Image Moment-based Visual Servoing for Satellite Target Tracking using a Robotic Manipulator

Shayan Ghiasvand

A Thesis

in

The Department

of

Mechanical, Industrial and Aerospace Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Mechanical Engineering) at

Concordia University

Montréal, Québec, Canada

January 2024

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Shayan Ghiasvand**

Entitled: **Image Moment-based Visual Servoing for Satellite Target Tracking using a Robotic Manipulator**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Mechanical Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Youmin Zhang

_____ External Examiner
Dr. Shahin Hashtrudi Zad

_____ Examiner
Dr. Youmin Zhang

_____ Supervisor
Dr. Wen-Fang Xie

_____ Co-supervisor
Dr. Abolfazl Mohebbi

Approved by _____
Dr. Martin D. Pugh, Chair
Department of Mechanical, Industrial and Aerospace Engineering

_____ 2024

_____
Dr. Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Image Moment-based Visual Servoing for Satellite Target Tracking using a Robotic Manipulator

Shayan Ghiasvand

Robotic manipulators have become indispensable tools in space operations. Over the past few decades, manipulators like Canadarm2 have played significant roles, ranging from repair tasks to the complex process of capturing servicing satellites. On this basis, the main objective of this research is to automate the satellite-catching process on the International Space Station (ISS). The study employs an image moment-based visual servoing to fulfill the task. Various image features have been introduced to control the manipulator's movements using image moments. Yet, these features often face the challenging issue of coupling between six degrees of freedom movements, a problem that many researchers have aimed to solve. Nevertheless, previous literature has not completely addressed decoupling, which reveals the need for alternative approaches.

In this research, two novel approaches, function-based visual servoing and deep neural network (DNN)-based visual servoing, were developed to address this challenge. In the first approach, we introduce a novel general image feature function whose numerator and denominator are the polynomials with terms consisting of various image moments and adjustable parameters. Through the optimization process, two distinct rotational features about the x and y axes are formulated with the optimally tuned parameters. The second approach integrates DNN to estimate the 6D pose of the camera, yielding six decoupled image features.

Experimental results from the Denso manipulator confirm that decoupling image features can improve the controlling performance of the manipulator for capturing servicing satellites. The DNN-based visual servoing method could potentially enhance the performance of Canadarm2 in catching satellites, achieving a 32.04% average reduction in pose error and enhancing the velocity's precision by 21.67% over traditional methods.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my advisors, Dr. Wen Fang Xie and Dr. Abolfazl Mohebbi, for their unwavering support, invaluable guidance, and continuous encouragement throughout the course of this dissertation. Their expertise, patience, and dedication have been a great source of inspiration for me, and their insightful critiques have significantly improved this work.

I am also immensely grateful to my parents and my sister for their unwavering support and love. I would also like to thank my partner, Sheida, who has been a pillar of strength throughout this challenging journey. Their constant love, understanding, and belief in my abilities have been my driving force, and I dedicate this work to them.

I would also like to extend my sincere thanks to my friends Ehsan Zakeri, Alireza Saboukhi, Shervin Hakimi, Shervin Rasoulzadeh, and Pouria Alikhanifard. Their help, support, and camaraderie have been instrumental in the completion of this work. Their willingness to give their time and share their knowledge has made my research a rewarding and enjoyable experience.

Last but certainly not least, I wish to express my gratitude to the Mechanical Engineering Department of Concordia University for providing an environment conducive to research and innovation. The resources and facilities provided by the department have been invaluable in the successful completion of this dissertation.

This journey has been a significant milestone in my life, and I am grateful to everyone who has been a part of it.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the complex and dynamic field of space robotics, the quest for precision, efficiency, and reliability is gaining increasing attention. As our reliance on space-based technologies grows, so does the importance of robotic manipulators in maintaining and managing the diverse array of space assets. These manipulators, like Canadarm2, play a critical role in the International Space Station (ISS) operations.

Canadarm2 serves various functions, including the high-precision task of capturing incoming satellites (servicing or cargo) for berthing to the ISS. Traditionally, astronauts have been responsible for this complex activity by manually controlling the manipulator. Equipped with extensive training and aided by a camera mounted on Canadarm2's end-effector, astronauts use visual cues to align the arm with the grapple fixture[1] and finally catch the satellite. Specifically, the incoming satellites are outfitted with a grapple fixture and a 3D targeting pin with known dimensions (Figure 1.1). The astronaut's objective is to align the center of the camera's field of view with the targeting pin. The geometry is designed such that this alignment ensures that the end-effector can successfully connect to the grapple fixture.

Despite their skills and training, astronauts are not immune to human error. Any misalignment can have severe consequences, jeopardizing the mission, the costly hardware involved and even the satellite. In addition, astronaut involvement has significant financial costs, including training,

[1] Grapple fixtures are the ports on spacecrafts or satellites to provide a secure connection for a robotic arm (Wikipedia, 2023).

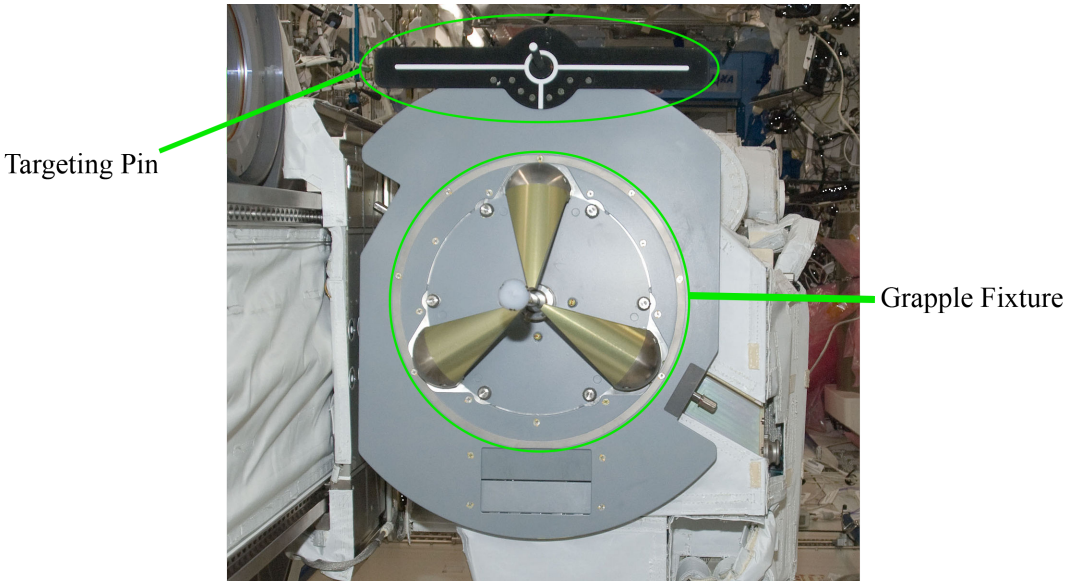


Figure 1.1: The grapple fixture and the 3D targeting pin on a servicing satellite. (JAXA, 2010)

launch, and safe return. These challenges highlight the necessity of automating the manipulator tasks, which is the major motivation of this research.

The final goal of this research is to fully automate this delicate satellite-capturing task by developing novel approaches to enhance precision and eliminate the risk associated with human error. While methods like visual servoing could potentially solve this problem, there exist some challenges. Since the visual features (one for each degree of freedom) used in such control strategies are intended for general objects, they are highly coupled, and thus the control signal generated based on them will result in motions in undesired directions. As a result, the performance is suboptimal for the specific geometry of the targeting pin and could lead to slow convergence rates and unpredictable end-effector movements, which are highly undesirable in the critical environment of space operations.

The methods and principles deduced from this study are transferable, so they can be generalized to improve any manipulator task, not just for Canadarm2 and the specific targeting pin but potentially for other robotic arms used in various marker-based applications.

The following sections will provide the research's specifics, the objectives and scope, and an overview of the methods that have been implemented. This research will contribute to the continuous evolution and advancement of space robotics, driving efficiencies and reducing reliance on

human intervention.

1.1 Objectives and Scope

The interaction matrix in robotic visual servoing, particularly in Image-Based Visual Servoing (IBVS), describes how the changes in the 3D pose of a scene lead to changes in its 2D projection in the image. In practice, this is often represented as a six-by-six matrix, where each row corresponds to an image feature chosen specifically for a degree of freedom of the robot's end effector. Meanwhile, each column of the matrix represents the relationship of these image features with a specific degree of freedom. By having these relations in the interaction matrix, the robot can adjust its movements based on the feedback from the camera, ensuring precise and desired actions.

A diagonal interaction matrix improves the system's performance by making the end effector's trajectory smoother and more linear in Cartesian space. Diagonalization of the interaction matrix reduces the interdependence or 'coupling' between various motions, which is key to achieving accurate and efficient convergence.

The interaction matrices derived from the set of image features frequently used in IBVS are not diagonal. Significant efforts have been directed toward finding the decoupled image features. Tahri and Chaumette (2005) suggested a set of image features aimed at decoupling the translational movements from each other. For the rotational movements around the camera's x and y axes (R_x and R_y), which are the most problematic features during the control process, Chaumette (2004) and S. Liu, Xie, and Su (2009) introduced a pair of image features that decoupled rotational movements from the translational movements. Although their interaction matrices exhibit some degree of decoupling, the image features presented do not result in a completely diagonal interaction matrix, and their effectiveness is conditional on the object's shape.

With this understanding, the main objective is to introduce a set of features specifically decoupled for the geometry of the targeting pin (Figure 1.1) used in satellite grasping. By doing so, this research aims to construct a diagonal interaction matrix (our principal performance metric), thereby reducing motion coupling and enabling a smooth and more precise convergence. A foundational step in our research is the 3D printing of a 2D object similar to the targeting pin. This was followed

by an image processing strategy for its detection, resulting in binary images suitable for further steps. Furthermore, the generation of a diverse data set from both simulated and real-world environments is an essential step for the development of our methods. Finally, two methods are being explored for the goal of decoupling image features. The first employs an optimization algorithm to find two feature functions with optimized parameters for the rotational degrees of freedom around the x and y axes of the camera (Chapter 4). The second uses a deep neural network to derive all six features needed for the control process (Chapter 5). These visual servoing methods can be used to control different robotic manipulators in marker-based applications.

1.2 Thesis Organization

This thesis has six main chapters, which are organized as follows. Chapter 1 briefly introduces the problem statement and objectives of this study. In Chapter 2, we conduct a literature survey on robotic manipulators, introduce the principles of image moment-based visual servoing, and investigate the ongoing efforts to obtain a diagonal interaction matrix. This chapter also gives an overview of optimization and neural networks and their application in visual servoing.

Chapter 3 provides the basis for the methods presented in the subsequent chapters. The chapter covers essential image processing techniques for converting RGB images to binary, allowing for image moment computations. Furthermore, the chapter introduces the main equipment used in our research and the algorithm used to generate the data set.

In Chapter 4, we introduce our initial method, focusing on the significance of decoupling image features. Following this, we discuss function-based visual servoing and the optimization algorithm used to decouple the image features. Subsequently, we will present the simulation results of the optimized image features.

In Chapter 5, we continue our attempt to decouple image features by introducing the 6D pose of the end effector as the six novel decoupled image features. To this end, the state-of-the-art deep neural network architecture has been leveraged. It is worth mentioning that, at the end of Chapters 4 and 5, our proposed methods are tested and validated.

Finally, Chapter 6 wraps up our research by summarizing our findings, highlighting our novel

contributions, and pointing out potential areas for future exploration.

1.3 Summary

This chapter starts by discussing the role of robotic manipulators (such as Canadarm2) in space operations, especially at the ISS. It discusses the problems astronauts face when trying to capture satellites manually and the risks of human errors. This leads to the main motivation of the research, which is mitigating such risks through automation. The challenge of automating the capturing operation using image-based visual servoing is then pointed out. Two methods, function-based visual servoing and DNN-based visual servoing, are presented as potential solutions. Finally, the chapter concludes with the organization of the thesis in section 1.2, providing a brief overview of the following chapters.

Chapter 2

Literature Review

This chapter will explore the foundational aspects and recent advancements in robotic manipulators and visual servoing, preparing for the topics explored in the subsequent chapters. We will start by discussing robotic manipulators, focusing on their utilization in space environments. A general explanation of the mathematical fundamentals of robotic systems (such as transformation matrices and geometric Jacobian) will follow. Subsequently, we will focus on visual servoing, a technique that uses visual feedback to control robotic systems. Here, its diverse applications, classification systems, and the aspects of controller design will be explained. Our literature review will then take a turn towards image-based visual servoing techniques that utilize image moments – the main principle of this research. Moreover, this chapter will clarify optimization concepts in the context of control systems and visual servoing. In addition, an introduction to neural networks and their revolutionary applications in control systems and visual servoing will be provided.

2.1 Robotic Manipulators

A robotic manipulator, often known as the ”arm” of the robot, is a mechanical structure made out of rigid parts, termed ”links,” which are connected through pivot points or ”joints” (Siciliano et al., 2010). Each manipulator has an end-effector - which could be a gripper, a tool, or any other device - that carries out the specific task designated to the robot.

The primary source of a manipulator’s movement comes from its joints. These joints can be of

two types: prismatic, which allows a straight or sliding movement between two links, and revolute, which enables a rotational or spinning motion between two links. Even though prismatic joints offer the advantage of linear motion, revolute joints are commonly chosen for their compact design and reliable performance (Siciliano et al., 2010).

Let's consider a typical open-chain manipulator with $n + 1$ links that are interconnected by n joints, with the first link (Link 0) traditionally anchored to a stationary point. Each joint forms the bridge between two adjacent links, which we'll refer to as Link $i - 1$ and Link i. We'll denote the joint as Axis i. We assume that each joint introduces a single DOF, symbolized by the joint variable. To compute the manipulator's position and orientation given the joint angles, we attach a coordinate frame to each link, starting from Link 0 and going up to Link n. Then, the transformation that describes the position and orientation of Frame n relative to Frame 0 can be represented as (Siciliano et al., 2010):

$$ {}^0\mathbf{T}_n(\mathbf{q}) = {}^0\,\mathbf{T}_1(q_1){}^1\mathbf{T}_2(q_2)\ldots{}^{n-1}\,\mathbf{T}_n(q_n). \tag{2.1} $$

Thus, we derive the direct kinematics by recursively multiplying the transformation matrices ${}^{i-1}\mathbf{T}_i(q_i)$ (where $i = 1, \ldots, n$), each of which is a function of a single joint variable (Siciliano et al., 2010).

To calculate the relative motion and position of two links, we utilize a technique known as the Denavit–Hartenberg convention (DH). This convention assigns a coordinate frame to each link and defines four parameters to describe the relative position and orientation of Frame i with respect to Frame $i - 1$ (Siciliano et al., 2010). These parameters are illustrated in Figure 2.1.

Depending on the type of joint connecting the links, either θ_i changes (for a revolute joint) or d_i changes (for a prismatic joint), while the other parameters stay constant.

To make this all more tangible, these parameters are used to form a transformation matrix between Frame i and Frame $i - 1$ (Siciliano et al., 2010):

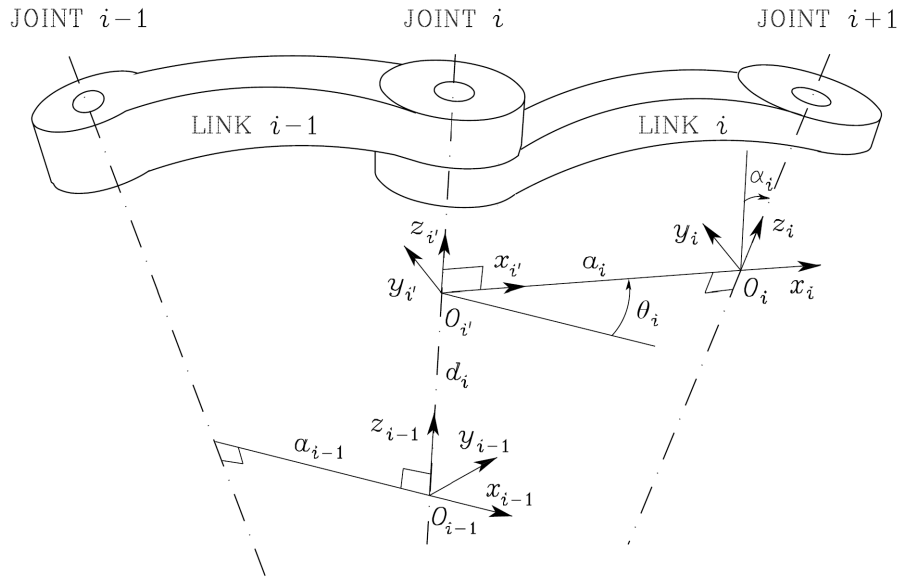


Figure 2.1: Denavit–Hartenberg kinematic parameters (Siciliano et al., 2010)

$$ {}^{i-1}\mathbf{T}_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.2} $$

This matrix is an instrumental tool to compute the direct kinematics of an open-chain manipulator systematically (Siciliano et al., 2010).

Differential kinematics aims to establish a connection between the velocities of the joint, denoted as $\dot{\mathbf{q}}$, and the end-effector velocities, denoted as $\mathbf{v}_e$. This relationship can be expressed linearly with respect to the joint velocities as (Siciliano et al., 2010):

$$ \mathbf{v}_e = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}. \tag{2.3} $$

This equation signifies the differential kinematics of the manipulator, where the matrix $\mathbf{J}$, which typically depends on the joint variables, denotes the geometric Jacobian of the manipulator.

2.1.1 Robotic Manipulators in Space

Robotic manipulators have been increasingly essential in space exploration and satellite maintenance as they improve operational safety and efficiency. Among them, Canadarm2 and Dextre are remarkable examples of such manipulators. As shown in Fig. 2.2 and 2.3, these manipulators are primarily used at the International Space Station (ISS).

The Canadarm2, or Space Station Remote Manipulator System (SSRMS), is a versatile 7-joint serial manipulator (Nokleby, 2007). Its distinguishing feature is the design that allows either end of the manipulator to act as the device's base. Both ends are equipped with Latching End Effectors (LEEs) that supply power and data to the manipulator. These LEEs attach to Power Data Grapple Fixtures (PDGFs) on the ISS. Using this mechanism, Canadarm2 can move around the ISS, providing unprecedented mobility and flexibility in operations (MDRobotics, n.d.).

Further adding to its capabilities, Canadarm2 is equipped with four cameras delivering wide and close-up views to the operators. These cameras are located on the booms and LEEs to provide comprehensive visibility during operations (MDRobotics, n.d.).

Dextre, also known as the Special Purpose Dexterous Manipulator (SPDM), is a twin-armed robot designed to handle intricate servicing and assembly tasks that previously required astronaut spacewalks. Operated by astronauts within the ISS, Dextre shares the unique mobility feature with Canadarm2, allowing it to move around the station's exterior (MDRobotics, n.d.).

A summary of Canadarm2 and Dextre specifications is provided in Table 2.1. Both manipulators are designed to stay in space for their entire operational life, emphasizing the importance of repairability. Thus, both have been built in easily removable sections called Orbital Replacement Units (ORUs) that can be replaced either by an astronaut or by Dextre itself (MDRobotics, n.d.).

Table 2.1: Specifications of Canadarm2 and Dextre (MDRobotics, n.d.)

Specification	Canadarm2	Dextre
Length (m)	17.6	3.5
Mass (kg)	1800	1662
Mass Handling Transportation Capacity (kg)	116,000	600
Degrees of Freedom	7	15

Having established the capabilities and specifications of Canadarm2 and Dextre, we can now

Figure 2.2: Canadarm2 operating on the International Space Station (CSA, 2020)

delve into the academic works that further illustrate the significance and potential of these robotic manipulators, specifically in the context of On-Orbit Servicing (OOS). OOS includes a range of tasks, such as satellite refueling, upgrading, and repairing. The application of robotic manipulators like Canadarm2 and Dextre in these tasks represents an active area of exploration and innovation.

Several noteworthy research contributions have emerged in this domain. For instance, Luo and Sakawa (1990) proposed a control law to synchronize the motion of a manipulator with a tumbling object, while Flores-Abad, Zhang, Wei, and Ma (2017) introduced an optimal capture strategy to minimize the impact on the servicing spacecraft. The academic realm has also expanded into autonomous operations, with Rekleitis et al. (2007) developing an autonomous capture and servicing framework that optimizes existing OOS tasks and paves the way for future applications of robotic manipulators in space.

The roles of robotic manipulators in Space Manipulator Systems (SMS) have broadened significantly beyond satellite servicing to include tasks like orbital debris removal and the maintenance of large orbital assets and infrastructures (Papadopoulos, Aghili, Ma, & Lampariello, 2021). The rise in space debris has amplified the risk of collisions, making debris removal an emergent application

Figure 2.3: Dextre mounted on Canadarm2 (MDA, 2023)

area for SMS. Aghili (2012) addressed a key challenge in these missions: capturing a tumbling, drifting space object with operational and environmental constraints. Using strategies such as state estimation, time-optimal trajectory planning, and momentum dampening, the study demonstrated how these new challenges could be tackled, thus extending the applicability of robotic manipulators in space.

2.2 Visual Servoing

Visual servoing is an advanced method employed in robotics, merging computer vision and camera systems' strengths to control a manipulator's end-effector pose. (Corke et al., 1996). This pose, demonstrated by a six-element vector, specifies the robot's position and orientation in a three-dimensional space, a crucial component that also holds significance in the context of mobile robots.

Prior to the development of visual servoing, robotics faced numerous challenges. Traditional robots were disadvantaged by their inability to 'see' their operational environments (Corke et al., 1996). This absence of sensory feedback resulted in significant engineering efforts to create highly structured workspaces, leading to high non-recurring engineering costs. This expense often limited

the versatility and adaptability of robotic systems (Corke et al., 1996).

Another significant issue lay in the robots' need for precise workspace knowledge, including the coordinates of workpieces or other objects. Without this, tasks were confined to being strictly repetitive, such as assembly with precise fixturing (Corke et al., 1996). Moreover, ensuring that a robot could accurately achieve a desired pose was often difficult due to assumptions regarding the accuracy of the robot's kinematic mode (Corke, 2017; Corke et al., 1996).

The development of visual servoing has addressed many of these challenges. Instead of relying on an expected pose of an object, visual servoing uses continuous visual monitoring and adjustment of the robot's movements according to the observed position of the object (Corke, 2017). This eliminates the need for prior knowledge of the object's pose and alleviates the requirement for a highly structured work environment. Moreover, visual servoing proves especially valuable in unstructured environments, where the exact location of the robot and workpiece are unknown and often impracticable to measure, such as in field service robotics and space robotics (Corke et al., 1996).

Furthermore, visual servoing allows for greater robustness against errors and can handle dynamic situations such as moving parts. It achieves this by continuously measuring the target and the robot, creating a feedback signal, and moving the robot until the visually observed error between the robot and the target is zero (Corke, 2017).

However, visual servoing is not without its own challenges. Practical complexities can arise from issues such as camera placement and focus. For instance, when a camera is mounted on the end of a robot, it might interfere with the task or be unable to focus when the robot is close to the target (Corke et al., 1996). Moreover, depth information is lost due to the 2D projection of the scene by the camera lens, necessitating additional information to determine the 3D coordinates corresponding to image plane points (Corke et al., 1996). Despite these complexities, visual servoing still significantly advances robotic control systems, offering enhanced versatility and adaptability in both structured and unstructured environments (Corke, 2017; Corke et al., 1996).

Visual servoing can be configured in two ways: end-point closed loop (or eye-in-hand) and end-point open-loop (or eye-to-hand). In the eye-in-hand configuration, the camera is mounted on the robot's end-effector observing the goal (Fig. 2.4,a) On the other hand, in the eye-to-hand

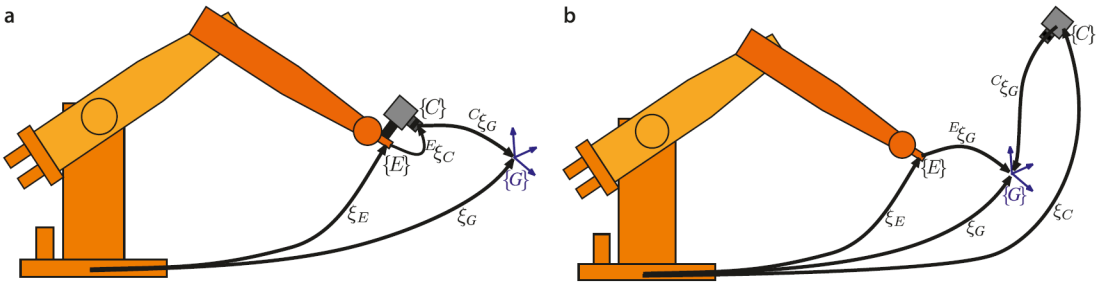


Figure 2.4: Visual servoing configurations a. Eye-in-hand configuration b. Eye-to-hand configuration (Corke, 2017)

configuration, the camera is at a fixed point in the world observing both the goal and the robot's end-effector (Fig. 2.4, b)(Corke, 2017)."

The fundamental objective of any vision-based control scheme, including visual servoing, is reducing an error term, denoted as $\mathbf{e}(t)$. This error, in a general case, is formulated as (Chaumette & Hutchinson, 2006):

$$\mathbf{e}(t) = \mathbf{s}(\mathbf{m}(t), \mathbf{a}) - \mathbf{s}^*. \tag{2.4}$$

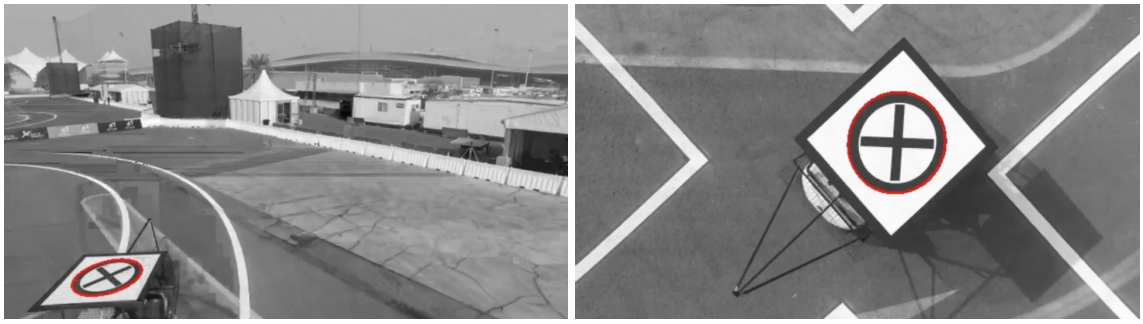
In Equation (2.4), the parameters can be characterized as follows. The vector $\mathbf{m}(t)$ constitutes a set of image metrics, which might include, for example, the image coordinates of particular interest points or the centroid of an object. These measurements are then used to derive a vector of k visual attributes, denoted as $\mathbf{s}(\mathbf{m}(t), \mathbf{a})$. Here, $\mathbf{a}$ indicates a group of parameters that contain additional system-related knowledge, potentially comprising camera intrinsic parameters or 3-D object models. The desired feature values are captured within the vector $\mathbf{s}^*$ (Chaumette & Hutchinson, 2006).

We consider here the case of a stationary target and a predetermined goal pose, implying that $\mathbf{s}^*$ is constant and any variations in $\mathbf{s}$ depend solely on the camera's motion. Furthermore, we focus on controlling a camera with six degrees of freedom (6 DOF), such as a camera affixed to the end effector of a six-degree-of-freedom arm (Chaumette & Hutchinson, 2006).

2.2.1 Applications of Visual Servoing

Visual servoing has found a wide range of applications in various fields, demonstrating its versatility and effectiveness. This section presents a few notable applications of visual servoing in different domains.

One of the most challenging applications of visual servoing is in the field of Unmanned Aerial Vehicles (UAVs). Keipour et al. (2022) proposed a visual servoing approach to enable a UAV to land on a moving vehicle autonomously. The authors developed a visual servoing controller that uses the image of the landing pad captured by a downward-facing camera on the UAV. The controller adjusts the UAV's position to align the landing pad's image with the desired image, allowing the UAV to track and land on the moving vehicle.



(a) (b)

Figure 2.5: Result of the deck detection on sample frames. The red ellipse shows the detected pattern on the moving vehicle (Keipour et al., 2022).

In the medical field, visual servoing has been used to enhance the precision and safety of surgical procedures. Wei, Arbter, and Hirzinger (1997) presented a real-time visual servoing system for laparoscopic surgery. The system uses color image segmentation to track surgical tools and tissue in real time. A visual servoing controller then uses this information to control the robot's motion, enabling precise tool positioning and movement.

Visual servoing has also been used to control unmanned ground vehicles (UGVs) using an airborne monocular camera. In the work by Mehtatt, Dixon, Mac Arthur, and Crane (2006), the authors proposed a visual servoing method that uses the image captured by a moving airborne camera to control a UGV. The method uses visual features extracted from the image to compute the

control input for the UGV, allowing it to follow a desired path.

Lastly, visual servoing has found applications in space robotics. Shi, Liang, Wang, Xu, and Liu (2012) proposed a visual servoing approach (Switching between Image-based Visual Servoing and Position-based Visual Servoing) for a space robot to capture a cooperative target. The authors developed a visual servoing controller that uses the target image captured by a binocular camera on the robot. The controller adjusts the robot's position and orientation to align the target's image with a desired image, enabling the robot to capture the target accurately.

In conclusion, visual servoing has proven to be a powerful tool in various applications, from UAVs and manufacturing to medical procedures and space robotics. Its ability to use visual information to control a system's motion makes it a versatile and effective solution for many control problems.

2.2.2 Classifications of Visual Servoing Systems

In visual servoing systems, the nature of the feature vector $\mathbf{s}$ is a key factor that differentiates various approaches. This crucial differentiation was initially brought to light by Sanderson and Weiss (1980), paving the way for the categorization of visual servoing structures into two principal types: Image-based Visual Servoing (IBVS) and Position-based Visual Servoing (PBVS). These two techniques are visually represented in Figures 2.6 and 2.7.

Position-based Visual Servoing (PBVS), the first of these two categories, designs its feature vector $\mathbf{s}$ around a set of 3-D parameters. These parameters are inferred from image data. In the PBVS approach, visual features are harnessed from the image and applied alongside a geometric model of the target. This facilitates the determination of the target's orientation with respect to the camera, as depicted in Figure 2.6. Subsequently, the robot aligns its movement to this pose, executing control in the task space (Corke, 2017). Despite the existence of robust algorithms for pose estimation, PBVS can prove to be a computationally expensive technique, requiring both precise camera calibration and an accurate model of the object's geometry.

On the other hand, An image feature $\mathbf{s}$, a scalar or vector quantity derived from the image, is used in IBVS. As such, it bypasses the need for pose estimation, carrying out control directly in the image coordinate space $\mathbb{R}^2$ (Figure 2.7). Typical selections for such features include the

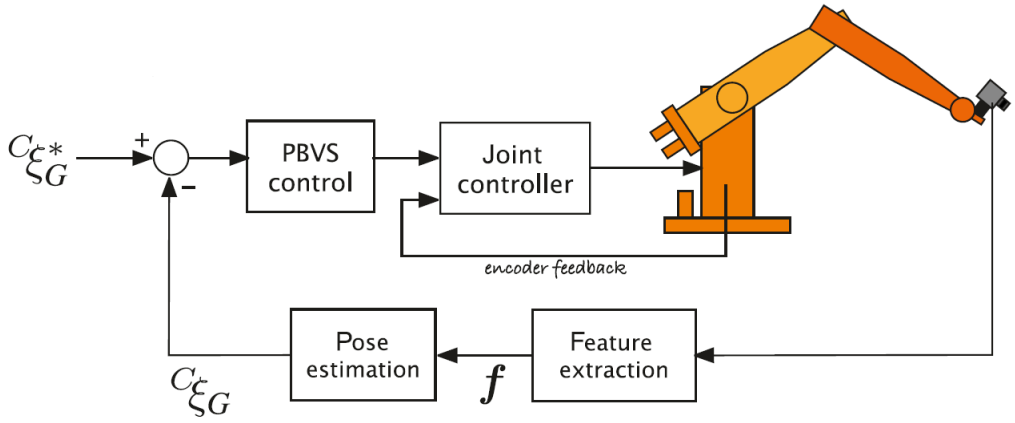


Figure 2.6: Position-based Visual Servoing block diagram (Corke, 2017)

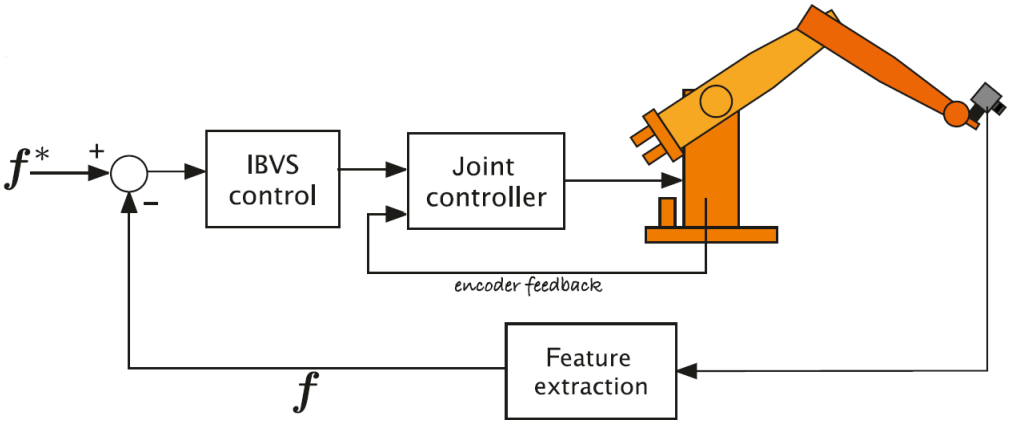


Figure 2.7: Image-based Visual Servoing block diagram (Corke, 2017)

coordinates of distinctive points or the centroid of specific regions in the image, such as a hole in a gasket (Corke et al., 1996; Feddema & Mitchell, 1989). Despite IBVS potentially reducing computational latency and eradicating the need for detailed image interpretation, along with errors linked to sensor modeling and camera calibration, it nonetheless presents a considerable challenge. This is due to the highly nonlinear relationship between the image features and the camera pose (Corke, 2017).

To overcome the limitations of classical Position-Based Visual Servoing (PBVS) and Image-Based Visual Servoing (IBVS), a novel approach named 2-1/2-D Visual Servoing has been proposed. This strategy, visually represented in Figure 2.8, offers unique advantages over its predecessors by avoiding the necessity for a geometric three-dimensional (3-D) model of the object, which

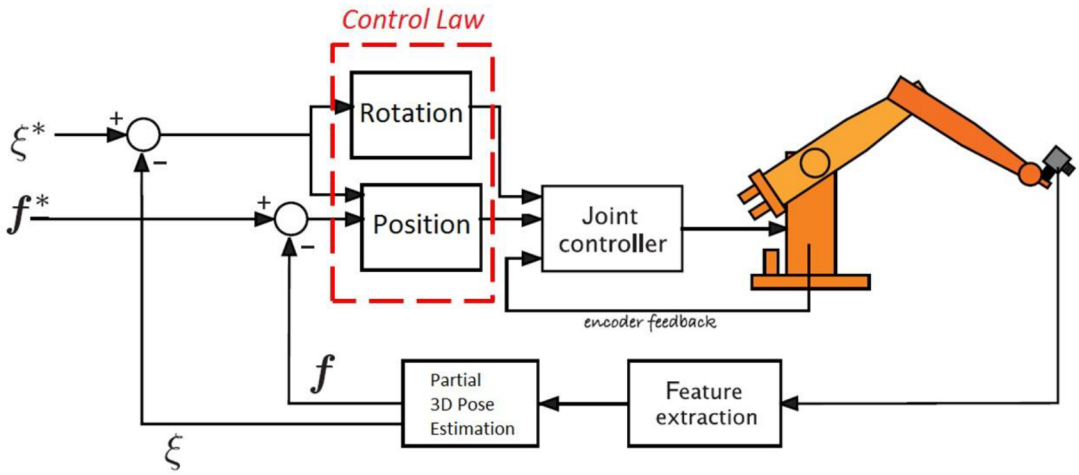


Figure 2.8: 2 1/2-D Visual Servoing block diagram (Mohebbi, 2013)

is a fundamental requirement in PBVS. In addition, unlike IBVS, it guarantees the convergence of the control law across the entire task space (Malis, Chaumette, & Boudet, 1999).

2-1/2-D Visual Servoing estimates the partial camera displacement from the current to the desired camera poses at each control law iteration. This iterative process forms a decoupled control law to manage the camera's six degrees of freedom, which includes both rotational and translational movements. In this hybrid approach, input is partially expressed in the 3-D Cartesian space and partially in the 2-D image space, which is why it's termed 2-1/2-D. This lack of reliance on a 3-D model of the target widens the versatility and application area of visual servoing (Malis et al., 1999).

The rotational control loop is computed at each iteration, while the control of the translational camera degrees of freedom necessitates the introduction of extended image coordinates of a reference point on the target. This leads to a triangular interaction matrix with excellent decoupling properties and importantly, no singularity across the whole task space (Malis et al., 1999).

2.2.3 Classical Image-based Visual Servoing

Once the feature set $\mathbf{s}$ is chosen, the controller design can be simplified significantly. An intuitive strategy is to design a velocity controller. For this, the relationship between the time variation of $\mathbf{s}$ and the camera velocity is required. Let the spatial velocity of the camera be denoted by $\mathbf{v}_c = (v_c, w_c)$, with v_c representing the instantaneous linear velocity of the camera frame's origin, and w_c being the instantaneous angular velocity of the camera frame. The relationship between $\dot{\mathbf{s}}$

and $\mathbf{v}_c$ is given by (Chaumette & Hutchinson, 2006):

$$\dot{\mathbf{s}} = \mathbf{L}_\mathbf{s}\mathbf{v}_c. \tag{2.5}$$

The matrix $\mathbf{L}_\mathbf{s}$ of dimensions $\mathbb{R}^{k\times 6}$ is referred to as the interaction matrix associated with the feature vector $\mathbf{s}$ (Chaumette & Hutchinson, 2006). From equations (2.4) and (2.5), we can directly establish a relationship between the camera velocity and the change over time of the error, assuming that $\mathbf{s}^*$ is constant. This relationship is expressed as:

$$\dot{\mathbf{e}} = \mathbf{L}_\mathbf{e}\mathbf{v}_c, \tag{2.6}$$

where $\mathbf{L}_\mathbf{e}$ is equal to $\mathbf{L}_\mathbf{s}$ (Chaumette & Hutchinson, 2006).

In controlling the camera velocity $\mathbf{v}_c$, which serves as input to the robot controller, we assume an exponential decay to minimize the error. The rate of change of error $\dot{\mathbf{e}}$ is defined by $\dot{\mathbf{e}} = -\lambda\mathbf{e}$, a differential equation derived from the expression of exponential decay, $\mathbf{e}(t) = \mathbf{e}(0)e^{-\lambda t}$. This equation models the error $\mathbf{e}(t)$ diminishing exponentially over time towards zero, driven by the factor λ, a positive scalar constant. Hence, the error is expected to progressively decrease through this formulation, eventually leading the system towards the desired state (Chaumette & Hutchinson, 2006). Applying this principle to Equation (2.6), we obtain:

$$\mathbf{v}_c = -\lambda\mathbf{L}_\mathbf{e}^+\mathbf{e}. \tag{2.7}$$

In this equation, $\mathbf{L}_\mathbf{e}^+$, which is a $\mathbb{R}^{6\times k}$ matrix, is designated as the Moore-Penrose pseudoinverse of $\mathbf{L}_\mathbf{e}$. This pseudoinverse is given by $\mathbf{L}_\mathbf{e}^+ = (\mathbf{L}_\mathbf{e}^T\mathbf{L}_\mathbf{e})^{-1}\mathbf{L}_\mathbf{e}^T$ and is employed when $\mathbf{L}_\mathbf{e}$ is of full rank 6 (Chaumette & Hutchinson, 2006).

This formulation in Equation (2.7) allows the controller to compute the required camera velocity to minimize the error $\mathbf{e}$, thereby guiding the robot towards the desired pose.

Various controller designs have been explored in IBVS, including linear controllers such as the PD controller highlighted by Keshmiri, Xie, and Mohebbi (2014), and the fuzzy adaptive PID controller presented by Dong, Hu, and Peng (2012). Nonlinear types are also common, including the

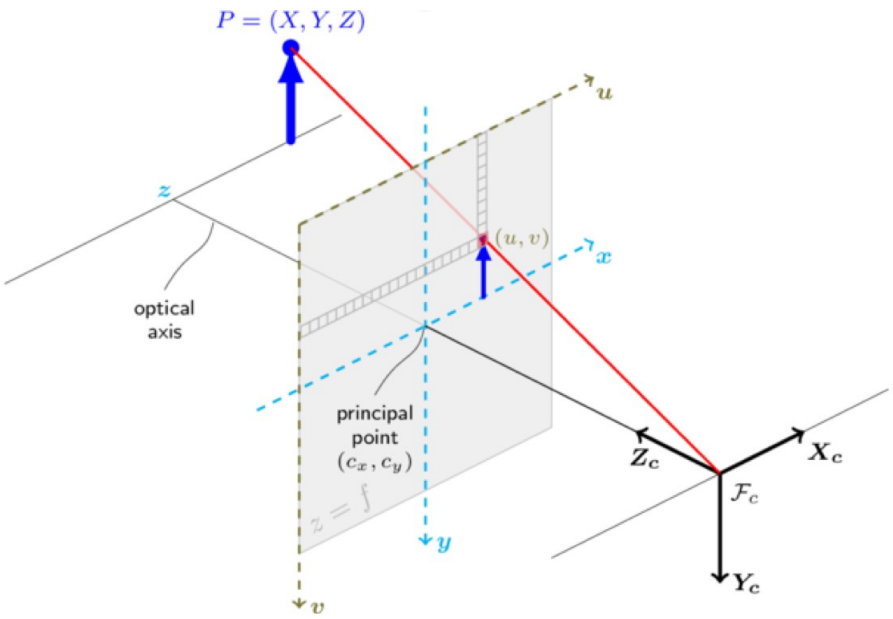


Figure 2.9: Camera 3-D to 2-D projection schematic (Kurnaz, 2019)

sliding mode controller of Kim, Kim, Choi, and Won (2006) and the adaptive nonlinear controller of Fang, Liu, and Zhang (2011). Moreover, innovative hybrid models that combine PD and sliding mode, as proposed by Li, Xie, and Gao (2017), further show the diverse types of controllers applied to the IBVS.

The process of image-based visual servoing can be understood by examining the relationship between 3D points in the camera frame and their corresponding 2D projections in the image plane.

For a point $\mathbf{X} = (X, Y, Z)$ in the 3D camera frame that projects onto the image plane as a 2D point $\mathbf{x} = (x, y)$, we use the following equations (Chaumette & Hutchinson, 2006) (derived from simple triangulation):

$$\begin{cases} x = X/Z = (u - c_x)/f\alpha \\ y = Y/Z = (v - c_y)/f \end{cases} . \tag{2.8}$$

In this context, $\mathbf{m} = (u, v)$ indicates the pixel coordinates of the image point, and $\mathbf{a} = (c_x, c_y, f, \alpha)$ represents the camera intrinsic parameters, where c_x and c_y are the coordinates of the principal point, f denotes the focal length, and α is the ratio of pixel dimensions. (Chaumette & Hutchinson, 2006) In this particular setup, we define the visual features $\mathbf{s}$ as the image plane

coordinates of the point, i.e., $\mathbf{s} = \mathbf{x} = (x, y)$.

If we differentiate the projection Equations 2.8 with respect to time, we acquire (Chaumette & Hutchinson, 2006):

$$\begin{cases} \dot{x} = \dot{X}/Z - X\dot{Z}/Z^2 = (\dot{X} - x\dot{Z})/Z \\ \dot{y} = \dot{Y}/Z - Y\dot{Z}/Z^2 = (\dot{Y} - y\dot{Z})/Z \end{cases} . \tag{2.9}$$

Now, we can relate the 3-D velocity of the point to the spatial velocity of the camera $\mathbf{v}_c$ with the well-known equation (Chaumette & Hutchinson, 2006):

$$\dot{\mathbf{X}} = -\mathbf{v}_c - w_c \times \mathbf{X} \Longleftrightarrow \begin{cases} \dot{X} = -v_x - w_y Z + w_z Y \\ \dot{Y} = -v_y - w_z X + w_x Z \\ \dot{Z} = -v_z - w_x Y + w_y X \end{cases} . \tag{2.10}$$

Substituting Equation (2.10) into Equation (2.9) and rearranging the terms, we get (Chaumette & Hutchinson, 2006):

$$\begin{cases} \dot{x} = -v_x/Z + xv_z/Z + xyw_x - (1 + x^2)w_y + yw_z \\ \dot{y} = -v_y/Z + yv_z/Z + (1 + y^2)w_x - xyw_y - xw_z \end{cases} . \tag{2.11}$$

This can be compactly written as (Chaumette & Hutchinson, 2006):

$$\dot{\mathbf{x}} = \mathbf{L}_\mathbf{x}\mathbf{v}_c. \tag{2.12}$$

Here, $\mathbf{L}_\mathbf{x}$ is the interaction matrix associated with $\mathbf{x}$ (Chaumette & Hutchinson, 2006):

$$\mathbf{L}_\mathbf{x} = \begin{bmatrix} -\frac{1}{Z} & 0 & \frac{x}{Z} & xy & -(1 + x^2) & y \\ 0 & -\frac{1}{Z} & \frac{y}{Z} & 1 + y^2 & -xy & -x \end{bmatrix} . \tag{2.13}$$

For a 6 DOF robot, a minimum of six feature points is necessary, or equivalently, three distinct points assuming the chosen features correspond to these points' x and y coordinates. This requirement can be represented as needing at least $k \geq 6$ feature points (Chaumette & Hutchinson,

2006).

Assuming the feature vector $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$, each component representing a distinct point, we can express the overall interaction matrix for these three points by simply stacking the interaction matrices of each individual point. This gives us the following structure (Chaumette & Hutchinson, 2006):

$$\mathbf{L}_{\mathbf{x}} = \begin{bmatrix} \mathbf{L}_{\mathbf{x}_1} \\ \mathbf{L}_{\mathbf{x}_2} \\ \mathbf{L}_{\mathbf{x}_3} \end{bmatrix}. \tag{2.14}$$

Here, $\mathbf{L}_{\mathbf{x}_i}$ represents the interaction matrix corresponding to the feature point $\mathbf{x}_i$. This equation concisely represents the combined interaction matrices for three distinct points in the visual field.

2.3 Image-based Visual Servoing using Image Moments

Visual Servoing is a crucial control method within the realm of robotics, where the control law directly exploits data obtained from vision sensors. Among its various strategies, Image-based Visual Servoing (IBVS) has seen significant traction due to its ability to accommodate more complex scenarios, where it commands the robot based on the error in image features instead of the error in the pose. This study relies upon the concept of Image Moments (a mathematical tool that provides critical information about an image) within the framework of IBVS.

Image Moments have proved to be invaluable in numerous image processing tasks, primarily due to their easy computation from binary, segmented images, or from a collection of selected points of interest. The primary advantage of low-order Image Moments is their inherent association with the object's physical properties in the image, such as area, centroid, inertial moments, and orientation (Tahri & Chaumette, 2005).

This intuitive understanding of Image Moments and their invariance to certain transformations, including scale, 2-D translation, and/or 2-D rotation, has encouraged extensive research in moment invariants. Such properties are beneficial for pattern recognition and also for visual servoing (Tahri & Chaumette, 2005).

2.3.1 Image Moments Definition

The object under observation is represented as O, and the image captured by the camera at time t is defined as $\pi(t)$. The projection of the object on the image at time t is denoted by $R(t)$, while the contour of this projection is represented by $C(t)$ (Figure 2.10). Instead of considering individual pixels' intensity, we assume that we either obtain binary images directly or apply a spatial segmentation algorithm on the captured images to produce binary images (Chaumette, 2004). This implies a simplification of the image data to the basic form, allowing us to focus on the shape and positioning of objects within the image rather than the detailed pixel intensity values. This approach simplifies the computation and allows for a higher level of invariance to changes in lighting conditions or object texture.

The 2-D moments m_{ij} of order $i + j$ of a dense object O in an image can be mathematically defined by (Tahri & Chaumette, 2005)

$$m_{ij}(t) = \int\int_{R(t)} f(x, y)dxdy, \tag{2.15}$$

where $f(x, y) = x^i y^j$. Separately, the object centroid (x_g, y_g) is used to compute the centered moments μ_{ij}. They are defined by:

$$\mu_{ij}(t) = \int\int_{R(t)} (x - x_g)^i (y - y_g)^j dxdy. \tag{2.16}$$

Here, $x_g = m_{10}/m_{00}$ and $y_g = m_{01}/m_{00}$ denote the centroid of the object in the x and y directions, respectively. The Image Moments can also be computed for a discrete set of n image pixels as (Tahri & Chaumette, 2005):

$$m_{ij} = \sum_{k=1}^{n} x_k^i y_k^j. \tag{2.17}$$

The corresponding centered moments are given by:

$$\mu_{ij} = \sum_{k=1}^{n} (x_k - x_g)^i (y_k - y_g)^j. \tag{2.18}$$

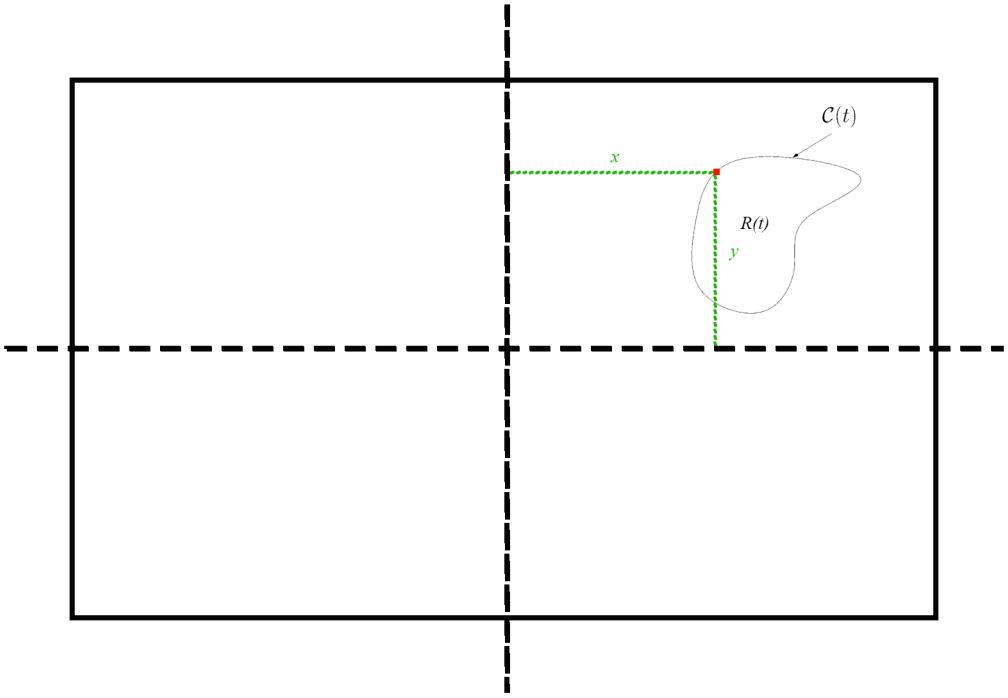


Figure 2.10: x & y coordinate of a pixel from R(t) in image plane

2.3.2 Interaction Matrix of Image Moments

Remarkably, these moments—whether continuous or discrete—are known to be invariant to 2-D translational motion. This fact has led to several works in the literature presenting methods to derive moment invariants to other transformations such as scale and 2-D rotation (Tahri & Chaumette, 2005).

We aim to determine the analytical form which describes the time variation of the moment m_{ij}, represented as $\dot{m}_{ij}$, as a function of the relative kinematic screw $\mathbf{v} = (\mathbf{v}, \mathbf{w})$ between the camera and the object, where $\mathbf{v}$ and $\mathbf{w}$ represent the translational and rotational velocity components, respectively. Similar to classical geometrical features, we obtain a linear link that can be expressed in the form (Chaumette, 2004)

$$\dot{m}_{ij} = \mathbf{L}_{m_{ij}} \mathbf{v}, \tag{2.19}$$

where $\mathbf{L}_{m_{ij}}$ is the interaction matrix related to m_{ij}. This interaction matrix serves as the bridge between the camera's movement (velocity) and the change in image moments, allowing us to directly

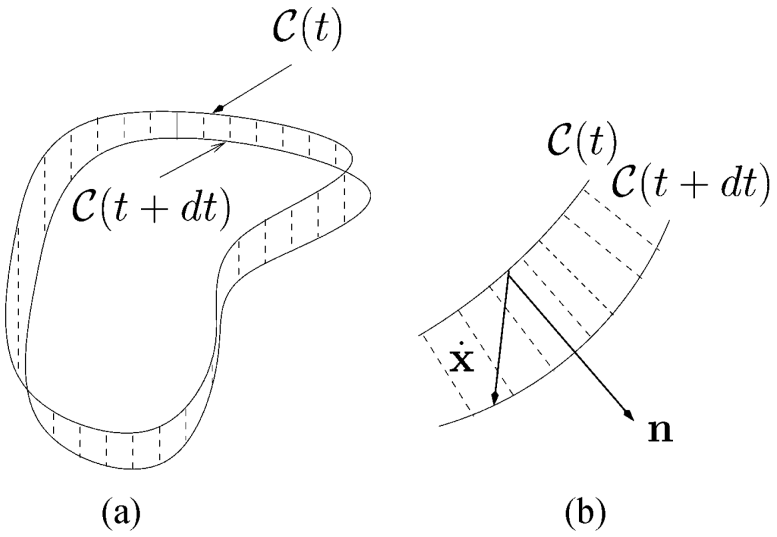


Figure 2.11: (a) Time variation of contour C(t). (b) A more detailed view (Chaumette, 2004).

control the camera based on the observed changes in image moments.

In Equation (2.15), the only time-dependent variable is $R(t)$. Consequently, the change in m_{ij} over time can be obtained from the fluctuation of $C(t)$. To elaborate further on this, the time variation of m_{ij} can be expressed as (Chaumette, 2004):

$$\dot{m}_{ij} = \oint_{C(t)} f(x,y)\dot{\mathbf{x}}^T \mathbf{n} dl. \tag{2.20}$$

In this equation, $\dot{\mathbf{x}}$ signifies the velocity of the point $\mathbf{x} = (x, y)$ located on the contour. The unitary vector, $\mathbf{n}$, is perpendicular to $C(t)$ at the point $\mathbf{x}$, and dl represents an infinitesimally small segment of the contour $C(t)$ (Chaumette, 2004).

Visualizing this from a geometrical standpoint (Figure 2.11), we can infer that the change in m_{ij} is determined by evaluating m_{ij} over the infinitesimal area that exists between $C(t)$ and $C(t + dt)$. Essentially, this calculation involves integration along $C(t)$ where the product of (x, y) (used to compute the moment) is multiplied by the dot product of $\dot{\mathbf{x}}$ and $\mathbf{n}$, ensuring the progression of $C(t)$ to $C(t + dt)$ (Chaumette, 2004).

We can use Green's Theorem if the following conditions are satisfied (Chaumette, 2004):

(1) The contour $C(t)$ is piecewise continuous.

(2) The vector function $f(x,y)\dot{\mathbf{x}}$ is tangent to $R(t)$ and is continuously differentiable, $\forall \mathbf{x} \in R(t)$.

This powerful result from vector calculus transforms our line integral, which is along the object contour $C(t)$, into a double integral over the area $R(t)$ covered by the object in the image plane.

The motivation for this transformation arises from the practical consideration of computations in visual servoing. In the image processing context, it is typically easier and more computationally efficient to perform operations over the area of an object rather than along its contour. The double integral form is more tractable because it allows for computations over the segmented or binary image regions, which we usually have as a result of image processing algorithms.

Applying Green's theorem to our case, Equation (2.20) can be rewritten as (Chaumette, 2004):

$$\dot{m}_{ij} = \int\int_{R(t)} div[f(x,y)\dot{\mathbf{x}}]dxdy. \tag{2.21}$$

We can further simplify this expression by expanding the divergence operator. This results in (Chaumette, 2004):

$$\dot{m}_{ij} = \int\int_{R(t)} [\frac{\partial f}{\partial x}\dot{x} + \frac{\partial f}{\partial y}\dot{y} + f(x,y)(\frac{\partial \dot{x}}{\partial x} + \frac{\partial \dot{y}}{\partial y})]dxdy. \tag{2.22}$$

Equation (2.22) provides a more detailed insight into the time variation of the image moments, accounting for the gradient of $f(x,y)$ as well as the divergence of the velocity field $\dot{\mathbf{x}}$. This mathematical transformation further consolidates the link between the image moments and the camera motion in image-based visual servoing.

By the definition of image moments $f(x,y) = x^i y^j$. The partial derivatives of $f(x,y)$ with respect to x and y can be easily computed as follows (Chaumette, 2004):

$$\frac{\partial f}{\partial x} = i x^{i-1} y^j, \tag{2.23}$$

and

$$\frac{\partial f}{\partial y} = j x^i y^{j-1}. \tag{2.24}$$

Substituting these results into Equation (2.22), we obtain:

$$\dot{m}_{ij} = \int\int_{R(t)} [ix^{i-1}y^j\dot{x} + jx^iy^{j-1}\dot{y} + x^iy^j(\frac{\partial \dot{x}}{\partial x} + \frac{\partial \dot{y}}{\partial y})]dxdy. \tag{2.25}$$

As a result, we are interested in determining the equations for $\dot{x}$ and $\dot{y}$ to compute the interaction matrix $\mathbf{L}_{m_{ij}}$. The point $\mathbf{x} = (x, y)$ in the image plane is assumed to have a depth of Z from the camera. The velocity of the image point $\dot{\mathbf{x}}$ is linked to the camera velocity $\mathbf{v}$ through the well-known relationship (Chaumette, 2004):

$$\dot{\mathbf{x}} = \mathbf{L}_{\mathbf{x}}\mathbf{v}. \tag{2.26}$$

Here, $\mathbf{L}_{\mathbf{x}}$ is the interaction matrix of a point $\mathbf{x}$, given by (Chaumette, 2004):

$$\mathbf{L}_{\mathbf{x}} = \begin{bmatrix} -\frac{1}{Z} & 0 & \frac{x}{Z} & xy & -(1+x^2) & y \\ 0 & -\frac{1}{Z} & \frac{y}{Z} & 1+y^2 & -xy & -x \end{bmatrix}. \tag{2.27}$$

We assume that the object exists on a continuous surface, implying that the depth Z varies smoothly across the object's extent without any sudden jumps or discontinuities. This assumption allows us to express the depth of any point on the object as a smooth function of its image coordinates x and y as follows (Chaumette, 2004):

$$\frac{1}{Z} = \sum_{p\geq 0, q\geq 0} A_{pq}x^py^q. \tag{2.28}$$

This equation allows us to represent the depth of a point on the object (expressed as $1/Z$ for computational convenience) as a function of its image coordinates (x, y). It's expressed as a polynomial series, accommodating various depth variations across the object.

A planar object is a special case that often arises in practice. If the equation of the plane in the camera frame is given by $Z = \gamma_1 X + \gamma_2 Y + Z_0$, then the depth function simplifies to (Chaumette, 2004):

$$\frac{1}{Z} = Ax + By + C, \tag{2.29}$$

where $A = -\frac{\gamma_1}{Z_0}$, $B = -\frac{\gamma_2}{Z_0}$, and $C = \frac{1}{Z_0}$.

Using 2.29 in 2.27, 2.26 can be written as (Chaumette, 2004):

$$\begin{cases} \dot{x} = -(Ax + By + C) \cdot v_x + x(Ax + By + C) \cdot v_z + xy \cdot w_x - (1 + x^2) \cdot w_y + y \cdot w_z \\ \dot{y} = -(Ax + By + C) \cdot v_y + y(Ax + By + C) \cdot v_z + (1 + y^2) \cdot w_x - xy \cdot w_y - x \cdot w_z \end{cases}, \tag{2.30}$$

from which we can obtain (Chaumette, 2004):

$$\begin{cases} \frac{\partial \dot{x}}{\partial x} = -Av_x + (2Ax + By + C)v_z + yw_x - 2xw_y \\ \frac{\partial \dot{y}}{\partial y} = -Bv_y + (Ax + 2By + C)v_z + 2yw_x - xw_y \end{cases}. \tag{2.31}$$

We can rearrange Equation (2.25) by plugging in Equations (2.30) and (2.31), and express it in the desired format given in Equation (2.19). With a bit of manipulation, we obtain (Chaumette, 2004):

$$\mathbf{L}_{m_{ij}} = [m_{vx}, m_{vy}, m_{vz}, m_{wx}, m_{wy}, m_{wz}], \tag{2.32}$$

where

$$\begin{cases} m_{vx} = -i(Am_{ij} + Bm_{i-1,j+1} + Cm_{i-1,j}) - Am_{ij} \\ m_{vy} = -j(Am_{i+1,j-1} + Bm_{ij} + Cm_{i,j-1}) - Bm_{ij} \\ m_{vz} = (i + j + 3)(Am_{i+1,j} + Bm_{i,j+1} + Cm_{ij}) - Cm_{ij} \\ m_{wx} = (i + j + 3)m_{i,j+1} + jm_{i,j-1} \\ m_{wy} = -(i + j + 3)m_{i+1,j} - im_{i-1,j} \\ m_{wz} = im_{i-1,j+1} - jm_{i+1,j-1} \end{cases}. \tag{2.33}$$

This implies that the interaction matrix $\mathbf{L}_{m_{ij}}$ can be represented as a six-dimensional vector with components related to the linear velocities (m_{vx}, m_{vy}, m_{vz}) and angular velocities (m_{wx}, m_{wy}, m_{wz}) in the camera frame.

Following a similar development as before, but this time for the central moments, we can derive

an analogous form for the interaction matrix of central moments. This yields (Chaumette, 2004):

$$\mathbf{L}_{\mu_{ij}} = [\mu_{vx}, \mu_{vy}, \mu_{vz}, \mu_{wx}, \mu_{wy}, \mu_{wz}], \tag{2.34}$$

where each component of $\mathbf{L}_{\mu_{ij}}$ corresponds to the contributions from the linear and angular velocities in the camera frame. The expressions for these components are given by (Chaumette, 2004):

$$\begin{cases} \mu_{vx} = -(i+1)A\mu_{ij} - iB\mu_{i-1,j+1} \\ \mu_{vy} = -jA\mu_{i+1,j-1} - (j+1)B\mu_{ij} \\ \mu_{vz} = -A\mu_{wy} + B\mu_{wx} + (i+j+2)C\mu_{ij} \\ \mu_{wx} = (i+j+3)\mu + ix_g\mu_{i-1,j+1} + (i+2j+3)y_g\mu_{ij} - 4in_{11}\mu_{i-1,j} - 4jn_{02}\mu_{i,j-1} \\ \mu_{wy} = -(i+j+3)\mu_{i+1,j} - (2i+j+3)x_g\mu_{ij} - jy_g\mu_{i+1,j-1} + 4in_{20}\mu_{i-1,j} + 4jn_{11}\mu_{i,j-1} \\ \mu_{wz} = i\mu_{i-1,j+1} - j\mu_{i+1,j-1} \end{cases}, \tag{2.35}$$

where n_{ij} is the normalized central moment, defined as:

$$n_{ij} = \mu_{ij}/m_{00}. \tag{2.36}$$

2.3.3 Image Features based on Image Moments

In visual servoing, a critical aspect is the selection of effective image features for controlling the robot's degrees of freedom (DOF). Specifically, these image features are algebraic expressions calculated from image moments. A set of these features, typically six for a 6 DOF robotic system, are used to form an interaction matrix, which plays a crucial role in the control algorithm.

An ideal image feature directly associates each DOF with a single visual feature (Tahri & Chaumette, 2005), leading to minimal interference among different DOFs. The ideal scenario involves choosing image features that correspond uniquely to each DOF in a way that the interaction matrix becomes a 6x6 matrix with only one non-zero element per row.

However, the reality of achieving such ideal features is complex. A noted example in the literature, (Chaumette, 2004), indicates the difficulty of controlling all six DOFs using only six moments of order less than three. Therefore, moments of a higher order often need to be incorporated into the selection of visual features. Moreover, as pointed out by Tahri and Chaumette (2005), an ideal interaction matrix, ideally the identity matrix, is probably impossible to achieve due to the inherent nonlinearities in the relation between the change of feature time and camera velocities (Equation (2.5)). Nonetheless, the selection of image features in a way that minimizes these nonlinearities remains an important consideration in visual servoing.

For the translational movements, several image features have conventionally been utilized. These include the object's centroid coordinates $x_g = \frac{m_{10}}{m_{00}}$ and $y_g = \frac{m_{01}}{m_{00}}$, and its area $a = m_{00}$. When the object is parallel to the image plane, the following interaction matrices can be obtained:

$$\begin{aligned}
\mathbf{L}_{x_g}^{||} &= \begin{bmatrix} -C & 0 & Cx_g & \epsilon_1 & -(1+\epsilon_2) & y_g \end{bmatrix} \\
\mathbf{L}_{y_g}^{||} &= \begin{bmatrix} 0 & -C & Cy_g & 1+\epsilon_3 & -\epsilon_1 & -x_g \end{bmatrix}, \\
\mathbf{L}_{a}^{||} &= \begin{bmatrix} 0 & 0 & 2aC & 3ay_g & -3ax_g & 0 \end{bmatrix}
\end{aligned} \tag{2.37}$$

where $\epsilon_1 = x_g y_g + 4n_{11}$, $\epsilon_2 = x_g^2 + 4n_{20}$, and $\epsilon_3 = y_g^2 + 4n_{02}$. It can be seen from their interaction matrices that there is still an issue of coupling between these features.

In order to enhance the performance and reduce the coupling effect, an improvement can be made by introducing normalization to these features. Specifically, Tahri and Chaumette (2005) defined $a_n = Z^* \sqrt{\frac{a^*}{a}}$, $x_n = a_n x_g$, and $y_n = a_n y_g$, where a^* denotes the desired area of the object in the image, and Z^* represents the desired depth between the camera and the object. Similarly, for an object parallel to the image plane, the following normalized interaction matrices can be obtained:

$$\begin{aligned}
\mathbf{L}_{x_n}^{||} &= \begin{bmatrix} -1 & 0 & 0 & a_n\epsilon_{11} & -a_n(1+\epsilon_{12}) & y_n \end{bmatrix} \\
\mathbf{L}_{y_n}^{||} &= \begin{bmatrix} 0 & -1 & 0 & a_n(1+\epsilon_{21}) & -a_n\epsilon_{22} & -x_n \end{bmatrix}, \\
\mathbf{L}_{a_n}^{||} &= \begin{bmatrix} 0 & 0 & -1 & -a_n\epsilon_{31} & a_n\epsilon_{32} & 0 \end{bmatrix}
\end{aligned} \tag{2.38}$$

with $\epsilon_{11} = \epsilon_{22} = 4n_{11} - x_g y_g/2$, $\epsilon_{12} = 4n_{20} - x_g^2/2$, $\epsilon_{21} = 4n_{02} - y_g^2/2$, $\epsilon_{31} = 3y_g/2$, and

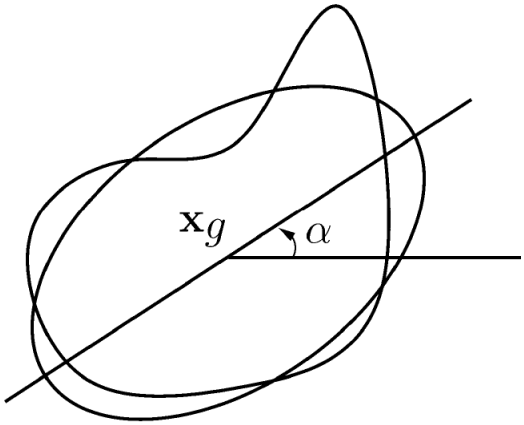


Figure 2.12: Orientation α of an object (Chaumette, 2004).

$\epsilon_{32} = 3x_g/2$.

The application of this normalization technique facilitates a better partitioning of the selected features to the translational degrees of freedom, thereby reducing coupling. However, while these features decoupled the translational elements, it is worth noting that achieving perfect decoupling for the translational degrees of freedom remains a challenging task.

As for rotational movements, let's focus first on the rotation about the camera's z-axis. The image feature α, representing the object's orientation is commonly used for this motion which is invariant to any translational motion when the object is parallel to the image plane (Chaumette, 2004).

A visual depiction of the orientation α of an object is presented in Figure 2.12, and the mathematical representation of α in terms of the central moments of order less than three is given by (Chaumette, 2004):

$$\alpha = \frac{1}{2}\arctan\left(\frac{2\mu_{11}}{\mu_{20} - \mu_{02}}\right). \tag{2.39}$$

The corresponding interaction matrix $\mathbf{L}_\alpha$ is then given by:

$$\mathbf{L}_\alpha = \frac{(\mu_{20} - \mu_{02})\mathbf{L}_{\mu_{11}} - \mu_{11}(\mathbf{L}_{\mu_{20}} - \mathbf{L}_{\mu_{02}})}{(\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2}. \tag{2.40}$$

When the object is parallel to the image plane, the interaction matrix $\mathbf{L}_\alpha$ simplifies to:

$$\mathbf{L}_{\alpha}^{||} = \begin{bmatrix} 0 & 0 & 0 & \alpha_{wx} & \alpha_{wy} & -1 \end{bmatrix}, \tag{2.41}$$

with:

$$\begin{cases} \alpha_{wx} = -bx_g + ay_g + d \\ \alpha_{wy} = ax_g - cy_g + e \end{cases} \quad \begin{cases} a = \mu_{11}(\mu_{20} + \mu_{02})/\Delta \\ b = [2\mu_{11}^2 + \mu_{02}(\mu_{02} - \mu_{20})]/\Delta \\ c = [2\mu_{11}^2 + \mu_{20}(\mu_{20} - \mu_{02})]/\Delta \\ d = 5[\mu_{12}(\mu_{20} - \mu_{02}) + \mu_{11}(\mu_{03} - \mu_{21})]/\Delta \\ e = 5[\mu_{21}(\mu_{02} - \mu_{20}) + \mu_{11}(\mu_{30} - \mu_{12})]/\Delta \end{cases}, \tag{2.42}$$

where $\Delta = (\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2$

The feature α provides a direct link to the rotational motion w_z around the optical axis and in practice, provides satisfactory performance. However, the interaction matrix still falls short of the ideal scenario.

In 1962, Hu (1962) developed a series of seven formulas. These formulas are based on central moments and remain constant under changes in position (translation) and rotation around the camera's optical axis, given the object stays parallel to the image plane. Three of these moments can be represented as:

$$\begin{cases} I_1 = c_1^2 + s_1^2 \\ I_2 = c_2^2 + s_2^2 \\ I_3 = \mu_{20} + \mu_{02} \end{cases}, \tag{2.43}$$

where:

$$\begin{cases} c_1 = \mu_{20} - \mu_{02} \\ c_2 = \mu_{03} - 3\mu_{21} \\ s_1 = 2\mu_{11} \\ s_2 = \mu_{30} - 3\mu_{12} \end{cases}. \tag{2.44}$$

Building on Hu's invariants, Chaumette (2004) proposed two new features that are also invariant to changes in size (scale). They are as follows:

$$\begin{cases} P_x = I_1/I_3^2 \\ P_y = \mu_{00} I_2/I_3^3 \end{cases}. \tag{2.45}$$

So when the object is parallel to the image plane, the interaction matrix for any of these two features, denoted as $\mathbf{L}_P^{||}$, has the desired form (Chaumette, 2004):

$$\mathbf{L}_P^{||} = \begin{bmatrix} 0 & 0 & 0 & l_{wx} & l_{wy} & 0 \end{bmatrix}. \tag{2.46}$$

However, a notable concern arises here. The values of l_{wx} and l_{wy} (which are too complex to be illustrated here) become zero when viewing a symmetrical object centered in the image. This issue is common for all of Hu's invariants as well (Chaumette, 2004). For this reason, when dealing with symmetrical objects, selecting a different set of invariants is essential.

Therefore, to address the limitation, Chaumette (2004) proposed two new features, particularly for symmetrical objects (with $\mu_{30} = \mu_{03} = \mu_{21} = \mu_{12} = 0$). These two features are:

$$\begin{cases} s_x = (c_2 c_3 + s_2 s_3)/K \\ s_y = (s_2 c_3 - c_2 s_3)/K \end{cases}. \tag{2.47}$$

The parameters c_3, s_3, and K are defined as follows:

$$\begin{cases} c_3 = c_1^2 - s_1^2 \\ s_3 = 2 s_1 c_1 \\ K = I_1 I_3^{(3/2)} / \sqrt{\mu_{00}} \end{cases} . \tag{2.48}$$

These features were chosen due to their invariance to 2D rotations and scaling, and the first feature, s_x, is as independent from w_y as possible, while s_y is similarly decoupled from w_x. For situations where the object is parallel to the image plane, the interaction matrix $\mathbf{L}^{||}$ takes the same desirable form as Equation (2.46) (Chaumette, 2004).

However, for non-symmetrical objects, s_x and s_y lose their invariance to 2D rotations as they are not combinations of Hu's invariants. This leads to an interaction matrix of the form:

$$\mathbf{L}_s = \begin{bmatrix} 0 & 0 & 0 & l_{wx} & l_{wy} & l_{wz} \end{bmatrix} . \tag{2.49}$$

Hence, it is suggested to utilize s_x and s_y only for symmetrical objects and P_x and P_y for all other cases (Chaumette, 2004).

In search of more practical features for capturing the rotational motions around the x and y axes of the camera, S. Liu et al. (2009) innovated a pair of features that displayed enhanced performance. The proposed features are presented as follows:

$$\begin{cases} s_x = 0.1 - (c_1 c_2 + s_1 s_2) / I_3^{(9/4)} \\ s_y = (s_1 c_2 - c_1 s_2) / I_3^{(9/4)} \end{cases} . \tag{2.50}$$

Here, the features are derived from the moments c_1, c_2, s_1, s_2, and I_3, showing how these variables can be creatively recombined to create new, effective features for rotational motion about the x and y axes. The interaction matrices for these two features are in the same form as Equation (2.49). However, these features are primarily suitable for small-sized objects. This is due to the fact that for larger objects, the magnitude of I_3 becomes so large that it diminishes the values of s_x and s_y to levels not suitable for visual servoing tasks. Instead of I_3, the term $\mu_{20} - \mu_{02}$ is used when dealing with larger objects.

It should also be mentioned that the interaction matrices (Eqs. 2.37, 2.38, 2.41, 2.46, and 2.49) corresponding to all of the above features assume that the object and the camera are parallel. A non-parallel orientation produces additional non-zero elements to the interaction matrices, denoting more coupling.

In order to maximize the decoupling property of the interaction matrix and minimize the non-linearities, Zhao, Xie, and Wang (2012) introduced a method involving two Neural Network (NN) models. They assumed the existence of two moment invariants, referred to as virtual moments and denoted as m_x and m_y, which are invariant to 2D translation, 2D rotation, and scale changes.

The interaction matrices are defined as:

$$\begin{aligned} \mathbf{L}_{m_x} &= \begin{bmatrix} 0 & 0 & 0 & c_x & 0 & 0 \end{bmatrix} \\ \mathbf{L}_{m_y} &= \begin{bmatrix} 0 & 0 & 0 & 0 & c_y & 0 \end{bmatrix} \end{aligned}. \tag{2.51}$$

In these matrices, c_x and c_y represent constant parameters.

Moreover, they proposed that m_x and m_y take the following forms:

$$\begin{aligned} m_x &= c_x \beta \\ m_y &= c_y \gamma \end{aligned}. \tag{2.52}$$

Here, β and γ represent the rotational angles around the x and y axes of the camera frame, respectively.

To apply this approach, based on Hu invariants, four invariants to 2D translation, rotation, and scale were identified. A Neural Network was utilized to establish the non-linear relationship between these invariants and the angles β and γ.

While this approach shows potential, there are notable limitations to consider. The decoupling process is only partially successful, as it leaves other interaction matrices with non-zero elements that could produce undesired rotational velocities around the x and y axes. Secondly, the method employed for generating the data set (specifically rotating the camera around the x and y axes from a fixed point) results in a lack of diversity within the data set. It fails to cover the full range of possible movements and positions, limiting the decoupling of features to a narrow portion of the

operational workspace.

2.4 Optimization

Optimization is a fundamental concept in many areas of research and industry, including engineering, economics, and computer science. It involves finding the optimal solution from a set of feasible solutions for a given problem (Guenin, Könemann, & Tunçel, 2014). The 'optimal' solution is typically defined in terms of minimizing or maximizing an objective function, which is a mathematical representation of the problem's goal. A standard form of a mathematical optimization problem can be formulated as follows (Boyd & Vandenberghe, 2004):

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq b_i; \quad i = 1, \ldots, m \end{array} . \tag{2.53}$$

In the equation above, the vector $x = (x_1, \ldots, x_n)$ denotes the optimization variable of the problem, $f_0 : \mathbb{R}^n \to \mathbb{R}$ is the objective function, and $f_i : \mathbb{R}^n \to \mathbb{R}$, for $i = 1, \ldots, m$, represent the inequality constraint functions. The constants $b_1, \ldots, b_m$ are the bounds for these constraints. A vector x^* is termed as optimal or a solution to the problem (Equation (2.53)), if it yields the smallest objective value among all vectors satisfying the constraints: for any z with $f_1(z) \leq b_1, \ldots, f_m(z) \leq b_m$, we have $f_0(z) \geq f_0(x^*)$ (Boyd & Vandenberghe, 2004).

Optimization problems can be categorized into several types based on the nature of the objective function, the constraints, and the decision variables. These include linear optimization, non-linear optimization, convex optimization, etc. (Boyd & Vandenberghe, 2004). Each type of optimization problem has its unique characteristics and requires specific techniques for solving.

Linear optimization problems involve linear objective functions subject to linear constraints, which means:

$$f_i(\alpha x + \beta y) = \alpha f_i(x) + \beta f_i(y). \tag{2.54}$$

Non-linear optimization problems, on the other hand, involve objective functions and/or constraints that are non-linear. Convex optimization problems are generalizations of linear optimization

problems where the objective function is convex[1], and the constraints form a convex feasible region, which means they satisfy (Boyd & Vandenberghe, 2004):

$$f_i(\alpha x + \beta y) \leq \alpha f_i(x) + \beta f_i(y). \tag{2.55}$$

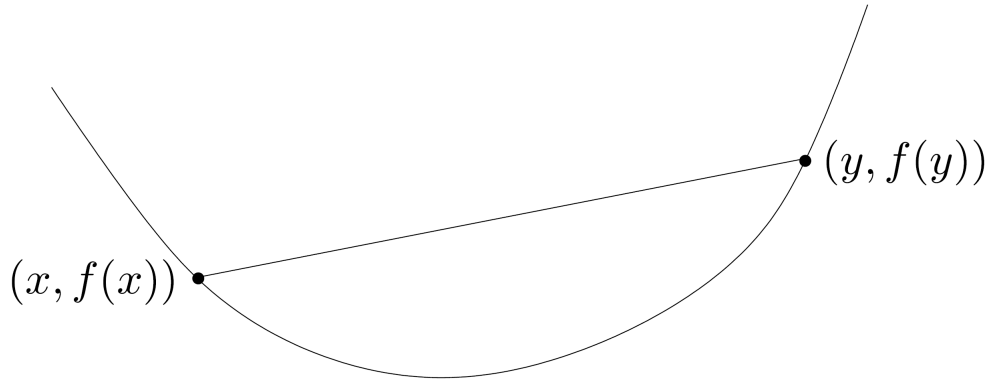


Figure 2.13: Graph of a convex function. The line segment between any two points on the graph lies above the graph (Boyd & Vandenberghe, 2004).

2.4.1 Optimization Techniques

Optimization problems can be solved using a range of techniques that generally fall into two categories: deterministic and stochastic methods. The former, such as gradient descent, offer consistent outcomes and are particularly suited for problems featuring differentiable objective functions and constraints (Sra, Nowozin, & Wright, 2012). Gradient descent operates as a first-order iterative optimization algorithm that aims to find a local minimum of a differentiable function by continuously moving in the direction of the steepest descent, i.e., opposite to the gradient (or an approximation thereof) of the function at the current point (Kingma & Ba, 2014). In robotics, this approach is often used in conjunction with the Adam optimizer (a popular tool in machine learning) for its ability to dynamically adjust the learning rate throughout the training process, thereby navigating the cost landscape more efficiently and pinpointing optimal coefficients with greater accuracy (Kingma & Ba, 2014).

[1] In mathematics, a function is said to be convex if its line segment between any two points lies above or on the graph.

Meanwhile, stochastic methods incorporate elements of randomness into their search processes. These techniques, such as evolutionary algorithms, come into play when dealing with non-differentiable objective functions and constraints, or when the feasible region is not convex (Bäck, Fogel, & Michalewicz, 1997). Evolutionary algorithms are metaheuristic, population-based optimization strategies that use principles of natural evolution—like inheritance, mutation, selection, and crossover—to generate solutions to optimization problems (Bäck et al., 1997).

In conclusion, optimization is a powerful tool for solving complex problems in various fields. Understanding the fundamentals of optimization, the different types of optimization problems, and the techniques for solving these problems is crucial for effectively applying optimization in practice.

The application of optimization in control systems is broad, extending from predictive control methodologies to the direct computation of control input from pixel luminance. However, to the best of my knowledge, the precise application of optimization to decouple image feature functions in visual servoing has not been documented in the literature, presenting a unique direction for this research.

2.5 Neural Networks

Neural networks, especially Deep Neural Networks (DNNs), have revolutionized the field of machine learning. DNNs consist of multiple layers of interconnected nodes or "neurons", enabling them to learn complex patterns and representations from vast amounts of data. The depth of these networks, characterized by the number of layers, allows them to capture intricate details, making them particularly suited for tasks such as image and speech recognition, among others.

A DNN can be mathematically represented as a composition of several functions, each corresponding to a layer in the network. Given an input x, the output y of a DNN with L layers can be represented as:

$$y = f_L(f_{L-1}(...f_2(f_1(x))...)),$$

where f_i denotes the function corresponding to the i^{th} hidden layer (Figure 2.14). Each function involves a linear transformation followed by a non-linear activation function.

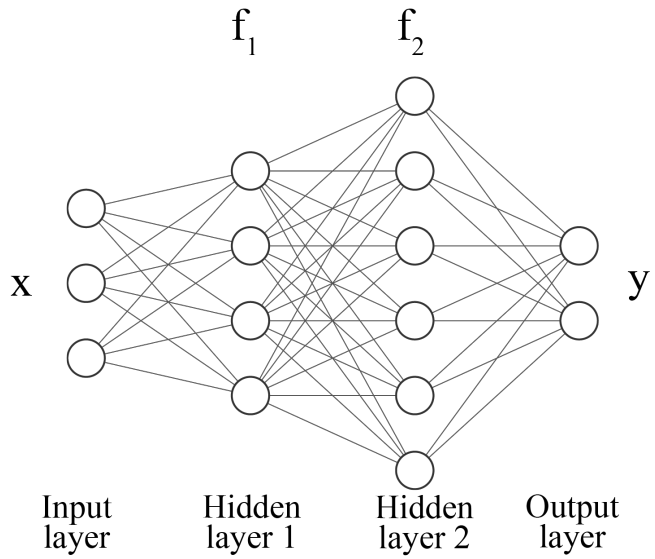


Figure 2.14: Neural Networks Schematic

2.5.1 Neural Networks in Visual Servoing

Visual servoing has been significantly enhanced with the integration of DNNs. These networks, with their ability to process visual data and extract meaningful information, have made visual servoing more accurate and adaptable.

The work by J. Liu and Li (2019) introduces an image-based visual servoing approach combined with deep learning for robotic manipulation. The proposed architecture employs a CNN model to estimate parameters such as x, y, z, and R_z directly from input images, streamlining the robotic manipulation process (Figure 2.15). This method eliminates the need for external sensors or intricate calculations, enhancing the precision and efficiency of robotic tasks.

Bateux, Marchand, Leitner, Chaumette, and Corke (2018) present a method for 6 DOF visual servoing using a deep neural network. The network estimates the relative pose between two images of the same scene, even under challenging conditions like occlusions and lighting variations. The output of the network is then used in a visual servoing control scheme, achieving sub-millimeter positioning accuracy. On another research, Kumra and Kanan (2017) introduce a robotic grasp detection system that predicts the best grasping pose for novel objects using RGB-D images. Their deep convolutional neural network model achieves an accuracy of 89.21% on their dataset.

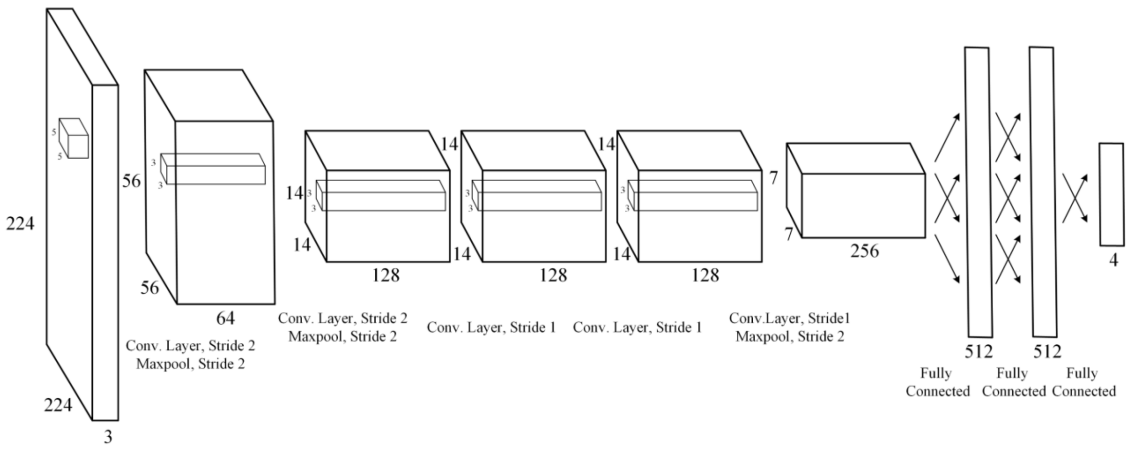


Figure 2.15: CNN Architecture proposed by J. Liu and Li (2019)

2.6 Summary

To summarize, this chapter comprehensively explained the delicate domain of robotic manipulators and visual servoing. Beginning with a description of robotic manipulators and their variety of applications, particularly in space, we transitioned into visual servoing and its numerous applications. The chapter revolved around image-based visual servoing with image moments, where we delved deep into their definitions, interaction matrices, and the widely recognized image features derived from them. We observed that image features coupling is a critical concern within IBVS. Therefore, many researchers have attempted to tackle this problem, yet these approaches come with their limitations.

The techniques and applications of optimization in control systems and visual servoing were introduced in section 2.4. Finally, the potential of neural networks, especially in control systems and visual servoing, was presented in section 2.5. Through this chapter, we have built a robust foundation, preparing for the novelties in the following chapters of this work.

Chapter 3

Prerequisites and Implementation Setup

As a preliminary to the novel methods proposed in the following chapters, this chapter provides a comprehensive explanation of the algorithms for processing images, generating data sets, and the implementation setup used for data collection and experimental validation.

Chapters 4 and 5 introduce methods that use image moments, which are computed from binary images. These binary images are the result of a transformation applied to RGB images. Therefore, Section 3.1 describes the image processing algorithm employed not only to perform this transformation but also to eliminate noise.

Section 3.2 presents the simulation environment employed to create a synthetic data set and to preliminarily validate the proposed methods, avoiding potential risks of implementation on the actual robotic system. For the real-world implementation, a detailed explanation of the required equipment is provided in section 3.3. The description includes the entire number of devices essential for data collection and testing in real-world conditions.

Subsequently, section 3.4 expresses the algorithm developed for the generation of a diverse and random dataset. This dataset includes both image moments and the corresponding poses of the end effector, encompassing simulated and real data to provide a basis for the optimization process (Chapter 4) and the deep neural network (Chapter 5).

3.1 Image Processing

Image processing is crucial in many computer vision and robotics applications, as it allows for extracting meaningful information from captured images. The procedure typically involves several key steps, including segmentation, erosion, dilation, grayscale conversion, and thresholding. The concept of mathematical morphology, initially developed by Matheron and Serra (2000), forms the foundation for these operations.

3.1.1 Segmentation

Image segmentation is the process of partitioning an image into multiple regions or segments, often corresponding to different objects or parts of objects in the scene. Segmentation aims to simplify or change the representation of an image to something more meaningful and easier to analyse.

One of the quickest and most effective methods for object detection in image frames is based on color segmentation. This technique is especially popular in real-time applications due to its speed. The RGB (Red, Green, Blue) color model plays a vital role in such color-based segmentation, where pixels are assigned to a segment based on color. However, due to the variability in brightness and lighting conditions, tolerances for Red, Green, and Blue values are typically considered. By defining a range of RGB values for each feature color, pixels falling within these ranges can be detected and marked accordingly.

3.1.2 Erosion and Dilation

Erosion and dilation are fundamental operations in morphological image processing, used for reducing noise and enhancing the significant components of an image.

Erosion is an operation that chips away at the edges of an object in an image (Figure 3.1 (b)). The result of this operation makes the output image darker than the original, and any light details smaller than the structuring element[1] are weakened or removed.

Dilation, on the other hand, gradually expands the boundaries of regions of foreground pixels

[1] small matrix of pixels with a fixed size, each with a value of zero or one

(Figure 3.1 (c)). The output image becomes lighter than the original following this operation, and any dark details smaller than the structuring element are weakened or removed. The dialation followed by erosion is called opening and is usually used to remove noise without harshly changing the number of pixels (Figure 3.1 (d)).

(a) (b) (c) (d)

Figure 3.1: Binary image morphology: (a) original image. (b) erosion. (c) dialation. (d) opening. (Szeliski, 2022)

3.1.3 Grayscale Conversion

Grayscale conversion, also known as color to grayscale conversion, is the process of changing a full-color image into shades of gray, ranging from black at the weakest intensity to white at the strongest. This process is particularly beneficial in image processing as it simplifies the image data while retaining essential information. In the RGB color model, grayscale conversion is typically achieved by taking the average of the Red, Green, and Blue color channels.

3.1.4 Thresholding

Thresholding is a process that modifies the pixels of an image based on a threshold value, converting an input image into a binary image. This operation is often performed after converting the image to grayscale, as it simplifies the image data and aids in object detection tasks. Mathematically, the output image after thresholding is defined as (Szeliski, 2022):

$$g(x,y) = \begin{cases} 1 & \text{if } f(x,y) > \mathrm{T} \\ 0 & \text{otherwise} \end{cases}, \tag{3.1}$$

where $f(x, y)$ is the input image, $g(x, y)$ is the output image, T is the threshold value, and 0 and 1 correspond to black and white, respectively.

3.1.5 Proposed Image Processing Algorithm

The proposed image processing algorithm involves a series of steps to enhance the significant components of the captured image and to facilitate object detection:

(1) **Segmentation:** The color image is segmented based on a range of RGB values. The pixels falling within the defined RGB ranges are marked as the detected object in the image.

(2) **Erosion:** The segmented image is then subjected to an erosion operation, which chips away at the edges of the object in the image. This operation uses a structuring element of size 3.

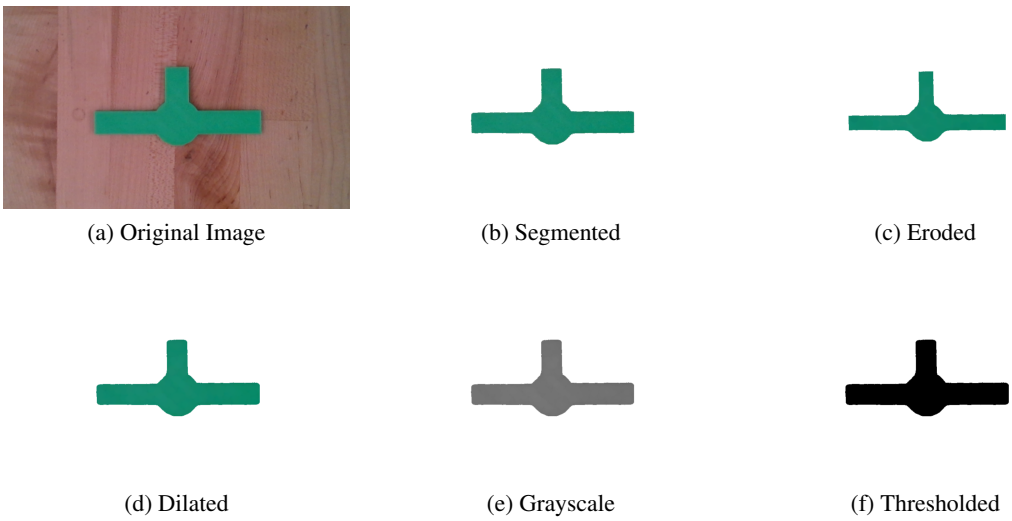
(3) **Dilation:** The eroded image is further processed using a dilation operation. This expands the object's boundaries in the image, further enhancing its features. The dilation operation also uses a structuring element of size 3.

(4) **Grayscale Conversion:** The dilated image is then converted to grayscale, simplifying the image data while retaining significant information.

(5) **Thresholding:** The grayscale image is thresholded to produce a binary image. This further simplifies the image and facilitates object detection.

(6) **Calculation of Moments:** The moments and central moments of the final image are then calculated using functions from the Scikit-image library. The moments calculated by the Scikit-image library are initially computed with respect to the edge of the image. However, they are later converted via a Taylor series expansion so that these moments are instead calculated with respect to the center of the image.

This sequence of operations on the 2D targeting pin are illustrated in Figure 3.2 and finally, image moments and central image moments could be calculated from the thresholded image.



(a) Original Image (b) Segmented (c) Eroded

(d) Dilated (e) Grayscale (f) Thresholded

Figure 3.2: Stages of image processing: (a) Original image, (b) Segmented, (c) Eroded, (d) Dilated, (e) Converted to grayscale, and (f) Thresholded.

3.2 Simulation Environment

For the purpose of this study, the simulation environment utilized was RoboDK software. As an intern at RoboDK, I gained familiarity with the software, which proved beneficial in facilitating this research. RoboDK is a versatile simulation platform offering a range of features suitable for robotic research. One of its significant advantages is the availability of a kinematic representation of several predefined manipulators. This functionality allows for offline programming in various languages, enabling an easy adaptation to different user preferences.

RoboDK's environment is visually appealing and more accurate when compared to similar tools such as Matlab's robotic toolbox. Its precision is particularly beneficial for research requiring meticulous representation of the environment, such as the current study.

Moreover, RoboDK's Python API was another feature that made it an excellent choice for the simulation environment, due to my proficiency in Python for manipulator control. While the original intent was to simulate Canadarm2, RoboDK's library did not include this manipulator. Therefore, integrating its kinematic model into RoboDK would require in-depth backend programming, extending beyond the boundaries of my internship and research aims. As a result, the Denso manipulator was selected for its availability and compatibility with our lab's resources, thus allowing

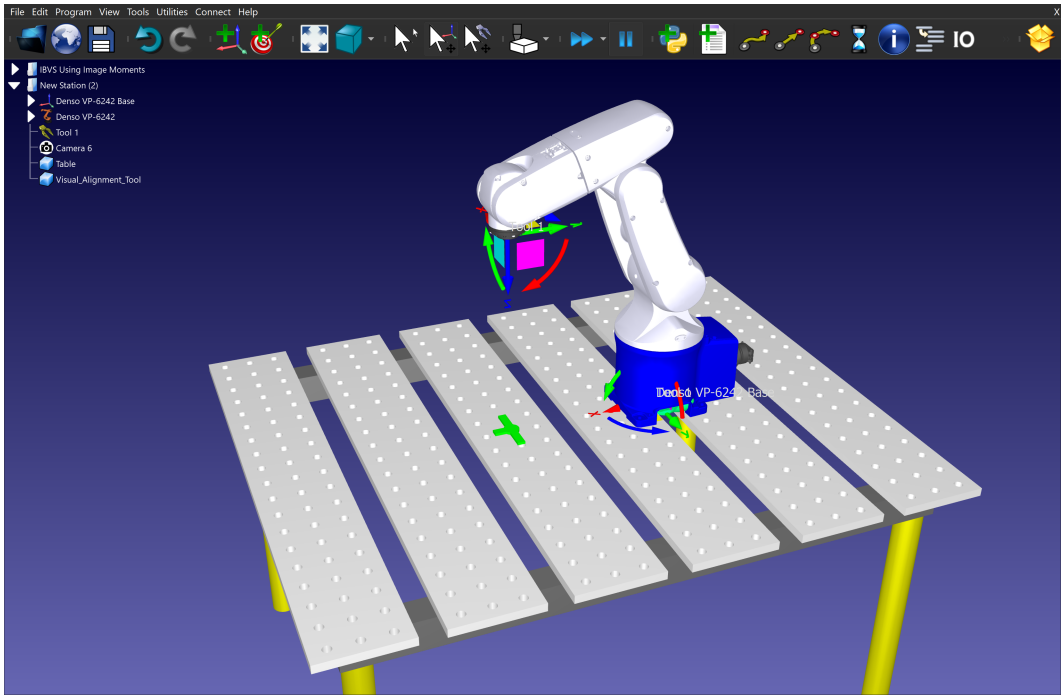


Figure 3.3: RoboDK environment including the Denso robot, the camera, and the targeting pin

the research to focus on the primary goal of enhancing visual servoing techniques.

To test the preliminary findings of this study, the Denso manipulator was first programmed in the RoboDK environment. Using the simulation environment as an initial testing ground reduced the risk of complications that could arise while working directly with the physical Denso robot. This two-step approach facilitated efficient testing and transition from simulation to the physical robot.

In the RoboDK station, a 6 Degree-of-Freedom (DOF) Denso manipulator was added and a 2D camera was attached to its end effector. The manipulator was situated on a table, a setup that closely mimicked our laboratory's physical arrangement. The CAD model of the 3D printed targeting pin was imported into the environment and positioned in front of the manipulator, replicating the real-life scenario.

This systematic setup in the RoboDK environment facilitated an efficient transition from the simulated scenario to the actual physical experiment. The following section describes the real-life experimental setup and the process of implementing the simulation findings on the physical Denso robot.

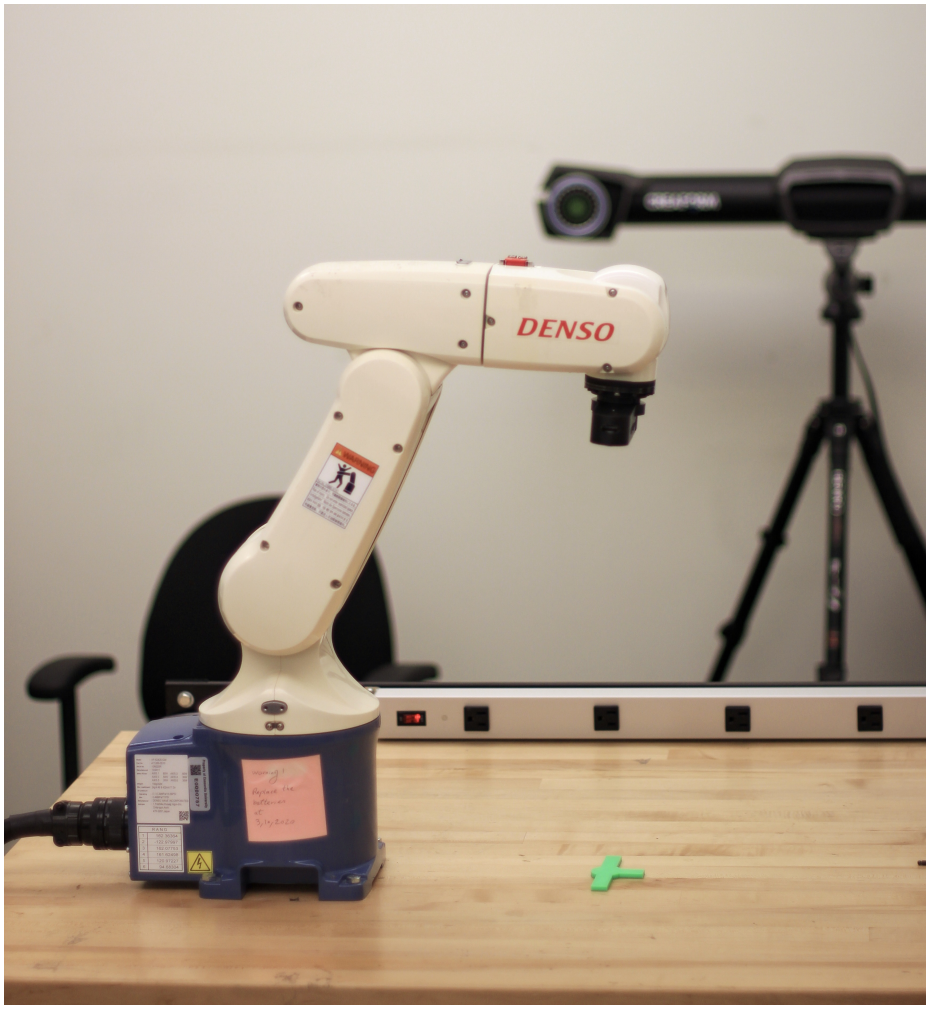


Figure 3.4: Implementation setup within the workspace

3.3 Implementation Setup

Understanding the details of our visual servoing study requires diving into the specifics of the implementation setup. This section aims to explain the components that shape our robotic control system. First, we introduce the Denso Robot (Subsection 3.3.1), the manipulator we used in our research, and discuss its features, components, and kinematics. Next, we investigate the specifics of our camera (Subsection 3.3.2), elaborating on its specifications, functionality, and integration with our robotic arm. Figure 3.4 provides a visual representation of the Denso manipulator, the mounted camera and the targeting pin within the workspace. Further, to ensure smooth communication between devices, we explain our choice of the User Datagram Communication Protocol (UDP) and its

Figure 3.5: six degree of freedom Denso Robot by Quanser (Quanser, 2011)

configuration (Subsection 3.3.3). Finally, we discuss the velocity transformation process necessary for robot control (Subsection 3.3.4).

3.3.1 Robotic Manipulator: Denso

The Denso 6-Axis Robot, as displayed in Figure 3.5, is a versatile and powerful piece of equipment, used widely in both industrial applications and research. This robotic manipulator is renowned for its precision, robustness, and compatibility with various control strategies, making it an ideal choice for our visual servoing study.

A significant advantage of the Denso 6-Axis robot is its open-architecture control module supplied by Quanser. This module, equipped with six amplifiers and built-in FeedForward (FF) plus Proportional-Integral-Derivative (PID) controllers, offers extensive control over the robot's movements. It allows for tuning of the controller gains directly from the QUARC interface or even designing custom control strategies in the Simulink environment, thus offering a high degree of flexibility (Quanser, 2011).

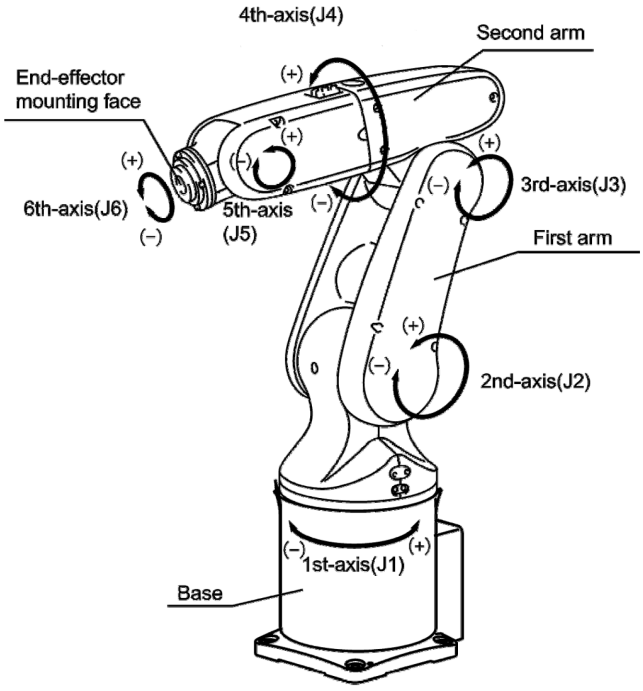


Figure 3.6: Denso components and rotation directions (Denso, 2009)

An essential aspect of the Denso robot's design and functionality can be understood by examining its components and rotation directions, as illustrated in Figure 3.6. This piece of information help us assign frames to each joint correctly.

The Denso 6-Axis robot's kinematic chain is composed of six links with specified lengths, as depicted in Figure 3.7. The forward kinematics and the calculation of the Jacobian matrix rely heavily on the Denavit-Hartenberg (DH) parameters. As discussed in section 2.1, the transformation matrix and geometric Jacobian are computed using the DH parameters. The specifics of the Denso robot's DH parameters are provided in the table below.

3.3.2 Camera: Intel RealSense D415

Our system utilizes the Intel RealSense D415 camera, as illustrated in Figure 3.8. The D415 is compact and versatile, making it an ideal choice for our setup. The camera's dimensions, just 99mm wide, 23mm high, and 20.05mm deep (as per Figure 3.9a), allow it to integrate with our robotic arm setup.

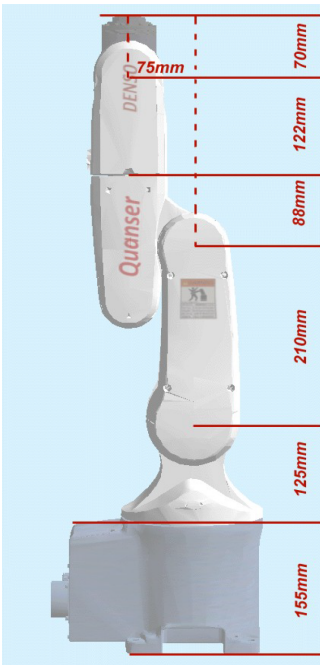


Figure 3.7: Denso link lengths (Quanser, 2011)

Table 3.1: Denavit-Hartenberg parameters for the Denso 6-Axis Robot

Joint	θ_i	$\mathbf{d}_i$ (mm)	$\mathbf{a}_i$ (mm)	α_i	**Joint Limits** (degrees)
1	q_1	125	0	$\pi/2$	-160, 160
2	q_2	0	210	0	-120, 120
3	q_3	0	75	$\pi/2$	-162, -38
4	q_4	210	0	$\pi/2$	-160, 160
5	q_5	0	0	$\pi/2$	-120, 120
6	q_6	70	0	0	-360, 360
Camera	-$\pi/6$	20	0	0	—

One of the standout features of the D415 is its RGB sensor, which is capable of capturing high-resolution images in a variety of sizes, from 424 $\times$ 240 pixels up to a detailed 1920 $\times$ 1080 pixels. To maintain image consistency, which is crucial for effective image processing, it is essential to disable the camera's automatic settings for features such as exposure and white balance. Instead, these settings are fixed at predetermined optimal values, ensuring consistent image quality throughout the system's operation.

Moreover, the D415 comes with a Python API provided by RealSense. This facilitates straightforward and robust communication with the camera, further simplifying the process of image acquisition.

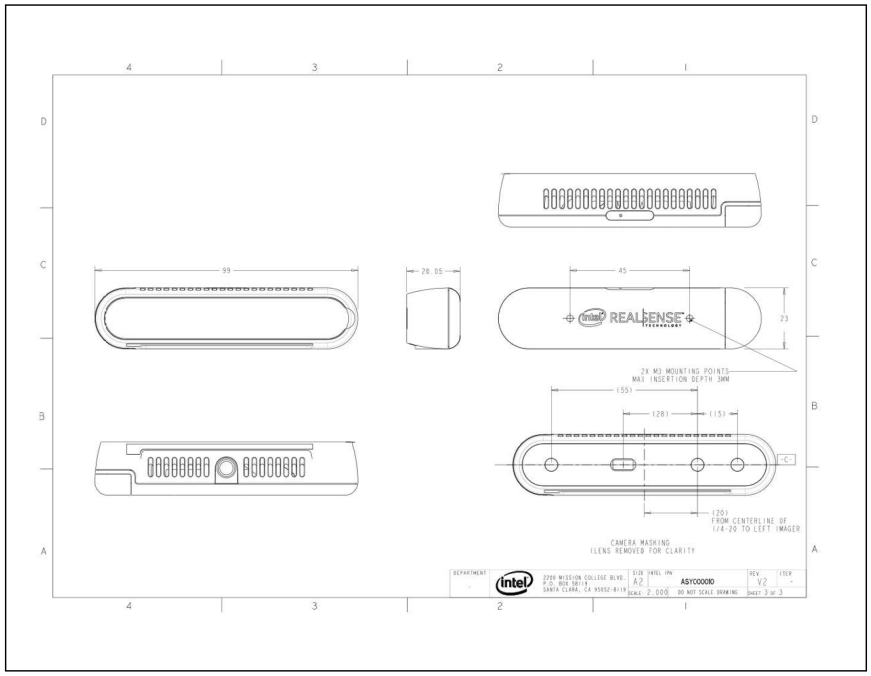
Figure 3.8: Intel RealSense D415 camera

To attach the D415 to the robotic arm, we designed a custom camera holder (shown in Figure 3.9b). Careful consideration was taken in the design process to ensure that once mounted, the camera's RGB lens is centered with the end-effector. Furthermore, the holder includes ventilation gaps at the side and a port at the end for connecting the wire, providing functionality without compromising on design. The mounting process begins by securing the back of the camera to the holder using two screws. Once the camera is fixed, the holder, with the camera in place, is then attached to the end-effector using two additional screws. This design and mounting process guarantees a secure and functional integration of the camera into our system.

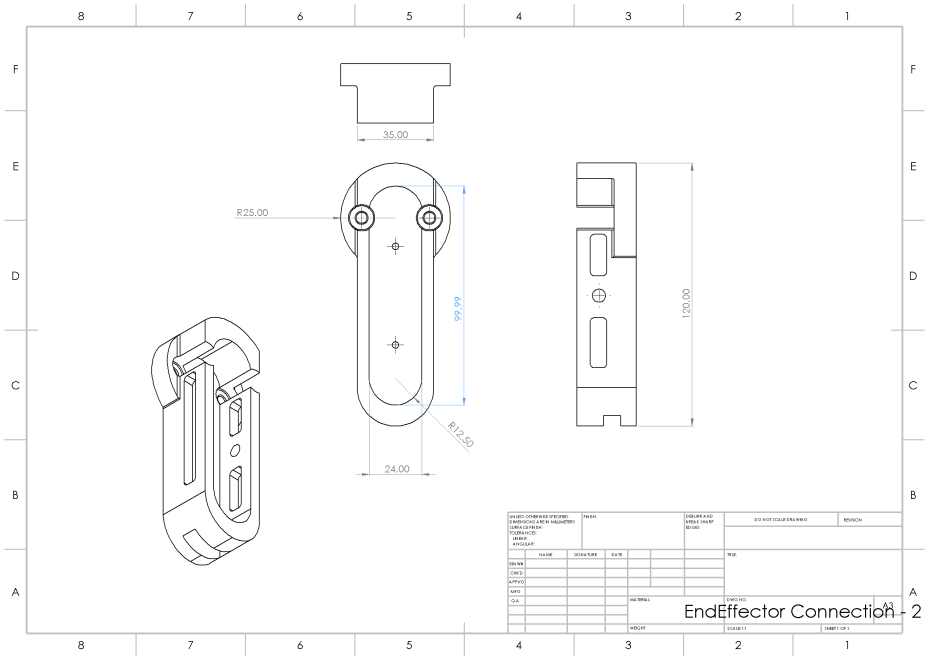
3.3.3 User Datagram Communication Protocol

Communication between devices, especially in the context of robotics, is crucial for effective operation. User Datagram Protocol (UDP), an established standard for data transmission, has found widespread application in the field due to its simplicity and efficiency. Unlike its counterpart, Transmission Control Protocol (TCP), UDP is a connectionless protocol[2] that doesn't require the establishment of a dedicated path for data transfer, making it faster and more suitable for real-time operations where time efficiency is crucial.

[2]In a connectionless protocol, data packets are sent from one point to another without a prior arrangement. Each packet is independent, meaning it may take different paths and may not even arrive at the destination.

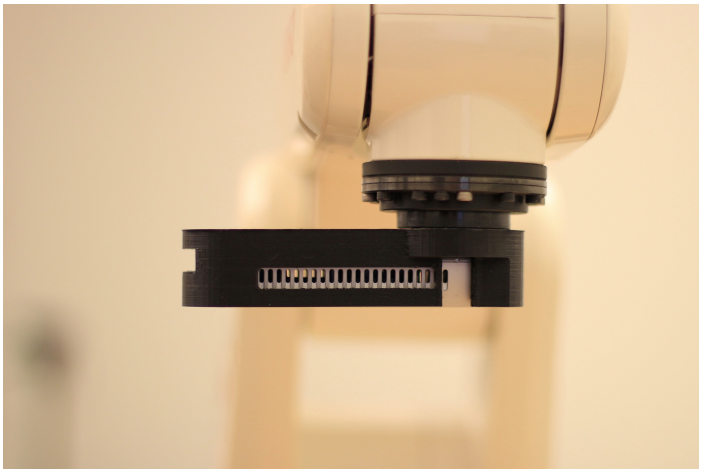


(a)



(b)

Figure 3.9: (a) Intel RealSense D415 dimensions (Intel, 2019) (b) Camera holder dimensions



(a)

(b)

(c)

Figure 3.10: Mounted camera on Denso's end-effector (a) Side view (b) Front view (c) 3D view

The vision system requires substantial computational resources to perform image processing and velocity calculations. In order to mitigate any possible delays in the implementation process that could affect real-time performance, the vision system is associated with a secondary PC (PC 2). This setup assigns image processing and velocity calculation tasks to PC 2, with the processed data (whether velocity or pose) sent to the main robot controller PC via a UDP network connection (Figure 3.11).

The communication setup involved configuring two elements: the server address and the port number. The server address is the IP address of the device that is intended to receive the data. The port number, on the other hand, helps identify the specific process to which data is to be delivered on the receiving device. When connected to a local network, to ensure a consistent connection between

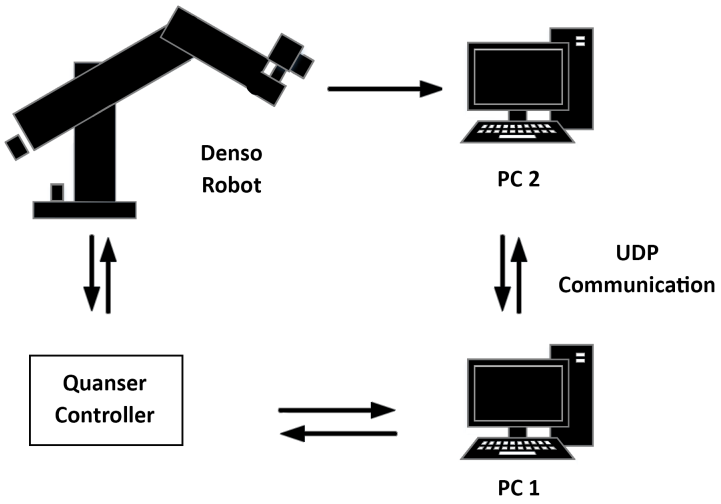


Figure 3.11: Schematic representation of the implementation setup

the machines, it is recommended to use static IP addresses.

Figure 3.12 illustrates the UDP send and receive blocks in Simulink, which are essential to facilitate the transfer of data packets between our personal laptop (PC 2) and the lab computer (PC 1).

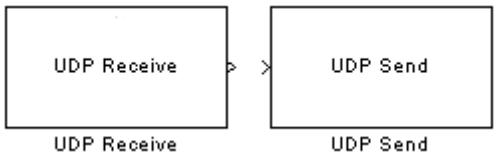


Figure 3.12: UDP (a) Receive Block. (b) Send Block.

UDP allowed us to utilize the processing power of our personal computer to enhance the overall efficiency of our robot control setup while maintaining a stable and reliable communication link with the robot via the lab's computer. It indeed proved to be a powerful solution for our specific needs in this project.

3.3.4 Velocity Conversion

In the Image-Based Visual Servoing (IBVS) method, the output is the velocity of the end effector in the end effector's reference frame, denoted as ${}^{ee}\mathbf{v}$. However, the Denso manipulator system

requires inputs in the form of joint velocities. This necessitates a transformation process to convert the end effector velocity to the corresponding joint velocity.

The joint velocity can be obtained through the following equation:

$$\dot{\mathbf{q}} = \mathbf{J}^{-1} \times {}^{o}_{ee}\mathbf{T}_v \times {}^{ee}\mathbf{v}, \tag{3.2}$$

where:

- $\dot{\mathbf{q}}$ is the joint velocity vector.
- $\mathbf{J}$ is the geometric jacobian of the manipulator.
- ${}^{o}_{ee}\mathbf{T}_v$ is the velocity transformation matrix, converting velocity from the end effector's reference frame to the robot origin's reference frame.

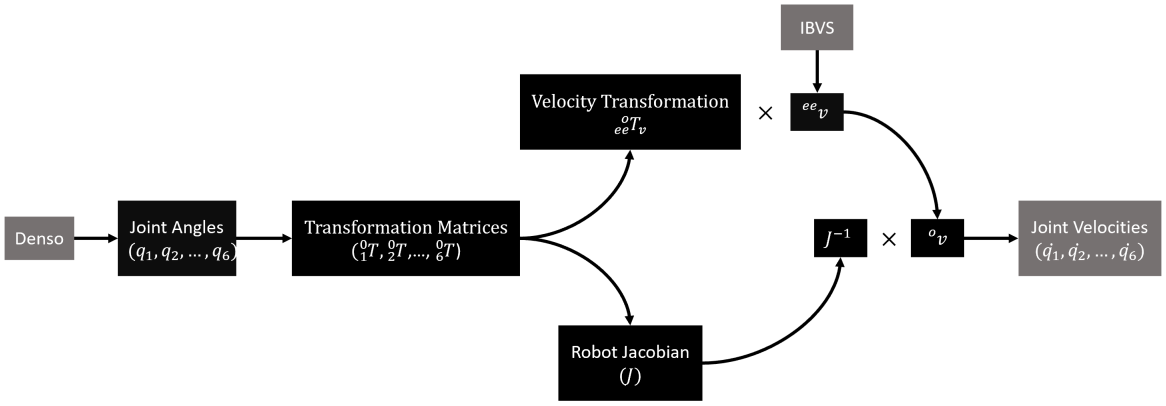


Figure 3.13: Velocity conversion flowchart

The entire process of this velocity conversion is also visually represented in Figure 3.13, providing a comprehensive flowchart of the described procedure.

3.4 Dataset Generation

A comprehensive simulated dataset is required for optimizing the image feature and validating the algorithm in simulation. To this end, we used RoboDK, a sophisticated offline programming and

simulation platform designed specifically for robotics applications. Our simulation setup consisted of a Denso manipulator equipped with a camera mimicking the properties of the Intel RealSense D415.

Great care was taken in constructing the simulation setup to ensure the object of interest remained within the camera's field of view throughout all robot manipulations. To achieve this, we strategically placed the target object on the table within the Denso manipulator's operational field. Crucial to the success of this setup was the definition of suitable pose ranges for the manipulator's end effector.

Firstly, the x, y, and z coordinates of the end effector (camera) were randomly generated within the working range of the manipulator. The pose needed for the camera to focus on the object with the object perfectly at the center was then computed using a "Look at" function. The "Look at" function is typically designed to orient the camera or the end effector of a robotic arm towards a specific point (object's centroid in our case) in the environment.

This function starts by defining a source point (camera) and a target point (object), along with initial vectors for up (U), front (F), and right (V). V, U and F are initially considered as the unit vectors pointing in the positive x,y and z axes, respectively:

$$\mathbf{V} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \mathbf{U} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \mathbf{F} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \tag{3.3}$$

The first step is to calculate the new front vector $\mathbf{F}^{'}$, which points from the source to the target. This vector is obtained by subtracting the source position ($\mathbf{x}_c$) from the target position ($\mathbf{x}_o$) and normalizing the resulting vector:

$$\mathbf{F}^{'} = \frac{\mathbf{x}_o - \mathbf{x}_c}{||\mathbf{x}_o - \mathbf{x}_c||}. \tag{3.4}$$

Next, we calculate the new up vector $\mathbf{U}^{'}$. We start by subtracting the projection of U onto $\mathbf{F}^{'}$ from U and then normalize it:

$$\mathbf{U}^{'} = \frac{\mathbf{U} - (\mathbf{U} \cdot \mathbf{F}^{'})\mathbf{F}^{'}}{||\mathbf{U} - (\mathbf{U} \cdot \mathbf{F}^{'})\mathbf{F}^{'}||}. \tag{3.5}$$

In case the resulting vector has zero magnitude, we default $\mathbf{U}^{'}$ to be the same as the original front vector $\mathbf{F}$ ($\mathbf{U}^{'} = \mathbf{F}$).

The third axis, $\mathbf{V}^{'}$, is calculated as the cross product of $\mathbf{U}^{'}$ and $\mathbf{F}^{'}$:

$$\mathbf{V}^{'} = \mathbf{U}^{'} \times \mathbf{F}^{'}. \tag{3.6}$$

These new basis vectors ($\mathbf{V}^{'}$, $\mathbf{U}^{'}$, $\mathbf{F}^{'}$) form the rotation matrix for the new camera pose:

$${}^{c}\mathbf{R}_{o} = \begin{bmatrix} \mathbf{V}^{'} \\ \mathbf{U}^{'} \\ \mathbf{F}^{'} \end{bmatrix}. \tag{3.7}$$

Finally, the pose of the camera is represented as a 4×4 transformation matrix:

$$\mathbf{P}_{c} = \left[\begin{array}{c:c} {}^{c}\mathbf{R}_{o} & \mathbf{x}_{c} \\ \hdashline \mathbf{0} & 1 \end{array}\right] \tag{3.8}$$

This matrix represents the transformation that must be applied to the camera to look at the object.

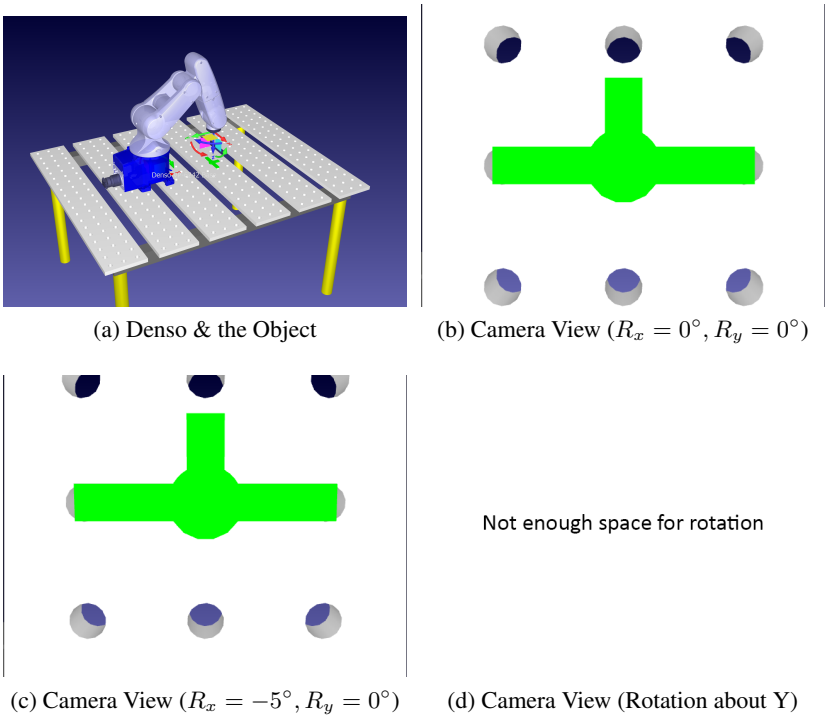
The camera's rotational degrees of freedom must also be random. First, we rotate the camera around its optical axis within a specified range. Then, we determine the rotation limits for the camera around its x and y axes based on the camera's distance from the object. We could obtain the rotation limits by performing linear interpolation between predefined limits at two known distances, ensuring the object would remain in the image plane. Consequently, the camera was rotated around its x and y axes to a random value within these limits. The exact ranges of the pose parameters were:

As shown in Figures 3.14 and 3.15, the camera's x- and y-axis rotation limits are determined by

Table 3.2: Ranges of the pose parameters (Limit$_{R_x}$ and Limit$_{R_y}$ are determined by linear interpolation)

Parameter	Minimum Value	Maximum Value
X	207.5 (mm)	407.5 (mm)
Y	-150 (mm)	150 (mm)
Z	150 (mm)	500 (mm)
R_x	$-\text{Limit}_{R_x}$	Limit_{R_x}
R_y	$-\text{Limit}_{R_y}$	Limit_{R_y}
R_z	45°	135°

interpolating between situations at minimum and maximum distances.



(a) Denso & the Object

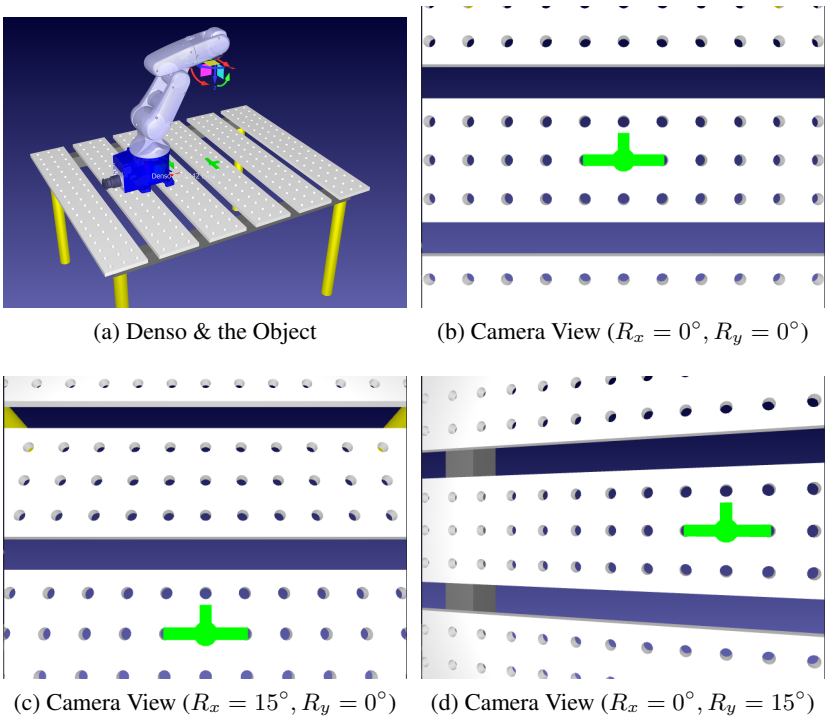
(b) Camera View ($R_x = 0^\circ, R_y = 0^\circ$)

(c) Camera View ($R_x = -5^\circ, R_y = 0^\circ$)

(d) Camera View (Rotation about Y)

Figure 3.14: Camera Rotations at Minimum Distance (160 mm)

In the data set, each entry corresponds to a specific pose of the Denso manipulator and comprises the pose itself, the calculated image moments, and the central moments of the image captured at that pose. Initially, in the simulated environment, 434,528 random poses were generated within the ranges of Table 3.2, sequentially commanded to the Denso manipulator. At each pose, an image was captured by the mounted camera and processed into a binary representation, and the moments and central moments were then computed. The entire process of generating the synthetic data set,



(a) Denso & the Object

(b) Camera View ($R_x = 0°, R_y = 0°$)

(c) Camera View ($R_x = 15°, R_y = 0°$)

(d) Camera View ($R_x = 0°, R_y = 15°$)

Figure 3.15: Camera Rotations at Maximum Distance (500 mm)

from iterating over all the random poses to processing the images and compiling the dataset, took approximately 13 hours. The synthetic data set consisted of 405 MB of data saved into a .npz file, forming an integral part of our resources for training.

In addition to the synthetic data, we also captured real data to enrich the data set and enhance the robustness of our methods against real-world variations. For this, we used a similar approach to generate random end effector poses. However, due to the slower operational tempo of the physical setup compared to the simulator, we recorded data during the motion of the end effector from one random pose to another. However, it is necessary to apply a few conditions to avoid capturing useless data. As a result, data points were only recorded when a single contour larger than 500 pixels was detected in the image, ensuring the presence of the object, and when the bounding box of the target pin was at least 10 pixels away from the image borders. This cautious approach resulted in 1912 distinct poses being fed to the Denso robot, yielding a total of 198,588 valid real-world data points.

To provide a visual representation of how the data set is created, two videos were prepared to

show the process in action. The first video demonstrates the simulation environment data generation, accessible at this link, while the second video shows the real environment data generation, available at this link.

The final data set was carefully partitioned, with all synthetic and half of the real data allocated for training. The remaining real data was evenly divided between the validation and test sets (Figure 3.16). This approach originated from our experimental findings that relying only on either synthetic or real data reduced the performance on the test set, likely due to the real environment's noise and lighting conditions and the limited diversity of poses in the real data. As a result of combining both data sources, we were able to achieve a balance that captured both the complexity of real-world scenarios and provided enough variability for robust model training.

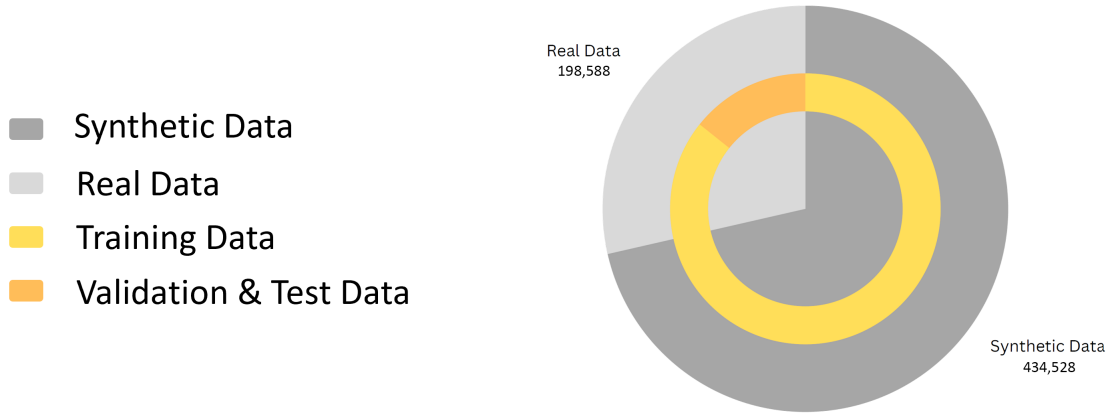


Figure 3.16: Pie chart representation of synthetic and real data used for training, validation, and testing phases

3.4.1 Outlier Removal

Outlier removal is a crucial step in the data preparation process, especially for deep neural networks (DNNs). The necessity for this step is pronounced when dealing with real-world data, which often includes noise and redundancy due to inaccuracies in camera and manipulator readings. Such outliers can adversely affect the training of the DNN, leading to suboptimal performance. While our simulation data is less susceptible to these issues, real-world data calls for rigorous outlier removal methods. Two principal techniques have been deployed for this purpose:

- **Interquartile Range (IQR):** Interquartile Range (IQR) is a measure of statistical distribution.

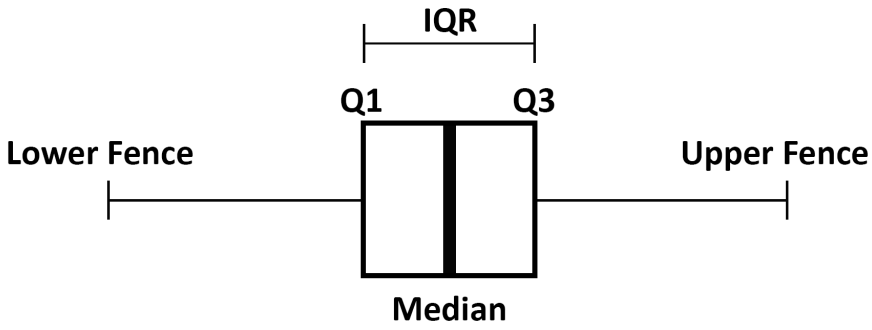


Figure 3.17: Boxplot with an Interquartile Range

Unlike the range, which only considers the extreme values, the IQR gives us a better idea about the spread of the central data.

As depicted in figure 3.17, the IQR is defined as the difference between the third quartile (75th percentile, denoted as Q_3) and the first quartile (25th percentile, denoted as Q_1). The quartiles divide the data into four equal parts. Q_1 is the median of the lower half (not including the overall median if the number of data points is odd), and Q_3 is the median of the upper half of the data.

$$IQR = Q_3 - Q_1 \tag{3.9}$$

One typically uses the IQR to define bounds for the data to identify outliers. Any data point below the lower fence or above the upper fence is generally considered an outlier. These bounds are defined as follows:

$$\begin{cases} \text{Lower Fence} = Q_1 - 1.5 \times IQR \\ \text{Upper Fence} = Q_3 + 1.5 \times IQR \end{cases} . \tag{3.10}$$

- **Density-Based Spatial Clustering of Applications with Noise (DBSCAN):** DBSCAN (introduced by Ester, Kriegel, Sander, Xu, et al. (1996)) is a clustering algorithm that identifies clusters in a dataset based on the density of points. Unlike traditional clustering methods such as K-means, DBSCAN does not require the number of clusters to be predefined. This makes it particularly effective for outlier detection and handling irregularly shaped clusters.

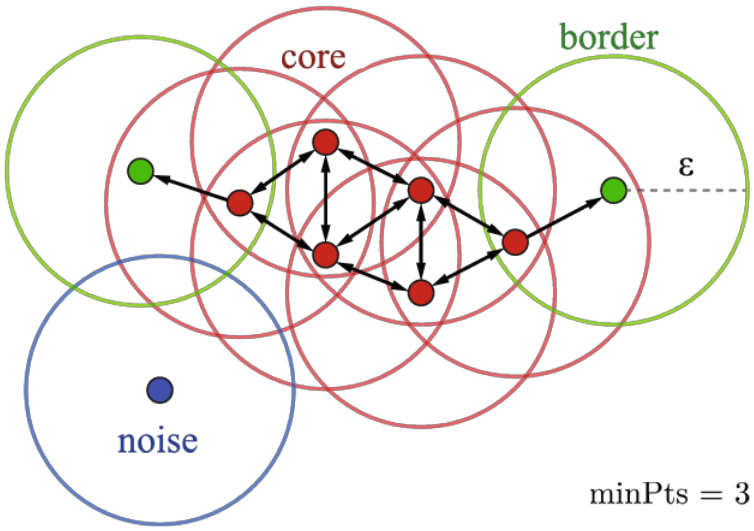


Figure 3.18: A cluster consists of core points (red), border points (green), and noise points (blue). (Mehle et al., 2017)

The algorithm starts by selecting an arbitrary data point. If there are sufficient nearby points (defined by a distance ϵ and minimum number of points, MinPts), a cluster is formed. The algorithm then iteratively adds neighboring data points within ϵ distance to the cluster. As depicted in figure 3.18, points in the dataset are classified into three categories:

- **Core Points:** A point is a core point if there are at least MinPts within ϵ distance.
- **Border Points:** A point which is within ϵ distance of a core point but itself is not a core point.
- **Noise Points:** A point that is neither a core point nor a border point.

The performance of DBSCAN depends on the selection of its two main parameters. We used 4 and 1 as our MinPts and ϵ, respectively. These values were obtained by trial and error.

By performing the DBSCAN algorithm, we can filter out noise points as outliers.

An example is only flagged as an outlier if both IQR and DBSCAN mark it as such. This ensures a more robust and conservative outlier removal process, which balances statistical and density-based methods.

Each IQR and DBSCAN method found 109563 and 1793 outliers, respectively. Among these, 1777 were mutual and were removed from the dataset.

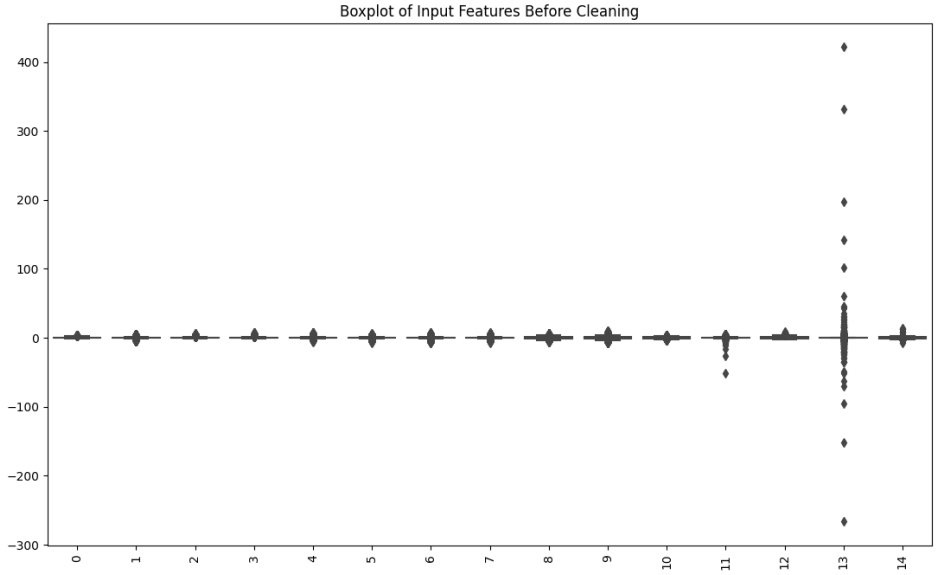
Figure 3.19 illustrates the input boxplots before and after outlier removal.

3.5 Summary

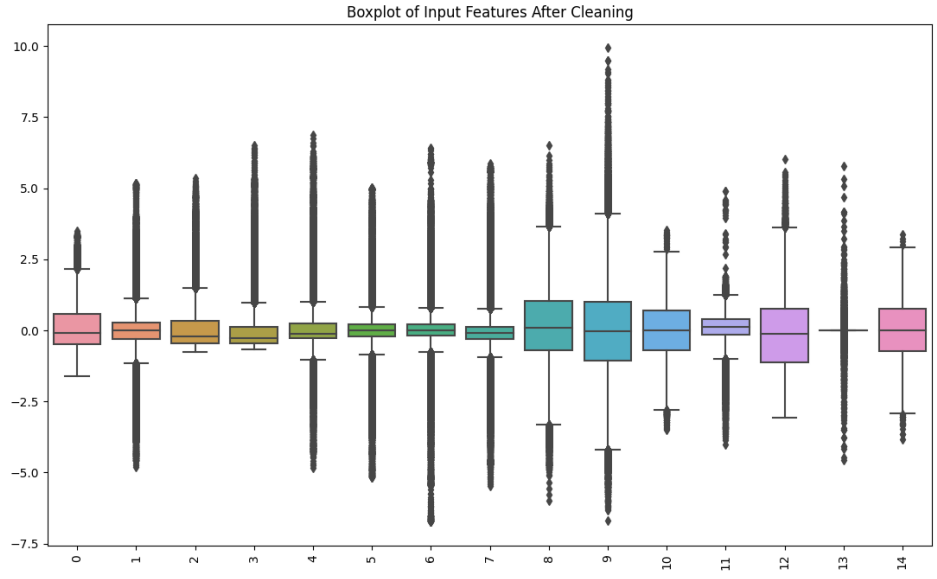
We start in the first section with Image Processing, which involves transforming RGB images into binary images to facilitate image moment calculation.

Subsequent sections, 3.2 and 3.3, explain the simulated and real-world setups. These sections introduce the requirements for the data set generation and the experiments to verify the proposed methods. In the aforementioned sections, we introduce the Denso manipulator, the imaging system, the UDP communication protocol, and the velocity conversion methods, which all are used in Chapters 4 (Function-based visual servoing) and 5 (Deep neural network-based visual servoing).

Following this, in Section 3.4, a method is described to maneuver the end effector to specific poses, randomly generating each of its six degrees of freedom. This method ensures that the object remains in the camera's view. As a result, datasets from both the RoboDK simulation and the real-world environment using the Denso Manipulator were generated. While the simulated environment was ideal, the real-world data showed inevitable noise and potential errors in segmentation, necessitating an outlier removal step to purify the data.



(a)



(b)

Figure 3.19: Input boxplots (a) before and (b) after outlier removal

Chapter 4

Function-based Visual Servoing

This chapter proposes a novel method to address the important problem of coupled image features within the context of image-based visual servoing (described in Subsection 2.3.3). Chaumette (2004)'s fundamental research introduced a set of features for manipulator control using image moments (Eqs. 2.37, 2.47, and 2.39), which are effective but not ideal due to their coupled characteristics. Subsequent attempts in the works of Tahri and Chaumette (2005), S. Liu et al. (2009), and Zhao et al. (2012), have provided slight improvements but also showed limitations in existing methods. This provides the motivation for the development of alternative strategies.

In this chapter, we explain an innovative approach to decouple the rotational image features around the camera's x and y axes (R_x and R_y) using an optimization process. Here, we have defined an image feature function which consists of image moments and a multitude of trainable parameters. The main point of this method involves training the parameters of this generalized image feature function with the objective of providing an ideal interaction matrix for all of the possible image scenarios.

4.1 Image Feature Definition

The central goal of this method involves optimizing the coefficients of two general functions, which are two novel image features based on central image moments.

One key objective is to formulate an image feature that yields a linear relationship with the

rotation about the x-axis (R_x) and the y-axis (R_y). This is crucial as it allows for a meaningful error computation when controlling the robot. In other words, it is expected to construct an image feature such that the sign of the error is associated with the robot rotation direction, i.e. positive error related to one direction and negative error related to the opposite direction.

The optimization process is further directed towards aligning the interaction matrix, computed from this image feature, as closely as possible with a pre-specified target interaction matrix. Here, for instance, are the target interaction matrices for R_x and R_y:

Table 4.1: Target interaction matrices for R_x and R_y

Rotation Axis	**Target Interaction Matrix**
x	[0,0,0,1,0,0]
y	[0,0,0,0,1,0]

As the image feature, we established a general function (Equation (4.1)), inspired by the previously defined image features in the literature, using the combination of central moments up to the third order. In particular, we use eight central moments μ_{00}, μ_{11}, μ_{20}, μ_{02}, μ_{21}, μ_{12}, μ_{30}, μ_{03} while μ_{01} and μ_{10} are excluded as they are consistently zero. These central moments serve as key components, forming various terms in our fraction's numerator and denominator.

The power and sign of each central moment are critical in formulating the function. A table that summarizes this information is presented below:

Table 4.2: Central moments with their corresponding signs and powers.

Central Moment	μ_{00}	μ_{11}	μ_{20}	μ_{02}	μ_{21}	μ_{12}	μ_{30}	μ_{03}
Sign	+	+/-	+	+	+/-	+/-	+/-	+/-
Power	Variable	1, 2	Variable	Variable	1, 2	1, 2	1, 2	1, 2

Because of the even powers for both x and y, the central moments μ_{00}, μ_{20}, and μ_{02} are always positive (refer to Equation (2.16)) and hence can take any real number as their powers (Table 4.2). On the other hand, the remaining central moments can be either positive or negative. Thus, we limit their powers to integers 1 and 2 to avoid imaginary values.

We then form the numerator and denominator of our function from these central moments. Each term in the numerator is a product of up to two central moments and a corresponding coefficient. We also treat the variable powers of μ_{00}, μ_{20}, and μ_{02} as trainable parameters (Table 4.2). Hence,

the numerator consists of the sum of all these terms. Similarly, we construct the denominator with the same terms as the numerator but use different coefficient indices.

The predefined set for orders of the central image moments (μ) is defined as:

$$\mathcal{M} = \{\text{"00"}, \text{"11"}, \text{"20"}, \text{"02"}, \text{"30"}, \text{"03"}, \text{"21"}, \text{"12"}\}$$

.

And the full expression of the function we aim to optimize is:

$$F = \frac{A}{B}, \tag{4.1}$$

where polynomials A and B are:

$$\begin{aligned} A = c_0 + \sum_{t \in M} \Big(\gamma(t)(a_{t_1}\mu_t + a_{t_2}\mu_t^2) + (1-\gamma(t))\mu_t^{c_{t_1}}\Big) + \\ \sum_{\substack{t,s \in M \\ t \neq s}} \Big(\gamma(t)\gamma(s)(b_{t_1s_1}\mu_t\mu_s + b_{t_2s_1}\mu_t^2\mu_s + b_{t_1s_2}\mu_t\mu_s^2 + b_{t_2s_2}\mu_t^2\mu_s^2) + \\ \gamma(t)(d_{t_1}\mu_t + d_{t_2}\mu_t^2)(1-\gamma(s))\mu_s^{c_{s_2}} + e_{ts}(1-\gamma(t))\mu_t^{c_{t_3}}(1-\gamma(s))\mu_s^{c_{s_4}}\Big) \end{aligned} \tag{4.2}$$

$$\begin{aligned} B = c_1 + \sum_{t \in M} \Big(\gamma(t)(g_{t_1}\mu_t + g_{t_2}\mu_t^2) + (1-\gamma(t))\mu_t^{f_{t_1}}\Big) + \\ \sum_{\substack{t,s \in M \\ t \neq s}} \Big(\gamma(t)\gamma(s)(h_{t_1s_1}\mu_t\mu_s + h_{t_2s_1}\mu_t^2\mu_s + h_{t_1s_2}\mu_t\mu_s^2 + h_{t_2s_2}\mu_t^2\mu_s^2) + \\ \gamma(t)(k_{t_1}\mu_t + k_{t_2}\mu_t^2)(1-\gamma(s))\mu_s^{f_{s_2}} + j_{ts}(1-\gamma(t))\mu_t^{f_{t_3}}(1-\gamma(s))\mu_s^{f_{s_4}}\Big), \end{aligned} \tag{4.3}$$

in which, the function $\gamma(n)$ is defined as:

$$\gamma(n) = \begin{cases} 0 & \text{if order n} \equiv 0 \text{ (mod 2)}, n \neq \text{"11"} \\ 1 & \text{otherwise} \end{cases}, \tag{4.4}$$

and the order of a central moments μ_{mn} is $m+n$. It is noteworthy to mention that in the equations 4.2 and 4.3, c_i, a_{t_i}, $b_{t_i s_j}$, c_{t_i}, c_{s_i}, d_{t_i}, e_{ts}, g_{t_i}, $h_{t_i s_j}$, f_{t_i}, f_{s_i}, k_{t_i}, and j_{ts} are all trainable parameters. The numerator and denominator each have 126 trainable parameters, making a total of 252 trainable parameters (refer to Appendix A.1) that we aim to optimize during the optimization process.

4.2 Interaction Matrix of Image Features

Once the image feature has been selected, the next step is to compute its interaction matrix. Formally, the interaction matrix of a feature represents how small changes influence the variation of the image feature in its variables. It is computed by taking the derivative of the feature with respect to each of its variables (in this case, the moments) and then multiplying each of these derivatives by the corresponding interaction matrix of the variable.

Using the chain rule to compute the partial derivative of a multivariable function, the general formula for the interaction matrix of a feature F, denoted as $\mathbf{L}_F$, is given by:

$$\mathbf{L}_F = \sum_{t \in M} \frac{\partial F}{\partial \mu_t} \mathbf{L}_{\mu_t}. \tag{4.5}$$

In this equation, the summation is over all the variables t of the feature, and $\frac{\partial F}{\partial \mu_t}$ represents the partial derivative of the feature F with respect to the central moment μ_t. Each of these partial derivatives is then multiplied by the corresponding interaction matrix $\mathbf{L}_{\mu_t}$.

For the specific case of the image feature defined by its central moments up to the third order $\mu_{00}, \mu_{11}, ..., \mu_{03}$, this general formula expands to:

$$\mathbf{L}_F(\mu_{00}, \mu_{11}, \ldots, \mu_{03}) = \frac{\partial F}{\partial \mu_{00}} \mathbf{L}_{\mu_{00}} + \frac{\partial F}{\partial \mu_{11}} \mathbf{L}_{\mu_{11}} + \ldots + \frac{\partial F}{\partial \mu_{03}} \mathbf{L}_{\mu_{03}}, \tag{4.6}$$

where $\mathbf{L}_{\mu_{ij}}$ is defined in 2.3.2.

4.3 Optimization Algorithm

The optimization process begins either with a random initialization of the coefficients or by loading them from a previously saved state. This is the initial step on the path towards optimal coefficients. An adaptive learning rate strategy is implemented to allow the algorithm to adjust the learning rate during the training process. When the cost value does not show a significant decrease, the learning rate is reduced to allow finer adjustment of the coefficients. The adaptive learning strategy aims to efficiently navigate the cost landscape to identify the optimal coefficients more precisely.

The steps of the optimization process are as follows:

(1) **Initialization:** The coefficients are either randomly initialized or loaded from a previously saved state.

(2) **Compute Cost Function:** The cost function is computed with the current set of coefficients. It takes the coefficients and the training data as inputs. It outputs several metrics, including the cost, mean squared error (MSE), difference penalty, correlation penalty, and correlation value (see subsection 4.3.1). The cost function can be expressed as follows:

$$J(\theta) = \text{MSE} + \lambda_1 \times \text{Difference Penalty} + \lambda_2 \times \text{Correlation Penalty}. \tag{4.7}$$

(3) **Calculate Gradients:** The gradients of the cost function with respect to the coefficients are calculated. This step determines the direction in which we update our coefficients.

$$\text{Gradients} = \nabla_\theta J(\theta). \tag{4.8}$$

(4) **Update Coefficients:** Using the gradients, the Adam[1] optimizer updates the coefficients.

(5) **Monitor Progress and Save Coefficients:** At regular intervals (defined by the print frequency), the progress is monitored by printing out the current cost and other metrics. If the save flag is set to True, the current coefficients are also saved for future use.

[1]The Adam optimizer is a popular choice in machine learning tasks due to its efficiency.

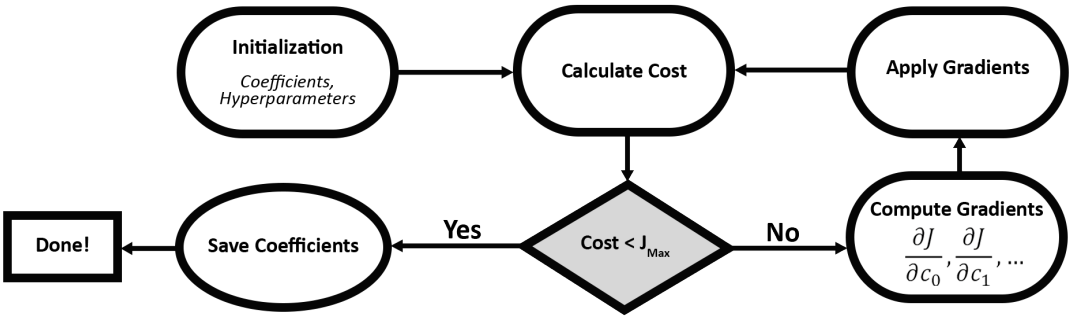


Figure 4.1: Optimization flowchart

(6) **Check for Convergence:** After each iteration, we check whether the algorithm has converged, i.e., whether the decrease in cost has become negligible. If the cost is not reducing significantly, we adjust the learning rate to fine-tune the coefficient updates.

This process is repeated for a predetermined number of iterations or epochs (figure 4.1). The primary goal is to discover the coefficients that minimize the cost, thus proposing an image feature that best fits our data.

This section laid the groundwork for the training and optimization process we have adopted for the coefficients of our image feature. The following section will present the inner workings of the cost function that drives this optimization. We will dissect how the cost function is formulated and how it aids in achieving an ideal interaction matrix for the selected image feature.

4.3.1 Cost Function

Creating a robust cost function is a pivotal step in optimizing our coefficients for image feature extraction in the context of robot movement control. The cost function provides a measure that quantifies the suitability of a set of coefficients toward achieving the objective.

We have introduced two distinct cost functions. The first one comprises three key components: Mean Squared Error (MSE), Difference Penalty, and Correlation Penalty. Each component serves a unique role, addressing a specific aspect of the problem, and collectively they guide the model towards optimal solutions.

In the following sections, these components will be introduced by outlining their formulations

and importance in the optimization process.

- **Mean Squared Error Term:**

  The purpose of the Mean Squared Error (MSE) term is central to our optimization strategy. We aim to adjust the coefficients of our image feature to make the resulting interaction matrix resemble the target interaction matrix (Table 4.1) as closely as possible.

  In simple terms, we desire meaningful values in the fourth or fifth elements (corresponding to R_x and R_y rotations), and other elements should strive to be as close to zero as possible.

Table 4.3: Mapping of optimized features to the targeted elements of the interaction matrix.

Optimized Feature	Targeted Element
R_x	4
R_y	5

Mathematically, for each example i, the interaction matrix can be computed as $\mathbf{L}_f^{(i)}$. By excluding the targeted element (as defined in Table 4.3) from $\mathbf{L}_f^{(i)}$ and $\mathbf{L}_f^*$ (the ideal interaction matrix as defined in Table 4.1), two 1-by-5 matrices can be computed as $\mathbf{L}_f^{'(i)}$ and $\mathbf{L}_f^{'*}$, respectively.

By subtracting $\mathbf{L}_f^{'(i)}$ from $\mathbf{L}^{'*}$, a vector is obtained that is then passed through the squared Euclidean norm (or the squared L2 norm), which squares each element, then sums them up, resulting in a scalar value that indicates the squared distance from the ideal interaction matrix. This value is then averaged over all examples in our dataset. The MSE equation for our case is given as:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N} ||\mathbf{L}_f^{'(i)} - \mathbf{L}_f^{'*}||^2, \tag{4.9}$$

where N is the total number of examples in our dataset, and the double bars denote the Euclidean norm.

The role of the MSE term is to ensure the elements are as close to zero as possible during the optimization process. Consequently, this steers the coefficients of the image feature towards values that deliver an interaction matrix closer to our target. However, securing that the

targeted element (Table 4.3) is significantly different from zero is equally crucial. This aspect is explained in depth in the following subsection, focusing on the Difference Penalty term in our cost function.

- **Difference Penalty Term:**

  It is essential that the targeted element (Table 4.3) stands out distinctly from the remaining elements in the interaction matrix. To ensure this, we introduce a penalty when the difference between the targeted element and the mean of the other elements is less than a predetermined minimum difference (δ_{min}).

  This penalty is computed as the positive part (ReLU) of δ_{min} minus the difference between the targeted element and the mean of other elements. This calculation implies that if the difference is less than δ_{min}, a penalty will be applied, increasing the overall cost. This encourages the optimization process to search for coefficients that generate a larger difference, thus avoiding the penalty.

  The equation for the Difference Penalty term is as follows:

$$\text{Difference Penalty} = \frac{1}{N}\sum_{i=1}^{N} \text{ReLU}\left(\delta_{min} - \left|\mathbf{L}^{(i)}_{f,targeted} - \frac{1}{5}\sum_{j=1, j\neq targeted}^{6} \mathbf{L}^{(i)}_{f,j}\right|\right), \tag{4.10}$$

  where $\mathbf{L}^{(i)}_{f,targeted}$ is the targeted element for the i-th example, $\mathbf{L}^{(i)}_{f,j}$ are the other interaction matrix elements for the same example, and the Rectified Linear Unit function (ReLU) is defined as below:

$$\text{ReLU}(x) = \max(0, x). \tag{4.11}$$

- **Correlation Penalty Term:**

  In the specific context of visual servoing, maintaining a strong linear relationship between the image feature value and the rotation angle (R_x or R_y) profoundly impacts the effectiveness of error computation and control signal generation. As we know, the control logic for the robot

is heavily reliant on comparing the current feature value with the desired one. The difference, termed as error, forms the basis for the controller signal that directs the robot's movement.

This error computation assumes that the feature value and the rotation angle exhibit a linear relationship. If this assumption holds, a positive error signifies the need for the end-effector to move in a specific direction. In contrast, a negative error implies a movement in the opposite direction.

Correlation between two variables can be defined mathematically as follows:

$$\rho = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2(y_i - \bar{y})^2}}, \tag{4.12}$$

where ρ is the correlation, x_i and y_i are the values of the two variables for each sample i, and $\bar{x}$ and $\bar{y}$ are the means of x and y, respectively.

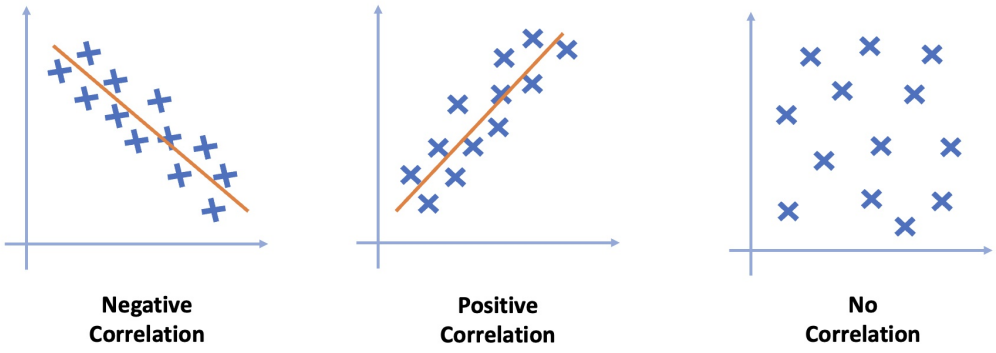


Figure 4.2: Strength and direction of correlation between co-variables

In the context of our optimization, x_i corresponds to the feature values and y_i to the R_x or R_y angles. We aim to maximize the absolute value of this correlation.

With this understanding, the correlation penalty term aims to encourage solutions with a high absolute correlation, formulated as:

$$\text{Correlation Penalty} = \frac{1}{N}\sum_{i=1}^{N}(1 - |\rho|), \tag{4.13}$$

where $|\rho|$ is the absolute correlation between the feature values and the R_x or R_y values. This penalty term increases as the absolute correlation decreases, thus encouraging solutions that result in a strong linear relationship between the feature values and the corresponding robot movement.

- **Overall Cost Function (First definition):**

  In the function-based visual servoing, the overall cost function plays a central role. It combines three essential parts: the Mean Squared Error (MSE), the Difference Penalty, and the Correlation Penalty. Each of these parts has a specific job. The MSE makes sure non-targeted parts of the interaction matrix are near zero. The Difference Penalty ensures a good gap between the targeted element and the average of the rest. Lastly, the Correlation Penalty term emphasizes the necessity of a linear relationship between the image feature value and the R_x or R_y rotation values.

  These three parts, each having their unique weights, add up to make the total cost function. The equation is as follows:

$$\text{Total Cost} = \text{MSE} + \lambda_1 \times \text{Difference Penalty} + \lambda_2 \times \text{Correlation Penalty}, \tag{4.14}$$

  where λ_1 and λ_2 are the weights assigned to Difference Penalty and Correlation Penalty respectively.

  The total cost is a single number that reflects how well our image feature is doing. By finding the best coefficients that make this total cost as small as possible, we get an image feature that does a great job at helping control the robot. This total cost function is the heart of our work, including the different goals we're trying to reach.

- **Overall Cost Function (Second definition):**

  The second cost function is designed with the intent of making the interaction matrix closer to the ideal one (Table 4.1). We introduced a threshold to penalize elements of the interaction matrix that deviate from their ideal values. Specifically, if the difference between an

element of the interaction matrix and the corresponding element of the ideal matrix exceeds the threshold, a penalty is applied that increases exponentially with the difference.

For each data example j in the dataset and for each element i of its interaction matrix, the penalty function is defined as:

$$\phi(i) = \begin{cases} |\mathbf{L}_f^{(i)} - \mathbf{L}_f^{*(i)}| & \text{if } |\mathbf{L}_f^{(i)} - \mathbf{L}_f^{*(i)}| \leq \delta \\ \lambda_3 \times e^{\eta(|\mathbf{L}_f^{(i)} - \mathbf{L}_f^{*(i)}| - \delta)} & \text{otherwise} \end{cases}. \tag{4.15}$$

Here, $\mathbf{L}_f^{(i)}$ represents the i^{th} element of the interaction matrix, while $\mathbf{L}_f^{*(i)}$ is the i^{th} element of the target interaction matrix (see Table 4.1). The variable δ specifies the threshold, the parameter η is the scale for the exponential increase, and λ_3 provides the penalty weight for elements whose differences surpass the threshold.

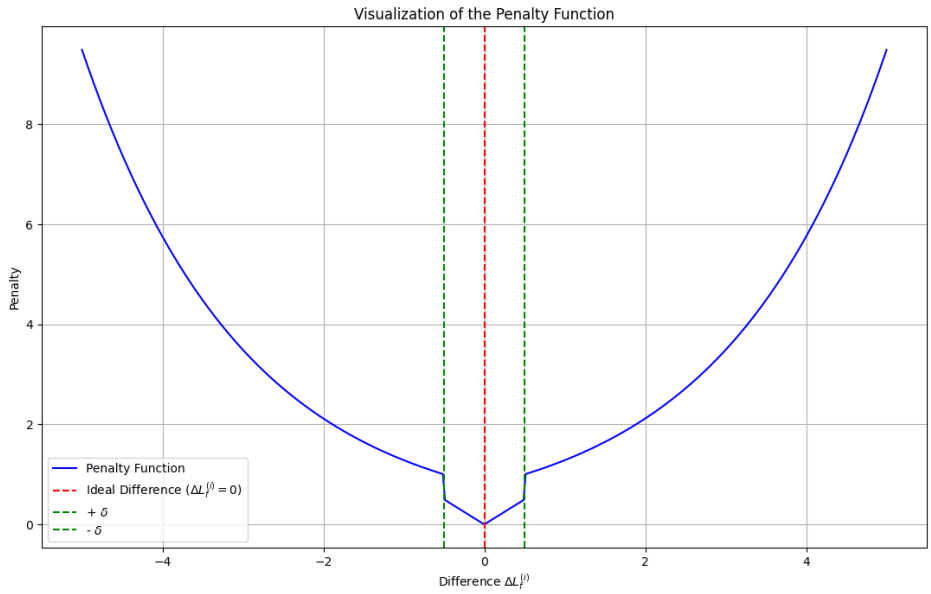


Figure 4.3: Visualization of the penalty function with parameters: $\delta = 0.5$, $\lambda_3 = 1$, and $\eta = 0.5$.

As depicted in Figure 4.3, the penalty rises sharply once the difference surpasses the threshold, showing our intention to heavily penalize large deviations from the ideal values.

The overall cost is the average penalty across all data examples and is given by:

$$\text{Total Cost} = \frac{1}{N}\sum_{j=1}^{N}\left(\sum_{i=1}^{6}\phi(i)^{[j]}\right). \tag{4.16}$$

In the above equation, $\phi(i)^{[j]}$ represents the penalty associated with the i^{th} element of the interaction matrix for the j^{th} data.

4.3.2 Hyperparameter Tuning

The process of optimization is a deliberate and iterative task. It depends heavily on a trial and error mechanism, where each iteration and its outcome provide insights into the next possible step. As we explore a model, the 'hyperparameters' are our guides. They modulate the learning process of the model and determine its trajectory toward finding an optimal set of parameters that can accurately represent the underlying relationship within the data.

Researchers often tune these hyperparameters to adjust the learning and adaptation of the model. While the primary objective is to reach the lowest cost function value, ensuring that the model generalizes well and does not just memorize the training data is essential. Thus, hyperparameter tuning becomes crucial in achieving an effective balance between these needs.

For the aforementioned cost functions, the following hyperparameters have been identified and carefully tuned:

Table 4.4: Hyperparameters used in the optimization process.

Hyperparameter	Symbol
Number of epochs	N
Learning Rate	α
Difference Penalty Weight	λ_1
Correlation Penalty Weight	λ_2
Minimum Difference	δ_{min}

Hyperparameter	Symbol
Number of epochs	N
Learning Rate	α
Threshold	δ
Exponential Scale	η
Penalty Weight	λ_3

- **Number of epochs (N)**: This parameter defines the number of times the complete dataset is passed forward and backward through the optimization algorithm. An appropriate number of epochs ensures that the model learns the patterns in the data without overfitting.

- **Learning Rate (α)**: This hyperparameter controls the step size at each iteration while moving

toward a minimum of a cost function. A suitable learning rate allows the optimizer to reach the minimum effectively.

- **Difference Penalty Weight** (λ_1): This hyperparameter weights the influence of the difference penalty term in the total cost. It's a balancing factor to adjust the importance of this term relative to others.

- **Correlation Penalty Weight** (λ_2): Similarly, this hyperparameter weights the influence of the correlation penalty term in the total cost, balancing its importance.

- **Minimum Difference** (δ_{min}): This parameter sets the threshold for the minimum acceptable difference between the targeted element of the interaction matrix and the mean of the other elements. It's crucial for implementing the difference penalty term in our cost function.

- **Threshold** (δ): This parameter sets the boundary beyond which the deviation of an interaction matrix element from its ideal value is considered significant. If the difference surpasses this threshold, a heavier penalty is applied, encouraging the model to closely match the ideal values.

- **Exponential Scale** (η): This hyperparameter determines the rate at which the penalty increases once the threshold δ is surpassed. A higher value of η will result in a steeper increase in the penalty for deviations beyond the threshold.

- **Penalty Weight** (λ_3): This hyperparameter weights the influence of the interaction matrix's deviation penalty term in the total cost. It adjusts the importance of penalizing deviations from the ideal interaction matrix relative to other terms in the cost function.

Tuning these hyperparameters requires careful consideration and numerous iterations. Observing the cost at each step and how it evolves through different iterations is a valuable guide to finding a set of hyperparameters that allows the cost functions to reach their minimum.

4.4 Optimization Results

Through iterative testing and adjustments, we fine-tuned the hyperparameters to get interaction matrices aligning closely with our desired ones (Table 4.1). The specifics of the adjusted hyperparameters are provided in Table 4.5.

Table 4.5: Tuned hyperparameters used in the optimization process.

Hyperparameter	Value
Learning Rate (α)	10^{-2}
Difference Penalty Weight (λ_1)	10^3
Correlation Penalty Weight (λ_2)	10
Minimum Difference (δ_{min})	1

Hyperparameter	Value
Learning Rate (α)	10^{-2}
Threshold (δ)	10^{-1}
Exponential Scale (η)	10^{-2}
Penalty Weight (λ_3)	10^3

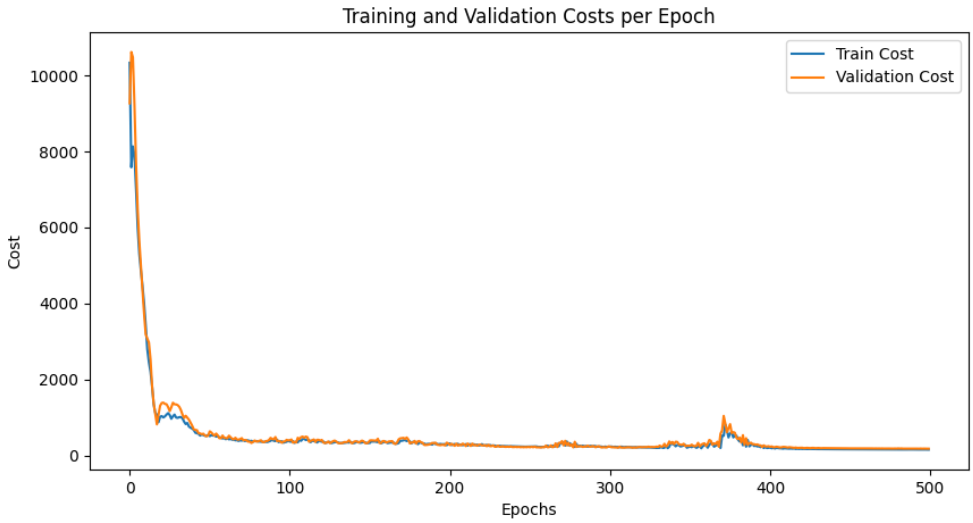
Every training ran for 500 epochs because the cost showed a small change beyond this point. We selected the best-performing model using the least validation cost, ensuring that the model did not just memorize the data but actually learned from it.

4.4.1 Cost Function I

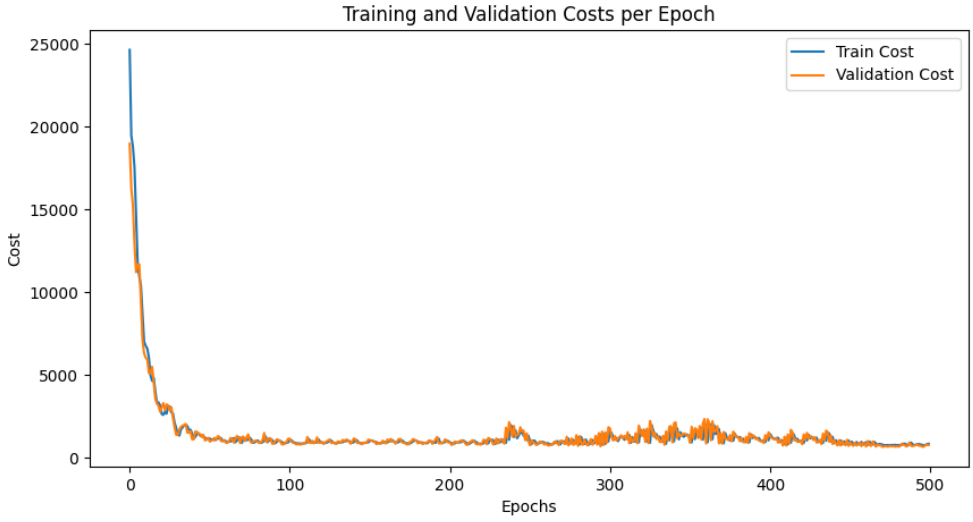
For the initial definition of the cost function (see Equation (4.14)), we optimized the image feature function (Equation (4.1)) twice (separately for the rotational degrees of freedom around the x-axis (R_x) and the y-axis (R_y)). We have plotted the progression of the cost for both the training and validation sets for R_x and R_y in Figures 4.4a and 4.4b, respectively.

The plots show that towards the end of the training, the reduction in cost plateaued, representing minimal improvement in the final epochs. The minimal values of the cost for R_x and R_y on the test dataset are 168.83 and 817.75, respectively. For a more detailed understanding, we have broken down the cost into its components: Mean Squared Error (MSE), Difference Penalty, and Correlation Penalty. These values are presented in Table 4.6 where they are also compared with the well-known features derived by S. Liu et al. (2009) as shown in Equation (2.50).

When reviewing Table 4.6, we observe that our optimization method yields lower cost components compared to the Liu features, except for the Correlation Penalty of the R_y feature. Even though these numerical results are promising, the actual performance during experiments did not meet the expectations. The velocities computed using the optimized features did not make the end



(a) R_x



(b) R_y

Figure 4.4: Cost over epoch for both R_x and R_y features (with the first cost definition)

Table 4.6: Cost values comparison on the test set (First cost definition)

Element	Method	Cost Value	MSE	Difference Penalty	Correlation Penalty
R_x	Liu	15691.43	766.69	14.915	0.941
	Optimization	168.83	12.97	0.147	0.921
R_y	Liu	4663.08	800.46	3.857	0.551
	Optimization	817.75	2.17	0.806	0.940

effector converge to the desired pose. Modifying the cost function or increasing the weight of the Correlation Penalty may be a potential solution to this issue. Therefore, the image feature is optimized with the next cost definition.

4.4.2 Cost Function II

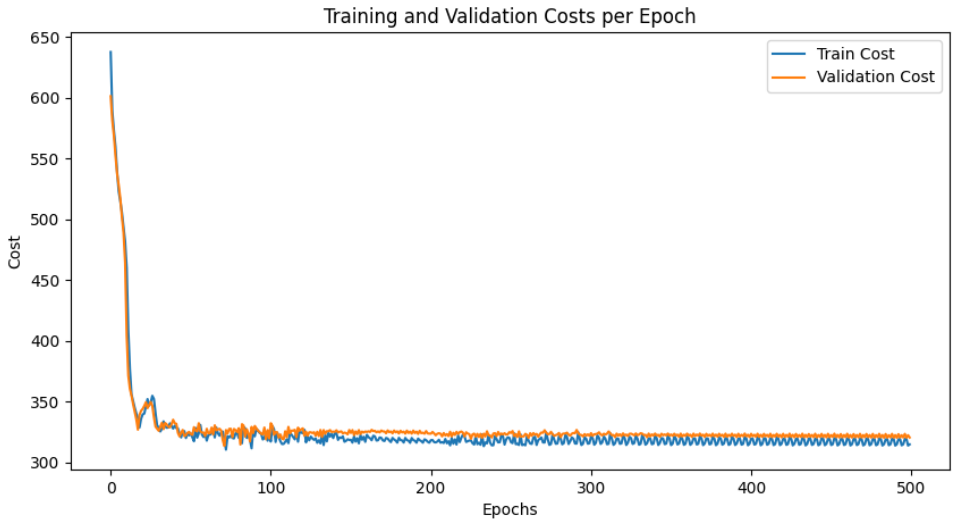
As with the initial cost function definition, we optimized the image feature for both the rotational component around the x-axis (R_x) and y-axis (R_y). The cost over epoch for the training and validation datasets are illustrated in Figures 4.5a and 4.5b.

Furthermore, we compared the cost value of the optimized features on the test set with those derived from Liu's method in Table 4.7. Given the particular definition of the penalty function (Equation (4.15)), direct comparisons to measure improvements can be challenging. Consequently, we also calculated the Mean Absolute Error (MAE) for the interaction matrices on the test dataset, which is detailed in the last column of Table 4.7.

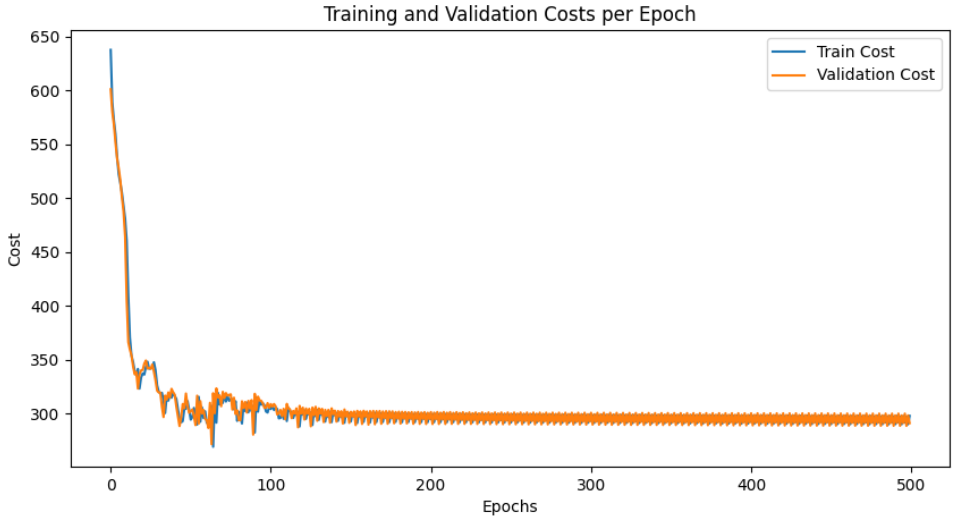
Table 4.7: Cost values comparison on the test set (Second cost definition)

Element	Method	Cost Value	Mean Absolute Error
R_x	Liu	424098.34	9.776
	Optimization	357.68	0.433
R_y	Liu	2.52×10^{26}	19.176
	Optimization	300.36	0.293

The computed cost for Liu's method reaches high numbers due to the particular formulation of the penalty function, which applies an exponential penalty on differences exceeding a certain threshold. On the other hand, the MAE provides a clearer metric for comparison, revealing a significant improvement in the interaction matrices of the test set, with both MAEs under 0.5. Despite these promising numerical improvements, the velocities calculated by these optimized image features



(a) R_x



(b) R_y

Figure 4.5: Cost over epoch for both R_x and R_y (with the second cost definition)

resulted in divergence.

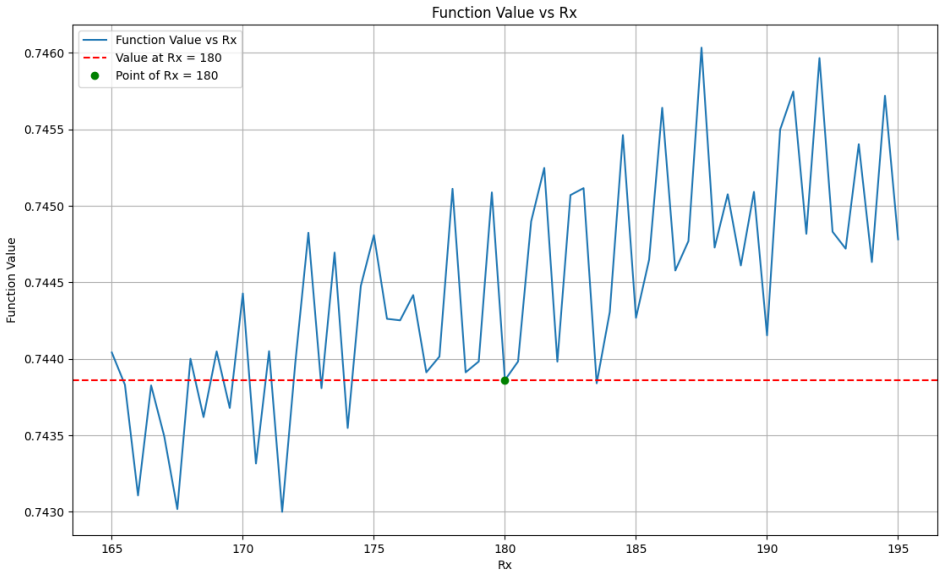
This observation emphasizes the potential for further improvements in this approach, which we intend to discuss in future works chapter (Section 6.2). In our continuous endeavour to develop decoupled image features, we shift our focus towards a novel method named deep neural network-based visual servoing, aiming for another attempt in image feature decoupling."

4.5 Simulation Results

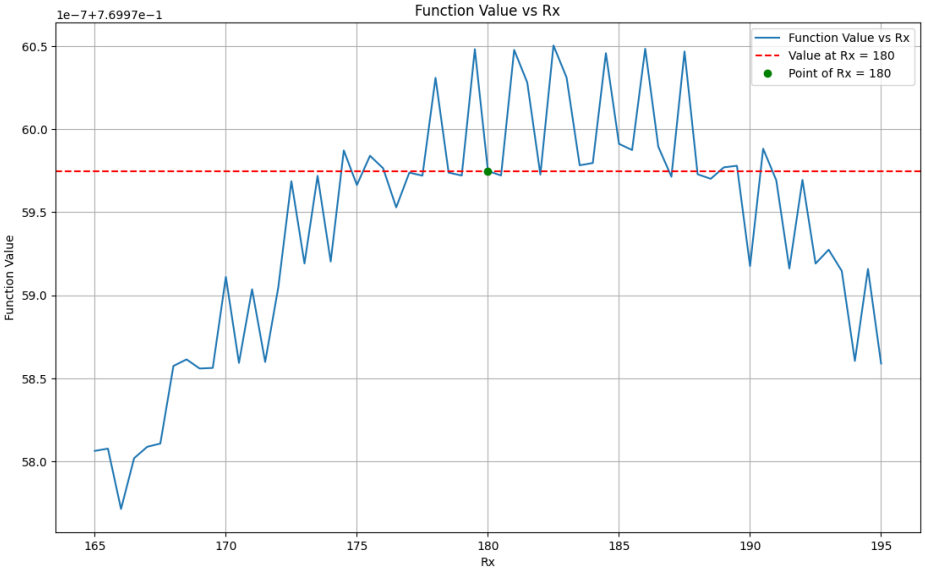
After training Equation (4.1) using two different cost functions, we use a simulation environment (as discussed in Section 3.2) to validate the effectiveness of our final models. Initially, we conducted a simulation to observe the functions' outputs with optimized parameters. Specifically, the manipulator's end effector was fixed at a constant position, 40 cm directly above the targeting pin, which itself was centered and parallel to the image plane. The end effector was then systematically rotated about the x-axis from 165 to 195 degrees, effectively covering a range of 30 degrees centered at 180 degrees. The process was performed in 0.5-degree increments, and the corresponding outputs of the trained function were recorded for cost functions I and II. A similar procedure was followed for rotations about the y-axis, where the end effector was rotated from -15 to 15 degrees, centered at 0 degrees.

The results of these simulations are illustrated in Figures 4.6 and 4.7, which graphically depict the relationship between the function 4.1's output and the rotational angles R_x and R_y, respectively. These figures provide a clear visualization of how the function values vary with changes in the end effector's orientation along the specified axes.

The analysis of these simulation results has shown a few key observations. In particular, Figure 4.6a displays a semi-linear trend, suggesting a positive correlation although there are considerable oscillations. This indicates a certain degree of predictability and linearity in the response which is a favorable attribute for visual servoing. In contrast, Figures 4.6b,4.7a and 4.7b show plots that are somewhat symmetrical around their central values. This symmetry poses a control issue since deviations from the center in either direction produce errors of the same sign. Such a scenario can result in divergence in at least one direction based on the sign chosen for the controller.

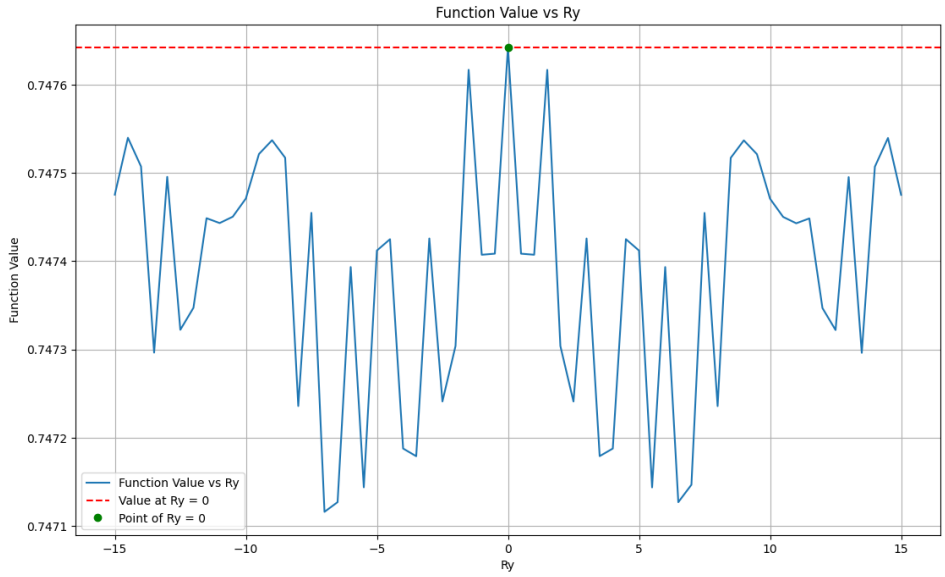


(a) Cost Function I

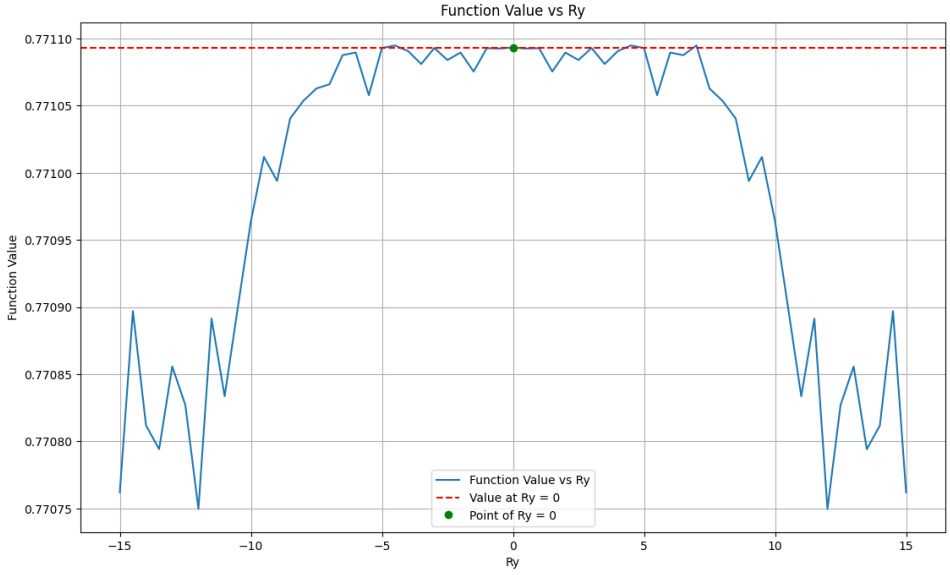


(b) Cost Function II

Figure 4.6: Rotational Image Feature about x-axis (Function 4.1) Value vs R_x angle



(a) Cost Function I



(b) Cost Function II

Figure 4.7: Rotational Image Feature about y-axis (Function 4.1) Value vs R_y angle

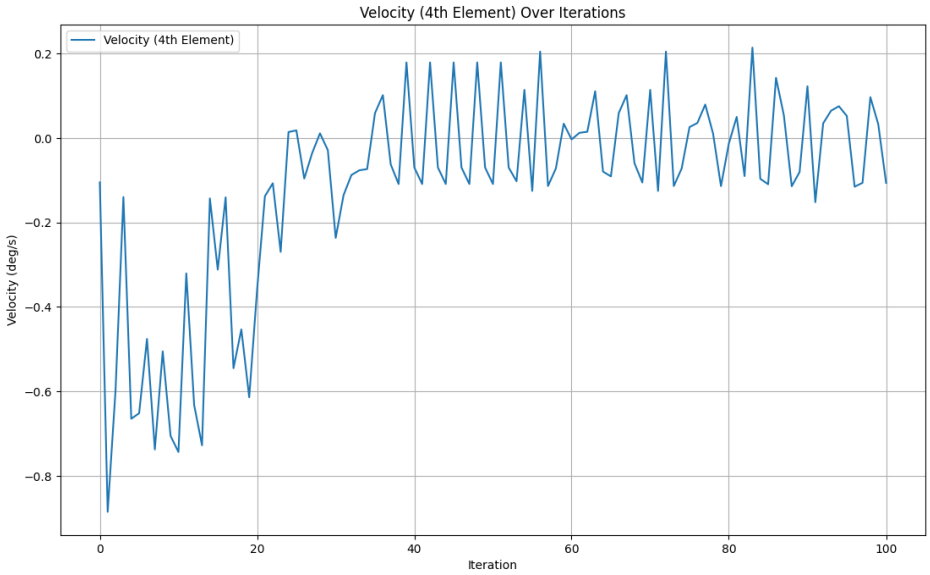
Drawing horizontal lines on Figures 4.6 and 4.7 will intersect the graph at multiple points. Therefore, there are several end effector angles corresponding to the same function value. Therefore, these positions produce a zero error, indicating that they are not distinguishable from each other using this method. This limitation significantly challenges the control strategy's precision and effectiveness, as it limits the ability to uniquely determine and correct the end effector's position.

Then, we focused our experiments on evaluating the control effectiveness of our method under highly constrained conditions. We specifically examined the velocity generated by the function-based visual servoing approach, while intentionally excluding errors associated with other degrees of freedom and image features. To do this, we established the desired pose at the same position used in the previous tests (40 cm above the targeting pin, centered and parallel to the image plane). We then introduced 10 degree deviations from this pose as initial conditions in both R_x and R_y experiments. The results of these experiments were plotted for both cost function definitions applied to the R_x and R_y features, focusing on velocity and pose error over iterations. These plots are illustrated in Figures 4.8 through 4.11. These figures provide insights into the behavior and response accuracy of the system under the stated constraints, providing a focused perspective on the individual contributions of R_x and R_y movements to the overall control performance.

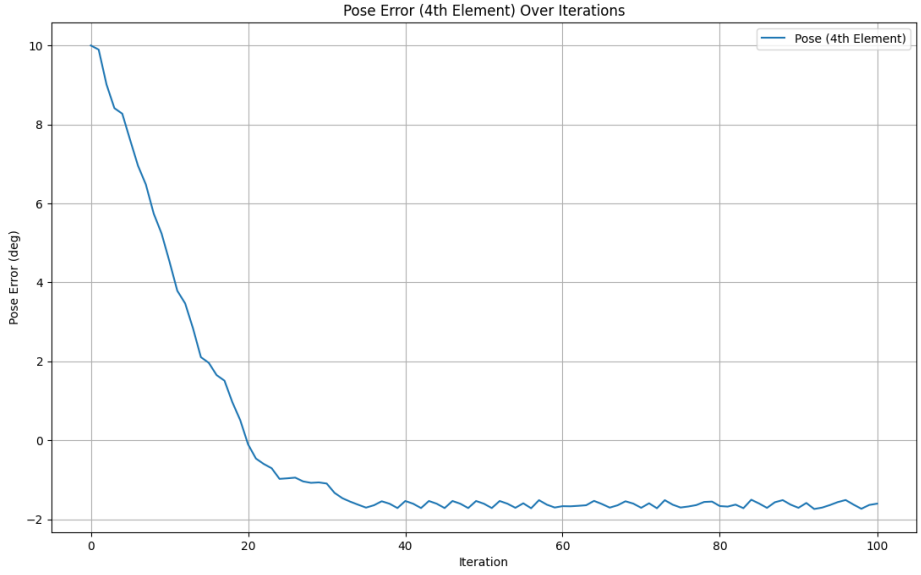
Upon detailed analysis of the results, distinct behaviors were observed in the system's response based on the applied cost function definition. Specifically, in the case of the second cost definition for the R_x feature, the system exhibited persistent oscillations without converging to the desired pose. This phenomenon is clearly illustrated in Figure 4.9, where the error margin did not diminish to zero, indicating an unsuccessful convergence.

In contrast, there was a notable difference in the response for the experiment of the first cost definition of R_x feature and for both cost definitions of R_y feature. In these cases, the end effector converged to a pose that was close to the desired pose, indicating the possibility that the function value may have reached a local minimum in the vicinity of the desired pose. At this local minimum, the function value matched the desired value, resulting in a convergence.

In conclusion, using the trained functions under the aforementioned constraints proved to be unsatisfactory. Consequently, employing these functions on the actual manipulator (Denso), in addition to features for other movements to control 6 DOF of the robot, could be risky. Therefore,

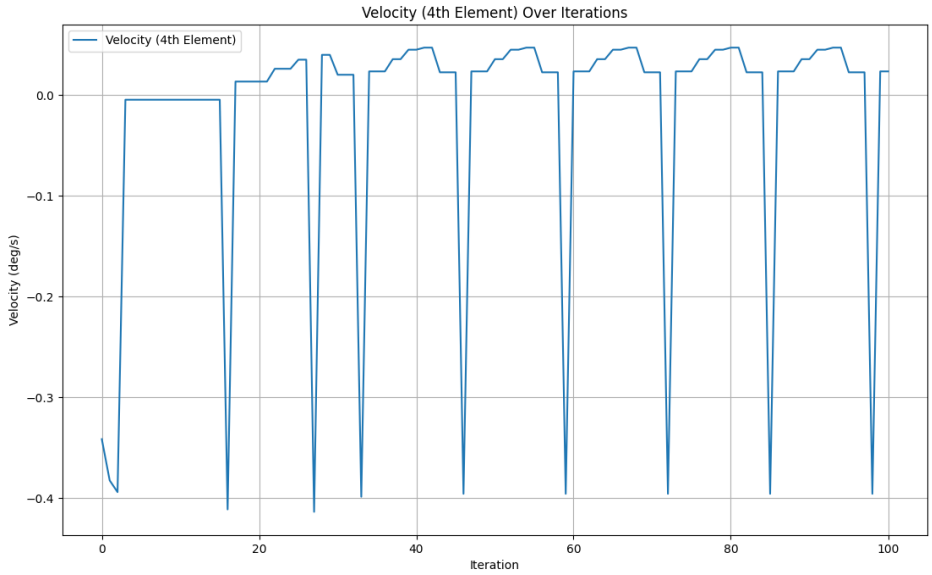


(a) Velocity over Itarations

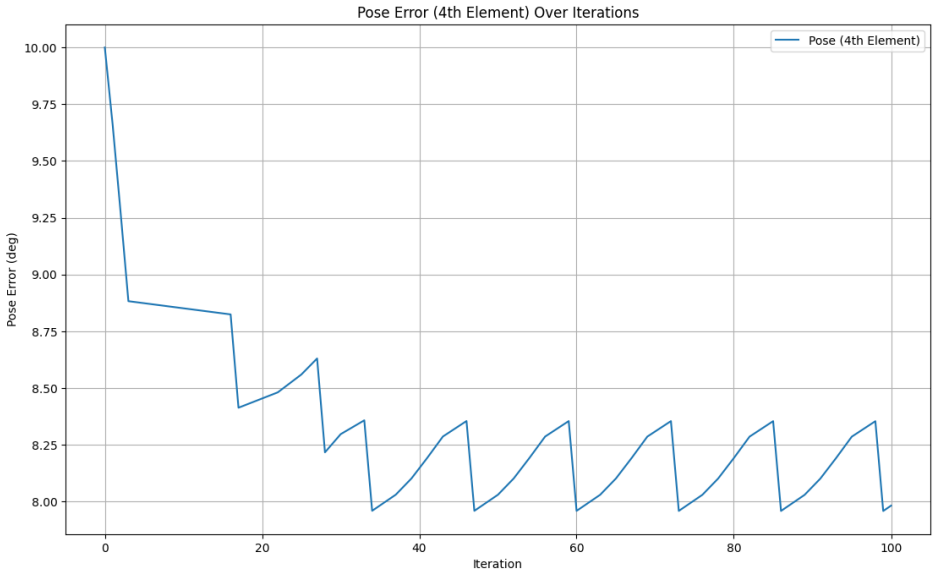


(b) Pose Error over Iterations

Figure 4.8: R_x Cost Function I

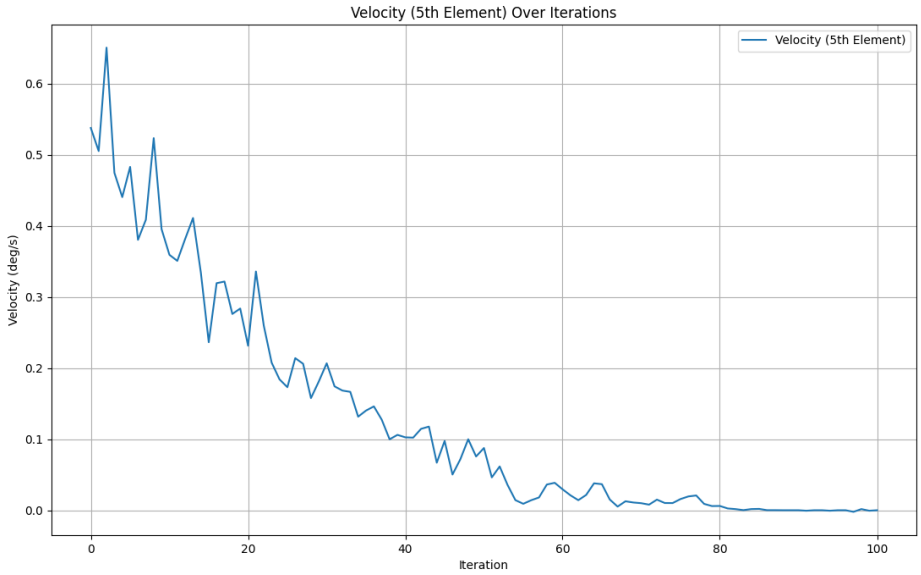


(a) Velocity over Itarations

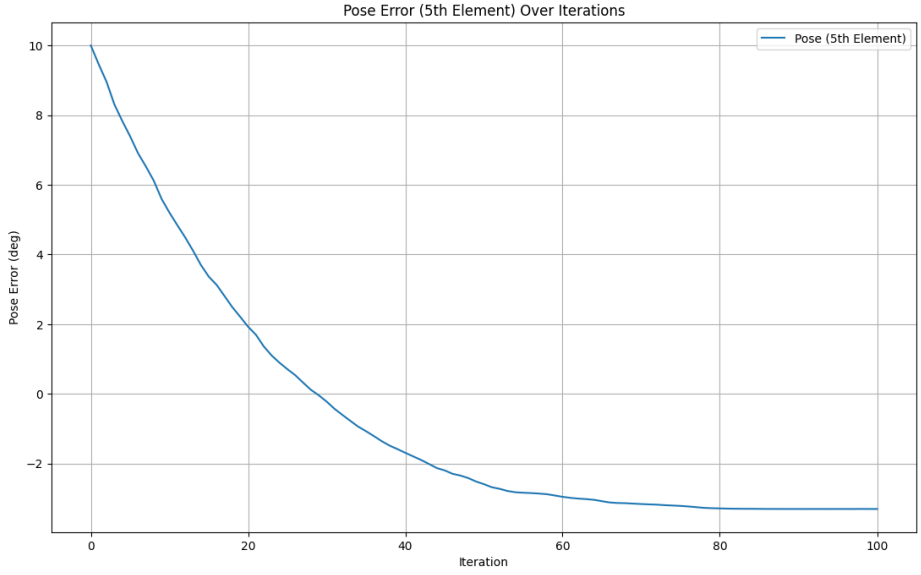


(b) Pose Error over Iterations

Figure 4.9: R_x Cost Function II

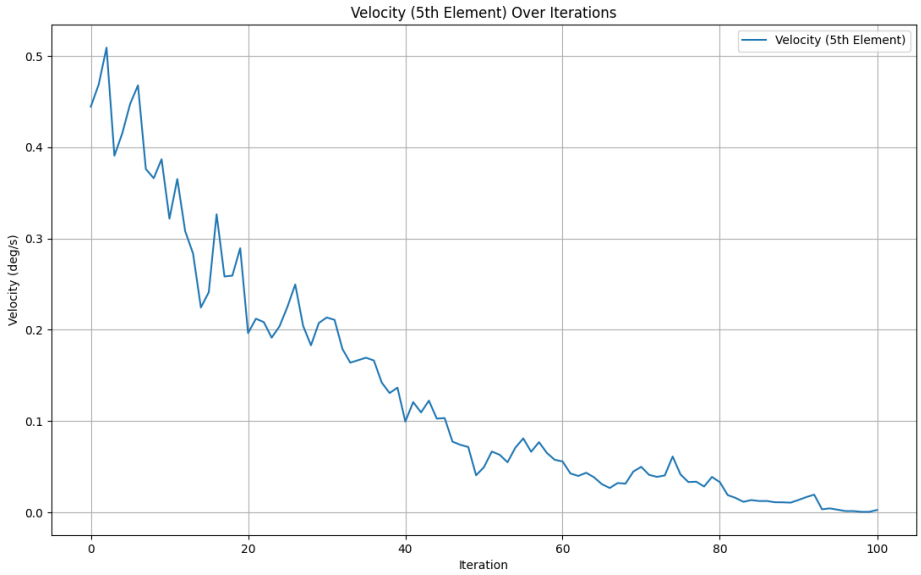


(a) Velocity over Itarations

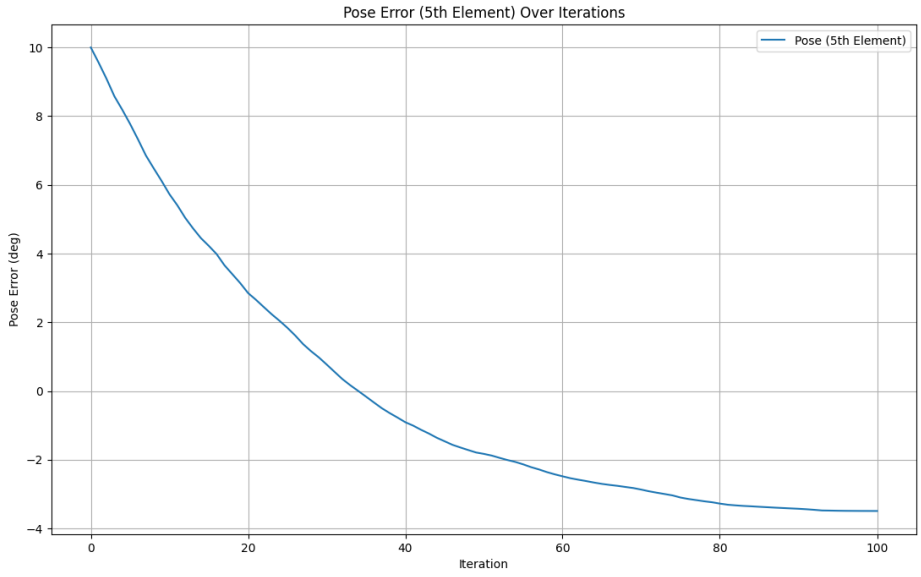


(b) Pose Error over Iterations

Figure 4.10: R_y Cost Function I



(a) Velocity over Itarations



(b) Pose Error over Iterations

Figure 4.11: R_y Cost Function II

we will move on to the next approach, DNN-based visual servoing, to solve the coupling problem.

4.6 Summary

This chapter is the basis for the complicated process of decoupling image features through a function-based method.

A novel approach is introduced in Section 4.1, where we optimized the parameters of a fraction (considered as the image feature function) for the rotational components R_x and R_y. In Section 4.3, we initially describe the optimization algorithm and subsequently define two distinct cost functions, each designed to emphasize the attributes of a decoupled image feature. The objective was to tune the function's parameters to minimize these cost functions.

Finally, in Section 4.4, we present the results of the training phase and the cost metrics for the optimized features on the test set. As discussed in Section 4.5, although there were numerical improvements in the cost values, these optimized features did not perform successfully in the experiments.

Chapter 5

Deep Neural Network-based Visual Servoing

The objective of this study is to develop perfectly decoupled image features. However, Chapter 4 investigated function-based image features which did not deliver satisfactory performance. As a result, to continue our search for perfectly decoupled image features, an artificial intelligence technique, i.e. DNN, has been explored due to its universal approximation capability.

The following sections define the proposed image features and describe the details of the procedures for designing and training a neural network model. This in-depth coverage will explain how DNNs can be employed to achieve perfectly decoupled image features.

5.1 Image Feature Definition

Imagine our set of image features represented as

$$s = \begin{bmatrix} c_x x \\ c_y y \\ c_z z \\ c_\beta \beta \\ c_\gamma \gamma \\ c_\alpha \alpha \end{bmatrix}, \tag{5.1}$$

where each feature has a linear correlation with the pose of the camera with respect to the object. If such a linear relationship exists, the corresponding 6×6 interaction matrix $\mathbf{L}_s$ simplifies into a perfect diagonal matrix as follows:

$$\mathbf{L}_s = diag(c_x, c_y, c_z, c_\beta, c_\gamma, c_\alpha). \tag{5.2}$$

Under ideal conditions where the pose is precisely estimated, the interaction matrix becomes the identity matrix.

$$\mathbf{L}_s = I_6. \tag{5.3}$$

A robust tool for this type of estimation is Deep Neural Networks. The input to these networks is a set of moments, central moments, and possibly some engineered features. The output from these networks aims to predict the camera's six-dimensional (6D) pose. A graphical illustration showcasing the neural network's architecture, detailing its inputs and outputs, is presented in Figure 5.1.

5.2 Hyperparameter Tuning

Training a Deep Neural Network (DNN) involves many choices, and one of the most critical decisions involves setting the hyperparameters. These settings, such as the learning rate and activation function, shape the learning process and ultimately determine how well the model performs. A poorly-tuned set of hyperparameters can result in a model that either does not learn effectively or one

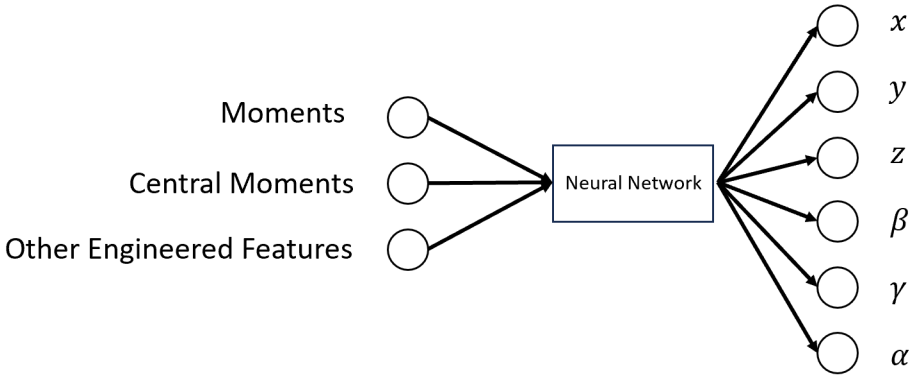


Figure 5.1: Neural network's inputs & outputs

that overfits the training data. Therefore, hyperparameter tuning is an essential step in optimizing our DNN model.

In our search for optimal model performance, we explored several hyperparameters:

- **Activation Functions:** These are mathematical expressions that determine the output of a node in our network. We considered various options, including 'Relu', 'Leaky Relu', 'Tanh', and 'Sigmoid' (Figure 5.2).

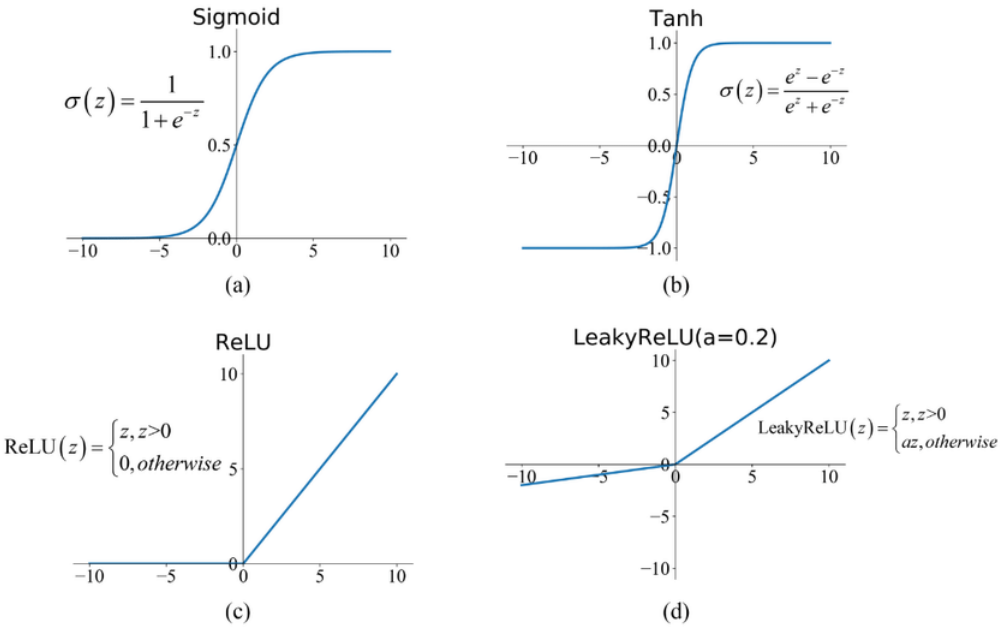


Figure 5.2: Most common activation functions (Leppich, 2021)

- **Batch Size:** This refers to the number of training examples utilized in one iteration. We

explored a range from 32 to 512.

- **Network Architecture:** The layout of neurons and layers in our DNN is a significant design choice. We varied the number of layers, as well as the number of units within each layer, to experiment with both shallow and deep architectures.

- **Learning Rate:** This hyperparameter determines the step size at each iteration while moving towards a minimum of the loss function. We considered values of 10^{-2}, 10^{-3}, and 10^{-4}.

- **Optimizers:** These are algorithms or methods used to adjust model parameters to minimize the model error. We looked into two options: Adam and Adamw (Adam weight decay).

Given the vast hyperparameter space and the computational cost associated with exhaustively exploring every combination, we needed an efficient strategy like random search. Instead of trying out every possible hyperparameter combination (a method known as grid search), random search samples a fixed number of hyperparameter combinations from the total pool. This strategy provides a balanced approach between computational efficiency and the broadness of exploration, increasing the probability of finding a near-optimal set of hyperparameters.

To implement our random search, we defined a function, '$build\_model$', that constructs a model based on a given set of hyperparameters. We then used the '$RandomSearch$' method from Keras, which iteratively builds and evaluates models using different hyperparameters drawn randomly from the specified ranges. After evaluating a predefined number of combinations, the best hyperparameters are chosen based on their performance on the validation dataset.

Below is a table summarizing the hyperparameter values that were found to be most effective:

Table 5.1: Optimal Hyperparameter Values from Random Search

Hyperparameter	Optimal Value
Activation Function	ReLU
Batch Size	512
Number of Hidden Layers	3
Number of Units for each Hidden Layer	80, 224, 112
Learning Rate	1×10^{-3}
Optimizer	Adam

By carefully tuning these hyperparameters, we have created an efficient and effective model.

5.3 Deep Neural Network Architecture

The design of a deep neural network (DNN) is often an iterative, trial-and-error process. We can continually improve the system's performance by refining the input features and modifying the number of neurons and layers. The initial phase of training started with an architecture that was recommended based on the random search. This suggested a three-layer network with 80, 224, and 112 neurons for each layer, respectively.

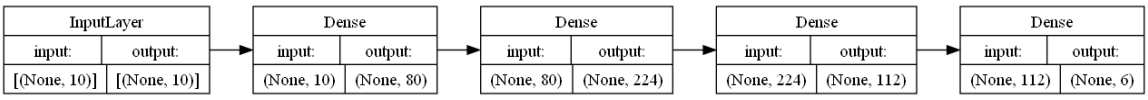


Figure 5.3: Initial DNN architecture based on random search.

After around 5,000 epochs, it is noted that the system performance plateaued. Specifically, the predictions for both translational and rotational movements were not at the desired accuracy. The mean absolute errors (MAE) for the best model (selected based on validation loss) were:

Table 5.2: Mean Absolute Errors for the initial model.

Output	Mean Absolute Error
1	12.09 (mm)
2	8.62 (mm)
3	6.99 (mm)
4	2.02°
5	2.38°
6	1.57°

To enhance the translational and rotational estimates, the following measures were taken:

Rotational: Five new inputs are introduced. Among these, one is the renowned 'α' (Equation (2.39)), commonly adopted in academia for controlling the final degree of freedom of a robot, closely related to R_z. The other four, labelled c_1 to c_4, are moment invariants proposed by Tahri and Chaumette (2005). These are invariants to 2D translation, 2D rotation, and scaling and were also used by Zhao et al. (2012) for estimating R_x and R_y angles. The definitions of c_1 through c_4 are as

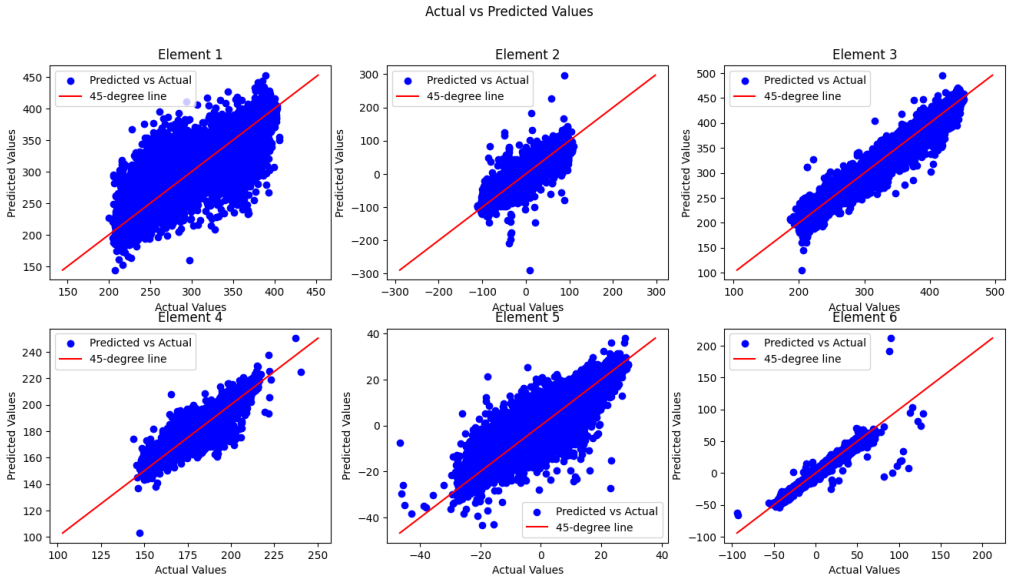


Figure 5.4: Actual vs Predicted for all six elements using the initial model.

follows:

$$
\begin{cases}
I_1 &= -\mu_{20}\mu_{02} + \mu_{11}^2 \\
I_2 &= (\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2 \\
I_3 &= (\mu_{30} - 3\mu_{12})^2 + (3\mu_{21} - \mu_{03})^2 \\
I_4 &= (\mu_{30} + \mu_{12})^2 + (\mu_{21} + \mu_{03})^2 \\
I_5 &= -\mu_{30}^2\mu_{03}^2 + 6\mu_{30}\mu_{21}^3 - 4\mu_{30}\mu_{12}^3 - 4\mu_{21}^3\mu_{03} + 3\mu_{21}^2\mu_{12}^2 \\
I_6 &= 3\mu_{30}^2\mu_{12}^2 + 2\mu_{30}^2\mu_{03}^2 - 6\mu_{30}\mu_{21}^2\mu_{12} - 6\mu_{30}\mu_{21}\mu_{12}\mu_{03} \\
&\quad + 2\mu_{30}\mu_{12}^3 + 3\mu_{21}^4 + 2\mu_{21}^3\mu_{03} + 3\mu_{21}^2\mu_{03}^2 - 12\mu_{21}\mu_{12}^2\mu_{03} + 6\mu_{12}^4 \\
I_7 &= -\mu_{30}^3\mu_{03} + 3\mu_{30}^2\mu_{21}\mu_{12} - 2\mu_{30}\mu_{21}^3 - 3\mu_{30}\mu_{21}^2\mu_{03} \\
&\quad + 6\mu_{30}\mu_{21}\mu_{12}^2 + 3\mu_{30}\mu_{12}^2\mu_{03} + \mu_{30}\mu_{03}^3 - 3\mu_{21}^3\mu_{12} \\
&\quad - 6\mu_{21}^2\mu_{12}\mu_{03} + 3\mu_{21}\mu_{12}^3 - 3\mu_{21}\mu_{12}\mu_{03}^2 + 2\mu_{12}^3\mu_{03}
\end{cases}
\qquad
\begin{cases}
c_1 &= \frac{I_1}{I_2} \\
c_2 &= \frac{I_3}{I_4} \\
c_3 &= \frac{I_5}{I_6} \\
c_4 &= \frac{I_7}{I_6}
\end{cases}
\tag{5.4}
$$

Translational: For translational movements, the network is separated into two parallel sub-networks, one each for translation and rotation. The translational sub-network was further deepened.

A visualization of the architecture is depicted below:

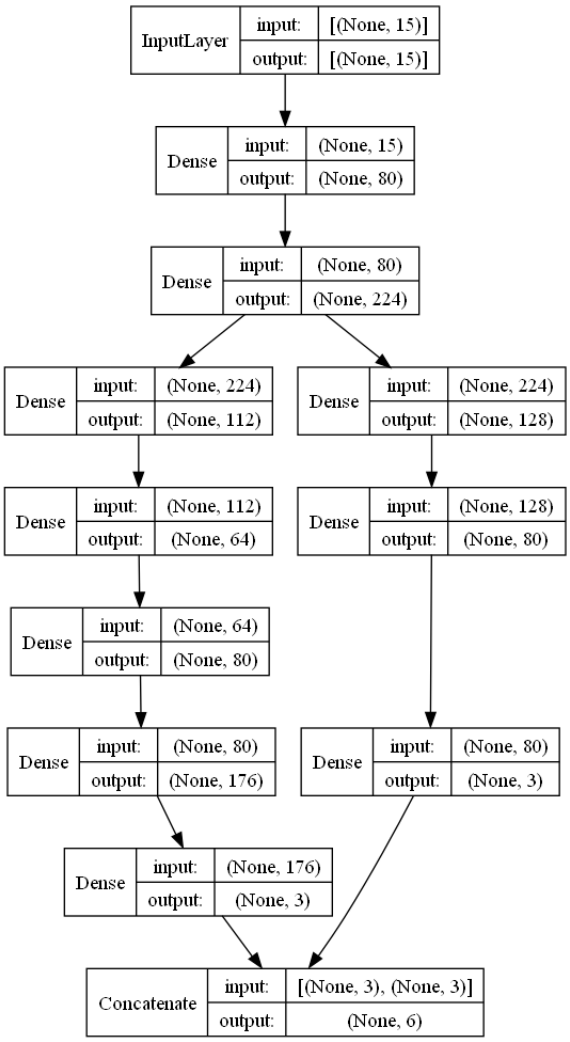


Figure 5.5: Final DNN architecture

The resulting test set's MAE (as presented in Table 5.3) showcased significant improvements.

5.4 Deep Neural Network Algorithm

Training a Deep Neural Network (DNN) is a complex yet essential task in machine learning research and applications. The objective of a DNN is to learn meaningful representations of input data, and the algorithm's capability to do so largely depends on an iterative optimization process. In this section, we'll discuss the intricate steps of this process, shedding light on how we ensure

Table 5.3: Improved Mean Absolute Errors after modifications.

Output	Mean Absolute Error
1	7.45 (mm)
2	5.82 (mm)
3	4.18 (mm)
4	1.32°
5	1.78°
6	1.02°

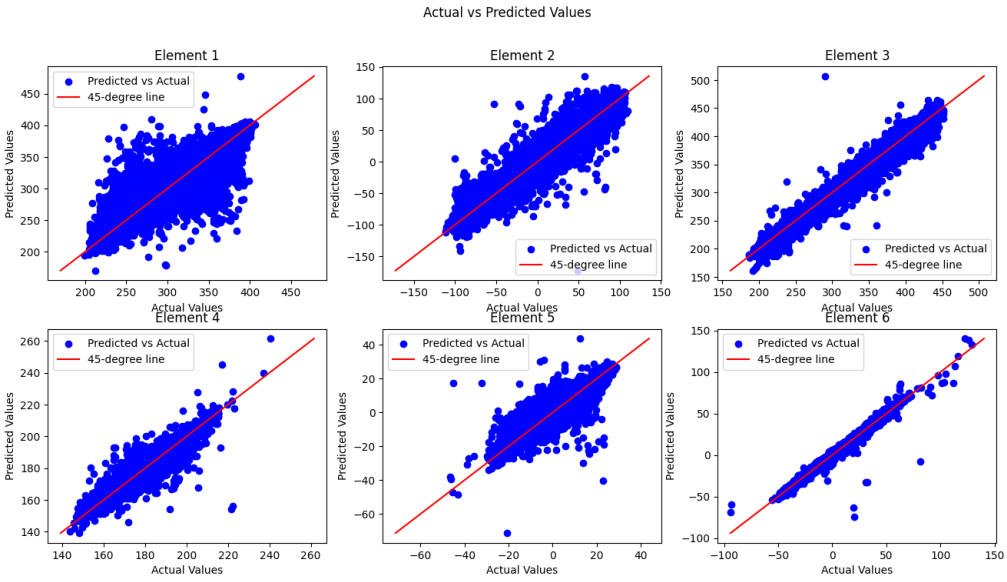


Figure 5.6: Actual vs Predicted for all six elements after modifications.

efficient and effective learning for our DNN model. The steps include Data Preprocessing, Initialization, Hyperparameter Setting, Model Training, and Model Selection. Each step has its own significance, and together they build the foundation for a successful learning model.

(1) **Data Preprocessing:** In the Data Preprocessing phase, the data is thoroughly prepared to ensure optimal compatibility with Deep Neural Network (DNN) training. This dataset, containing distinct elements such as '$c\_moments$', '$moments$', and '$pose$', undergoes several transformation processes to become effectively ingestible by the DNN.

The input data is prepared from a specific set of indices from '$c\_moments$' and '$moments$', while all the elements of '$pose$' data are utilized as the target data.

Notably, the fourth element in the target data, representing the rotation angle about the x-axis (R_x), is adjusted. The original data varies around 180 degrees, with some values being positive close to 180 and others negative close to -180. Such large variance can impede the learning process of machine learning models. The negative angles are increased by 360 degrees to alleviate this, effectively converting the range from 0 to 360 degrees. This reduces the variance and ensures the data is more uniformly distributed around +180.

Following this, the dataset is partitioned into training, validation, and test sets. This strategic division ensures a robust structure that prevents overfitting by providing distinct subsets for model training, hyperparameter tuning, and performance evaluation. We have incorporated all of our simulation data, along with half of the real-world data, into the training set. This approach offers a comprehensive mix of simulation and actual scenarios for the model to learn from. The remaining half of the real-world data is further split equally to form the validation and test sets. Prior to processing, the entire dataset undergoes a random shuffle to ensure diverse data distribution.

The reason behind this data arrangement is to leverage the extensive volume of synthetic data we possess, allowing our model to generalize and predict real-world scenarios more accurately. We must note that relying solely on synthetic data for training has proven sub-optimal in our experiments. This is largely attributed to the fact that real-world imagery is often subjected to variances in lighting conditions and varying degrees of noise. These elements, absent in synthetic data, can significantly impact the performance of a model trained exclusively on simulated data.

A critical step in the preprocessing phase is data normalization. Normalization scales the values of datasets to a common range, typically between 0 and 1, helping to balance the feature scales and improve model convergence speed. The normalization process in this research is conducted using the mean and variance of the data through a process called z-score normalization. For a given data point x, the normalized value x' is calculated as:

$$x' = \frac{x - \mu}{\sigma}, \tag{5.5}$$

where μ is the mean of the data and σ is the standard deviation. This process is applied to both the input and target data.

The means of the normalized inputs and label data are saved for later use. Preserving these statistical properties is crucial as they allow the denormalization of the model's outputs when used for visual servoing. The denormalization process returns the predicted data to its original scale, facilitating the interpretation of results and allowing direct comparison with real-world measurements.

This prepared data is then fed into the network for training.

(2) **Initialization:** The next step is initializing the model's parameters. This can be done randomly, or one could start from a previously trained model if available. This step is crucial as it determines the starting point of the optimization process.

(3) **Hyperparameters Setting:** At this stage, the model's hyperparameters are set. These include the number of epochs, the learning rate, and other hyperparameters specific to the network architecture. These settings have a significant impact on the learning process and the resulting model performance.

(4) **Model Training:** Here, the model is trained using the prepared dataset. The Model Training step can be divided into these detailed steps:

- **Forward Propagation:** The training data is passed through the network, and each layer applies weights, biases, and activation functions to the input data to produce an output.
- **Loss Computation:** The DNN's output is compared to the actual target data using a loss function. This function quantifies the difference between the predicted and actual values, acting as a metric of the model's performance.
- **Backward Propagation:** The error computed by the loss function is propagated back through the network. This involves computing the gradient of the loss function with respect to the network's parameters.
- **Parameter Update:** In this step, the model's parameters (weights and biases) are updated using Adam optimizer to minimize the loss.

(5) **Model Selection:** After training, the model with the lowest validation loss is selected. This helps to ensure that the final model generalizes well and does not just memorize the training data.

5.5 Training Results

To evaluate the models, a comprehensive and systematic approach was adopted. Every modification—whether it related to hyperparameters or the model architecture—was meticulously documented and assessed. Each model was trained consistently for 200 epochs to establish a uniform baseline for comparison. It is important to emphasize that the initialization could profoundly influence the training results. While multiple initializations were trialed during training, for the sake of clarity and conciseness, only the best results for each model are presented here.

Although the results from the final model are promising, it is conceivable that one might propose an architecture or set of hyperparameters that could yield even more precise pose estimates. The method employed in this study was designed to determine the most effective hyperparameter and architecture configurations within the constraints of available time. For each set of results, the following visual aids are provided:

- A figure depicting the train and validation loss over epochs.
- A table detailing the mean absolute error (MAE) of model predictions on the test set

To avoid memorizing the training data, the 'best' model is chosen based on the validation set's loss, and the MAE table is only for the 'best' model.

To offer a more comparative perspective on the models' performances, a 'Scaled MAE' metric was introduced. Given that the models predict multiple outputs with different units and magnitudes (translational values in millimeters and rotational values in degrees), it is essential to normalize these errors for a fair comparison. The Scaled MAE is computed by dividing the MAE of each output by its range, thereby standardizing the errors across outputs. This ensures that despite having different units or scales, the errors are presented in a unified manner. The final Scaled MAE is the

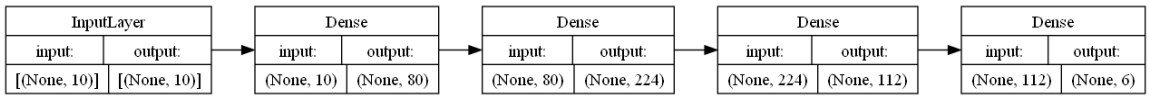
average of these normalized errors from all outputs. The range of each component of the whole dataset is presented in Table 5.4.

Table 5.4: Ranges for output elements of the dataset

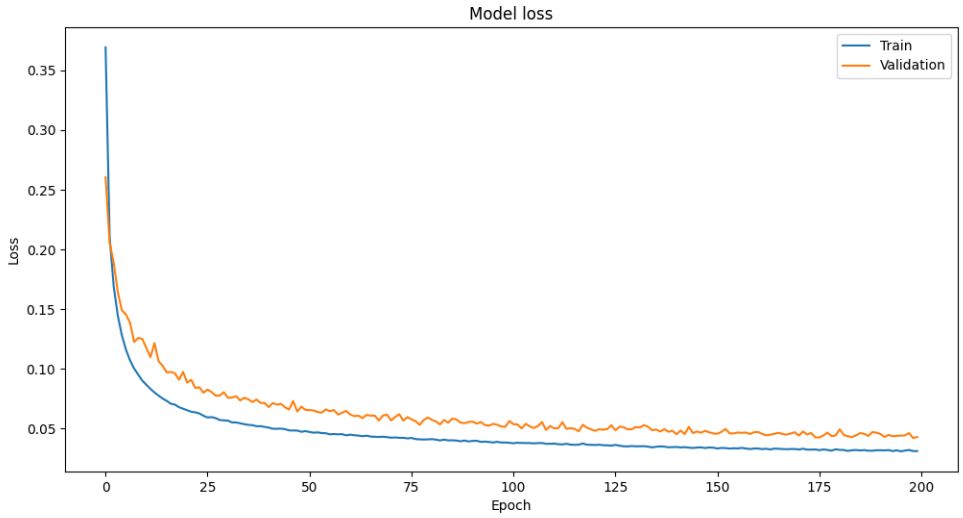
Element	x (mm)	y (mm)	z (mm)	β°	γ°	α°
Minimum	157.5	-150	150	129.91	-55.03	-117.36
Maximum	411.54	150	500	240.5	56.65	129.11
Range	254.04	300	350	110.59	111.68	246.47

5.5.1 Initial Architectures

The random search suggested two primary architectures. We start by comparing these two. The first has three layers with node distributions of 80, 224, and 112. This architecture, including ten inputs and six outputs, is illustrated in Figure 5.7a. Meanwhile, the loss trend over 200 epochs for both training and validation datasets can be viewed in Figure 5.7b.



(a)



(b)

Figure 5.7: First initial model's (a) architecture and (b) loss over epoch

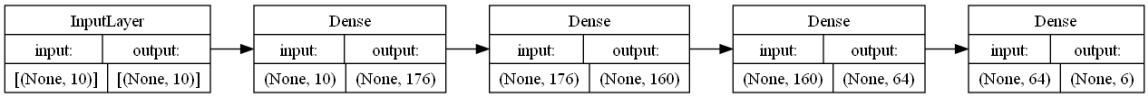
A closer look at Table 5.5 reveals that the translational and rotational dimensions have average

Table 5.5: First initial model's Mean Absolute Error data

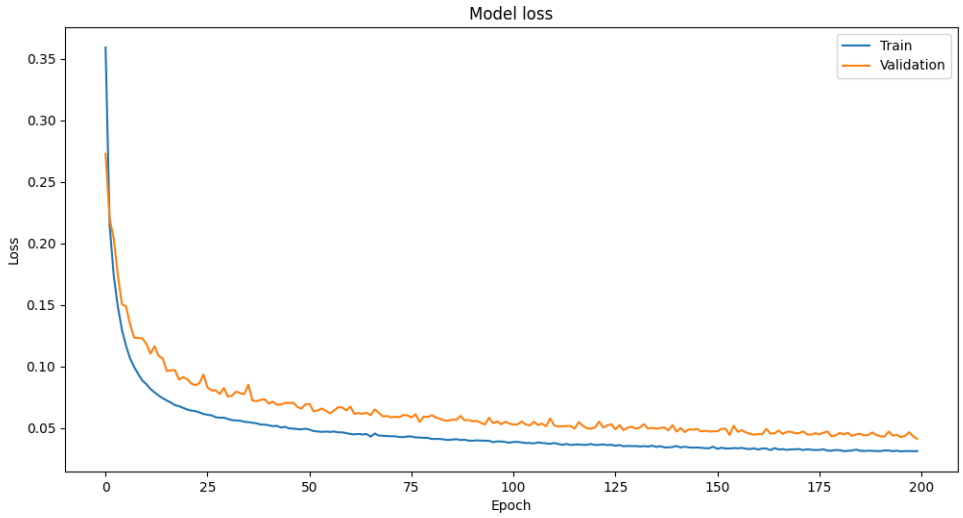
Element	MAE	Average MAE	Scaled MAE	Average Scaled MAE
x	14.3 (mm)	10.22 (mm)	5.63×10^{-2}	2.64×10^{-2}
y	9.41 (mm)		3.14×10^{-2}	
z	6.96 (mm)		1.99×10^{-2}	
β	2.29°	2.11°	2.07×10^{-2}	
γ	2.78°		2.49×10^{-2}	
α	1.26°		0.51×10^{-2}	

MAEs of 10.22 (mm) and 2.11°, respectively. The scaled MAE indicates notable underperformance on the x and y coordinates, whereas the remaining dimensions show relatively satisfactory results.

The second architecture also spans three layers, with 176, 160, and 64 nodes, respectively.



(a)



(b)

Figure 5.8: Second initial model's (a) architecture and (b) loss over epoch

Based on our evaluations of the MAE tables, the architecture with 80,224,112 nodes slightly outperformed its counterpart in terms of translational and overall average scaled MAE. The 176,160,64 node configuration showed a marginally superior capability in approximating rotational degrees of freedom. However, the former was selected as our foundational architecture. The reason behind this

Table 5.6: Second initial model's Mean Absolute Error data

Element	MAE	Average MAE	Scaled MAE	Average Scaled MAE
x	14.6 (mm)	10.57 (mm)	5.75×10^{-2}	2.65×10^{-2}
y	9.72 (mm)		3.24×10^{-2}	
z	7.39 (mm)		2.11×10^{-2}	
β	2.08°	2.05°	1.88×10^{-2}	
γ	2.64°		2.36×10^{-2}	
α	1.42°		0.57×10^{-2}	

choice is its superiority in translational estimates. As evidenced by Tables 5.5 and 5.6, translational estimations reveal a more significant challenge due to their higher scaled MAE values.

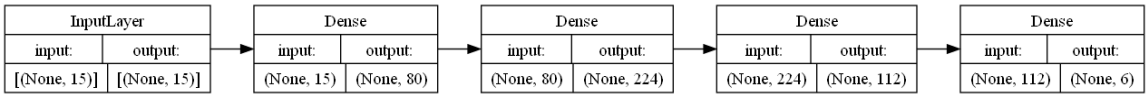
5.5.2 Increase the Number of Inputs (Feature Engineering)

To enhance the accuracy of the model, we added five additional features to the inputs. The first four, denoted as c_1, c_2, c_3, and c_4 (as described in Equation (5.4)), have been recognized as the invariants suggested by Tahri and Chaumette (2005). When the camera and the object are parallel, these features remain invariant under 2D translation, rotation, and scaling. Meanwhile, the fifth one α (Equation (2.39)) was specifically chosen due to its correlation with the R_z component.

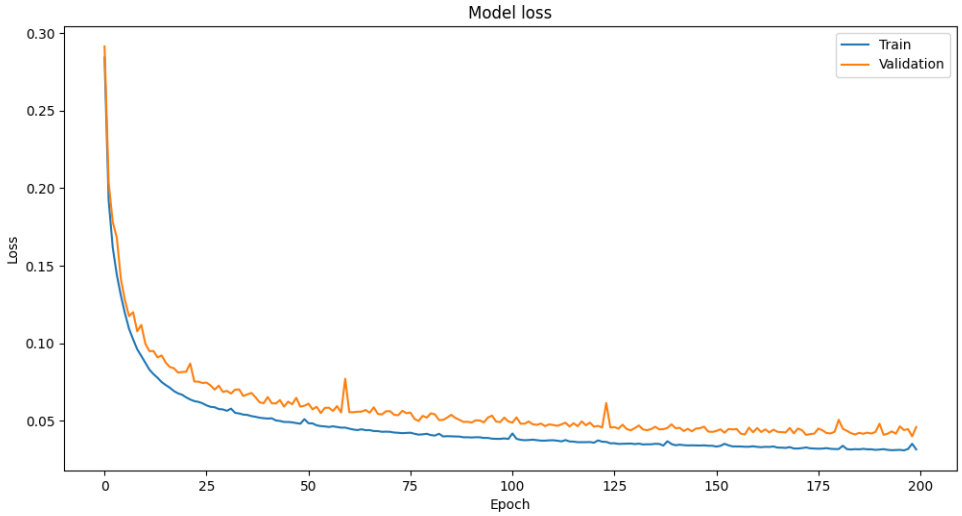
Table 5.7: Model's Mean Absolute Error data after increasing the number of inputs

Element	MAE	Average MAE	Scaled MAE	Average Scaled MAE
x	13.21 (mm)	9.90 (mm)	5.20×10^{-2}	2.54×10^{-2}
y	9.18 (mm)		3.06×10^{-2}	
z	7.32 (mm)		2.09×10^{-2}	
β	2.18°	2.05°	1.97×10^{-2}	
γ	2.64°		2.37×10^{-2}	
α	1.33°		0.54×10^{-2}	

Adding five additional inputs to the neural network has evidently influenced the model's prediction accuracy. A direct comparison between Tables 5.5 and 5.7 reveals that the average MAE for the translational components has been reduced, especially notable in the x direction. This improvement is more obvious when considering the scaled MAE, which decreased from 2.64×10^{-2} to 2.54×10^{-2}. While improvements in rotational estimations are less significant, this shift marked the model's better performance after augmenting new input features.



(a)



(b)

Figure 5.9: Model's (a) architecture and (b) loss over epoch after increasing the number of inputs

5.5.3 Parallel Sub-Networks

To further improve the performance of translational elements, a new architecture was devised. This was inspired by the shallow three-layer architecture of Figure 5.7. Since the performance of the translational degrees of freedom was poor, we made a deeper parallel sub-network for the first three elements, leading to six hidden layers, including 80, 224, 112, 64, 80, and 176 nodes, subsequently. The last three elements utilized four hidden layers with 80, 224, 128, and 80 nodes, respectively. The first two layers are being shared, and the remaining two are operating in parallel. A visual representation of this architecture can be found in Figure 5.5. This design was meticulously designed over multiple experimental iterations.

Table 5.8 shows significant enhancements across all estimated elements, with the translational degrees of freedom standing out notably. The average MAE for these translational elements reduced from 9.90 mm (In Table 5.7) to 6.71 mm. Such a decline indicates the effectiveness of the new deeper sub-network approach. Furthermore, the rotational elements also benefited, with a

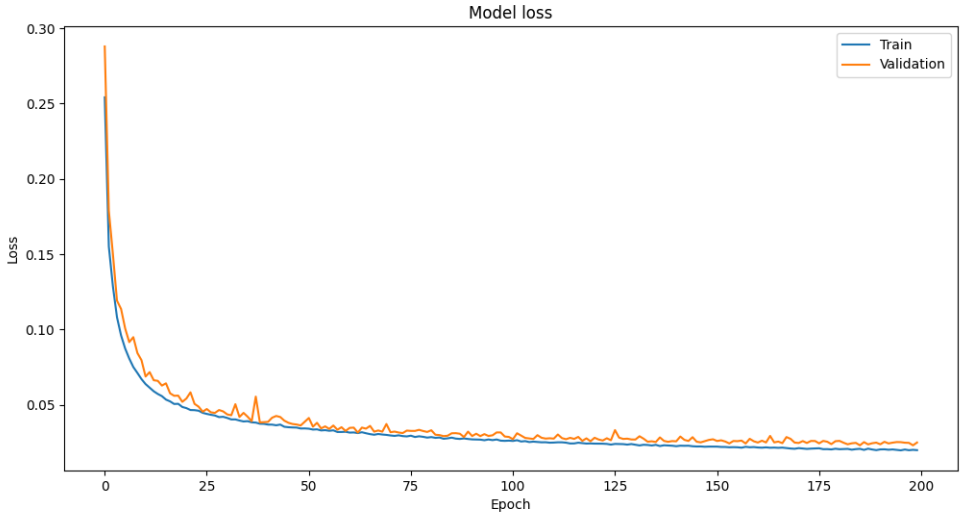


Figure 5.10: Further enhancements in translational DOFs using a deeper sub-network.

Table 5.8: Mean Absolute Error after integrating a deeper sub-network

Element	MAE	Average MAE	Scaled MAE	Average Scaled MAE
x	8.78 (mm)	6.71 (mm)	3.46×10^{-2}	1.80×10^{-2}
y	6.53 (mm)		2.18×10^{-2}	
z	4.81 (mm)		1.37×10^{-2}	
β	1.59°	1.60°	1.44×10^{-2}	
γ	2.14°		1.91×10^{-2}	
α	1.07°		0.43×10^{-2}	

slight reduction in their average MAE from 2.05° to 1.60°. The scaled MAE values confirm these improvements, pointing towards a more accurate and efficient model.

Finally, we decided to stop our iterative modifications and proceed to train the final architecture over a more extended period. The decision to train our model for 1,000 epochs was influenced by our observations on the validation loss. As depicted in Figure 5.11, beyond a certain point, there was not any noticeable improvement in the validation loss. This plateau indicated that further training might not significantly enhance the model's performance and could risk overfitting.

The benefits of extended training are clearly evident when comparing the results from Tables 5.8 and 5.9. The average MAE for the translational elements decreased from 6.71 mm to 5.82 mm. The rotational components also improved from 1.60° to 1.37°. More importantly, the main metric (Average Scaled MAE) remarkably decreased to 1.54×10^{-2}.

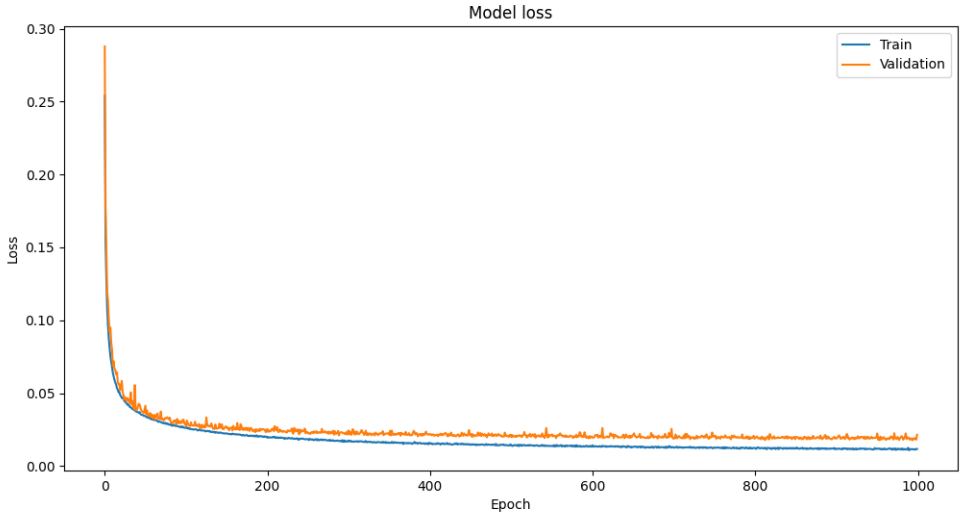


Figure 5.11: Loss over epoch of the final model

Table 5.9: Final model's Mean Absolute Error data

Element	MAE	Average MAE	Scaled MAE	Average Scaled MAE
x	7.45 (mm)	5.82 (mm)	2.93×10^{-2}	1.54×10^{-2}
y	5.83 (mm)		1.94×10^{-2}	
z	4.18 (mm)		1.20×10^{-2}	
β	1.32°	1.37°	1.19×10^{-2}	
γ	1.78°		1.59×10^{-2}	
α	1.02°		0.41×10^{-2}	

5.6 Experimental Results

To confirm the reliability of our final model, as presented in Subsection 5.5, it is needed to apply the final model on the Denso manipulator. It necessitated the derivation of an interaction matrix for the final DNN model. While our initial aim was to derive a diagonal interaction matrix, the practical limitations in achieving zero-error pose estimation necessitated the use of the actual interaction matrix in our experiments. As described by Equation (2.5), the 6×6 interaction matrix represents how the DNN model's predicted image features correlate with the manipulator's six motion axes. For every data point in the test set, the model predicted six image features. The elements in the interaction matrix represent the slopes of the linear regression lines, each comparing a predicted image feature against every actual degree of freedom. This approach helps us understand

the impact of each actual movement on the predicted features.

$$\mathbf{L}_{s_{DNN}} = \begin{bmatrix} \frac{\Delta s_x}{\Delta x} & \frac{\Delta s_x}{\Delta y} & \frac{\Delta s_x}{\Delta z} & \frac{\Delta s_x}{\Delta R_x} & \frac{\Delta s_x}{\Delta R_y} & \frac{\Delta s_x}{\Delta R_z} \\ \frac{\Delta s_y}{\Delta x} & \frac{\Delta s_y}{\Delta y} & \frac{\Delta s_y}{\Delta z} & \frac{\Delta s_y}{\Delta R_x} & \frac{\Delta s_y}{\Delta R_y} & \frac{\Delta s_y}{\Delta R_z} \\ \frac{\Delta s_z}{\Delta x} & \frac{\Delta s_z}{\Delta y} & \frac{\Delta s_z}{\Delta z} & \frac{\Delta s_z}{\Delta R_x} & \frac{\Delta s_z}{\Delta R_y} & \frac{\Delta s_z}{\Delta R_z} \\ \frac{\Delta s_{R_x}}{\Delta x} & \frac{\Delta s_{R_x}}{\Delta y} & \frac{\Delta s_{R_x}}{\Delta z} & \frac{\Delta s_{R_x}}{\Delta R_x} & \frac{\Delta s_{R_x}}{\Delta R_y} & \frac{\Delta s_{R_x}}{\Delta R_z} \\ \frac{\Delta s_{R_y}}{\Delta x} & \frac{\Delta s_{R_y}}{\Delta y} & \frac{\Delta s_{R_y}}{\Delta z} & \frac{\Delta s_{R_y}}{\Delta R_x} & \frac{\Delta s_{R_y}}{\Delta R_y} & \frac{\Delta s_{R_y}}{\Delta R_z} \\ \frac{\Delta s_{R_z}}{\Delta x} & \frac{\Delta s_{R_z}}{\Delta y} & \frac{\Delta s_{R_z}}{\Delta z} & \frac{\Delta s_{R_z}}{\Delta R_x} & \frac{\Delta s_{R_z}}{\Delta R_y} & \frac{\Delta s_{R_z}}{\Delta R_z} \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{0.94} & 0.03 & -0.12 & -0.16 & -3.24 & 0.03 \\ 0.05 & \mathbf{0.98} & 0 & -3.57 & -0.12 & -0.22 \\ -0.21 & -0.01 & \mathbf{0.99} & -0.15 & 0.53 & -0.16 \\ -0.02 & -0.18 & -0.01 & \mathbf{0.97} & 0.05 & 0.04 \\ -0.14 & 0 & 0.01 & 0.01 & \mathbf{0.93} & 0 \\ 0 & -0.04 & -0.02 & 0.16 & -0.02 & \mathbf{0.99} \end{bmatrix} \tag{5.6}$$

As evident from the interaction matrix, the diagonal elements $\mathbf{L}_{s_{DNN}}[i,i]$ (where i ranges from 1 to 6) are very close to 1, while the non-diagonal elements are close to zero, which aligns with our objective. However, notable exceptions are the elements $\mathbf{L}_{s_{DNN}}[1,5] = -3.24$ and $\mathbf{L}_{s_{DNN}}[2,4] = -3.57$. These values indicate a correlation between the x prediction of the DNN during R_y movement and the y prediction during R_x movement. This correlation is understandable, as rotations around the x (R_x) and y (R_y) axes in the manipulator's frame cause corresponding movements along the y and x axes in the image plane. Additionally, the elements $\mathbf{L}_{s_{DNN}}[4,2] = -0.18$ and $\mathbf{L}_{s_{DNN}}[5,1] = -0.14$ in the fourth and fifth rows are higher than other non-diagonal elements, emphasizing the 'x and R_y' and 'y and R_x' interconnections in the final DNN model. Improving the DNN's accuracy in the estimations can further address these interconnections.

We tested the model with five distinct initial poses, ensuring a mix of positive and negative initial errors for each degree of freedom. The chosen initial poses, labelled A through E, are detailed in Table 5.10.

Table 5.10: Initial and desired poses

	Pose					
	x (mm)	y (mm)	z (mm)	β (deg)	γ (deg)	α (deg)
A	314.05	37.05	413.32	166.21	-11.40	-16.25
B	308.93	-57.99	434.48	200.75	-8.26	10.25
C	368.71	-74.09	386.80	200.14	-3.59	12.00
D	257.79	-27.34	495.22	196.62	6.91	22.33
E	276.98	51.01	249.78	163.99	15.68	16.01
Desired	307.5	0	300	180	0	0

The block diagram of the DNN-based visual servoing is depicted in Figure 5.12, where we used a proportional controller and the DNN extracts feature (pose) from the images. For these tests, we adjusted the P controller for each degree of freedom to ensure the manipulator's end effector converges within 1 cm and 3 degrees to the desired pose. The resulting trajectories for each initial pose are depicted in Figure 5.13.

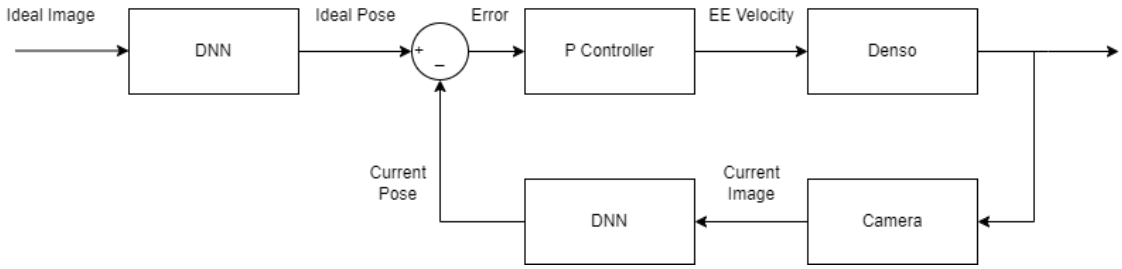


Figure 5.12: DNN-based visual servoing block diagram

As can be seen from the figure, the end effector follows an almost straight path from its start to the target. However, in practice, as the end effector gets close to the desired pose, we noted minor shakiness in its movement. This is caused by small oscillations in the pose estimates, which are the outputs of the neural network.

To validate the proposed features derived from the DNN method, some comparisons were made with a prominent set of features in the literature. This set consists of Tahri and Chaumette (2005)'s features, which are the centroid coordinates x_g and y_g, the area a (Equation (2.37)), and the rotation α (Equation (2.39)). Additionaly, for rotations about the x and y axes, S. Liu et al. (2009)'s features (s_x and s_y as described in Equation (2.50)) are used. From now on, the combination of these features is referred to as the Liu method ($[x_g, y_g, a, s_x, s_y, \alpha]$).

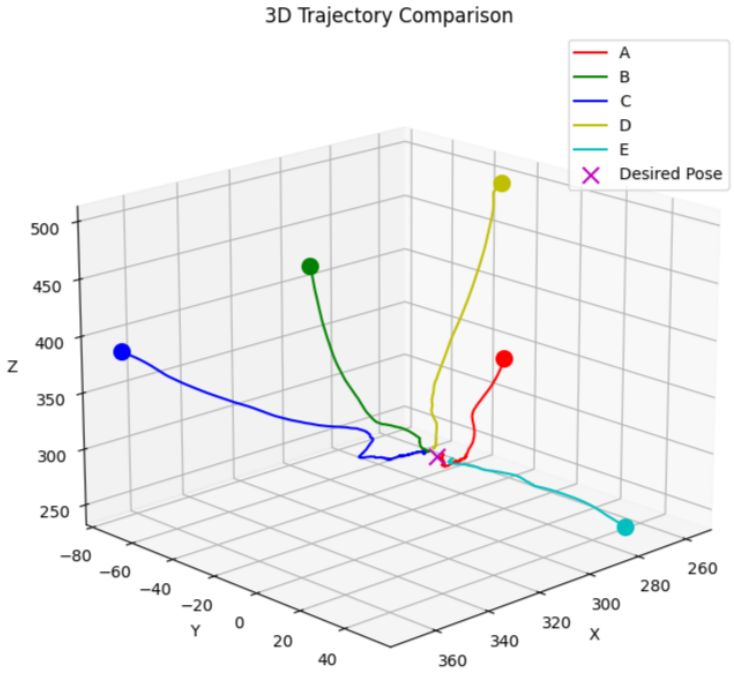


Figure 5.13: Trajectory comparison of five different initial poses

The Liu method's features have the units of $[px, px, px^2, px^{\frac{10}{9}}, px^{\frac{10}{9}}, rad]$. In contrast, the DNN method's features, which represent pose $[x, y, z, \beta, \gamma, \alpha]$, have units of $[mm, mm, mm, deg, deg, deg]$. Because of these unit differences, each method needs its own set of controller gains. To ensure a fair comparison, the P controllers were carefully adjusted for each method, aiming for convergence within 1 cm for translational movements and 3 degrees for rotational ones. The resulting P controllers can be seen in Table 5.11.

Table 5.11: Tuned P controllers for the DNN and Liu methods

Method	Features	Controller Gains
DNN	$[x, y, z, \beta, \gamma, \alpha]$	[0.09, 0.09, 0.12, 0.003, 0.003, 0.003]
Liu	$[x_g, y_g, a, s_x, s_y, \alpha]$	[0.9, 0.9, 150, 30, 60, 0.3]

For validation, both methods were run for 100 seconds, starting from three distinct initial poses (A, B, and C as presented in Table 5.10). Figure 5.14 displays the trajectory plots, contrasting the end effector's path for both the Liu and DNN methods given the aforementioned initial poses. The plots clearly show that the DNN method achieves a direct and efficient trajectory from the starting pose to the target. In contrast, the Liu method often results in curved, less efficient paths. It's

important to highlight that the Liu method operates on feature error, not pose error. As a result, in certain experiments, the end effector stopped close to the desired pose due to minimal feature differences between images. However, the DNN method almost consistently identified these differences, ending up at the correct pose.

Figures 5.15 to 5.20 present the Pose Error and Normalized Velocity change with time. The normalized velocity is obtained by dividing each component of the velocity by the order of magnitude of its starting value, as expressed in:

$$\mathbf{v}_n[i] = \frac{\mathbf{v}[i]}{\text{order}(\mathbf{v}[0])}, \tag{5.7}$$

where $\mathbf{v}_n$ is the normalized velocity, $\mathbf{v}$ is the original velocity, and $\mathbf{v}_n[i]$ and $\mathbf{v}[i]$ are the i^{th} elements of the normalized and original velocities, respectively. The 'order' function is defined as:

$$\text{order}(x) = \begin{cases} 1 & \text{if } x = 0 \\ 10^{\lfloor \log_{10}(|x|) \rfloor} & \text{otherwise} \end{cases}. \tag{5.8}$$

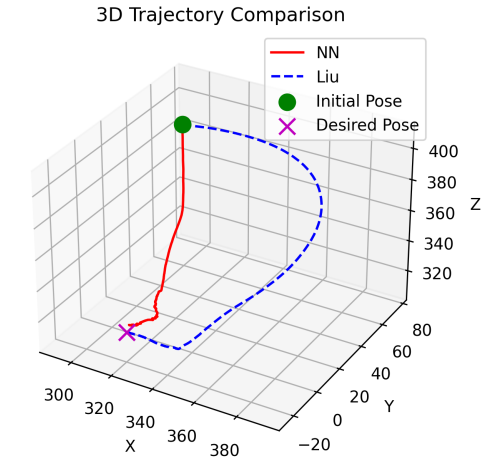
The motivation behind using normalized velocities is primarily visual. Due to the varied scales among velocity elements, presenting them on a single graph in their original form was challenging. Through normalization, we can depict all data points together.

From the figures, a certain fluctuation in the end effector's movement via the DNN method is clear (particularly in the γ element). While the visual plots are insightful, they do not provide the details needed for a comprehensive analysis. Thus, we use three metrics: RMS (Root Mean Square), Max (Maximum value), and STD (Standard Deviation).

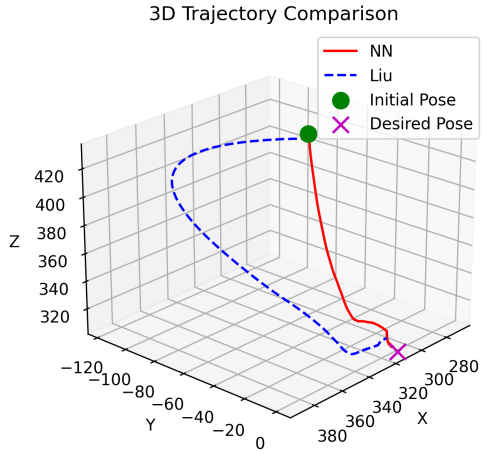
- **RMS:** This metric measures the overall oscillation intensity, whether in terms of pose error or velocity. A high RMS value in the pose error indicates deviations from the desired pose, and when observed in velocity, it points to speed fluctuations.

- **Max:** Serving as a measure for extremes, the Max metric identifies the largest positional deviation or the most significant speed variation.

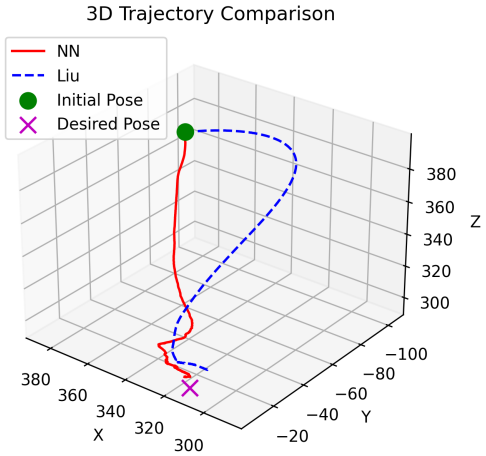
- **STD:** It shows the variability of the pose error or velocity around its mean value. High STD



(a) Initial Pose A

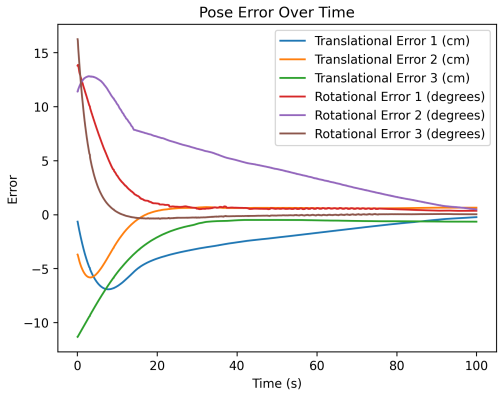


(b) Initial Pose B

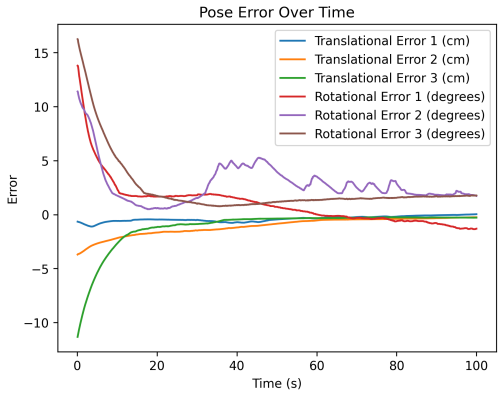


(c) Initial Pose C

Figure 5.14: Trajectory comparison for DNN and Liu Methods for different initial poses.

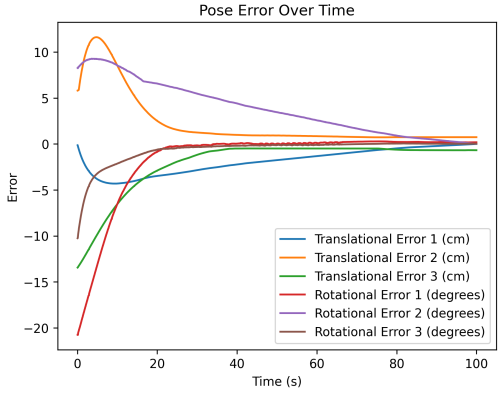


(a) Liu

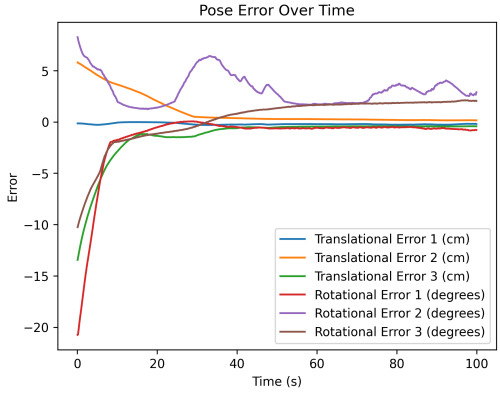


(b) DNN

Figure 5.15: Comparison of Pose Error for Initial Pose A between Liu and DNN methods

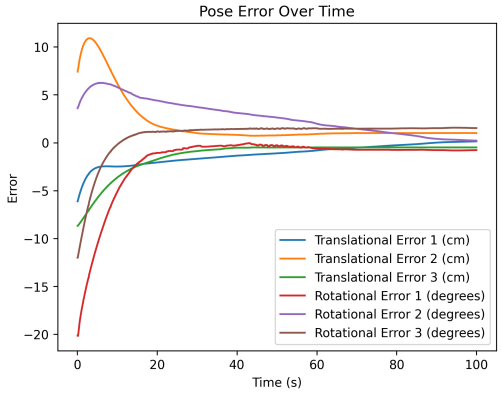


(a) Liu

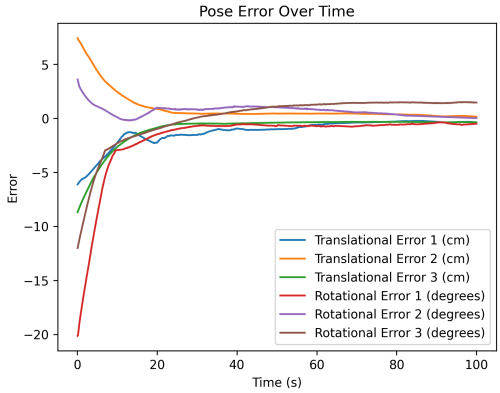


(b) DNN

Figure 5.16: Comparison of Pose Error for Initial Pose B between Liu and DNN methods

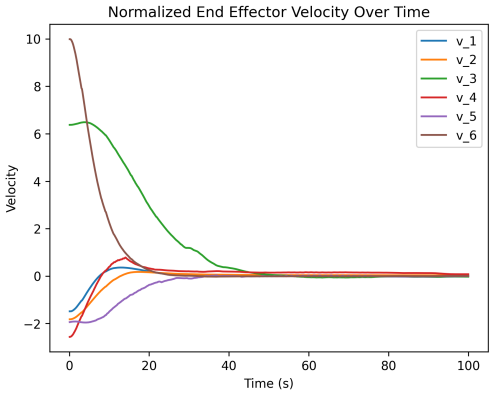


(a) Liu

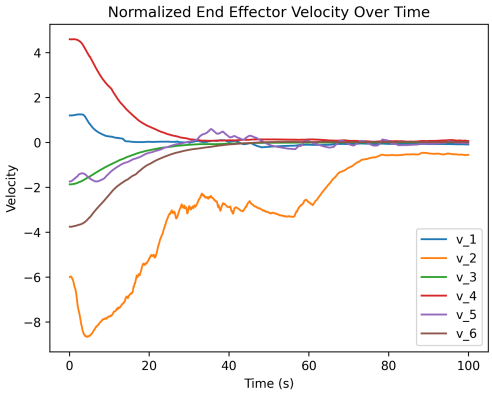


(b) DNN

Figure 5.17: Comparison of Pose Error for Initial Pose C between Liu and DNN methods

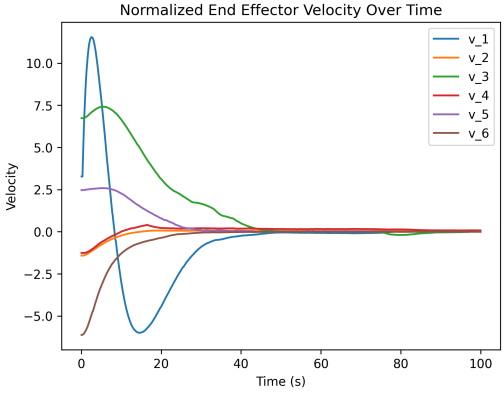


(a) Liu)

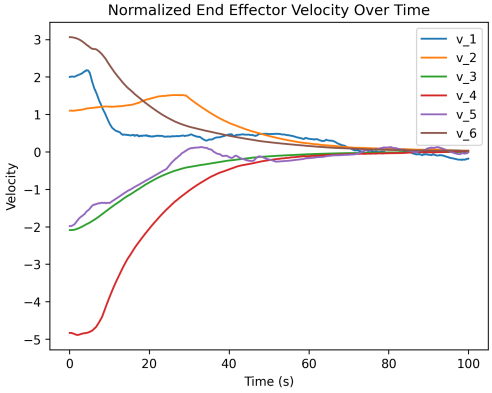


(b) DNN

Figure 5.18: Comparison of Normalized Velocity for Initial Pose A between Liu and DNN methods

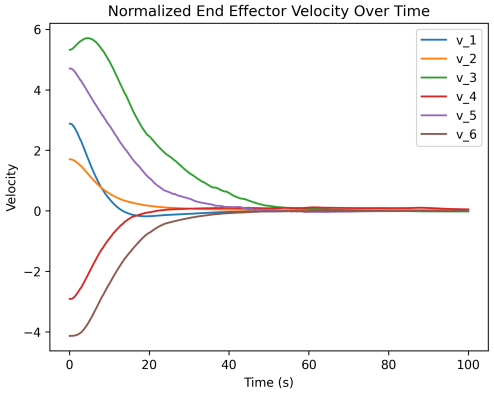


(a) Liu

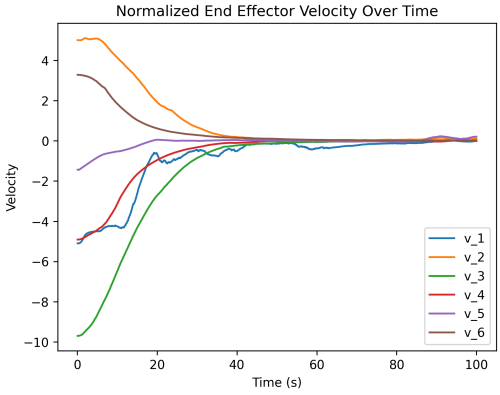


(b) DNN

Figure 5.19: Comparison of Normalized Velocity for Initial Pose B between Liu and DNN methods



(a) Liu



(b) DNN

Figure 5.20: Comparison of Normalized Velocity for Initial Pose C between Liu and DNN methods

values emphasize inconsistencies.

For an organized overview, Tables 5.12 to 5.14 list these metrics over the same three experiments, each with initial poses A, B, and C (as outlined in Table 5.10).

Table 5.12: Metrics comparison for Initial Pose A

Data	Element	Method	RMS	Max	STD
Pose Error	x	Liu	30.883	-69.239	18.374
		DNN	4.826	-11.107	2.505
	y	Liu	16.855	-58.302	16.832
		DNN	13.388	-37.049	7.746
	z	Liu	30.487	-113.317	24.771
		DNN	20.725	-113.316	17.629
	β	Liu	2.932	13.864	2.528
		DNN	2.396	13.790	2.144
	γ	Liu	5.861	12.810	3.409
		DNN	3.384	11.399	1.848
	α	Liu	2.006	16.246	1.968
		DNN	3.523	16.246	2.655
Velocity	x	Liu	2.259	-14.897	2.227
		DNN	0.398	2.379	0.394
	y	Liu	2.862	-18.219	2.862
		DNN	0.439	-2.180	0.346
	z	Liu	2.298	6.671	2.008
		DNN	3.044	-18.756	2.842
	β	Liu	0.004	-0.026	0.004
		DNN	0.008	0.050	0.007
	γ	Liu	0.006	-0.022	0.005
		DNN	0.008	0.036	0.008
	α	Liu	0.013	0.100	0.013
		DNN	0.008	-0.040	0.007

By detailed analysis of Tables 5.12, 5.13 and 5.14, it is evident that the DNN method's performance significantly improved for most of the degrees of freedom. The consistently lower RMS, Max, and STD values indicate a more stable and predictable performance. However, there is a

Table 5.13: Metrics comparison for Initial Pose B

Data	Element	Method	RMS	Max	STD
Pose Error	x	Liu	23.113	-43.142	13.282
		DNN	2.164	-3.048	0.779
	y	Liu	37.497	116.106	29.762
		DNN	18.213	57.992	14.754
	z	Liu	36.142	-134.481	29.702
		DNN	25.334	-134.480	21.287
	β	Liu	4.557	-20.767	4.269
		DNN	3.395	-20.754	3.098
	γ	Liu	4.803	9.276	2.819
		DNN	3.493	8.258	1.501
	α	Liu	1.522	-10.253	1.380
		DNN	2.349	-10.253	2.337
Velocity	x	Liu	4.280	32.010	4.259
		DNN	0.109	0.475	0.109
	y	Liu	1.836	-14.206	1.836
		DNN	0.840	2.657	0.687
	z	Liu	2.633	7.836	2.287
		DNN	3.824	-21.211	3.579
	β	Liu	0.003	-0.014	0.002
		DNN	0.012	-0.057	0.012
	γ	Liu	0.009	0.029	0.008
		DNN	0.007	-0.032	0.007
	α	Liu	0.008	-0.061	0.007
		DNN	0.006	0.031	0.006

Table 5.14: Metrics comparison for Initial Pose C

Data	Element	Method	RMS	Max	STD
Pose Error	x	Liu	16.156	-61.212	10.394
		DNN	17.234	-61.206	12.007
	y	Liu	34.253	109.058	26.731
		DNN	16.534	74.092	13.674
	z	Liu	22.790	-86.787	17.954
		DNN	18.610	-86.796	15.570
	β	Liu	4.125	-20.167	3.625
		DNN	3.673	-20.137	3.215
	γ	Liu	3.284	6.247	1.784
		DNN	1.033	3.592	0.546
	α	Liu	2.266	-12.018	2.146
		DNN	2.332	-12.000	2.332
Velocity	x	Liu	3.666	28.843	3.619
		DNN	1.740	-5.675	1.634
	y	Liu	2.131	17.032	2.021
		DNN	1.517	6.226	1.367
	z	Liu	1.902	6.403	1.679
		DNN	2.166	-9.765	1.986
	β	Liu	0.004	-0.031	0.004
		DNN	0.011	-0.050	0.010
	γ	Liu	0.009	0.047	0.009
		DNN	0.002	-0.018	0.002
	α	Liu	0.009	-0.041	0.008
		DNN	0.007	0.033	0.006

notable exception in the α pose where the DNN method shows a marginally worse performance. Interestingly, when we focus on velocity, the DNN method compensates for the aforementioned pose error. For velocities, it's worth noting that the DNN method's performance metrics for the z and γ directions are higher, indicating a more variable or unpredictable movement.

The DNN-based visual servoing method's adaptability to unanticipated scenarios is demonstrated in this video, showing the manipulator's response when the targeting pin is arbitrarily repositioned in the workspace. The video highlights the system's capability to efficiently track the targeting pin, ensuring it remains centered and parallel in the camera's view within a short amount of time.

5.7 Summary

In this chapter, we introduced a DNN-based method to derive six fully decoupled image features. The proposed image features correspond to the 6D pose of the end effector that results in a nearly identity interaction matrix.

Through several experiments with various architectures, and hyperparameters, we searched for the optimal combination that would lead to accurate and reliable pose estimations. Finally, in section 5.6, the most promising model was implemented on the Denso robot and was tested with various initial poses. Compared to the features presented by Chaumette (2004) and S. Liu et al. (2009), the DNN-based approach demonstrated superior performance in trajectory efficiency, pose accuracy, and velocity.

Chapter 6

Conclusion and Future Works

In this chapter, the main conclusions and contributions of this thesis are summarized. This section is followed by potential extensions and recommendations for future research to further enhance the scope and applicability of the presented study.

6.1 Contributions and Conclusion

This research addresses the challenge of autonomously capturing servicing satellites approaching the ISS using a robotic manipulator, a task previously performed manually by astronauts, which was susceptible to human error. Thus, the motivation of this thesis is to address the need for a more precise and automated approach.

The initial technique we investigated was classical visual servoing. However, this method, while controlling a single degree of freedom of the end effector, unintentionally produced unnecessary movements in other degrees. The interaction matrices of the well-known image features validate this issue. This problem is called coupling, and it necessitates the investigation of innovative solutions.

A fundamental step in our research was the design and 3D printing of a 2D object resembling the targeting pin. An image processing strategy was subsequently developed to detect this pin, yielding a binary image. A comprehensive dataset is required for the proposed methods, so we have designed a procedure for generating datasets from both simulated and real-world environments.

The main contributions and conclusions of this research are as follows:

- **Development of Function-based Visual Servoing:** Introduction of a unique function based on image moments and trainable parameters as the image feature (Equation (4.1)).
- **Function Optimization:** Optimization of the function to achieve the ideal interaction matrix (Table 4.1), focusing on rotations about the camera's x and y axes.
- **Practical Experimentation and Assessment:** Despite theoretical promises, practical experiments with the optimization approach indicated the need for alternative strategies due to unsatisfactory performance.
- **Development of DNN-based Visual Servoing:** Shift to the pose estimation of the camera, treating the 6D pose as the set of image features for end effector control, offering a nearly diagonal interaction matrix.
- **Neural Network Design and Hyperparameter Tuning:** Designing a deep neural network for pose estimation using image moments and enhancing its performance through hyperparameter tuning.
- **Comparative Analysis with Established Techniques:** Comprehensive experimental validation of the neural network approach, demonstrating significant improvements in trajectory, pose accuracy, and velocity of the end effector compared to established visual servoing techniques.

The DNN-based visual servoing method's most important impact is its adaptability for controlling various robotic manipulators in marker-based applications. By using our training procedure for a specific targeting pin, one can potentially achieve performances surpassing classical visual servoing methods.

6.2 Future Works

The following suggestions can potentially improve the proposed methods' performance and generalizability:

6.2.1 General Suggestions

- **Dataset Enhancement:** Creating a dataset that uses the real targeting pin (Figure 1.1) or ensuring that the dataset's environment closely resembles space lighting conditions can improve the accuracy of pose predictions.
- **Data Augmentation:** Adding images captured from various cameras with different fields of view or focal lengths can expand the dataset's diversity. Including common objects of different shapes, such as rectangles, can also be beneficial.
- **Canadarm2 Kinematics:** Investigate the application of the proposed methods by testing or simulating on the Canadarm2 kinematics.

6.2.2 Optimization Improvements

- **Image Feature Function Enhancement:** Modify the image feature function (Equation (4.1)) to be more general by introducing a new definition from scratch, including additional terms or converting all powers to trainable parameters.
- **Literature-based Image Features:** Integrate relevant image features from existing literature, such as c_1 to c_4 (Equation (5.4)) or α (Equation (2.39)), into the optimization function.
- **Cost Function Adjustments:** Explore adjustments or modifications to the existing cost function.
- **Degree of Freedom Extension:** Expand the method's application to serve all six degrees of freedom, aiming to determine optimized image features for each

6.2.3 Deep Neural Network Enhancements

- **Hyperparameter Refinement:** Continuous tuning and experimentation with the network's architecture and hyperparameters can improve performance.
- **Transfer Learning:** Using insights from established pre-trained pose estimation models and adapting them to the current problem might yield better results.

- **Including 3D Data:** Pose estimation can be enhanced by feeding 3D data, such as point clouds or depth maps, into the neural network.
- **Custom Loss Function:** Designing a new loss function specially made for image moment-based pose estimation can result in more precise estimations.
- **Network Ensembling:** Aggregating outputs from diverse network architectures can enhance accuracy, as different models might specialize in recognizing distinct features.
- **Direct Image Input:** Utilizing the image itself (rather than its moments) as the network's input could provide insights potentially overlooked when solely relying on image moments.

Appendix A

Additional Notes

A.1 Number of Trainable Parameters in Equation (4.1)

The function defined in Equation (4.1) has a total of 252 trainable parameters, with an equal division of 126 parameters each in the numerator and denominator.

Table A.1: Breakdown of Trainable Parameters in Equation (4.1)

Type	Example	# of Parameters / Term	# of Unique Terms	Total # of Parameters
1	$c\mu_{11}^{1}\mu_{30}^{2}$	1	$\binom{10}{2} - \binom{5}{1}$	40
2	$c\mu_{02}^{c}\mu_{11}^{1}$	2	$\binom{3}{1} \times \binom{10}{1}$	60
3	$c\mu_{02}^{c}\mu_{20}^{c}$	3	$\binom{3}{2}$	9
4	$c\mu_{00}^{c}$	2	$\binom{3}{1}$	6
5	$c\mu_{21}^{1}$	1	$\binom{10}{1}$	10
6	c	1	$\binom{13}{0}$	1
				Total: 126

In the development of the function stated in Equation (4.1), there are six distinct types of trainable parameters, as described below. These types are defined based on the powers of the central image moments presented in the Table 4.2:

- **Type 1:** Includes terms with two distinct image moments, each having powers of 1 or 2, where the image moment order is a multiple of 2 (excluding '11').
- **Type 2:** Involves terms combining one image moment with a variable power (either μ_{00}, μ_{02},

or μ_{20}) and another moment with a power of 1 or 2.

- **Type 3:** Consists of terms with two distinct image moments, both having variable powers.
- **Type 4:** Contains terms with a single image moment that have variable powers.
- **Type 5:** Includes terms with a single image moment, having powers of 1 or 2.
- **Type 6:** Represents a term which consists of only one trainable parameter and does not include any image moment.

References

Aghili, F. (2012). Active orbital debris removal using space robotics. In *Proc. of International Symposium on Artificial Intelligence, Robotics and Automation in Space i-SAIRAS. Italy: Turin.*

Bäck, T., Fogel, D. B., & Michalewicz, Z. (1997). Handbook of Evolutionary Computation. *Release*, *97*(1), B1.

Bateux, Q., Marchand, E., Leitner, J., Chaumette, F., & Corke, P. (2018, May). Training deep neural networks for visual servoing. In *2018 IEEE International Conference on Robotics and Automation (ICRA)* (p. 3307-3314). doi: 10.1109/ICRA.2018.8461068

Boyd, S. P., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

Chaumette, F. (2004, Aug). Image moments: a general and useful set of features for visual servoing. *IEEE Transactions on Robotics*, *20*(4), 713-723. doi: 10.1109/TRO.2004.829463

Chaumette, F., & Hutchinson, S. (2006, Dec). Visual servo control. i. basic approaches. *IEEE Robotics & Automation Magazine*, *13*(4), 82-90. doi: 10.1109/MRA.2006.250573

Corke, P. (2017). *Robotics, vision and control*. Springer International Publishing. doi: 10.1007/978-3-319-54413-7

Corke, P., et al. (1996). *Visual Control of Robots: high-performance visual servoing*. Research Studies Press Taunton, UK.

CSA. (2020). *International Space Station news.* `https://www.asc-csa.gc.ca/eng/iss/news-2020.asp`.

Denso. (2009). *Denso Robot - Vertical articulated, VP-G Series - General information about robot.*

Dong, J., Hu, Y., & Peng, K. (2012, May). Robot visual servo control based on fuzzy adaptive pid.

In *2012 International Conference on Systems and Informatics (ICSAI2012)* (p. 1337-1341). doi: 10.1109/ICSAI.2012.6223282

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD* (Vol. 96, pp. 226–231).

Fang, Y., Liu, X., & Zhang, X. (2011). Adaptive active visual servoing of nonholonomic mobile robots. *IEEE Transactions on Industrial Electronics*, *59*(1), 486–497.

Feddema, J., & Mitchell, O. (1989, Oct). Vision-guided servoing with feature-based trajectory generation (for robots). *IEEE Transactions on Robotics and Automation*, *5*(5), 691-700. doi: 10.1109/70.88086

Flores-Abad, A., Zhang, L., Wei, Z., & Ma, O. (2017). Optimal capture of a tumbling object in orbit using a space manipulator. *Journal of Intelligent & Robotic Systems*, *86*, 199–211.

Guenin, B., Könemann, J., & Tunçel, L. (2014). *A Gentle Introduction to Optimization*. Cambridge University Press. doi: 10.1017/CBO9781107282094

Hu, M.-K. (1962). Visual pattern recognition by moment invariants. *IRE transactions on information theory*, *8*(2), 179–187.

Intel. (2019). *Intel realsense d400 series product family datasheet.*

JAXA. (2010). *Electro mechanical grapple fixture used on kibo's small fine arm.* `http://iss.jaxa.jp/library/photo/iss022e020034.php`.

Keipour, A., Pereira, G. A., Bonatti, R., Garg, R., Rastogi, P., Dubey, G., & Scherer, S. (2022). Visual servoing approach to autonomous uav landing on a moving vehicle. *Sensors*, *22*(17), 6549.

Keshmiri, M., Xie, W.-F., & Mohebbi, A. (2014). Augmented image-based visual servoing of a manipulator using acceleration command. *IEEE Transactions on Industrial Electronics*, *61*(10), 5444–5452.

Kim, J.-K., Kim, D.-W., Choi, S.-J., & Won, S.-C. (2006). Image-based visual servoing using sliding mode control. In *2006 SICE-ICASE International Joint Conference* (pp. 4996–5001).

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kumra, S., & Kanan, C. (2017, Sep.). Robotic grasp detection using deep convolutional neural

networks. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (p. 769-776). doi: 10.1109/IROS.2017.8202237

Kurnaz, O. (2019, 01). Real-time implementation of image based plc control for a robotic platform. *Balkan Journal of Electrical and Computer Engineering*, *7*. doi: 10.17694/bajece.487212

Leppich, R. (2021). *Pre-training of Deep Transformer Encoders for Time Series Representation Models* (Doctoral dissertation). doi: 10.13140/RG.2.2.18298.82882

Li, S., Xie, W., & Gao, Y. (2017). Enhanced IBVS controller for a 6DOF manipulator using hybrid PD-SMC method. In *IECON 2017-43rd annual conference of the IEEE industrial electronics society* (pp. 2852–2857).

Liu, J., & Li, Y. (2019). An image based visual servo approach with deep learning for robotic manipulation. *arXiv preprint arXiv:1909.07727*.

Liu, S., Xie, W.-F., & Su, C.-Y. (2009). Image-based visual servoing using improved image moments. In *2009 International Conference on Information and Automation* (pp. 577–582).

Luo, Z. H., & Sakawa, Y. (1990). Control of a space manipulator for capturing a tumbling object. In *29th IEEE Conference on Decision and Control* (pp. 103–108).

Malis, E., Chaumette, F., & Boudet, S. (1999, April). 2 1/2 d visual servoing. *IEEE Transactions on Robotics and Automation*, *15*(2), 238-250. doi: 10.1109/70.760345

Matheron, G., & Serra, J. (2000). *The birth of mathematical morphology.* `http://cmm.ensmp.fr/~serra/pdf/birth_of_mm.pdf`.

MDA. (2023). *Dextre.* `https://mda.space/en/dextre/`.

MDRobotics. (n.d.). *Mobile servicing system data sheet.*

Mehle, A., Likar, B., & Tomaževič, D. (2017). In-line recognition of agglomerated pharmaceutical pellets with density-based clustering and convolutional neural network. *IPSJ Transactions on Computer Vision and Applications*, *9*(1), 7.

Mehtatt, S., Dixon, W., Mac Arthur, D., & Crane, C. (2006, June). Visual servo control of an unmanned ground vehicle via a moving airborne monocular camera. In *2006 American Control Conference* (p. 6 pp.-). doi: 10.1109/ACC.2006.1657561

Mohebbi, A. (2013). *Real-time stereo visual servoing of a 6-dof robot for tracking and grasping moving objects* (Unpublished master's thesis). Concordia University.

Nokleby, S. B. (2007). Singularity analysis of the canadarm2. *Mechanism and Machine Theory*, *42*(4), 442–454.

Papadopoulos, E., Aghili, F., Ma, O., & Lampariello, R. (2021). Robotic manipulation and capture in space: A survey. *Frontiers in Robotics and AI*, 228.

Quanser. (2011). *Quanser 6-axis articulated robot - open architecture enabled with quarc.*

Rekleitis, I., Martin, E., Rouleau, G., L'Archevêque, R., Parsa, K., & Dupuis, E. (2007). Autonomous capture of a tumbling satellite. *Journal of Field Robotics*, *24*(4), 275–296.

Sanderson, A., & Weiss, L. (1980). *Image-based visual servo control using relational graph error signals.*

Shi, Y., Liang, B., Wang, X., Xu, W., & Liu, H. (2012, June). Study on intelligent visual servoing of space robot for cooperative target capturing. In *2012 IEEE International Conference on Information and Automation* (p. 733-738). doi: 10.1109/ICInfA.2012.6246915

Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2010). *Robotics: Modelling, planning and control*. Springer London. Retrieved from `https://books.google.ca/books?id=jPCAFmE-logC`

Sra, S., Nowozin, S., & Wright, S. J. (2012). *Optimization for machine learning*. MIT Press.

Szeliski, R. (2022). *Computer vision: algorithms and applications*. Springer Nature.

Tahri, O., & Chaumette, F. (2005, Dec). Point-based and region-based image moments for visual servoing of planar objects. *IEEE Transactions on Robotics*, *21*(6), 1116-1127. doi: 10.1109/TRO.2005.853500

Wei, G.-Q., Arbter, K., & Hirzinger, G. (1997). Real-time visual servoing for laparoscopic surgery. controlling robot motion with color image segmentation. *IEEE Engineering in Medicine and Biology Magazine*, *16*(1), 40–45.

Wikipedia. (2023). *Grapple fixture.* `https://en.wikipedia.org/wiki/Grapple_fixture`.

Zhao, Y., Xie, W., & Wang, T. (2012). Neural network-based image moments for visual servoing of planar objects. In *2012 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)* (pp. 268–273).