POLITECNICO DI TORINO

Master's Degree in Mechatronics Engineering

Master's Degree Thesis

Non-linear Model Predictive Control for GPS-free Autonomous Navigation in Vineyards

Supervisors:

Prof. Marcello CHIABERGE

Doct. Marco AMBROSIO

PhD. Mauro MARTINI

Doct. Alessandro NAVONE

Doct. Andrea OSTUNI

Candidate:

Matteo SPERTI

December 2023

Summary

Precision agriculture has made significant progress in recent years by utilizing technology to optimize crop production, enhance farming efficiency, and automate harvesting processes. Autonomous navigation is a critical component for ground rovers in the agricultural field. This thesis focuses on developing an advanced autonomous navigation system for a rover operating within row-based crops. A position-agnostic system is proposed to address the challenging situation when standard localization methods, like GPS, fail due to unfavorable weather or obstructed line-of-sight. This breakthrough is especially vital in densely vegetated regions, including areas covered by thick tree canopies or pergola vineyards.

The primary objective of the control system is to navigate through entire rows, effectively avoiding obstacles in its path. To ensure versatility across crop types with different row spacing, the rover is designed to operate within the entire inter-row area for crops with small row spacing or predefined lanes for crops with larger ones. The navigation system utilizes a vision-based approach, relying on an RGB-D camera for real-time video streaming analysis to detect and identify row spaces and obstacles. Then, a NMPC (Non-linear Model Predictive Control) strategy is used to compute trajectory and control sequence. The proposed navigation system is implemented in Python and runs into a ROS2 (Robot Operating System 2) dedicated subsystem. Moreover, the strategy proposed can also be employed in navigation with similar constraints, i.e., a long straight path between two "walls", such as in passages, galleries, etc.

A distinctive feature of this system is its ability to recognize and approach objects of interest, such as fruit boxes. Upon identifying a target, the system adjusts its navigation to approach the target object and then resumes its row traversal until it reaches the end of the row. However, the primary scope of this work is the navigation system, so basic image segmentation techniques are employed as a demonstration to identify the targets and validate the approaching and recovery maneuvers.

Extensive experimentation is conducted on simulated and real vineyards to demonstrate the competitive advantages of the proposed solution. The controller has exhibited robustness in handling heterogeneity in crop density, height, and other environmental factors. Moreover, it successfully navigates through pergola vineyards and maintains functionality on rough terrains. This research contributes to the ongoing efforts to advance precision agriculture and autonomous navigation in row-based crop environments.

Acknowledgements

I wish to my heartfelt gratitude to all those who have been with me, not just during the completion of this Master's Thesis but throughout my entire five-year journey (Bachelor and Master) at Politecnico di Torino.

Under the supervision of Professor Marcello Chiaberge at the Politecnico di Torino Interdepartmental Centre for Service Robotics, PIC4SeR[1], this thesis came to fruition. I am sincerely grateful to Professor Chiaberge for his guidance during these months. Special thanks are extended to Marco Ambrosio, Mauro Martini, Alessandro Navone, and Andrea Ostuni for their invaluable contributions to this thesis. The collaborative and supportive atmosphere fostered by the entire team at PIC4SeR deserves acknowledgment.

A profound debt of gratitude is owed to my family, whose unwavering support has been a cornerstone throughout my years of study. The encouragement of my parents, my brother Stefano, my grandparents, and all my family empowered me to surmount the challenges and difficulties inherent in this academic journey.

Last but certainly not least, I extend my appreciation to all my friends who have been companions on this academic odyssey, particularly those who shared this degree course with me.

This Master's Thesis represents the culmination of a challenging and rewarding academic journey, and I am deeply thankful to all those who have been a part of it.

Matteo

[1] `www.pic4ser.polito.it`

Table of Contents

List of Tables

List of Figures

Acronyms

BT Behavior Tree. xii, 67, 97–99

CNN Convolutional Neural Network. 7, 8, 10, 63

CV Computer Vision. 2, 3, 9, 11, 12, 17, 28, 29, 35, 85

DDS Data Distribution Service. 93, 94

DGPS Differential Global Positioning System. 4

DRL Deep Reinforcement Learning. 5, 7, 10

EKF Extended Kalman Filter. 49

FOV Field of View. 5, 8, 9, 16, 39, 57, 59, 67, 73

FSM Finite State Machine. 97

GLONASS Global Navigation Satellite System. 4

GNSS Global Navigation Satellite System. 3–5, 10, 70

GPS Global Positioning System. xii, 4, 5, 55, 70, 77–79

IMU Inertial Measurement Unit. 5, 55, 70, 77

IPOPT Interior Point OPTimizer. 91, 92

LIDAR Laser Imaging Detection and Ranging. 39, 57, 70, 77

LQ Linear-Quadratic. 51, 52

MAE Mean Absolute Error. 74, 76, 81

MIMO Multiple-Input Multiple-Output. 52, 53

MPC Model Predictive Control. xi, 2, 41, 48, 50, 52–55

MPPI Model Predictive Path Integral. 41, 54–56

MSE Mean Squared Error. 74, 76, 81

NCC Normalized Cross Correlation. 5, 24, 25

NMPC Non-linear Model Predictive Control. xi, xii, 9, 10, 52–54, 58–60, 63–65, 83

NURBS Non-Uniform Rational Spline. 9

PCD Point Cloud Data. xi, xii, 9, 10, 35–39, 57, 59, 60, 63, 66, 70, 76, 77, 80, 81

RF Reference Frame. xi, xii, 10, 12, 14, 15, 33, 35–37, 48, 59, 62, 63, 66, 67, 89, 96, 97

RH Receding Horizon. 51–54

ROI Region of Interest. 7, 26

ROS2 Robot Operating System 2. 41, 55, 59, 66, 67, 70, 81, 93–98, 100

RTK Real-Time Kinematics. 4, 10, 70

SAD Sum of the Absolute Differences. 24

SLAM Simultaneous Localization And Mapping. 49, 77

SSD Sum of the Squared Differences. 24

ToF Time-of-flight. 39

UDP User Datagram Protocol. 93

UGV Unmanned Ground Vehicle. ix, 4, 5, 69–72, 77

UTM Universal Transverse Mercator. 78

VO Visual Odometry. 4, 5, 10

ZNCC Zero-offset Normalized Cross Correlations. 24, 25

ZSAD Zero-offset Sum of the Absolute Differences. 24

ZSSD Zero-offset Sum of the Squared Differences. 24, 25

Chapter 1

Introduction

In recent years, precision agriculture has made significant strides in harnessing technology to optimize crop production, enhance the efficiency of farming operations, and reduce agricultural waste [1]. Modern agricultural systems are now expected to not only gather vital data from the environment but also to make informed decisions based on this information and execute actions with precision and impeccable timing.

Particularly, row-based crops represent a pivotal focus in precision agriculture. They constitute more than 75% of all planted acres of cropland across the United States [2]. Research in this domain encompasses various aspects, such as crop localization [3], path planning [4], navigation, monitoring, harvesting [5], spraying, and vegetative assessment.

This project was born at the PIC4SeR (PoliTo Interdepartmental Centre for Service Robotics)[1] in a wider context that focuses on service robotics for the development of highly innovative solutions in several fields such as precision agriculture, smart cities, well-being, cultural heritage, and space applications. The development of a novel controller for row-based crops aligns seamlessly with the center's research endeavors in the field of agriculture.

Objective of the project

This thesis is aimed to develop a robust navigation system for a rover operating within row-based crops. A particularly challenging situation in this context arises when standard localization methods, such as GPS, fail to achieve the desired precision, often due to unfavorable weather conditions or line-of-sight obstructions. This situation is particularly evident in densely vegetated areas, such as those covered by dense tree canopies.

To address these challenges and adapt to several environmental conditions, a position-agnostic system is proposed. This approach not only avoids the issues related to integration and localization systems but also excels in scenarios where traditional methods fall short.

[1] `www.pic4ser.polito.it`

The primary objective of the control system is to navigate through entire rows, effectively avoiding obstacles in its path. Furthermore, to enhance the versatility of this solution across several types of crops with varying row spacing, the rover is designed to operate within inter-row spaces for crops with small row spacing or within predefined lanes, such as the right half of the row space, for crops with larger row spacing, such as zucchinis.

In addition to these challenges, the controller must exhibit robustness in the face of heterogeneity in crop density, variations in crop height, and other environmental factors. It should seamlessly handle situations involving very high canopies or pergolas and maintain its functionality on rough terrains.

A distinctive feature of this system is its ability to recognize and approach objects of interest, such as fruit boxes. Upon identifying a target, the system adjusts its navigation to approach the target object, then resumes its row traversal until it reaches the end of the row. It is important to note that the scope of this work does not encompass the comprehensive development of target recognition mechanisms. Instead, a basic color filter is employed as a demonstration to identify the target and validate the approaching and recovery maneuvers.

Organization of this work

In this chapter, we have introduced the context and goals of this thesis while summarizing the adopted methodology. The subsequent chapters are organized as follows:

- *Chapter 2* provides an extensive review of the state of the art in navigation systems for agricultural contexts. This chapter discusses the advantages and challenges of various solutions proposed in the field.
- *Chapter 3* serves as a dedicated introduction to the realm of Computer Vision tools, setting the stage for their pivotal role in this project.
- *Chapter 4* focuses on mobile robots, particularly emphasizing kinematics models and the Model Predictive Control (MPC) controller.
- *Chapter 5* offers a detailed presentation of the proposed controller designed in this thesis project to achieve the desired goals.
- *Chapter 6* describes the robot platform employed in this work and outlines the evaluation metrics for the project. It includes the presentation and analysis of tests conducted in both simulated and real-world scenarios.
- *Chapter 7* concludes the thesis, summarizing the findings and discussing potential future developments in the field.
- The *Appendices* contains additional insights on several topics cited throughout the thesis, providing supplementary information to enhance the reader's understanding, such as homogeneous coordinates, quaternions, etc.

Chapter 2

State of the art in navigation system for agriculture

This thesis project focuses on developing a navigation system for precision agriculture. In this context, the straightforward solution involves utilizing Global Navigation Satellite System (GNSS) signals to localize agricultural rovers in the field and navigate them using standard procedures. However, in densely vegetated areas, such as those covered by dense tree canopies, GNSS fails to achieve the desired precision due to obstructed line-of-sight. To address this limitation, various methods have been proposed in the literature. In this chapter, I will present some of these approaches.

There are two main strategies:

- Refining odometry and GNSS positioning with other technologies, such as Computer Vision (CV), where cameras are employed to capture and interpret visual information from the environment. This information can then be used to refine the rover's localization.
- An alternative approach involves developing navigation systems that are not reliant on precise localization. Instead of depending on an absolute position, these systems operate based on relative positioning or other environmental cues.

Each of these approaches has its advantages and challenges, and the choice between them depends on the specific requirements of the precision agriculture application and the characteristics of the environment in which the rover operates.

2.1 Localized navigation systems

In the pursuit of successful navigation within agricultural environments, a primary strategy involves harnessing various technologies to enhance the localization precision of rovers traversing expansive fields. One such technology is the Global Navigation Satellite System (GNSS), bolstered by all possible improvements and corrections, proving effective in open fields with smaller crops. Another avenue explores the integration of visual perception

through approaches like Visual Odometry (VO). The objective remains the augmentation of rover position precision, thereby ensuring reliable movement feedback for effective UGV motion control.

2.1.1 Global Navigation Satellite System

Global Navigation Satellite System (GNSS) refers to navigation systems providing users with a three-dimensional positioning solution through passive ranging using radio signals from orbiting satellites [6]. Each satellite continuously transmits data indicating its location and current time, and receivers use this information to determine their positions. The timing of satellite transmissions is crucial, with nanosecond-level accuracy required. Ground stations regularly update and synchronize satellite transmissions, which, along with on-board atomic clocks, ensures precise timing. While triangulation theoretically requires only three satellites, GNSS receivers utilize four (minimum) to account for three position axes and a time correction [7].

Various GNSS systems are currently in use worldwide, including the American *Global Positioning System (GPS)*, Russian *Global Navigation Satellite System (GLONASS)*, European *Galileo*, and Chinese *Compass*. Factors contributing to GNSS error include satellite position and number, receiver clock timing, ionospheric and atmospheric delays, and multipath effects. In mobile robot applications, GPS latency, typically around 200 to 300 milliseconds, may limit updates to approximately 5 Hz. Fast-moving robots might need local motion integration for proper control due to these latency limitations.

GPS applications typically operate with a resolution of around 15 meters. Differential Global Positioning System (DGPS) improves this by utilizing a second static receiver at a known position, correcting errors, and achieving resolutions of approximately 1 meter. Incorporating carrier signal phase information further enhances precision, enabling resolutions of 1 cm for point positions. For mobile robots, the DGPS technique requires a stationary reference unit within kilometers' range of the robot. Real-Time Kinematics (RTK) correction is the real-time version of the DGPS technique. Multi-GNSS receivers, considering signals from different navigation systems, enhance accuracy and performance.

In precision agriculture, multi-GNSS receivers demonstrated superior accuracy, particularly in challenging conditions like orchards and mountainous areas, offering precise positioning for applications such as yield monitoring and variable rate applications. However, the increased cost of multi-GNSS may be justified for users requiring precise farm operations and research in conditions with poor visibility.

2.1.2 Visual odometry systems

Visual Odometry (VO) aim is to measure the pose of a system using the information provided by a set of successive images. So, it can provide reliable movement feedback in UGV motion control [8]. The idea is to evaluate the relative movement of a solid camera having occurred during a time interval $t_k - t_{k+1}$, comparing the image pair I_k and I_{k+1}, acquired in the ordered time instants t_k and t_{k+1}, respectively.

The available image processing algorithms for VO applications have two main approaches:

- **Feature-based algorithms**: detect and track specific features or details in successive images.
- **Appearance-based algorithms**: examine changes in appearance across successive frames by extracting information regarding pixel displacement using the template matching process (presented in Sec. 3.2.3).

To enhance accuracy and reliability, VO systems are often integrated with sensors such as Inertial Measurement Unit (IMU), GPS technology, or wheel and track encoders. This integration helps mitigate error accumulation during long missions.

Let us consider the VO system proposed by [8] and represented schematically in Fig. 2.1: it is an appearance-based algorithm that exploits Normalized Cross Correlation (NCC) as a cost function for template matching.

The relative movement of I_{k+1} with respect to I_k, in terms of translation $[\hat{u}, \hat{v}]^T$ $[pixels]$ and rotation $\hat{\theta}$ $[deg]$, is performed by assessing the position of the templates T_k in the subsequent image I_{k+1}. Then, the relative movement of the camera is estimated by knowing the position of the template T_k in I_k.

However, the accuracy of this approach is limited by the digital discretization of the FOV performed by the digital camera, i.e., it is related to the adopted image resolution. Increasing the image resolution, while increasing the accuracy, increase also the required computed load, which does not fit with the real-time requirement of some VO applications or requires too expensive technologies.

To address these challenges, [8] proposed an enhanced VO algorithm. Translation and rotation are computed as the weighted centroids of a neighborhood around the maximum of the discrete cross-correlation function. This approach allows for more accurate UGV movement evaluation with continuous values, significantly enhancing precision.

Visual Odometry remains a pivotal component in UGV motion control, providing valuable insights into the vehicle's relative movement and aiding in navigation and control strategies.

2.2 Position-agnostic navigation systems

In certain agricultural environments characterized by dense canopies and abundant vegetation, the reliability of Global Navigation Satellite System (GNSS) sensors diminishes, particularly in seasons of heightened foliage such as spring and summer. This scenario strengthens the need for alternatives to reduce the cost of the system without affecting its robustness. Visual Odometry (VO), as detailed in Section 2.1.2, emerges as a viable solution for field navigation challenges. However, its applicability encounters limitations, especially in prolonged outdoor trajectories with repetitive visual patterns, typical of extensive row crop fields.

To overcome the precision localization challenge, position-agnostic vision-based navigation algorithms have been extensively explored. The fundamental concept involves real-time analysis of data from a camera to generate velocity commands without explicit knowledge of the current field position. Diverse methodologies have been proposed, ranging from the utilization of a Deep Reinforcement Learning (DRL) agent for action determination to the

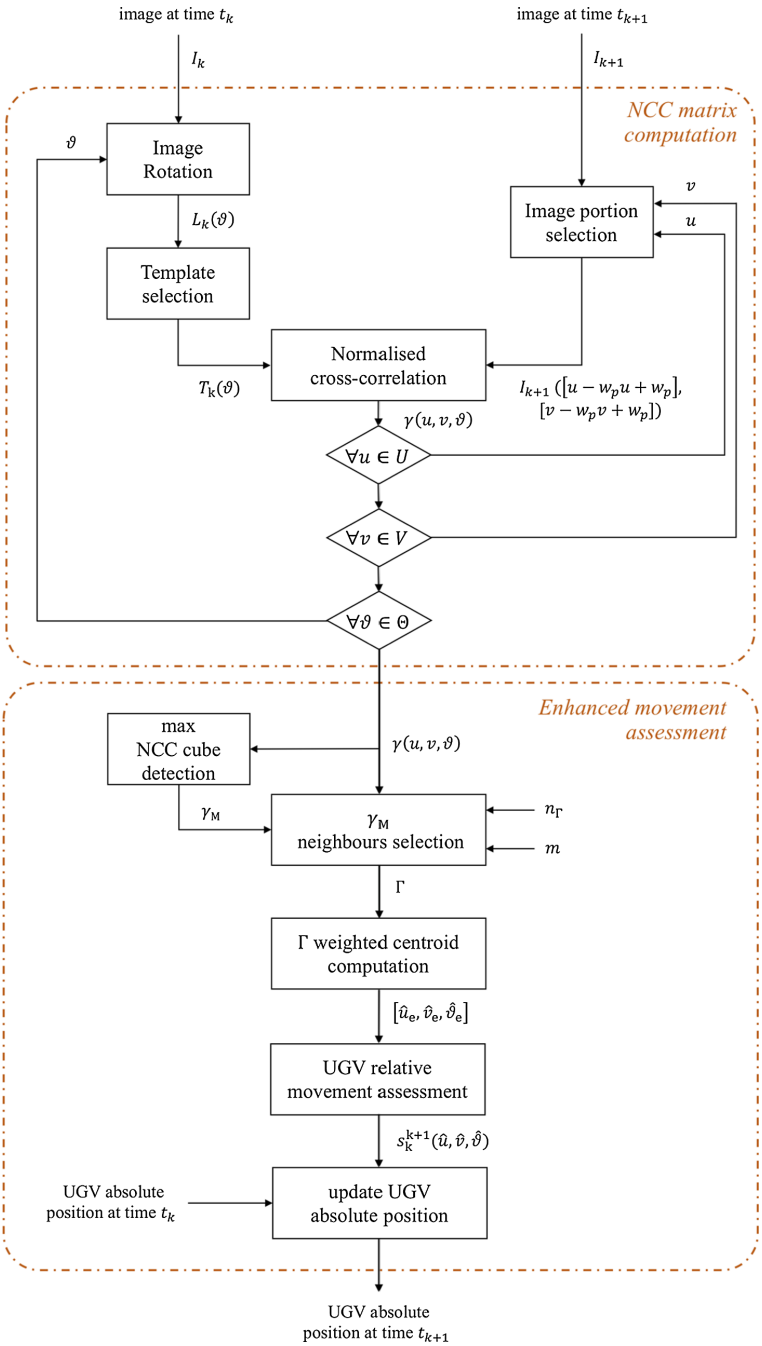


Figure 2.1: Scheme of the visual odometry pipeline presented in [8].

integration of a segmentation model and a proportional controller aimed at aligning the robot with the center of the row. These innovative approaches strive to enhance navigation adaptability in complex agricultural landscapes.

2.2.1 Deep Reinforcement Learning approach

Traditional navigation algorithms conventionally compartmentalize the process into distinct stages for perception, planning, and control, potentially accumulating errors across sub-modules. In contrast, policy learning methods offer a paradigm shift by directly mapping raw input data to actions, fostering the creation of a *sensorimotor* agent through the integration of vision and control systems. This paradigm shift streamlines the entire navigation algorithmic pipeline [9].

Of particular note is model-free Deep Reinforcement Learning (DRL), which optimizes a parametric policy without requiring explicit knowledge of the environment's dynamic model. This flexibility empowers agents to navigate in previously unseen environments, like curved rows. The model's position-agnostic nature allows the robot to traverse the end of the row without the need for global localization.

However, applying DRL to agricultural settings, such as vineyards, introduces specific challenges. The constrained geometry of vineyards may prematurely terminate episodes due to collisions, limiting the diversity of encountered states during policy training. To address this, [9] introduced a technique involving the continuous alteration of the robot's starting point during training. Additionally, given the absence of information about the robot's position, key details such as distance or heading with respect to the goal are not included in the input state.

2.2.2 Segmentation-based controllers

An alternative approach leverages semantic segmentation to analyze the input images and a simpler controller to compute the corresponding velocity commands. For instance, a custom Deep Convolutional Neural Network (CNN) can be employed for segmenting specific Region of Interest in the input image, such as the crops delimiting a row. Notably, both [10] and [11] propose a custom CNN based on MobileNetV3 [12] as the backbone of their architectures. The segmentation mask is then analyzed to find the free space between the crops, and a straightforward proportional controller computes the velocity commands. The pipeline schema of this approach, as presented in [11], is illustrated in Fig. 2.2.

To determine the coordinate of the center of the free space from the segmented mask, [11] proposes summing the values in the columns and searching for clusters of zero, indicating areas where no crops are present. However, this solution is limited to scenarios where physical space exists between the left and right crop boundaries. In contrast, for environments like pergola vineyards, where no space exists in all the cells of a range of columns (due to the presence of the pergola in the upper part of the image), this method fails. To address this limitation, [10] introduces a novel approach that selects the desired central pixel as the minimum of the histogram of columns in the segmented mask. This modification ensures robust performance even when columns are not empty.

A similar strategy has been proposed by [13]. However, to determine the set point for adjusting the vehicle's heading, this research opts to identify the sky region between the crops. The rationale behind this choice lies in the consistent nature of the sky, which remains relatively unchanged throughout seasons in contrast to the varying appearance of

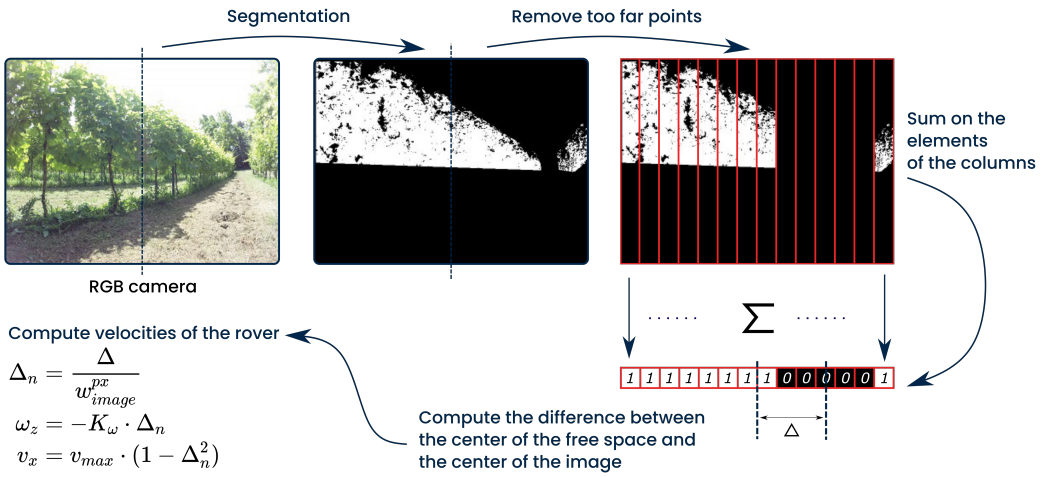


Figure 2.2: Pipeline of the vision-based controller presented in [11].

crops. To achieve sky segmentation, the research uses the green channel of the image and employs a thresholding process, computationally less demanding than the CNN approach utilized by [11, 10]. Once the sky is segmented, also in this study, a basic PID controller is employed to align the rover's heading with the desired set point.

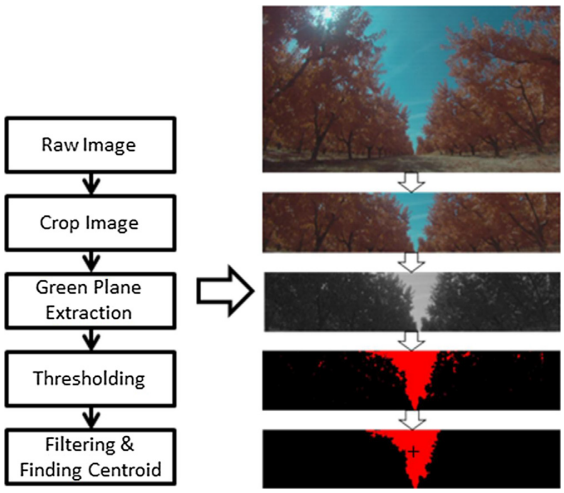


Figure 2.3: Image processing algorithm for autonomous navigation proposed by [13].

[14] instead decides to train a Deep Convolutional Neural Network to directly output one of a discrete set of moves for the rover. The rover's actions are determined as follows:

1. It moves forward only if both sides of tree rows are visible in the camera FOV.
2. It turns left or right if and only if one side of a row can be perceived.

3. It stops driving and prepares for headland turning upon detecting the last row.

To facilitate network training, the researchers propose a sample collection method that enables the robot to autonomously drive and gather data without human intervention or remote control, eliminating the need for manual labeling of training samples. Techniques such as batch normalization, dropout, data augmentation, and 10-fold cross-validation are employed to enhance the network's generalization capabilities.

On the contrary, [15] proposed utilizing the Hough Transform to extract the boundary lines between trees and the terrain. The Hough Transform is a robust feature extraction technique that identifies straight lines in an image through a transformation between the Cartesian space and a parameter space. One key advantage of the Hough Transform is its ability to detect straight lines even when pixels are not perfectly aligned, making it effective in handling breaks caused by noises. In this approach, the Hough Transform is applied to identify the left and right borders of the trees, and the desired reference path is then computed as the middle line between these borders.

To enhance the clarity of the input image for the Hough Transform, the input frame undergoes a segmentation process using a graph partitioning classification. This segmentation classifies the frame into three categories: terrain, trees, and sky. Subsequently, the terrain class is selected, filtered, and provided as input to the Hough Transform for accurate extraction of the desired reference path.

2.2.3 Non-linear Model Predictive Control in row-crops

In their work, [16] proposes a comprehensive navigation system based on the analysis of the Point Cloud Data (PCD) derived from the fusion of data from four RGBD cameras, aimed at expanding the Field of View (FOV) of the algorithm. Initially, a CV algorithm is employed to detect and estimate tree trunk positions by identifying shadows in the PCD generated by the presence of trees. Subsequently, the reference path is constructed as a local path that is iteratively updated utilizing the positions of the trees obtained from the vision system.

To facilitate row traversals, a Voronoi diagram is employed, while a spiral model is applied for headland maneuvers. Furthermore, a Non-Uniform Rational Spline (NURBS) curve is computed to connect different sections of the path, providing a smooth and continuous reference for the robot to follow. This approach enables the integration of both in-row and headland navigation during path computation, accommodating various orchard layouts beyond rectangular shapes. Moreover, it offers a more consistent reference compared to the conventional straight-line following approach, taking into account the robot's orientation.

The path following is executed using a Non-linear Model Predictive Control (NMPC) scheme, optimizing the error between the computed and desired paths over the entire prediction horizon. This approach ensures efficient navigation while considering specific constraints such as actuator saturation.

2.3 Remarks

In summary, the navigation system plays a pivotal role in modern precision agriculture, with extensive research exploring various methodologies. An established solution involves the utilization of GNSS systems, often enhanced by corrections from multiple receivers and RTK corrections. While GNSS positioning can complement odometry systems, its accuracy is affected by dense canopies and trees.

Alternative methods, such as Visual Odometry (VO), have been investigated to localize rovers using image streams from cameras. However, challenges arise in row-crop fields due to environmental repetitiveness. To address these issues, position-agnostic systems have been proposed. These systems directly map sensor data to rover velocity commands without relying on fixed Reference Frames. For instance, [11, 10, 13] proposed to segment the input image to compute a set point in the camera frame and to use a proportional controller to align the rover towards the set point. These methods, however, fail in the case of pergola vineyards or high trees in which the sky is not visible [11, 13] or the crops are not uniform on both sides [10]. [15] also decides to use an input segmentation system to cluster the system into terrain, crops, and sky, but then it computes a reference path to be followed. Segmentation-based methods, however, encounter difficulties in dealing with different seasons, and with unexpected obstacles in the path.

Decision algorithms provide another avenue, with DRL agents trained by [9] for decision-making and CNN trained by [14] to output actions from a discrete set. Additionally, [16] introduced a path-following NMPC approach, leveraging a PCD from four cameras to generate the reference path.

Chapter 3

Fundamentals of Computer Vision

The ability of robots to effectively interact with their environment hinges upon their capacity to perceive and understand the surrounding world. One interesting approach similar to human perception involves exploiting the sense of vision. In today's technological landscape, cameras offer a cost-effective, versatile, and widely applicable sensor solution compared to more specialized alternatives such as LIDARs or GPS systems. This chapter will introduce the fundamental tools and concepts essential for implementing Computer Vision (CV) applications in robotics, highlighting the pivotal role of vision-based perception in enabling robots to navigate and comprehend their surroundings.

3.1 Image Formation

Firstly, the fundamental process of image formation and capture will be discussed as a cornerstone in the domain of CV systems. This initial step is pivotal as it lays the foundation for subsequent image analysis and interpretation. Essentially, this process entails *projection*, whereby the three-dimensional world is mapped onto a two-dimensional plane. From the physical point of view, light emanating from the external environment converges onto a two-dimensional surface, such as the human retina or the semiconductor chip equipped with an array of light-sensitive elements found in modern cameras. In this transformation, the depth information is lost, and, for this reason, it is not possible to distinguish a bigger object sufficiently far from the observation point of view and a smaller closer object (to overcome this problem, a possible approach is discussed in Sec. 3.4, which introduces the principles of stereo vision).

3.1.1 Pin-hole camera model

The simplest camera model is the *pin-hole camera model*. This concept was employed in the creation of the first camera (Fig. 3.1): within a completely enclosed and darkened chamber, a small aperture is made on one side. Light enters through this aperture, allowing

images to be visible and captured on a light-sensitive medium (such as photographic film). The size of the aperture directly influences the amount of light that enters the camera, affecting the brightness of the resulting images. However, a larger aperture comes at the cost of reduced image sharpness. Convex lenses play a pivotal role in forming images similar to those produced by a pinhole, but due to their larger diameter, they permit a more substantial amount of light to pass through. It is also noteworthy that the image formed on the opposite wall to the aperture is inverted with respect to the external world.

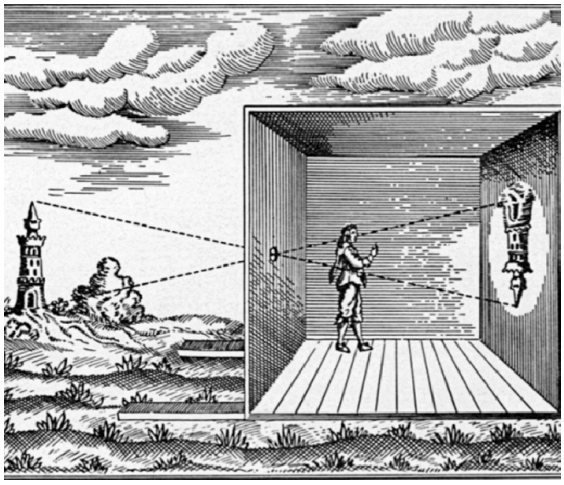


Figure 3.1: Camera obscura depiction of A. Kircher (Ars Magna, 17th century). [17]

The *central perspective imaging model* is a modified version of the previous one, and it is commonly used in Computer Vision (CV). In this model, the image plane is positioned at $z = f$, with f focal length of the camera $[mm]$, with respect to the camera Reference Frame (RF) (located in correspondence of the pin-hole) and oriented as shown in Fig. 3.2.

Analyzing the scheme in Fig. 3.2 and using similar triangles, it is possible to obtain the transformation between a point at the world coordinates $\boldsymbol{P} = (X, Y, Z)$ and the projected point on the image plane $\boldsymbol{p} = (x, y)$.

$$x = f\frac{X}{Z}, \qquad y = f\frac{Y}{Z} \tag{3.1}$$

The perspective projection from the world RF to the image plane exhibits the following characteristics:

1. It maps a 3-component input vector (representing the three-dimensional world) onto a two-dimensional space (the image plane): $\mathbb{R}^3 \mapsto \mathbb{R}^2$.

2. Straight lines in the three-dimensional world are projected as straight lines on the image.

3. Parallel lines in the world are projected as parallel lines on the image plane only if they are also parallel to the image plane itself; otherwise, they will intersect at a vanishing point.

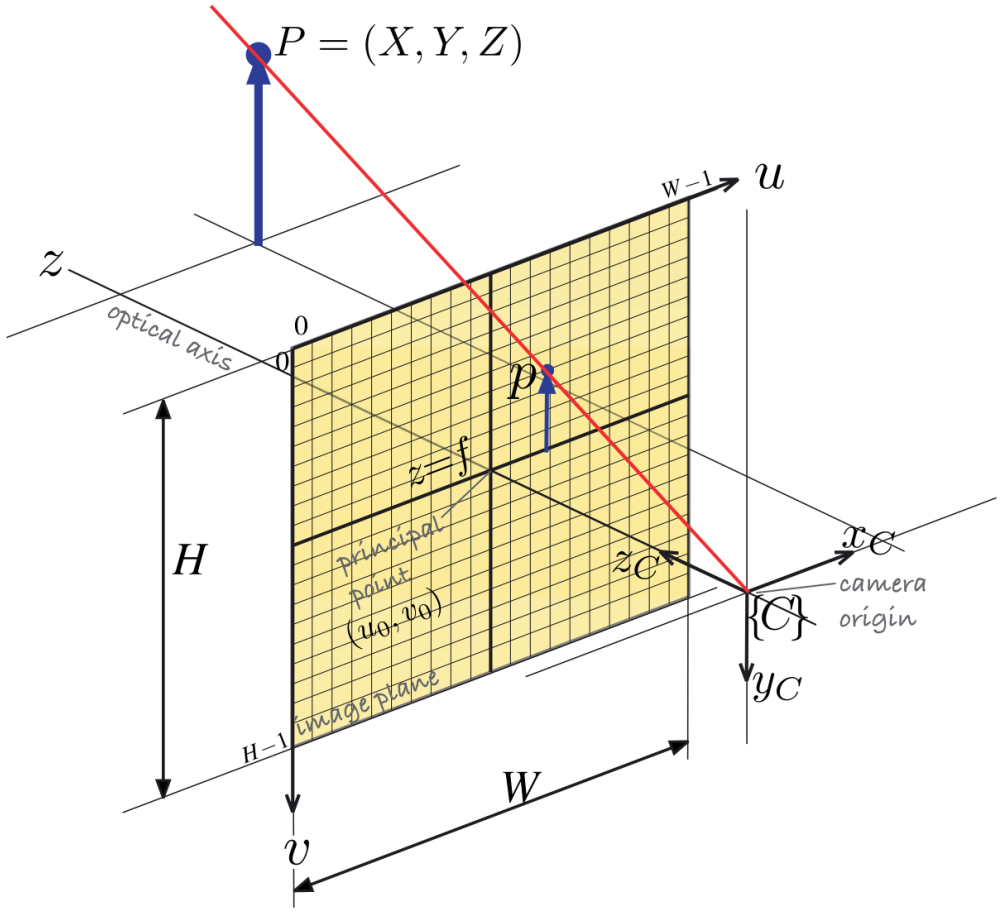


Figure 3.2: Central perspective imaging model. The image plane is located between the point $\boldsymbol{P}$ in the world and the camera origin (the pin-hole) to obtain a non-inverted image. [18]

4. Conic sections in the world are transformed into conics on the image plane, although not necessarily identical ones. For instance, a circle may be projected as an ellipse due to the perspective transformation.

5. The mapping is not injective (one-to-one): multiple points in the world can be mapped to the same point on the image plane, particularly all points in the world lying on the straight line passing through the camera's origin and the point on the image plane. This characteristic is responsible for the loss of depth information of objects following the perspective transformation.

6. Since the internal angles are not preserved, this transformation does not preserve shape: is not *conformal*. Examples of conformal transformation are translations, rotations, and scaling.

3.1.2 Intrinsic and extrinsic matrix

Starting from Eq. 3.1, it is possible to rewrite the image plane points in homogeneous coordinates $\tilde{\boldsymbol{p}} = (x', y', z')^T$ where

$$x' = f \cdot X, \qquad y' = f \cdot Y, \qquad z' = Z \tag{3.2}$$

The tilde underlines that a general vector is expressed in homogeneous coordinates (see Appendix A for a brief discussion about homogeneous coordinates). Then, exploiting Eq. 3.1 and Eq. 3.2, the correspondent non-homogeneous image plane coordinates are

$$x = \frac{x'}{z'}, \qquad y = \frac{y'}{z'} \tag{3.3}$$

Let's also consider the world coordinate in homogeneous form $\tilde{\boldsymbol{P}} = (X, Y, Z, 1)^T$. Moreover, in general, the camera will have a generic pose with respect to a different RF of interest, e.g., the robot RF, or the fixed world RF. From now on, all the vectors expressed in the camera RF will be denoted with the apex ^{C}, while vectors in the world RF with ^{0}. For this reason, it is important to underline the relation between the two RFs:

$$\tilde{\boldsymbol{P}}^0 = \boldsymbol{T}_C^0 \cdot \tilde{\boldsymbol{P}}^C \tag{3.4}$$

where $\boldsymbol{T}_C^0$ represent the homogeneous transformation between a coordinate vector (representation of a point in the space) expressed in camera RF and the correspondent vector expressed in the world RF.

In a digital camera, the image plane is not continuous, but is composed of a grid of small light-sensitive devices called *photo-diode*. So, also the images generated from a digital camera are a grid of small elements called *pixels*. Each pixel corresponds to a photo-diode, and can be denoted as a two-dimensional vector of coordinates (u, v); since u and v represent an index on a grid, they are non-negative integers (the origin is on the top-left corner of the image, as shown in Fig. 3.2). The transformation between the pixel indexes (u, v) and the image plane coordinates (x, y) is

$$u = \frac{x}{\rho_w} + u_0, \qquad v = \frac{y}{\rho_h} + v_0 \tag{3.5}$$

where ρ_w and ρ_h are the width and height of a single photo-diode $[mm]$ and (u_0, v_0) are the coordinates of the principal points, a.k.a. the point of intersection between the optical axis (the z axis of the camera RF) and the image plane. If we combine these results with Eq. 3.1, we obtain

$$\begin{aligned} u &= \frac{f}{\rho_w}\frac{X}{Z} + u_0 = f_w\frac{X}{Z} + u_0 \\ v &= \frac{f}{\rho_h}\frac{Y}{Z} + v_0 = f_h\frac{Y}{Z} + v_0 \end{aligned} \tag{3.6}$$

or in homogeneous coordinates $\tilde{\boldsymbol{p}} = (u', v', w')^T$:

$$\begin{aligned} u' &= f_w \cdot X + u_0 \cdot Z \\ v' &= f_h \cdot Y + v_0 \cdot Z \\ w' &= Z \end{aligned} \tag{3.7}$$

It is important to notice that f_w and f_h are adimensional quantities since they are the ratio between two distances $[mm]$. From the homogeneous pixel coordinates it is possible to recover the non-homogeneous ones:

$$u = \frac{u'}{w'}, \qquad v = \frac{v'}{w'} \tag{3.8}$$

To summarize this process, it is possible to write the perspective projection transformation in homogeneous coordinates in *linear* form.

$$\begin{aligned} \tilde{\boldsymbol{p}} = \begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} &= \begin{bmatrix} f_w & 0 & u_0 \\ 0 & f_h & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot (\boldsymbol{T}_C^0)^{-1} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \\ &= \boldsymbol{K}\boldsymbol{N}(\boldsymbol{T}_C^0)^{-1}\tilde{\boldsymbol{P}}^0 \\ &= \boldsymbol{C}\tilde{\boldsymbol{P}}^0 \end{aligned}$$

where:

1. $\boldsymbol{K}$ is the camera parameters matrix, it contains the four parameters that describe the camera f_w, f_h, u_0, v_0.

2. $\boldsymbol{N}$ is called the *projection matrix*.

3. The product $\boldsymbol{K}\boldsymbol{N}$ is the *intrisic matrix*: it is the relation between the pixel homogeneous coordinates and the three-dimensional world coordinates expressed in camera RF.

4. $(\boldsymbol{T}_C^0)^{-1}$ is the *extrinsic matrix*.

5. The matrix $\boldsymbol{C} = \boldsymbol{K}\boldsymbol{N}(\boldsymbol{T}_C^0)^{-1}$ is a 3×4 homogeneous transformation that converts a point expressed in homogeneous coordinate in the world RF $\tilde{\boldsymbol{P}}^0$ into the correspondent point expressed in homogeneous coordinate in the image RF $\tilde{\boldsymbol{p}}$.

Finally, using the non-linear Eq. 3.8 it is possible to obtain the pixel coordinates (u, v) corresponding to a generic point $\boldsymbol{P}^0$.

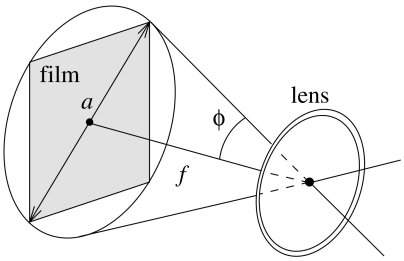


Figure 3.3: Field of view of a camera. [19]

3.1.3 Field of View

The Field of View (FOV) of a camera is *"the portion of the scene that projects onto the retina of the camera"* [19]. It is a function of both the camera focal length f and the physical effective area of the retina (the area of the light-sensitive film exposed in the camera; see Fig. 3.3).

Analyzing the system's geometry and regarding the FOV as a cone, as depicted in Fig. 3.3, we can define the FOV as 2θ, where:

$$\theta = \arctan \frac{a}{2f} \tag{3.9}$$

Here, a represents the diameter of the sensor used $[mm]$, and f denotes the camera's focal length $[mm]$.

A more in-depth analysis takes into account the distinctions between the vertical and horizontal planes, considering that the film is typically rectangular. With these considerations, it is possible to define the FOV as *"an open rectangular pyramid that subtends angles $2\theta_h$ and $2\theta_v$ in the horizontal and vertical planes respectively"* [18]. These angles are defined as follows:

$$\begin{aligned} \theta_h &= \arctan \frac{W\rho_w}{2f} = \arctan \frac{W}{2f_w} \\ \theta_w &= \arctan \frac{H\rho_h}{2f} = \arctan \frac{H}{2f_h} \end{aligned} \tag{3.10}$$

where W and H are the number of pixels of the image in the horizontal and vertical direction respectively.

It is important to notice that:

- Since $\rho_w \times \rho_h$ is the physical dimension of a single sensor (a.k.a. a single pixel), $W\rho_w \times H\rho_h$ is the size of the entire light-sensitive chip inside the camera.
- Eq. 3.9 and 3.10 essentially convey the same concept. In Eq. 3.9 a and f are both distances, so their ratio is adimensional; the same results are obtained in Eq. 3.10 with $W\rho_w$ (or $H\rho_h$) and f (both distances), or with two adimensional quantities such as W and f_w (or H and f_h).

3.2 Image Filtering

In real-world scenarios, where images are inevitably influenced by various forms of noise and disturbances, the application of image filtering is crucial in digital signal processing. Whether in the context of Computer Vision, medical imaging, or remote sensing, the use of image filtering methods contributes significantly to improving the overall robustness and clarity of visual data, enabling more accurate analysis and interpretation.

3.2.1 Monodic Operations

Monodic operations are the simplest class of image-processing operations. Given an input image of size $W \times H$, it processes each pixel independently with the same function f and returns as output another image of the same size as the input one. This process can be formally written as

$$\boldsymbol{O}[u, v] = f(\boldsymbol{I}[u, v]), \quad \forall (u, v) \in \boldsymbol{I} \tag{3.11}$$

and it is schematically presented in Fig. 3.4.

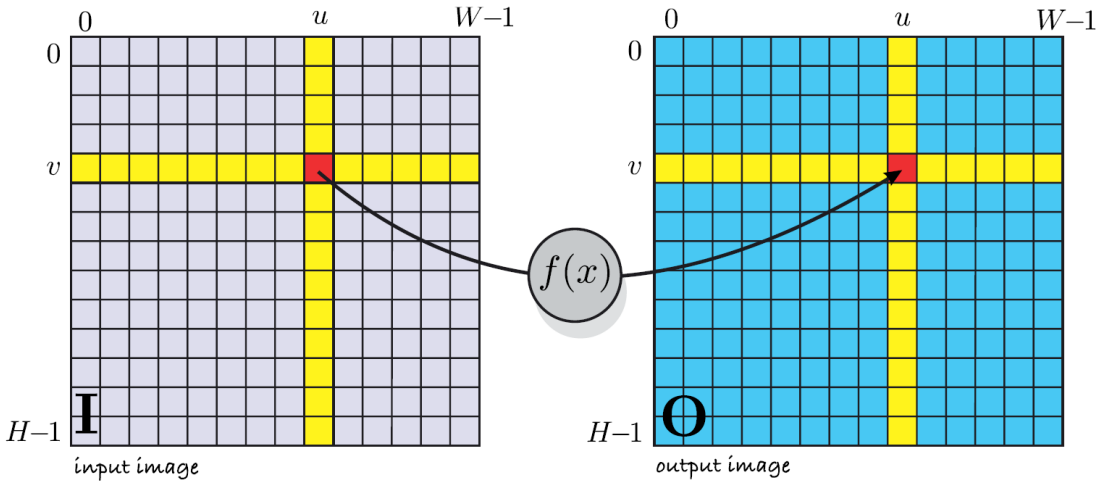


Figure 3.4: Monadic image-processing operations. Each output pixel is a function of the corresponding input pixel (shown in red). [18]

For instance, a first monadic operation is the change of the datatype of each pixel, e.g., from `uint8` (integer in range $[0, 255]$) to double precision values in the range [0,1]. Another basic operation is the conversion of a color image to the corresponding grey-scale image. It is important to notice that a color image has 3-dimension (since each pixel is a 3-tuple representing the color, e.g., RGB format), while a grey-scale image is a 2-dimension grid: at each pixel is associated a number representing its grey-scale level.

Moreover, frequently utilized unary operations include *thresholding* operations, which involve selecting specific pixels within a given range from a gray-scale image. Additionally,

some operations focus on manipulating the distribution of gray-scale levels within the image. This distribution can be quantified and visualized through a *histogram* of the image, where each bin corresponds to a gray-scale level (e.g., in a `uint8` image, each bin represents an integer within the range $[0, 255]$).

In certain situations, the input image may not cover the entire spectrum of available levels. For instance, an underexposed image may lack pixels with high values. In such cases, a straightforward linear mapping, as depicted in Eq. 3.12, can be employed to *stretch* the histogram and span the complete range of values.

$$p_{\text{new}}(u, v) = p_{\text{old}}(u, v) \cdot \frac{255}{\max(p_{\text{old}}(u, v))} \quad \forall (u, v) \in \boldsymbol{I} \tag{3.12}$$

Likewise, using a similar approach, *histogram normalization* or *histogram equalization* can be defined. This method utilizes a linear mapping to obtain a linear cumulative distribution of pixel intensities, resulting in a uniform histogram where each bin contains an equal number of pixels.

Image stretching and histogram normalization do not introduce new information to the input image; consequently, subsequent image processing steps may not experience significant improvements. Nevertheless, these operations can enhance the image from a human observer's perspective.

3.2.2 Dyadic Operations

Dyadic operations involve the manipulation of two input images of identical dimensions, yielding a single output matrix of the same size as the input images. Each output pixel's value is determined by a function of the corresponding pixels in the two input images (refer to Fig. 3.5 for a schematic illustration).

$$\boldsymbol{O}[u, v] = f(\boldsymbol{I_1}[u, v], \boldsymbol{I_2}[u, v]), \quad \forall (u, v) \in \boldsymbol{I_1} \tag{3.13}$$

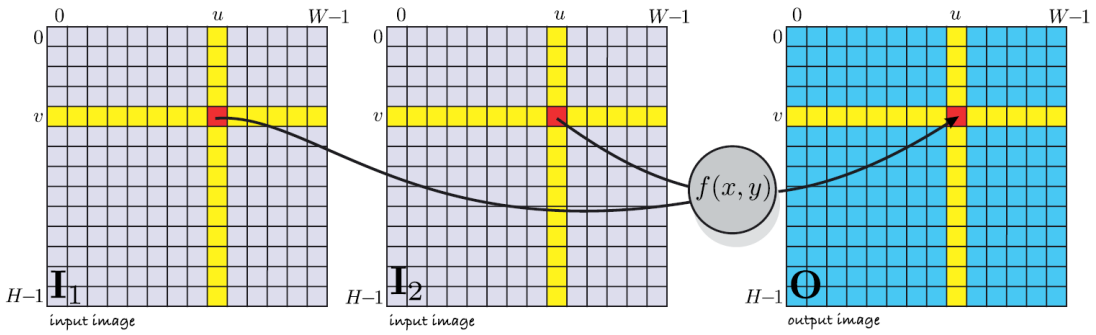


Figure 3.5: Dyadic image-processing operations. Each output pixel is a function of the two corresponding input pixels (shown in red). [18]

Some useful dyadic operations are binary arithmetic operators such as addition, subtraction, element-wise multiplication, max, min, etc. Moreover, for example, this

technique is used in green-screen technology: after the subject is distinguished from the green background, it is obtained a mask of the same size as the input image, with binary values differentiating subject (1 or *True*) from the background (0 or *False*). Then, the output image is constructed as

$$\boldsymbol{O}[u,v] = \boldsymbol{I_1}[u,v] \cdot \boldsymbol{M}[u,v] + \boldsymbol{I_2}[u,v] \cdot (\mathbf{1} - \boldsymbol{M}[u,v]), \quad \forall (u,v) \in \boldsymbol{I_1} \tag{3.14}$$

where $\boldsymbol{I}_1$ is the subject image with the green screen background, $\boldsymbol{I}_2$ is the new background image (same size of $\boldsymbol{I}_1$), M is the binary mask obtained from $\boldsymbol{I}_1$ (also of the same size of $\boldsymbol{I}_1$) and $\cdot$ represent the element-wise multiplication (or the logical *and*). To better explain the formula, in the output image all the pixels selected by the mask are taken from the subject image $\boldsymbol{I}_1$, while all the others are a copy of the corresponding ones in the new background image $\boldsymbol{I}_2$.

3.2.3 Spatial Operations

Spatial image operations are performed on a single input image and return a single output image of the same dimensions. Each pixel in the output image is a function of all the pixels in a neighborhood of the corresponding pixel in the input image (refer to Fig. 3.6 for a schematic illustration).

$$\boldsymbol{O}[u,v] = f(\boldsymbol{I}[u+i, v+j]), \quad \forall (i,j) \in \mathcal{W}, \quad \forall (u,v) \in \boldsymbol{I} \tag{3.15}$$

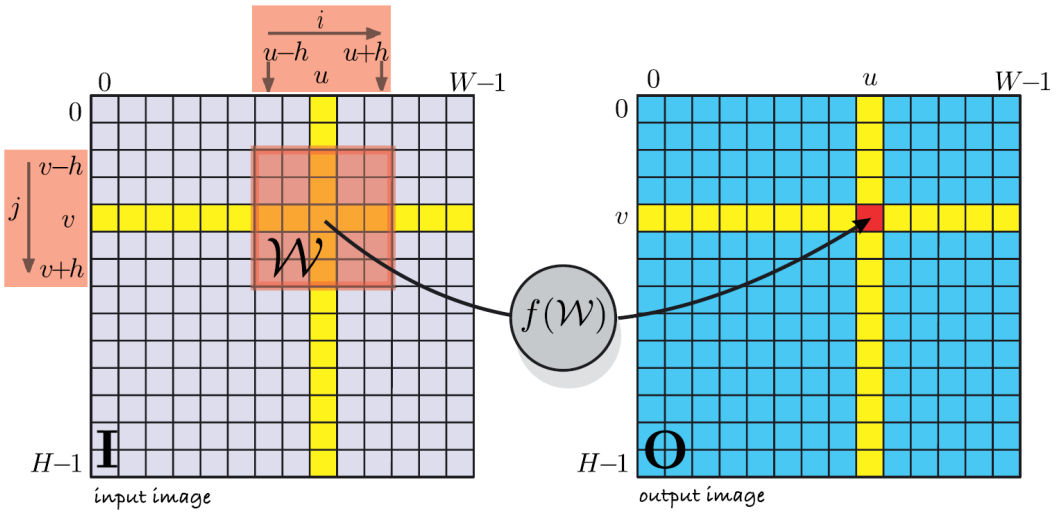


Figure 3.6: Spatial image processing operations. The red-shaded region shows the window $\mathcal{W}$, which is the set of pixels used to compute the output pixel (shown in red). [18]

The mathematical operations that describes a linear spatial filter is *convolution* $\boldsymbol{O} = \boldsymbol{K} \otimes \boldsymbol{I}$ defined as

$$\boldsymbol{O}[u,v] = \sum_{(i,j)\in\mathcal{W}} \boldsymbol{I}[u+i, v+j]\boldsymbol{K}[i,j], \quad \forall (i,j) \in \mathcal{W}, \quad \forall (u,v) \in \boldsymbol{I} \tag{3.16}$$

where $\boldsymbol{K} \in \mathbb{R}^{w\times w}$ is called *convolution kernel* and is element-wise multiplied with the window of pixels $\mathcal{W}$, for every output pixel. In other words, the kernel $\boldsymbol{K}$ is the matrix of weight of a weighted sum within the window. It is important to notice that:

- Different kernels correspond to different functions, such as smoothing, edge detection, etc.
- Convolution is a computationally expensive operation: a $N \times N$ input image with a $w \times w$ kernel requires w^2N^2 multiplications and an equal number of additions.
- If the image has multiple color planes (e.g., the RGB format), the convolution is performed independently for each input plane with the kernel $\boldsymbol{K}$.
- Convolution is not well-defined at the edges of the input image since the window $\mathcal{W}$ will contain pixels outside the input image (not defined pixels). Lots of different approaches can be defined to overcome this problem, such as assigning a fixed value (e.g., 0) to all the pixels outside the input image, replicating the value of the pixels at the border of the image, or not considering the results when the window exceeds the boundary of the image (the output image will be smaller than the input one).

A first important kernel is the *Gaussian kernel*. It is defined from the corresponding 2D Gaussian function

$$\boldsymbol{G}(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}} \tag{3.17}$$

This function is symmetric about the origin, has a unitary volume under the curve, and its spread in both directions is controlled by the variance σ^2 (refer to Fig. 3.7 for an example). Convolution with this kernel is suitable for image *smoothing*.

Edge detection

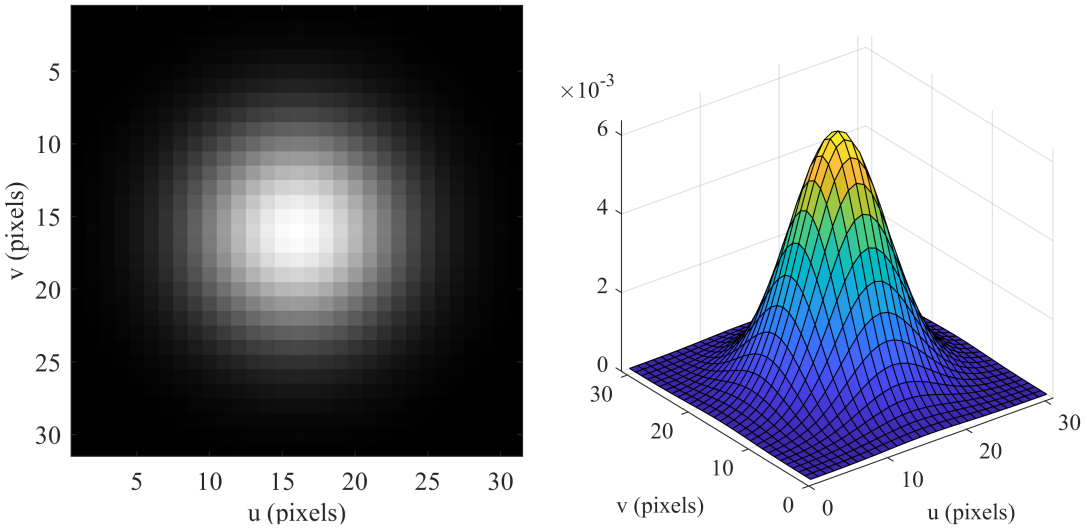
Another important application for linear kernels is *edge detection*. The basic idea is that a very rapid variation of the intensity of the grey level is a reliable indication of an edge. Let us consider the horizontal first-order derivative of the grey level, and rewrite it as a *symmetrical* first-order difference

$$p'[u^\star, v] = \frac{1}{2}(p[u^\star, v+1] - p[u^\star, v-1]), \quad \forall (u,v) \in \boldsymbol{I} \tag{3.18}$$

that is equivalent to convolution with a 1D kernel

$$\boldsymbol{K} = \begin{bmatrix} -\frac{1}{2} & 0 & \frac{1}{2} \end{bmatrix} \tag{3.19}$$

Starting from this idea, many kernels can be constructed, such as the Sobel Kernel



(a) Gaussian function shown as intensity image

(b) 2D Gaussian function.

Figure 3.7: Gaussian kernel of size 31×31 with $\sigma^2 = 25$.

$$\boldsymbol{D_u} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{3.20}$$

It is important to notice that the derivative kernel in the vertical direction is simply the transpose of the horizontal one. Then, the gradient along the horizontal or vertical direction can be computed as

$$\begin{aligned} \nabla_u \boldsymbol{I} &= \boldsymbol{D}_u \otimes \boldsymbol{I} \\ \nabla_v \boldsymbol{I} &= \boldsymbol{D}_u^T \otimes \boldsymbol{I} \end{aligned} \tag{3.21}$$

where ∇_i represent the gradient along the i-direction and $\boldsymbol{D}$ the derivative kernel such as the Sobel kernel presented in Eq. 3.20.

Moreover, must be considered that the derivative accentuates high-frequency noise [18], so usually a smoothing operation is performed before taking the derivative.

$$\nabla_u \boldsymbol{I} = \boldsymbol{D}_u \otimes (\boldsymbol{G}(\sigma) \otimes \boldsymbol{I}) \tag{3.22}$$

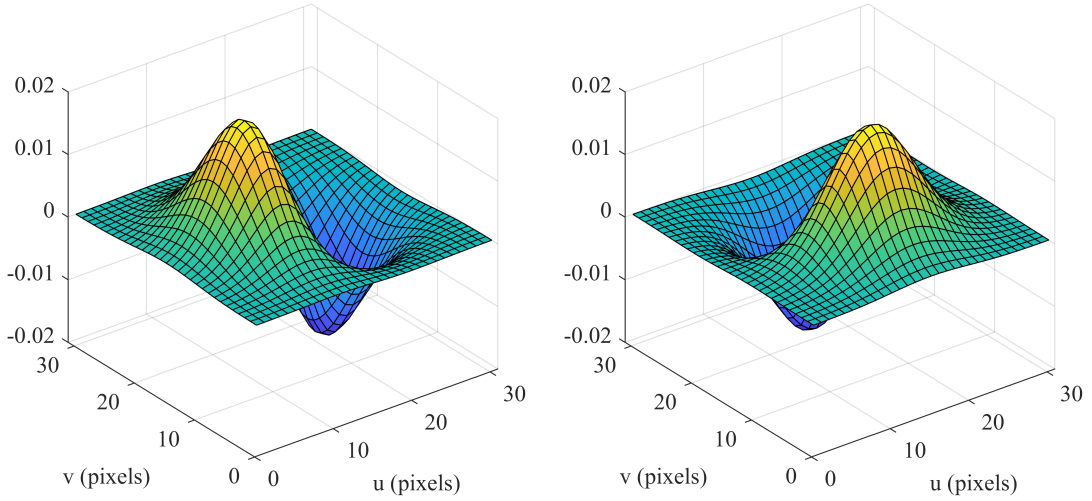
and using the associative property of convolution

$$\nabla_u \boldsymbol{I} = (\boldsymbol{D}_u \otimes \boldsymbol{G}(\sigma)) \otimes \boldsymbol{I} = \boldsymbol{DoG_u}(\sigma) \otimes \boldsymbol{I} \tag{3.23}$$

where $\boldsymbol{DoG}$ represent the *derivative of the Gaussian*, in the u-direction, that can be computed analytically from Eq. 3.17 obtaining

$$\boldsymbol{DoG}_u(u, v) = -\frac{u}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}} \tag{3.24}$$

The standard deviation σ must be tuned according to the *scale* of the edges that the system should detect: larger σ implies increased smoothing and so only edges of large features will be detected, while fine texture will be attenuated. The transpose of this kernel can be used for vertical gradient (as shown in Fig. 3.8).



(a) Derivative of the Gaussian in the u-direction. **(b)** Derivative of the Gaussian in the v-direction.

Figure 3.8: Derivative of the Gaussian kernel of size 31×31 with $\sigma^2 = 25$.

A step further in the edge detection problem is to consider the second derivative: an edge is no longer defined as a pixel with a high gradient, but it is a maximum in the local neighborhood. So, we need to compute the second derivative and determine where this is zero. The Laplacian operator can be defined as

$$\nabla^2 \boldsymbol{I} = \nabla_u^2 \boldsymbol{I} + \nabla_v^2 \boldsymbol{I} = \boldsymbol{L} \otimes \boldsymbol{I} \tag{3.25}$$

or, in other words, as the sum of the second spatial derivative in both horizontal and vertical direction. Considering that it is possible to rewrite the second order derivative in the u-direction (in the v-direction it is simply the transpose one) as a discrete finite-difference equation

$$p''[u^\star, v] = p[u^\star, v+1] - 2p[u^\star, v] + p[u^\star, v-1], \quad \forall (u, v) \in \boldsymbol{I} \tag{3.26}$$

that is equivalent to convolution with a derivative kernel

$$\boldsymbol{D_u^2} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \tag{3.27}$$

it is possible to obtain the Laplacian kernel

$$\boldsymbol{L} = \boldsymbol{D}_u^2 + \boldsymbol{D}_u^{2^T} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{3.28}$$

An important characteristic of L is that it is isotropic, so responds equally to edges in any direction.

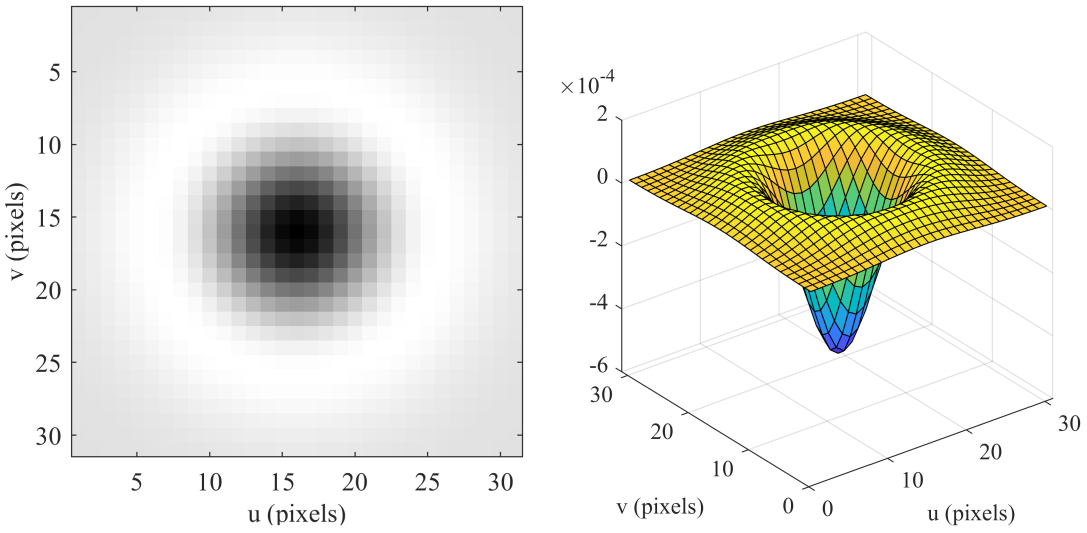
Usually, the second derivative is used in conjunction with a Gaussian smoothing

$$\boldsymbol{L} \otimes (\boldsymbol{G}(\sigma) \otimes \boldsymbol{I}) = (\boldsymbol{L} \otimes \boldsymbol{G}(\sigma)) \otimes \boldsymbol{I} = \boldsymbol{LoG}(\sigma) \otimes \boldsymbol{I} \tag{3.29}$$

where $\boldsymbol{L}$ is the Laplacian kernel defined above. The $\boldsymbol{LoG}$ can be also computed analytically as

$$\boldsymbol{LoG}_u(u, v) = \frac{1}{\pi\sigma^4}(\frac{^2 + v^2}{2\sigma^2} - 1)e^{-\frac{u^2+v^2}{2\sigma^2}} \tag{3.30}$$

which is known as the Marr-Hildreth operator, or the *Mexican hat kernel* and it is shown in Fig. 3.9.



(a) Laplacian of Gaussian function shown as intensity image

(b) 2D Laplacian of Gaussian function.

Figure 3.9: Laplacian of Gaussian kernel of size 31×31 with $\sigma^2 = 25$.

Inherent limitations are present in all of these approaches, as they tend to associate intensity edges with the boundaries of objects. For example, shadows may exhibit extremely sharp edges, yet they cannot be classified as distinct "objects." Additionally, the object of interest may exhibit a limited contrast when compared to its background, leading to unreliable boundary detection.

Template Matching

The convolution kernel can be also seen as an *image* (or a part of), so the goal of the application can be to find the part of our input image most similar to the kernel chosen. In this context, the kernel is usually referred to as *template*. So, *template matching* can be expressed as

$$\boldsymbol{O}[u,v] = s(\boldsymbol{T}, \mathcal{W}[u,v]), \quad \forall (u,v) \in \boldsymbol{I} \tag{3.31}$$

where $\mathcal{W}[u,v]$ is the window of size $w \times w$ centered in (u,v), $\boldsymbol{T}$ is the template (same size of $\mathcal{W}$) and $s(\boldsymbol{I}_1, \boldsymbol{I}_2)$ is a function of two equally sized images that returns a scalar measure of the *similarity* of the two inputs.

Sum of absolute differences	
SAD	$s = \sum_{(u,v)\in I} \lvert I_1[u,v] - I_2[u,v] \rvert$
ZSAD	$s = \sum_{(u,v)\in I} \left\lvert \left(I_1[u,v] - \bar{I}_1\right) - \left(I_2[u,v] - \bar{I}_2\right) \right\rvert$
Sum of squared differences	
SSD	$s = \sum_{(u,v)\in I} \left(I_1[u,v] - I_2[u,v]\right)^2$
ZSSD	$s = \sum_{(u,v)\in I} \left(\left(I_1[u,v] - \bar{I}_1\right) - \left(I_2[u,v] - \bar{I}_2\right)\right)^2$
Normalized Cross correlation	
NCC	$s = \dfrac{\sum_{(u,v)\in I} I_1[u,v] \cdot I_2[u,v]}{\sqrt{\sum_{(u,v)\in I} I_1^2[u,v] \cdot \sum_{(u,v)\in I} I_2^2[u,v]}}$
ZNCC	$s = \dfrac{\sum_{(u,v)\in I} \left(I_1[u,v] - \bar{I}_1\right) \cdot \left(I_2[u,v] - \bar{I}_2\right)}{\sqrt{\sum_{(u,v)\in I} \left(I_1[u,v] - \bar{I}_1\right)^2 \cdot \sum_{(u,v)\in I} \left(I_2[u,v] - \bar{I}_2\right)^2}}$

Table 3.1: Similarity measures for two equal-sized image regions $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$. $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$ are the mean value in the regions $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$ respectively. The Z-prefix indicates that the measure accounts for zero-offsets. [18]

Different functions have been proposed as measured for the similarity between the two image regions; some of them have been presented in Tab. 3.1. It is important to analyze some of the characteristics of these functions:

- Sum of the Absolute Differences (SAD) and Sum of the Squared Differences (SSD) return a value that is > 0, and return 0 if the two inputs are perfectly equal, while Normalized Cross Correlation (NCC) yields a score in the range $[-1, +1]$, with $+1$ in case of perfect match.

- Each similarity function has a *zero-offset* version, underlined with the Z-prefix, that accounts for offsets between the two input images: $\text{ZSAD}(\boldsymbol{T}, \boldsymbol{T} + \beta) = 0, \quad \beta \in \mathbb{R}$ (analog results also for ZSSD and ZNCC).

- Only cross-correlation functions NCC and ZNCC are invariant to gain variation: $\text{NCC}(\boldsymbol{T}, \alpha\boldsymbol{T}) = 1, \quad \alpha \in \mathbb{R}$, while the other measures will indicate a high degree of dissimilarity.

- NCC and ZNCC are computationally more expensive than the others, and also their result cannot be defined in case of denominator equal to zero, a.k.a. when the pixels in one (or both) of the two input images are all equal to zero (for NCC) or to the mean value (ZNCC).

- All these methods fail in case of relative scale or rotation, even small, of the two input images. Moreover, the square template includes also some pixels from the background together with the "subject": template matching can encounter some difficulties in case the background is changed. This problem is usually referred to as *mixed pixel problem* and can be attenuated using non-parametric similarity measures such as the *census* metric and the *rank transform*.

- Usually, several rules can be checked before a match is accepted, for instance, a threshold can be applied to the similarity functions, or more information about the motion of the camera or the subject can be used to try to avoid false matches.

Morphological Operations

Finally, another category of spatial operations is grounded in the application of *non-linear* functions to the pixels within windows. These functions include measures such as variance, maximum, minimum, and median. A distinct sub-category is found within *mathematical morphology*, where each pixel in the output is determined by a non-linear function applied to a *subset* $\mathcal{S}$ of the window $\mathcal{W}$. Typically, $\mathcal{S}$ is referred to as the *structuring element*. This particular class of operations, as the name suggests, deals with the manipulation of object *shapes*. These filters preserve only objects that can encompass the structuring element $\mathcal{S}$, while other shapes are either diminished or removed from the output image (refer to Fig. 3.10).

Morphological operation can be expressed in operator form:

- *Erosion*, that corresponds to use $f(\cdot) = min(\cdot)$ within the pixels in the window, is $\boldsymbol{O} = \boldsymbol{I} \ominus \mathcal{S}$.

- *Dilation*, that correspond to use $f(\cdot) = max(\cdot)$ within the pixels in the window, is $\boldsymbol{O} = \boldsymbol{I} \oplus \mathcal{S}$.

- *Opening*, shown in Fig. 3.10, is the sequence of erosion, then dilation, and it "opens up gaps". It is denoted as $\boldsymbol{O} = \boldsymbol{I} \circ \mathcal{S} = (\boldsymbol{I} \ominus \mathcal{S}) \oplus \mathcal{S}$ and it is a very powerful tool in cleaning the images and removing noises.

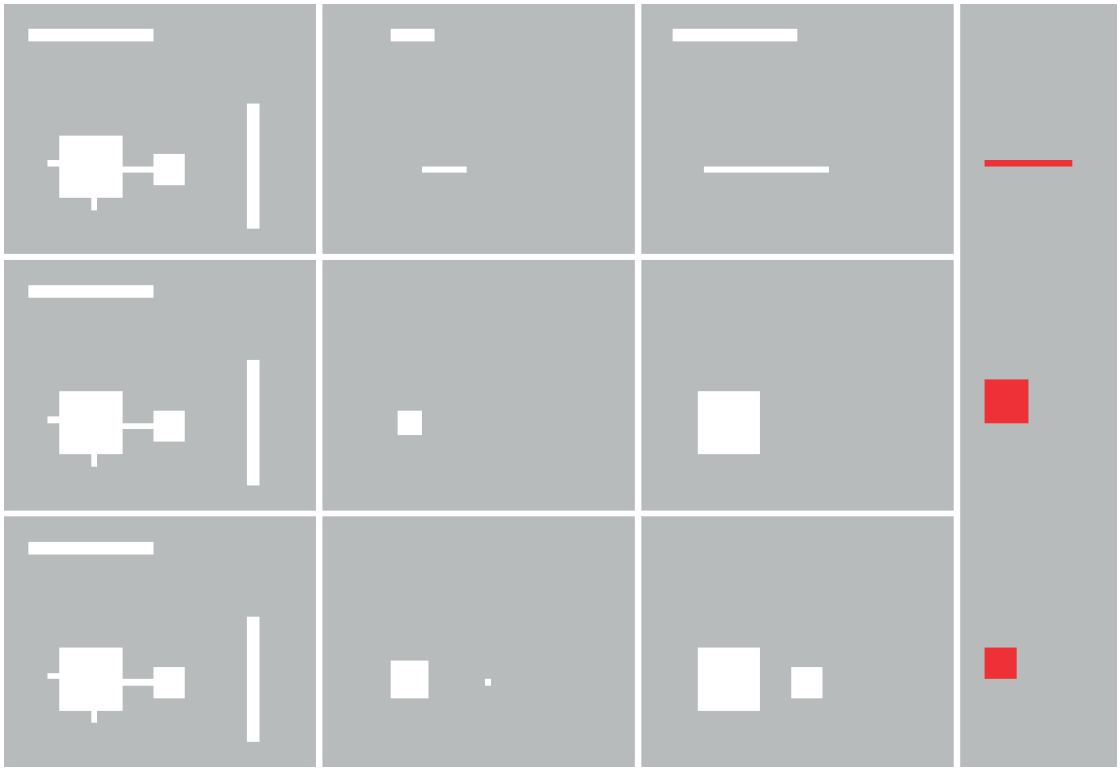


Figure 3.10: Opening example. Binary images with 0 (grey) or 1 (white). The structuring element for each row is shown in red in the last column. The first column represents the original image, the second is the image after the erosion by the corresponding structuring element, and the third one is the output after the second column is dilated. [18]

- *Closing*, shown in Fig. 3.11, is the sequence of dilation, then erosion, and it "closes gaps". It is denoted as $\boldsymbol{O} = \boldsymbol{I} \bullet \mathcal{S} = (\boldsymbol{I} \oplus \mathcal{S}) \ominus \mathcal{S}$ and it is the inverse sequence of operation of Opening.

3.2.4 Shape changing

Finally, there is a category of image processing operations designed to modify an image's shape or size.

One of the most straightforward and familiar operations in this context is *cropping*. This operation entails selecting all pixels within a specified Region of Interest (ROI), defined by specific coordinate ranges (u, v).

Another significant operation involves image resizing. To reduce the image's dimensions, a common technique is "sub-sampling." In this approach, only every m^{th} pixel in both the u- and v-directions is retained, with $m \in \mathbb{Z}^+$ serving as the sub-sampling factor. Consequently, the output image contains only $1/m^2$ of the pixels present in the input image, resulting in substantial memory savings. It is essential to acknowledge that sub-sampling can reduce the image's spatial sampling rate, potentially leading to spatial aliasing of high-frequency

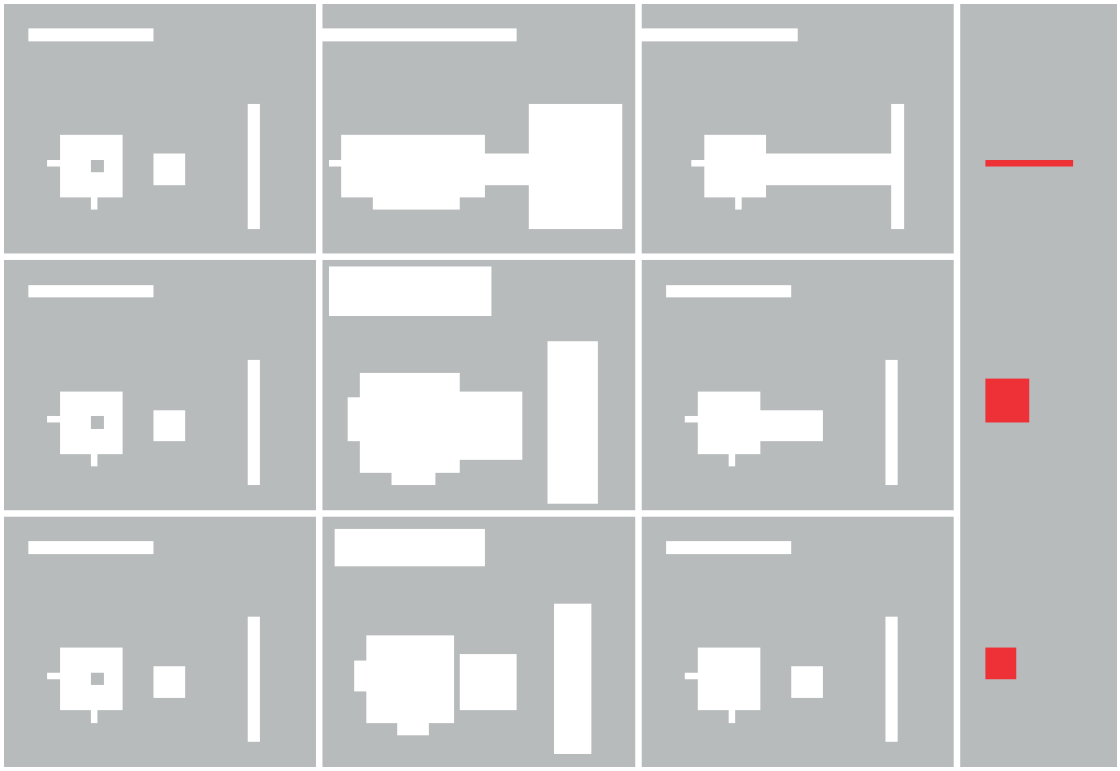


Figure 3.11: Closing example. Binary images with 0 (grey) or 1 (white). The structuring element for each row is shown in red in the last column. The first column represents the original image, the second is the image after dilatation by the corresponding structuring element, and the third one is the output after the second column is eroded. [18]

components, particularly those associated with texture or sharp edges [18]. Therefore, under the Shannon-Nyquist sampling theorem, a low-pass spatial filter, such as image blurring or the Gaussian kernel discussed in Sec. 3.2.3, should be applied to the input image before sub-sampling. Conversely, the inverse operation is *pixel replication*, where each pixel in the input image is substituted with a window of size $m \times m$, featuring the same pixel value in the output image. Once again, the application of a smoothing operation serves to mitigate the impact of the window's edges.

A far more sophisticated method, as the one used in the OpenCV library [20], considers that the resizing operation maps the input pixel grid to the destination one. So, if two scale factors f_v and f_u are provided (one for horizontal scaling and the other for vertical scaling) the theoretical mapping will be

$$\boldsymbol{O}[u, v] = \boldsymbol{I}[\frac{u}{f_u}, \frac{v}{f_v}] \quad \forall (u, v) \in \boldsymbol{O} \tag{3.32}$$

with $f_u, f_v < 1$ for image downsizing, and $f_u, f_v > 1$ for image up. However, this equation is not always feasible, since the pixel index must be integer numbers, while $\frac{u}{f_v}$ as well as $\frac{v}{f_v}$ are in general floating-point numbers. To overcome this problem, a polynomial function

is fit into some neighborhood of the computed pixel $(\frac{u}{f_u}, \frac{v}{f_v})$, and then the value of the polynomial at $(\frac{u}{f_u}, \frac{v}{f_v})$ is taken as the interpolated pixel value [20].

Finally, it's important to note that these methods are not suitable for depth images, such as those described in Sec. 3.4, because this resizing approach can alter the 3D shape of the reconstructed scene. A potential solution to this challenge is presented in [21], where the depth images are scaled while preserving the same aspect ratio between height and width, followed by a padding step to achieve the final size (refer to Fig. 3.12 for a schematic representation of the proposed solution).

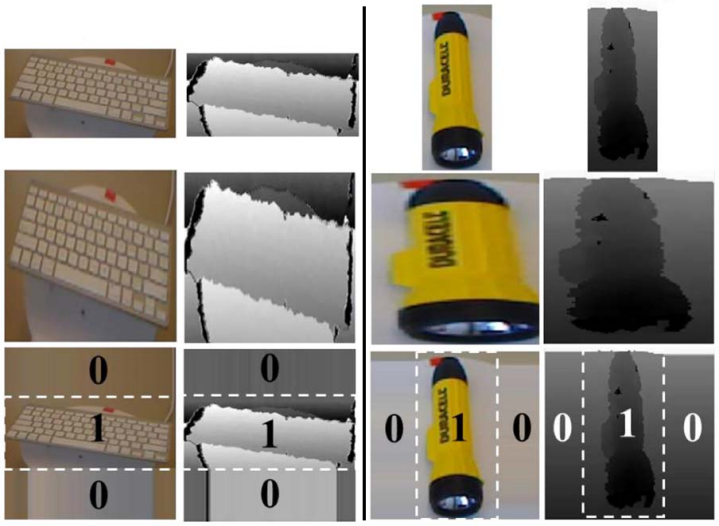


Figure 3.12: Resizing RGB-D images. The two columns show two different examples: the images of a keyboard and a flashlight. In the first row, the original images are presented, while in the second the resized ones are shown. The differences in the shape of the objects can be easily seen. Finally, in the last row, a possible idea of solution is presented. [21]

3.3 Image Segmentation

Image segmentation is a crucial task in Computer Vision that involves dividing an image into meaningful regions or segments [18]. The goal is to group pixels or pattern elements into summary representations that highlight important and distinctive properties [19]. This process can be compared to the statistical concept of *clustering* [22].

The segmentation process is typically divided into three key sub-problems [18]:

1. **Classification:** In this step, every pixel is assigned to one of a predetermined set of classes, often based on specific application requirements. For instance, classes could represent different colors, objects, or motion characteristics. The assumption is that regions within the image are *homogeneous* concerning a certain pixel property. Although misclassifications can occur, later stages of the process address and correct

them. Simple classification rules often involve thresholding on gray-scale or color information.

2. **Representation:** Once pixels of the same class are identified, they are *connected* to form spatial sets $S_1 \dots S_m$. Each set can be represented in various ways, such as by defining the boundaries of the connected region.

3. **Features extraction:** In the final step, each set S_i is described using scalar or vector-valued features that convey information about its size, position, shape, or other relevant characteristics.

The segmentation process plays a critical role in CV applications, enabling the analysis of visual data by breaking it down into more manageable and meaningful parts. It finds uses in a wide range of fields, including object recognition, scene analysis, and image understanding.

3.4 3D Reconstruction

In Sec. 3.1.1, we emphasized the loss of depth information inherent in the image formation process. In this section, we will introduce the *stereopsis* technique for reconstructing the three-dimensional structure of the world using a pair of images captured from different viewpoints. Specifically, we will focus on the *dense stereo matching* approach, which enables the retrieval of world coordinates $\boldsymbol{P} = (X, Y, Z)$ for each pixel within the image. Typically, this technique employs a stereo pair of images obtained from two RGB cameras with parallel optical axes, separated by a well-defined and precisely measured distance known as the *camera baseline*.

The Epipolar Constraint

In the stereo vision process, the initial step involves the fusion of features observed by two or, in more general cases, multiple cameras. Under the assumption that the cameras are internally (the internal parameters of the camera such as the focal length f, etc, are known) and externally calibrated (well-defined and fixed positions of these stereo cameras in space), this section will elucidate that the matching challenge between pairs of pixels is constrained by the requirement that pixel pairs from the two images must lie on corresponding *epipolar lines* within both images.

As illustrated in Fig. 3.13, the point $\boldsymbol{P} = (X, Y, Z)$ in the physical world, the optical centers O and O' of the two cameras, as well of the two projection $\boldsymbol{p}$ and $\boldsymbol{p}'$ of $\boldsymbol{P}$ onto the two images all lie on the same plane referred to as the *epipolar plane*. The camera baseline (the line between O and O') intersects the two image planes in e and e' respectively, which are called the *epipoles* of the two cameras. An alternative interpretation is that e' represents the projection of the optical center O from the first camera onto the image plane Π' of the second camera, and vice versa. Furthermore, the intersection between the epipolar plane and the two images' planes generates two lines, referred to as *epipolar lines*. Both the projected point $\boldsymbol{p}$ and $\boldsymbol{p}'$, as well as the two epipoles e and e' must lie on those lines, namely l and l'. In simpler terms, when the projection $\boldsymbol{p}$ is known, along with

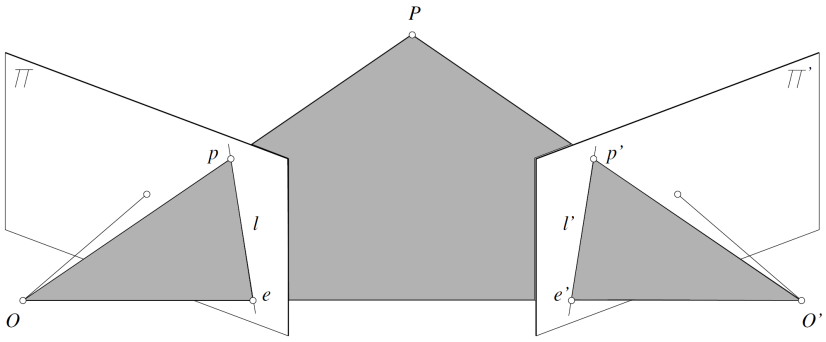


Figure 3.13: Epipolar geometry. In this figure, as well as in Fig. 3.2 the *virtual* image plane is located between the pin-hole O or O' and the point in the external world. [19]

the relative pose of the cameras, it becomes feasible to compute the epipolar line l' in the second image plane, as depicted in Fig. 3.14, then $\boldsymbol{p'}$ must lie on l'. This geometric property of stereo vision systems is termed the "epipolar constraint."

To summarize, within the stereopsis pipeline, establishing correspondences between pixels in the two images is a pivotal task. The presence of epipolar constraints significantly restricts the search for these correspondences. Given a pixel $\boldsymbol{p}$ in the first image, it is only necessary to seek its corresponding counterpart $\boldsymbol{p'}$ along the epipolar line l' within the second image, as opposed to scanning the entire image.

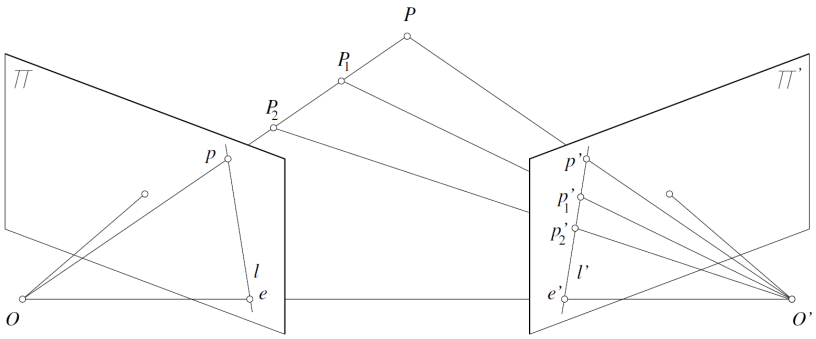


Figure 3.14: Epipolar constraint. [19]

In a lot of modern stereo cameras (as the one better described in Sec. 6.1.2), the two cameras are aligned with parallel optical axes (and so perpendicular to the baseline). In this configuration, the epipolar lines are horizontal in the image planes, and a very simple inverse relation can be established between the 3D depth Z and the disparity d

$$d = \frac{f_w b}{Z} \tag{3.33}$$

where b and Z are both distances, namely the baseline and the 3D-depth, f_w is the

horizontal focal length (in pixel) as defined in Sec. 3.1.2, and the disparity d describes the relation between the coordinate of a pixel in the two images.

$$u^L = u^R + d(u, v) \qquad v' = v \tag{3.34}$$

Additionally, it should be noted that Z must be positive, as objects situated behind the camera are not observable. Consequently, this condition dictates that $d > 0$ and $u_L > u_R$. In simpler terms, this implies that the search for the corresponding pixel in the right image must encompass all the pixels in the same row, with the added condition that the pixel in the right image should be to the left of the corresponding pixel in the left image. So the problem of reconstructing the 3D scene can be re-conducted to estimate the disparity map $d(u, v)$ for all pixels (u, v) in an image pair.

Dense correspondence

In order to be able to reconstruct the 3D scene, a.k.a. find world coordinates $\boldsymbol{P} = (X, Y, Z)$ for each pixel within the image, we have to find the horizontal shift of a pixel between the two images of a stereo pair (estimate the disparity map $d(u, v)$). During years of research, a lot of possible algorithms have been proposed to efficiently and robustly solve this problem. However, let me introduce a very simple approach to have a basic idea of how these algorithms work.

Algorithm 1 Template matching algorithm for a stereo pair.

procedure TEMPLATE MATCHING(I_L, I_R, H, V, W)
    ▷ I_L and I_R are the left and right images of the stereo pair
    ▷ The image pair has a resolution of $H \times V$ pixels.
    ▷ The template T will be a square of side W pixels.
    for `i in V` **do**
        for `j in H` **do**
            ▷ We are now considering pixel (i, j) of the first (*left*) image.
            Select the window T_1 across the selected pixel
            for `k in H` **do**
                ▷ We are now considering pixel (i, k) of the second (*right*) image.
                Select the window T_2 across the selected pixel
                Correspondence cost function $C(T_1, T_2)$ between the two windows in the two images.
            end for
            Select the pair (T_1, T_2) that maximize the cost function $C(T_1, T_2)$
            $d(i, j) = k - j$
        end for
    end for
    return d ▷ The *disparity map* is finally returned.
end procedure

In order to choose the corresponding pixel in the right image between the possible ones

(same row, $u^R < u^L$), a template matching approach like the one introduced in Sec. 3.2.3 can be used.

However, this algorithm is not impervious to errors and failures. For instance, in man-made scene such as the picket fence in Fig. 3.15, where regular vertical features are present, this algorithm struggle to decisively determine the optimal match for the template. With only two cameras, there is no real solution to this problem; however, it is possible to detect it. The *ambiguity ratio* is defined as the ratio between the height of the second peak (indicating the second-best match) and the maximum value of the correspondence cost function (representing the highest peak). A high ratio denotes a result fraught with uncertainty (where the second-best match closely resembles the first one), while conversely, very low values suggest successful outcomes.

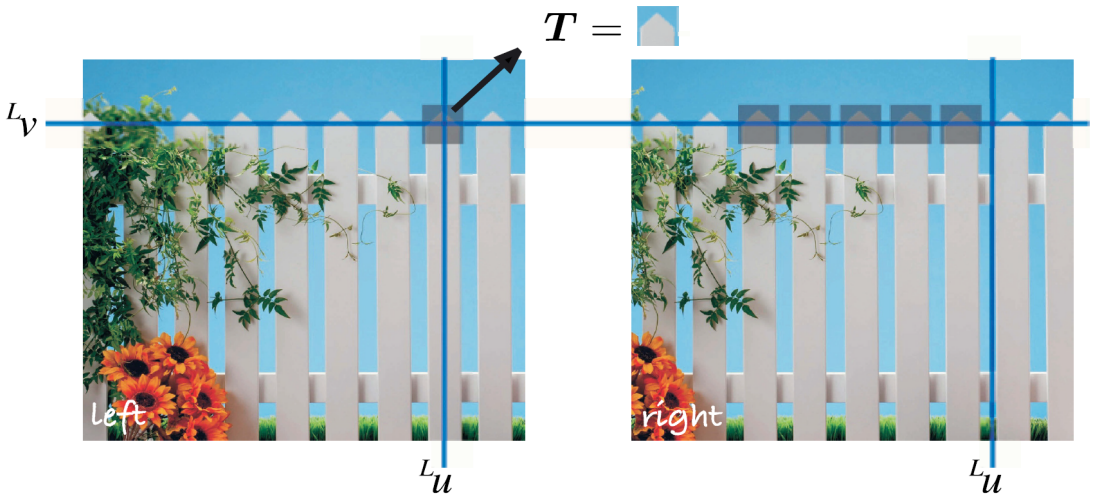


Figure 3.15: Failure of the Template matching approach. The template will match well at different disparities. This problem occurs in any scene with repeating patterns. [18]

Another significant challenge within this approach is *occlusion*. As demonstrated in Fig. 3.16, the presence of point 3 is exclusive to the left camera. Dense stereo matching will try to find a correspondent point in the right image, despite the absence of such a match. Consequently, the correspondence cost function records low values for every pixel within the right image. This situation is commonly referred to as a "weak peak" and is a frequent occurrence in real-world images, particularly at the boundaries of objects where rapid depth variations occur.

Reconstruction

The final step in the 3D reconstruction pipeline is recovering the world coordinates $\boldsymbol{P} = (X, Y, Z)$ for each pixel within the image, given the camera intrinsic and extrinsic calibration and the disparity map $d(u, v)$.

As anticipated in Eq. 3.33, in a parallel axis stereo camera rig (see Fig. 3.17) a

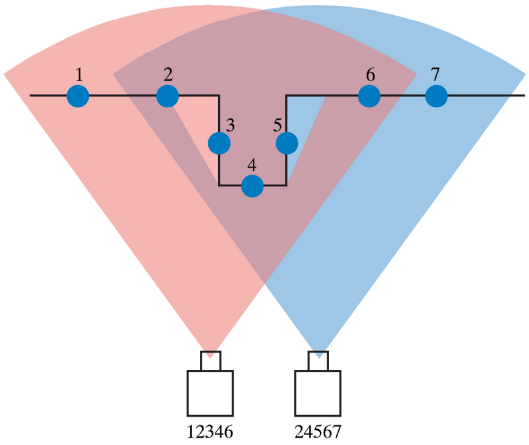


Figure 3.16: Occlusion in stereo vision. The fields of view of the two cameras are shown as colored sectors. Points 1 and 7 fall outside the overlapping view area and are seen by only one camera each. Point 5 is occluded from the left camera and point 3 is occluded from the right camera. The order of points seen by the cameras is given under each of the two. [18]

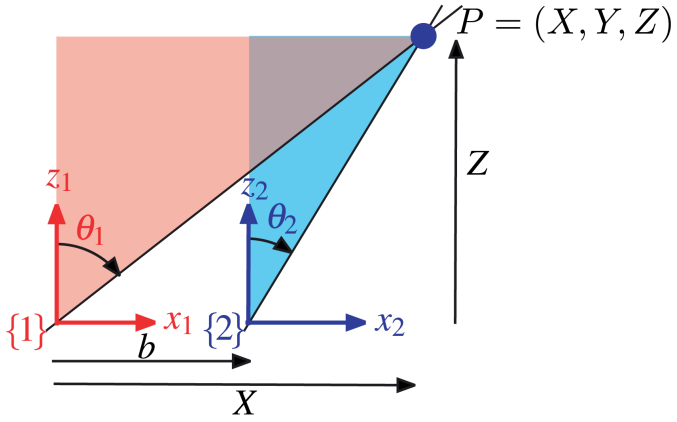


Figure 3.17: Stereo geometry for a parallel axis stereo camera rig. View of the XY plane, where the world RF is positioned on the first camera. b represents the baseline of the stereo pair. [18]

very simple inverse relation can be established between the depth Z and the disparity d. Considering the red and the blue triangles,

$$X = Z \tan\theta_1, \qquad X - b = Z \tan\theta_2 \tag{3.35}$$

where b is the baseline. The two angles can be expressed in terms of the horizontal coordinate u in the two images

$$\tan\theta_i = \frac{u^i - u_0}{f_w}, \quad i = L, R \tag{3.36}$$

Substituting Eq. 3.36 in Eq. 3.35, eliminating X, and solving for Z we obtain

$$Z = \frac{f_w b}{u^L - u^R} = \frac{f_w b}{d} = \frac{T_x}{d} \tag{3.37}$$

that corresponds to the previously anticipated relation. Usually, the product $f_w b$ is referred to as T_x and it is a distance $[mm]$. It is possible also to recover the X- and Y-coordinates given the pixel indexes (u, v)

$$X = \frac{b(u^L - u_0)}{d} \qquad Y = \frac{b(v^L - v_0)}{d} \tag{3.38}$$

It is essential to note that because the disparity $d = u^L - u^R$ represents the difference between two integer values, it is inherently an integer itself. Furthermore, as demonstrated previously, it must also be positive. Therefore, in the context of the *stereopsis* process, the disparity map $d(u, v)$ constitutes a grid of positive integer values, resulting in depth values being quantized. This phenomenon is more evident with high values of depth (low disparity), as shown in Fig. 3.18. A similar analysis can be applied to the X- and Y-coordinates, although, in these cases, they are directly proportional to a ratio between integers ($u^L - u_0$ and d), so they can assume a wider range of values.

Figure 3.18: Point-cloud generated using a simulated stereo camera in Gazebo environment. The quantized plane of the depth Z parallel to the XY plane can be easily seen. The higher the depth value, the higher the space between consecutive planes.

3.5 Point Clouds

Point Clouds, as well as other types of 3D data representation, serve as indispensable tools in CV applications, since they can capture the positional information of the objects. Therefore, this section will introduce fundamental operations that can be performed using a Point Cloud Data (PCD). Must be underlined that processing a PCD is a challenging task, due to the absence of connectivity information which leads to ambiguity about the surface information. [23]

Firstly, the outcome of the 3D reconstruction process can be archived in the form of a PCD, essentially constituting a collection of points. Each point within the cloud carries information about its spatial coordinates $\boldsymbol{P} = (X, Y, Z)$ relative to a specific RF (representing the 3D reconstruction of a pixel within the depth frame). Furthermore, when leveraging RGB images captured by the stereo camera, it becomes feasible to associate also color information with each point $\boldsymbol{P}$. In this scenario, the generated PCD takes the form of 6-tuples, denoted as (X, Y, Z, R, G, B), with each tuple corresponding to a distinct point/pixel from the image. 3D PCDs has some important characteristics:

- **Non-Euclidean data space**: Since PCDs do not have a *"gridded array structure"*[23], they can be considered as non-Euclidean representation. In particular, PCDs are *discrete manifolds* [24], since they are a globally non-euclidean data space, while locally they can appear Euclidean. It is the same phenomenon of the approximation of the Earth's surface: locally, from our perspective is flat and Euclidean, while globally it is a sphere.

- **Unstructured data space**: As mentioned before PCDs are not distributed over a regular grid [25]. It is interesting to notice that RGB-D images like the ones analyzed in Sec. 3.4 are, on the contrary, a structured dataset, since they are arranged on the pixels grid.

- **Unordered set**: PCDs are a set of points in $\mathbb{R}^3$ and each point is defined autonomously within this space [26] rather than from its position in an array. So, PCDs are *invariant to permutation* [27]: the order in which the points are processed does not change the resulting shape or set.

- **Invariant to rigid transformation**: PCDs preserve their geometric structure when rigid transformations are applied on the set [27], such as scaling, rotation, etc.

- **Irregularities or missing data**: PCDs generated by real sensors can be affected by multiple obstructions and irregularities that can affect the acquisition process. For instance, the density of the PCD can be non-uniform, with unevenly sampled zones, or completely missing zones. Moreover, this representation is subjected to a lot of noise [28].

Voxelization

It is possible to convert a PCD into a grid structure, which can be considered as the 3D analog of the pixel grid used in images. In this grid structure, each cell, or *voxel*,

contains information about the volumetric area it represents. This information can include whether the area is visible, occluded, or self-occluded, or more detailed attributes such as color, depending on the requirements of the application. Unlike PCDs, voxel grids do not explicitly encode their spatial position. Instead, their position is inferred based on their index coordinates in the 3D grid. This structured grid representation makes voxel grids a form of structured Euclidean data, distinguishing them from the unstructured nature of PCDs [23].

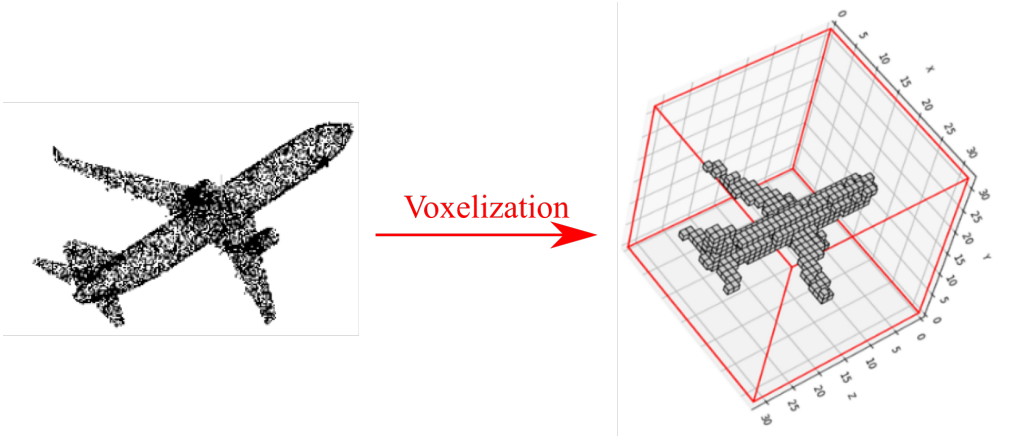


Figure 3.19: Example of voxelization of a PCD of an airplane to a $30 \times 30 \times 30$ volumetric occupancy grid. [25]

Voxel-based methods have gained interest in recent years, primarily due to their capability to extend well-established 2D grid operations to the 3D voxel grid. While these methods have demonstrated good performance, they are burdened by significant memory consumption issues stemming from the sparsity of voxels. The sparsity of voxels leads to inefficient computation when convolutions are performed over non-occupied regions, and it also imposes limitations on voxel resolution. Moreover, these limitations are in addition to the quantization artifacts introduced through the voxelization process [25].

3D Bounding boxes

In order to summarize information about an object (and so reduce memory occupation), *bounding boxes* are an effective tool. They efficiently summarize the volumetric occupancy of 3D objects, thereby providing a compact yet informative representation. As illustrated in Fig. 3.20, a PCD is enclosed by bounding boxes, which can be aligned with the principal moments of the object or the global fixed RF.

3.5.1 Clustering

Clustering algorithms for 3D spatial databases offer an attractive means of identifying classes within the data. Similar to the image segmentation discussed in Sec. 3.3, the

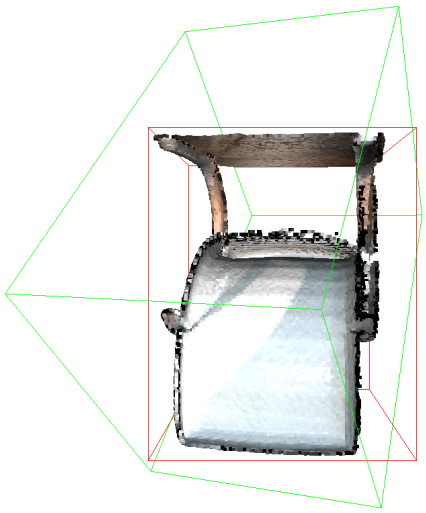


Figure 3.20: PCD with the corresponding bounding box aligned with the principal moments of the object (green) and its bounding box aligned with the axis of the RF (red). [29]

primary objective is to partition the input dataset into meaningful classes, facilitating the summarizing of information such as pose, shape, and more for each class.

In this chapter, the challenge of clustering is approached by leveraging the spatial properties of a PCD, specifically the 3D coordinates of each point in the input dataset. This stands in contrast to the image segmentation discussed in Sec. 3.3, where the color properties of pixels were predominantly utilized. Furthermore, the clustering problem is addressed through *unsupervised* approaches, where the algorithm's training phase is conducted without prior knowledge of the "correct" class for each point.

In general, clustering algorithms can be divided into two macro families:

- **Partitioning algorithms** know a priori the number of the clusters k, then divide the points of the dataset into the clusters. The algorithm can be summarized in two iterative steps:
  - A *data assignment step*, in which each data point is assigned to its nearest centroid, for instance based on the squared Euclidean distance (*k-means* clustering algorithm);
  - A *centroid update step* then recalculates optimal centroids based on the cluster assignments from the previous step.

  This iterative approach is guaranteed to converge to a result, but the result may be a local optimum, i.e., not necessarily the best possible outcome [30].
- **Hierarchical algorithms**, instead, create a hierarchical decomposition of the input dataset. This process can be visualized as a *dendrogram*, which is essentially a tree

that successively splits the dataset into smaller subsets until each subset contains only one object. Consequently, each node in the dendrogram represents a distinct cluster [31]. Notably, hierarchical algorithms do not require the number of clusters as input. However, a *termination condition* (e.g., a critical distance between all clusters) must be defined to exit the splitting loop.

In this work, as well as in the project related, I will use the Open3D [29] implementation of the DBSCAN algorithm [31]. This implementation employs a hierarchical algorithm to partition the input dataset into k meaningful clusters, along with a *noise* cluster that encompasses points not belonging to any cluster. The approach hinges on leveraging density information, defining a cluster as a high-density region in space that is well-separated from others. The low-density regions between clusters contain only noise points.

In particular, the algorithm starts with an arbitrary point p within the dataset and associates to it all point density-reachable from the point itself. Two parameters must be tuned for the effectiveness of the algorithm: the minimum number of points denoted as *MinPts*, which must be present in a neighborhood of radius ϵ around a point within a cluster.

(a) Initial point-cloud. **(b)** Point-cloud clusters

Figure 3.21: Point-cloud clusters founded using DBSCAN [31] algorithm implemented using Open3D[29] with a distance to neighbors in a cluster $\epsilon = 0.02$ $[m]$, and the minimum number of point to form a cluster $MinPts = 10$. In the figure, points that are not highlighted are considered as noise.

3.5.2 Acquisition technologies

Various sensors and technologies are employed to capture 3D data and create PCDs, each with its characteristics and suitability for specific applications.

One accessible technology is the RGB-D camera, capable of capturing both color (RGB) and depth information on a per-pixel basis in real-time. Depth frames in RGB-D cameras can be obtained through different technologies, such as:

- Stereoscopic cameras, which determine depth by comparing disparities between two spaced sensors, as presented previously in Sec. 3.4.

- Structured light cameras, which actively project infrared light patterns onto the scene and use a different perspective camera to analyze the visual distortion of the patterns for depth reconstruction, as represented in Fig. 3.22.
- Time-of-flight (ToF) sensors, which calculate depth based on the time it takes for signals emitted from an infrared light source to travel to the target object and return to an infrared sensor.

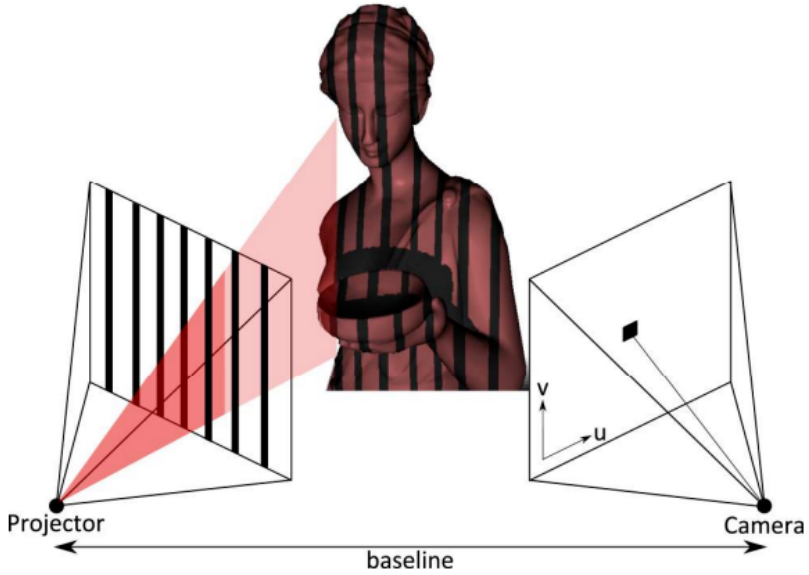


Figure 3.22: Principle of structured light based RGB-D camera. Depth information is extracted by analyzing the distortion in the projected pattern. [32]

Many methods in literature aim to reconstruct 3D geometry from RGB-D sensor data, making it possible to process also depth information alone and convert it into a PCD in real-time, ideal for various robotics applications.

Another category of sensors, such as Laser Imaging Detection and Rangings (LIDARs), is more reliable but expensive. LIDARs use ultraviolet and visible light signals to estimate depth information by measuring the time it takes for light to travel to an object and return to the sensor. These sensors are employed in performance-critical applications, and their working principles can be adapted for different scenarios. Traditional LIDAR scanners rotate mirrors to scatter laser bursts and achieve a 360° horizontal FOV. Recently, *Solid State* LIDARs without moving parts have been developed, offering a cost-effective and reliable alternative, but with a more limited FOV. Both 2D planar and 3D LIDARs are available, each suited for different use cases.

Chapter 4

Mobile robots

In this chapter, we explore the critical connection between a robot's perception of its environment and the subsequent actions it undertakes. Our focus narrows down to robots operating in two-dimensional spaces, particularly ground, wheeled robots navigating on flat terrains. This chapter will cover the following key topics:

- **Kinematic models:** We will explore the fundamental kinematic models that govern the motion of mobile robots. Starting with the constraints on their motion, we will construct basic kinematic models such as the unicycle and the bicycle, providing insights into how these models represent the motion of wheeled robots.
- **Autonomous navigation systems:** An overview of autonomous navigation systems, localization, and mapping will be presented. Understanding these components is essential for a robot to effectively navigate its environment and make informed decisions.
- **Model Predictive Control (MPC):** The chapter will introduce the concept of MPC, a strategy for computing control commands for rovers. We will explore how MPC utilizes a predictive model of the robot to optimize its trajectory over a specified time horizon. Moreover, we will discuss the Model Predictive Path Integral (MPPI) controller, an evolution of MPC, and its implementation in the ROS2 Nav2 package.

This foundational knowledge lays the groundwork for understanding how these robots effectively interact with and navigate their surroundings.

4.1 Kinematic models

Non-holonomic constraints

Wheeled vehicles encounter certain constraints that restrict their local mobility, although these constraints typically do not entirely preclude achieving arbitrary poses. Let's think about the parking of a car: it is not possible to move it sideways, but parallel parking is possible only with an appropriate sequence of maneuvers.

To illustrate this phenomenon, let's consider the example of a single steered wheel moving on a flat surface. Its *configuration*, i.e. position and orientation, is described by the vector $\boldsymbol{q} = [x \quad y \quad \theta]^T$, where (x, y) are the coordinate of the contact point between the wheel and the ground and θ represents the wheel orientation w.r.t $\boldsymbol{x}$ axis, as depicted in Fig. 4.1. Thus, the wheel's movement can be characterized by the function $\boldsymbol{q}(t)$.

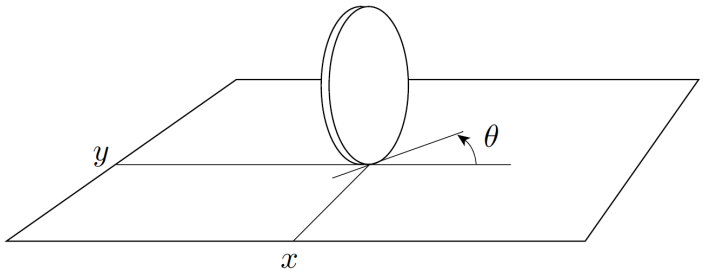


Figure 4.1: Single wheel rolling on a plane. [33]

In the case of *non-slipping* wheel motion, the velocity of the contact point should have no component in the direction perpendicular to the wheel's orientation, meaning the velocity vector should align with the wheel's orientation. Using the velocity vector $\vec{v} = (\dot{x}, \dot{y})$, this requirement, known as the *pure rolling constraint*, can be expressed as:

$$\frac{\dot{y}}{\dot{x}} = \frac{dy}{dt} \cdot \frac{dt}{dx} = \frac{dy}{dx} = \tan\theta \tag{4.1}$$

Such a constraint can be expressed in *Pfaffian form*, a linear expression in terms of generalized velocities $\dot{\boldsymbol{q}} = [\dot{x} \quad \dot{y} \quad \dot{\theta}]^T$ (as will be discussed later), as follows:

$$\begin{aligned} \frac{\dot{y}}{\dot{x}} = \tan\theta &= \frac{\sin\theta}{\cos\theta} \\ \dot{y}\cos\theta &= \dot{x}\sin\theta \\ \dot{x}\sin\theta - \dot{y}\cos\theta &= 0 \\ [\sin\theta \quad -\cos\theta \quad 0]\,\dot{\boldsymbol{q}} &= 0 \end{aligned} \tag{4.2}$$

This constraint is *non-holonomic*, since it does not imply any loss of accessibility in the wheel configuration space [34]. In other words, it's feasible to move the wheel from any initial configuration $\boldsymbol{q}_i$ to any final configuration $\boldsymbol{q}_f$ through a suitable sequence of motions without violating the pure rolling constraint.

In the general case, let $\boldsymbol{q} \in \mathbb{R}^n$ the vector generalized coordinates that describes the configuration of the robot, with the assumption that the robot's configuration space $\mathcal{C}$ (i.e., the set of all possible configurations) coincides with $\mathbb{R}^n$. The robot's motion is defined by the function $\boldsymbol{q}(t)$ and can be subject to constraints.

We classify constraints as *holonomic* (or *integrable*) if they can be written as:

$$h_i(\boldsymbol{q}) = 0, \qquad i = 1, \ldots, k < n \tag{4.3}$$

Holonomic constraints effectively reduce the dimension of the accessible configuration space to a subspace of $\mathcal{C}$ with a dimension of $n - k$.

Constraints that involve both the generalized coordinates and their derivatives (velocities) are called *kinematic* constraints and, in general, they are expressed as

$$a_i(\boldsymbol{q}, \dot{\boldsymbol{q}}) = 0, \qquad i = 1, \ldots, k < n \tag{4.4}$$

These kinematic constraints restrict the possible instantaneous motion of the system, constraining the set of generalized velocities that can be obtained in each configuration. Often, these constraints can be expressed in Pfaffian form, meaning they are linear in the generalized velocities, as:

$$\boldsymbol{a}_i^T(\boldsymbol{q})\dot{\boldsymbol{q}} = 0, \qquad i = 1, \ldots, k < n \tag{4.5}$$

or, in matrix form, as:

$$\boldsymbol{A}^T(\boldsymbol{q})\dot{\boldsymbol{q}} = 0 \tag{4.6}$$

If there are k holonomic constraints, this implies the existence of k kinematic constraints, as demonstrated by the following equivalence:

$$\frac{dh_i(\boldsymbol{q})}{dt} = \frac{dh_i(\boldsymbol{q})}{d\boldsymbol{q}}\dot{\boldsymbol{q}} = \boldsymbol{a}_i^T(\boldsymbol{q})\dot{\boldsymbol{q}} = 0, \qquad i = 1, \ldots, k \tag{4.7}$$

However, the reverse is not always true. Kinematic constraints can be reduced to holonomic constraints only when they are integrable. If not, they are called *non-holonomic* or *non-integrable*. Non-holonomic constraints do not imply any loss of accessibility in $\mathcal{C}$ (the number of generalized coordinates cannot be reduced), but they do constrain the velocities to a subspace with a dimension of $n - k$.

Derivation of the kinematics models

The matrix representation of the kinematic constraints in Eq. 4.6 makes it clear that the $n - k$ admissible generalized velocities can be found within the null space of $\boldsymbol{A}^T(\boldsymbol{q})$.

Denoting with $\{\boldsymbol{g}_1(\boldsymbol{q}), \ldots, \boldsymbol{g}_{n-k}(\boldsymbol{q})\}$ a base of the null space $\mathrm{N}(\boldsymbol{A}^T(\boldsymbol{q}))$, it is possible to define the admissible trajectories as the solution of the non-linear dynamical system

$$\dot{\boldsymbol{q}} = \sum_{j=1}^{m} \boldsymbol{g}_j(\boldsymbol{q})u_j = \boldsymbol{G}(\boldsymbol{q})\boldsymbol{u}, \qquad m = n - k \tag{4.8}$$

that represents the *kinematic model* of the constrained system. The trajectory $\boldsymbol{q}(t)$ results from integration and depends on the initial position.

Within this context, $\boldsymbol{u} \in \mathbb{R}^m$ represents the control inputs. The dimension of the generalized velocities is effectively reduced to $m = n - k$ due to the non-holonomic constraints. Additionally, it's essential to note that this system is *driftless*, meaning that a zero input results in zero velocity output. Lastly, it is important to recognize that the choice of the $\boldsymbol{g}_j(\boldsymbol{q})$ basis isn't unique. Consequently, the selection of the input vector $\boldsymbol{u}$ is not unique either. The components of $\boldsymbol{u}$ may have physical interpretations and can be linked to available control inputs, but this is not an exclusive requirement.

Unicycle model

The simplest model we can introduce is the unicycle, a vehicle with a single steerable wheel. Its configuration is described by $\boldsymbol{q} = [x \quad y \quad \theta]^T$ and it is subject to the pure rolling constraint $[\sin\theta \quad -\cos\theta \quad 0]\,\dot{\boldsymbol{q}} = 0$.

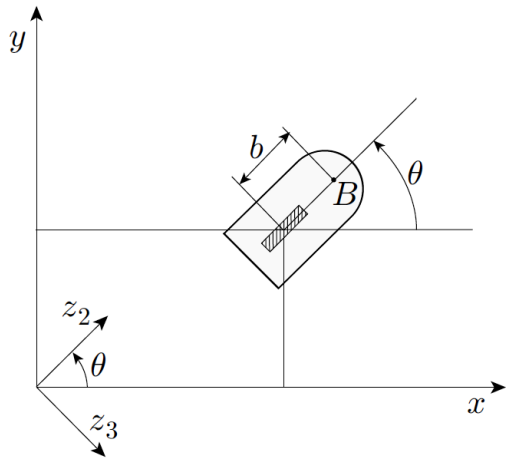


Figure 4.2: Unicycle model, a vehicle with a single steerable wheel. [33]

A base for the null space of the constraint matrix is:

$$\begin{aligned} \boldsymbol{g}_1(\boldsymbol{q}) &= [\cos\theta \quad \sin\theta \quad 0]^T \\ \boldsymbol{g}_2(\boldsymbol{q}) &= [0 \quad 0 \quad 1]^T \end{aligned} \tag{4.9}$$

Thus, we have:

$$\boldsymbol{G}(\boldsymbol{q}) = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \tag{4.10}$$

and, from Eq. 4.8, we can obtain the *unicycle kinematic model*:

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \boldsymbol{G}(\boldsymbol{q})\boldsymbol{u} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \omega \tag{4.11}$$

In this case the input $\boldsymbol{u} = [v, \omega]$ have a physical meaning and is related to straightforward control inputs: v is the driving velocity (modulus of the velocity of the contact point) and ω is the steering velocity.

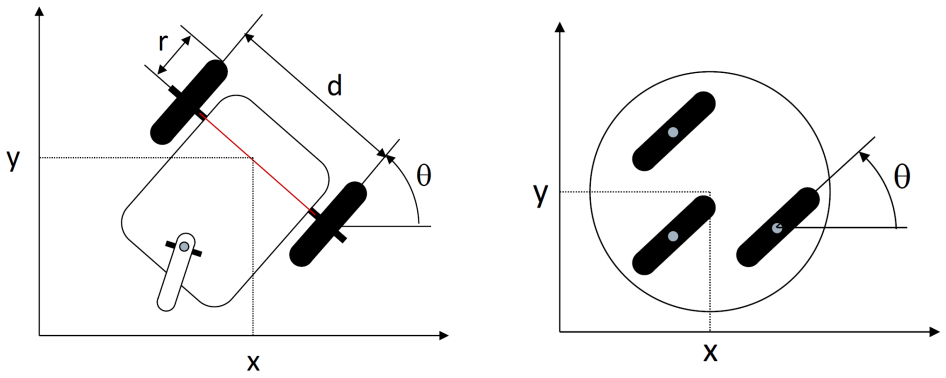
A unicycle would not be practically useful, due to its balancing problems. From a kinematic point of view, however, it is equivalent to more stable structures, such as:

- **Differential drive vehicles** represented in Fig. 4.3a. (x, y) are the coordinates of the middle point of the wheel axis, θ the vehicle heading. The physical inputs are

the rotation velocities of the two wheels, ω_R and ω_L, that are related to the input of the unicycle by

$$v = \frac{r(\omega_R + \omega_L)}{2} \qquad \omega = \frac{r(\omega_R - \omega_L)}{d} \tag{4.12}$$

- **Synchro dive vehicle** represented in Fig. 4.3b. The equivalence is straightforward, since both drive and steer velocity are common to the three wheels. Coordinates (x, y) can represent here any point of the chassis, while θ is the wheels heading (the chassis has constant heading).



(a) Differential drive vehicle **(b)** Synchro drive vehicle

Figure 4.3: Models equivalent to the unicycle from a kinematic point of view. The solid wheels are the active ones, while the white ones are passive. [34]

Bicycle model

A slightly more complex model is the *bicycle*, a vehicle with a steered front wheel and a fixed rear one, mounted at a distance L from the front one. A possible choice for the generalized coordinates is $\boldsymbol{q} = [x \quad y \quad \theta \quad \phi]^T$, where (x, y) are the coordinates of the contact point of the rear wheel, θ is the heading of the vehicle with respect to $\boldsymbol{x}$ axis, and ϕ is the steering angle of the front wheel, as shown in Fig. 4.4.

There are two pure rolling constraints, one for each wheel:

$$\begin{aligned} \dot{x_F} \sin(\theta + \phi) - \dot{y_F} \cos(\theta + \phi) = 0 \\ \dot{x} \sin\theta - \dot{y} \cos\theta = 0 \end{aligned} \tag{4.13}$$

where (x_F, y_F) are the coordinates of the contact point of the front wheel. We can rewrite the first constraint as a function only of the generalized coordinates $\boldsymbol{q}$, using the rigid body property,

$$\begin{aligned} x_F &= x + L\cos\theta \\ y_F &= y + L\sin\theta \end{aligned} \tag{4.14}$$

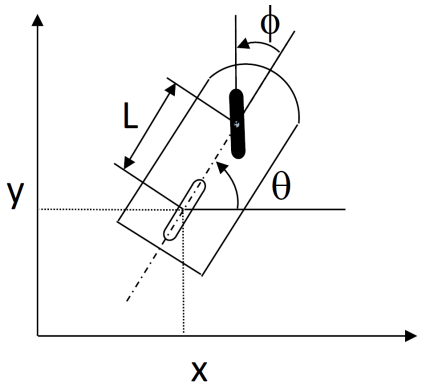


Figure 4.4: Bicycle model schematic. [33]

obtaining

$$\begin{aligned} \dot{x}\sin(\theta+\phi) - \dot{y}\cos(\theta+\phi) - L\dot{\theta}\cos\phi = 0 \\ \dot{x}\sin\theta - \dot{y}\cos\theta = 0 \end{aligned} \tag{4.15}$$

We can rewrite these constraints in the general matrix form presented previously in Eq. 4.6, where in this case

$$\boldsymbol{A}^T(\boldsymbol{q}) = \begin{bmatrix} \sin\theta & -\cos\theta & 0 & 0 \\ \sin(\theta+\phi) & -\cos(\theta+\phi) & -L\cos\theta & 0 \end{bmatrix} \tag{4.16}$$

The rank of $\boldsymbol{A}^T$ is 2 and, consequently, its null space has dimension $4-2=2$. A possible base for the null space of $\boldsymbol{A}^T$ is formed by the columns of

$$\boldsymbol{G}(\boldsymbol{q}) = \begin{bmatrix} \cos\theta\cos\phi & 0 \\ \sin\theta\cos\phi & 0 \\ \frac{\sin\phi}{L} & 0 \\ 0 & 1 \end{bmatrix} \tag{4.17}$$

With this choice the kinematic model obtained is

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \boldsymbol{G}(\boldsymbol{q})\boldsymbol{u} = \begin{bmatrix} \cos\theta\cos\phi \\ \sin\theta\cos\phi \\ \frac{\sin\phi}{L} \\ 0 \end{bmatrix} u_1 + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u_2 \tag{4.18}$$

where input $u_2 = \omega$ is the steering velocity of the front wheel. Instead, u_1 depends on the drive of the vehicle:

- If the bicycle has front drive, we have directly $u_1 = v$ where v is the driving velocity

of the front wheel. The resulting model is the following:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \cos\theta\cos\phi \\ \sin\theta\cos\phi \\ \frac{\sin\phi}{L} \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \omega \tag{4.19}$$

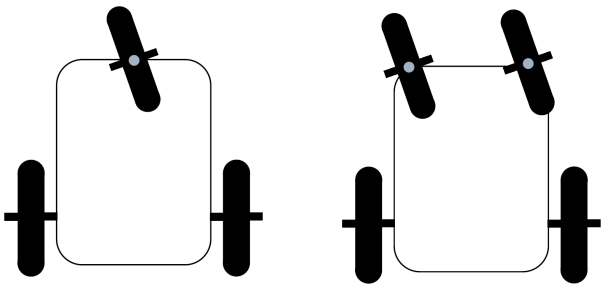
- If the vehicle has back drive, we can derive u_1 observing that the first two equations must coincide with those of the unicycle (Eq. 4.11), as the back wheel has the same generalized coordinates of a unicycle. This leads to set

$$u_1 = \frac{v}{\cos\phi} \tag{4.20}$$

where v is the drive velocity of the back wheel. The resulting kinematic model is

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ \frac{\tan\phi}{L} \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \omega \tag{4.21}$$

Also the bicycle presents balance problems in practice, but also in this case we have equivalent structures from the kinematic point of view, such as the *tricycle* and the *automobile* shown in Fig. 4.5. In both cases, we may have rear or front drive. The point (x, y) is the midpoint of the back wheel axis, θ represents the vehicle heading, and ϕ is the steering angle.



(a) Tricycle **(b)** Automobile

Figure 4.5: Models equivalent to the bicycle from a kinematic point of view. [34]

4.2 Navigation

For a rover to navigate effectively within its environment, it must tackle three fundamental challenges: trajectory planning, localization, and control. Trajectory planning involves determining the desired path the rover should follow, respecting constraints, and optimizing its motion. Localization, on the other hand, revolves around precisely determining the rover's position and orientation within its environment. Finally, control strategies are essential to ensure that the rover follows the planned trajectory and responds to changing conditions in real-time. These three challenges are essential components of implementing a feedback control system for the rover's autonomous movement.

Trajectory planning and control

The planning strategy aims to guide the rover from its initial configuration $\boldsymbol{q}_i$ to a target configuration $\boldsymbol{q}_f$. This task necessitates adherence to non-holonomic kinematic constraints and input constraints while often accommodating additional requirements such as obstacle avoidance, energy consumption reduction, and path curvature limitations.

This challenge can be decomposed into two distinctive phases:

1. **Path Selection:** This phase involves identifying the geometric path that the robot should ideally follow.

2. **Timing Law Definition:** It associates precise timing information to execute the path effectively.

Traditionally, these two phases are addressed by separate entities: the *planner* and *controller*. However, a more versatile approach entails the use of *optimal control* strategies, which allow to determine a control law that transfers the state of the dynamical system from $\boldsymbol{q}_i$ to $\boldsymbol{q}_f$. This approach seeks to minimize a relevant cost functional along the path, as exemplified by the Model Predictive Control (MPC) method detailed in Sec. 4.3.

Localization and mapping

The implementation of any feedback controller for trajectory tracking requires the knowledge of the configuration (position and orientation) at any time instant. Incremental encoders mounted on the wheel actuators provide measurements of wheel rotations, but do not directly provide the position and orientation of the vehicle w.r.t. to a fixed RF. This necessitates real-time localization procedures to estimate the robot's configuration.

The simplest yet less reliable solution is based on *odometry*. Given the robot's configuration $\boldsymbol{q}_k$ at time t_k, the next configuration $\boldsymbol{q}k + 1$ is determined by forward integration of the kinematic model using the applied inputs v_k and ω_k.

Irrespective of the integration algorithm, instead of using the nominal commands v_k and ω_k, it is preferable to reconstruct them starting from measurements, to overcome errors due to the non-idealities of actuators. However, this method is inherently susceptible to *drift* (error accumulation), due to several factors:

- Inaccuracy of the initial configuration $\boldsymbol{q}_0$.
- Wheel slippage.
- Measurement noise.
- Numerical errors introduced by the discrete integration.
- Inaccuracy in the kinematic parameters or in the model used for integration.

Active localization methods are more robust and reliable, since they exploit measurements from both odometry and "external" sensors, such as proximity, distance, vision, etc. Moreover, they may also perform comparisons with a map of the environment, given a priori or reconstructed by the robot during its motion. Estimates obtained through odometry are corrected using information from other sensors via probabilistic filters. These methods estimate, together with the robot configuration, the uncertainty associated with such an estimate, through a probability density function or *belief*.

The most commonly employed approaches include the *Extended Kalman Filter (EKF)*, where probability densities are supposed to be multivariate Gaussian, and *particle filters*. In the latter, probability densities are approximated by a set of weighted particles in a multidimensional space, where higher weights indicate a higher likelihood that a particle accurately represents the robot's actual pose.

The problem of *mapping* pertains to establishing a consistent representation of the environment in which the robot operates based on its pose knowledge. Different mapping methods can be used, according to the specific application of interest:

- Maps based on *landmarks* are stochastic maps that contain a probabilistic description of some salient elements (landmarks), such as doors, corners, etc.
- Maps based on *occupancy grid* consists of a cell grid, where each cell has associated the probability of being occupied.

While landmark-based maps offer a more compact representation with lower memory occupancy and can employ approaches similar to localization, occupancy grid-based maps provide more intuitive environment models that can be updated swiftly for navigation purposes.

When a robot navigates a completely unknown environment, both the localization and mapping problems must be solved concurrently. This scenario gives rise to the concept of *Simultaneous Localization And Mapping (SLAM)*. In this context, the robot constructs a map of the environment while simultaneously determining its own localization. An approach to tackle this problem could involve using the EKF in combination with a landmark-based map. Estimations consider an augmented state that includes both the robot's pose and the positions of landmarks within the map.

4.3 Model Predictive Control

In the dynamic and complex world of mobile robot navigation, the Model Predictive Control (MPC) strategy has emerged as a powerful and adaptive tool. As mobile robots traverse diverse environments, ranging from industrial facilities to outdoor terrains, the need for precise and responsive control becomes increasingly paramount. MPC, unlike traditional control methods, offers a forward-looking approach that considers the dynamic interplay of factors such as robot kinematics, sensor data, and environmental obstacles in real time. This enables mobile robots to make decisions that are not only reactive but also proactive, enhancing their ability to navigate safely and efficiently.

4.3.1 Introduction to the control strategy

Model Predictive Control (MPC) is a general and flexible approach to linear and non-linear system control. This solution allows us to deal directly with constraints, such as input constraints (e.g., limits in the actuators, maximum velocities, etc.), and state or output constraints (e.g., obstacles avoidance, etc.). Moreover, this solution manages systematically the trade-off performance/command effort. MPC is widely used in automotive systems, aerospace systems, chemical processes, robotics, biomedical devices, etc.

Physical limitations in actuator devices impose *hard constraints* on the control input $u(t)$, represented as saturation constraints. These constraints can be described as:

$$u^L \leq u(t) \leq u^U \qquad \forall t \geq 0. \tag{4.22}$$

where u^L and u^U are the lower and upper bounds, respectively. Usually, these bounds are symmetric with respect to the origin: given a maximum value u^M, then $u^L = -u^M$ and $u^U = u^M$. These saturation constraints can also be expressed as a non-linear static function of the control input as

$$u_s(t) = \begin{cases} u^L & \text{if } u(t) \leq u^L \\ u(t) & \text{if } u^L \leq u(t) \leq u^U \\ u^U & \text{if } u(t) \geq u^U \end{cases} \qquad \forall t \geq 0. \tag{4.23}$$

When input saturation is active, the feedback control system becomes non-linear. In this situation, the control system operates without feedback, and exceeding the input bounds can lead to unexpected behaviors like large overshoot, reduced performance, or, in the worst-case scenario, instability.

It is important to note that, in the presence of saturation, traditional stability analysis of the feedback system using linear systems tools (e.g., pole and eigenvalue analysis) is not applicable. Constraint satisfaction cannot be addressed directly by these tools and must be verified *a posteriori* through simulation. If the system does not meet the input requirements, one common approach is to slow down the transient response of the control system, typically by selecting dominant closed-loop poles with larger time constants. However, this strategy may result in an insufficient trade-off between transient

performance and constraint satisfaction. To prevent such performance degradation, the design procedure must explicitly account for input constraints.

Saturation constraints are effectively handled when the control input is obtained through a discrete-time *finite horizon* optimal design procedure, such as the Linear-Quadratic (LQ) *optimal control*. Given a linear, discrete-time system

$$\boldsymbol{x}(k+1) = \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}\boldsymbol{u}(k) \qquad \boldsymbol{x}(k) \in \mathbb{R}^n, \boldsymbol{u}(k) \in \mathbb{R}^{n_u} \tag{4.24}$$

the optimal input sequence computed at time k,

$$\boldsymbol{u}^\star(k|k) = [\boldsymbol{u}(k|k), \boldsymbol{u}(k+1|k), \dots, \boldsymbol{u}(k+H_p-1|k)]^T$$

can be found as

$$\boldsymbol{u}^\star(k|k) = \arg\min_{\boldsymbol{u}(k|k)} \mathcal{J}(\boldsymbol{x}(k|k), \boldsymbol{u}(k|k)) \tag{4.25}$$

The length H_p of the finite optimization horizon $[0, H_p - 1]$ is referred to as *prediction horizon*. The cost function $\mathcal{J}(\boldsymbol{x}(k|k), \boldsymbol{u}(k|k))$ is computed by evaluating the predicted state response up to the prediction horizon starting from the initial state $\boldsymbol{x}(k|k) = \boldsymbol{x}(k)$. The "$|k$" symbol underlines that the sequence is evaluated at time k.

To account for the presence of input saturation $\boldsymbol{u}^L \leq \boldsymbol{u}(k) \leq \boldsymbol{u}^U, \quad \forall k \geq 0$, the following constraints are added to the optimization problem.

$$\begin{aligned} &\boldsymbol{u}^L \leq \boldsymbol{u}(k|k) \leq \boldsymbol{u}^U \\ &\boldsymbol{u}^L \leq \boldsymbol{u}(k+1|k) \leq \boldsymbol{u}^U \\ &\vdots \\ &\boldsymbol{u}^L \leq \boldsymbol{u}(k+H_p-1|k) \leq \boldsymbol{u}^U \end{aligned} \tag{4.26}$$

where $\boldsymbol{u}^L$ and $\boldsymbol{u}^U$ are the upper and lower bounds respectively.

The Receding Horizon (RH) principle

Suppose that, at a time k, the optimal input signal $\boldsymbol{u}^\star(k : k+H_p|k)$ has been computed solving the corresponding optimization problem. In this scenario, $\boldsymbol{u}^\star(k : k+H_p|k)$ is an open-loop input: it depends on $\boldsymbol{x}(k)$, but not on $\boldsymbol{x}(\tilde{k}), \quad \tilde{k} > k$. If we apply the entire sequence for the entire time interval $[k, k+H_p-1]$, no feedback action is involved. This can lead to increased errors and disturbances, and a reduction in precision and adaptability. [35]

To overcome this problem, the Receding Horizon (RH) principle can be exploited. It can be defined by the following recursive procedure:

1. At time k:
   (a) Compute $\boldsymbol{u}^\star(k : k+H_p|k)$ by solving the corresponding optimization problem.
   (b) Apply only the first input value $\boldsymbol{u}(k) = \boldsymbol{u}^\star(k|k)$ and keep it constant for a sampling time interval.

2. Repeat steps 1a-1b for $k+1, k+2, \ldots$

If the model and the cost function are time-invariant, the RH principle implicitly defines a non-linear, time-invariant, static, state feedback control law [36] of the form

$$\boldsymbol{u}(k) = \mathcal{K}(\boldsymbol{x}(k)) \tag{4.27}$$

Unfortunately, the analytical expression of $\mathcal{K}(\boldsymbol{x}(k))$ cannot be computed in general. Moreover, this controller introduces non-linearity in the state equation of the controlled system (if not already present). Finally, the RH procedure can be also employed in the context of finite horizon LQ unconstrained control to obtain a feedback controller.

The MPC methodology

Constrained finite horizon optimal control and feedback Receding Horizon (RH) lead to the Model Predictive Control (MPC) methodology. In particular, MPC leverages a dynamical model of the plant to predict the future behavior of the variables of interest and compute an optimal control action.

"MPC is like playing chess" [36]. Similar to a chess player, the MPC controller selects a move by visualizing the game scenario and trying to anticipate the opponent's sequence of moves. If the opponent makes an unexpected move in the next turn, the controller must adjust its game plan to counteract this move (RH feedback).

The MPC strategy computes the solution online, making it computationally expensive. Additionally, it relies on a plant model for prediction. In practice, simplified plant models are often used to reduce computational complexity. Since the optimization problem solved by the controller is generally non-convex, an efficient numerical algorithm is required to find a solution (usually a local minimum in non-convex scenarios). For an example of such an optimizer, you can refer to Appendix C.

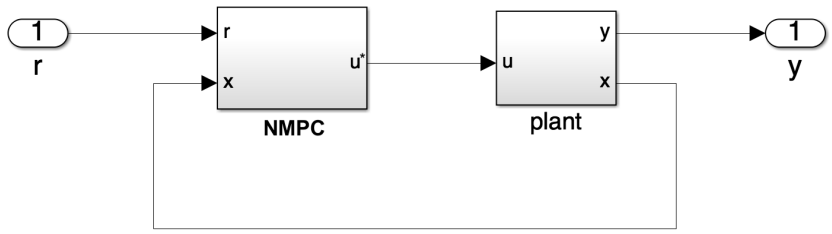


Figure 4.6: MPC schematic. In this case a Non-linear Model Predictive Control (NMPC) controller is used to follow the reference signal r.

Let consider a linear, discrete-time, Multiple-Input Multiple-Output (MIMO) system

$$\begin{aligned} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}\boldsymbol{u}(k) \\ \boldsymbol{y}(k+1) &= \boldsymbol{C}\boldsymbol{x}(k) + \boldsymbol{D}\boldsymbol{u}(k) \end{aligned} \tag{4.28}$$

where $\boldsymbol{x}(k) \in \mathbb{R}^n$ is the state, $\boldsymbol{u}(k) \in \mathbb{R}^{n_u}$ is the command input and $\boldsymbol{y}(k) \in \mathbb{R}^{n_y}$ is the output. Moreover, let's assume a zero-regulation problem, i.e. control all the states to zero. Then, the MPC control input is computed by solving at each sampling time k the problem

$$\begin{aligned} &\min_{\boldsymbol{u}(k|k)} \sum_{i=0}^{H_p-1} (\|\boldsymbol{x}(k+i|k)\|_Q^2 + \|\boldsymbol{u}(k+i|k)\|_R^2) + \|\boldsymbol{x}(k+H_p|k)\|_S^2 \\ &\text{s.t.} \begin{cases} \boldsymbol{x}(k+1) = \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}\boldsymbol{u}(k) \\ \boldsymbol{u}^L \leq \boldsymbol{u}(k+i|k) \leq \boldsymbol{u}^U \qquad i = 0, 1, \ldots, H_p - 1 \end{cases} \end{aligned} \tag{4.29}$$

and, according to the RH principle, by applying the first element of the minimizer.

$$\boldsymbol{u}(k) = \boldsymbol{u}^\star(k|k) \tag{4.30}$$

It is important to notice that in Eq. 4.29 $\|*\|_W$ represents the weighted vector norm. Here, Q is the matrix of state cost weights, R accounts for the input cost and allows us to manage the trade-off between performances and command activity, while S weights the terminal cost.

State and output constraints can be added to the problem to obtain particular performances. Moreover, it is important to notice that a longer prediction horizon provides more degrees of freedom within the optimization problem but simultaneously increases its complexity. To mitigate this phenomenon, various techniques, such as *variable blocking*, can be employed. Variable blocking involves grouping control moves to hold their values constant for multiple prediction steps, effectively reducing the number of optimization variables.

In case of a non-linear system

$$\begin{aligned} \boldsymbol{x}(k+1) &= f(\boldsymbol{x}(k), \boldsymbol{u}(k)) \\ \boldsymbol{y}(k+1) &= h(\boldsymbol{x}(k), \boldsymbol{u}(k)) \end{aligned} \tag{4.31}$$

the NMPC control input is computed, in an analog way, by solving at each sampling time k the problem

$$\begin{aligned} &\min_{\boldsymbol{u}(k|k)} \sum_{i=0}^{H_p-1} (\|\boldsymbol{x}(k+i|k)\|_Q^2 + \|\boldsymbol{u}(k+i|k)\|_R^2) + \|\boldsymbol{x}(k+H_p|k)\|_S^2 \\ &\text{s.t.} \begin{cases} \boldsymbol{x}(k+1) = f(\boldsymbol{x}(k), \boldsymbol{u}(k)) \\ \boldsymbol{u}^L \leq \boldsymbol{u}(k+i|k) \leq \boldsymbol{u}^U \qquad i = 0, 1, \ldots, H_p - 1 \end{cases} \end{aligned} \tag{4.32}$$

and, as in the linear case, by applying the first element of the minimizer.

Conceptual issues

MPC is a powerful technique in the realm of control systems, offering a general and flexible approach for managing complex Multiple-Input Multiple-Output (MIMO) systems. One of its key strengths lies in its intuitive formulation, which relies on optimality concepts

to craft control strategies. MPC also excels in accounting for constraints and addressing input saturation, even when these constraints are subject to variations over time. This method efficiently navigates the delicate trade-off between system performance and input activity, all while providing optimal trajectories over finite time intervals. Remarkably, MPC achieves a unified computation of both optimal trajectories and control laws [35].

However, the online computational cost of MPC can be high, which may limit its real-time applications. Additionally, MPC encounters challenges when dealing with unstable zero dynamics, as do most control methods. Moreover, since the feedback control action is realized through the RH principle via the solution to a finite horizon constrained optimization problem [36], two issues arise:

- *Feasibility* of the optimization problem for every point of the state space, i.e., for every sampling time. In particular fulfillment of state/output constraints is not guaranteed in general.
- *Stability* of the closed loop system.

Only in linear MPC with convex constraints, the optimization problem is convex, so it is guaranteed that numerical algorithm converges to the global optimum, while the non-linear formulation can find only local optima in general. Moreover, in contrast to linear MPC, which necessitates linearization of system dynamics, Non-linear Model Predictive Control (NMPC) directly employs non-linear models, making it particularly suited for systems with complex behavior. While linear MPC demands complex constraint convexification for non-convex constraints, NMPC easily accommodates simple inequalities to define constraints. Its flexibility also allows for seamless integration with other techniques, such as Artificial Potential Fields, opening doors to innovative control strategies.

Up to this point, full state feedback has been assumed in MPC design. However, in many cases, the use of a state observer is required to estimate unmeasured states and filter out noise. State estimation can be obtained using either an asymptotic state observer or a Kalman filter. In the presence of measurement noise and process disturbance, asymptotic estimation is not feasible, and an error exists in the estimated state. To accommodate this state estimation error, constraint tightening can be incorporated into MPC design. Nevertheless, this approach results in more complex optimization problems.

4.3.2 Model Predictive Path Integral Controller

The optimal control approach, which seeks to determine an optimal sequence of controls (or a control law) by considering a given cost function and the system dynamics, seamlessly integrates trajectory planning and execution. However, it's important to note that for systems characterized by nonlinear dynamics and non-convex cost functions, solving the optimal control problem poses substantial computational challenges.

In this section, I will introduce the Model Predictive Path Integral (MPPI) controller proposed in [37]. It is a MPC algorithm based on the path integral control framework. By merging the strengths of hierarchical methods (first planning a trajectory and then applying a simple feedback controller) and optimal control paradigms, it avoids dividing

the problem into separate planning and execution phases. This unified approach simplifies problem formulation and delivers optimal behavior while considering the system dynamics.

One of the remarkable features of this algorithm is its capability to accommodate objective functions that are neither required to be convex nor differentiable, offering enhanced flexibility. The controller presented in [37] has been effectively implemented in the `Nav2`[1] platform designed for ROS2.

The Model Predictive Path Integral (MPPI) algorithm is a variant of MPC designed to compute control velocity for a robot through an iterative approach. Using the previous time step's best control solution and the robot's current state, a set of randomly sampled perturbations from a Gaussian distribution are applied. These noised controls are forward simulated to generate a set of trajectories within the robot's motion model. Subsequently, these generated trajectories are evaluated using a predefined set of cost functions to identify the optimal trajectory within the batch. A `soft max` function is employed to determine the best control based on the trajectory scores. This process iterates multiple times until a converged solution is achieved.

The iterative update law discovered through this research can be effectively applied within an MPC framework. In this context, optimization is performed dynamically, with trajectory optimization taking place before executing a single control input, followed by re-optimization at the following time step. Given that the path integral control provides a formula for optimizing the entire sequence of controls, rather than just the current time instant, the unexecuted part of the control sequence can be re-used to initialize subsequent optimizations. This proves critical for the algorithm's performance since, for a complex system operating at a reasonable control frequency, only a limited number of iterations can be performed within each time step.

It's important to note that the accurate positioning of the rover is essential for the effective implementation of this solution. To achieve this, localization methods and sensors such as IMU and GPS must be employed.

The MPPI controller has been subjected to rigorous testing in a practical setting, specifically, with an autonomous off-road rally car model (see Fig. 4.7). Researchers employed the controller to address the challenge of navigating an approximately ellipsoidal racetrack, treating it as a finite-horizon optimal control problem.

The comprehensive cost function designed for this task consists of four terms:

- A cost for staying on the track. Moreover, if a given trajectory leaves the track, then the dynamics are set to zero and the car remains in its current location for the rest of the simulation.
- A cost for achieving a desired velocity.
- A control cost, in order to deal with the performance/command effort trade-off.
- A cost on the side-slip angle introduced to enhance the vehicle's stability.

[1] `https://navigation.ros.org/index.html`

Figure 4.7: Time-lapse video of the cornering maneuver of the Auto-Rally car. [37]

In the challenging rally scenario, the MPPI controller demonstrated its real-world applicability by effectively addressing the intricate dynamics of off-road racing.

Chapter 5

The proposed navigation system

This thesis proposes a position-agnostic system, that not only avoid the issues related to integration and localization systems but also excels in scenarios where traditional methods, such as GPS, fail to achieve the desired precision, often due to unfavorable weather conditions or line-of-sight obstructions.

The controller takes in input a Point Cloud Data (PCD) (data representation presented in Sec. 3.5) message, and it must compute a velocity `Twist` command as output (in particular a linear velocity v_x and an angular velocity ω_z), that the rover will use to move. The input PCD can be derived from different sources, including RGBD cameras or 3D LIDAR sensors. Alternatively, RGB images and depth frames can be used as separate inputs to generate the PCD through the 3D reconstruction process introduced in Sec. 3.4. The latter is the more computationally expensive one, since it requires reconstructing the scene inside the controller process, while using a LIDAR should provide the best result, since this technology produces a 360° uniform, light, and precise PCD. However, since RGBD cameras are very general, reusable, and quite cheap sensors, and since these devices (such as the one presented in Sec. 6.1.2) compute and provide directly the PCD in their FOV, the most tested solution is the one exploiting as input the PCD generated from an RGBD camera.

So the controller should be able to process in real-time the input PCD and for each one compute the corresponding velocity output. Since the rover is designed to navigate through rows in agricultural environments where high-speed travel is unnecessary, a reasonable minimum control frequency for the system is $15Hz$.

As introduced in Sec. 1, the primary objective of the control system is row navigation, ensuring effective obstacle avoidance. The rover's versatility is enhanced to accommodate various crop types and row spacing, from small-row crops where it operates within inter-row spaces to larger-row crops, where it adheres to predefined lanes, such as the right half of the row space. The controller must exhibit robustness in varying crop density, height, and other environmental factors, even when dealing with high canopies or rough terrains.

Furthermore, the system is designed to recognize and approach specific objects, such as

fruit boxes. Once a target object is identified, the system adapts its navigation to approach the object, pausing momentarily when it reaches the target. This momentary stop serves to showcase the correctness of the maneuver and, in the future, may be utilized to perform specific tasks related to the target. Following the brief pause, the system resumes its row traversal until it reaches the end of the row, demonstrating a seamless integration of targeted object recognition into the overall navigation strategy.

This comprehensive system aims to address the complexity of agricultural environments and enhance the autonomous capabilities of agricultural rovers.

5.1 The computation graph

The schematic of the proposed system is shown in Fig. 5.1.

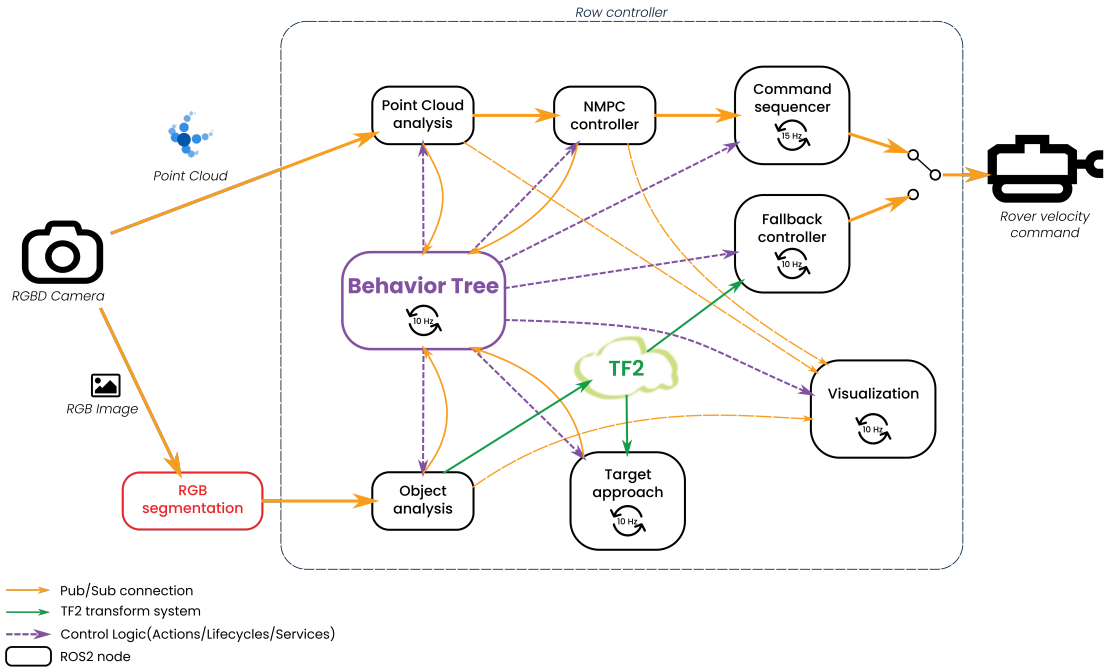


Figure 5.1: Computation graph of the proposed solution. The thick lines correspond to the principal data flows.

The RGBD camera data undergo analysis by two main subsystems:

- The **Point Cloud Analysis** node extracts information and computes two lines delimiting intra-row space. The subsequent **NMPC Controller** node uses a Non-linear Model Predictive Control (NMPC) strategy to compute the control sequence, minimizing input variation, distance from the lane center, misalignment with row direction, and maximizing distance traveled.

- The object recognition pipeline starts with the **RGB Segmentation** node. It is important to note that the scope of this work does not encompass the comprehensive development of target recognition mechanisms. Instead, a basic color filter is employed as a demonstration to identify the target and validate the approaching and recovery maneuvers. Then the **Object analysis** node clusters the segmented points into objects, and then computes their position in space. Finally, the **Target approach** validates target reaching.

Moreover, a **Fallback Controller** node is added to the system to manage faulting situations in which the rover rotates toward a row crop and does not see the free space in front of him. This node is a very simple controller that aligns and moves back the rover to the last correctly computed center of the intra-row space. This objective is done with a very simple proportional controller, where the feedback is represented by the odometry of the rover managed in ROS2 (see Appendix. D) by the TF2 subsystem. First, the controller moves back the rover toward the center of the free space, then it rotates it in the direction of the row, to have the free space in its FOV.

Finally, a **Visualization** node can be optionally employed to visualize in real-time the vision analysis and the results of the NMPC objective minimization, in particular the predicted trajectory. Moreover, if a target is recognized in the current frame a marker is added to the figure. This node is essential to debug and check the correct behavior of the system.

The system is orchestrated by a **Behavior Tree**, overseeing high-level logic, mission switches, start/stop commands, failure detection, and initiating fallback procedures.

5.2 The control algorithm

5.2.1 The vision algorithm for row detection

The **PCD Analysis** node is responsible for detecting the row in the input PCD. This node can be utilized with either an input RGBD image (reconstructing the PCD using the pipeline illustrated in Sec. 3.4) or a PCD, and it associates a callback function to process each incoming message. The output of this procedure includes the array of obstacle points and two lines representing the borders of the row, where each line is defined by two real numbers $a_i, b_i \in \mathbb{R} : y = a_i x + b_i, i \in [l, r]$.

Let's consider the whole procedure from the PCD to the extraction of obstacles and the lines.

The first part consists of reducing the input PCD to a set of 2D points:

1. The PCD is scaled to meters if it's not already in that unit.
2. It undergoes a transformation from the camera RF to the rover RF.
3. Down-sampling is performed using voxelization (refer to Sec. 3.5) at a specified resolution (e.g., 5 cm).
4. The number of points at this moment is recorded for subsequent reference.

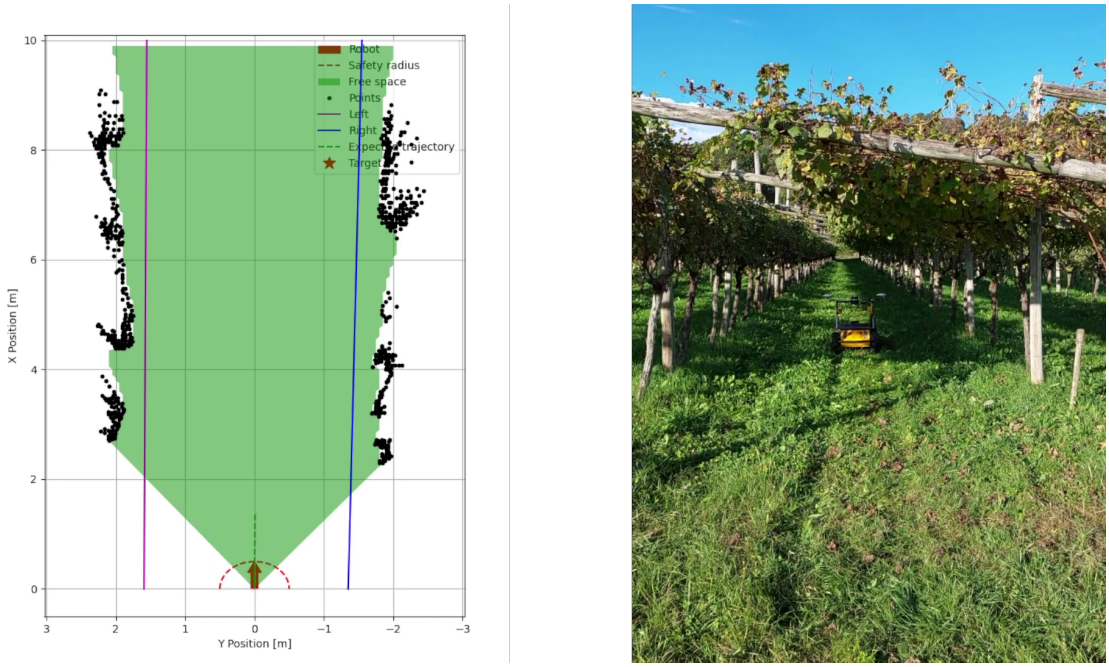


Figure 5.2: Example of the graph provided by the Visualization node on the left with its corresponding position in the vineyard on the right. The PCD computed by the camera mounted on the rover is analyzed and a step of the NMPC pipeline is performed. All the results are shown in real time on the graph on the left.

5. Cropping of the PCD is done to eliminate points that are too far and lack sufficient precision, as well as points in the sky and on the ground. A minimum height threshold is set to ensure the removal of ground points even in cases where the rover is not perfectly parallel to the ground plane due to rough terrains.

6. Statistical outlier removal is applied to filter the PCD.

7. If the current number of points is less than a given specific percentage of the number of points computed at (4), the row is considered empty. Otherwise, the z-coordinate is removed to obtain a set of 2D points.

Continuing from the 2D points array, the subsequent steps of the pipeline are as follows:

1. A grid map is generated at a specified resolution, matching the one used in the down-sampling process.

2. The 2D grid is illuminated from the rover's perspective (and hence the camera), shadowing all the occupied areas (considering space behind an occupied cell as also occupied).

3. A morphological filtering process is applied to the binary grid to eliminate noise and fill gaps between occupied zones.

4. The contours of the occupied zones are computed. Subsequent analysis depends on the number of distinct zones detected:
   - If only one contour is detected, the system first checks for an evident hole in the occupied zone. If present, indicating that at the end of the depth range, some points have connected the left and right crops, the points are divided into two clusters using this visible, far point as a separator. If no such hole is evident, information from the past frame is used. For example, if in the previous frame, only the right row was visible, then in the subsequent frame, the only area present should again be on the right side. This information is utilized to determine whether the cluster represents the left or right side. If previous frames are not available, the main trend in the cluster of points is used to decide whether this cluster is the left or right one.
   - If two contours are detected, it is a straightforward case where the two areas represent the left and right row crops, delineating the space.
   - If multiple contours are detected, the two extreme zones are associated with the left and right borders. The remaining contours are iteratively united with the closest one.
5. From the left and right clusters of points, the border facing the row is selected.

Continuing from the arrays of points representing the left and/or right border of the row (in meters), a linear weighted fit is applied to detect an initial version of the two straight polynomials:

- If both borders are represented by a sufficiently large number of unique points (greater than a given threshold, e.g., 5):
  1. Compute the median width between the left and right points. If this value is smaller than a given threshold, an error is raised since the computed row is too narrow.
  2. Update the current estimate of the width of the row:
  $$\hat{w}_{k+1} = (1 - K)\hat{w}_{k+1} + K \cdot median(w) \tag{5.1}$$
  where K represents the weight of the novelty update.
  3. Compute the two linear polynomials starting from the two arrays of points using a linear weighted fit. The weight is the sum of a term that penalizes more distant points and a term that penalizes points with a distance from the other side very different from the estimated mean value (representing points where there may be an obstacle or a hole in the crops).
- Use the linear fit with the distance weight presented earlier to compute the right line coefficients. The left ones are computed using the mean width value of the row computed at past time steps as the distance between the two parallel lines:
$$\begin{aligned} a_l &= a_r \\ b_l &= a_l + w_{avg}\sqrt{1 + a_l^2} \end{aligned} \tag{5.2}$$

- A similar approach is used in the case of left border points only:

$$\begin{aligned} a_r &= a_l \\ b_r &= a_l - w_{avg}\sqrt{1+a_r^2} \end{aligned} \tag{5.3}$$

To conclude and refine the results, several closure operations are performed:

1. A safe margin is added to the border of the row. Recalling that we are dealing with $a_i, b_i \in \mathbb{R} : y = a_i x + b_i, i \in [l, r]$, we modified the offset of the two lines to move them away from the crops, but the angular coefficient remains the same to maintain the direction:

$$\begin{aligned} a'_l &= a_l \\ b'_l &= b_l - R_m\sqrt{1+a_l^2} \\ a'_r &= a_r \\ b'_r &= b_r + R_m\sqrt{1+a_r^2} \end{aligned} \tag{5.4}$$

   where R_m represent the margin. The distance between the two parallel lines (before and after this operation) is exactly equal to R_m. The rover RF has the y-axis pointing to the left, so the right border is moved more to the left ($+R_m\sqrt{1+a_r^2}$ in Eq. 5.4), while the left one is moved more to the right ($-R_m\sqrt{1+a_r^2}$ in Eq. 5.4).

2. Starting from $a'_i, b'_i \in \mathbb{R} : y = a'_i x + b'_i, i \in [l, r]$ calculated at the previous step, the lane lines are adjusted:

   - If the trajectory is set to *"middle"*, the rover can use all the space in the intra-row space, so, no operation is needed, and the process continues to the next points.
   - If the trajectory is set to *"right"*, only the right half-space can be used, so:

$$\begin{aligned} a''_l &= \frac{a'_l + a'_r}{2} \\ b''_l &= \frac{b'_l + b'_r}{2} \end{aligned} \tag{5.5}$$

   - A specular operation is performed in the case of the *"left"* lane:

$$\begin{aligned} a''_r &= \frac{a'_l + a'_r}{2} \\ b''_r &= \frac{b'_l + b'_r}{2} \end{aligned} \tag{5.6}$$

3. Finally, an error is raised if one of the two lines is too (given a predefined maximum angle) perpendicular to the $x-$axis, i.e., the direction of motion of the rover. This means that if the rover turns to face the crops, an error is raised, and the subsequent fallback procedure is initiated to realign the rover with row direction.

As a slightly alternative solution, exploiting the segmentation Convolutional Neural Network (CNN) used by [10] to recognize the row is possible. With this approach, the input RGBD image is segmented, reconstructing only the points associated with the crops. Subsequently, the PCD analysis is performed exclusively on the crops, and the line delimiting the row is computed on the filtered PCD. In parallel to this operation, the obstacle points are computed from the complete PCD, as obstacles can also differ from the crops (e.g., boxes, people, etc.). This method allows for a more targeted and efficient analysis of the row-specific points, potentially improving the accuracy and performance of the system.

5.2.2 The NMPC controller

To implement the NMPC controller, the DO-MPC library [38] was chosen for its versatility. Subsequently, a customized kinematic model and cost function were meticulously tailored to address the specific requirements and characteristics of the rover's navigation scenario. This involved carefully calibrating the model parameters and formulating the cost function terms, as well as the problem constraints.

The inputs of the NMPC controller are the points representing the obstacles and the two first-order polynomials representing the two straight lines delimiting the lane, expressed in the robot RF. Each line is represented by two real numbers $a_i, b_i \in \mathbb{R} : y = a_i x + b_i, i \in [l, r]$.

Kinematic model

The NMPC approach employed in this project requires a plant model to predict future states. For this purpose, a modified version of the Unicycle model (introduced in Section 4.1), was selected. In particular, quaternions, discussed deeply in Appendix B, are used for the representation of the state variable $x_3 = \theta$ (representing the orientation angle). This adjustment was motivated by the periodicity of the angle expressed in radians, where multiple distinct states can represent the same angle.

In this modified model, the angle θ around the Z axis was substituted with the corresponding quaternion $\mathfrak{q} = (\cos(\theta/2), 0, 0, sin(\theta/2)$. Leveraging quaternion kinematics, its derivative $\dot{\mathfrak{q}}$ was then employed to describe the rover's orientation in relation to the input angular velocity ω_z. This approach enhances the representation of orientation, mitigating the challenges associated with the periodic nature of the angle variable and facilitating more robust predictions of future states within the NMPC framework.

Starting from the quaternion dynamics

$$\begin{aligned} \dot{\mathfrak{q}} &= \frac{1}{2}\mathbf{Q}\boldsymbol{\omega} \\ &= \begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{bmatrix} \cdot \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \end{aligned} \tag{5.7}$$

and recalling that we are considering only rotations around Z axis (rover moving on a plane), $q_1 = q_2 = 0$ and $\omega_x = \omega_y = 0$, then:

$$\dot{\mathfrak{q}} = \frac{1}{2}\begin{bmatrix} 0 & 0 & -q_3 \\ q_0 & -q_3 & 0 \\ q_3 & q_0 & 0 \\ 0 & 0 & q_0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ \omega_z \end{bmatrix} = \begin{bmatrix} -\omega_z \frac{q_3}{2} \\ 0 \\ 0 \\ \omega_z \frac{q_0}{2} \end{bmatrix} \tag{5.8}$$

$\dot{q_1} = \dot{q_2} = 0$ for every possible orientation so we can eliminate these two components. So, we can define two new states for substituting the angle θ, $x_3 = q_0$ and $x_4 = q_3$.

Moreover, to complete the model, $\cos\theta$ and $\sin\theta$ must be expressed as a function of the new states x_3 and x_4.

$$\begin{aligned} \cos\theta &= \cos\left(\frac{\theta}{2} + \frac{\theta}{2}\right) = \left(\cos\frac{\theta}{2}\right)^2 - \left(\sin\frac{\theta}{2}\right)^2 = q_0^2 - q_3^2 = x_3^2 - x_4^2 \\ \sin\theta &= \sin\left(\frac{\theta}{2} + \frac{\theta}{2}\right) = 2\cos\frac{\theta}{2} sin\frac{\theta}{2} = 2q_0q_3 = 2x_3x_4 \end{aligned} \tag{5.9}$$

To summarize, the kinematic model of the unicycle

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} \tag{5.10}$$

has been modified to

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \begin{bmatrix} v(x_3^2 - x_4^2) \\ v(2x_3x_4) \\ -\omega\frac{x_4}{2} \\ \omega\frac{x_3}{2} \end{bmatrix} \tag{5.11}$$

where $x_1 = x, x_2 = y, x_3 = \cos\frac{\theta}{2}, x_4 = \sin\frac{\theta}{2}$. The inverse relations are

$$x = x_1, \quad y = x_2, \quad \theta = 2\arctan\frac{x_4}{x_3}$$

Constraints

Firstly, input saturation constraints were incorporated into the NMPC minimization problem, allowing for the specification of maximum linear and angular velocities as parameters before the system's initiation.

In addition, non-linear constraints of the form $d(\boldsymbol{x}, \boldsymbol{o}^i) \geq R$ were integrated to ensure obstacle avoidance, where $d(\cdot)$ represents the Euclidean distance between the rover pose $\boldsymbol{x}$ and the i-th obstacle $\boldsymbol{o}^i$. The parameter R represents a predetermined safe distance between the rover and an obstacle point. By eliminating the squared root (due to domain issues with its derivative) and expressing the constraint in standard form $g(\dots) \leq 0$, the resulting formulation is:

$$-(x_1 - o_1^i)^2 - (x_2 - o_2^i)^2 + R^2 \leq 0 \tag{5.12}$$

This constraint must hold for each time step $t_k = 1 \dots T_H$ and for every obstacle point, providing a robust mechanism for obstacle avoidance throughout the prediction horizon.

Objective function

The core of the NMPC formulation lies in defining an objective function, which is an optimization problem represented as follows:

$$C = \sum_{k=0}^{n-1} (\underbrace{l\left(\boldsymbol{x}_k, \boldsymbol{u}_k, p\right)}_{\text{lagrange term}} + \underbrace{\Delta \boldsymbol{u}_k^T R \Delta \boldsymbol{u}_k}_{\text{r-term}}) + \underbrace{m\left(\boldsymbol{x}_n\right)}_{\text{meyer term}} \tag{5.13}$$

One key element of the objective function is the penalty for control inputs, which can be utilized to smooth the obtained optimal solution and serve as a crucial tuning parameter. A quadratic penalty on changes is added:

$$\Delta \boldsymbol{u}_k = \boldsymbol{u}_k - \boldsymbol{u}_{k-1} \tag{5.14}$$

The DO-MPC library automatically provides the solver with the previous solution of $\boldsymbol{u}_{k-1}$ for $\Delta \boldsymbol{u}_0$. Two parameters, K_{lin} and K_{ang}, are introduced as the diagonal elements of the matrix R for tuning this quadratic penalty on the two control inputs of the system (linear velocity and angular velocity, respectively).

Additionally, the objective function is designed to maximize the distance traveled by the rover in the prediction horizon time interval. So, recalling that $\max f = \min -f$, the terminal (or *meyer*) term is set as follows

$$m\left(\boldsymbol{x}\right) = -K_{travel} \frac{x_1 + a_{avg} \cdot x_2}{\sqrt{1 + a_{avg}^2}} \tag{5.15}$$

Here, K_{travel} represents the parameter for weighting this term, $a_{avg} = (a_l + a_r)/2$ is the angular coefficient of the line in the middle of the row, and x_1, x_2 are the coordinates of the rover in plane at the horizon $t_k = T_H$. The distance traveled by the row is projected onto the middle line to weigh only the distance traveled in the direction of the row.

For the *lagrange* term, which is evaluated and summed at each time step until the prediction horizon, two main contributions are defined: one for maintaining a central trajectory with respect to the lane and one for minimizing misalignment from the row direction.

$$l\left(\boldsymbol{x}_k, \boldsymbol{u}_k, p\right) = K_{lane} C_{lane}\left(\boldsymbol{x}_k, \boldsymbol{u}_k, p\right) + K_{orientation} C_{alignment}\left(\boldsymbol{x}_k, \boldsymbol{u}_k, p\right) \tag{5.16}$$

where K_{lane} and $K_{orientation}$ are the parameters weighing the corresponding contributions.

The cost term for the lane computes a paraboloid that is equal to 0 in the middle of the row and 1 in correspondence with the line delimiting the borders of the lane. For a given position $\boldsymbol{x} = [x_1, x_2, x_3, x_4], x_1 = x, x_2 = y$ of the rover, the corresponding cost is:

$$\begin{aligned} y_l &= a_l x_1 + b_l \\ y_r &= a_r x_1 + b_r \\ C_{lane} &= \frac{4}{(y_l - y_r)^2} x_2^2 - 4 \frac{(y_l + y_r)}{(y_l - y_r)^2} x_2 + \frac{(y_l + y_r)^2}{(y_l - y_r)^2} \end{aligned} \tag{5.17}$$

For each value of depth x_1 a convex-upward parabola is constructed along the axis x_2 with a value of zero in the middle of the lane. The trajectory with the minimum cost ($= 0$) aligns perfectly with the middle of the lane.

The cost term for the alignment is computed considering the difference between the angular coefficient of the middle line $a_{avg} = (a_l + a_r)/2$ and the angular coefficient of a straight line oriented as the rover a_{rover}. Recalling the trigonometric relation presented in Eq. 5.9,

$$\begin{aligned} a_{rover} = \tan\theta = \frac{\sin\theta}{\cos\theta} = \frac{2x_3x_4}{x_3^2 - x_4^2} \\ C_{alignment} = (a_{avg} - a_{rover})^2 \end{aligned} \tag{5.18}$$

5.3 Target approach

The idea behind the target approach relies on the relatively confined space, where standard navigation procedures already avoid obstacles, including target objects like fruit boxes. Given that the rover naturally traverses near the targets while evading obstacles, a straightforward strategy involves momentarily stopping the rover (to perform some task) as it passes by the target.

The initial component of this target approach pipeline is the **RGB Segmentation** node, tasked with segmenting the target in the RGB image. However, the primary scope of this work is the navigation system, so a basic HSV color filter has been employed as a demonstration to identify the targets. The mask obtained from the color filter is then applied to the depth image, eliminating background points.

Subsequently, the **Object Analysis** node executes the following sequence of operations:

1. Computes the PCD corresponding to the input depth frame using the 3D reconstruction process outlined in Section 3.4.
2. Performs down-sampling through voxelization to reduce the number of points and achieve a more uniformly dense PCD.
3. Utilizes the DBSCAN algorithm (detailed in Section 3.5.1) to cluster points into one or more objects. Multiple target objects may coexist in the same image, such as distinct fruit boxes at different depths.
4. Considers only the closest target, computing its pose and bounding box.
5. Checks all previously labeled targets to determine if there is one in proximity to the current object (considering both pose and volume, i.e., bounding box). If the current object is already labeled, its pose is updated; otherwise, a new label is assigned, and a new Reference Frame (RF) attached to the object is published in the TF2 subsystem of ROS2.

The RGB Segmentation and Object Analysis nodes execute their operations upon the arrival of each image message from the camera.

The final component, the **Target Approach** node, monitors if the target has been reached. Utilizing the `Row RF`, the goal is considered achieved when $x_{rover} \geq x_{target}$. In

essence, this node periodically compares the positions of the object and the rover relative to the `Row RF`. When the rover is at the same distance or slightly beyond the target, signifying that the rover is side by side with the target, the node deems the target reached and dispatches a message to the Behavior Tree (BT) to signal goal accomplishment. The target label to be reached, and consequently the corresponding RF, is set by the BT using a ROS2 service within this node. This approach operation, in the final portion, relies solely on the rover's odometry system, as the target exits the camera's FOV.

5.4 The Behavior Tree

The proposed navigation system is governed by a Behavior Tree (BT) (see Appendix D), completely represented in Fig. 5.3, and implemented using the Python library `py_trees`[1].

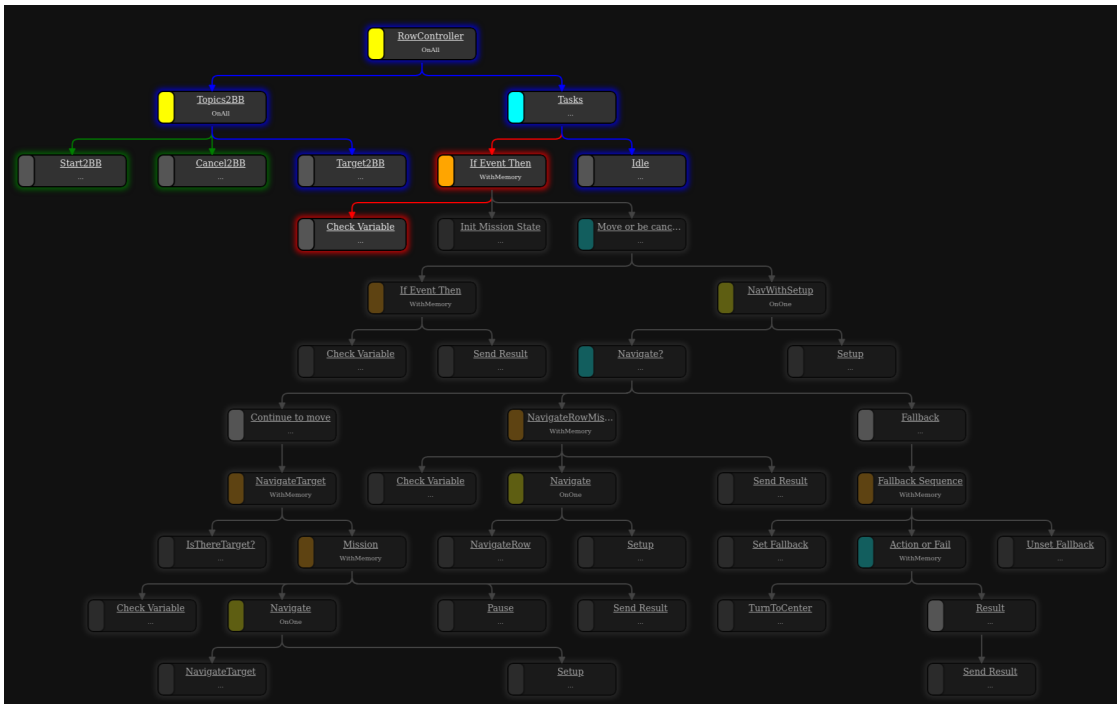


Figure 5.3: Behavior Tree (BT) used for orchestrating the navigation system.

Starting from the root, two parallel sub-trees unfold:

- A parallel sub-tree is responsible for listening to topics and updating the corresponding variables in the Blackboard.
- The `Tasks` sub-tree. The initial fork checks the `Start` variable. When a message arrives in the start topic, the `Start` blackboard variable is set to True. This sub-tree

[1] `https://py-trees.readthedocs.io/en/devel/introduction.html`

returns `SUCCESS`, proceeding to the next node, `Move or be canceled`. A similar fork monitors the `Cancel` variable and, if it is True, aborts the mission, otherwise, it proceeds by ticking the `Navigate?` node. Upon completion of the `Navigate?` node (whether reaching the end-of-the-row, encountering a cancel, or failure), the system transitions to the `Idle` node.

The navigation sub-tree is subdivided using a `Selector` into three distinct branches:

- The high-priority branch activates when a target is detected. Before navigation, it checks if the target has not already been approached. If not, this branch initiates navigation toward the target, pausing for a brief duration (simulating future tasks on the target) before returning `SUCCESS` or `FAILURE` (if an error occurs). A decorator is employed to seamlessly resume the main mission, i.e., reaching the end of the row.
- The mid-priority branch navigates through the row when no target is recognized.
- The low-priority branch is an action branch designed for fallback and recovery tasks. If this branch also returns `FAILURE`, the main tree returns `FAILURE` and transitions to the `Idle` state.

Chapter 6

Tests and experiments

6.1 The experimental platform

6.1.1 Rovers

(a) Jackal UGV

(b) Husky UGV

Figure 6.1: Jackal and Husky rovers from Clearpath Robotics used in the real environment tests.

This research utilized two distinct mobile robots: the Clearpath Robotics[1] Jackal UGV and Husky UGV (shown in Fig. 6.1).

The Jackal UGV is a compact robot designed for indoor and didactic robotics applica-

[1] `https://clearpathrobotics.com/`

tions. It features an onboard computer with GPS, IMU, and WiFi, integrated with ROS2. The robot's 4x4 drivetrain, aluminum chassis, and compatibility with various accessories make it suitable for several scenarios.

The Husky UGV, designed as a field robotics platform, is much bigger and more powerful. Its robust, low-maintenance design, high-resolution encoders, and lug-tread tires enable effective navigation across challenging terrains. The Husky UGV serves as a benchmark in robotics research, providing precise control, customization options, and compatibility with various accessories.

The complete technical specifications of the two rovers are shown in Tab. 6.1 (Jackal) and Tab. 6.2 (Husky).

6.1.2 Sensors

The Husky and the Jackal are equipped with several sensors, with a particular emphasis on RGBD cameras, which will be discussed in the following section.

Additionally, the Husky is equipped with a Velodyne[2] 3D LIDAR VLP16, which produces highly uniform and low-payload PCDs. The VLP16 is employed to compare the performance of the control system using PCDs obtained by the RGBD camera with the high-precision ones acquired by the LIDAR.

Furthermore, the Husky is equipped with a SwiftNav[3] Duro receiver GNSS sensor. This sensor leverages RTK technology, providing centimeter-level accuracy in location solutions. It offers accuracy that is 100 times greater than traditional GNSS solutions. The precise data from this sensor are used to compute an estimate of the rover's trajectory, enabling comparisons with a predefined desired trajectory.

The Husky also features a Microstrain[4] GX5 Inertial Measurement Unit (IMU). This sensor computes the odometry of the system, but its reliability diminishes over long paths (hundreds of meters) due to drift introduced by the numerical integration of accelerations measured directly by the IMU, and by the wheel slippage.

Intel Realsense RGBD camera D435 and D455

The Jackal UGV has been equipped with an Intel RealSense[5] RGBD camera D435, while the Husky UGV has been equipped with an Intel RealSense RGBD camera D455.

Both depth cameras employ some of the techniques presented in Sec. 3.5.2. Depth in these models is primarily derived from solving the stereoscopic problem (refer to Sec. 3.4). To enhance robustness, "active" methods are also used in conjunction with stereoscopic approaches. These cameras are equipped with an optical infrared light projector that overlays the observed scene with a semi-random texture. This texture

[2] `https://velodynelidar.com/`

[3] `https://www.swiftnav.com/`

[4] `https://www.microstrain.com/`

[5] `https://www.intelrealsense.com/`

Size and weight	
External dimensions ($L \times W \times H$)	$508 \times 430 \times 250$ mm
Internal dimensions	$250 \times 100 \times 85$ mm
Weight	17 kg
Ground Clearance	65 mm
Speed and performance	
All-terrain payload	10 kg
Max speed	2.0 m/s
Drive power	500 W
Battery and power system	
Capacity	270 Wh
Runtime - basic usage	8 Hours
Runtime - heavy usage	2 Hours
Charge time	4 Hours
Interfacing and communication	
Control modes	Direct voltage
	Wheel Velocity Commands
	Linear and angular velocity
Feedback	Battery and motor current
	Wheel velocity and travel
	Integrated GPS receiver
	Integrated gyroscope and accelerometer
Drivers and APIs	ROS, ROS2, C_{++}, and Python.
Communication	Ethernet, USB 3.0, RS232.
Integrated accessories	Wireless Game controller
	GPS
	IMU
	On-Board Computer
	WIFI Adapter
	Accessory Mounting Plates
Computer	
CPU	Intel Core i3-4330TE, Dual core, $2.4GHz$
RAM	8 GB
Enviromental	
Operating ambient temperature	-20 to 45°C Not in direct sunlight
Rating	IP62

Table 6.1: Technical specifications of Jackal UGV. [39]

facilitates finding correspondences, particularly in scenarios with texture-less surfaces or dimly lit environments.

In the current systems, these projectors are positioned between the left and right stereo imagers and are synchronized to turn on only when required. It's important to note that this is not a strict requirement. For *active stereo depth systems*, no a priori knowledge

Size and weight	
External dimensions ($L \times W \times H$)	$990 \times 670 \times 390$ mm
Internal dimensions	$296 \times 411 \times 155$ mm
Weight	50 kg
Wheels	330 mm
Ground Clearance	130 mm
Speed and performance	
All-terrain payload	20Kg
Max speed	1.0 m/s
Drivetrain / Drive power	4×4 Zero-Maintenance
Max climb grade	45°(100% Slope)
Max traversal grade	30° (58% Slope)
Battery and power system	
Capacity	$24\ \mathrm{V}, 20\mathrm{Ah}$
Runtime - standby	8 Hours
Runtime - nominal usage	3 Hours
Charge time	4 Hours
Interfacing and communication	
Control modes	Direct voltage
	Wheel Velocity Commands
	Linear and angular velocity
Feedback	Battery voltage
	Motor currents
	Wheel odometry
	Control system output
Drivers and APIs	ROS, ROS2, C++, and Python
Communication	RS232
Computer	
CPU	Intel Core i7-6700TE, Quad core, $2.4GHz$
RAM	16 GB
Enviromental	
Operating ambient temperature	-10 to 40°C Not in direct sunlight
Rating	IP 44 (upgrade to IP 55 available)

Table 6.2: Technical specifications of Husky UGV. [40]

of the projection pattern is needed, and there is no requirement for strict stability over time of these patterns. Additionally, it doesn't matter if other cameras point at the same scene with their projectors (changing the pattern). Multiple projectors can improve overall performance by adding more light and more texture. This property contrasts with "structured light" depth sensors (presented in Sec. 3.5.2), where there are strong requirements for pattern stability across time and temperature, leading to increased cost and susceptibility to external interference.

The main differences between the two models (D435 vs D455) lie in the ideal range and

Figure 6.2: Intel Realsense Depth Camera D455. [41]

FOV, with the D455 model offering wider specifications, as illustrated by the respective data sheets in Tables 6.3 (D435) and 6.4 (D455).

General information	
External dimensions ($L \times W \times H$)	$90 \times 25 \times 25$ mm
Ideal Range	0.3 to 3 m
Use environment	Indoor/Outdoor
Depth	
Depth technology	Stereoscopic
FOV	$87^\circ \times 58^\circ$
Resolution	Up to 1280×720
Frame rate	Up to 90 fps
Depth Accuracy	$< 2\%$ at 2 m
Minimum Depth Distance at Max Resolution	28 cm
RGB	
RGB technology	Rolling shutter
FOV	$69^\circ \times 42^\circ$
Resolution	Up to 1920×1080
Frame rate	Up to 30 fps
Sensor resolution	2MP

Table 6.3: Technical specifications of Intel Realsense Depth Camera D435. [41]

6.2 The evaluation metrics

For testing and validation, extensive experiments were conducted on both realistically simulated and real vineyards to illustrate the competitive advantages of the proposed solution. The metrics used to evaluate the performance of the navigation system include:

- **Clearance Time** [s], representing the duration the rover takes to fulfill its task of navigating through the row. A lower value for this metric is indicative of better performance.

General information	
External dimensions ($L \times W \times H$)	$124 \times 26 \times 29$ mm
Ideal Range	0.6 to 6 m
IMU	Bosch BMI055
Use environment	Indoor/Outdoor
Depth	
Depth technology	Stereoscopic
FOV	$87° \times 58°$
Resolution	Up to 1280×720
Frame rate	Up to 90 fps
Depth Accuracy	$< 2\%$ at 4 m
Minimum Depth Distance at Max Resolution	52 cm
RGB	
RGB technology	Global shutter
FOV	$90° \times 58°$
Resolution	Up to 1280×800
Frame rate	Up to 30 fps
Sensor resolution	1MP

Table 6.4: Technical specifications of Intel Realsense Depth Camera D455. [41]

- **Mean linear velocity** v_{avg} [m/s], serving as a measure of the effectiveness of the proposed solution. The benchmark is set by the maximum velocity defined for the rover in the test, and the optimal scenario involves the rover traversing the row at this maximum speed.
- **Cumulative heading average** $Cum.\gamma_{avg}$ or **standard deviation of the heading** γ_{std} [rad], gauging the oscillation around the trajectory. A lower value for these metrics signifies more stable navigation.
- **Standard deviation of the angular velocity** ω_{std} [rad/s], used to quantify the oscillation around the trajectory.
- Trajectory **Mean Absolute Error (MAE)** [m], defined as

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_{rover} - y_{wanted}(x_{rover})| \tag{6.1}$$

  where N is the number of time steps at which we evaluate the performance. Additionally, the trajectory **Mean Squared Error (MSE)** [m^2] is employed

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_{rover} - y_{wanted}(x_{rover}))^2 \tag{6.2}$$

  These metrics serve to quantify the error in the rover's trajectory concerning a predefined desired trajectory, such as the center of the row or the lane.

6.3 Tests in simulation environment

For simulated tests, the Gazebo[6] platform, the Jackal model and description, and the PIC4rl_gym[42] evaluation tool are utilized. This tool automates the assessment of the aforementioned metrics by executing multiple iterations of the same test and calculating the corresponding performance metrics.

Fig. 6.3 provides a visual representation of the simulated environment used in the tests. Notably, the image captures both straight and curved vineyards, offering a comprehensive simulation scenario for evaluating the proposed navigation system. For the tests in this environment, a maximum linear velocity of $v_{max} = 0.4\ m/s$ has been set.

Figure 6.3: Gazebo world used for simulation tests.

Results

The extensive simulations conducted in simulated vineyard environments have demonstrated the reliability and robustness of the proposed navigation system. As illustrated in Fig. 6.4, the rover's trajectory closely aligns with the desired central path, exhibiting minimal oscillations in both straight and curved vineyards.

Detailed results are provided in Tab. 6.5 (straight vineyard) and Tab. 6.6 (curved vineyard), revealing several key performance indicators:

- In both straight and curved vineyards, the rover consistently achieves speeds close to the maximum limit ($v_{avg} \simeq 0.39\ m/s$ for $v_{max} = 0.4\ m/s$), resulting in effective clearance times.

[6] `https://gazebosim.org/home`

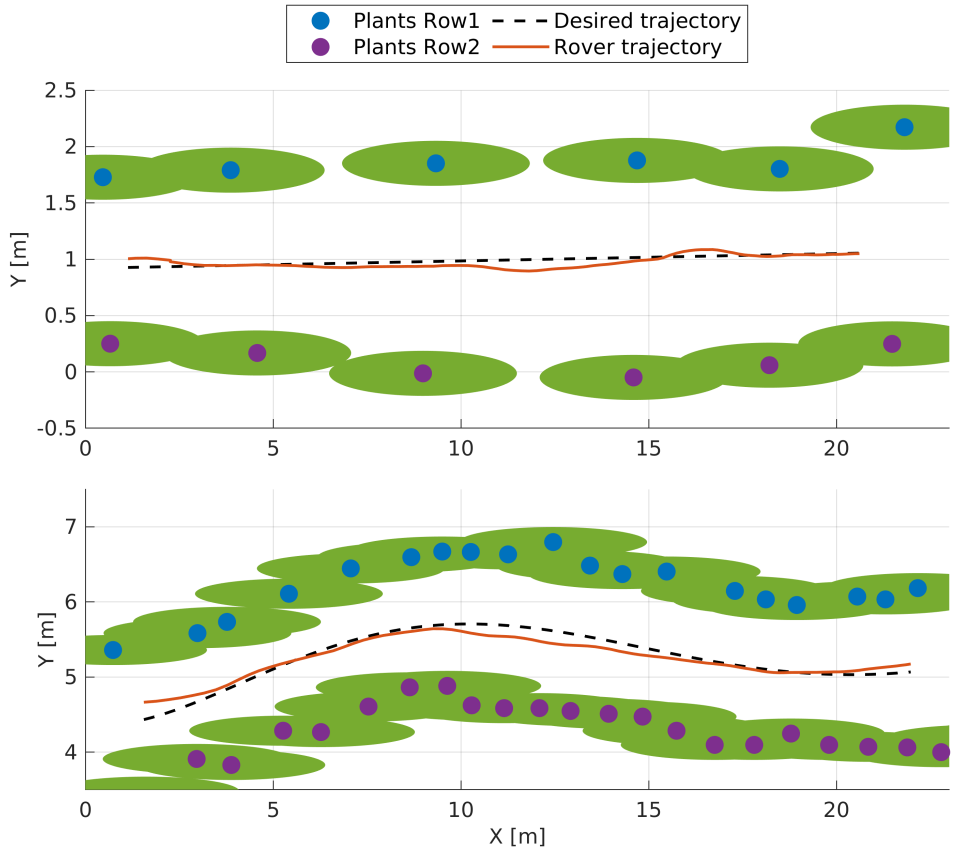


Figure 6.4: Tests in a simulated vineyard using the PCD of the camera as input in two different scenarios.

- The rover's trajectory shows minimal oscillations, as indicated by a small standard deviation of angular velocity ($\omega_{std} \simeq 0.05\ rad/s$), reflecting stable and smooth behavior.
- Path metrics, including MAE and MSE, are remarkably small, on the order of centimeters. This demonstrates the rover's precise adherence to the center of its lane. In the curved vineyard, a slightly larger path error is observed (MAE up to 20 cm in the worst case), attributed to the controller's inclination to cut curves. This behavior can be mitigated through parameter tuning.
- Interestingly, in both scenarios, the use of input segmentation to detect the crops did not yield benefits, as the vision algorithm primarily relied on geometric considerations, and it adversely affected computing capabilities. So, this resulted in a trajectory with a slightly worse MAE error and a significantly smaller mean linear velocity v_{avg}, leading to a larger clearance time.
- The algorithm's consistent performance across different input sensors, including RGBD cameras, highlights its reliability and versatility. This robustness, even when

compared to more expensive technologies such as LIDAR, underscores the algorithm's adaptability to various sensor configurations. The ability to achieve comparable results with RGBD cameras suggests a cost-effective alternative for applications where LIDARs may be cost-prohibitive.

Overall, these findings underscore the effectiveness and versatility of the proposed navigation system across diverse vineyard scenarios.

Sensor	Input Seg	Clearance time [s]	Cum. γ_{avg} [rad]	v_{avg} [m/s]	ω_{std} [rad/s]	MAE [m]	MSE [m^2]
LIDAR	-	49.528±0.167	0.036±0.001	**0.395±0.002**	**0.034±0.001**	**0.034±0.001**	**0.001±0.000**
PCD cam	No	52.586±4.130	0.045±0.001	0.377±0.019	0.038±0.001	0.048±0.005	0.003±0.001
	Yes	68.768±8.545	0.024±0.009	0.303±0.032	0.040±0.002	0.080±0.008	0.009±0.002
RGBD cam	No	**49.321±0.356**	**0.011±0.005**	0.395±0.001	0.046±0.004	0.104±0.011	0.018±0.004
	Yes	57.796±7.277	0.023±0.008	0.351±0.033	0.041±0.003	0.072±0.014	0.008±0.002

Table 6.5: Results of a series of experiments in a simulated straight vineyard. The desired trajectory is in the middle of the row.

Sensor	Input Seg	Clearance time [s]	Cum. γ_{avg} [rad]	v_{avg} [m/s]	ω_{std} [rad/s]	MAE [m]	MSE [m^2]
LIDAR	-	52.080±0.220	-0.024±0.001	**0.397±0.001**	**0.036±0.001**	0.102±0.001	0.015±0.000
PCD cam	No	52.157±0.673	**0.002±0.002**	0.393±0.002	0.041±0.003	**0.068±0.004**	**0.007±0.001**
	Yes	66.705±1.388	-0.016±0.012	0.322±0.006	0.047±0.003	0.188±0.011	0.049±0.005
RGBD cam	No	**51.763±0.228**	-0.011±0.002	0.394±0.001	0.056±0.007	0.188±0.005	0.051±0.003
	Yes	70.645±6.252	-0.026±0.015	0.313±0.023	0.047±0.036	0.213±0.023	0.064±0.011

Table 6.6: Results of a series of experiments in a simulated curved vineyard. The desired trajectory is in the middle of the row.

6.4 Tests in real-world scenario

In tests in a real vineyard, the Jackal and Husky rovers from Clearpath Robotics, an Intel Realsense RGBD camera, and a Velodyne 3D LIDAR for comparison have been utilized.

The path metrics presented earlier were evaluated using the Husky UGV. Rover localization in the row was necessary for comparing its position to a desired path. This is quite paradoxical, since, as I said in Sec. 1, it is very difficult to localize the system within this environment (it is for these reasons that I proposed a position-agnostic system).

The odometry system of the IMU of the rover failed to localize the rover due to significant drifts. In Fig. 6.5 it is visible the drift of the odometry trajectory (on the right), which is more curved with respect to the more accurate GPS one (on the left).

Visual SLAM methods, like KISS-ICP[43], also failed to correctly localize the system due to the repetitiveness of the environment, resulting in loop formations and registration failures in the generated PCD.

The GPS position provided by the MicroStrain GNSS Inertial Sensor was used as a reference to compute the metrics, along with a precise geo-localization of the row in the vineyards. In Fig. 6.6 the satellite view of the vineyards is shown together with its

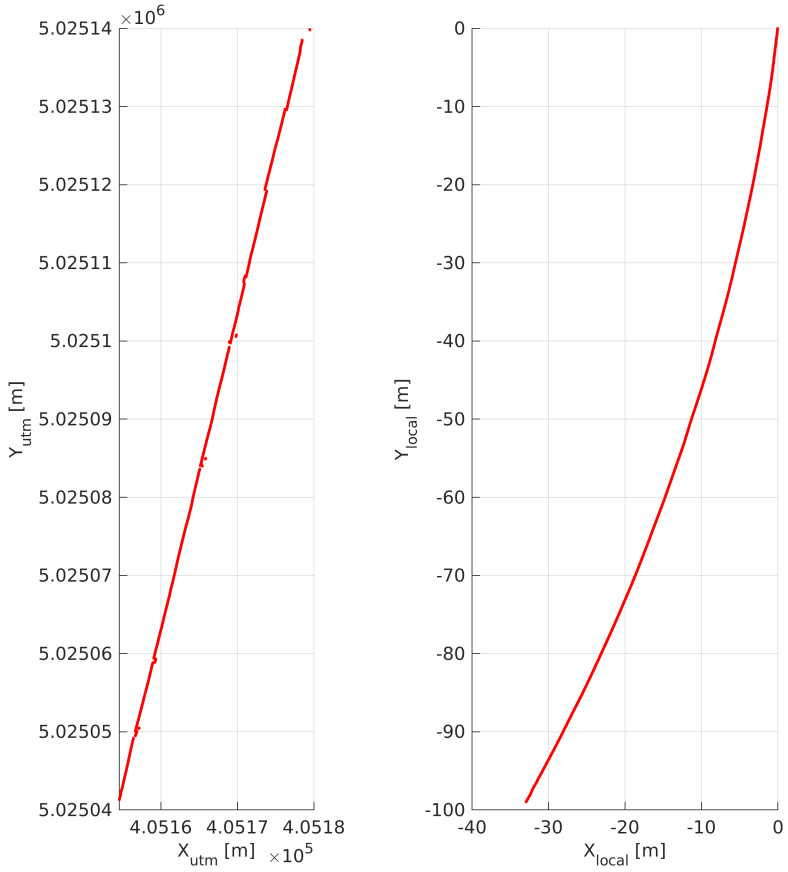


Figure 6.5: Trajectory measured using the GPS (on the left) and the Odometry (on the right).

geo-reference in Universal Transverse Mercator (UTM) coordinates. On the image, the trajectory measured by the GPS of the rover has been superimposed.

This GPS-based reference, however, comes with its challenges, especially in scenarios where foliage obstructs GPS visibility, leading to signal failures and inaccuracies in position tracking. These localization complexities underscore the importance of the position-agnostic controller developed in this project.

Results

The real-world tests conducted in vineyards have validated the results obtained in the simulated environment, demonstrating the efficacy of the proposed navigation system. In Fig. 6.7, two segments from tests in a straight vineyard (intra-row distance of around 2 m) are presented, representing both the center and the right lane configurations. In the center configuration, the rover exhibits straight motion with minimal oscillation and an offset of a few centimeters from the estimated center of the row. This small deviation can be attributed to the manual estimation of the lane position based on the geo-referenced

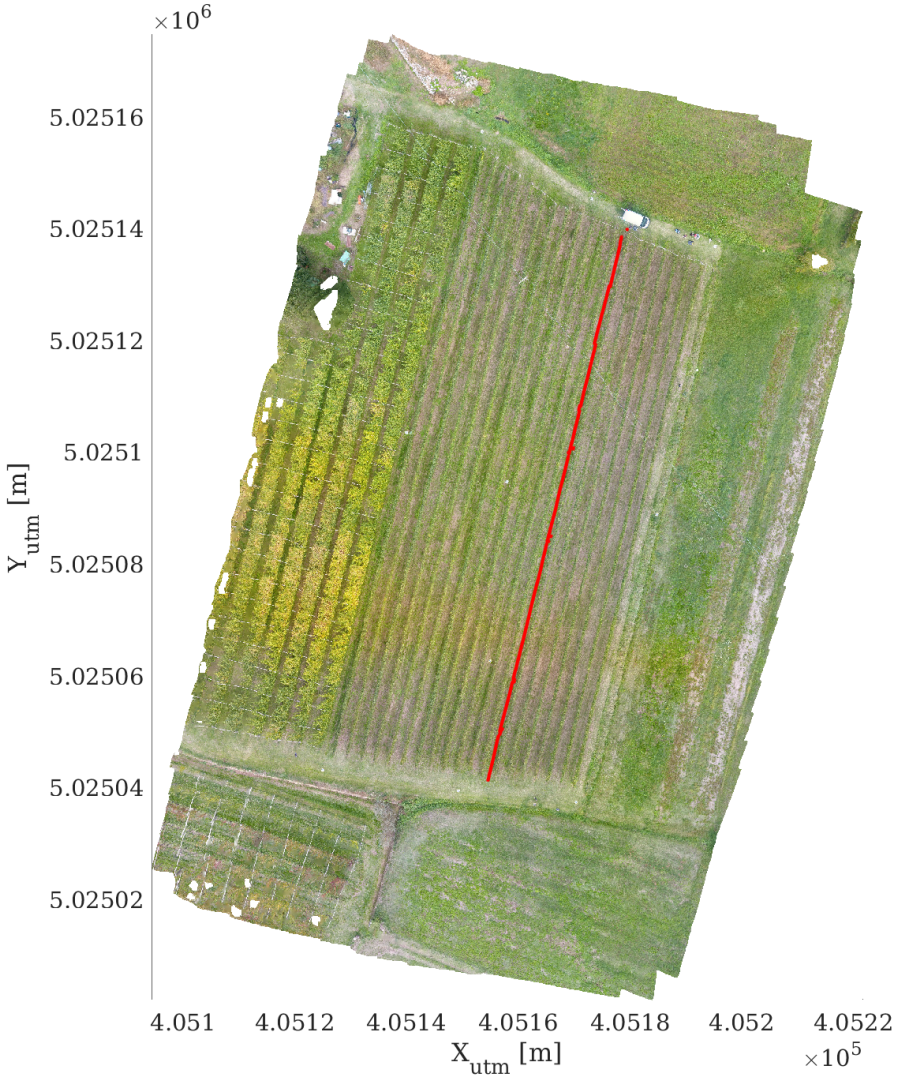


Figure 6.6: Satellite view of the vineyard. In red is the trajectory followed by the Husky rover during a test session.

satellite image in Fig. 6.6.

For the right lane configuration in the narrow vineyard (2 m intra-row distance), the rover displays a more oscillatory behavior, likely due to the proximity of the right lane to the crops. This behavior is less prominent in the pergola vineyard test (Fig. 6.8) with a larger intra-row distance (4 m), where the rover shows a smooth convergence to the right lane without significant oscillations.

Fig. 6.8 illustrates the results of tests conducted in a pergola vineyard, where occasional discrepancies in the estimated trajectory are observed. These deviations are attributed to errors introduced by dense canopies obstructing satellite line-of-sight, affecting the GPS localization system. Moreover, on the bottom part of the figure, it is shown how the rover

smoothly converges to the right lane from the middle of the row.

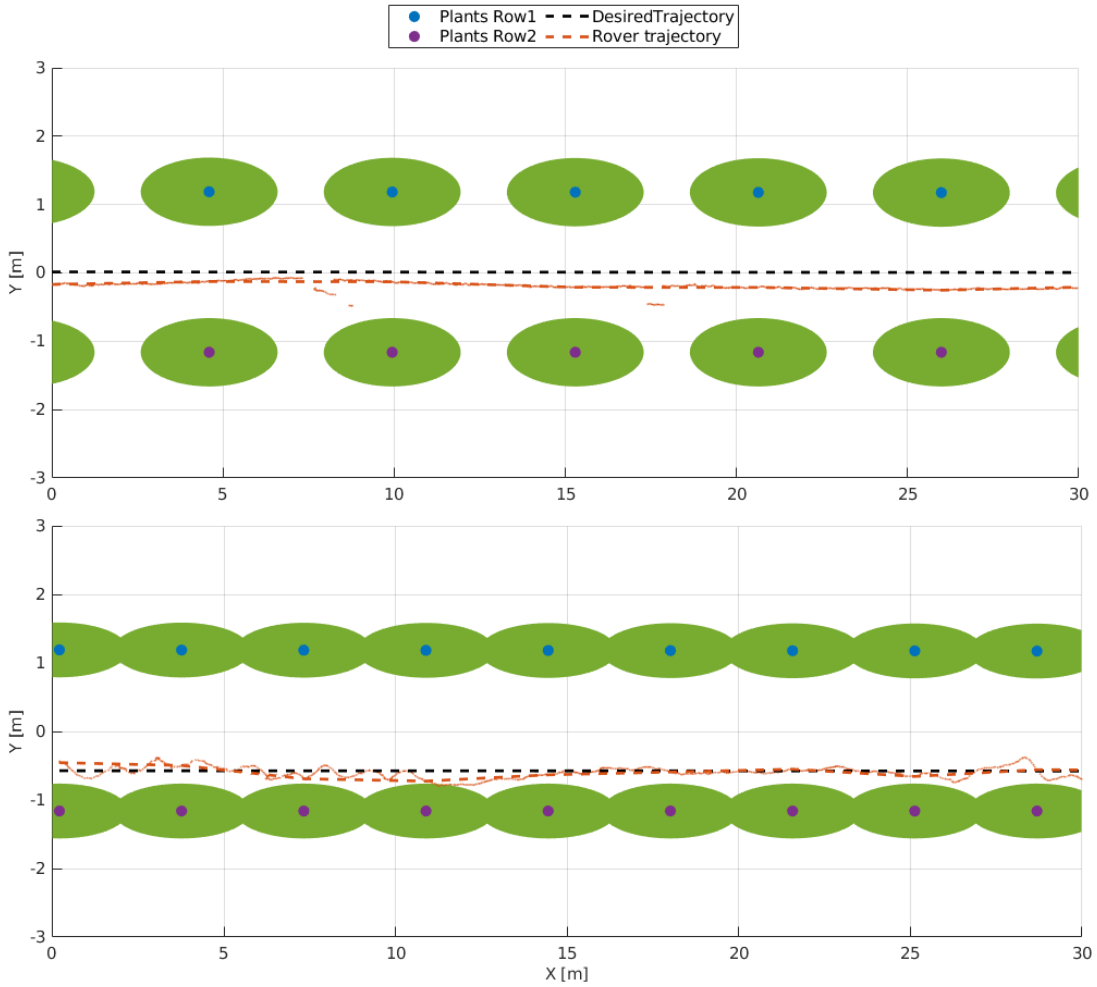


Figure 6.7: Tests in a real straight vineyard using the PCD of the camera as input in two different configuration. On top the desired trajectory is in the middle of the row, while on the bottom figure, it is in the middle of the right lane.

Detailed results are presented in Tab. 6.7 (straight vineyard) and Tab. 6.8 (pergola vineyard), highlighting the robust performance of the controller in real scenarios:

- Also in real scenarios, the rover consistently achieves speeds close to the maximum limit ($v_{avg} \simeq 0.399\ m/s$ for $v_{max} = 0.4\ m/s$ and $v_{avg} \simeq 0.49\ m/s$ for $v_{max} = 0.5\ m/s$).
- The rover's trajectory shows minimal oscillations, as indicated by a small standard deviation of angular velocity ($\omega_{std} \simeq 0.05\ rad/s$), reflecting stable and smooth behavior. As discussed, the exception is the narrow straight vineyard in the right lane configuration, where this metric is slightly larger ($\omega_{std} \simeq 0.18\ rad/s$).

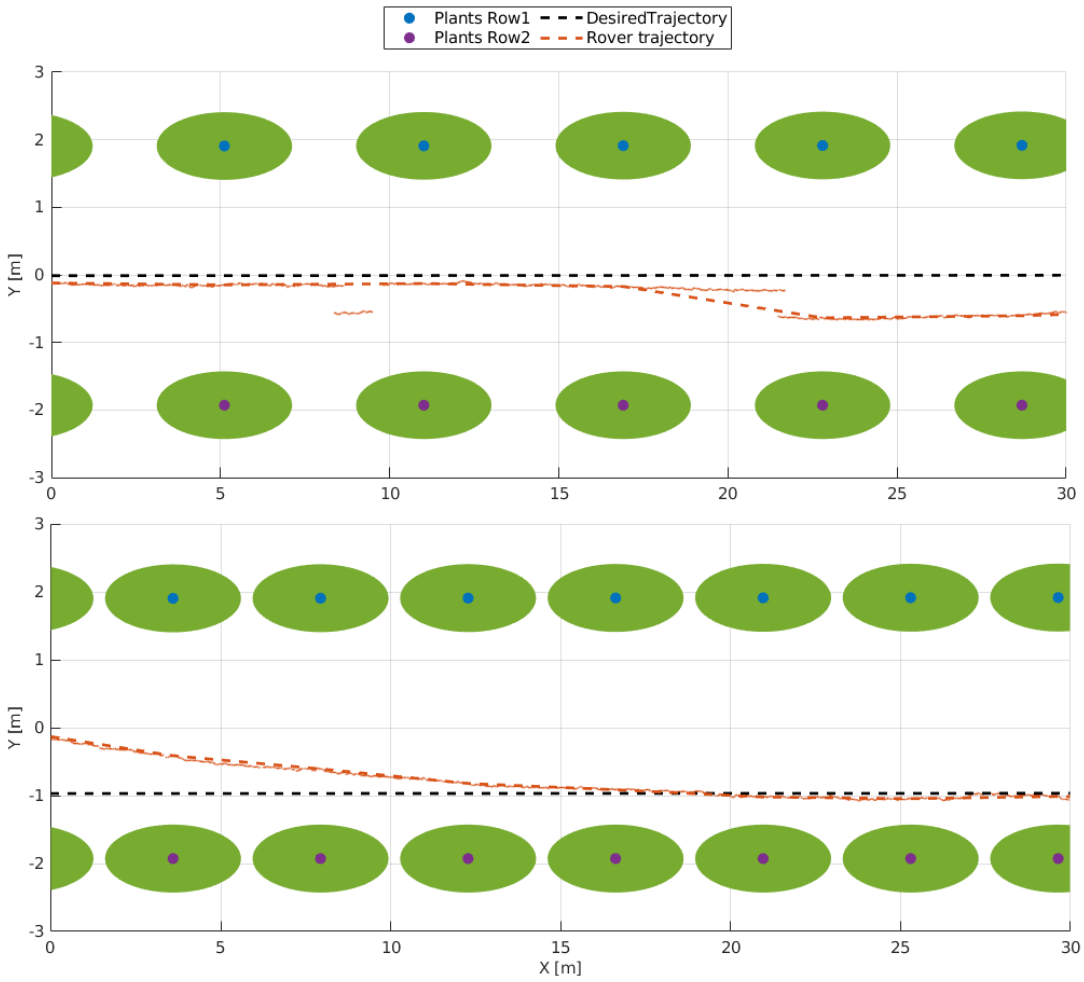


Figure 6.8: Tests in a real pergola vineyard using the PCD of the camera as input in two different configuration. On top the desired trajectory is in the middle of the row, while on the bottom figure, it is in the middle of the right lane.

- Path metrics, including MAE and MSE, are minimal, on the order of centimeters (up to 20 *cm* for the narrow vineyard and up to 30 *cm* for the larger pergola vineyard). However, it's important to consider the error in the reference trajectory when interpreting these results.

The evaluation of the Jackal's ability to recognize and approach fruit boxes provided insightful observations. The system demonstrated good effectiveness, achieving a success rate of $11/14 = 79\%$ in correctly identifying and approaching the target objects. However, a noteworthy anomaly was identified in $2/14 = 14\%$ of the cases. In these instances, multiple stops occurred per fruit box, attributable to a failure in the TF2 subsystem of ROS2, specifically arising from synchronization issues.

Sensor	Position	v_{max} [m/s]	γ_{std} [rad]	v_{avg} [m/s]	ω_{std} [m/s]	MAE [m]	MSE [m²]
PCD camera	Centered	0.4	0.031±0.007	0.399±0.000	0.042±0.002	0.165±0.007	0.035±0.000
	Right lane	0.5	0.388±0.395	0.488±0.007	0.184±0.108	0.204±0.098	0.070±0.044
LIDAR	Right lane	0.5	0.0153	0.4989	0.0271	0.1519	0.0294

Table 6.7: Results of a series of experiments in a real straight vineyard. Intra-row space of around $2.5m$.

Sensor	Position	v_{max} [m/s]	γ_{std} [rad]	v_{avg} [m/s]	ω_{std} [m/s]	MAE [m]	MSE [m²]
PCD camera	Centered	0.4	0.122	0.399	0.063	0.313	0.129
	Right lane	0.4	0.047	0.399	0.04	0.092	0.011

Table 6.8: Results of a series of experiments in a real pergola vineyard. Intra-row space of around $4m$.

This synchronization issue led to the unintended labeling of the same fruit box twice. Consequently, the rover's approach to the target followed an expected sequence, coming to a stop as intended. However, the system then initiated a restart of the navigation, but it immediately stopped again upon reaching the new target (same fruit box as the previous one, second label). Importantly, it's crucial to highlight that no collisions with the fruit boxes were detected throughout these occurrences.

Chapter 7

Conclusion and future works

The developed controller has demonstrated robustness in effectively handling the diverse challenges presented by variations in crop density, height, and other environmental factors. Its successful navigation through pergola vineyards and resilient functionality on rough terrains underscore its adaptability to real-world agricultural conditions. This research significantly contributes to the continuous advancement of precision agriculture and the evolution of autonomous navigation systems for row-based crop environments.

Looking ahead, the research lays the groundwork for future investigations and advancements. One crucial avenue for exploration involves meticulous parameter tuning for specific scenarios, such as navigating through curved or very narrow vineyards. This fine-tuning process can increase the controller's adaptability and performance in diverse agricultural settings.

The controller's capabilities to perform transitions between rows must be added to the system to complete the autonomous navigation in a field. To address this challenge, a multi-camera setup is proposed, featuring three cameras, one front-facing and two positioned on the sides of the rover. This configuration aims to enhance the rover's perception and recognition capabilities, enabling it to turn and identify the characteristics of the next row. The two side cameras provide additional perspectives, offering a wider field of view to gather information about adjacent rows and facilitate a smooth turn. The transition process begins with the front-facing camera identifying the end of the current row. Once detected, the system triggers the side cameras to assess the characteristics of the upcoming row, including its spacing, orientation, and potential obstacles. This information is then fed into the NMPC controller. Leveraging the input from the multi-camera setup, the NMPC computes an optimized trajectory for the rover's movement during the turn and subsequent navigation along the new row.

The long-term vision involves the implementation of a more versatile and robust image segmentation and recognition algorithm. This enhancement aims to increase the system's capabilities not only for target recognition purposes, such as identifying relevant objects like fruit boxes, but also in refining navigation through advanced row recognition.

In essence, this research project represents a significant stride towards addressing the challenges posed by row-based crop environments, offering a foundation for ongoing exploration and innovation in the realm of autonomous agricultural systems.

Appendix A

Homogeneous Coordinates

Consider a point in a n-dimensional Euclidean space, $\boldsymbol{x} \in \mathbb{R}^n$ where $\boldsymbol{x}$ represents the coordinate vector $(x_1, x_2, \dots x_n)$. It is possible to define the corresponding point in homogeneous coordinates $\tilde{\boldsymbol{x}} \in \mathbb{P}^n$ (the tilde is used to underline that a given quantity is homogeneous) as the coordinate vector $(\tilde{x}_1, \tilde{x}_2, \dots \tilde{x}_{n+1})$, where

$$x_i = \frac{\tilde{x}_i}{\tilde{x}_{n+1}}, \quad i = 1, 2, \dots n \tag{A.1}$$

The linear space where points in homogeneous coordinates reside is commonly called the *projective space* $\mathbb{P}_n$. Notably, a vector in $\mathbb{P}_n$ has $n+1$ coordinates or *degree of freedom*. On the other hand, a homogeneous coordinate vector can be easily constructed from a Euclidean coordinate vector $\boldsymbol{x} = (x_1, x_2, \dots x_n)$ as

$$\tilde{\boldsymbol{x}} = (x_1, x_2, \dots x_n, 1) \tag{A.2}$$

Vectors in $\mathbb{P}^n$ can be easily translated or rotated by multiplication with an appropriate $(n+1) \times (n+1)$ homogeneous transformation matrix $\boldsymbol{T}$. It is also essential to note that an Euclidean point $\boldsymbol{x}$ can be mapped not only to $\tilde{\boldsymbol{x}} = (x_1, x_2, \dots x_n, 1)$, but also to $\tilde{\boldsymbol{x}}' = \alpha\tilde{\boldsymbol{x}}$ for all $\alpha \neq 0$; in other words, a point $\boldsymbol{x} \in \mathbb{R}^n$ corresponds to a line in $\mathbb{P}^n$. This property holds great significance in Computer Vision applications since *"the relationship between points and rays is at the core of the projective transformation"* [18]. Points in the 2-dimensional image plane ($\mathbb{R}^2$) correspond to lines in the 3-dimensional physical world (projective space $\mathbb{P}^2$).

In $\mathbb{P}^2$ a line is defined by a 3-tuple, $\tilde{\boldsymbol{l}} = (\tilde{l}_1, \tilde{l}_2, \tilde{l}_3)^T$, and the corresponding set of points satisfies the equation:

$$\tilde{\boldsymbol{x}} : \tilde{\boldsymbol{l}}^T\tilde{\boldsymbol{x}} = 0 \tag{A.3}$$

Expanding this representation using Euclidean coordinates $\boldsymbol{x} = (x_1, x_2)$ results in $l_1x_1 + l_2x_2 + l_3 = 0$, which is the *canonical form* of a line in $\mathbb{R}^2$. All lines, including those parallels to an axis of the Cartesian plane, can be represented in this form with a 3-tuple $\tilde{\boldsymbol{l}}$ of real numbers.

Moreover, as a point can be defined by the intersection of two lines $\tilde{\boldsymbol{l}}_1$ and $\tilde{\boldsymbol{l}}_2$, it is possible to obtain the *line equation of a point*

$$\tilde{\boldsymbol{p}} = \tilde{\boldsymbol{l}}_1 \times \tilde{\boldsymbol{l}}_2 \tag{A.4}$$

Similarly, it is also possible to define the line passing through two distinct points (from $\tilde{\boldsymbol{p_1}}$ to $\tilde{\boldsymbol{p_2}}$) as

$$\tilde{\boldsymbol{l}}_{12} = \tilde{\boldsymbol{p}}_1 \times \tilde{\boldsymbol{p}}_2 \tag{A.5}$$

In the particular case of two parallel lines $\tilde{\boldsymbol{l}}_1 = (\tilde{a}, \tilde{b}, \tilde{c}_1)$ and $\tilde{\boldsymbol{l}}_2 = (\tilde{a}, \tilde{b}, \tilde{c}_2)$, using Eq. A.4, we obtain a point

$$\tilde{\boldsymbol{p}} = \begin{pmatrix} \tilde{b}(\tilde{c}_2 - \tilde{c}_1) \\ \tilde{a}(\tilde{c}_1 - \tilde{c}_2) \\ 0 \end{pmatrix} \tag{A.6}$$

Since its last coordinate is equal to 0, this point $\tilde{\boldsymbol{p}} \in \mathbb{P}^2$ corresponds to a point at infinity in Euclidean space $\mathbb{R}^2$, and, for this reason, is referred to as an *ideal point*. Using real numbers, homogeneous coordinates simplify the representation and manipulation of points and lines at infinity.

Appendix B

Quaternions

Quaternions are a generalization of complex numbers to a 3D space [35], and they were first introduced by Sir W. Hamilton in 1843. Their primary utility is to efficiently represent 3D rotations, whether in a static or dynamic scenario. As a result, quaternions have become an indispensable component in various technical domains, especially in fields like computer graphics, robotics, and aerospace.

Quaternions can be defined as element of a 4D linear space $\mathbb{H}(\mathbb{R})$ defined on the real number field, with base $\{1 \quad i \quad j \quad k\}$ [44]. In particular, $\{\mathbf{i}, \mathbf{j}, \mathbf{k}\}$ are *hypercomplex* numbers that satisfy the following *anticommutative* multiplication rules:

$$\begin{aligned} &\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{i} \otimes \mathbf{j} \otimes \mathbf{k} = -1 \\ &\mathbf{i} \otimes \mathbf{j} = -\mathbf{j} \otimes \mathbf{i} = \mathbf{k} \\ &\mathbf{j} \otimes \mathbf{k} = -\mathbf{k} \otimes \mathbf{j} = \mathbf{i} \\ &\mathbf{k} \otimes \mathbf{i} = -\mathbf{i} \otimes \mathbf{k} = \mathbf{j} \end{aligned} \tag{B.1}$$

The following notations are equivalent to indicate a quaternion $\mathfrak{q}$:

$$\begin{aligned} \mathfrak{q} &= q_0 + \mathbf{q} \\ &= q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} \\ &= \cos\frac{\beta}{2} + \mathbf{u}\sin\frac{\beta}{2} \\ &= e^{\mathbf{u}\frac{\beta}{2}} \\ &= \left(\cos\frac{\beta}{2}, u_1\sin\frac{\beta}{2}, u_2\sin\frac{\beta}{2}, u_3\sin\frac{\beta}{2}\right) \\ &= (q_0, q_1, q_2, q_3) \\ &= (q_0, \mathbf{q}) = \begin{bmatrix} q_0 \\ \mathbf{q} \end{bmatrix} = \begin{bmatrix} \cos\frac{\beta}{2} \\ \mathbf{u}\sin\frac{\beta}{2} \end{bmatrix} \end{aligned}$$

where q_0 is the real part, and $\mathbf{q}$ is the imaginary (or vector) part. Quaternions are mathematical objects that include real numbers $\mathfrak{q} = (r,0,0,0), \quad r \in \mathbb{R}$, complex number

$\mathfrak{c} = a + ib = (a, b, 0, 0), \quad a, b \in \mathbb{R}$ and also real coordinate vector in a 3D-world $\mathfrak{p} = (0, p_1, p_2, p_3), \quad p_i \in \mathbb{R}$.

Quaternion Algebra

Let us analyze briefly some algebraic operations with quaternions.

- There exists the null element, which is $\mathfrak{O} = (0, \mathbf{0})$.
- The complex conjugate of a quaternion $\mathfrak{q} = q_0 + \mathbf{q}$ is

$$\mathfrak{q}^* \doteq q_0 - \mathbf{q} = (q_0, -\mathbf{q}) = \cos\frac{\beta}{2} - \mathbf{u}\sin\frac{\beta}{2} = e^{-\mathbf{u}\frac{\beta}{2}} \tag{B.2}$$

- The norm of a quaternion is

$$|\mathfrak{q}| = \|\mathfrak{q}\| = \|\mathfrak{q}\|_2 = |\mathfrak{q}^*| = \sqrt{\mathfrak{q}\cdot\mathfrak{q}^*} = \sqrt{\sum_{i=0}^{3} q_i^2}. \tag{B.3}$$

  A quaternion with unit norm is usually referred to as *unit quaternion*.

- The reciprocal of a quaternion $\mathfrak{q}$ is

$$\begin{aligned} \mathfrak{q}^{-1} &= \mathfrak{q}^*/|\mathfrak{q}| \\ \mathfrak{q}^{-1} &= \mathfrak{q}^* \quad \text{for a unit quaternion.} \end{aligned} \tag{B.4}$$

- Sum: $\mathfrak{q} + \mathfrak{p} = \underbrace{(q_0 + p_0)}_{\text{real part}} + \underbrace{(\mathbf{q} + \mathbf{p})}_{\text{imaginary part}}$
- Dot product: $\mathfrak{q}\cdot\mathfrak{p} = \sum_{i=0}^{3} q_i p_i$.
- Quaternion product (Hamilton product):

$$\mathfrak{q}\otimes\mathfrak{p} = (q_0 + \mathbf{q})\otimes(p_0 + \mathbf{p}) = \underbrace{(q_0 p_0 - \mathbf{q}\cdot\mathbf{p})}_{\text{real part}} + \underbrace{(q_0\mathbf{p} + p_0\mathbf{q} + \mathbf{q}\times\mathbf{p})}_{\text{imaginary part}} \tag{B.5}$$

  where:

$$\mathbf{q}\cdot\mathbf{p} = \sum_{i=1}^{3} q_i p_i \qquad \text{is the vector dot product and}$$

$$\mathbf{q}\times\mathbf{p} = \begin{bmatrix} q_2 p_3 - q_3 p_2 \\ q_3 p_1 - q_1 p_3 \\ q_1 p_2 - q_2 p_1 \end{bmatrix} \qquad \text{is the cross product.}$$

  The identity element for the quaternion product is $\mathfrak{I} \doteq (1, \mathbf{0}) : \quad \mathfrak{q}\otimes\mathfrak{I} = \mathfrak{q}, \quad \mathfrak{I}\otimes\mathfrak{q} = \mathfrak{q}$. Moreover, the quaternion product is associative, and non-commutative, differently from the cross product which is non-associative, and anti-commutative.

Quaternion and Rotations

Moreover, quaternions can be also used to model rotations. Let a 3D vector $\mathbf{r} = (x, y, z)$ be given, and consider a rotation of $\mathbf{r}$ about an axis $\mathbf{u} = (u_1, u_2, u_3)$ of an angle β :

$$\mathbf{r}' = \mathbf{T}(\beta, \mathbf{u})\mathbf{r}.$$

Both $\mathbf{r}$ and $\mathbf{r}'$ can be seen as the vector parts of quaternions with null real part, given by $(0, \mathbf{r})$ and $(0, \mathbf{r}')$. It can be proven that, defined the unit quaternion,

$$\mathfrak{q} \doteq \left(\cos\frac{\beta}{2}, u_1 \sin\frac{\beta}{2}, u_2 \sin\frac{\beta}{2}, u_3 \sin\frac{\beta}{2}\right).$$

the rotated vector $\mathbf{r}'$ can be computed as

$$(0, \mathbf{r}') = \mathfrak{q} \otimes (0, \mathbf{r}) \otimes \mathfrak{q}^*$$

Of course, the inverse rotation (same axis of rotation but opposite angle) can be modeled with the inverse quaternion $\mathfrak{q}^{-1} = \mathfrak{q}^*$, since $\mathfrak{q}$ is a unit quaternion. Moreover, given a rotation composition

$$\mathbf{T} = \mathbf{T}_1\mathbf{T}_2 \dots \mathbf{T}_n$$

the quaternion corresponding to the rotation $\mathbf{T}$ is

$$\mathfrak{q} = \mathfrak{q}_1 \otimes \mathfrak{q}_2 \otimes \cdots \otimes \mathfrak{q}_n$$

where $\mathfrak{q}_i$ is the quaternion corresponding to the rotation $\mathbf{T}_i$. To better exploit the composition property, a set of basic rotation/quaternions can be defined:

$$\begin{aligned}
&\text{Rotation around the } \mathbf{X} \text{ axis} \quad \leftrightarrow \quad \mathbf{T}_1(\phi) \quad \leftrightarrow \quad \mathfrak{q}_1(\phi) \quad = \left(\cos\frac{\phi}{2}, \sin\frac{\phi}{2}, 0, 0\right) \\
&\text{Rotation around the } \mathbf{Y} \text{ axis} \quad \leftrightarrow \quad \mathbf{T}_2(\theta) \quad \leftrightarrow \quad \mathfrak{q}_2(\theta) \quad = \left(\cos\frac{\theta}{2}, 0, \sin\frac{\theta}{2}, 0\right) \\
&\text{Rotation around the } \mathbf{Z} \text{ axis} \quad \leftrightarrow \quad \mathbf{T}_3(\psi) \quad \leftrightarrow \quad \mathfrak{q}_3(\psi) \quad = \left(\cos\frac{\psi}{2}, 0, 0, \sin\frac{\psi}{2}\right)
\end{aligned}$$

These are called the *elementary* quaternions.

Quaternion Kinematics

Consider a rigid body rotating with respect to some observer Reference Frame (RF) (fixed), with angular velocity $\boldsymbol{\omega} = \omega_1\mathbf{b}_1 + \omega_2\mathbf{b}_2 + \omega_3\mathbf{b}_3$ expressed in the body RF (origin in the center of mass of the body, rotating with it).

The goal is to describe the time evolution of the rotation quaternion $\mathfrak{q}$ (corresponding to the body RF) as a function of $\omega_1, \omega_2, \omega_3$. Note that both the quaternion and the angular velocity change in time:

$$\begin{aligned}
&\mathfrak{q} \equiv \mathfrak{q}(t) \\
&\boldsymbol{\omega} = (\omega_1, \omega_2, \omega_3) \equiv \boldsymbol{\omega}(t).
\end{aligned}$$

At time $t + \Delta t$, we have the rotation $\mathfrak{q}(t)$ at time t composed with the rotation $\Delta\mathfrak{q}(t)$ occurred from time t to time $t + \Delta t$. The quaternion at time $t + \Delta t$ is thus given by

$$\mathfrak{q}(t + \Delta t) = \mathfrak{q}(t) \otimes \Delta\mathfrak{q}(t).$$

Let $\omega = |\boldsymbol{\omega}|$ be the angular speed magnitude; then, for a small Δt, the rotation angle is $\omega\Delta t$. Let $\mathbf{u}$ be the rotation axis, with $|\mathbf{u}| = 1$, then $\boldsymbol{\omega} = \omega\mathbf{u}$. So, for small Δt,

$$\Delta\mathfrak{q} \cong \begin{bmatrix} \cos\frac{\omega\Delta t}{2} \\ \mathbf{u}\sin\frac{\omega\Delta t}{2} \end{bmatrix} \cong \begin{bmatrix} 1 \\ \mathbf{u}\frac{\omega\Delta t}{2} \end{bmatrix} = \begin{bmatrix} 1 \\ \frac{\boldsymbol{\omega}\Delta t}{2} \end{bmatrix}$$

The quaternion derivative is thus given by

$$\begin{aligned}
\dot{\mathfrak{q}} &= \lim_{\Delta t\to 0} \frac{\mathfrak{q}(t + \Delta t) - \mathfrak{q}(t)}{\Delta t} = \lim_{\Delta t\to 0} \frac{\mathfrak{q} \otimes \Delta\mathfrak{q} - \mathfrak{q}}{\Delta t} = \\
&= \lim_{\Delta t\to 0} \frac{\mathfrak{q} \otimes (\Delta\mathfrak{q} - (1, \mathbf{0}))}{\Delta t} = \lim_{\Delta t\to 0} \frac{\mathfrak{q} \otimes \left(\left(1, \frac{\boldsymbol{\omega}\Delta t}{2}\right) - (1, \mathbf{0})\right)}{\Delta t} = \\
&= \lim_{\Delta t\to 0} \frac{\mathfrak{q} \otimes \left(0, \frac{\boldsymbol{\omega}\Delta t}{2}\right)}{\Delta t} = \frac{1}{2}\mathfrak{q} \otimes (0, \boldsymbol{\omega})
\end{aligned}$$

and it is usually referred to as *quaternion kinematic equations*. Equivalent equations are the following:

$$\dot{\mathfrak{q}} = \frac{1}{2}\boldsymbol{\Omega}\mathfrak{q} \qquad \dot{\mathfrak{q}} = \frac{1}{2}\mathbf{Q}\boldsymbol{\omega}$$

where

$$\boldsymbol{\Omega} \doteq \begin{bmatrix} 0 & -\omega_1 & -\omega_2 & -\omega_3 \\ \omega_1 & 0 & \omega_3 & -\omega_2 \\ \omega_2 & -\omega_3 & 0 & \omega_1 \\ \omega_3 & \omega_2 & -\omega_1 & 0 \end{bmatrix}, \quad \mathbf{Q} \doteq \begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{bmatrix}$$

Using quaternions, no singularities such as the *gimbal lock* can occur, and this is a huge advantage of using quaternions in kinematics applications.

Appendix C

IPOPT solver for large-scale non-linear optimization

The general non-linear programming problem can be formulated as follows:

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} f(x) \\ &s.t. \qquad g^L \leq g(x) \leq g^U \\ &\qquad\quad x^L \leq x \leq x^U \end{aligned} \tag{C.1}$$

In this formulation, the optimization variables are represented by $x \in \mathbb{R}^n$, and they are bounded within the intervals $x^L \in (\mathbb{R} \cup -\infty)^n$ and $x^U \in (\mathbb{R} \cup +\infty)^n$. The objective function to be minimized is denoted as $f(\cdot) : \mathbb{R}^n \longrightarrow \mathbb{R}$. Additionally, there are the constraints defined by $g(\cdot) : \mathbb{R}^n \longrightarrow \mathbb{R}^m$, subject to lower and upper bounds g^L and g^U.

It's worth noting that equality constraints $g_i(x) = \bar{g}_i$ can also be accommodated within this framework by setting $g_i^L = g_i^U = \bar{g}_i$. Furthermore, the functions $f(x)$ and $g(x)$ should exhibit sufficient smoothness, typically requiring at least once or twice differentiability. However, they may take on linear or non-linear forms and exhibit convex or non-convex characteristics [45]. Convex functions, which are characterized by having level curves that define convex sets, are an extremely valuable class of functions. What makes them particularly useful is that for convex functions, any local minimum is surely a global minimum. As a result, when dealing with convex functions, numerical algorithms can be relied on to find a global minimum.

Interior Point OPTimizer (IPOPT) is an open-source software package for numerically solving large non-linear optimization problems. In particular, it is suitable for large problems with a Jacobian matrix of constraint function sufficiently sparse.

IPOPT implements an *interior-point line-search filter method*. In this paragraph, I will present only an idea of the algorithm (please refer to [45] for further details).

1. Firstly the inequality constraints are substituted by an equality constraint with a new slack variable (e.g. $g_i(x) - s(i) = 0, \quad g_i^L \leq s_i \leq g_i^U$). So, the variables' bound

constraints remain the only inequality constraint (the variables x have been expanded with all the s_i). For simplicity purposes, let assume that x span in the range $[0, +\infty)$, so the problem is rewritten as

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} f(x) \\ &s.t. \qquad\quad c(x) = 0 \\ &\qquad\qquad\quad x \geq 0 \end{aligned} \tag{C.2}$$

where $c(x)$ represents the new equality constraints with the new slack variables.

2. IPOPT constructs the auxiliary barrier problem formulation as an *interior point* method

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} f(x) - \mu \sum_{i=1}^{n} \ln x_i \\ &s.t. \qquad\qquad\quad c(x) = 0 \end{aligned} \tag{C.3}$$

where the variable bound constraints $x \geq 0$ have been replaced by the logarithmic barrier term in the objective function. For any variable x_i, if $x_i \longrightarrow 0$, then $\ln x_i \longrightarrow -\infty$, and the objective function goes to infinity. So, the optimal solution to this problem must be in the interior of the region defined by the variables' bounds. $\mu > 0$ is called *barrier parameters* and controls the influence of the barrier term on the objective function. It can be proven that, under certain conditions, the optimal solution for this auxiliary problem converges to the solution of C.1 for $\mu \longrightarrow 0$.

   2.1. IPOPT solves a first barrier problem with a moderate value of μ (e.g., 0.1) starting from a user-supplied starting point, with a relaxed accuracy.

   2.2. The previous solution is used as the starting point of a new iteration, with a tighter accuracy, and a lower value of μ.

   2.3. Repeat 2.2. until a solution for the problem in Eq. C.1, or at least a point satisfying the first-order optimality conditions up to user tolerances, has been found[45].

It is worth noting that this algorithm, along with various other methods, is intended to find the global minimum of the problem. However, in practice, it often converges to a local minimum (in a finite time). In non-convex problems, numerous stationary points may exist, resulting in multiple minima associated with different objective function values. For this reason, the outcome of an optimization algorithm depends on factors such as the initial starting point and the maximum number of iterations (or time allocated) [46]. Surely finding the global maximum of a non-convex objective function may require an infinite amount of time for the algorithm to converge to the solution.

Appendix D

Introduction to ROS2

Robot Operating System 2 (ROS2) [1], as described in [47], emerges as a robust and powerful instrument for programming robots. In its essence, ROS2 operates as a middleware, functioning as an intermediary layer of software between the underlying operating system and the user's application (as illustrated in Fig. D.1). This versatile framework provides a broad spectrum of functionalities, such as drivers, libraries, development tools, integration capabilities, execution management, and advanced monitoring tools [48]. Despite the acronym "ROS" stands for "Robot Operating System," ROS2 does not replace traditional operating systems such as Linux or Windows; instead, it operates as a middleware layer built on top of them. The numerical label "2" signifies that we are engaging with the framework's second generation. The officially supported programming languages are C++ and Python, which are also the most used in Robotics contexts.

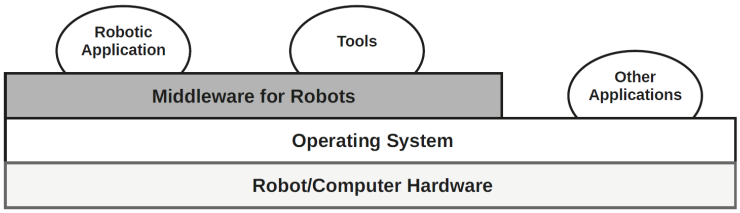


Figure D.1: Software layers in a robot. [48]

Given the importance of communication within the ROS2 framework, ROS2 has adopted the Data Distribution Service (DDS)[2] protocol, operating on top of UDP, as its communication layer. It allows processes to exchange information, providing them with real-time capabilities, robust security features, and the ability to customize the quality of service for each connection [48].

[1] `https://www.ros.org/`

[2] `https://www.omg.org/omg-dds-portal/`

DDS introduces a Pub/Sub communication paradigm, facilitating the discovery of publishers and subscribers without relying on a centralized service. This discovery mechanism uses multicast for initial discovery, subsequently transitioning to unicast connections. Notably, the DDS ecosystem comprises several vendors, each offering comprehensive or partial implementations of the DDS standard.

Nodes network

The core computational units within ROS2 are known as *Nodes*, and they dynamically interconnect during run-time to compose the application. Each node serves as an active component responsible for executing specific processing or control tasks. Moreover, since ROS2 makes intensive use of Objective-Oriented Programming, a node is an *object* of class `Node`, whether it is written in C++ or Python.

The execution of a node can be characterized in two distinct ways:

- **Iterative execution**: In this mode, the node sets up a timer with an associated callback responsible for performing a defined control task. This ensures that the callback is executed at a specific, predefined frequency, making it easier to manage the computational resources required by the node.

- **Event-oriented execution**: In this mode, the node associates a callback with asynchronous events, such as the reception of messages at that particular node. The node responds to events as they occur.

The collection of nodes and the communication channels connecting them form what is known as the Computation Graph.

For example, consider the scenario depicted in Fig. D.2. It shows an application that performs some tasks based on the preprocessed information (people and objects) obtained from a robot's RGBD camera. In this example:

- The *Control Application* node operates at a fixed frequency, utilizing iterative execution.

- *People Perception* and *Object Perception* nodes follow an event-oriented execution approach, processing each image as it arrives from the camera.

- The hardware sensors, such as the RGBD camera and Laser, provide ROS-compatible interfaces.

- Multiple nodes can be encapsulated within the same process, as observed in the *People Perception* and *Object Perception* nodes.

This setup exemplifies how ROS2 facilitates the development of robotic applications by orchestrating the interactions between nodes within the computation graph.

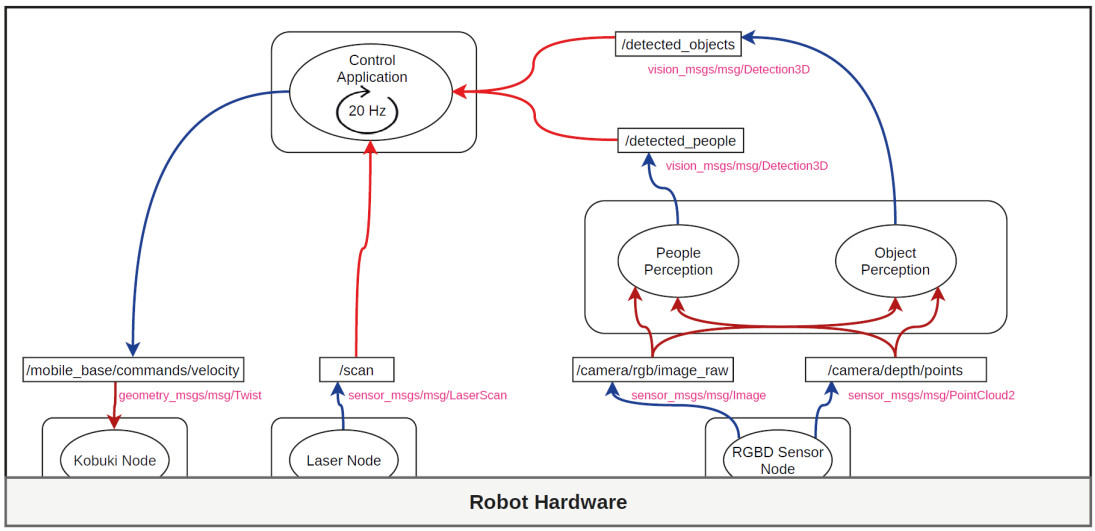


Figure D.2: Example of a Computation Graph. Each rounded rectangle represents an independent process, while the ellipses denote the nodes. Subscriptions to topics are depicted in red, and publications are shown in blue. Additionally, the rounded rectangles correspond to topics and the pink text within the rectangles denotes the corresponding message types associated with each topic. [48]

Topics, Services, Actions

In ROS2 each node can access the Computation Graph and communicate with other nodes. Three types of paradigms are provided by ROS2:

- **Publication/Subscription:** This is an asynchronous communication method where nodes publish messages to a topic that reaches its subscribers. A topic accepts messages of a unique, well-defined type, and supports multiple publishers and multiple subscribers connected simultaneously. Moreover, the communication is anonymous, since the subscribers do not generally know which is the correspondent publisher of the message. This is the most common communication paradigm in ROS2. For instance, a sensor like a camera can publish images in a topic, while the control algorithm subscribes to the same topic to use the images for its goal.

- **Services:** Services involve asynchronous communication in which a node makes a remote procedure call to another node which will do a computation and return a result. This type of communication usually requires an immediate response to avoid problems in the control cycle of the calling node. An example could be a request to the mapping service to reset a map, with a response indicating if the call succeeded.

- **Actions:** Actions are asynchronous communications in which a node makes a request to another node, and waits for a response without blocking state, as in the synchronous communication. These requests typically take time to complete, and the calling node may periodically receive feedback or notifications regarding its status — whether it

has finished successfully or encountered an error. The goal of the action can also be preempted or canceled. An example of this communication type is a navigation request, which could be time-consuming, requiring periodic updates about its progress.

TF2 subsystem

The TF2 [3] subsystem is the geometric transformation subsystem integrated into ROS2. It enables the definition of various Reference Frames (RFs) within the system and facilitates the continuous tracking of geometric relationships between them. Using this tool, any coordinate within one RF can be effortlessly recalculated in another. This subsystem plays a critical role in numerous applications, including navigation, localization, and manipulation.

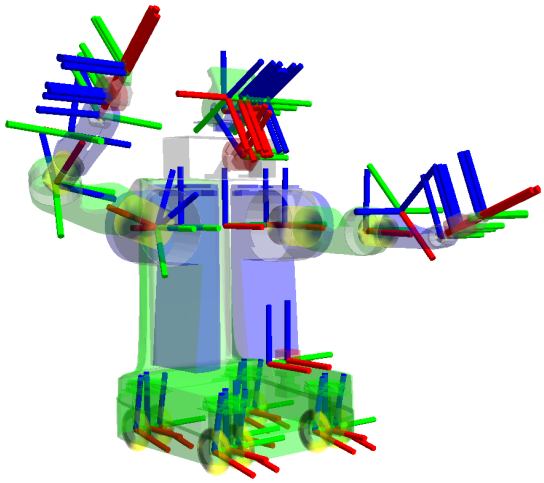


Figure D.3: Example of Reference Frames (RFs) in a robot.

The geometric relationship between RFs involves the application of rotation and translation operations from one RF to another. Algebraically, this operation is performed in homogeneous coordinates using the appropriate transformation matrix. Given the coordinate vector $\boldsymbol{p}^A = (x_A, y_A, z_A)$ of a point in RF A and the transformation matrix T_B^A from A to B, it is possible to calculate $\boldsymbol{p}^B = (x_B, y_B, z_B)$ as

$$\begin{gathered} \tilde{\boldsymbol{p}}^B = \boldsymbol{T}_A^B \cdot \tilde{\boldsymbol{p}}^A \\ \begin{bmatrix} x_B \\ y_B \\ z_B \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_A^B & \boldsymbol{t}_A^B \\ \boldsymbol{0} & 1 \end{bmatrix} \cdot \begin{bmatrix} x_A \\ y_A \\ z_A \\ 1 \end{bmatrix} \end{gathered} \tag{D.1}$$

where

[3] https://docs.ros.org/en/humble/Concepts/Intermediate/About-Tf2.html

- The tilde highlights that the vector is expressed in homogeneous coordinates.
- $\boldsymbol{R}_A^B$ is the rotation matrix 3×3 from RF A to RF B.
- $\boldsymbol{R}_A^B$ is the translation vector 3×1 from the origin of RF A to the one of RF B.

Moreover, in addition to the complexity of these operations, it is remarkable that these relationships are highly dynamic in an articulated robot[48].

TF2 organizes RFs in a tree structure, where each RF should have at most one parent but can have several children. An error will be raised if a RF has more than one parent, although no error occurs if several trees are not connected. However, it is advisable to avoid this situation as it could lead to run-time errors and may indicate that the robot's model is not accurate.

TF2 can operate within a distributed system, making all information about a robot's coordinate RFs available to all ROS2 components in the system. Additionally, TF2 allows each component in a distributed system to build its own transformation information database or utilize a central node to gather and store all transform information.

When a node needs to access transformation information, it uses `TFListeners`. These objects maintain a buffer containing the latest published TFs and provide an API for:

- Determining if there is a TF transform from one RF to another at time t.
- Obtaining the rotation or translation between two RFs at time t.
- Transforming a coordinate vector from one RF to another at time t.

The buffer may not only contain the TF at time t, but if it has earlier and later TFs, it performs interpolation. Similarly, RFs A and B may not be directly connected, but if there are additional RFs in between, TF2 automatically handles the necessary matrix operations.

Behavior Trees

A Behavior Tree (BT) is a mathematical model used to encode the high-level logic of an application. In recent years, BT have gained popularity across various applications, especially in video games and robot control [49]. These structures are usually compared to Finite State Machines (FSMs), their closest competitors, although they have distinct characteristics. While FSMs rely on state and transition concepts, BT necessitate thinking in terms of sequences, fallbacks, and similar constructs.

In a general sense, a tree represents a hierarchical data structure that is recursively defined, originating from a root node and branching into multiple child nodes. Each child node can further branch into additional children, creating a tree-like structure. Nodes without children are typically referred to as *leaves* of the tree.

The fundamental operation of a BT is the **tick**, which propagates from the root to the first active leaves. When a node is ticked, it returns one of the following three values:

- **SUCCESS**: The node has completed its task successfully.

- **RUNNING**: The node has not yet completed its task.
- **FAILURE**: The mission is failed.

Furthermore, all the BT nodes can be categorized into four distinct classes:

- **Control**: Control nodes distribute the tick to their children according to a predefined rule. For example, the `Sequence` node is a control node that propagates the tick to its children in sequence until a child returns FAILURE or all children have been completed.
- **Decorators**: Decorators are control nodes with only one child. For instance, nodes that regulate the execution rate of their child nodes are decorators.
- **Action**: Action nodes implement specific tasks within the application and are positioned at the leaves of the tree.
- **Condition**: Condition nodes are action nodes that can only return SUCCESS if the associated condition is met or FAILURE otherwise. They cannot return RUNNING.

The library of existing nodes can also be easily extended with custom nodes created by the users.

Finally, a BT incorporates a *blackboard*, a key/value storage that can be accessed from all the nodes in the tree. The blackboard is important for facilitating the exchange of information between nodes.

With these functionalities, users can implement a vast array of logic applications. A quite standard implementation of these concepts is found in the Behavior Trees library, behaviourtree.CPP[4], where BTs are specified in the XML format. An alternative Python implementation is provided by the py-tree library[5].

Executors

Given that ROS2 operates as an event-based middleware, it is important to highlight how the system responds to events. Specifically, the orchestration of node execution is handled by `Executors`, which harness one or more threads provided by the underlying operating system to invoke and manage callbacks associated with timers, subscriptions, events, and more.

The most straightforward approach is the `Single-Threaded Executor`, where a single thread processes messages and events sequentially. The thread queries incoming events, invoking the corresponding callbacks until the node completes its tasks. An alternative is the `Static Single-Threaded Executor`, which optimizes the scans of the node's structure by performing them only once during node addition.

On the other hand, the `Multi-Threaded Executor` introduces a level of parallelism by creating a variable number of threads to facilitate concurrent processing of multiple

[4] `https://www.behaviortree.dev/`

[5] `https://py-trees.readthedocs.io/en/release-2.2.x/`

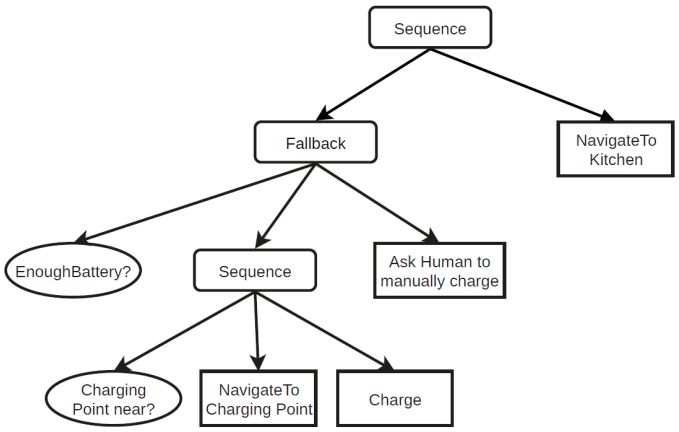


Figure D.4: Example of Behavior Tree (BT) with a fallback strategy for charging battery. The rounded rectangles denote the control nodes, the circles represent the condition nodes, and the rectangles correspond to the action nodes. [48]

messages or events. The extent of parallelism within the callback execution hinges on the callback group to which they belong.

Two distinct types of callback groups exist, each requiring specification upon instantiation:

- **Mutually Exclusive:** Callbacks within this group are explicitly disallowed from concurrent execution.

- **Reentrant:** Callbacks associated with this group are permitted to execute concurrently.

Importantly, callbacks from different callback groups may always be executed in parallel. The `Multi-Threaded Executor` employs its threads as a resource pool to maximize the parallel processing of callbacks, all contingent upon these defined conditions.

When the processing duration of callbacks is shorter than the interval between message and event occurrences, the Executor typically processes them in a first-in, first-out (FIFO) order. However, if certain callbacks exhibit longer processing times, messages and events begin to queue up within the lower stack layers. The Executor is informed only about the presence or absence of messages for specific topics; it uses this information to process the messages (including services and actions) in a round-robin fashion, although not strictly adhering to FIFO ordering.

Despite their versatility, the provided executors might not meet the stringent demands of real-time applications. These applications require predictable execution times, deterministic behavior, and fine-grained control over execution order. Challenges include complex scheduling semantics, potential priority issues, and limited execution control. Finally, Executors also introduce a notable overhead in CPU and memory usage.

Final remarks

In conclusion, ROS2 represents a significant evolution from its predecessor, ROS1, with improved real-time capabilities, enhanced security, and cross-platform compatibility. Its unique combination of flexibility, open-source accessibility, and the support of a collaborative community make ROS2 an indispensable tool for the development of resilient and scalable robotic systems.

Nevertheless, ROS2 is dealing with ongoing challenges with real-time communication, particularly when dealing with heavy payload messages. This critical issue has garnered the attention of researchers and developers, as evidenced by the innovative solutions presented in the field, such as Composition [50]. Furthermore, the promising integration of GPU computing holds the potential to further improve ROS2 to new levels of efficiency and performance.

Bibliography

[1] Z. Zhai, J. F. Martínez, V. Beltran, and N. L. Martínez, «Decision support systems for agriculture 4.0: Survey and challenges», *Computers and Electronics in Agriculture*, vol. 170, p. 105 256, 2020, ISSN: 0168-1699. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169919316497 (cit. on p. 1).

[2] D. Bigelow and A. Borchers, «Major Uses of Land in the United States, 2012», Economic Information Bulletin Number 178, no. 1476-2017-4340, p. 69, 2017. [Online]. Available: http://ageconsearch.umn.edu/record/263079 (cit. on p. 1).

[3] W. Winterhalter, F. Fleckenstein, C. Dornhege, and W. Burgard, «Localization for precision navigation in agricultural fields—Beyond crop row following», *Journal of Field Robotics*, vol. 38, no. 3, pp. 429–451, [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21995 (cit. on p. 1).

[4] F. Salvetti, S. Angarano, M. Martini, S. Cerrato, and M. Chiaberge, «Waypoint Generation in Row-Based Crops with Deep Learning and Contrastive Clustering», in *Machine Learning and Knowledge Discovery in Databases*, Springer Nature Switzerland, 2023, pp. 203–218. [Online]. Available: https://doi.org/10.1007%2F978-3-031-26422-1_13 (cit. on p. 1).

[5] C. W. Bac, E. J. van Henten, J. Hemming, and Y. Edan, «Harvesting Robots for High-value Crops: State-of-the-art Review and Challenges Ahead», *Journal of Field Robotics*, vol. 31, no. 6, pp. 888–911, 2014. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21525 (cit. on p. 1).

[6] M. Kabir *et al.*, «Performance Comparison of Single and Multi-GNSS Receivers under Agricultural Fields in Korea», *Engineering in Agriculture, Environment and Food*, vol. 9, pp. 27–35, Feb. 2016 (cit. on p. 4).

[7] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT press, 2011 (cit. on p. 4).

[8] S. Zaman *et al.*, «Cost-effective visual odometry system for vehicle motion control in agricultural environments», *Computers and Electronics in Agriculture*, vol. 162, pp. 82–94, 2019, ISSN: 0168-1699. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169918317058 (cit. on pp. 4–6).

[9] M. Martini, S. Cerrato, F. Salvetti, S. Angarano, and M. Chiaberge, «Position-Agnostic Autonomous Navigation in Vineyards with Deep Reinforcement Learning», in *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, IEEE, Aug. 2022. [Online]. Available: https://doi.org/10.1109%2Fcase49997.2022.9926582 (cit. on pp. 7, 10).

[10] A. Navone, M. Martini, A. Ostuni, S. Angarano, and M. Chiaberge, *Autonomous Navigation in Rows of Trees and High Crops with Deep Semantic Segmentation*, 2023 (cit. on pp. 7, 8, 10, 63).

[11] D. Aghi, S. Cerrato, V. Mazzia, and M. Chiaberge, «Deep Semantic Segmentation at the Edge for Autonomous Navigation in Vineyard Rows», in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, Sep. 2021. [Online]. Available: https://doi.org/10.1109%2Firos51168.2021.9635969 (cit. on pp. 7, 8, 10).

[12] A. Howard *et al.*, «Searching for mobilenetv3», in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1314–1324 (cit. on p. 7).

[13] J. Radcliffe, J. Cox, and D. M. Bulanon, «Machine vision for orchard navigation», *Computers in Industry*, vol. 98, pp. 165–171, 2018, ISSN: 0166-3615. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166361517305389 (cit. on pp. 7, 8, 10).

[14] P. Huang, L. Zhu, Z. Zhang, and C. Yang, «An End-to-End Learning-Based Row-Following System for an Agricultural Robot in Structured Apple Orchards», *Mathematical Problems in Engineering*, vol. 2021, pp. 1–14, Sep. 2021. [Online]. Available: https://ideas.repec.org/a/hin/jnlmpe/6221119.html (cit. on pp. 8, 10).

[15] M. Sharifi and X. Chen, «A novel vision based row guidance approach for navigation of agricultural mobile robots in orchards», in *2015 6th International Conference on Automation, Robotics and Applications (ICARA)*, 2015, pp. 251–255 (cit. on pp. 9, 10).

[16] A. Villemazet, A. Durand-Petiteville, and V. Cadenat, «Multi-Camera GPS-Free Nonlinear Model Predictive Control Strategy to Traverse Orchards», in *2023 European Conference on Mobile Robots (ECMR)*, IEEE, 2023, pp. 1–7 (cit. on pp. 9, 10).

[17] I. Aslan Seyhan, «Mathematical Instruments Commonly Used among the Ottomans», *Advances in Historical Studies*, vol. 08, pp. 36–57, Jan. 2019 (cit. on p. 12).

[18] P. Corke, *Robotics, Vision and Control - Fundamental Algorithms in MATLAB®* (Springer Tracts in Advanced Robotics), First. Springer Berlin, Heidelberg, 2011, vol. 73. [Online]. Available: https://doi.org/10.1007/978-3-642-20144-8 (cit. on pp. 13, 16–19, 21, 24, 26–28, 32, 33, 85).

[19] D. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*, Second. Prentice Hall, Nov. 2011 (cit. on pp. 16, 28, 30).

[20] G. Bradski, «The OpenCV Library», *Dr. Dobb's Journal of Software Tools*, 2000 (cit. on pp. 27, 28).

[21] L. Tang, Z.-X. Yang, and K. Jia, «Canonical Correlation Analysis Regularization: An Effective Deep Multiview Learning Baseline for RGB-D Object Recognition», *IEEE Transactions on Cognitive and Developmental Systems*, vol. 11, no. 1, pp. 107–118, 2019 (cit. on p. 28).

[22] R. Szeliski, *Computer vision: algorithms and applications*. Springer Nature, 2022 (cit. on p. 28).

[23] E. Ahmed *et al.*, *A survey on Deep Learning Advances on Different 3D Data Representations*, 2019 (cit. on pp. 35, 36).

[24] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, «Geometric Deep Learning: Going beyond Euclidean data», *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, Jul. 2017. [Online]. Available: `https://doi.org/10.1109%2Fmsp.2017.2693418` (cit. on p. 35).

[25] S. A. Bello, S. Yu, C. Wang, J. M. Adam, and J. Li, «Review: Deep Learning on 3D Point Clouds», *Remote Sensing*, vol. 12, no. 11, 2020, ISSN: 2072-4292. [Online]. Available: `https://www.mdpi.com/2072-4292/12/11/1729` (cit. on pp. 35, 36).

[26] E. C. Ilas, «Self-Supervised Deep Learning via Colorization on 3D Point Clouds for Object Part Segmentation», M.S. thesis, Politecnico di Torino, Torino, Dec. 2021 (cit. on p. 35).

[27] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*, 2017 (cit. on p. 35).

[28] X.-F. Han, J. S. Jin, M.-J. Wang, and W. Jiang, «Guided 3D point cloud filtering», *Multimedia Tools and Applications*, vol. 77, no. 13, pp. 17 397–17 411, Jul. 2018, ISSN: 1573-7721. [Online]. Available: `https://doi.org/10.1007/s11042-017-5310-9` (cit. on p. 35).

[29] *Open3D, A Modern Library for 3D Data Processing*. [Online]. Available: `http://www.open3d.org/` (cit. on pp. 37, 38).

[30] G. Fracastoro, *Lecture notes in Optimization for Machine Learning*, Politecnico di Torino, 2022 (cit. on p. 37).

[31] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, «A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise», in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96, Portland, Oregon: AAAI Press, 1996, pp. 226–231 (cit. on p. 38).

[32] H. Sarbolandi, D. Lefloch, and A. Kolb, *Kinect Range Sensing: Structured-Light versus Time-of-Flight Kinect*, 2015 (cit. on p. 39).

[33] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics* (Advanced Textbooks in Control and Signal Processing), First. Springer London, 2009. [Online]. Available: `https://doi.org/10.1007/978-1-84628-642-1` (cit. on pp. 42, 44, 46).

[34] A. Rizzo, *Lecture notes in Robotics*, Politecnico di Torino, 2022 (cit. on pp. 42, 45, 47).

[35] C. Novara, *Lecture notes in Nonlinear control and aereospace applications*, Politecnico di Torino, 2022 (cit. on pp. 51, 54, 87).

[36] A. Canale, *Lecture notes in Digital Control Technologies and Architectures*, Politecnico di Torino, 2021 (cit. on pp. 52, 54).

[37] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, «Aggressive driving with model predictive path integral control», in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1433–1440 (cit. on pp. 54–56).

[38] F. Fiedler *et al.*, «do-mpc: Towards FAIR nonlinear and robust model predictive control», *Control Engineering Practice*, vol. 140, p. 105 676, 2023, ISSN: 0967-0661. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0967066123002459 (cit. on p. 63).

[39] *Jackal datasheet*, Clearpath Robotics, 2020. [Online]. Available: https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/ (cit. on p. 71).

[40] *Husky datasheet*, Clearpath Robotics, 2022. [Online]. Available: https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/ (cit. on p. 72).

[41] *Intel Realsense Prouct Family D400 Series datasheet*, Intel Realsense, 2020. [Online]. Available: https://www.intelrealsense.com/depth-camera-d455/ (cit. on pp. 73, 74).

[42] M. Martini, A. Eirale, S. Cerrato, and M. Chiaberge, *PIC4rl-gym: a ROS2 modular framework for Robots Autonomous Navigation with Deep Reinforcement Learning*, 2022 (cit. on p. 75).

[43] I. Vizzo *et al.*, «KISS-ICP: In Defense of Point-to-Point ICP – Simple, Accurate, and Robust Registration If Done the Right Way», *IEEE Robotics and Automation Letters*, vol. 8, no. 2, pp. 1029–1036, 2023 (cit. on p. 77).

[44] A. Proskurnikov, *Lecture notes in Modelling and Simulaion of Mechatronic Systems*, Politecnico di Torino, 2021 (cit. on p. 87).

[45] A. Wächter, «Short Tutorial: Getting Started With Ipopt in 90 Minutes», in *Combinatorial Scientific Computing*, U. Naumann, O. Schenk, H. D. Simon, and S. Toledo, Eds., ser. Dagstuhl Seminar Proceedings (DagSemProc), vol. 9061, Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2009, pp. 1–17. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2009/2089 (cit. on pp. 91, 92).

[46] M. Ghirardi, *Lecture notes in Software architecture for Automation*, Politecnico di Torino, 2022 (cit. on p. 92).

[47] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, «Robot Operating System 2: Design, architecture, and uses in the wild», *Science Robotics*, vol. 7, no. 66, eabm6074, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074 (cit. on p. 93).

[48] F. M. Rico, *A Concise Introduction to Robot Programming with ROS2*. Chapman and Hall/CRC, Sep. 2022. [Online]. Available: https://doi.org/10.1201/9781003289623 (cit. on pp. 93, 95, 97, 99).

[49] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ögren, «Towards a unified behavior trees framework for robot control», in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 5420–5427 (cit. on p. 97).

[50] S. Macenski, A. Soragna, M. Carroll, and Z. Ge, «Impact of ROS 2 Node Composition in Robotic Systems», *IEEE Robotics and Autonomous Letters (RA-L)*, 2023. [Online]. Available: `https://arxiv.org/abs/2305.09933` (cit. on p. 100).