MIR MARINE & MARITIME INTELLIGENT ROBOTICS

UNIVERSITÉ DE TOULON

Visual servoing
Implementation on a BlueRov

Ayesha ASLAM, Worachit KETRUNGSRI, Yosef GUEVARA.

Marine Mechatronics

June 2, 2023

Contents

1 Introduction

Visual servoing using ROVs has become a popular technology due to its wide-ranging applications in underwater robotics. In this report, we will focus on visual servoing using a Blue Rov underwater robot, covering various aspects such as 2D point detection and tracking, interaction matrix, compute the control law, compute control values for mavros, change feature set, interaction matrix, and control behavior. The goal is to provide readers with a clear understanding of the principles and practical applications of visual servoing, along with a step-by-step guide to implementing visual servoing on a Blue Rov.

Remotely Operated Vehicles or ROVs are mobile underwater robots that are operated remotely from a surface vessel. These vehicles can perform a wide variety of tasks, such as observation, survey, intervention, construction, and more. For underwater robots to be useful, they must possess some level of autonomy, which refers to their ability to make independent decisions without human input. Visual servoing is one such method of controlling a robot's motion using real-time feedback from vision sensors to execute tasks.

In this project, we study a simple image-based visual servoing method to control the BlueROV2 ROV. The technique allows the ROV to track a submerged planar 8 point target despite unbalanced forces acting on the vehicle, such as buoyancy, gravity, and environment perturbations. Successful visual servoing requires a combination of good camera calibration, an accurate kinematic model of the robot, good inverse and forward kinematics solvers, and satisfactory camera performance (good pixel resolution, FPS, shutter speed, etc.).

2 Image-Based Visual Servoing (IBVS)

Image-Based Visual Servoing (IBVS) is a technique in robotics and computer vision that uses visual information from images to control the motion of a robot or camera. IBVS does not require explicit knowledge of the 3D scene or camera calibration, making it particularly useful in situations where these parameters may not be available or difficult to obtain accurately.

In IBVS, the control system operates by continuously analyzing the images obtained from one or more cameras and computing the necessary control commands to minimize the error between the current image and the desired image. The desired image represents the desired configuration or pose of the scene or objects being tracked.

The key steps in an IBVS system typically include:

1. Feature Extraction: Relevant visual features or landmarks are extracted from the images. These features can include edges, corners, keypoints, or other distinctive patterns that can be easily tracked and matched between images.
2. Image Error Calculation: The error between the current image and the desired image is computed. This error can be measured using various techniques such as image moments, image intensity differences, or feature correspondences.
3. Control Law Computation: Based on the image error, a control law or feedback controller is designed to generate the appropriate control commands for the robot or camera. The control law defines how the image error should be reduced over time by adjusting the robot's or camera's motion.
4. Control Command Execution: The control commands computed by the control law are sent to the robot or camera actuators to drive the desired motion. The process is repeated iteratively to continuously update the control commands based on the changing image error.

2.1 The Error in IBVS

The aim of all vision-based control schemes is to minimize an error $\mathbf{e}(t)$, which is typically defined by

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

The vector $m(t)$ represents a collection of image measurements (for example, the image coordinates of interest points or the image coordinates of an object's centroid). These image measurements are utilized to generate a vector of k visual features, $s(m(t), a)$, where a is a collection of parameters that indicate potential additional system information.

A velocity controller simplifies the control scheme in visual servoing. The camera velocity is related to the visual features and error through the interaction matrix. By choosing an exponential decrease in the error, the camera velocity can be computed using the pseudoinverse of the interaction matrix. This allows for error minimization and precise camera motion control.

2.2 The Interaction matrix and its geometrical interpretation

The interaction matrix $\mathbf{L_x}$ is a key component in visual servoing, connecting image features to camera velocity. It considers factors such as point depth, camera intrinsic parameters, and image coordinates. The velocity controller utilizes this matrix to compute the camera velocity for error minimization. However, accurate implementation requires careful estimation or approximation of the involved parameters.

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

The three different approaches for approximating the interaction matrix in visual servoing control are:

1. One approach is to use the actual interaction matrix if it is known, denoted as $\widehat{\mathbf{Le}^+} = \mathbf{Le}^+$, by assuming the current depth of each point is available. However, in practice, these parameters must be estimated at each iteration.

2. Another approach is to set $\widehat{\mathbf{Le}^+} = \mathbf{Le}^+$, where $\mathbf{L_e}$ represents the value of the interaction matrix for the desired position $\mathbf{e} = \mathbf{e}^* = 0$. In this case, $\widehat{\mathbf{L}_\mathbf{e}^+}$ remains constant, and only the desired depth of each point needs to be specified.

3. A proposed choice is $\widehat{\mathbf{Le}^+} = 1/2\,(\mathbf{Le} + \mathbf{L}_{\mathbf{e}^*})^+$, which offers good performance in practice. It combines both the current and desired interaction matrices, but the current depth of each point is still required.

The goal is to position the camera to observe a square as a centered square in the image. The simulations show the trajectories of image points, camera velocity components, and 3D camera trajectories for different choices of $\widehat{\mathbf{Le}^+}$. The results indicate that the choice of $\widehat{\mathbf{Le}^+} = 1/2\,(\mathbf{Le} + \mathbf{Le}^*)^+$ provides favorable behavior in terms of image, camera velocity, and 3D trajectory.

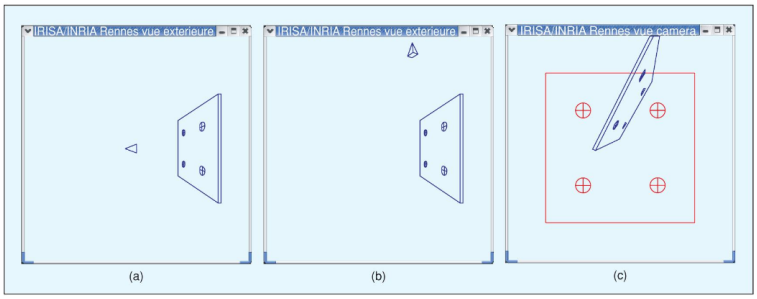

Figure 1: BlueROV Frontview

Nevertheless, using the interaction matrix $\mathbf{Le}^+$ in the control scheme induces straight-line trajectories for points in the image but leads to an unexpected translational motion of the camera.Alternatively, using $\mathbf{Le}^+$ generates straight-line trajectories in the opposite direction. The choice of $\widehat{\mathbf{Le}^+} = 1/2\,(\mathbf{Le} + \mathbf{L_e})^+$ results in straight-line trajectories in the desired direction and a pure rotational camera motion around the optical axis.

3 BlueROV camera characteristics

The camera is a critical component of visual servoing, providing real-time feedback on the robot's position and orientation. To achieve accurate and precise control of a robot's motion, the camera must capture high-quality images at a sufficient resolution and frequency. The BlueRov underwater robot's camera, with its high resolution of 1080 pixels and a frequency of 200ms, provides a high level of detail and responsiveness. Moreover, the camera is specifically designed to operate effectively in the challenging and often unpredictable underwater environment, with features such as high-pressure resistance and low-light capabilities.

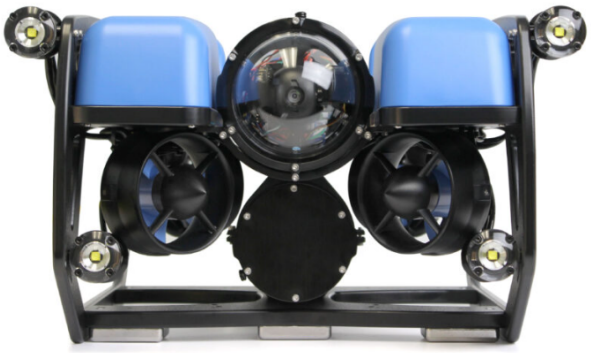

Figure 2: BlueROV Frontview

Accurate calibration and kinematic modeling are also essential for successful visual servoing, correcting for lens distortion and calculating the robot's position and orientation based on the camera's feedback. With its high-quality imaging and specialized design, the BlueRov camera is an ideal platform for experimenting with visual servoing in challenging underwater environments.

Camera	
Resolution	1080p
Camera Field of View	110 degrees horizontally
Tilt Range	+/- 90 degree camera tilt *(180 total range)*
Tilt Servo	Hitec HS-5055MG

Figure 3: BlueROV Camera Datasheet

4 Code explanation

4.1 Visual servoing code

To compute interaction matrices that are essential for visual servoing applications, the interaction matrices relate the motion of the camera or robot to the observed points or features in the scene.

```
# Interaction matrix for a point with rho theta representation
 def interactionMatrixFeaturePointRhoTheta(rho, theta):
     L = np.zeros((2, 6))
     return L
```

Listing 1: Interaction matrix for a point with rho theta representation

This function calculates the interaction matrix for a point represented by its rho and theta values.

```
 # Interaction matrix for a segment
 def interactionMatrixFeaturePointSegment(xm, ym, l, alpha):
     L = np.zeros((2, 6))
     return L
```

Listing 2: Interaction matrix for a segment

This function calculates the interaction matrix for a segment. The parameters xm and ym represent the midpoint coordinates of the segment, l represents the length of the segment, and alpha represents the orientation of the segment. However, in the provided code, the function returns a **2x6** matrix initialized with zeros.

```
 # Interaction matrix for point coordinates
 def interactionMatrixFeaturePoint2D(x, y, Z=1):
     Lx = np.array([-1 / Z, 0, x / Z, x * y, -(1 + x * x), y])
     Ly = np.array([0, -1 / Z, y / Z, 1 + y * y, -x * y, -x])
     L = np.stack((Lx.T, Ly.T), axis=0)
     return L
```

Listing 3: Interaction matrix for point coordinates

This function calculates the interaction matrix for a point represented by its 2D coordinates x and y. The optional parameter Z represents the depth or scale of the point and defaults to 1. The function calculates the interaction matrix based on the provided coordinates and depth and returns a 2x6 matrix L.

```
 # Stack of interaction matrices for a list of points
 def interactionMatrixFeaturePoint2DList(points, Zs):
     n = int(np.shape(points)[0] / 2)
     if len(Zs) != n:
         Zs = np.ones(n)

     L = []
     for p, Z in zip(np.array(points).reshape(n, 2), Zs):
         Lp = interactionMatrixFeaturePoint2D(p[0], p[1], Z)
         L.append(Lp)

     return np.concatenate(L, axis=0)
```

Listing 4: Stack of interaction matrices for a list of points

This function calculates the interaction matrix for a point represented by its 2D coordinates x and y. The optional parameter Z represents the depth or scale of the point and defaults to 1. The function calculates the interaction matrix based on the provided coordinates and depth and returns a 2x6 matrix L.

4.2 The trackercallback function

The **trackercallback** function handles the visual servoing process by computing the control law, transforming velocities between camera frame and robot frame, and publishing the necessary messages for visual servoing control and error visualization.

```
    # convert points to meters
    current_points_meter = cam.convertListPoint2meter(current_points)
    desired_points_meter = cam.convertListPoint2meter(desired_points_vs)

    # compute error
    error_vs = current_points_meter - desired_points_meter
```

Listing 5: Computing the error

The error computing is calculated by subtracting the desired points from the current points. The error represents the deviation or mismatch between the desired and actual positions of the points in the camera frame.

```
    # compute interaction matrix
    L = vs.interactionMatrixFeaturePoint2DList(current_points_meter, np.
    array([1]))

    # compute velocity
    vcam_vs = -lambda_vs * np.linalg.pinv(L) @ error_vs
```

Listing 6: Computing the Velocity

The velocity of the camera is computed using the interaction matrix L and the error vector. It involves finding the pseudo-inverse of the interaction matrix and multiplying it with the error vector. The negative value of lambda is also applied to scale the velocity to minimize the error and achieve the desired points.

```
    rtc = np.array([0, 0, 0])
    rrc = np.array([0, 90, 90])
    rVc = velocityTwistMatrix(rtc[0], rtc[1], rtc[2], rrc[0], rrc[1], rrc
    [2])

    vrobot = rVc.dot(vcam_vs)
```

Listing 7: Right relative position deduce relative position

The relative position between the camera frame and the robot frame is defined using the translation and rotation angles, respectively. Then velocity twist matrix is caculated to transform the camera frame velocity ($vcam_{vs}$) into the robot frame velocity ($vrobot$).

```
    roll_left_right = mapValueScalSat(vel.angular.x)
    pitch_left_right = mapValueScalSat(vel.angular.y)
    yaw_left_right = mapValueScalSat(vel.angular.z)
    forward_reverse = mapValueScalSat(vel.linear.x)
    lateral_left_right = mapValueScalSat(vel.linear.y)
    ascend_descend = mapValueScalSat(-vel.linear.z)

    setOverrideRCIN(pitch_left_right, roll_left_right, ascend_descend,
    yaw_left_right, forward_reverse, lateral_left_right)
```

Listing 8: Velocity mapping

The robot frame velocity ($vrobot$) is assigned to the corresponding fields of the vel variable, which represents the motor control commands. These commands control the movement of the robot in different directions (linear and angular).

5 In-Water Visual Servoing Testing and Performance Evaluation.

For the experiment, we encountered some issues with the camera on our initial BlueROV named "BRICE".These failures involved the robot capturing light reflections as features and attempting to exit the water, as can be seen in the following video.

Video - Issues camera reflections with BRICE.

As a result, we had to switch to a different BlueROV called "BELLA". However, we faced difficulties in calibrating both the yaw control and depth control of Bella. These challenges arose due to differences in the buoyancy points between the two robots.

5.1 Deep-control

Deep control is essential in IBVS for accurately controlling the distance between the camera and the target. The interaction matrix plays a key role in achieving deep control by relating the camera's motion to the visual features. Accurate calibration and estimation of the interaction matrix are crucial for reliable and robust performance of the IBVS system.

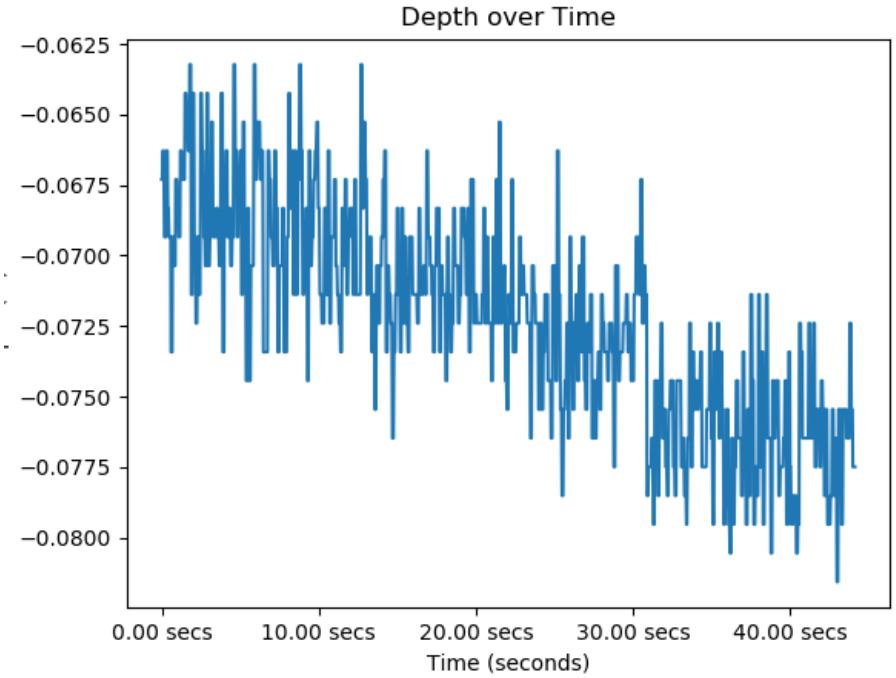

Figure 4: Depth

Nevertheless, during the experiment, we encountered two issues. Firstly, during the calibration process, the camera rotation was consistently biased towards the right when moving downwards. This resulted in a misalignment between the camera's orientation and the desired calibration angle. Secondly, when applying yaw control to the robot, it unexpectedly moved forward instead of rotating around its vertical axis as intended. As a result, we were compelled to keep the robot at the surface fixing the depth, as depicted in the previous figure, and only generate yaw control to align the visual features with the camera.

5.2 Camera tuning for underwater conditions

The following video exhibits how, after applying yaw control while maintaining a fixed depth and without moving towards the features, we observe the camera briefly aligning itself at 220° to match the 5 detected features. However, due to issues with the calibration of the robot's yaw control, maintaining the position that generates the optimal orientation towards the target points becomes challenging.

Video link results, In-Water Visual servoing.

The following graph illustrates how the robot consistently returns to the desired position. However, at around the 30-second mark, it loses sight of the features, causing the ROV's orientation to reach 230°. Nevertheless, it rapidly reverts to the desired position, showing the action of the yaw control.

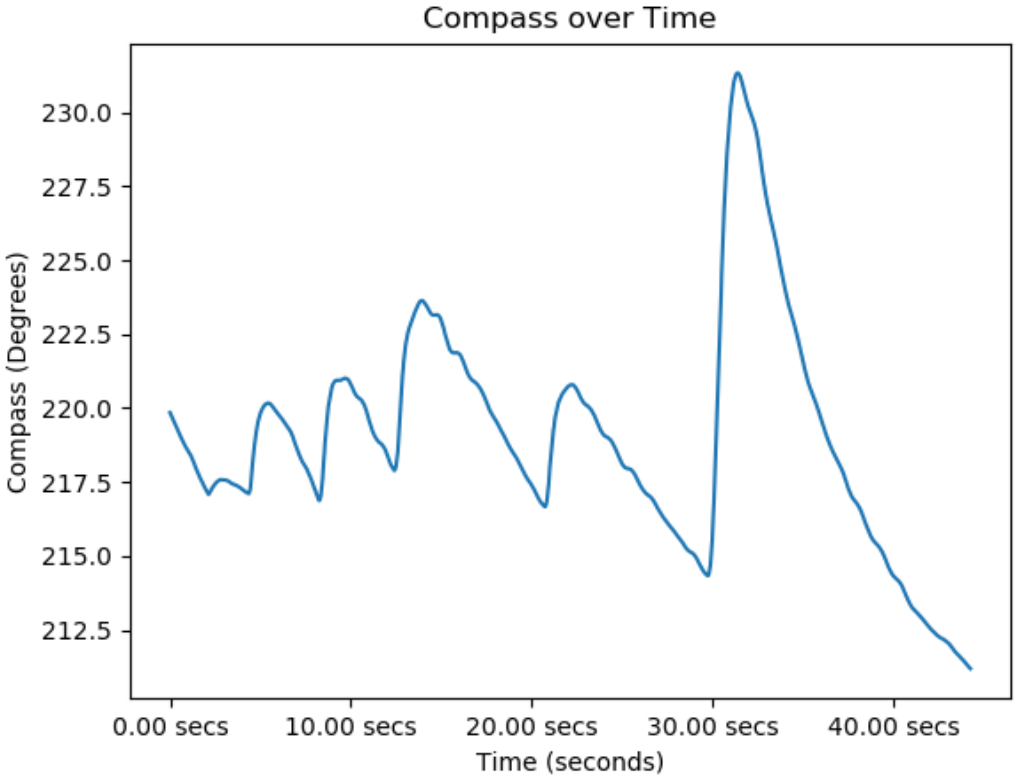

Figure 5: Compass

To measure the error in the camera alignment, we use the norm of the error for evaluating accuracy in this experiment. By calculating the norm, we considered both the magnitude and direction of the error in a unified manner. This approach allows us to assess the overall deviation between the desired and actual visual features. Additionally, using the norm facilitates straightforward comparison and analysis of the error across different time instances and experimental conditions. By leveraging the norm of the error, we gain a comprehensive understanding of the alignment performance and the effectiveness of the control strategy in minimizing discrepancies between the desired and observed features.

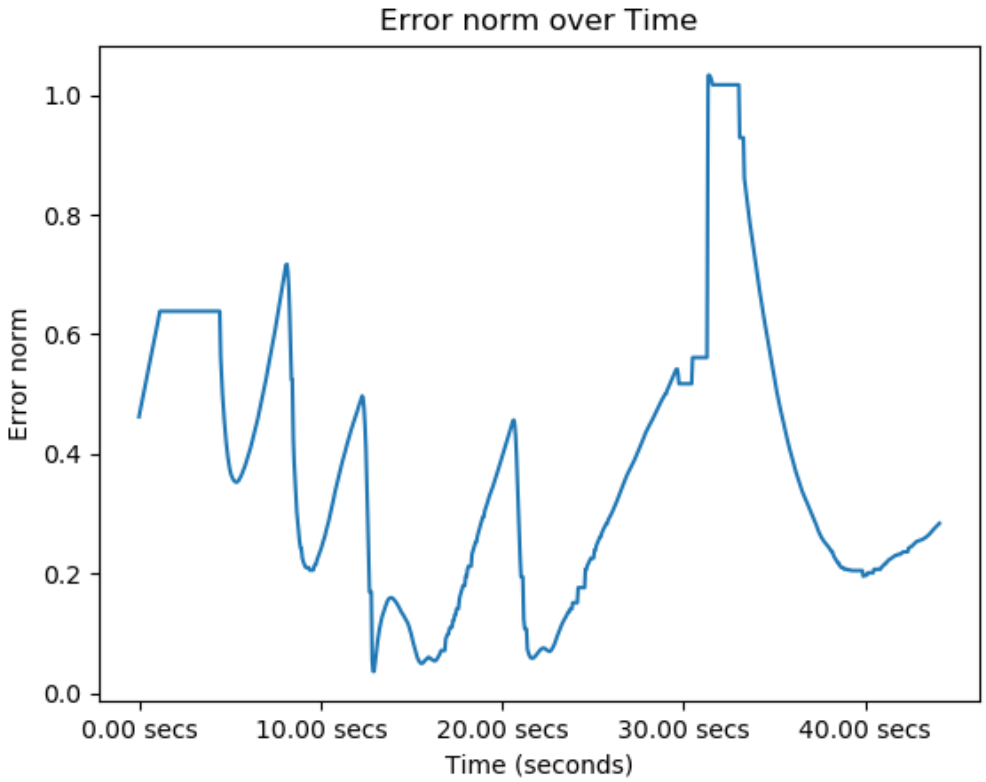

Figure 6: Error Norm

The previous plot describes how the norm of the error varies as the camera aligns with the robot. Initially, the error decreases until it reaches values close to zero. However, due to difficulties in maintaining the BlueROV orientation, the error increases around the 30-second mark. Nevertheless, the application of the control law helps reduce the error once again.

6 Conclusions.

The practical results highlight the challenges encountered during the application of image-based visual servoing (IBVS) using the Bluerov "Bella." Due to the difficulties in calibrating the yaw control and depth control, the system demonstrated intermittent moments of alignment with the desired features. Therefore, maintaining a stable position turns out difficult, leading to deviations in the robot's orientation.

The observed discrepancies can be attributed to calibration issues and the inherent differences in buoyancy points between the Bluerov models used. These factors affected the accuracy of the control system, causing unexpected motions and hindering the maintenance of optimal orientation towards the target points.

To address these challenges, careful calibration of the yaw control and depth control are crucial for further experiments. Ensuring and accurate alignment and stability for an IBVS system enables the precise positioning and tracking of the desired features.

Therefore, further investigations and refinements are necessary to overcome these calibration issues and optimize the performance of the visual servoing system. By addressing these challenges, we can improve the overall accuracy and reliability of the system, facilitating effective underwater robotic tasks and applications.