aquatic settings.

2 Image-Based Visual Servoing Control (IBVS)

All vision-based control methods seek to minimize an error $e(t)$, which is commonly described as

$$e(t) = s(m(t), a) - s^* \tag{1}$$

where s^* represents the desired values of features, and
$s(m(t), a)$ is the computed values of features.

Here, vector $m(t)$ is a set of image measurements (centroid) and a is a set of parameters providing additional knowledge about the system.

Assuming a stationary target and a set goal position, s^* is assumed to be constant and changes in s are only dependent on camera motion. In IBVS, s consists of a set of features that are immediately available in the image data.

Here, we look at the scenario of regulating the motion of a camera with a velocity controller. The relationship between the time variation of s and $\dot{v}_c$ is given by

$$\dot{s} = L_s \dot{v}_c \tag{2}$$

in which $L_s \in \mathbb{R}^{k \times 6}$ is named the **interaction matrix** or **feature Jacobian** related to s.

$v_c = (v_c, \omega_c)$, v_c is the instantaneous linear velocity of the origin of the camera frame and ω_c) the instantaneous angular velocity of the camera frame.

Using equations 1 and 2, we get, the relations between camera velocity $\dot{v}_c$ and the time variation of the error $\dot{e}$:

$$\dot{e} = L_s \dot{v}_c \tag{3}$$

Here, $\dot{e} =$. Therefore,

$$\dot{v}_c = -\lambda L_e^+ e \tag{4}$$

where L_e^+ is the Moore-Penrose pseudo-inverse of L_s. When $k = 6$, if $detL_s = 0$ it is possible to invert L_s, stating the control as $\dot{v}_c = -\lambda L_e^{-1} e$.

More precisely, for a 3-D point with coordinates $X = (X, Y, Z)$ in the camera frame, which projects in the image as a 2-D point with coordinates $x = (x, y)$, we have:

$$x = X/Z = \frac{u - c_u}{f_\alpha}, \quad y = Y/Z = \frac{v - c_v}{f}, \tag{5}$$

where, c_u and c_v are the coordinates of the principal point,
f is the focal length, and
α is the ratio of the pixel dimensions
(u, v) gives the image point coordinates.

Deriving Eq 5, we get,

$$\dot{x} = \frac{\dot{X}}{Z} - \frac{X\dot{Z}}{Z^2} = \frac{\dot{X} - x\dot{Z}}{Z}; \dot{y} = \frac{\dot{Y}}{Z} - \frac{Y\dot{Z}}{Z^2} = \frac{\dot{Y} - y\dot{Z}}{Z}. \tag{6}$$

We can establish a connection between the velocity of the 3-D point and the spatial velocity of the camera using the well-known equation

$$\dot{X} = -v_c - \omega_c \times X \tag{7}$$

Grouping 6 and 7, we obtain,

$$\dot{x} = -\frac{v_x}{Z} + \frac{xv_z}{Z} + xy\omega_x - (1 + x^2)\omega_y + y\omega_z \quad (9) \tag{8}$$

$$\dot{y} = -\frac{v_y}{Z} + \frac{yv_z}{Z} + (1 + y^2)\omega_x - xy\omega_y - x\omega_z. \tag{9}$$

which can be written as $\dot{x} = L_x v_c$, where the interaction matrix L_x is:

$$L_x = \begin{bmatrix} -1/Z & 0 & x/Z & xy & -(1+x^2) & y \\ 0 & -1/Z & y/Z & 1+y^2 & -xy & -x \end{bmatrix} \tag{10}$$

Here, Z is the depth of the point relative to the camera frame. There are several options for constructing the estimate L^+_{est} to be used in the control law:

1. One common approach is to set $L^+_{\text{est}} = L^+_e$ if L_e is known, meaning if the current depth Z of each point is available. However, in practice, these parameters need to be estimated at each iteration of the control scheme.

2. Another option is to set $L^+_{\text{est}} = L^+_{e^*}$, where L_{e^*} is the value of L_e for the desired position ($e = e^* = 0$). In this case, L^+_{est} remains constant, and only the desired depth of each point needs to be specified, eliminating the need to estimate varying 3-D parameters during the visual servo.

3. A recently proposed approach is to set $L^+_{\text{est}} = \frac{1}{2}(L_e + L^*_e)^+$. This method requires L_e to be known, so the current depth of each point must also be available.

All three methods were used to estimate the interaction matrix between the current and desired position.

3 Methodology

This report's methodology section summarises the systematic way visual servoing was implemented on the BlueROV for it to track an orange buoy autonomously in a marine environment.

This section provides a comprehensive guide covering the entire process, from system setup and software configuration to detailed aspects of feature detection, tracking, and the application of control laws. It aims to provide practical insights into applying visual servoing techniques in marine robotics, addressing challenges and offering solutions for stable navigation of the BlueROV.

3.1 System Setup and Configuration

The foundation for implementing visual servoing on the BlueROV is laid through a software setup process. This involves the installation of the Robot Operating System (ROS) on the control computer and configuring it with Catkin to make sure that the entire system is fully functional. It will provide the framework to drive the data flows within and between the sensors, actuators, and visual servoing algorithms of the BlueROV.

3.1.1 Setting up the autonomous_rov Package:

The core of our project is the autonomous_rov package, designed specifically for this task. Once ROS is up and running, this package easily fits into the ROS environment. It includes bag files, system configurations, and launch files. In the scripts folder, we add code for image processing, tracking, and essential tools for controlling the robot.

We fine-tune this package for the BlueROV to recognize the orange buoy and accurately track its geometric centre according to our set conditions.

3.1.2 Calibration and Parameter Adjustment

Before deploying the system, a crucial step was calibrating the camera and adjusting essential camera parameters. This involves setting precise Hue, Saturation, and Value (HSV) thresholds for the orange buoy to ensure the camera can detect it effectively under different lighting conditions and water turbidity levels. Fine-tuning these parameters ensures that the feature detection algorithms operate optimally. For testing this, we increased the Hue, saturation and blue gradient of the input camera feed which helped in simulating the water effect.

3.2 Feature Detection and Tracking

The objective of this initial code is to designate a tracked point and a desired point, and then transmit them via a ROS topic to the visual servoing node. The tracked point will be set as the geometric centre of an orange buoy.

3.2.1 Conversion to HSV Color Space:

The image processing starts by converting the captured image from the RGB colour space to the HSV (Hue, Saturation, Value) colour space using **cv2.cvtColor()**. The resultant image can be seen in Figure 1(B) HSV colour space is often preferred for colour segmentation tasks as it separates intensity information from colour information, making it easier to threshold colours.

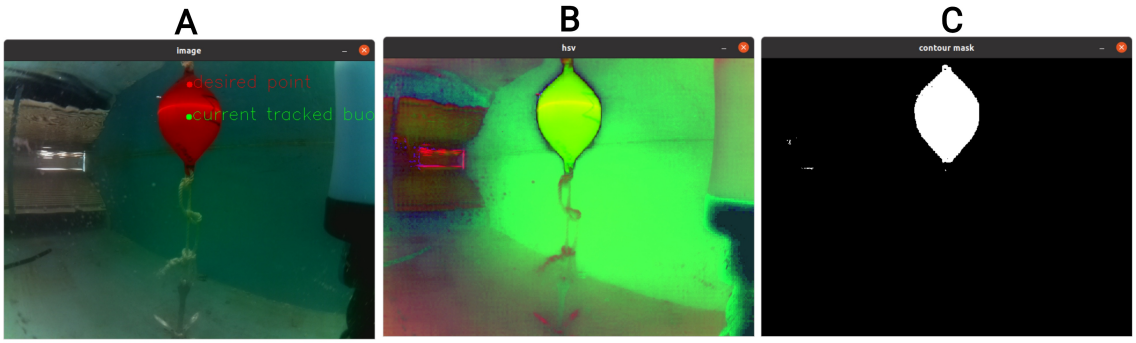

Figure 1: (A) RGB Image with track point and desired point overlay; (B) Image in HSV format; (C) Contour mask for the buoy

3.2.2 Color Masking:

After converting to HSV, a colour mask is applied to isolate the buoy's colour (assumed to be red/orange).

The **cv2.inRange()** function creates a binary mask where pixels within the specified HSV range(all orange pixels) are set to white(255), and others are set to zero (black). Considering our camera configurations, we took the lower range of red colour in HSV as $(0, 50, 50)$ and the upper range as $(10, 255, 255)$.

This creates a binary image where the buoy appears as white against a black background as seen in Figure 1(C). The image processing techniques employed focus on isolating the buoy's colour and extracting its position from the image.

3.2.3 Contour Detection:

Contours are then detected in the binary mask using **cv2.findContours()**. Contours are simply the boundaries of white regions in the binary image. Contour detection helps in precisely delineating the boundaries of the buoy, enabling accurate position estimation.

3.2.4 Centroid Calculation:

Finally, the centroid of the largest contour is computed to determine the buoy's current position in the image.

The code identifies the largest contour in the binary mask, which is assumed to correspond to the buoy. The centroid (centre of mass) of this contour is computed using the moments of the contour **(cv2.moments())**. The centroid coordinates (cX, cY) represent the current position of the buoy in pixel coordinates.

3.3 Interaction Matrix Computation

The interaction matrix (L) is the relation between the camera motion and the change in the observed feature point in the image plane. In simple words, it is the Jacobian between the current and desired position. For this project, we performed one feature point tracking, so the dimensionality would be 2x6. Using equation 10, we get,

$$L_e^* = \begin{bmatrix} -1 & 0 & cX & cXcY & -(1+cX^2) & cY \\ 0 & -1 & cY & 1+cY^2 & -cXcY & -cX \end{bmatrix} \tag{11}$$

Here, L_e^* is the interaction matrix for the actual point where (cX, cY) is the centroid coordinates of the buoy and $Z = 1$.

$$L_e = \begin{bmatrix} -1 & 0 & X_d & X_dY_d & -(1+X_d^2) & Y_d \\ 0 & -1 & Y_d & 1+Y_d^2 & -X_dY_d & -X_d \end{bmatrix} \tag{12}$$

Here, L_e is the interaction matrix for the desired point where,
$X_d = img\_width/2$,
$Y_d = img\_height/2 - 200$
and $Z = 1$.

We subtract 200 from Y_d to project the desired point at the upper side of the image frame rather than at the centre as in most cases, the buoy is near to the surface of the water and with a desired point at the centre, the BlueROV tries to "jump out" of the water.

All the 3 above-discussed methods in section 2 to predict the interaction matrix are implemented here. However, we got the best results with the 3rd option,i.e., $L_{\text{est}}^+ = \frac{1}{2}(L_e + L_e^*)^+$.

This arrangement reflects that tracking a single point would actually impose a limitation: the robot can, at best, control any two of six independent freedom degrees on visual feedback from just one feature.

3.3.1 Degrees of Freedom Management

The computation of control laws depends on the number of degrees of freedom we want to control. Two methods were employed to streamline the interaction matrix and control the BlueROV:

1. Deleting all degrees of freedom except **yaw and heave**.
2. Deleting all degrees of freedom except **sway and heave**.

This reduces the dimensionality of the v_{cam} vector and the resulting v_{cam} matrix to 2×2 and 2×1, respectively. Focusing on specific robot motions enhances tracking precision and stability.

We also tested with **5 degrees of freedom** of the BlueROV, i.e., keeping all the degrees of freedom except the pitch.

Experimentation with reducing degrees of freedom is conducted by adjusting "L" values post-computation. This matrix of interactions forms the foundation for efficient and targeted control actions in the upcoming stages of the methodology.

3.4 Control System Implementation

The control system computes all the necessary changes in the velocity such that, after implementing them, the interaction matrix can allow, while taking into consideration the error between the current and the desired position of the buoy.

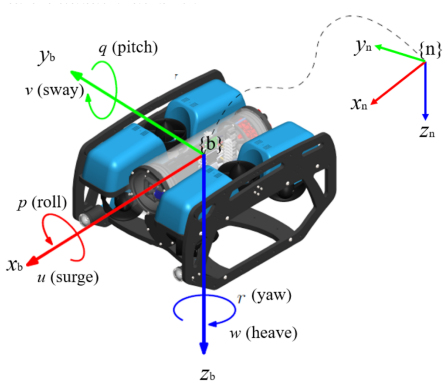

Figure 2: Degrees of Freedom of a BlueROV.

3.4.1 Proportional Controller

A proportional controller is a fundamental type of feedback control system that adjusts the control input based on the error between the desired setpoint and the current state of the system. It operates by multiplying the error by a constant gain, known as the proportional gain λ, to produce the control signal. Mathematically, the output of a proportional controller $u(t)$ can be expressed as:

$$u(t) = \lambda \cdot e(t)$$

Where:

- $u(t)$ is the control signal at time t,
- $e(t)$ is the error between the desired setpoint and the current state at time t, and
- λ is the proportional gain, which determines the sensitivity of the controller's response to changes in the error.

In essence, the proportional controller adjusts the control signal linearly in proportion to the error. Increasing the proportional gain increases the controller's responsiveness but may lead to instability or oscillations if set too high. Conversely, reducing the gain may improve stability but may result in slower response times.

3.4.2 Frame Transformation and Real-time Application

The computed velocities have to be converted to the body frame from the camera frame with the first frame as the camera frame, in order to get a meaningful and accurate control action. To achieve this, we apply a transformation matrix.

$$\text{Transformation\_matrix} = \begin{pmatrix} R & \text{skew\_pos} \cdot R \\ \mathbf{0}_{3\times3} & R \end{pmatrix}$$

where R is the Rotation matrix and *skew_pos* is the skew matrix of the position vector (position of the camera in the robot frame).

$$R = \begin{pmatrix} 0 & 0 & 1.0 \\ 1.0 & 0 & 0 \\ 0 & 1.0 & 0 \end{pmatrix}$$

$$\text{skew\_pos} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -0.159 \\ 0 & 0.159 & 0 \end{pmatrix}$$

This allows the transformation to be applied that considers the spatial orientation and configuration of the camera with respect to the body of BlueROV. These velocities are then applied in real-time to direct the robot's motion, so it continues to maintain visual focus on the buoy.

3.5 ROS Integration and Velocity Broadcasting

The project infrastructure is based on the Robot Operating System (ROS), which is aimed at facilitating communication between visual servoing algorithms and the hardware of BlueROV. The integration of developed software components in ROS makes a seamless pipeline for the transmission of control commands and visual data processing.

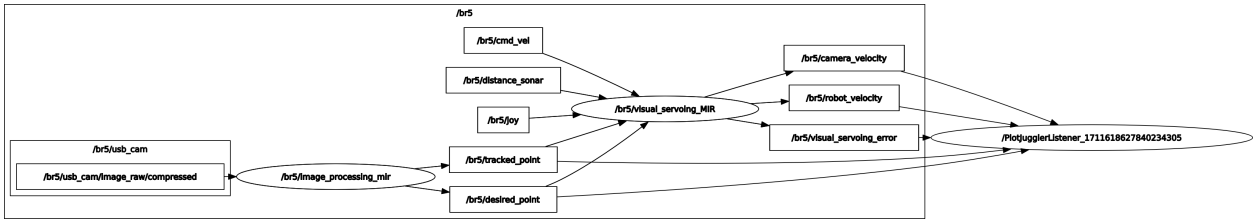

Figure 3: ROS Node structures.

Publishers in ROS are in charge of attaching data(Topics) to the ROS ecosystem, providing effective and efficient data transmission for the ecosystem. The camera velocity "camera_velocity", BlueROV body velocity "robot_velocity" and "visual servoing error" are published as ROS topics.

4 Results and Discussion

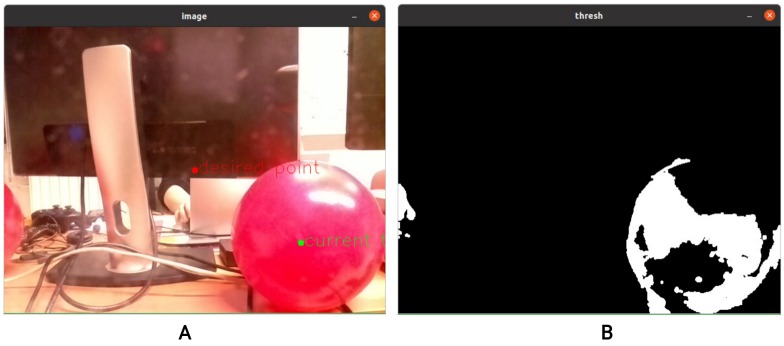

Figure 4: A. Orange buoy in the bag file; B. Color mask along with the current and desired point

4.1 System Responsiveness on bag files

Initially, the Buoy recognition algorithm was built and tested on the provided bag files. The bag files had videos of the buoy both with and without water. The image_processing algorithm worked almost perfectly in recognizing the buoy and providing the current and desired points as outputs.

4.2 Yaw-Heave Control

After testing the image processing on bag files, we tested the controller on the real tank. We tested for yaw and heave with a lambda(λ) value of 2. The figure 5 consists of 4 graphs. Graph (A) portrays the desired point in red and the actual point in green along with the trajectory of the actual point. Graph (B) plots the Euclidian norm of error. We can clearly see that after the initialization of the control in the BlueROV, the error decreased drastically and the actual point came very close to the desired point. The "hills" or oscillations observed are mainly due to the moving of the buoy to test the controller. It also depicts the inherent oscillations of a P controller around the desired points.

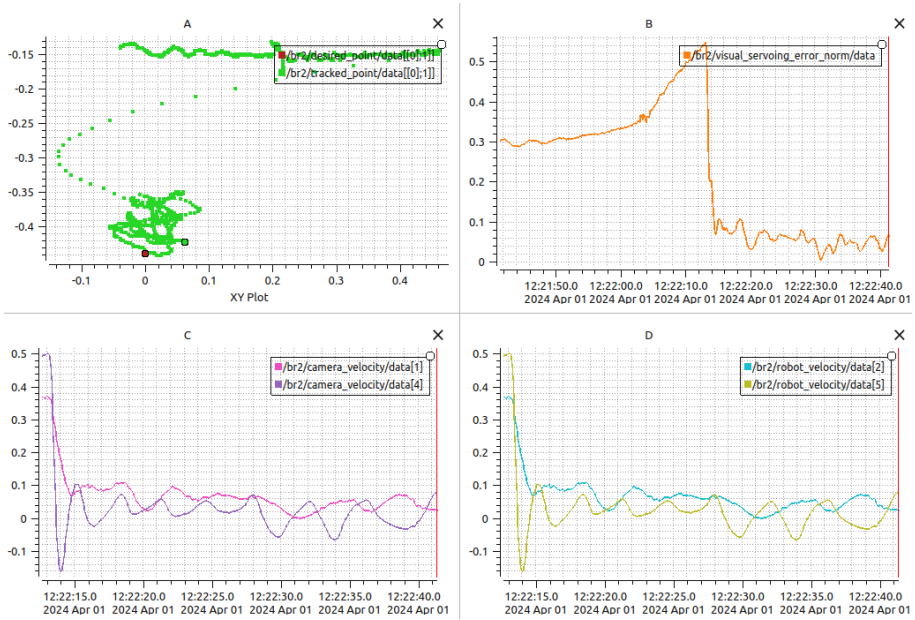

Figure 5: Yaw-Heave Control

Graphs (C) and (D) show the camera velocity and robot velocity respectively. The publishing of these topics starts after the controller is initialised in the ROV. Thus, the plot starts at a different time frame than the rest. In Graph(C), $data[1]$ and $data[4]$ show the y-axis and rotation along the y-axis of the camera respectively. This is the heave ($data[5]$) and yaw motion ($data25]$) for the body as seen in Graph (D).

4.3 Sway-Heave Control

Figure 6 consists of 4 graphs for sway and heave with a lambda(λ) value of 1.2. Just like the yaw and heave controller, we observe an immediate drop in error as the controller kicks in. Graph (A) shows the trajectory of the actual point with respect to the desired point in the camera frame. Since we are controlling the sway and not any angular movement like yaw, the trajectory is mostly linear, especially when compared to the yaw-heave control. Due to the high linear momentum, we encounter large oscillations in the x-axis of the camera frame during initial stabilisation.

Graph (B) is the L2-norm of the error. Here, we observe that after initial oscillations, the error converges to almost zero. To check the robustness of the controller, we moved the buoy externally, resulting in the subsequent error "bumps" in the graph. Graphs (C) and (D) illustrate the x and y velocities of the camera frame and the y and z velocities of the robot frame resulting in sway and heave.

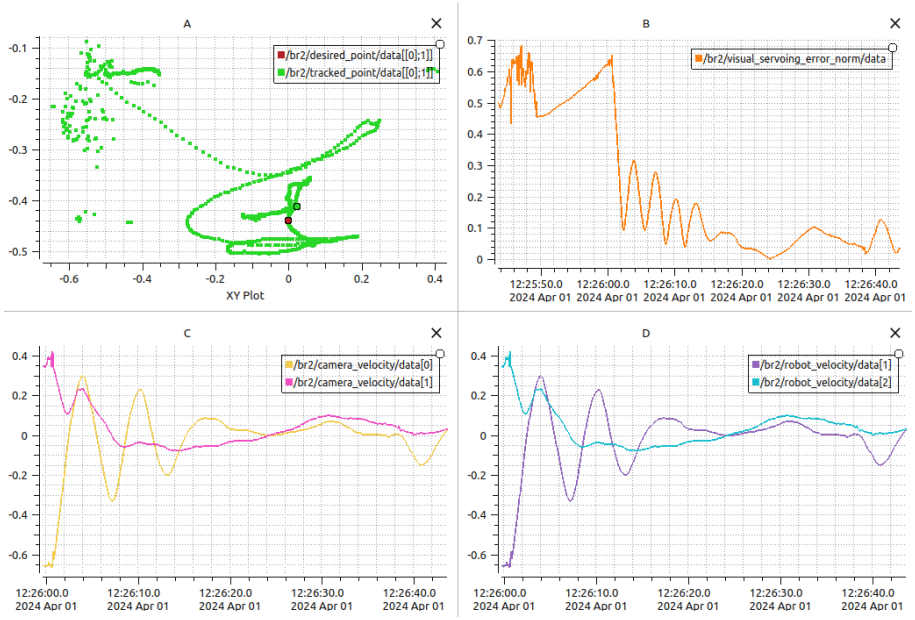

Figure 6: Sway-Heave Control

4.4 5-degrees of Freedom Control

Figure 7 demonstrates the graphs for the 5-degree controller with a lambda(λ) value of 1.5. Here we control all the degrees of freedom, except the pitch. We don't use pitch for the controller as the ROV can turn upside down to match the desired point, which is not what we intend to do. Graph (A) provides us with the trajectory of the desired point in the XY plane. We observed a constant offset in the heave here. This may be happening because the P-controller is not sufficient to take care of all 5 degrees of freedom. The L2-norm of errors in Graph (B) also confirms the error of $\leq 0.1m$. The jitters observed were mainly due to the struggling P-controller. We were also trying to move the buoy to different positions to check the ROV robustness, thus resulting in steep "hills".

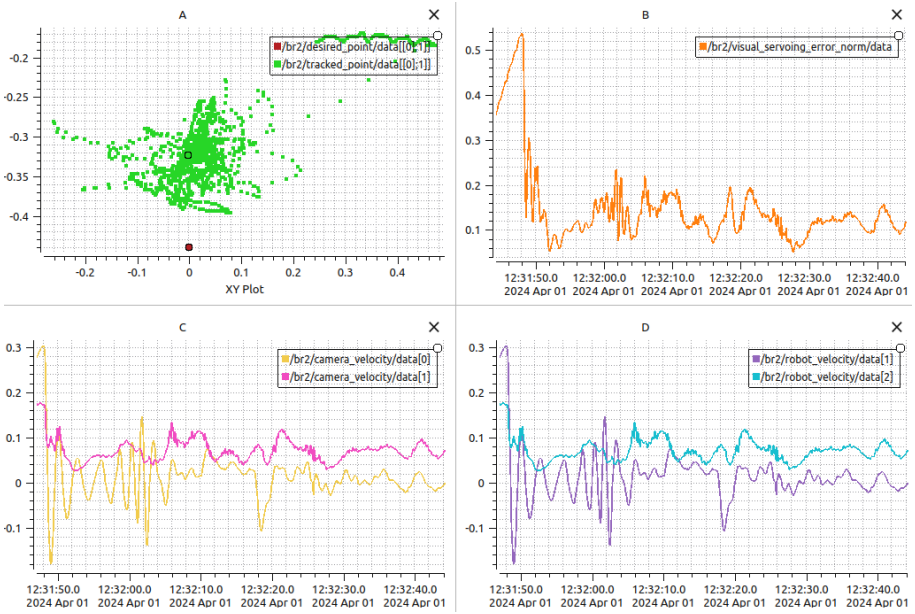

Figure 7: 5-Degrees Control

Graph (C) and (D) portrays the camera and ROV velocity in all directions except the pitch.