CENTRO DE INVESTIGACIONES EN OPTICA, A.C.

POSGRADO INTERINSTITUCIONAL EN CIENCIA Y TECNOLOGÍA

CENTRO DE INVESTIGACIONES EN ÓPTICA A.C.

VISUAL CONTROL BASED TRAJECTORY DESIGN FOR A MOBILE ROBOT IN AN AGRICULTURAL ENVIRONMENT

Visto Bueno, 28 de Abril 2025

Dr. Gerardo Flores
Director de Tesis

Tesis

QUE PARA OBTENER EL GRADO ACADÉMICO DE

MAESTRA EN CIENCIA Y TECNOLOGÍA

ESPECIALIDAD EN MECATRÓNICA

PRESENTA:

Ing. Erandi Ireri Vega Guzmán

DIRECTOR: **Dr. Gerardo Ramón Flores Colunga**

Versión Final

LEÓN DE LOS ALDAMA, GUANAJUATO.
Abril, 11, 2025

POSGRADO INTERINSTITUCIONAL CIENCIA Y TECNOLOGIA

Dr. David Monzón Hernández
Director de Posgrado
PICYT – CIO
León

Los abajo firmantes, miembros del Comité Tutorial de la alumna **ERANDI IRERI VEGA GUZMÁN**, una vez leída y revisada la Tesis titulada "**VISUAL CONTROL BASED TRAJECTORY DESIGN FOR A MOBILE ROBOT IN AN AGRICULTURAL ENVIRONMENT**", aceptamos que la referida tesis revisada y corregida sea presentada por la alumna para aspirar al grado de **Maestría en Ciencia y Tecnología** en **Mecatrónica** durante el Examen de Grado correspondiente.

Y para que así conste firmo la presente a los once días del mes de abril del año dos mil veinticinco.

Dr. Gerardo Ramón Flores Colunga
Director de Tesis

Dr. Noé Guadalupe Aldana Murillo
Asesor de Tesis

Dr. Luis Manuel Valentín Coronado
Asesor de Tesis

Dr. David Monzón Hernández
Director de Posgrado
PICYT – CIO
León

Los abajo firmantes, miembros del Jurado del Examen de Grado de la alumna **ERANDI IRERI VEGA GUZMÁN**, una vez leída y revisada la Tesis titulada "**VISUAL CONTROL BASED TRAJECTORY DESIGN FOR A MOBILE ROBOT IN AN AGRICULTURAL ENVIRONMENT**", aceptamos que la referida tesis revisada y corregida sea presentada por la alumna para aspirar al grado de **Maestría en Ciencia y Tecnología** en **Mecatrónica** durante el Examen de Grado correspondiente.

Y para que así conste firmo la presente a los once días del mes de abril del año dos mil veinticinco.

Dr. Andrés Montes de Oca Rebolledo
Presidente

Dr. Francisco Garibaldi Márquez
Secretario

Dr. Gerardo Ramón Flores Colunga
Vocal

Dr. Alejandro Flores Reyes
Suplente

Resumen

Este trabajo integra técnicas de visión por computadora y control no lineal para el desarrollo de sistemas robóticos autónomos, con aplicaciones en la creación de trayectorias en diferentes ambientes, como en un cultivo de fresas. Se explora el Image-Based Visual Servoing (IBVS) para guiar el movimiento del robot utilizando información visual de imágenes, ajustando su trayectoria de manera dinámica. También se aborda la estimación de la homografía entre imágenes para reconstruir movimientos relativos, destacando la importancia de una correcta selección de puntos clave. Finalmente, se analiza el uso de Nonlinear Model Predictive Control (NMPC) para optimizar el comportamiento del robot en tiempo real, ajustando las velocidades lineales y angulares según el modelo y las predicciones de su movimiento futuro.

Palabras clave: Homografía, Tiempo Real, Control Robótico Autónomo, Visión por Computadora, NMPC.

Abstract

This work integrates computer vision and nonlinear control techniques for the development of autonomous robotic systems, with applications in trajectory creation in different environments, such as in a strawberry field. Image-Based Visual Servoing (IBVS) is explored to guide the robot's movement using visual information from images, dynamically adjusting its trajectory. The estimation of homography between images is also addressed to reconstruct relative motion, highlighting the importance of proper keypoint selection. Finally, the use of Nonlinear Model Predictive Control (NMPC) is analyzed to optimize the robot's behavior in real time, adjusting linear and angular velocities based on the model and predictions of its future movement.

Keywords: Homography, Real-Time Optimization, Autonomous Robotic Control, Computer Vision, NMPC.

To everyone who made this work possible, especially to my family.

Acknowledgments

I would like to express my deepest gratitude to my family for their support and encouragement throughout this journey. Their patience and belief in me have been invaluable. I sincerely thank my colleagues at the Perception and Robotics Laboratory. Their help has greatly contributed to my learning and growth. I also acknowledge my advisor, Dr. Gerardo, for his guidance, patience, and support. Likewise, I would like to thank Dr. Noe for his valuable insights and advice, which have enriched my research.

Contents

List of Figures

List of Tables

Glossary

Blender	Open-source software for 3D modeling, animation, rendering, and simulation, widely used in robotics and video games.
Gazebo	3D robot simulator for physics and sensor testing, allowing algorithm validation without real hardware.
IBVS	Vision-based control using image features to minimize the difference between observed and desired images.
NMPC	Model predictive control using a nonlinear system model, applied in robotics and autonomous vehicles.
Nodes in ROS	Processes in ROS that perform specific tasks, communicating via topics, services, or actions.
ORB Points	Feature detection method in computer vision, combining FAST keypoint detection and BRIEF descriptors.
PBVS	Position-based visual servoing, using object or camera position data for robotic control.
ROS	Software framework providing tools for robotics applications, enabling hardware control and communication.
Topics in ROS	Communication channels in ROS for message exchange between nodes.

Chapter 1

Introduction

This chapter addresses the application of visual control in robotics, emphasizing robot autonomy and precision in dynamic and unstructured environments. Within this context, applications in the field of agricultural robots are also explored, where challenges include interaction with natural and changing environments, as well as the need to operate efficiently and accurately. The field of robotics is advancing rapidly, driven by the integration of engineering, artificial intelligence, and automation technologies. Finally, the fundamental approaches of this thesis are presented, including the main objectives and the hypothesis guiding the research.

1.1 Background

The development of this work finds its origin in the interest to deepen the knowledge of the field of robotics, to specialize in this area of growing technological relevance. Robotics represents a multidisciplinary discipline that combines concepts from engineering, artificial intelligence, and automation, and whose impact is reflected in various sectors, including industry, healthcare, agriculture, and transportation. Within this field, one aspect that aroused the most interest was

vision due to its fundamental role in the interaction between robotic systems and their environment.

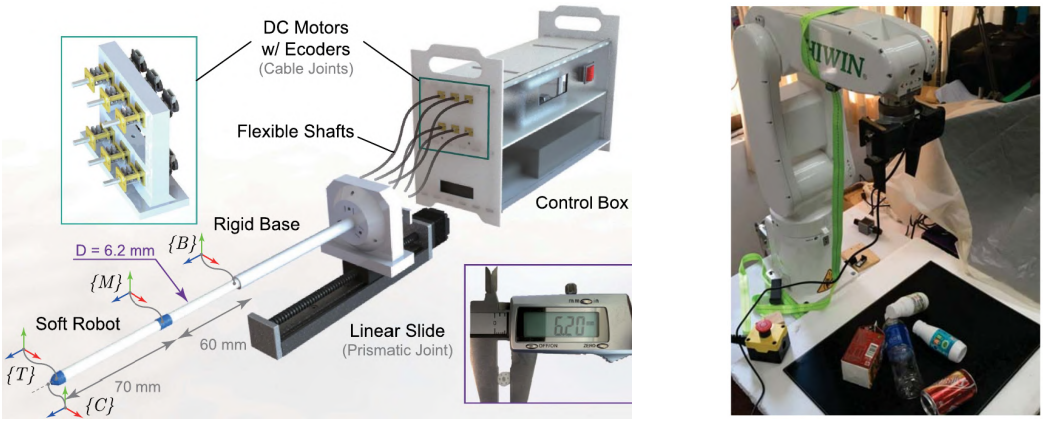
Vision is an essential tool in everyday human life, allowing us to perceive and analyze our surroundings, identify obstacles, and make decisions based on what we observe. For example, in seemingly simple tasks like writing, we use our vision to locate a pencil, pick it up, and direct it towards a sheet of paper to act. This process, though intuitive for humans, involves a complex system of perception and coordination that operates precisely and in real-time. This reasoning raised a central question in the development of this work: How can this visual capability be translated to the realm of robotics?

As part of machine vision systems, visual control allows robots to analyze visual data to make real-time adjustments during task execution. Implementing this approach contributes to greater precision and flexibility in interactions with the environment, representing a significant advance in areas such as industrial automation, autonomous systems, and service robotics.

Currently, numerous studies focus on visual control for different types of environments. Cameras can be used to create detailed maps, allowing the robot to localize itself based on these maps, as is the case of Simultaneous Localization and Mapping (SLAM) technology. In the paper Deng et al. (2020), it is demonstrated how SLAM is used specifically in rescue robots, enabling them to navigate and operate effectively in complex and unfamiliar environments, thus improving their ability to perform tasks autonomously in emergencies. Another work that focuses on point cloud mapping and applies control in conjunction with vision can be found in Ramírez et al. (2024). In this case, control is used to mitigate problems such as information loss and data transmission delays.

Visual control has also found prominent applications in industrial robotics. The work of Sun and Lin (2020) presents an eye-in-hand robotic arm for object recognition and grasping tasks. In this approach, RGB-D cameras are used to obtain 2D images and depth data, which are then processed by convolutional neural networks (CNNs) to perform instance segmentation. Visual control has also shown great potential in medical robotics. In the paper Lai et al. (2023), a novel approach is presented to guide a wired soft robot in a static environment using the Simulation Open Framework Architecture (SOFA). Figure 1.1 shows

the two robots, whose designs differ from each other because they are aimed at achieving different goals. However, both share the need to use a camera for proper operation, using different vision techniques.



(a) (b)

Figure 1.1: (a) CAD schematic of the soft robotic system. The soft robot has a total of six cable joints. Notation of frames: B: base; M : middle; T : tip; and C: camera. The camera is located in the reference frame C, which corresponds to a small camera mounted on the robot. (Lai et al., 2023). (b) The experiment environment. A Hiwin RA 605 robotic arm and several objects placed on the workbench are shown for detection (Sun & Lin, 2020).

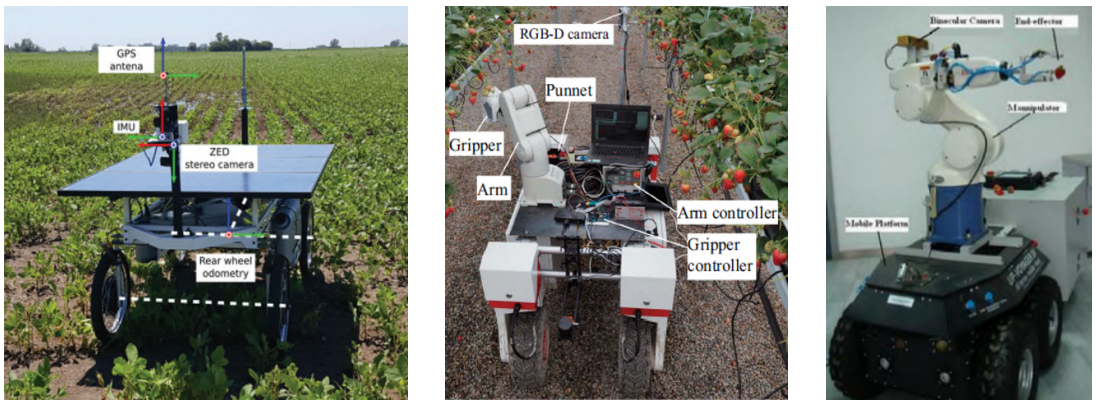
1.1.1 Agricultural Robotics

In order to define the possible applications of the robot developed in this work more precisely, it was decided to align it with one of the Strategic National Programs (PRONACES) focused on food sovereignty (SECIHTI, 2025). In this context, an agricultural robot was proposed, oriented toward autonomous navigation within a strawberry crop, to support tasks such as monitoring, data collection, and potentially, precision agriculture operations.

Agricultural robotics has emerged as a key tool to address the challenges of food production, such as increasing demand, labor shortages, and the need for more sustainable practices. Such robots are designed to perform specific tasks, such as planting, crop monitoring, harvesting fruits and vegetables, or even the precise application of fertilizers and pesticides. For example, Pire et al. (2019) describes a robot designed to weed soybeans. On the other hand, Xiong et al.

(2020) and Feng et al. (2012) discuss the development of robots focused on strawberry harvesting. The former operates in an unstructured environment within tabletop strawberry crops, while the latter is implemented in a more structured environment, using a white background to facilitate the detection of strawberries.

The purpose and design of this work will be defined according to the operating environment, which can be an open field (Figure 1.2 (a)), a raised table cultivation system (Figure 1.2 (b)), or a controlled laboratory (Figure 1.2 (c)). Furthermore, not only the environment must be considered but also the type of crop to be monitored and harvested. In the present work, one of the objectives is the monitoring of strawberry crops, a fruit characterized by its small size, delicate biometric property, and color changes associated with its degree of ripeness. Therefore, it is essential to accurately estimate the maturity of the fruit, as detailed in the paper Zheng et al. (2021).



(a) (b) (c)

Figure 1.2: Examples of agricultural robots: a) robot designed for weeding soybean crops (Pire et al., 2019); b) robot operating in a strawberry growing tunnel (Xiong et al., 2020); c) modular robot specializing in strawberry harvesting (Feng et al., 2012)

1.1.2 Laboratory background

Previously, the Perception and Robotics Laboratory (LAPYR) worked on an autonomous mobile robot to detect strawberries and estimate their maturity in a planting environment. In Mejía et al. (2023) and Jiménez et al. (2023), an ideal strawberry inspection system was proposed for planting in ridge-row plantations. A rover (mobile robot) equipped with a vision system, developed in the laboratory,

is used. The rover navigates through rows between the strawberry ridges while detecting ripe fruits and constructing a map with their locations (see Figure 1.3).

The strawberry detection system consisted of two tasks: a prediction through image processing and a deep learning detection approach. In the image processing, the estimation of strawberry ripening was proposed, differentiating between green color (immature strawberries) and red color (ripe strawberries), subsequently, the MobileNetV2 neural network was employed to attain improved performance results.

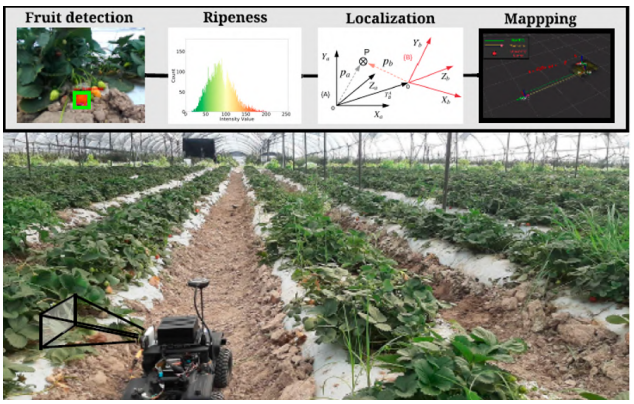


Figure 1.3: The rover built in LAPYR navigates through a strawberry crop. It detects the strawberries, determines their ripeness, locates them along the ridges, and obtains a map of the distribution of strawberries in space (Mejía et al., 2023).

Based on the above, the aim is to develop an autonomous vehicle capable of navigating using visual memory, integrating this technique with the advances previously achieved in the prior work.

1.2 Problem definition

In the development of autonomous vehicles, accurate navigation in unstructured environments is a crucial challenge, as seen in Bai et al. (2023). This process relies on the system's ability to interpret its environment through sensors or cameras, allowing real-time decisions to be made. In this context, the use of computer vision techniques, such as homography-based navigation (Delfin et al., 2016), offers an approach to estimating vehicle motion and relative position, using only image-to-image comparison.

Despite recent advances, current methods for autonomous navigation face significant limitations in terms of accuracy and robustness. In particular, in scenarios where traditional sensors such as LiDAR or GPS are not feasible or reliable (e.g. indoors or in environments with interference), alternatives based solely on visual information are required (Zhong et al., 2022). However, motion estimation using homographs presents challenges related to variations in lighting conditions, changes in terrain texture, and errors in the correspondence of key points between images.

1.3 Justification

Mexico, as a megadiverse country, has a highly relevant agricultural sector, not only in economic terms but also in social and environmental terms. Agriculture supplies the local population with essential foodstuffs for food security and plays a crucial role as a source of raw materials for multiple industries worldwide. This sector is fundamental for the economic development of rural communities, as it encourages families to stay rooted in their localities, promotes employment, and contributes to preserving natural resources.

Agricultural production in Mexico depends on various industrial inputs that optimize the cultivation and harvesting processes. Among these inputs are fertilizers, herbicides, pesticides, and specialized machinery, which have been key to increasing crop efficiency and sustainability. However, this technological dependence also poses significant challenges, as it is associated with high costs and the environmental impact generated by the excessive or inappropriate use of these products (SADER, 2023).

In addition, the sector faces a number of challenges that affect its efficiency and sustainability. Problems such as soil erosion, which reduces the productive capacity of land; the negative impact of pests that can devastate entire crops; the contamination of soils and water bodies due to the intensive use of agrochemicals; and, increasingly alarmingly, labour shortages in agricultural activities, represent critical barriers to the growth of the sector. The lack of agricultural workers limits the ability to perform key tasks such as timely harvesting of produce, which in turn can lead to increased food waste and significant economic losses (Economista,

2023).

In response to these challenges, this project proposes the development of an autonomous robotic vehicle designed to assist in the navigation task within a strawberry crop. Its implementation aims to reduce the physical workload of employees for repetitive and demanding tasks, and the tasks could also include preventive disease monitoring and harvest tracking for short- and medium-term predictions.

1.4 Objectives

1.4.1 General Objective

Developing a navigation planning algorithm for unmanned autonomous vehicles to optimize and facilitate primary food production, specifically strawberry cultivation.

1.4.2 Specific Objectives

Extracting of information from a vision sensor

- Feature detection: Use of computational methods to identify key points in images, such as corners, edges, or textures.
- Image processing: Development of algorithms to transform and analyze the captured images, allowing them to be compared with the current camera view. This will enable the identification of point correspondences between both images, facilitating more accurate analysis and decision-making.
- Visual Feedback: Implementation of control systems based on the information provided by vision sensors, where the data obtained is used in real-time to adjust the behavior of the autonomous vehicle.

Design of Visual Control Strategies

- Path planning: Route planning based on visual and other sensor data, considering physical constraints, vehicle dynamics, and possible obstacles in the

environment.

- Homography-based control: Application of homography theory to estimate the relative motion between the vehicle and a plane in the environment, using consecutive images to guide robot navigation.

Simulation Implementation

- Use of Gazebo with ROS Noetic: Integration of the Gazebo simulator with the ROS Noetic framework to model the environment and simulate sensors and actuators.
- Control implementation: Image-Based Visual Servoing (IBVS) and Nonlinear Model Predictive Control (NMPC) were implemented within the simulation to evaluate and compare their performance.
- Testing and adjustment: Initial testing to assess system performance, identifying faults or areas for improvement, and adjusting parameters as necessary.

Testing in relevant environment

- Initial laboratory testing: Execution of controlled tests in a laboratory environment to verify basic functionality and resolve initial problems before moving on to more complex testing.

1.5 Hypothesis

It is expected that the use of visual control strategies will improve the motion accuracy of the autonomous vehicle for navigation between crop rows, enabling trajectory generation. This hypothesis will be evaluated through simulations and preliminary laboratory testing.

Chapter 2

State of the Art and Theoretical Framework

This chapter discusses visual servoing, a technique that combines image processing and control algorithms to generate motion in a robot using visual information. A detailed analysis of the main approaches in the field of visual servoing is presented, with an emphasis on Image-Based Visual Servoing (IBVS) and Position-Based Visual Servoing (PBVS). Both paradigms represent the fundamental foundations of visual control and are essential for understanding the hybrid methods that have emerged as an alternative in more complex applications. In the context of hybrid methods, the study of the estimation matrix, the fundamental matrix, and the homography matrix is introduced. These matrices are essential mathematical tools in visual processing as they allow key information about geometric transformations between consecutive views to be extracted. In particular, these matrices are essential for decomposing and estimating rotation and translation parameters. In the end, it is analysed how these geometric transformations are integrated into the construction of the homogeneous matrix, which fully describes the spatial relationship between the robot and its environment. This approach provides a solid basis for applications in navigation, robotic manipulation, and other tasks that require accurate interaction with the environment based on visual information.

2.1 Visual Control

Visual control is a branch of robotics that combines image processing and automatic control techniques to guide robot movement. This approach allows a robotic system to interpret visual data in real-time, making it suitable for autonomous navigation, object tracking, and interaction with dynamic environments. This can be further analyzed in Robinson et al. (2023).

An example of object tracking can be the implementation of visual control with image moments, where an observer is applied to track a vehicle with a drone, Kumar et al. (2023), and Shi et al. (2024); the application of image moments can also be seen in Tahri and Chaumette (2005). Another example is the method of navigation based on pre-registered keyframes, where known landmarks are used for localization (see George and Mazel (2013)). One of the things that you want to control with visual control is velocity (see Haviland et al. (2020)) or acceleration (see Mohebbi et al. (2014)).

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

$$\mathbf{e}(t) = \mathbf{s}(\mathbf{m}(t), \mathbf{a}) - \mathbf{s}^*. \tag{2.1}$$

The vector $\mathbf{m}(t)$ represents a set of image features extracted from the visual data at time t (e.g., the image coordinates of points of interest or the centroid of an object). These image measurements are used to compute a vector of k visual features, $\mathbf{s}(\mathbf{m}(t), \mathbf{a})$, where $\mathbf{a}$ is a set of parameters that may encapsulate additional system knowledge (e.g., approximate intrinsic camera parameters or 3D models of objects). The vector $\mathbf{s}^*$ denotes the desired values of the visual features, serving as a reference for the control system.

In robotics, camera control systems vary depending on their location and how they are integrated into a visual servo system. There are two main configurations:

Eye-in-Hand camera: In this configuration, the camera is mounted directly on the end of the robotic manipulator or mobile vehicle. As a result, the camera moves along with the robot, allowing it to observe the environment from a mobile perspective, as can be seen in Saviolo et al. (2024), where the camera is

implemented on a drone, or in Sarabandi et al. (2022), where the camera is on a robotic arm.

Eye-to-Hand camera: In this case, the camera is fixed in a static position, usually on a table or in a space external to the robot. From this location, the camera has a wide view that allows it to observe both the workspace and the robot itself and the target; for example, in Li, Lai, and Pan (2024) and in Ghrairi et al. (2024), the use of a camera in a fixed position that visualizes the robot's workspace can be seen.

This thesis focuses solely on the use of the eye-in-hand camera, as it offers unique advantages for tasks requiring precise interaction with the environment, such as path following, due to its dynamic perspective.

The most straightforward approach to control is to design a velocity controller, eye-in-hand. To do this, we require the relationship between the time variation of $\mathbf{s}$ and the camera velocity. Let the spatial velocity of the camera be denoted by $\mathbf{v}_c = (\mathbf{v}_c, \boldsymbol{\omega}_c)$, in which $\mathbf{L}_s \in \mathbb{R}^{k\times 6}$ is named the interaction matrix related to $\mathbf{s}$. The term *feature Jacobian* is also used somewhat interchangeably in the visual-servo literature.

$$\dot{\mathbf{s}} = \mathbf{L}_s \mathbf{v}_c, \tag{2.2}$$

where $\mathbf{v}_c$ is the instantaneous linear velocity of the origin of the camera frame and $\boldsymbol{\omega}_c$ is the instantaneous angular velocity of the camera frame. In this context, $\dot{\mathbf{s}}$ represents the time derivative of $\mathbf{s}$, i.e., the rate of change of the visual features over time.

Using (2.1) and (2.2), we immediately obtain the relationship between the camera velocity and the time variation of the error:

$$\dot{\mathbf{e}} = \mathbf{L}_e \mathbf{v}_c, \tag{2.3}$$

where $\mathbf{L}_e = \mathbf{L}_s$. Taking $\mathbf{v}_c$ as input to the robot controller, this relationship forms the foundation of visual servo based on velocity. Finally, to compute the

velocities, the Moore-Penrose pseudo-inverse is applied to the interaction matrix, resulting in an approximation or estimate denoted $\hat{\mathbf{L}}_e^+$. Here, λ represents a gain. All of this can be seen in more detail in Chaumette et al. (2016), Chaumette and Hutchinson (2006), and Chaumette (2020).

Using the notation, the control law is as follows:

$$\mathbf{v}_c = -\lambda \hat{\mathbf{L}}_e^+ \mathbf{e}. \tag{2.4}$$

Traditionally, visual control is classified into two main approaches, as described in the following.

2.1.1 Image-Based Visual Servo Control (IBVS)

The IBVS is a visual control approach that uses visual features extracted directly from the image, such as points, lines, or edges, to compute robot control commands by projecting them onto the camera plane. This method eliminates the need to reconstruct 3D models or estimate the robot's pose.

IBVS has been applied in a variety of contexts, such as the control of UAVs, where it guides their movement solely from visual features (He et al., 2023). It has also been used in conjunction with Reinforcement Learning to minimize the error between current and desired visual features (Dani & Bhasin, 2023). In cases of underactuated systems, such as quadcopters, it is combined with Model Predictive Control (MPC) to translate visual commands into actions that respect their physical and dynamic constraints (Roque et al., 2020) or an adaptive control can be used as in the case of (Asl et al., 2014).

The core of IBVS is the interaction matrix, which relates the variation of image features to camera motion. For a single point feature, the interaction matrix is given by:

$$L_s = \begin{bmatrix} -\frac{1}{Z} & 0 & \frac{x}{Z} & xy & -(1+x^2) & y \\ 0 & -\frac{1}{Z} & \frac{y}{Z} & 1+y^2 & -xy & -x \end{bmatrix} \tag{2.5}$$

where x and y represent the image coordinates of the point (in pixels or nor-

malized coordinates, depending on the camera model), and Z denotes the depth of the point in space, which corresponds to the distance between the camera and the object. This matrix plays a fundamental role in computing the camera velocity required to regulate the visual features towards their desired configuration.

2.1.2 Position-Based Visual Servo Control (PBVS)

In Position-Based Visual Servoing (PBVS), the controller uses the estimated 3D position and orientation (pose) of the target and the camera to generate control commands. The error is computed in Cartesian space, typically as the difference between the desired and actual 3D poses. This approach requires reconstructing the position and orientation from visual features using geometric models.

PBVS has been applied in various scenarios, such as controlling cable-driven parallel robots (Dallej et al., 2019) and robots using adaptive sliding mode control (Zhu et al., 2024).

A fundamental component of PBVS is the **position-based interaction matrix**, which describes the relationship between the camera velocity and the variation of the pose in Cartesian space. In general, for a system with six degrees of freedom (6-DoF), the interaction matrix is given by:

$$\mathbf{L}_e = \begin{bmatrix} -\mathbf{I}_3 & [\mathbf{t}_o]_\times \\ 0 & \mathbf{L}_{\theta\mathbf{u}} \end{bmatrix}, \tag{2.6}$$

where $\mathbf{I}_3$ is the 3×3 identity matrix, and $\mathbf{L}_{\theta\mathbf{u}}$ defines the rotational component of the interaction matrix:

$$\mathbf{L}_{\theta\mathbf{u}} = \mathbf{I}_3 - \frac{\theta}{2}[\mathbf{u}]_\times + \left(1 - \frac{\sin\theta}{\operatorname{sinc}\frac{\theta}{2}}\right)[\mathbf{u}]_\times^2. \tag{2.7}$$

In this formulation, $\mathbf{u}$ is a unit vector representing the axis of rotation, and θ is the angle of rotation around that axis, which together correspond to the axis-angle representation of the relative orientation between the current and desired poses.

It is important to note that the structure of the interaction matrix $\mathbf{L}_e$ varies depending on the visual servoing approach—*position-based* or *image-based*—as each framework relies on different reference quantities (pose vs. image features) to describe the system's dynamics.

2.1.3 Differences between IBVS and PBSV

Both approaches have specific applications depending on the system requirements and environmental constraints. For example, IBVS is preferred when working in unfamiliar or poorly structured environments, while PBVS is useful for tasks that require high accuracy in the final pose.

	IBVS	**PBVS**
Domain	Image space	Cartesian space
Involved variables	Visual feature coordinates (x, y, Z)	Relative pose between camera and target
Depth dependency (Z)	High	Low (but requires pose reconstruction)
Complexity of $\mathbf{L}_s$	Lower (directly computed from features)	Higher (requires transformations between frames)
Robot trajectory	Complex in 3D	More predictable and intuitive

Table 2.1: Comparison between IBVS and PBVS

2.2 Homography Matrix

The homography matrix is a fundamental concept in computer vision and visual control, as it describes the geometric relationship between two planar views of a three-dimensional scene. Mathematically, it is a projective transformation that maps points on one image plane to another, which is especially useful when working with cameras and flat surfaces. Roughly speaking, if two images capture the same flat surface from different positions or angles, their corresponding points are related through the equation:

$$\mathbf{x}' = \mathbf{H}\mathbf{x} \tag{2.8}$$

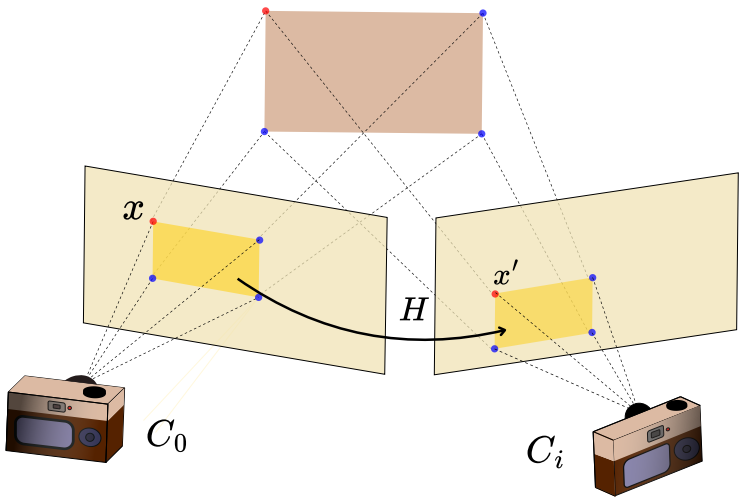


Figure 2.1: Relationship between two cameras (C_0 and C_i)through a homography (H). This process is key in motion estimation and scene reconstruction.

where: $\mathbf{x}$ and $\mathbf{x}'$ are the homogeneous coordinates of points in the 2D images. $\mathbf{H}$ is the 3×3 homography matrix. This matrix encapsulates information on the rotation, translation, and scaling between the two camera views.

2.2.1 Relevance in Visual Servoing

In the context of visual servoing, the homography matrix plays a critical role, particularly in applications such as autonomous navigation, pose estimation, and object tracking. Key uses include:

1. **Motion Estimation:** The homography matrix enables the calculation of camera motion relative to a fixed reference plane, which is essential for adjusting the robot's pose.

2. **Image-Based Control:** In image-based visual servoing (IBVS), the homography matrix facilitates the computation of visual errors directly from image data, eliminating the need for 3D reconstruction.

3. **Hybrid Control:** In hybrid visual servoing approaches, the homography matrix combines information from the image space and the Cartesian space, improving accuracy and robustness.

The homography matrix provides a robust framework for mapping 2D image features to their counterparts in another view, making it a powerful tool for visual servoing systems.

All this can be seen in works such as Wang et al. (2025) where a UAV-based approach is proposed to measure bridge displacement, using four static laser points as a reference and a homography transformation to correct the image. An approach for homography estimation in robotic applications with known periodic motions is proposed in de Marco et al. (2021), using the internal model principle to model complex motions and design observers in the non-compact SL(3) group.

Recently, methods based on deep neural networks have revolutionised homography estimation by automating the process of feature extraction and matching. Unlike traditional approaches, which rely on manual detectors and descriptors such as SIFT or ORB, neural networks can directly learn the relationships between images from the data, eliminating the need for extensive pre-processing; this can be seen in Liu et al. (2023) and Li, Fang, et al. (2024).

2.2.2 Methods for estimating homography

There are several methods for estimating homography, depending on the data and the problem conditions. Common steps include collecting point correspondences and calculating the matrix H:

1. Point correspondences

First, at least 4 points are needed in correspondence between the two images. These points can be:

- Manually marked.
- Automatically detected with methods such as SIFT, SURF, or FAST.
- Matched using feature descriptors.

ORB points are used in the detection, which extracts key features from an image using the ORB algorithm (Oriented FAST and Rotated BRIEF), a widely

used method in computer vision. ORB points are specific locations in an image that contains distinctive information (such as corners, edges, or local textures), making them useful for identifying and comparing parts of one image with another.

2. Estimation Algorithms

a. **Direct Linear Transformation (DLT):** The DLT method is the most commonly used for estimating H. It requires a minimum of 4 correspondences (non-coplanar points) and works by solving a system of linear equations that arises when expanding the homography equation. This method is straightforward, but it can be sensitive to noise.

b. **Robust Methods (RANSAC):** In situations with noisy correspondences or outliers, RANSAC (Random Sample Consensus) is used along with DLT to identify a consistent subset of matching points. This ensures a robust estimation of H.

3. Data Normalization

Before applying DLT, it is common to normalize the input points to improve the numerical stability:

- Center the points at the origin (0,0).
- Scale the points so that their average magnitude is close to 1.

2.3 NMPC - Nonlinear Model Predictive Control

Nonlinear Model Predictive Control (NMPC) is a model-based control approach that optimizes control actions for a system based on its nonlinear dynamic model. In NMPC, a mathematical model of the system is used to predict its future behavior, and an optimization problem is solved in real-time to determine the best control actions that either drive the system to a desired trajectory or

minimize a cost function (e.g., deviation from the target, energy consumption, etc.).

Unlike other predictive control approaches that assume linear dynamics, NMPC can handle systems with nonlinear dynamics, making it suitable for a wide variety of complex applications such as mobile robots, autonomous vehicles, and nonlinear industrial processes.

The basic process in NMPC involves the following steps:

1. **Dynamic modeling:** A nonlinear model is used to describe the system's behavior.
2. **Predictive optimization:** A future optimization problem is solved (typically with a prediction horizon) to find the control actions that minimize the cost or prediction error.
3. **Real-time recalculation:** Upon receiving new system measurements, the controller recalculates the control actions.

This approach is powerful but computationally demanding, as it requires solving an optimization problem in real-time.

Currently, several studies apply NMPC for trajectory planning. For example, in Wu et al. (2023), an NMPC-based approach is proposed for hybrid terrestrial and aerial quadrotors (HyTAQs), which enables safe and efficient hybrid locomotion in unknown environments. Similarly, in Boggio et al. (2023), a "fast" NMPC method is presented that utilizes data-derived information for trajectory planning and control of autonomous vehicles, significantly improving computational efficiency.

2.4 ROS (Robot Operating System)

Currently, the Robot Operating System (ROS) has become one of the most widely used platforms for the development of autonomous systems in robotics and

mechatronics due to its flexibility and ability to integrate multiple hardware and software components. Its applicability can be observed in various works, such as the development and testing of a prototype unmanned aerial vehicle (UAV) with tilt control position control (Flores et al., 2023), the control of an aerial manipulator through teleoperation and virtual reality (Verdín et al., 2021), and simulations for testing mobile robots (Takaya et al., 2016).

These applications highlight the versatility of ROS in different robotics domains, enabling the implementation of complex systems that require efficient module communication, interoperability, and advanced simulation tools.

ROS (Robot Operating System) is an open-source framework for developing robotic systems, offering tools for node communication, topic management, services, actions, and simulation using Gazebo and RViz. Its middleware architecture allows for the integration of devices and algorithms, making it ideal for autonomous systems.

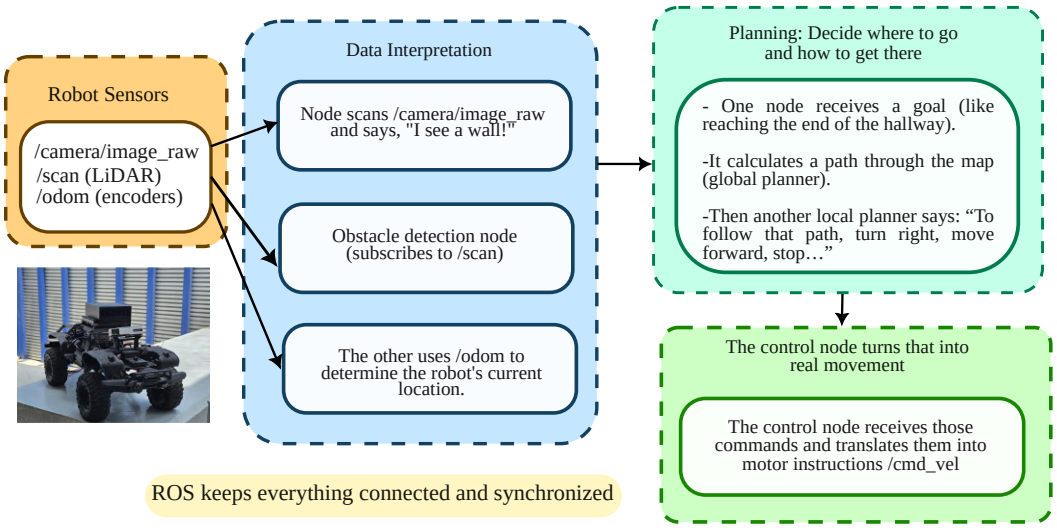


Figure 2.2: This simplified explanation is meant to help understand how ROS connects sensors, processing nodes, and actuators in a robotic system.

An example of how ROS works can be seen in Figure 2.2. Through the robot's sensors, information is received in the form of topics. These topics can contain different types of data, such as images from a camera, measurements from a LiDAR, or readings from encoders. This information is then interpreted to understand the robot's surroundings: for instance, the camera can identify objects or walls; the LiDAR detects obstacles through waves; and the encoders provide

the robot's velocity and help estimate its position on a map.

Based on this sensory interpretation, decisions can be made about the robot's behavior, such as stopping, turning right or left, or continuing its movement. This type of planning is translated into a control signal sent to the `/cmd_vel`topic, which carries the desired velocity information that is then executed by the robot's motors.

Robotic systems often rely on ROS for image processing tasks, particularly in visual servoing applications. Through integration with libraries like OpenCV and `image_transport`, ROS enables real-time image handling, camera calibration, and feature extraction—key components for object detection, tracking, autonomous navigation, and localization.

Many leading robotics companies, including Clearpath Robotics, Fetch Robotics, and Boston Dynamics, have incorporated ROS into their platforms. They apply it in areas such as autonomous vehicles, warehouse automation, and the development of advanced robots such as Spot and Atlas. Its open-source nature, combined with strong community support, positions ROS as a powerful and cost-effective tool in the robotics industry.

Chapter 3

Methodology

This chapter details the methodology used for the development and analysis of the autonomous navigation and control system of the mobile robot. It describes the procedures followed, the tools used, and the simulation and experimentation techniques that allowed the technical challenges identified in the previous stages of the research to be addressed.

3.1 Simulation

3.1.1 First visual control test

In a first visual control test, a proportional control based on feature detection was implemented using the KLT (Kanade–Lucas–Tomasi) method, which tracks distinctive image features across frames. During this process, four key points were identified and a desired location was set for them within a blue square. The control was implemented in the Rover model within PX4, where the vehicle advanced based on the error between the locations of the detected points and their desired positions. For this purpose, a proportional control was used that adjusted the forward speed as a function of the magnitude of the error. As the Rover

moved, the system continuously recalculated the difference between the current position of the points and their target location, gradually reducing the distance until the desired alignment was reached. Once the error was reduced to zero, the Rover stopped its movement, successfully completing the visual positioning task.

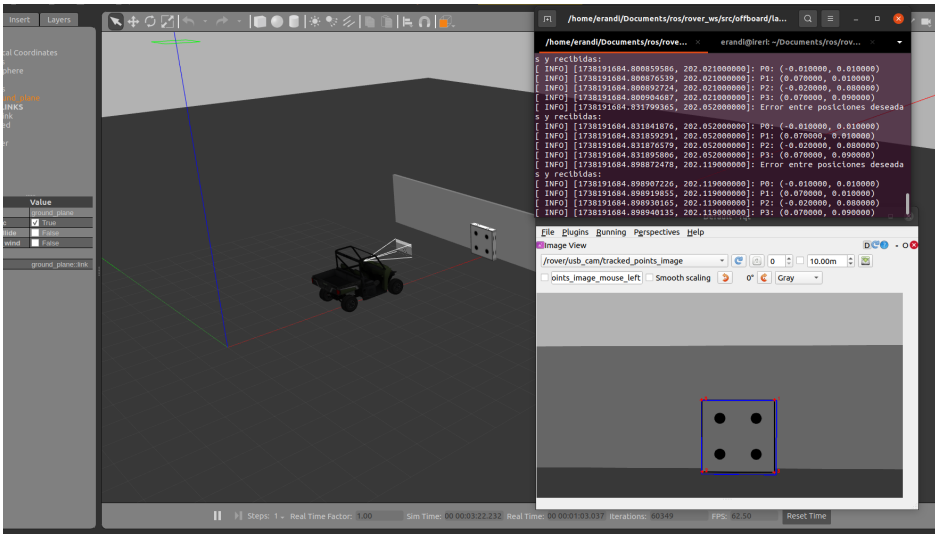


Figure 3.1: Rover trajectory in response to KLT-based visual control. The vehicle stops when the error between the detected and desired points disappears.

When using the Rover model, it was observed that the vehicle did not move correctly when attempting to reverse. This issue arises because, in PX4, motor control is designed primarily for drones rather than ground vehicles, which limits its functionality. To analyze this behavior, different values were sent to the `/mavros/setpoint_velocity/cmd_vel` topic, and the system's responses were recorded. However, the vehicle never moved in reverse, as shown in Table 3.1 which details the executed movements. Due to this limitation, a different model was chosen to enable reverse movement.

Given the difficulty with the rover, the decision was made to use the TurtleBot for future visual control testing. This choice was based on the advantages offered by the TurtleBot, particularly in the simulation field. Being fully integrated with ROS, it facilitates the performance of accurate simulations on Gazebo, allowing the testing of control and perception algorithms without the need to resort to physical hardware. In addition, TurtleBot is a flexible and accessible platform, making it an ideal choice to simulate complex scenarios and evaluate algorithms' performance under controlled conditions. Its compatibility with sensors such as

/mavros/setpoint_velocity/cmd_vel	Movement
`vel.twist.linear.x = +a`	Turns right
`vel.twist.linear.x = -a`	Turns left
`vel.twist.linear.y = +a`	Moves forward
`vel.twist.linear.y = -a`	Turns left
`vel.twist.linear.z = +a`	Moves forward
`vel.twist.linear.z = -a`	Moves forward

Table 3.1: Rover Logic

cameras and LIDAR, together with the ease of integrating navigation and visual control tools, positions it as a robust and reliable solution for the development and validation of visual control systems.

3.1.2 Creation of the Strawberry Growing Environment

In order to validate the robot's navigation and autonomous control techniques, a simulated environment was designed to replicate the conditions of a strawberry crop. This environment allowed testing in a controlled manner, eliminating the risks associated with real-world experimentation. The 3D terrain models were created using Blender modeling software, known for its ability to generate detailed and accurate representations in three-dimensional environments.

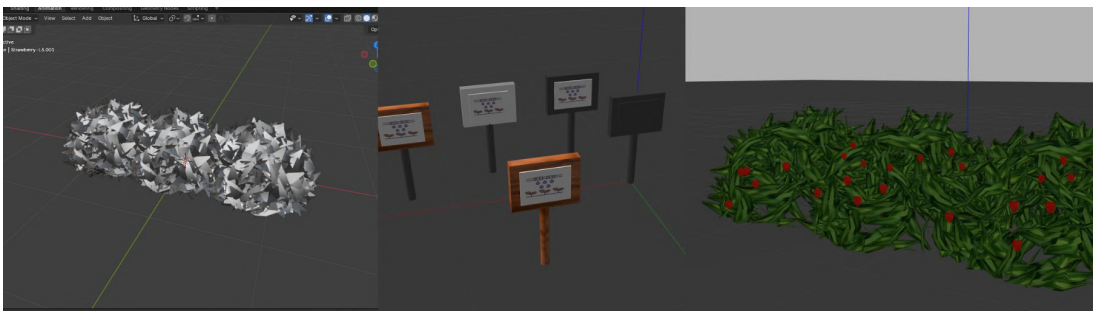


Figure 3.2: Initial design of the 3D models for the strawberry world.

In this process, it was necessary to adjust the scale of the strawberries and bushes, considering factors such as the number of strawberries that each bush would have and the appropriate size in relation to the robot to be used, the TurtleBot. In addition, the quality of the models was evaluated, as a higher resolution

would require more computational resources and would affect the performance of the simulation. Therefore, a middle ground was established in terms of quality to ensure a good balance between detail and efficiency, applying the same criteria to the ground model and other elements of the environment.

Other important models were the signs, which were designed entirely in Gazebo. These signs contain various images and have the function of guiding the vehicle through the strawberry crop.

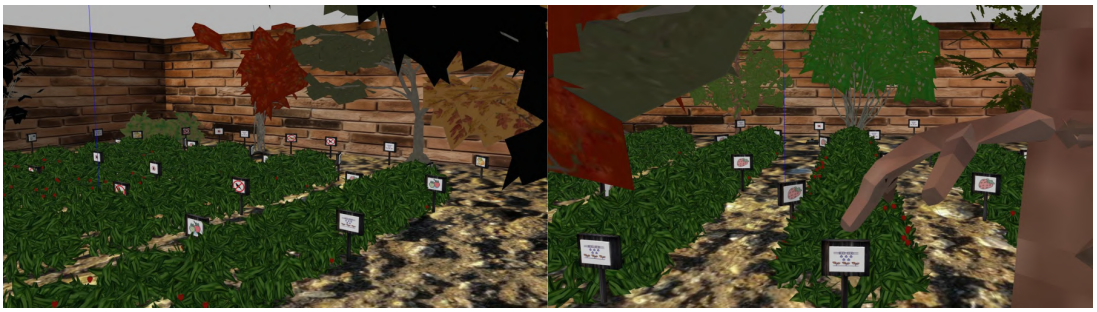


Figure 3.3: Final view of the strawberry world with all elements integrated.

Once completed, the models were exported and optimized for integration into the Gazebo simulator, allowing simulations to be performed within the ROS ecosystem. This simulated environment not only facilitated the evaluation of the system's performance in terms of navigation and control, but also provided a platform to test the robot's behavior in various scenarios without the need for immediate physical adjustments.

3.1.3 Image-Based Control Using Homography

Vision

Once the environment was finalized, a test was carried out both in the simulation environment and on the vehicle, evaluating an image-based control system. The homography used in this system was computed from ORB (Oriented FAST and Rotated BRIEF) feature points. For this purpose, a monocular camera was used together with a visual memory mechanism, allowing the estimation of the homography matrix.

First, ORB (Oriented FAST and Rotated BRIEF) points are detected, which

appear at corners and edges within an image. These key points allow distinctive features to be extracted from each image, facilitating comparison between the current image and images stored in visual memory.

The visual memory system operates as follows: first, a pre-mapping is performed in which images are captured of the desired trajectory that the robot, in this case, the TurtleBot, will follow. These images are stored sequentially, following the order in which they were acquired, named as image0.png, image1.png, image2.png, and so on. The visual memory can contain n images.

The reference images used for visual memory were captured at intervals of approximately 1.5 meters along the robot's path. This distance was chosen as a compromise between coverage and reliability, since capturing images at greater distances led to a decrease in the accuracy of visual feature matching. Larger intervals resulted in significant variations between consecutive images, making it more difficult to establish robust point correspondences, which are essential for tasks such as homography estimation and visual localization.

Special consideration was given to the sections where the robot had to turn. In these cases, image acquisition was based on the angular orientation of the robot. This is because large changes in viewpoint can significantly alter the appearance of the scene, reducing the number of reliable feature correspondences between the current view and the stored reference images. Therefore, during turns, reference images were taken more frequently and with smaller angular differences, ensuring that the visual information remained sufficiently similar to facilitate consistent and accurate feature matching.

During navigation, the images stored in the visual memory are compared with the real-time images captured by the camera. To perform this comparison, key points in both images are extracted and described using the ORB detector. Then, the obtained descriptors are converted to a format suitable for FLANN (Fast Library for Approximate Nearest Neighbors) matching using an LSH (Locality-Sensitive Hashing) based matcher. (Guzmán et al., 2024)

Once the descriptors have been matched, Lowe's ratio test is applied to filter out ambiguous correspondences by comparing the distance between the two closest matches, and obtain a set of reliable matches between the reference image

(stored in visual memory) and the current image. From these correspondences, the coordinates of the matched points are extracted and used to estimate the transformation between the two images.

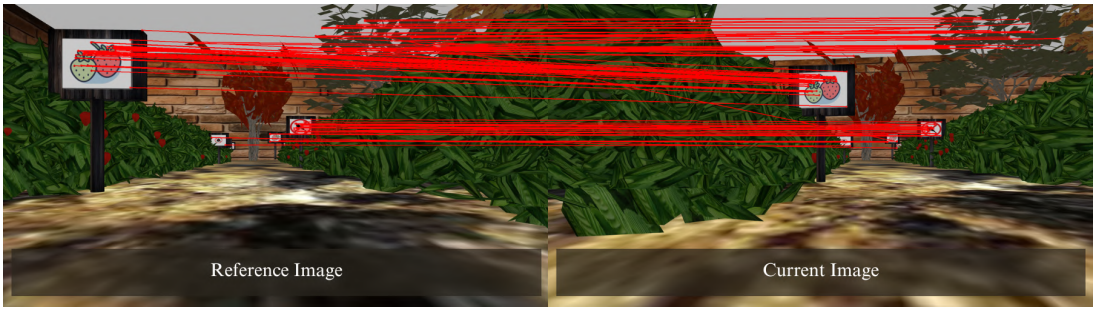


Figure 3.4: The image shows the result of key point matching between two views of a scene, using feature detection and description. The red lines indicate the correspondences between the detected points in both images.

To estimate the homography matrix H, first it is verified that there are at least four matching points between the images. Then, the `cv::findHomography` function from OpenCV is used with the RANSAC method to compute the homography matrix, which helps mitigate the impact of potential incorrect correspondences. The homography matrix is adjusted using the camera calibration matrix K, as shown in the following equation:

$$H_{\text{calibrated}} = K^{-1} H K \tag{3.1}$$

This step is crucial for converting the homography to the camera coordinate system, taking into account the intrinsic parameters of the camera. Finally, to normalize the homography and simplify its use in further calculations, it is often scaled such that the top-left element is 1, as shown by the following operation:

$$H_{\text{norm}} = \frac{H}{H_{1,1}} \tag{3.2}$$

This normalization ensures that the homography matrix is stable and easier to apply, especially when working with image transformations. All of this can be better visualized in Table 3.2.

Algorithm 1

Input: Reference image $refImg$, Current image $currImg$, Camera calibration matrix K
Output: Homography matrix H

// Step 1: Detect keypoints and compute descriptors using ORB
$(keypointsRef, descriptorsRef) \leftarrow ORB\_Detector(refImg)$
$(keypointsCurr, descriptorsCurr) \leftarrow ORB\_Detector(currImg)$

// Step 2: Match descriptors using Hamming distance
$matches \leftarrow BruteForce\_Hamming(descriptorsRef, descriptorsCurr)$
$goodMatches \leftarrow Filter\_Matches(matches, threshold = 0.7)$

// Step 3: Extract matched keypoints
$refPoints2f \leftarrow Get\_Points(keypointsRef, goodMatches.queryIdx)$
$currentPoints2f \leftarrow Get\_Points(keypointsCurr, goodMatches.trainIdx)$

// Step 4: Compute homography using RANSAC
if $size(refPoints2f) \geq 4$ **then**
    $H \leftarrow findHomography(refPoints2f, currentPoints2f, RANSAC)$
end if

// Step 5: Normalize using calibration matrix
$H \leftarrow K^{-1}HK$
$H \leftarrow H/H_{1,1}$

Table 3.2: Homography Estimation Algorithm

This approach allows TurtleBot to follow the desired trajectory based on visual information, without relying exclusively on traditional position sensors. In addition, detected key points and pairings can be visualized in real time, facilitating process validation.

Parameter Recovery from Homography

As described in Chapter 2, the homography matrix is a 3×3 matrix that provides the information necessary to implement it in control, specifically rotation and translation in this case. However, the homography matrix also gives us more information, such as the normal and the distance from this plane to the camera, as described below.

Recall that, in this case, only rotation and translation will be used. The process of recovering the camera parameters (rotation R, translation t, normal n, and distance to the plane distancePlane) from the homography matrix H involves several steps, based on the singular value decomposition (SVD) of the homography matrix. The details of this process are as follows:

1. SVD Decomposition of the Homography:

The homography matrix H is decomposed into three matrices U, Σ, and V using singular value decomposition (SVD):

$$H = U\Sigma V^T \tag{3.3}$$

U : The columns of the matrix U are the eigenvectors of the matrix HH^T. This matrix will be of size $m \times m$, where m is the number of rows of H.

Σ : The diagonal matrix Σ contains the singular values of H, which are the square roots of the eigenvalues of H^TH or HH^T. The singular values are always non-negative and are ordered in descending order along the diagonal of Σ.

V : The columns of the matrix V are the eigenvectors of the matrix H^TH. This matrix will be of size $n \times n$, where n is the number of columns of H.

2. Calculation of Normalized Singular Values:

The singular values s_1, s_2, and s_3 of the matrix Σ provide key information about the geometric structure of the homography. The relationship between s_1 and s_3 is calculated as follows:

$$s_1 = \frac{\Sigma(0)}{\Sigma(1)}, \quad s_3 = \frac{\Sigma(2)}{\Sigma(1)} \tag{3.4}$$

and the difference between these values, denoted as ζ, is calculated as:

$$\zeta = s_1 - s_3 \tag{3.5}$$

If ζ is small, it means the images are nearly in the same plane.

3. Calculation of Geometric Parameters

The geometric parameters a_1 and b_1 are calculated using the singular values s_1 and s_3, which describe the distortion introduced by the homography:

$$a_1 = \sqrt{1 - s_3^2}, \quad b_1 = \sqrt{s_1^2 - 1} \tag{3.6}$$

These parameters must be normalized to ensure proper scaling and numerical stability in the subsequent computations. Normalization is performed by dividing each parameter by its corresponding magnitude.

First, the values a_1 and b_1 are normalized as follows:

$$a = \frac{a_1}{\sqrt{a_1^2 + b_1^2}}, \quad b = \frac{b_1}{\sqrt{a_1^2 + b_1^2}} \tag{3.7}$$

Next, the terms related to the singular values are normalized:

$$c = \frac{1 + s_1 s_3}{\sqrt{(1 + s_1 s_3)^2 + (a_1 b_1)^2}}, \quad d = \frac{a_1 b_1}{\sqrt{(1 + s_1 s_3)^2 + (a_1 b_1)^2}} \tag{3.8}$$

The terms used in the translation vector calculations are also normalized:

$$e = \frac{-b/s_1}{\sqrt{(b/s_1)^2 + (a/s_3)^2}}, \quad f = \frac{-a/s_3}{\sqrt{(b/s_1)^2 + (a/s_3)^2}} \tag{3.9}$$

These normalized values ensure that all computed vectors maintain unit magnitude, making geometric calculations stable and independent of scale.

4. Recovery of Normals and Rotation Solutions

Using the vectors v_1 and v_3, extracted from the matrix V, the two possible normal vectors to the plane, n_1 and n_2, are computed:

$$n_1 = b v_1 - a v_3, \quad n_2 = b v_1 + a v_3 \tag{3.10}$$

These normals define two possible solutions for the orientation of the plane in 3D space.

Next, two possible rotation matrices, R_1 and R_2, are derived. These represent two possible orientations of the camera, and they are constructed using the singular vector matrices U and V along with an intermediate transformation matrix tmp:

$$R_1 = U(\mathrm{G} \cdot V), \quad R_2 = U(\mathrm{G}^T \cdot V) \tag{3.11}$$

where the matrix G is defined as:

$$\mathrm{G} = \begin{bmatrix} c & 0 & d \\ 0 & 1 & 0 \\ -d & 0 & c \end{bmatrix} \tag{3.12}$$

This matrix accounts for the necessary transformation between the estimated frame and the original coordinate system.

The corresponding translation vectors for each solution are given by:

$$t_1 = ev_1 + fv_3, \quad t_2 = ev_1 - fv_3 \tag{3.13}$$

5. Selection of the Best Solution

Since homography provides two possible solutions for camera pose, the correct one must be determined based on the orientation of the normal vectors n_1 and n_2. This is done by evaluating the Z-component of the normals. If $Z(n_1)$ is greater than $Z(n_2)$, then the first solution is chosen:

$$R = R_1, \quad t = t_1, \quad n = n_1 \tag{3.14}$$

Otherwise, the second solution is selected:

$$R = R_2, \quad t = t_2, \quad n = n_2 \tag{3.15}$$

If the normal vector points in the opposite direction of what is expected, it is inverted along with its corresponding translation:

$$\text{If } Z(n_1) < 0, \quad n_1 = -n_1, \quad t_1 = -t_1 \tag{3.16}$$

$$\text{If } Z(n_2) < 0, \quad n_2 = -n_2, \quad t_2 = -t_2 \tag{3.17}$$

These adjustments ensure that the camera orientation is physically meaningful.

6. Calculation of Distance to the Plane

The distance of the camera to the estimated plane is computed using the singular value ratio ζ, which measures the deviation from a pure planar transformation:

$$\zeta = s_1 - s_3 \tag{3.18}$$

Since a smaller ζ indicates that the homography represents a nearly planar transformation, the estimated distance is given by:

$$\text{distancePlane} = \frac{1}{\zeta} \tag{3.19}$$

This provides an approximation of the relative depth of the scene.

Algorithm 2

Input: Homography matrix H
Output: Rotation R, Translation t, Normal vector n, Plane distance d
Compute SVD: $H = USV^T$
Transpose V
if $\det(V) < 0$ **then**
    $V \leftarrow -V$, $S \leftarrow -S$
end if
if $\det(U) < 0$ **then**
    $U \leftarrow -U$, $S \leftarrow -S$
end if
Compute scale ratios: $s_1 = S(0)/S(1)$, $s_3 = S(2)/S(1)$
$zeta = s_1 - s_3$
if $|zeta| < \epsilon$ **then**
    Set identity transformation and return
end if
Compute normal vectors n_1 and n_2
Define matrices M_+ and M_-
Compute possible rotations: $R_1 \leftarrow UM_+^TV^T$, $R_2 \leftarrow UM_-^TV^T$
Compute possible translations: $t_1 \leftarrow \text{vex}(UM_+SU^T)$, $t_2 \leftarrow \text{vex}(UM_-SU^T)$
Compute plane distance: $d = 1/zeta$

Table 3.3: Decomposition and Motion Recovery from Homography Algorithm

Control

With the information of rotation and translation, a homogeneous matrix **cMo** was constructed, which contains the position and orientation of the camera relative to a reference frame. This matrix provided the necessary information to calculate the coordinates x and y, which are used in the interaction matrix L_s (see equation 2.3). In particular, the coordinates x and y are obtained as follows:

$$x = \frac{t_x}{t_z}, \quad y = \frac{t_y}{t_z}, \quad Z = t_z \tag{3.20}$$

where t_x, t_y, and t_z are the translation terms of the homogeneous matrix **cMo**, and Z represents the depth, i.e., the distance from the camera to the object.

Also, the visual error was calculated, which is defined as the difference between the current features of the camera and the desired ideal values. The ideal translation and rotation features were set as:

$$s_t^* = (0, 0), \quad s_R^* = (0, 0) \tag{3.21}$$

The error in translation and rotation is obtained by subtracting the current values s_t and s_R, which depend on the homogeneous matrix, from the ideal values s_t^* and s_R^*:

$$e_t = s_t - s_t^*, \quad e_R = s_R - s_R^* \tag{3.22}$$

This error is used to compute the necessary camera movements that minimize the difference between the current and desired state, through the control law described in **equation** (2.4).

The implemented control law generates six velocities, but since the TurtleBot has only two degrees of freedom, only two are needed: one linear and one angular. The linear velocity is applied along the X-axis, enabling forward and backward movement, while the angular velocity acts on the Z-axis to perform turns.

Algorithm 3

Input: Translation vector t, Rotation matrix R
Output: Control velocities v

// Initialize homogeneous transformation matrix
$cMo \leftarrow BuildFrom(t, R)$

// Construct current visual features
$s_t \leftarrow vpFeatureTranslation(cMo)$
$s_R \leftarrow vpFeatureThetaU(cMo)$

// Construct desired visual features (zero initialization)
$s_t^* \leftarrow vpFeatureTranslation(0, 0, 0)$
$s_R^* \leftarrow vpFeatureThetaU(0, 0, 0)$

// Configure visual servoing task
$task.setServo(EYEINHAND\_CAMERA)$
$task.setInteractionMatrixType(CURRENT)$
$task.setLambda(0.165)$
$task.addFeature(s_t, s_t^*)$
$task.addFeature(s_R, s_R^*)$

// Control loop
repeat
  if $received\_trn$ **then**
    $cMo \leftarrow BuildFrom(t, R)$
    $s_t.buildFrom(cMo)$
    $s_R.buildFrom(cMo)$
    // Compute control law
    $v \leftarrow task.computeControlLaw()$
    $error \leftarrow sumSquare(task.getError())$
    // Publish error
    $errorMsg.data \leftarrow error$
    $errorPub.publish(errorMsg)$
    // Assign control velocities
    $velMsg.linear.x \leftarrow -v_2$
    $velMsg.angular.z \leftarrow v_4 \times 2.1$
    $velPub.publish(velMsg)$
  else
    print("Waiting for information...")
  end if
until $indexValue \geq 16.0$

Table 3.4: VISP-based Visual Servoing Algorithm

The control law error determines the robot's speed: the greater the error, the higher the supplied speed. If the error is zero, the robot stops.

The system uses visual memory to guide movement. Initially, a reference image is captured and compared to the current image from the TurtleBot's camera. The robot moves forward until both images are sufficiently similar. At that point, a new reference image is selected on the basis of the homography.

Homography provides information on the transformation between the two images. When the homography matrix closely resembles an identity matrix, it indicates that the images are nearly identical. An error threshold is set to determine when to switch the reference image. This process repeats sequentially until the last stored image in visual memory is reached, at which point the robot stops.

The following figure illustrates the system's operation.

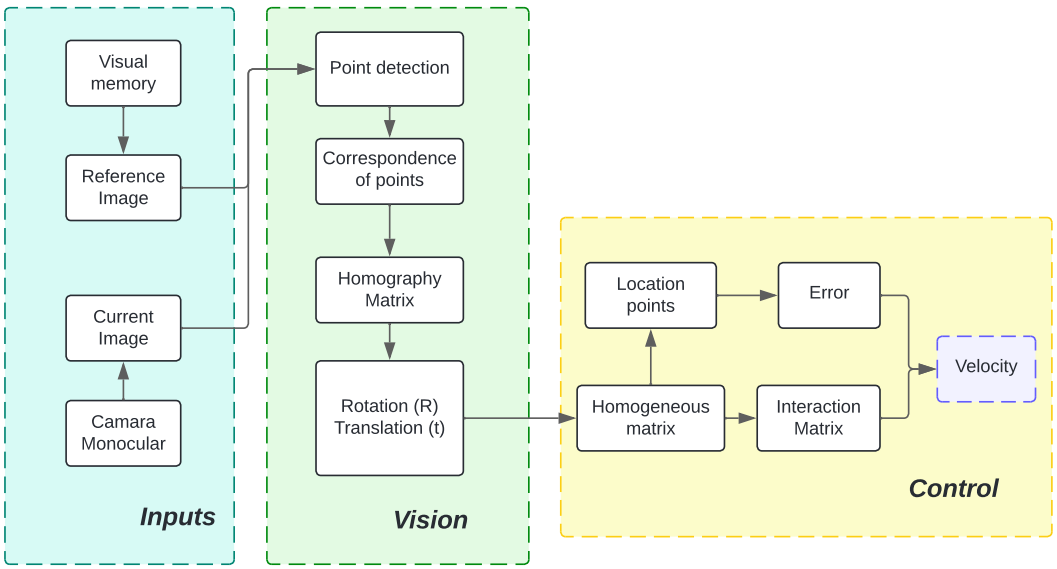


Figure 3.5: Vision and Control System Schematic

3.1.4 Predictive Visual Path Following Using Homography in Autonomous Robots

As observed in Chapter 2, a Nonlinear Model Predictive Control (NMPC) relies on a system model to predict its behavior and optimize control at each time step. In the case of a ground vehicle, the control is typically based on its kinematic model, which considers only velocities and positions, excluding forces and accelerations.

Kinematic Model of a Differential Drive Vehicle

For a differential drive vehicle, the kinematic model outputs are:

- **Linear velocity in x (v_x)**, which defines the forward or backward motion.
- **Angular velocity in z (ω_z)**, which defines the turning rate around the vertical axis.

The differential kinematic model at coordinates (x, y, θ), where θ represents vehicle orientation, is expressed as follows:

$$\begin{cases} \dot{x} = v_x \cos(\theta) \\ \dot{y} = v_x \sin(\theta) \\ \dot{\theta} = \omega_z \end{cases} \tag{3.23}$$

Where:
- x, y are the vehicle's position in the plane.
- θ is the vehicle's orientation relative to a fixed reference frame.
- v_x and ω_z are the control inputs optimized by the NMPC.

The NMPC uses this prediction model to compute the optimal control inputs (v_x, ω_z), ensuring that the vehicle follows the desired trajectory while satisfying constraints such as speed and turning limits.

Relation with the Homography Matrix

The homography equation:

$$H_k^t = ({}_t^c R_c)\left(I - \frac{{}^c t_{tc}\mathbf{n}^T}{d}\right) \tag{3.24}$$

describes the transformation between two camera views, modeling how the image perspective changes due to the vehicle's motion. This transformation depends on the rotation ${}_t^c R_c$ and translation ${}^c t_{tc}$, which can be predicted using the vehicle's kinematic model within the NMPC framework. Here, $\mathbf{n}$ is the normal vector of the observed planar surface in the scene, and d is the distance from the camera to that plane along the normal direction.

- The translation ${}^c t_{tc}$ is influenced by the linear velocity v_x.
- The rotation ${}_t^c R_c$ is governed by the angular velocity ω_z.

By leveraging these dependencies, the NMPC predicts the future homography based on control inputs (v_x, ω_z), enabling:

1. Modeling image deformation at each prediction step.
2. Adjusting controls to minimize the difference between predicted and desired homography.
3. Optimizing vehicle motion for alignment with the image horizon or a visual target.

In this work, a subset of the homography matrix $\mathbf{H}_k^{i^*}$ is selected, focusing on elements that capture key transformations affecting the image. Specifically, the selected components correspond to the indices:

$$h = \{11, 12, 13, 31, 32, 33\} \tag{3.25}$$

$$
\begin{aligned}
\mathbf{h}^{i^*}_{l,11} &= -\frac{n_x}{d}\left({}^{b_k}x_{b_l}\sin\phi_k + {}^{b_k}y_{b_l}\cos\phi_k\right) + \frac{n_x}{d}\left(-{}^{c_k}x_{ci^*}\cos\phi_k + {}^{c_k}z_{ci^*}\sin\phi_k\right) + \cos\left(\phi_k + {}^{b_k}\theta_{b_l}\right),\\
\mathbf{h}^{i^*}_{l,12} &= -\frac{n_y}{d}\left({}^{b_k}x_{b_l}\sin\phi_k + {}^{b_k}y_{b_l}\cos\phi_k\right) + \frac{n_y}{d}\left(-{}^{c_k}x_{ci^*}\cos\phi_k + {}^{c_k}z_{ci^*}\sin\phi_k\right),\\
\mathbf{h}^{i^*}_{l,13} &= -\frac{n_z}{d}\left({}^{b_k}x_{b_l}\sin\phi_k + {}^{b_k}y_{b_l}\cos\phi_k\right) + \frac{n_z}{d}\left(-{}^{c_k}x_{ci^*}\cos\phi_k + {}^{c_k}z_{ci^*}\sin\phi_k\right) - \sin\left(\phi_k + {}^{b_k}\theta_{b_l}\right),\\
\mathbf{h}^{i^*}_{l,21} &= 0, \quad \mathbf{h}^{i^*}_{l,22} = 1, \quad \mathbf{h}^{i^*}_{l,23} = 0,\\
\mathbf{h}^{i^*}_{l,31} &= \frac{n_x}{d}\left({}^{b_k}x_{b_l}\cos\phi_k - {}^{b_k}y_{b_l}\sin\phi_k\right) + \frac{n_x}{d}\left(-{}^{c_k}x_{ci^*}\sin\phi_k - {}^{c_k}z_{ci^*}\cos\phi_k\right) + \sin\left(\phi_k + {}^{b_k}\theta_{b_l}\right),\\
\mathbf{h}^{i^*}_{l,32} &= \frac{n_y}{d}\left({}^{b_k}x_{b_l}\cos\phi_k - {}^{b_k}y_{b_l}\sin\phi_k\right) + \frac{n_y}{d}\left(-{}^{c_k}x_{ci^*}\sin\phi_k - {}^{c_k}z_{ci^*}\cos\phi_k\right),\\
\mathbf{h}^{i^*}_{l,33} &= \frac{n_z}{d}\left({}^{b_k}x_{b_l}\cos\phi_k - {}^{b_k}y_{b_l}\sin\phi_k\right) + \frac{n_z}{d}\left(-{}^{c_k}x_{ci^*}\sin\phi_k - {}^{c_k}z_{ci^*}\cos\phi_k\right) + \cos\left(\phi_k + {}^{b_k}\theta_{b_l}\right),
\end{aligned}
\tag{3.26}
$$

The indices in (3.25) are used to reference the calculation of the predicted homography matrix, which will be presented in the following equation:

The equation (3.26) is explained in more detail in (Murillo, 2021), which outlines how the homography prediction equation is derived.

Control System for Target Image Transition

In order to achieve smoother control during image-based visual servoing, especially when changing the target image, two control mechanisms were incorporated. The first mechanism ensures a constant linear velocity along the x-axis, allowing the robot to perform its operations more efficiently and accurately. On the other hand, the second mechanism enables the robot to gradually reduce its speed as it approaches the target image. The choice between these mechanisms depends on whether the current image is compared with the last reference image. If this is the case, the deceleration mechanism is applied; otherwise, the mechanism that maintains a constant linear velocity is used. This method ensures that the control signal remains within the specified limits, preventing sudden fluctuations and facilitating a smooth transition between tasks (number of images) within the system.

This can also be observed in the following pseudocode, where it is specified when adjustments are made to the linear velocity in x and the angular velocity in z. These changes are determined on the basis of the values of the homography.

Algorithm 4

Input: Homogeneous transformation matrix H, control parameters
Output: Linear and angular velocities v_x, ω_z

```
while not rospy.is_shutdown() do
  if ShutDown == FALSE then
    if Number image < (Total number of images - 1) then
      Vector linear velocities ← smooth_function_exp(V_k[0], 0.19, 0.20)
    else
      if homography conditions are TRUE then
        ShutDown ← TRUE
      else
        Vector linear velocities ← smooth_function_phi(V_k[0] * 2, 0.19,
        0.20)
      end if
    end if
    if homo_matrix.data[6] > 0.08 AND .homo_matrix.data[2] < -0.08
    then
      angular_velocity[0] ← smooth_function_exp(angular_velocity[0],
      0.18, 0.20) * -1
    else if homo_matrix.data[6] < -0.08 AND homo_matrix.data[2] > 0.08
    then
      angular_velocity[0] ← smooth_function_exp(angular_velocity[0],
      0.18, 0.20)
    end if
  end if
end while
```

Table 3.5: Control with smooth transition Algorithm

3.2 Implementation of Control in the Mobile Jet-Bot Robot

In order to maintain a constant speed in the image changes and reduce the trajectory time, a controller based on Non Linear Predictive Control (NMPC) was implemented. For its development and validation, it was used in a JetBot equipped with a Jetson Nano (see Appendix A), running Ubuntu 18.04 as operating system using ROS Melodic. This platform allowed real-time calculations for trajectory prediction and optimization, ensuring efficient performance in navigation and image processing.

To establish communication between the JetBot and the laptop, a Wi-Fi connection was initially configured. This connection was achieved using ROSBridge, which is a fundamental tool in the Robot Operating System (ROS) ecosystem. Once the connection is established, it is crucial to configure the appropriate environment variables in the `.bashrc` file of the laptop. At the end of this file, the following lines should be added, ensuring to replace `<IP_OF_MASTER>` and `<YOUR_LOCAL_IP>` with the corresponding IP addresses for the network setup:

```
export ROS_MASTER_URI=http://<IP_OF_MASTER>:11311
export ROS_IP=<YOUR_LOCAL_IP>
export ROS_HOSTNAME=<YOUR_LOCAL_IP>
```

In the case of the JetBot, the same lines should be included in the `.bashrc` file, where the master IP will be that of the laptop, while the local IP should be that of the JetBot. This configuration is essential for both devices to communicate correctly and coordinate their actions. To execute the necessary `roslaunch` commands to use the camera and control the JetBot's motors without having to run these commands directly from the Jetson Nano, the command `ssh user@ip` is used. In this command, "user" represents the username of the Jetson Nano, and "ip" is the IP address of the Jetson or JetBot. By running this command from the laptop, communication with the remote device is initiated, allowing for joint execution with `roscore` and enabling the necessary programs to be launched via `roslaunch`. It is important to highlight that, to ensure stable connectivity and avoid potential communication issues, it is fundamental for both devices to be configured with the same time zone.

Chapter 4

Results

This chapter focuses on the presentation of the results obtained from the simulations carried out in the Gazebo environment, as well as the practical implementation carried out in an environment relevant to the study. Each of the results is discussed, providing a clear context for understanding their significance and implications.

The graphs included are fundamental to visualize and compare the data obtained from the approaches, seen in the previous chapter. A detailed description of each graph is provided, explaining the variables represented and the trends observed. In addition, the similarities and differences between the results of the simulations and those obtained in the actual implementation are discussed, highlighting how these differences may influence the interpretation of the data and the validation of the model.

4.1 Simulation results

The data presented in this chapter have been obtained from the two main simulations described in the previous chapter: image-based visual control and Nonlinear Model Prediction Control (NMPC). The results are shown below:

4.1.1 Image-Based Visual Control Results

The Figure 4.1 shows a simulated environment in Gazebo, where the image-based control system has been implemented for the navigation of a TurtleBot in a strawberry crop. The control strategy is based on homography, allowing the robot to estimate its movement and adjust its trajectory based on the captured visual information. This approach seeks to improve the autonomy of the robot in agricultural environments, facilitating monitoring and exploration tasks.

Figure 4.1: The TurtleBot navigates between rows of strawberries, scanning the environment. The code developed for this project is available on GitHub: github.com/erandivg/strawberry_field_ws

In image-based visual control, a test was conducted using 16 reference images for visual memory. These images were captured during a pre-trajectory and then stored. Once this process was completed, the program was executed within the virtual strawberry environment, thus initiating the visual trajectory. The trajectory followed by the TurtleBot can be seen in Figure 4.2. During the trajectory, the TurtleBot experienced variations in its speed due to the nature of its control. It dynamically adjusts its speed, increasing or decreasing it based on the detected error relative to the reference image used at each moment. As the vehicle approaches the end of a task, it tends to slow down or even move in reverse to achieve a better alignment between the reference and the current image.

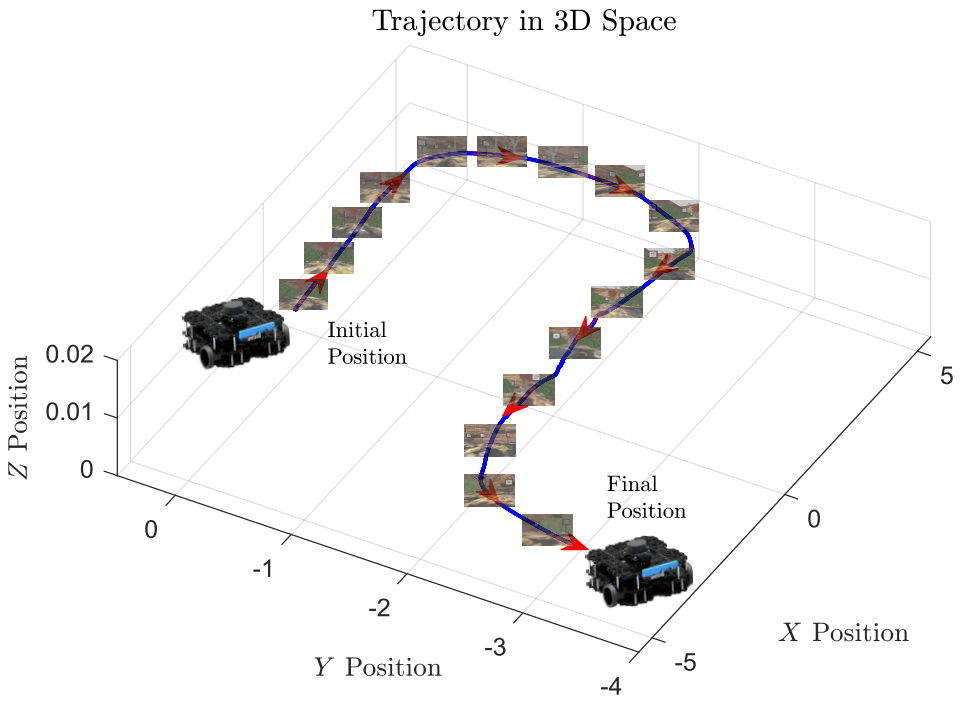


Figure 4.2: 3D trajectory of the robot, illustrating its movement across the environment.

This behavior can be observed in Figure 4.3, where the vehicle's linear velocity v_x and angular velocity ω_z are plotted alongside the obtained homography matrix throughout the navigation process. The homography serves as an indicator of the vehicle's relative position concerning the reference image, providing insight into how the system adapts to minimize the difference between the current and target views.

As the vehicle moves forward, the homography progressively approaches the identity matrix, signifying that the visual information captured by the onboard camera is becoming increasingly similar to the reference image. This gradual convergence reflects the control strategy to guide the vehicle toward the desired trajectory. Additionally, the graph reveals abrupt variations in the homography values, which correspond to transitions between different reference images. These discontinuities indicate moments when the system updates its target visual reference, prompting the control algorithm to adjust the vehicle's motion accordingly.

Despite these sudden changes, it is evident that the control system consis-

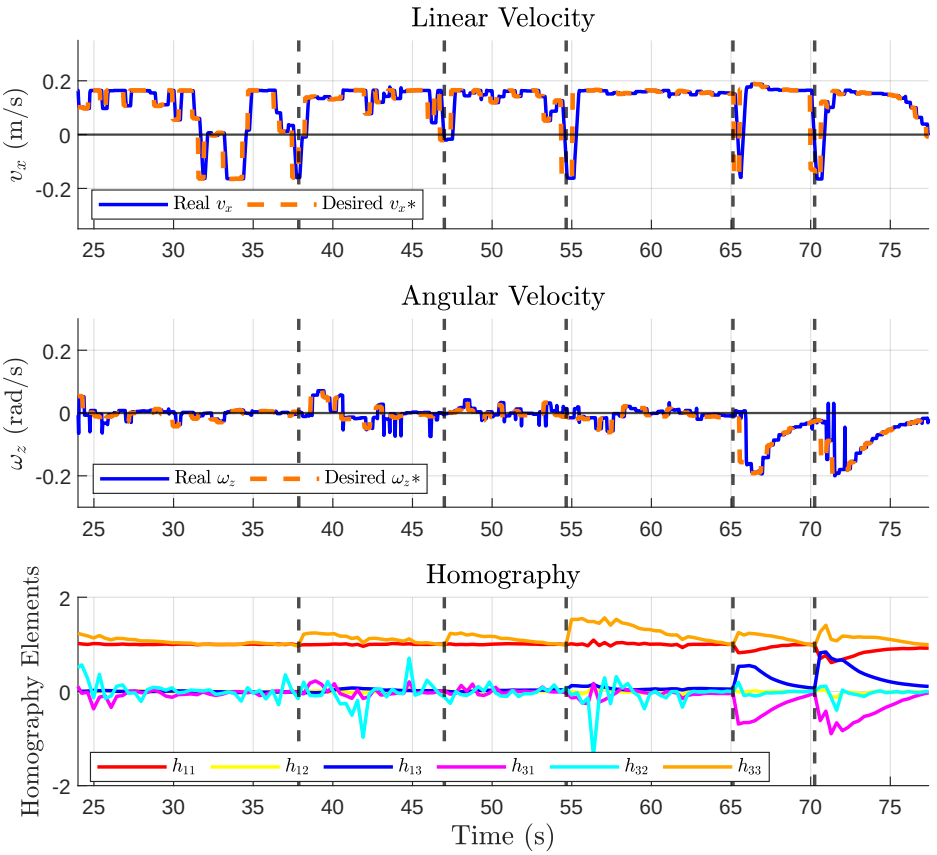


Figure 4.3: Linear velocity along the x-axis, angular velocity around the z-axis, and homography graphs. The Real label in the legend refers to the robot's response within the simulated ROS environment, as measured by odometry when the desired velocity commands are applied. As observed, this response tends to exhibit slight disturbances due to simulated friction between the robot and the ground.

tently works to steer the homography values toward the characteristic elements of an identity matrix, namely 0s and 1s, ensuring that the vehicle maintains a stable and accurate trajectory throughout its operation. This behavior highlights the implemented approach's robustness and capacity to adapt to dynamic environmental conditions while preserving control stability.

4.1.2 NMPC Results

The NMPC simulation was applied to different mobile robots, such as the TurtleBot, to demonstrate its capability of being implemented on various platforms. Although this control provides only two degrees of freedom, its versatility allows it to be adapted to different robots for navigation and control tasks.

Experiments with TurtleBot

The TurtleBot was implemented similarly in the strawberry crop, using NMPC control. This time, a different trajectory was designed based on eight reference images, allowing a more precise navigation adapted to the environment. This strategy optimizes the robot's displacement within the crop, improving trajectory tracking and movement stability.

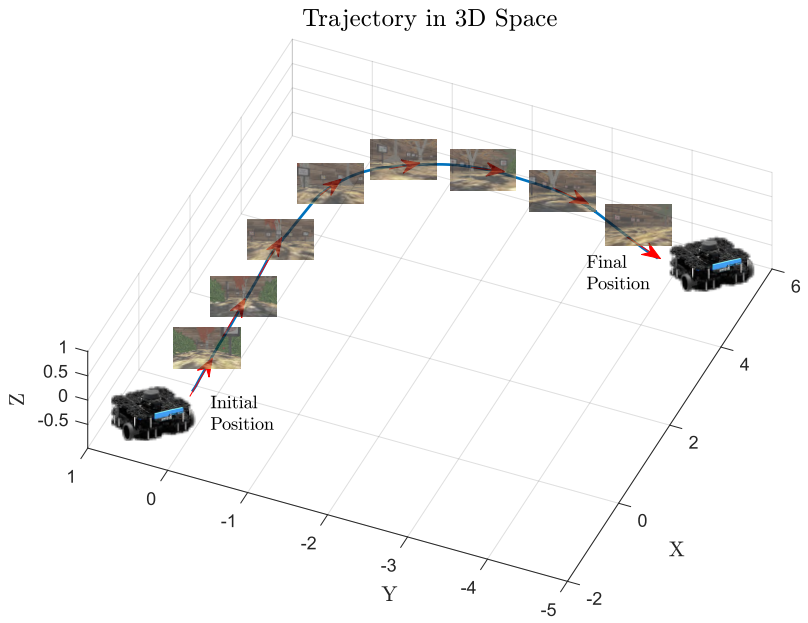


Figure 4.4: 3D trajectory of the robot, depicting its movement through the environment. The path was reconstructed using eight images, capturing key points along its motion.

When analyzing Figure 4.5 and comparing it with the graphs in Figure 4.3, where the basic image-based visual servoing control was applied, significant dif-

ferences in linear velocities can be observed. In Figure 4.3, the linear velocity along the x-axis presents a discontinuity, as the robot tends to decelerate when approaching the desired image. To better adjust its position relative to the reference image, the robot even reverses its direction. In contrast, with the NMPC-based approach, the robot maintains a more consistent velocity during the initial images and only begins to decelerate when reaching the final image. This behavior allows for smoother movement and a more uniform trajectory.

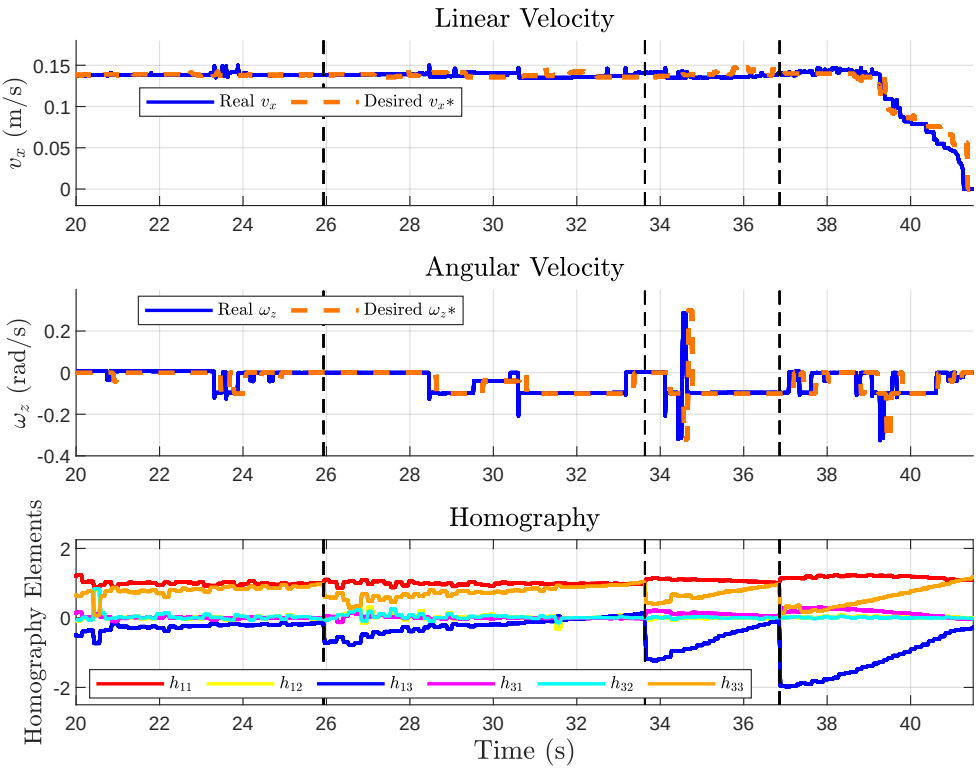


Figure 4.5: Graphs of linear velocity in the x-axis, angular velocity around the z-axis, and homography.The entry Real in the figure legend corresponds to the simulated robot's odometric trajectory, reflecting small perturbations caused by environmental friction modeling. These graphs were obtained from the NMPC.

When comparing the angular velocity in Figure 4.5 with that in Figure 4.3, it is observed that the variations in velocity take on a more stepwise pattern in the former. This behavior results from maintaining a constant linear velocity along the x-axis, making it more efficient to apply significant increments in angular velocity, thereby reducing the need for continuous adjustments in the robot's orientation.

Furthermore, regarding homography, it can be seen that in Figure 4.5, it

does not exhibit the same smoothness as in Figure 4.3. This difference is due to the NMPC controller requiring a greater number of computations to predict the future homography, which impacts processing frequency and, consequently, the continuity of the estimation.

Although the trajectory consisted of eight images, the results presented in Figure 4.5 include only four reference images. This selection was made to improve the clarity and readability of the graphs.

Experiments with TIAGo

Figure 4.6: Simulated image of the TIAGo robot in Gazebo, showing its mobile structure with a wheeled base, a robotic arm and an integrated camera. The simulation includes a virtual environment with realistic lighting and shadows.

To validate the functionality of NMPC for mobile robots, its implementation was extended to a different platform, the TIAGo robot. In addition, the control was tested in a different environment to demonstrate that its applicability is not limited to the strawberry field setting. Instead, it can also be effectively deployed in various scenarios, such as a room or a living space with multiple objects, while maintaining consistent performance.

The TIAGo robot followed a trajectory of four reference images. In this tra-

jectory, the reference images were captured at greater distances than in previous cases, with a maximum distance of 2 meters between them.

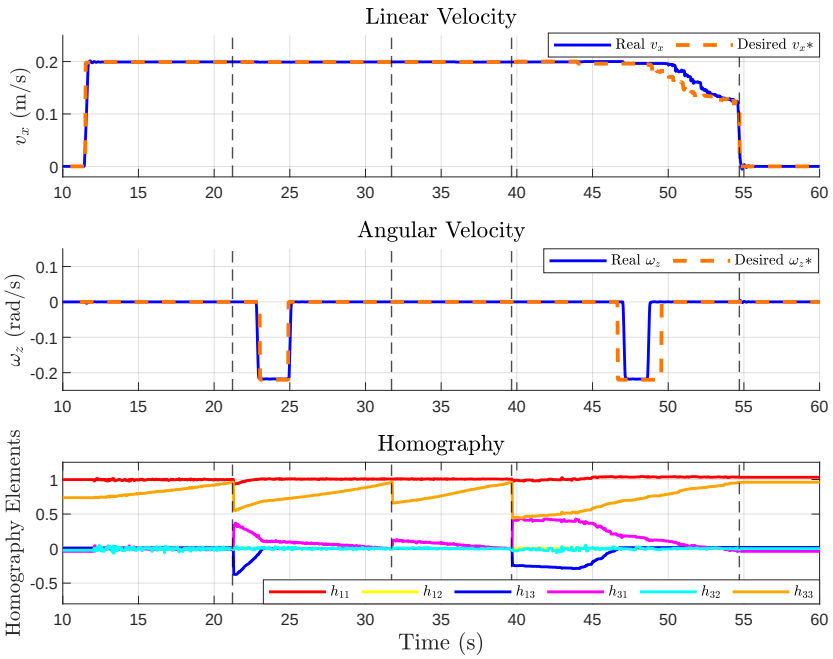


Figure 4.7: Graphs showing the variation of linear velocity along the x-axis, angular velocity around the z-axis, and homography over time. The legend entry Real denotes the robot's response in the Gazebo simulation, based on odometry measurements after receiving the desired velocity commands. As no friction model was included in the Tiago simulation environment, the trajectory exhibits near-perfect behavior with minimal deviations.

Figure 4.7 illustrates that the graphs exhibit less noise, indicating a more stable system response. Additionally, the homography appears smoother and less stepwise. This improvement can be attributed to the fact that the environment in which the control was implemented is computationally less demanding, reducing the processing load on the system and enhancing overall efficiency. Furthermore, in this simulation environment, friction between the wheels and the ground was not modeled, allowing the TIAGo robot to move more smoothly. The absence of frictional forces eliminates irregular velocity variations, contributing to a more continuous and predictable trajectory.

4.2 Implementation of the NMPC in a Real Robot

For the implementation, the data transfer and communication between the mobile robot and the computer were initially evaluated to determine the transmission frequency in Hertz (Hz). An image transmission test was conducted in which the JetBot camera operated at 30 Hz; however, upon reception on the computer, the frequency dropped to 6 Hz. Image compression was implemented to improve transmission efficiency, which helped mitigate the frequency drop. Nevertheless, network instability occasionally affected the transmission speed since communication relied on Wi-Fi.

Figure 4.8: JetBot in different positions with labeled main components.

The implementation of the control system was carried out in a semi-controlled environment within a laboratory, where certain conditions, such as lighting and the robot's movement space, could be regulated. Although some external variables were not fully controlled, this environment provided a suitable setting for initial testing, allowing the evaluation of the system's behavior under relatively stable conditions.

The JetBot camera was used to transmit the point correspondence between the current image and the reference image to determine the reference image in which the robot was located.

Additionally, this process enabled the evaluation of the quality of the point correspondence (see Figure 4.9), as errors in the matching process could lead to an inaccurate homography, introducing noise into the system, and affecting the accuracy of the vehicle's velocity calculations, potentially causing it to follow incorrect trajectories. The results obtained in the implementation were consistent

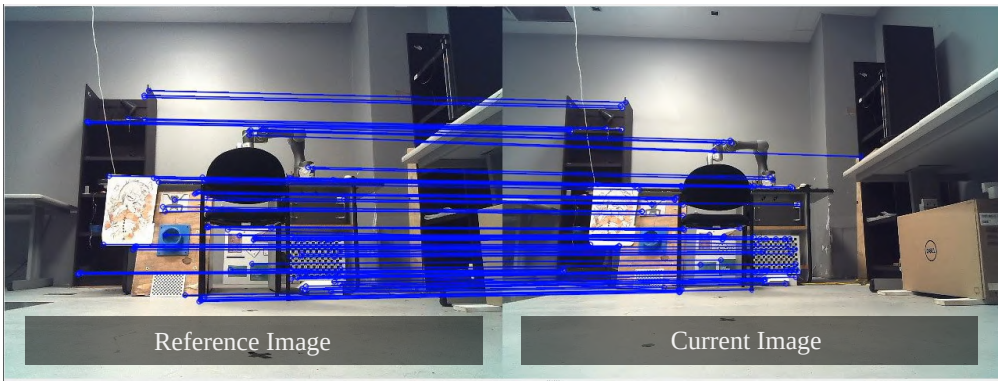


Figure 4.9: Point correspondence between the reference image and the current image. The detected key points in both images are used to compute the homography and determine the robot's position.

with those from the simulations. Specifically, a trajectory composed of three images was completed, yielding graphs similar to those obtained in the simulated experiments, which validates the system's behavior. These results are presented in Figure 4.10.

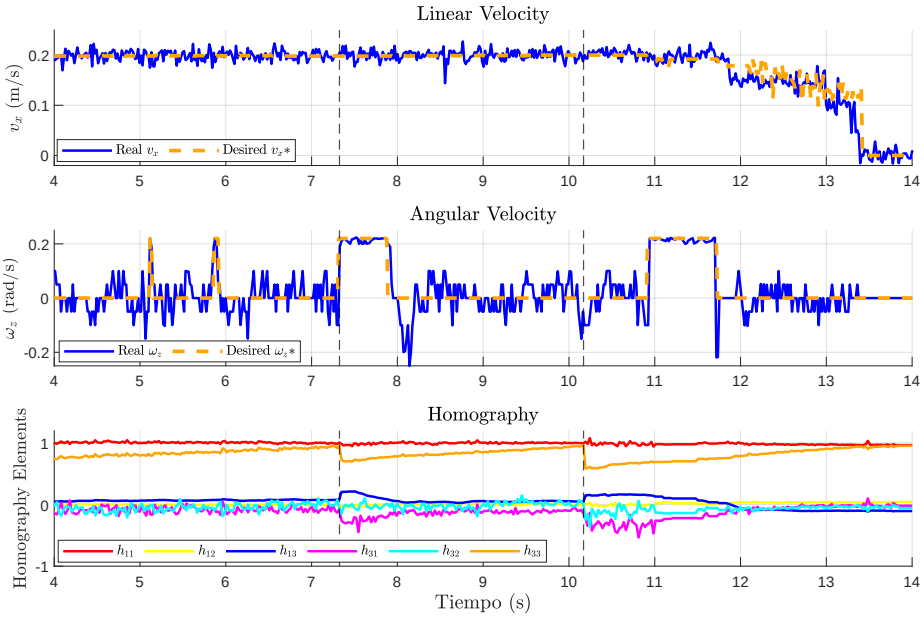


Figure 4.10: Graphs showing the variation of linear velocity, angular velocity, and homography over time, obtained from the JetBot during its operation. The desired speed is the one commanded by the control, and the actual speed is the result shown by the friction of the real world.

The difference observed between the implementation and simulation graphs is that, in the implementation, both linear and angular velocities exhibit higher noise levels. This is primarily due to physical factors in the real environment, such as variations in ground friction, minor surface irregularities, and potential wheel slippage or loss of traction. Unlike in simulation, where terrain conditions are ideal and homogeneous, these variations in the real-world implementation affect the robot's response, causing fluctuations in the estimated velocities and reducing stability during movement.

Chapter 5

Conclusions

This chapter presents the conclusions derived from the experiments performed, highlighting the achievements and challenges identified in implementing the system. In addition, the aspects that influenced the robot's performance are discussed, and future lines of work are suggested to improve its performance.

5.1 Analysis of Control Strategies: IBVS and NMPC

Both control approaches, IBVS and NMPC, successfully followed the proposed trajectory using the pre-mapped images.

In the case of IBVS, the results in terms of velocity were satisfactory, considering that the robot could not move at high speed due to the planned integration of an additional camera for strawberry detection in future work. Consequently, the TurtleBot velocity dynamically adjusted as the reference image changed, ultimately reaching the final image of the trajectory.

Similarly, the NMPC approach achieved the desired trajectory tracking. However, it was observed that the computation of homographies required significantly higher processing power, particularly in the strawberry environment. In contrast,

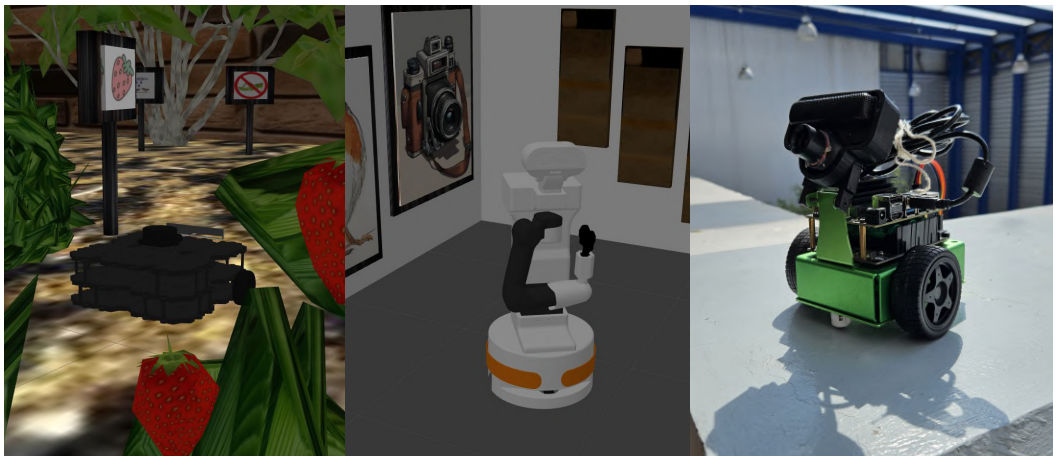


Figure 5.1: Representation of the robots used in the study, including two simulated models and one hardware test.

in the indoor setting where the Tiago robot operated, the computational demand was lower. As a result, NMPC functioned more efficiently in this scenario, producing smoother and more continuous signals.

The primary distinction between both approaches lies in velocity control. NMPC maintained a more consistent speed, allowing it to complete the trajectories faster than IBVS. However, in terms of angular velocity, IBVS proved to be more suitable for executing turns, as NMPC struggled with sharp turns due to its constant velocity, which prevented it from decelerating during rotations.

Despite these differences, NMPC was ultimately the more feasible option due to its smoother and more fluid trajectory execution.

5.2 Comparison between Simulations and Real-World Implementation

The difference between simulation and real-world implementation is evident in the graphs obtained. In simulations, the results tend to be cleaner and exhibit less noise. This is because the controlled environment in which the control is executed is free from external disturbances, leading to more ideal and predictable outcomes.

In contrast, in the actual implementation, the plots show a higher presence of noise due to various physical and environmental factors. Among these factors are ground irregularities, variability in the friction between the tires and the surface, and noise present in the images used for point matching. The latter can be affected by variations in lighting, changes in environmental textures, and possible inaccuracies in image capture and processing.

5.3 Limitations and Challenges Encountered

5.3.1 Vision

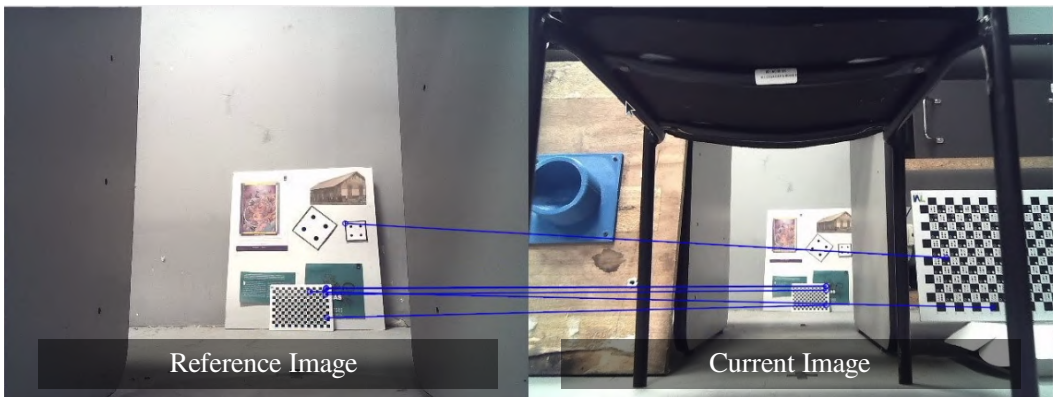


Figure 5.2: Incorrect point correspondences leading to a distorted homography.

One of the main limitations encountered when implementing the work on the JetBot was related to vision. In some images, the presence of certain objects caused noise in the point correspondence. This noise interfered with the robot's expected behavior, making it difficult for it to correctly complete the established trajectory. Accuracy in point correspondence is crucial for trajectory tracking.

5.3.2 Wi-Fi Communication

Another significant challenge was the communication between the laptop and the JetBot, especially in situations where the Wi-Fi connection was affected by interference or signal loss. These fluctuations reduced the transmission speed, preventing the received data from being processed correctly in real time. Addi-

tionally, data loss occurred on some occasions, which impacted the performance and accuracy of the tasks the robot had to perform. Reliable wireless communication is essential for the efficient control of the robot, and fluctuations in the connection quality led to complications during testing.

5.4 Future Research Directions

Based on the results obtained in this study, several areas for improvement and expansion have been identified that could be addressed in future research. These improvements would enhance the robustness and efficiency of the system, optimize navigation based on visual memory, and mitigate the limitations encountered during implementation.

5.4.1 Enhancing the Robustness of Vision-Based Navigation

One of the main challenges encountered in this work was the presence of noise in the point correspondence between the reference images and the images captured by the robot. To improve the robustness of the system, several strategies can be explored:

- Implement **advanced image processing techniques**, such as adaptive filtering, more precise edge detection algorithms, or geometric transformations that optimize key point detection.
- Incorporate **convolutional neural networks (CNNs)** trained to detect and reinforce the most relevant features in images, reducing the impact of irrelevant objects or changes in lighting.
- Apply **more robust point correspondence algorithms**, such as deep learning-based techniques or probabilistic methods that reduce uncertainty in key point identification.
- Develop **sensor fusion methods**, combining image-based vision with additional sensors, such as LiDAR or depth sensors, to improve the accuracy of the robot's position estimation.

5.4.2 Optimization of Wireless Communication

During the JetBot implementation, issues related to Wi-Fi connection stability were observed, affecting data transmission and real-time processing. To mitigate these difficulties, the following improvements can be implemented:

- **Explore alternative communication protocols**, such as ROS 2 with DDS, which offer greater robustness in data transmission and lower latency compared to ROS 1.
- **Optimize data compression** to reduce the required bandwidth for image transmission, minimizing the effects of latency in environments with an unstable connection.
- **Incorporate higher-quality communication hardware**, such as Wi-Fi modules with lower interference or even the use of 5G networks to improve transmission stability and speed.

5.4.3 Implementation in More Complex and Dynamic Environments

The current system was tested in a semi-controlled environment, allowing for an evaluation of its performance under predictable conditions. However, to validate its applicability in more complex real-world scenarios, additional testing is necessary:

- **Evaluate system performance in environments with variations in lighting, textures, and moving obstacles**, where visual memory may be affected.
- **Implement real-time adaptation methods**, where the system can learn and adjust its control parameters based on environmental changes. This could be achieved through online learning or continuous optimization algorithms.

5.4.4 Evaluation of New Control and Navigation Strategies

Although visual memory has proven to be a viable strategy for JetBot navigation, other techniques can be explored to improve system performance:

- **Compare visual memory with SLAM-based approaches (Simultaneous Localization and Mapping)**, utilizing additional sensors to improve the estimation of the robot's trajectory.
- **Investigate the integration of reinforcement learning algorithms**, allowing the robot to autonomously improve its behavior based on experience acquired in the environment.

These future research directions aim to address the limitations identified in this study and expand the system's capabilities for more complex and demanding applications. By integrating improvements in vision, communication, computing, and control, a more robust and adaptable system can be developed, suitable for navigation in dynamic and challenging environments.

References

Asl, H. J., Oriolo, G., & Bolandi, H. (2014). An adaptive scheme for image-based visual servoing of an underactuated uav. *International Journal of Robotics and Automation*, *29*, 92–104.

Bai, Y., Zhang, B., Xu, N., Zhou, J., Shi, J., & Diao, Z. (2023). Vision-based navigation and guidance for agricultural autonomous vehicles and robots: A review. *Computers and Electronics in Agriculture*, *205*.

Boggio, M., Novara, C., & Taragna, M. (2023). Trajectory planning and control for autonomous vehicles: A "fast" data-aided nmpc approach. *European Journal of Control*, *74*.

Chaumette, F. (2020). Visual servoing. *Encyclopedia of Robotics*, 1–9.

Chaumette, F., & Hutchinson, S. (2006). Visual servo control. i. basic approaches. *IEEE Robotics and Automation Magazine*, *13*, 82–90.

Chaumette, F., Hutchinson, S., & Corke, P. (2016). Visual servoing. *Springer Handbooks*, 841–866.

Dallej, T., Gouttefarde, M., Andreff, N., Hervé, P.-E., & Martinet, P. (2019). Modeling and vision-based control of large-dimension cable-driven parallel robots using a multiple-camera setup. *Mechatronics*, *61*, 20–36.

Dani, A. P., & Bhasin, S. (2023). Reinforcement learning for image-based visual servo control. *Proceedings of the IEEE Conference on Decision and Control*, 4358–4363.

Delfin, J., Becerra, H. M., & Arechavaleta, G. (2016). Humanoid localization and navigation using a visual memory. *IEEE Transactions on Intelligent Vehicle*, *8*, 1119–1129.

de Marco, S., Hua, M.-D., Mahony, R., & Hamel, T. (2021). Homography estimation of a moving planar scene from direct point correspondence. *IEEE Transactions on Control Systems Technology*, 1284–1295.

Deng, W., Huang, K., Chen, X., Zhou, Z., Shi, C., & Guo, R. (2020). Semantic rgb-d slam for rescue robot navigation. *IEEE Access*, *8*, 221320–221329.

Developer, N. (2025). Getting started with jetson nano developer kit [Consultado el 3 de marzo de 2025]. https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#intro

Economista, E. (2023). *Principales factores de riesgo en la producción agropecuaria de méxico*. https://www.eleconomista.com.mx/empresas/Principales-factores-de-riesgo-en-la-produccion-agropecuaria-de-Mexico-20231128-0023.html

Feng, Q., Zheng, W., Qiu, Q., Jiang, K., & Guo, R. (2012). Study on strawberry robotic harvesting system. *CSAE 2012 - Proceedings, 2012 IEEE International Conference on Computer Science and Automation Engineering*, *37*, 320–324.

Flores, A., Verdín, R., Moreno, H., & Flores, G. (2023). Development, model, simulation, and real test of a new fully actuated quadrotor. *International Conference on Unmanned Aircraft Systems (ICUAS)*.

George, L., & Mazel, A. (2013). Humanoid robot indoor navigation based on 2d bar codes: Application to the nao robot. *IEEE-RAS International Conference on Humanoid Robots*, 329–335.

Ghrairi, K., Lamine, H., Bennour, S., & Chaker, A. (2024). Camera-based control system of a planar cable-driven parallel robot intended for functional rehabilitation. *Robotica*.

Guzmán, E. I. V., Murillo, N. G. A., & Colunga, G. R. F. (2024). Implementación de control visual para planificación de trayectorias en un cultivo de fresas virtual. *Congreso Internacional de Electrónica y Computo Aplicado*, *33*.

Haviland, J., Dayoub, F., & Corke, P. (2020). Control of the final-phase of closed-loop visual grasping using image-based visual servoing.

He, G., Jangir, Y., Geng, J., Mousaei, M., Bai, D., & Scherer, S. (2023). Image-based visual servo control for aerial manipulation using a fully-actuated uav. *IEEE International Conference on Intelligent Robots and Systems*, 5042–5049.

Jiménez, L. A. M., Mejía, G., de Oca Rebolledo, A. M., & Flores, G. (2023). Robot autónomo para la detección y clasificación de fresas en campos de

cultivo mediante deep learning. *TIES, Revista de Tecnología e Innovación en Educación Superior*, 1–8.

Kumar, Y., Roy, S. B., & Sujit, P. B. (2023). Dynamic image-based visual servoing for quadrotor to track a planar target with unknown motion. *Proceedings of the IEEE Conference on Decision and Control*, 1398–1403.

Lai, J., Ren, T.-A., Yue, W., Su, S., Chan, J. Y. K., & Ren, H. (2023). Sim-to-real transfer of soft robotic navigation strategies that learns from the virtual eye-in-hand vision. *IEEE Transactions on Industrial Informatics*, *8*, 2365–2377.

Li, Z., Lai, B., & Pan, Y. (2024). Image-based composite learning robot visual servoing with an uncalibrated eye-to-hand camera. *Transactions on Mechatronics*, 2499–2509.

Li, Z., Fang, F., Wang, T., & Zhang, G. (2024). Homography estimation with adaptive query transformer and gated interaction module. *IEEE Transactions on Circuits and Systems for Video Technology*.

Liu, S., Lu, Y., Jiang, H., Ye, N., Wang, C., & Zeng, B. (2023). Unsupervised global and local homography estimation with motion basis learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7885–7899.

Mejía, G., de Oca Rebolledo, A. M., & Flores, G. (2023). Strawberry localization in a ridge planting with an autonomous rover. *Engineering Applications of Artificial Intelligence*, *119*.

Mohebbi, A., Keshmiri, M., & Xie, W.-F. (2014). An acceleration command approach to robotic stereo image-based visual servoing. *IFAC Proceedings Volumes*, *47*, 7239–7245.

Murillo, N. G. A. (2021, March). *Visually-guided humanoid walking pattern generation* [Doctoral dissertation, Centro de Investigación en Matemáticas A.C.].

Pire, T., Mujica, M., Civera, J., & Kofman, E. (2019). The rosario dataset: Multisensor data for localization and mapping in agricultural environments. *https://doi.org/10.1177/0278364919841437*, *38*, 633–641.

Ramírez, G., Verdín, R., & Flores, G. (2024). Real-time mapping for teleoperation systems in vr of unmanned aerial vehicles. *International Conference on Unmanned Aircraft Systems, ICUAS 2024*, 1002–1009.

Robinson, N., Tidd, B., Campbell, D., Kulić, D., & Corke, P. (2023). Robotic vision for human-robot interaction and collaboration: A survey and systematic review. *ACM Transactions on Human-Robot Interaction*, *12*, 12.

Roque, P., Bin, E., Miraldo, P., & Dimarogonas, D. V. (2020). Fast model predictive image-based visual servoing for quadrotors. *IEEE International Conference on Intelligent Robots and Systems*, 7566–7572.

SADER. (2023). Hablemos de la agricultura en méxico (parte 1). *Secretaría de Agricultura y Desarrollo Rural*. https://www.gob.mx/agricultura/articulos/hablemos-de-la-agricultura-en-mexico-parte-1

Sarabandi, S., Porta, J. M., & Thomas, F. (2022). Hand-eye calibration made easy through a closed-form two-stage method. *Robotics and Automation Letters*, 3679–3686.

Saviolo, A., Rao, P., Radhakrishnan, V., Xiao, J., & Loianno, G. (2024). Unifying foundation models with quadrotor control for visual tracking beyond object categories. *International Conference on Robotics and Automation (ICRA)*.

SECIHTI. (2025). Soberanía alimentaria [Accessed: 2025-03-4]. https://secihti.mx/pronaces/pronaces-soberania-alimentaria/

Shi, L., Li, B., & Shi, W. (2024). Vision-based uav adaptive tracking control for moving targets with velocity observation. *Transactions of the Institute of Measurement and Control*, *46*, 2255–2265.

Sun, G.-J., & Lin, H.-Y. (2020). Robotic grasping using semantic segmentation and primitive geometric model based 3d pose estimation. *2020 IEEE/SICE International Symposium on System Integration (SII)*.

Tahri, O., & Chaumette, F. (2005). Point-based and region-based image moments for visual servoing of planar objects. *IEEE Transactions on Robotics*, 1116–1127.

Takaya, K., Asai, T., Kroumov, V., & Smarandache, F. (2016). Simulation environment for mobile robots testing using ros and gazebo. *International Conference on System Theory, Control and Computing (ICSTCC)*.

Verdín, R., Ramírez, G., Rivera, C., & Flores, G. (2021). Teleoperated aerial manipulator and its avatar. communication, system's interconnection, and virtual world. *International Conference on Unmanned Aircraft Systems (ICUAS)*.

Wang, S., Zhai, G., Li, Y., & Jiang, J. (2025). Homography-based displacement correction of uav using four laser points and template method. *Measurement*.

Wu, T., Zhu, Y., Zhang, L., Yang, J., & Ding, Y. (2023). Unified terrestrial/aerial motion planning for hytaqs via nmpc. *IEEE Robotics and Automation Letters*, *8*, 1085–1092.

Xiong, Y., Ge, Y., Grimstad, L., & From, P. J. (2020). An autonomous strawberry-harvesting robot: Design, development, integration, and field evaluation. *Journal of Field Robotics*, *37*, 202–224.

Zheng, C., Abd-Elrahman, A., & Whitaker, V. (2021). Robotic grasping using semantic segmentation and primitive geometric model based 3d pose estimation. *Remote Sensing 2021*, *13*.

Zhong, H., Wang, Y., Miao, Z., Li, L., Fan, S., & Zhang, H. (2022). A homography-based visual servo control approach for an underactuated unmanned aerial vehicle in gps-denied environments. *IEEE Transactions on Intelligent Vehicle*, *8*, 1119–1129.

Zhu, N., Xie, W.-F., & Shen, H. (2024). Position-based visual servoing of a 6-rss parallel robot using adaptive sliding mode control. *ISA Transactions*, *144*, 398–408.

Appendix A: Jetson Nano Specifications

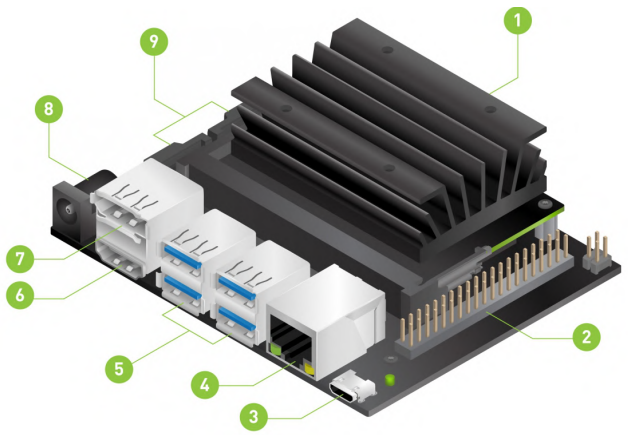


Figure A.1: Jetson Nano Parts (Developer, 2025)

1. microSD card slot for main storage
2. 40-pin expansion header
3. Micro-USB port for 5V power input, or for Device Mode
4. Gigabit Ethernet port
5. USB 3.0 ports (x4)
6. HDMI output port
7. DisplayPort connector
8. DC Barrel jack for 5V power input
9. MIPI CSI-2 camera connectors

A.1 Features of the Jetson Nano

The following table shows the characteristics of the Jetson:

Feature	**Specification**
Processor	Quad-core ARM Cortex-A57 @ 1.43 GHz
GPU	NVIDIA Maxwell with 128 CUDA cores
RAM	4 GB LPDDR4
Storage	microSD (minimum recommended: 32 GB)
Interfaces	4x USB 3.0, HDMI, DisplayPort, GPIO, I2C, I2S, SPI
Connectivity	Gigabit Ethernet (WiFi adapter optional)
Power Supply	5V/4A (high-performance mode) or 5V/2A (basic mode)
Operating System	Ubuntu 18.04 with NVIDIA JetPack (includes CUDA, cuDNN, TensorRT, OpenCV, ROS, etc.)
Dimensions	100 mm × 80 mm
Approximate Price	$99 USD (without additional accessories)

Table A.1: Jetson Nano Technical Specifications

A.2 Advantages of Jetson Nano

- **High AI Performance:** Its GPU with 128 CUDA cores allows real-time execution of deep learning and AI models.
- **Energy Efficiency:** Low power consumption (5-10 W), ideal for embedded applications.
- **ROS Compatibility:** Works with ROS Melodic and Noetic, facilitating robotics applications.
- **Support for Multiple Frameworks:** Compatible with TensorFlow, PyTorch, OpenCV, cuDNN, and TensorRT.
- **Good Connectivity:** Features multiple USB 3.0 ports, GPIOs, and Gigabit Ethernet.
- **Strong Development Ecosystem:** NVIDIA provides JetPack, an optimized suite for AI and computer vision applications.

A.3 Disadvantages of Jetson Nano

- **Limited Computing Power:** Although powerful for its size, it does not match high-end GPUs like Jetson Xavier or dedicated desktop GPUs.
- **No Built-in WiFi:** Requires an additional USB WiFi adapter.
- **Limited Storage:** Uses microSD, which may slow down performance for intensive applications.
- **CUDA Dependency:** Applications not optimized for CUDA may not fully utilize its potential.
- **Software Compatibility Issues:** Some software packages may not support the latest versions due to OS limitations.