ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN

INGEGNERIA BIOMEDICA, ELETTRICA E DEI SISTEMI

Ciclo 37

Settore Concorsuale: 09/G1 - AUTOMATICA

Settore Scientifico Disciplinare: ING-INF/04 - AUTOMATICA

DESIGN AND CONTROL OF A ROBOTIC MANIPULATOR FOR KIWI HARVESTING

Presentata da: Simone Rossi

Coordinatore Dottorato

Michele Monaci

Supervisore

Lorenzo Marconi

Esame finale anno 2025

This page intentionally left blank.

Abstract

This thesis presents the design and control of a robotic manipulator intended for kiwi harvesting, addressing the increasing need for automation in agriculture due to labour shortages and the pursuit of precision farming. The research introduces a novel, low-cost robotic solution integrated with a mobile platform to automate the harvesting process in kiwi orchards. The system involves the design of a custom anthropomorphic manipulator, including actuator selection, gripper design, and mobile platform integration, specifically adapted for the kiwi harvesting task.

The control architecture incorporates a combination of low-level motion control on an Arduino-based platform and high-level trajectory planning using ROS 2 and MoveIt 2, allowing for efficient navigation and adaptability in an orchard environment. The perception system combines depth and RGB cameras to detect and localize fruits with high precision, enhancing the robot's ability to autonomously identify, approach, and pick kiwi fruits.

The effectiveness of the system is evaluated through simulations and field experiments, demonstrating its capability to harvest kiwis effectively while maintaining fruit quality. Future work aims to address challenges related to system robustness in diverse environmental conditions and further improve the adaptability of the harvesting mechanism.

This page intentionally left blank.

Sommario

Questa tesi presenta la progettazione e il controllo di un manipolatore robotico destinato alla raccolta di kiwi, affrontando la crescente necessità di automazione nell'agricoltura a causa della carenza di manodopera e della ricerca di soluzioni per l'agricoltura di precisione. La ricerca introduce una soluzione robotica innovativa e a basso costo integrata con una piattaforma mobile per automatizzare il processo di raccolta nei frutteti di kiwi. Il sistema comprende la progettazione di un manipolatore antropomorfo su misura, inclusa la selezione degli attuatori, il design del gripper e l'integrazione con la piattaforma mobile, adattato specificamente al compito di raccolta del kiwi.

L'architettura di controllo combina il controllo del movimento a basso livello su una piattaforma basata su Arduino con una pianificazione delle traiettorie ad alto livello tramite ROS 2 e MoveIt 2, consentendo una navigazione efficiente e un'elevata adattabilità all'ambiente del frutteto. Il sistema di percezione combina telecamere di profondità e RGB per rilevare e localizzare i frutti con alta precisione, migliorando la capacità del robot di identificare, avvicinarsi e raccogliere autonomamente i kiwi.

L'efficacia del sistema è stata valutata attraverso simulazioni ed esperimenti sul campo, dimostrando la capacità del manipolatore di raccogliere i kiwi in modo efficace preservando la qualità del frutto. I risultati indicano che il manipolatore proposto può migliorare l'efficienza della raccolta dei kiwi, rappresentando una promettente via verso soluzioni agricole completamente automatizzate. Il lavoro futuro mira ad affrontare le sfide legate alla robustezza del sistema in diverse condizioni ambientali e a migliorare ulteriormente l'adattabilità del meccanismo di raccolta.

This page intentionally left blank.

Contents

This page intentionally left blank.

List of Figures

This page intentionally left blank.

List of Tables

This page intentionally left blank.

1

Introduction

In this chapter, we explore the current landscape of agricultural robotics, with a specific focus on the advancements in robotics for fruit harvesting. The agricultural sector has increasingly turned to automation to address labour shortages, efficiency challenges, and the need for precision farming solutions. Within this context, robotics has emerged as a promising avenue to enhance productivity, especially in the complex task of fruit harvesting. This chapter provides a comprehensive review of the state of the art, highlighting key technologies, challenges, and solutions currently being employed in the field. Furthermore, an overview of the contributions made in this research is presented, outlining the goals, methodologies, and anticipated impact of the work carried out.

1.1 Robotics in agriculture

1.1.1 History

The journey of robotics in agriculture began in the late 20th century, driven largely by advancements in mechanical engineering, control systems, and computing power. Early attempts at automation involved mechanized tools that enhanced productivity, such as tractor-mounted implements that reduced manual labour in plowing and sowing (Blackmore, 2000). By the 1990s, the agricultural industry started witnessing the first wave of actual robotic systems in the form of semi-autonomous harvesters and mobile robots for repetitive tasks, often drawing on technologies borrowed from industrial robotics (Giles, 1998).

Since then, agricultural robotics has evolved rapidly, incorporating increasingly sophisticated sensors, artificial intelligence, and machine learning to achieve precision and autonomy. The adoption of GPS guidance systems and advanced imaging in the early 2000s laid the foundation for fully autonomous machines, such as self-driving tractors and drones used to monitor crop health (Gebbers et al., 2010). Today, modern agricultural robots leverage cutting-edge AI technologies, enabling them to perform highly complex functions, from soil preparation to harvesting, with minimal human intervention.

1.1.2 Motivations

The primary motivations behind the development of robotics in agriculture stem from the need to increase efficiency, enhance productivity, and address labour shortages. Agriculture has long been a labour-intensive industry, where large areas need timely cultivation, and there is a constant need to manage pests, diseases, and the fluctuating weather conditions that impact yield (Bechar et al., 2016). In many regions, an ageing farming population and an insufficient workforce have put additional pressure on the agricultural sector to automate.

The integration of robotics promises to significantly reduce the reliance on manual labour and mitigate the effects of labour scarcity. Additionally, the drive towards sustainability has fuelled innovation in agricultural robotics. By enabling precise applications of water, fertilizers, and pesticides, robots help reduce environmental impact and contribute to more sustainable farming practices (Pedersen et al., 2006). Thus, the use of robotics not only addresses workforce constraints but also contributes to more environmentally friendly and efficient farming systems.

1.1.3 Applications

Robotics is now widely applied across various facets of agriculture, ranging from soil analysis and crop monitoring to precision weeding and harvesting. Autonomous tractors are perhaps one of the most notable examples, where advanced GPS and sensor technologies allow them to till fields with high precision, reducing fuel consumption and improving yields. Another important application is in crop health monitoring, where drones equipped with multispectral cameras are deployed to assess plant health, detect diseases, and guide targeted interventions (C. Zhang et al., 2012).

Harvesting robots have also become highly specialized. For example, strawberry-picking robots use cameras and sensors to determine ripeness and selectively harvest fruits without damaging the plants (E. Van Henten et al., 2002). Similarly, robotic systems designed for pruning, weeding, and even pollination are becoming more common, especially in high-value crops such as vineyards and orchards (Lehnert et al., 2017). Livestock farming has also benefited, with autonomous feeding systems and robotic milking machines that enhance productivity while improving animal welfare.

The future of agricultural robotics is likely to see even greater collaboration between AI and autonomous systems, creating farms where robots work seamlessly alongside humans to optimize every aspect of production. This evolution aims to not only improve productivity and sustainability but also to make farming more resilient to the challenges of climate change and an expanding global population.

1.2 State of the art in robotic fruit harvesting

In recent years, robotic systems for agricultural harvesting have advanced significantly, driven by the need for labour efficiency and increased productivity. Robotic fruit har-

Figure 1.1: *Sweeper robot*

vesters offer a promising solution to the growing challenges in agriculture, including labour shortages, rising costs, and the increasing global demand for food. Today, several real-world examples showcase the capabilities and limitations of robotic fruit harvesting technologies, highlighting ongoing developments in machine learning, sensor technology, and robotics.

One notable example is the SWEEPER robot (Figure 1.1), developed for sweet pepper harvesting. The SWEEPER system was designed as a collaboration between universities and research institutions across Europe. It utilizes a combination of RGB cameras and laser sensors to identify ripe peppers and a robotic arm equipped with a specialized end-effector to cut and collect them. The SWEEPER robot demonstrated significant progress in terms of accuracy and efficiency, achieving a harvest rate of approximately 60% of ripe fruit under laboratory conditions (Bac et al., 2017). Although effective in controlled environments, its limitations became evident in practical field conditions due to challenges related to foliage occlusion, lighting variability, and the precision needed to distinguish ripe from unripe fruits.

Another advanced robotic harvesting system is the Octinion strawberry harvester (Figure 1.2a), which utilizes soft robotic technology to gently pick strawberries without causing damage. The Octinion robot, called "Rubion," integrates machine vision to locate ripe strawberries and uses a soft, rotating gripper to detach the fruit. Unlike traditional clamping mechanisms, the soft gripper minimizes bruising, a crucial factor for strawberries, which are highly perishable. The Octinion system is capable of picking a strawberry every four to five seconds, which translates to an efficiency that is comparable to a human picker, with the added benefit of 24/7 operation under the right conditions (E. J. Van Henten et al., 2019).

Apple harvesting presents a more complex challenge due to the size, weight, and height variability of apple trees. FFRobotics is one company making strides in this

(**a**) *Octinion robot* (**b**) *Tevel's Robot*

Figure 1.2: *Examples of fruit picking robots*

area with a multi-arm robotic harvester capable of picking up to ten apples simultaneously. Their robotic system employs a combination of computer vision for fruit detection and multiple articulated arms that mimic human movements. This design ensures a balance between speed and delicacy, avoiding bruising or damaging the fruit. FFRobotics aims for a harvest rate that can rival human pickers in commercial orchards, and while initial results are promising, the system requires highly controlled environments to achieve optimum results (Shamshiri et al., 2018).

In addition, Abundant Robotics, a U.S.-based startup, has developed an apple-picking robot that uses a vacuum-based approach to gently suck apples from the trees. Unlike other mechanical pickers, the vacuum system reduces the risk of fruit damage while also simplifying the process of detachment. Field trials conducted in partnership with commercial apple growers have demonstrated the potential of this approach to achieve high throughput while maintaining the quality of harvested apples. The system uses advanced computer vision and AI algorithms to determine ripeness and ideal picking paths, allowing for more autonomous operation in commercial orchards (Kootstra et al., 2020).

Tevel Aerobotics (Tevel Aerobotics, n.d.) has introduced an innovative approach to fruit harvesting through the use of autonomous flying robots (AFRs) (Figure 1.2b). These small drone-like harvesters are equipped with computer vision, AI-powered decision-making, and robotic arms capable of picking fruits such as apples, peaches, and plums. The system is designed to operate collaboratively, with multiple AFRs working together to identify, assess ripeness, and harvest fruit in real-time. By leveraging aerial mobility, Tevel Aerobotics' technology effectively addresses challenges related to tree canopy occlusion and terrain variability, making it a promising solution for large-scale orchards.

Another significant development in agricultural robotics is the SlopeHelper Harvester (PeK Automotive, n.d.), a fully autonomous robotic system designed specifically for fruit and vineyard harvesting in sloped and uneven terrains. Unlike conven-

tional harvesters, the SlopeHelper Harvester is equipped with a continuous operation system that enables it to carry out harvesting tasks without human intervention. The system integrates AI-driven navigation, LiDAR sensors, and an advanced gripping mechanism that ensures precise fruit picking while minimizing damage. Its ability to operate in challenging landscapes makes it a valuable solution for vineyards and orchards located on hilly terrains, where traditional machinery struggles. The SlopeHelper Harvester also features modular functionality, allowing it to be adapted for additional agricultural tasks such as pruning and spraying, further enhancing its versatility and cost-effectiveness.

Despite these advancements, current robotic fruit harvesting technologies still face significant challenges. Foliage occlusion, fruit clustering, environmental variability, and the need for quick adaptability to different crop types are major hurdles. Sensor fusion, where data from multiple sources such as RGB cameras, LiDAR, and hyperspectral sensors are combined, has emerged as an approach to address some of these challenges (Santos et al., 2021). Moreover, machine learning and AI continue to enhance the robots' abilities to identify and pick fruit more effectively by improving their recognition of various stages of ripeness and adapting to new crop conditions in real time.

The state of the art in robotic fruit harvesting shows a landscape of rapid technological evolution, but also highlights the limitations that must be addressed before full-scale commercial adoption becomes a reality. Researchers continue to work on improving the robustness of these systems under field conditions, increasing the precision and efficiency of fruit detection, and minimizing damage to the harvested produce. Moving forward, it is expected that the integration of AI, robotics, and sensor technologies will bring about smarter, more adaptable harvesting solutions that will transform agricultural practices in the near future.

1.3 The Hammerhead rover

A special mention deserves the UGV developed by the University of Bologna (Dario Mengoli et al., 2020) and then industrialised and commercialized by FieldRobotics: it is called Hammerhead and it is a multi-purpose, modular autonomous ground vehicle, made to navigate in orchards and, more in general, in agricultural environments (Figure 1.3). The platform was designed with adaptability in mind, enabling various agricultural tasks such as mowing, spraying, and data collection, suitable for small to medium-sized farms.

The core platform is based on a rubber track locomotion system, which provides stability and flexibility, especially in rough terrains. The platform also includes a modular mount to support various implements, allowing it to perform multiple farming tasks with ease. The robotic system was built with considerations for scalability, focusing on replication of smaller units rather than developing a single larger machine.

The prototype incorporates a range of sensors, including a LiDAR, cameras, and

Figure 1.3: *Hammerhead rover*

GPS, which allow for precise navigation and data collection. The vehicle's computational architecture includes a low-level system to handle basic operations, using a dedicated industrial PC, and a high-level system that manages autonomous navigation using the Robot Operating System (ROS). Human-machine interaction is also supported via onboard panels and a graphical interface on the remote control, allowing users to control and monitor the platform's status.

This rover deserves this mention because it is the mobile platform that was used to carry the kiwi harvesting robotic manipulator developed in this thesis.

1.4 Overview of this work

This thesis focuses on the development of a robotic kiwi harvesting manipulator, an automated system designed to precisely and effectively perform the harvesting process, thereby contributing to enhanced efficiency and productivity in kiwi farming. The research addresses both the mechanical and algorithmic aspects of this complex problem, proposing a comprehensive solution that combines novel hardware design, advanced control methods, and sophisticated perception capabilities.

To explore the multi-faceted challenges involved in creating an effective robotic harvesting system, this thesis is structured into eight chapters, each dealing with different aspects of the development process. In Chapter 2, the design concept of the manipulator is introduced and analysed, providing insight into the foundational considerations that shaped the overall architecture. This chapter sets the stage for understanding the subsequent detailed work on the various components of the system.

Chapter 3 focuses specifically on the hardware design, with a particular emphasis on the sizing and selection of actuators. Actuator selection plays a critical role in determining the manipulator's performance, as it directly impacts the reach, speed, and delicacy required for effective kiwi harvesting.

Moving forward, Chapter 4 delves into the control systems and algorithms developed to govern the manipulator's movement. This includes motion control system, trajectory planning and integration of the camera feedback for trajectory correction.

Chapter 5 presents a deep dive into the software developed to support the entire system. This includes the architecture that integrates hardware components, facilitates communication between different subsystems, and ensures the smooth operation of the manipulator. Software is the glue that binds the mechanical and perception aspects, and this chapter illustrates how different software modules have been designed and implemented.

The perception capabilities of the manipulator are discussed in Chapter 6. To effectively locate and identify kiwis in a complex orchard environment, a perception pipeline based on cameras and computer vision algorithms has been developed.

The effectiveness of the robotic system is demonstrated in Chapter 7, which presents both real-world experimental results and results obtained through simulations. This chapter offers a comprehensive evaluation of the manipulator's performance, highlighting both its strengths and areas for improvement, based on trials conducted in controlled and practical environments.

Finally, Chapter 8 draws conclusions from the development and testing phases, summarizing key findings, identifying lessons learned, and proposing future directions for enhancing the system. These insights aim to inform both the refinement of this particular harvesting manipulator and the broader field of agricultural robotics.

2

Design concepts

In this chapter, the requirements for the fruit harvesting task are thoroughly examined, leading to the identification of key design objectives. These task requirements, including environmental considerations, fruit characteristics, and operational constraints, serve as the basis for developing effective robotic solutions. The chapter then details the design choices made to address these requirements, covering aspects such as system architecture, mechanical layout and integration with the mobile platform.

2.1 Preliminary tests

Preliminary tests have been conducted to achieve fully autonomous fruit picking and harvesting in kiwifruit orchards. The concept involves navigating each orchard row with an autonomous vehicle, like the one developed by our team and described in this work (Dario Mengoli et al., 2020), equipped with multiple robotic arms to replicate the approach presented in a similar study (Williams et al., 2019). Kiwifruit is particularly well-suited for this type of application since harvesting occurs before the fruit is fully ripened, which makes it compact and durable enough for mechanical handling.

To initiate the investigation, a low-cost RL-DP-5 robotic arm from Igus, featuring five Degrees of Freedom (DoF), was acquired for laboratory testing (Figure 2.1 - left). This setup allowed for the development of initial trajectories for fruit grasping and detachment. A custom-designed grasping hand was created using a commercial resin 3D printer (Figure 2.1 - right). For simplified initial field tests, an off-the-shelf Schunk parallel gripper was installed on the robotic arm to achieve a linear clamping motion. The grasping tool was tested both with and without soft padding to evaluate the fruit's response, potential damage, and slip prevention while optimizing the grasping position.

Initial trials were performed in a controlled laboratory environment, where kiwifruit was suspended to explore various approach and detachment trajectories (Figure 2.2). To accelerate the development process, all trajectories were predetermined and implemented relative to the manually acquired fruit position. The process was configured as follows:

Figure 2.1: *Igus RL-DP-5 robotic arm with the customized gripper*

- The position of the kiwifruit was manually retrieved and saved within the robotic arm's workspace.
- The approach position was set by manually moving downward from the fruit's location.
- The arm's starting position was randomly set near the approach position.
- The robot followed a computed trajectory from the starting position to the approach position.
- Upon reaching the approach position, the arm moved upward to align with the fruit's location.
- The gripper then closed to secure the fruit.
- The detachment maneuver was initiated, involving tilting the end effector to approximately 70 degrees.
- The arm moved downward to complete the detachment of the fruit.
- Finally, the robot transported the fruit to a designated delivery position for appropriate storage.

All trajectories were computed based on the kinematic capabilities of the robotic arm, utilizing both the provided Igus Robot Control software and the ROS package MoveIt motion planning framework (Coleman et al., 2014). This framework offers essential tools such as inverse kinematics and joint position solvers to facilitate precise motion planning.

Figure 2.2: *Laboratory setup for the preliminary kiwi grasping test*

2.2 Mechanical design

2.2.1 Kiwi plant structure

Kiwifruit training systems play a crucial role in optimizing the growth, productivity, and quality of the fruit. Two common training systems used in kiwifruit production are the Pergola (or Tatura trellis) system and the T-bar (or Horizontal Trellis) system. These systems have distinct features that affect vine management, fruit yield, light exposure, and labour requirements. Below, the key differences between the Pergola and T-bar training systems are described:

- T-bar System (Figure 2.3A): the T-bar system involves a central post with cross-arms at the top, which creates a structure shaped like a "T." Wires run horizontally along these arms, allowing the vine to grow laterally in a more linear, controlled manner. This keeps the vine growth at a lower height compared to the Pergola system, allowing for a more open training structure.
- Pergola System (Figure 2.3B): the Pergola system is characterized by a horizontal overhead trellis resembling a large "roof" or "canopy," similar to a traditional pergola. The vines grow up to a supporting wire and then spread out, forming

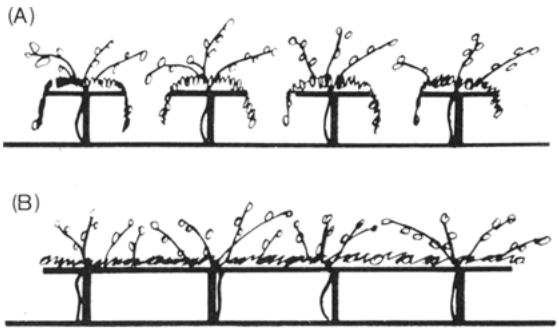


Figure 2.3: (*A*) *T-bar System;* (*B*) *Pergola System* (*Mcaneney et al., 1984*)

a flat, ceiling-like structure over the planting area. This system creates a dense, shaded area under which the fruit grows, allowing a large number of vines to be trained in all directions.

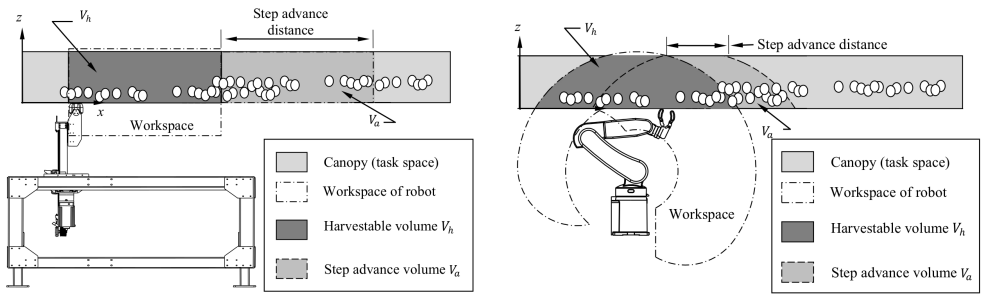
Although the Pergola system is much more friendly for robotic systems due to its overhead canopy and structured fruit hanging, we have developed our harvesting robot with the T-bar system in mind because this is the most commonly used system here in Italy. Additionally, in order to avoid making the project overly complex, we decided that our robot will only harvest kiwifruits from the top part of the plant, as indicated in Figure 2.4.

2.2.2 Workspace analysis

In designing a harvesting robot for kiwifruits, a comparison between Cartesian and articulated robotic configurations reveals distinct performance advantages and limitations. According to Au et al., 2020, the Cartesian robot system was found to be more efficient than an articulated design for harvesting kiwifruit trained on Pergola systems, largely due to its regular workspace geometry that closely matches the canopy's structure. In particular, it has been considered a robot mounted on a mobile platform that makes a step forward after the previous area has been completed, as shown in Figure 2.5. In this study, two real robots were compared and the results are summed up in Table 2.1: even if the workspace volume of the articulated robot is much larger than the Cartesian one, it requires more steps and has an overall greater harvest cycle time, due to the poor overlapping with the canopy (task space).

However, in our context of a T-bar training system, where the structure of the canopy differs significantly from the pergola, we decided to opt for an anthropomorphic (articulated) robot. The articulated design offers better adaptability for the irregular and more spatially complex layout of T-bar systems, where its ability to move with rotational flexibility allows for better coverage and accessibility of the fruits, despite the increased operational complexity compared to the Cartesian alternative. By

Figure 2.4: *Selected area for the harvesting activity*



(a) *Cartesian robot* **(b)** *Articulated robot*

Figure 2.5: *Workspace geometry of robotic arms* (*Au et al., 2020*)

	Cartesian	Articulated
No of steps	5	9
Workspace volume (m^3)	0.439	2.15
Harvestable volume (m^3)	0.293	0.348
Step advance volume (m^3)	0.293	0.148
Workspace efficiency	100%	42.529%
Estimated harvest cycle time (s)	4.48	8.88

Table 2.1: *Performance comparison between the Cartesian and an articulated design* (*Au et al., 2020*)

prioritizing flexibility and effective manoeuvrability over workspace regularity, the articulated robot becomes a more practical solution for this specific orchard layout. Moreover, it allows the possibility to work on both sides of a row (Figure 2.6) and potentially can become more compact when the robot is not working. Additionally, the anthropomorphic design is more convenient to parallelize, as the cycle time can be reduced depending on the number of manipulators installed on the machine, allowing for scalable efficiency.

2.3 Kinematic parameters

As mentioned before, the Igus RL-DP-5 robot was used for preliminary tests. However, due to several limitations regarding joint speed and mobility, it was decided to build

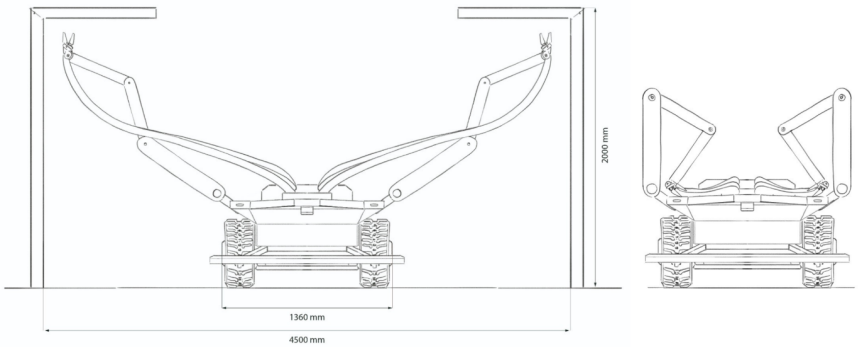


Figure 2.6: *The anthropomorphic design potentially allows to work on both rows and is more compact in idle state*

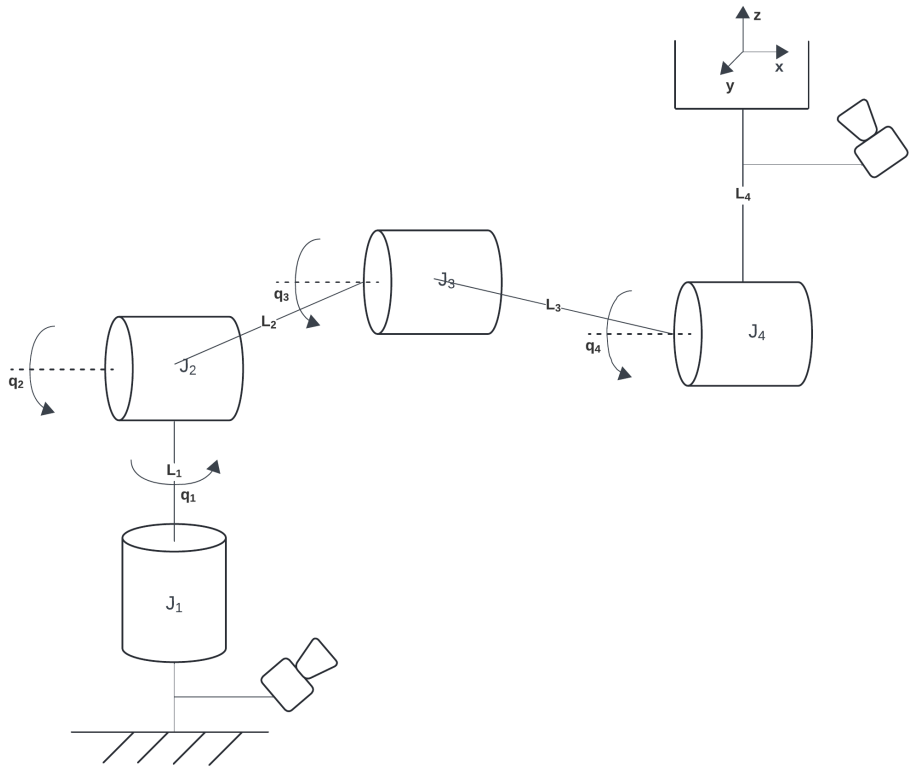


Figure 2.7: *Robot layout*

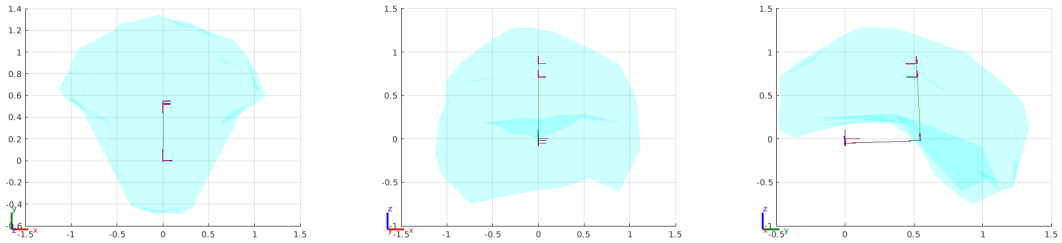
a custom robot from scratch. Another significant factor in this decision was the high cost of commercial robotic manipulators, which conflicted with the project's goal of maintaining low costs. The kinematic model of the newly designed robot is shown in Figure 2.7 and consists of four revolute joints arranged in an anthropomorphic arm configuration.

As seen in the figure, the robot is equipped with two cameras. One camera is attached to the base link, and the other is mounted on the robot's end-effector. The camera on the base link is a stereo camera, capable of providing depth measurements, while the one on the end-effector is a standard RGB camera. More details about the cameras will be presented in the next chapters.

The robot's kinematics are described using Denavit-Hartenberg (DH) parameters, which are summarized in Table 2.2. Additionally, the physical joint limitations are provided in Table 2.3, showing the constraints on each joint's movement.

Table 2.2: *DH parameters of the robot*

Link	a_i	α_i	d_i	ϑ_i
L_1	0.000	$\pi/2$	0	q_1
L_2	0.547	0	0	q_2
L_3	0.733	0	0	q_3
L_4	0.155	0	0	q_4



(a) *xy plane* (b) *xz plane* (c) *yz plane*

Figure 2.8: *Robot workspace considering the joints limits*

In light of the joint limitations presented above, a simulation was conducted to visualize the robot's effective workspace, which is shown in Figure 2.8. This simulation helps to illustrate the reachability and mobility of the robot, taking into account the constraints imposed by the joint limits.

It is important to note that the restrictions on Joint 1 are not due to mechanical constraints but are instead a design choice aimed at ensuring safety during operation. Limiting the range of Joint 1 helps to prevent excessive movements that could pose risks during manipulation tasks, particularly in environments where the robot operates near humans or other sensitive equipment. This design consideration, coupled with the remaining joint constraints, ensures that the robot can perform its tasks efficiently while maintaining a safe operational boundary.

Joint	Min(rad)	Max(rad)
J_1	-1.1	1.1
J_2	0.6981	2.5540
J_3	0.5436	2.5435
J_4	-1.9199	2.3399

Table 2.3: *Joint limits*

2.4 Gripper

2.4.1 Initial design

The design of a gripper for robotic kiwifruit harvesting poses a unique challenge: the fruit must be handled gently to avoid any damage both during detachment and release. Unlike many industrial applications where precision and strength are paramount, fruit harvesting requires a careful balance between delicacy and effective control to ensure the fruit reaches marketable quality. The kiwifruit's delicate skin, susceptibility to bruising, and the complexity of its surrounding environment with branches and leaves make this design task particularly demanding.

Initially, the proposed gripper design, as depicted in Figure 2.9, involved the use of flexible fingers mounted on a small conveyor belt. This concept was inspired by the

desire to achieve both fruit handling and release in a continuous, controlled motion, thereby minimizing the risk of abrupt shocks that could lead to damage. The conveyor belt featured soft, flexible elements that could adjust to the contour of the fruit, providing a gentle yet firm grip.

The envisioned mechanism included two flexible paddles that would open to gently encompass the kiwifruit, and then move it onto the conveyor belt where it would be directed towards a collection bucket. This design aimed to minimize damage during both detachment from the vine and the final release into the container. However, despite these promising features, the design had several inherent limitations that made it unsuitable for real-world harvesting.

The main issue with the conveyor belt-based gripper was its cumbersome nature. In an orchard setting, where branches and leaves can be obstacles, the relatively large footprint of this system posed a significant risk of collision, potentially damaging both the plant and the equipment itself. Its bulkiness also restricted the gripper's ability to move in tight spaces, a key requirement when navigating through dense foliage. As a result, while the flexible fingers were effective in maintaining a gentle grip, the complexity of the conveyor mechanism and the potential for collisions necessitated a reconsideration of the design.

2.4.2 Final design

Based on these observations, we decided to pivot towards a more compact and agile design. Instead of developing a novel gripper entirely from scratch, we sought inspiration from existing technologies and previous research on kiwifruit harvesting. This led us to evaluate and eventually adapt the work presented in Scarfe, 2012, in which a highly practical and efficient kiwifruit harvesting gripper was developed.

The original design by Scarfe utilized a soft, pneumatically actuated gripper that successfully demonstrated a balance between robustness and delicacy. The gripper employed soft, adaptive surfaces that conformed to the shape of the fruit, minimizing pressure points and thereby reducing the risk of bruising. The actuated fingers provided sufficient force to detach the fruit without causing damage, addressing one of

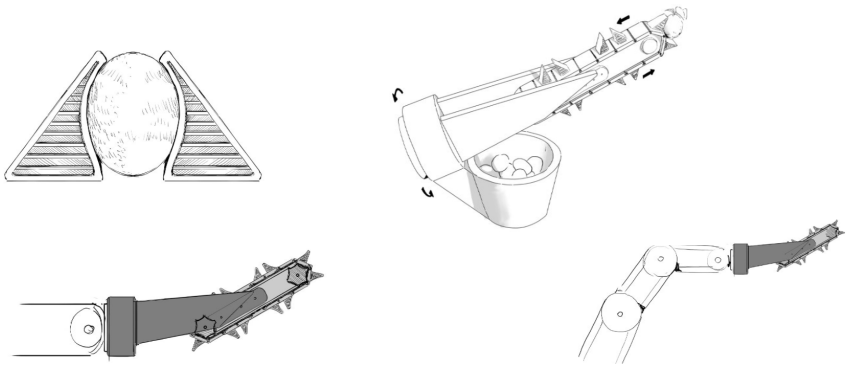
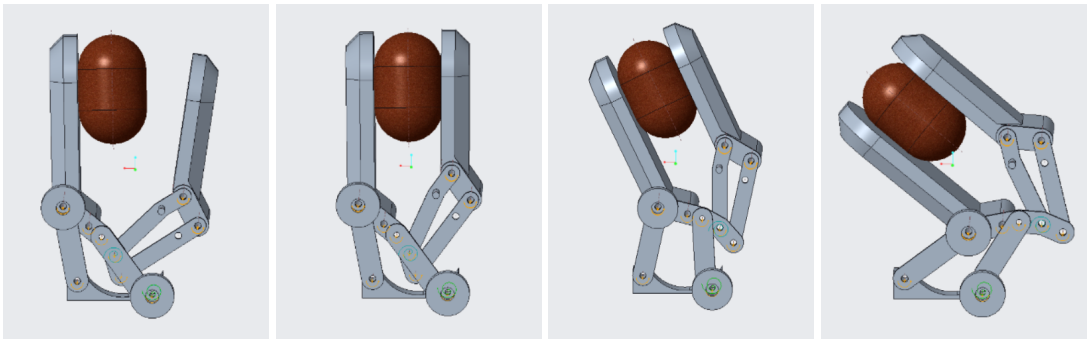


Figure 2.9: *Possible gripper design*



(**a**) *Open* (**b**) *Closed* (**c**) *Half tilted* (**d**) *Full tilted*

Figure 2.10: *Grasp and detach sequence*

the key design challenges effectively.

Our adaptation of this design focused on reducing weight and making the gripper more compact, allowing it to be more agile when interacting with different canopy structures. Unlike Scarfe's gripper, ours is actuated by only one electric motor with a limit switch, significantly simplifying the mechanism while reducing weight. The gripper also features a housing for an RGB camera that we introduced for precision picking, enhancing its ability to target individual fruits with higher accuracy. The compact design improves the ability to navigate around branches and selectively pick individual fruits without unnecessary collisions.

The Figure 2.10 shows our gripper: the original structure of the four-bar linkage with a fixed finger plus a moving thumb is preserved. The sequence depicted in the figure includes four stages of the grasping process:

1. (a) Open: The gripper starts in an open position, with both the fixed finger and the thumb apart, ready to approach the kiwifruit.
2. (b) Closed: The moving thumb closes towards the fixed finger, securely holding the kiwifruit without applying excessive pressure, thanks to the limit switch that stops the motor.
3. (c) Half Tilted: The gripper begins tilting to apply the correct shear force to the stalk, ensuring that the fruit detaches from the branch while maintaining a firm grip.
4. (d) Full Tilted: The gripper reaches a fully tilted position, completing the detachment process and preparing the fruit for subsequent release into the collection container.

Then, when the gripper comes back in the open position, the fruit is released into the expelling tube.

2.5 Mobile platform integration

The integration of a kiwi-harvesting robotic arm with the autonomous rover, described in Section 1.3, presents a set of unique design requirements. The objective is to seam-

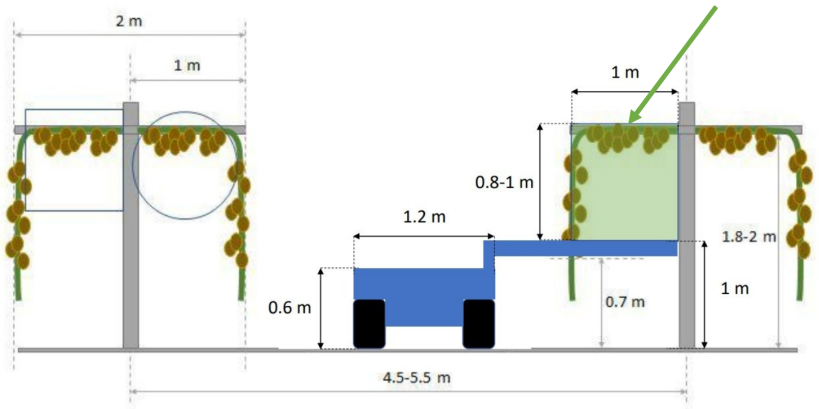


Figure 2.11: *Design concept of the rover-manipulator system*

lessly adapt the mechanical harvesting system onto the rover while respecting the geometric and operational constraints inherent in the orchard environment, as highlighted in Figure 2.11. This section explores how the integration can be carried out efficiently to meet these requirements.

The orchard architecture shown in Figure 2.11 delineates critical spatial constraints. The kiwi canopies are supported at a height ranging between 1.5 and 1.8 meters, with a 1-meter-wide overhanging structure. Given these dimensions, the robotic arm must be capable of reaching up to at least 1.8 meters with sufficient horizontal flexibility to access fruit located in the green square. The autonomous rover serves as the mobility base, providing the necessary support and power supply for the robotic arm, and must therefore be positioned and controlled to ensure complete coverage of each canopy section while manoeuvring around the orchard.

To accomplish effective integration, several engineering considerations were made:

- Height and Reach Optimization: The robotic arm is designed to extend vertically up to 1.8-2 meters, ensuring that it can adequately access all kiwis hanging within the target canopy height. The optimal reach of the arm accounts for both the vertical clearance and the 1-meter horizontal extension required to harvest kiwis located away from the main support poles.
- Rover Mobility and Stability: The rover, as depicted in the second image, utilizes tracked propulsion for enhanced manoeuvrability in the orchard environment, which often presents uneven terrain. The integration must ensure that the robotic arm is stable during harvesting operations. The arm is mounted on a stabilizing platform that sits above the rover's primary frame at a height of approximately 0.6 meters, as indicated in the orchard layout. This elevation ensures that the arm has the necessary leverage while maintaining the rover's balance. The compact design of the rover, featuring a width of 1.2 meters, allows it to navigate between rows, which typically span 4.5 to 5.5 meters apart, without risking damage to nearby plants.
- Fruit Evacuation System: To facilitate efficient collection of harvested kiwifruit,

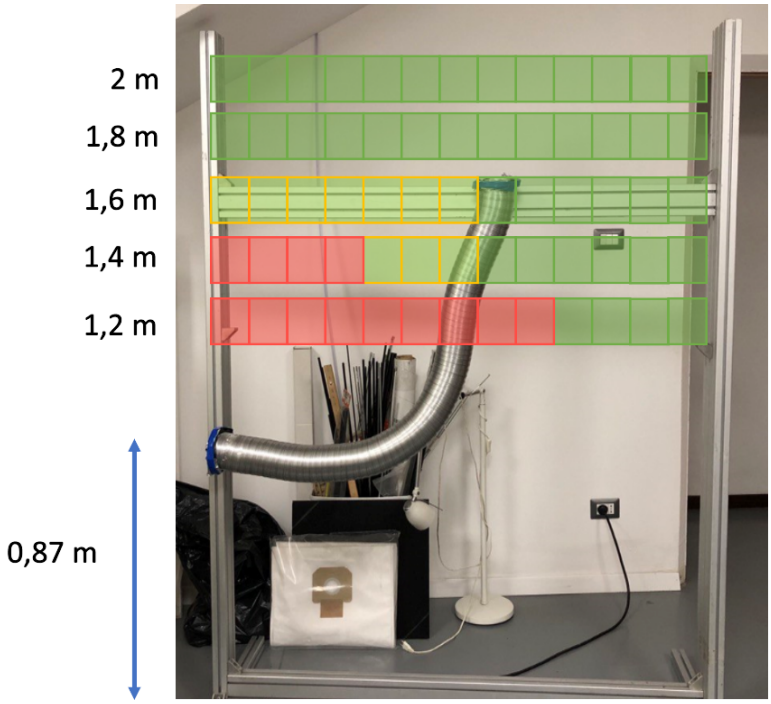


Figure 2.12: *Tube discharge experiment*

a bin is mounted on the rover to collect all the picked fruits. The harvested kiwis are transferred from the gripper to the bin via a discharging tube. This tube ensures a smooth conveyance of fruit from the robotic arm to the storage bin, minimizing the risk of damage during transfer. Experimental tests, as depicted in Figure 2.12, were performed to validate this evacuation system. The tests focused on ensuring the harvested fruits could travel through the flexible tube without bruising, given its positioning at a height of 0.87 meters from the ground and the required inclination to guide the fruit to the bin. The tube has been reinforced with an elastic band to prevent it from being loose. Then, several falling tests have been performed and, as it is shown in the figure, the green squared positions had a successful discharging, in the red ones the kiwi remained in the tube, in the yellow ones it was uncertain.

The final built prototype is shown in Figure 7.3.

3

Hardware

This chapter deals with the hardware components of the prototype robotic arm, with a particular focus on actuator sizing. The selection and sizing of actuators are critical to ensure that the robotic arm can perform the required tasks efficiently. The chapter discusses the different hardware components, including motors, sensors, and structural elements, and explains how they were chosen and integrated to meet the performance requirements established in the previous chapter.

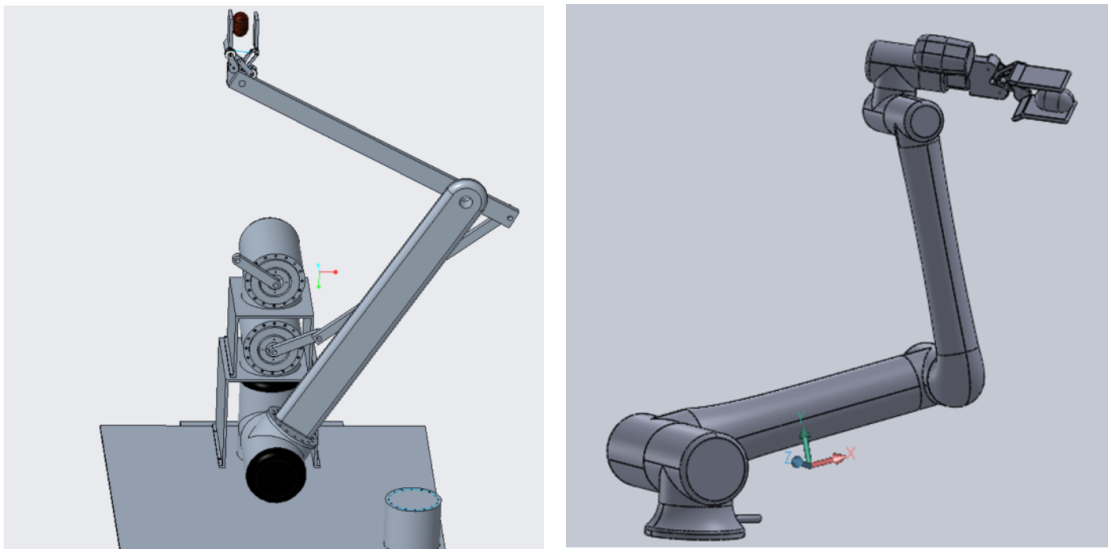
3.1 Actuators selection

3.1.1 Layout and manufacturer selection

The two solutions presented in Figure 3.1 highlight different design approaches for actuator placement in robotic arms, each with distinct advantages and disadvantages. In solution A, all actuators are attached to the base link of the robot. This design significantly reduces the overall mass of the moving parts, making the structure lighter and thus improving its dynamic efficiency. However, this advantage comes with trade-off in mechanical complexity. The use of mechanical transmissions to connect the base-mounted actuators to the joints introduces additional components that are prone to failure, increase the risk of collision, and make the entire system more cumbersome when operating in constrained environments, such as around trees.

Conversely, the solution B mounts each actuator directly to its corresponding joint, which offers a simpler mechanical setup. This arrangement reduces the need for complex transmissions, minimizing mechanical wear and potential collision risks, resulting in a cleaner and more straightforward design. However, dynamically, this design is less efficient, as the mass of the actuators contributes to the load carried by the arm, increasing inertia and affecting overall motion control.

The final solution that we adopted is a compromise between these two solutions: it has the motors of the first three joints on the base and the last motor mounted directly on the wrist. In this way, the robot is not too 'heavy' and the risk of collision between mechanical parts and the plants is minimized, because there is no mechanical trans-



(**a**) *Solution A* (**b**) *Solution B*

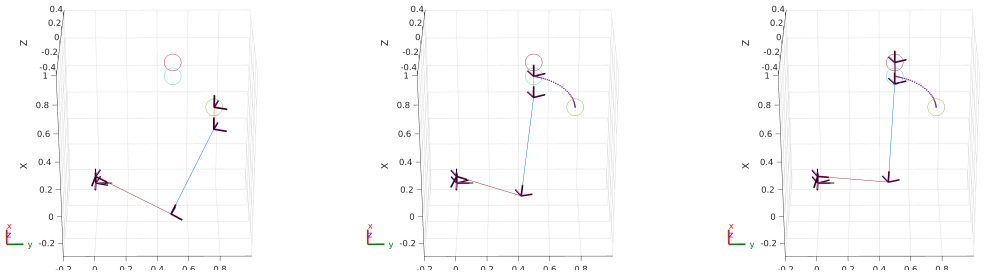
Figure 3.1: *Proposed layouts*

mission close to the tip of the robot, that is the part that works close to the kiwi plant.

When selecting a manufacturer for the robot actuators, several key characteristics were prioritized to match the needs of the design. The actuators had to operate at low voltage (48V) for safety, the motor electronics needed to be integrated internally to minimize space requirements, and they also needed to be affordable. Based on these criteria, two manufacturers were evaluated. The first option was Sumitomo, specifically their Tuaka series. These actuators offer significant advantages, including three different sizes, high mechanical performance, and integrated safety functions, which make them appealing for demanding robotic applications. However, they also come with notable drawbacks. The Tuaka actuators are relatively expensive, and their size and weight are not ideal for the intended robot design. Additionally, as will be further detailed in subsequent sections, even the smallest of the three available sizes are somewhat larger than needed given the required torque, which could negatively impact the overall efficiency and compactness of the system.

The second manufacturer considered was Cubemars, particularly their AK series. The Cubemars actuators have several advantages, including a compact and lightweight design, a wide range of available sizes, and affordability. They can also be easily interfaced over CAN bus, which adds flexibility to the control system. However, there are some drawbacks. The documentation provided by Cubemars is quite poor, which could complicate integration and troubleshooting. Additionally, the mechanical performance of the AK series is inferior compared to the Sumitomo Tuaka series, and their customer support has been noted to be less responsive, which could lead to delays in resolving technical issues.

These two actuator lines will be evaluated in the next paragraphs, determining the right size for each joint based on a typical task trajectory.



(**a**) *Start position* (**b**) *Approach position* (**c**) *Goal position*

Figure 3.2: *Reference task*

3.1.2 Task definition

The task trajectory for the robot is designed with a target duration of 6 seconds, split across distinct phases that support efficient harvesting of kiwifruit. Initially, the robot performs a fast transfer from the start position (Figure 3.2a) to an 'approach' position (3.2b) beneath the kiwifruit, using a minimum jerk trajectory to ensure smooth and precise movement. This phase lasts 2.5 seconds and covers a distance of 0.5612 meters, which is considered a challenging scenario for motor performance evaluation. Next, a 2.5-second quintic polynomial slow approach phase brings the end-effector into close proximity for picking, emphasizing accuracy and minimizing sudden force impacts on the fruit (Figure 3.2c). Finally, the robot remains stationary for 1.0 second, allowing for the gripper to complete the fruit-picking process. This sequence forms the reference trajectory (Figure 3.3) for repeated harvesting tasks and serves as a basis for motor sizing and performance assessment.

3.1.3 Motor sizing

In order to compute the required torque to execute the reference trajectory, the dynamic parameters of the links, joints, and motors are reported in Table 3.1. It can be noted that the moment of inertia of Link 1 has been neglected, while the joints and the motors have been considered as a point mass. Using these parameters, the torque required to execute the reference trajectory can be computed, and the resulting torque profile is shown in Figure 3.4.

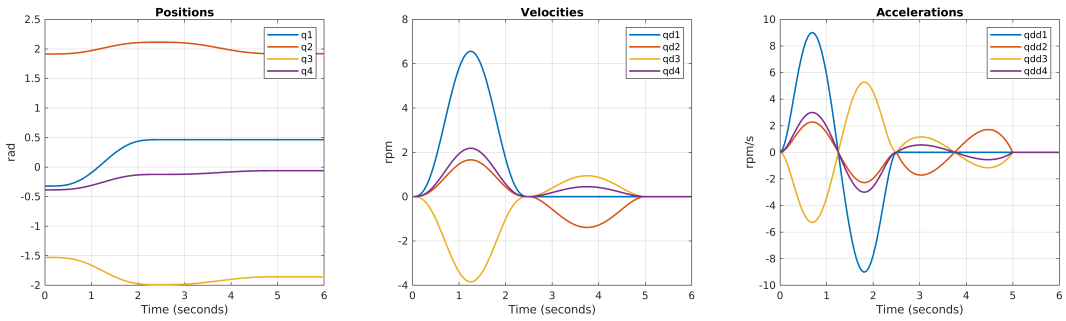


Figure 3.3: *Reference trajectory*

Element	Mass (Kg)	M. of inertia (Kgm^2)
L_1	0.100	$[0.0, 0.0, 0.0, 0.0, 0.0, 0.0]$
L_2	0.190	$[1.56, 0.01, 1.57, 0.0, 0.0, 0.0] \cdot 10^{-2}$
L_3	0.210	$[1.14, 0.01, 1.14, 0.0, 0.0, 0.0] \cdot 10^{-2}$
L_4	0.060	$[1.35, 0.22, 1.36, 0.0, 0.0, 0.0] \cdot 10^{-4}$
Joints	0.500	-
Gripper	2.000	-
CB_AK80-64	0.850	-
CB_AK70-10	0.521	-
CB_AK60-6	0.315	-
TUAKA107-80	3.500	-
TUAKA203-50	1.815	-

Table 3.1: *Dynamic parameters. The inertia tensor is expressed as* $[I_{xx}, I_{yy}, I_{zz}, I_{yz}, I_{xz}, I_{xy}]$

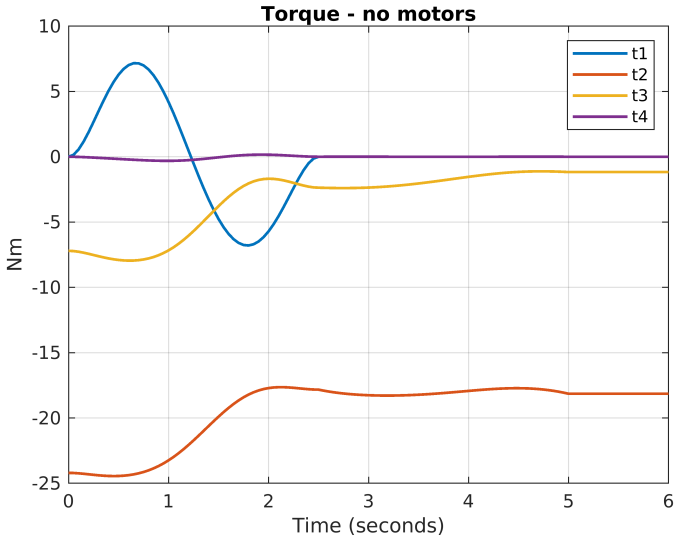


Figure 3.4: *Torque required to execute the reference trajectory, without the motors mass*

Considering the two actuator families (Sumitomo and Cubemars), I added the contribution to the masses and inertias of the actuators and iteratively simulated the execution of the reference trajectory, to measure the required torque. Therefore, I selected a compatible motor sizes for both families in terms of nominal torque, peak torque and nominal speed. For the Sumitomo family:

- Tuaka 107-80 on joint 2: $T_n = 63Nm$, $T_p = 167Nm$, $V_p = 35rpm$.
- Tuaka 203-50 on joint 1, 3 and 4: $T_n = 21Nm$, $T_p = 44Nm$, $V_p = 123rpm$.

For the Cubemars family:

- AK80-64 on joint 2: $T_n = 48Nm$, $T_p = 120Nm$, $V_p = 48rpm$.
- AK70-10 on joint 1 and 3: $T_n = 8.3Nm$, $T_p = 24.8Nm$, $V_p = 310rpm$.
- AK60-6 on joint 4: $T_n = 3Nm$, $T_p = 9Nm$, $V_p = 233rpm$.

Considering these configurations, the torque required to execute the reference trajectory is summed up in Table 3.2.

Motors	T_{max} **(Nm)**	T_{rms} **(Nm)**
No motors	$\begin{bmatrix}7.1614 & 24.4498 & 7.9522 & 0.3121\end{bmatrix}$	$\begin{bmatrix}3.0149 & 19.5570 & 3.8797 & 0.1117\end{bmatrix}$
Sumitomo	$\begin{bmatrix}12.3409 & 48.5423 & 13.5144 & 0.3121\end{bmatrix}$	$\begin{bmatrix}5.1894 & 40.0287 & 6.5828 & 0.1117\end{bmatrix}$
Cubemars	$\begin{bmatrix}11.2121 & 29.5605 & 9.1372 & 0.3121\end{bmatrix}$	$\begin{bmatrix}4.7554 & 23.9843 & 4.3776 & 0.1117\end{bmatrix}$

Table 3.2: *Torque required to execute the reference trajectory with the selected motors*

For the reasons expressed before and in view of this computation, the configuration with the Cubemars has been chosen for the construction of the final prototype.

3.2 Electrical architecture

This electrical schematic in Figure 3.5 illustrates the layout and power connections for a system controlled by the main PC and the Arduino Portenta H7. Below is a detailed description of how each component is interconnected and functions in the overall scheme:

1. Power Supply:
   - The primary power source is a 48V rover battery.
   - This 48V power is converted into 24V and 12V using DC-DC converters. These lower voltages are likely used to supply various components that require lower voltage levels for operation.
   - An Emergency Stop button is connected to the 48V power supply, which cuts off power in case of an emergency. This switch is directly placed in the power path, ensuring a complete shutdown of power to all components if pressed.
2. Motors and CAN Bus Network:

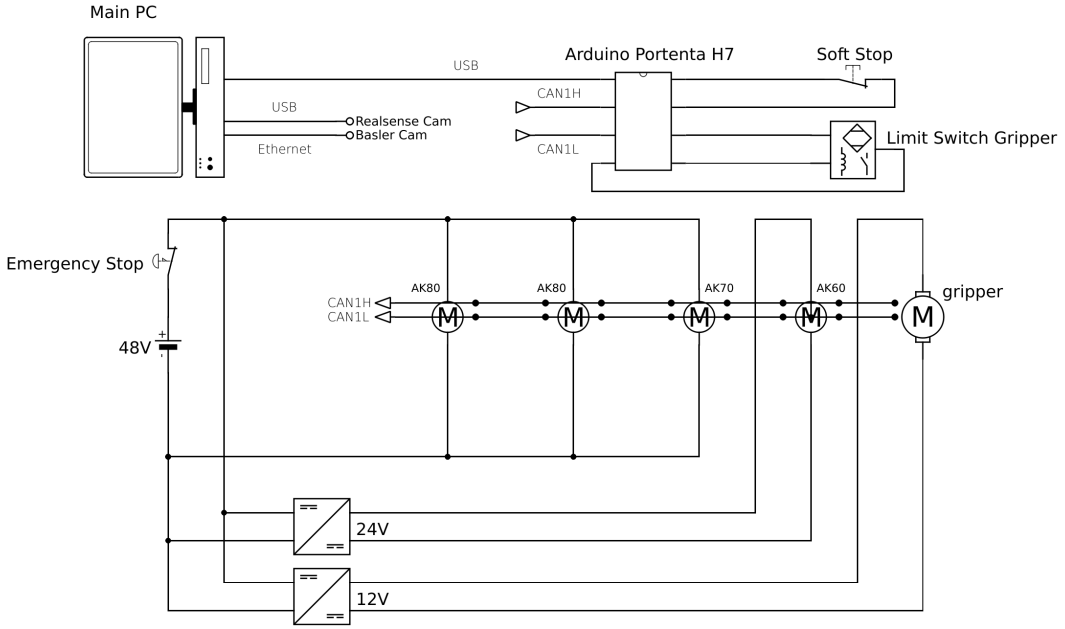


Figure 3.5: *Electrical scheme*

- There are four motors labeled AK80, AK80, AK70, and AK60 as well as a gripper motor.
- All the motors are connected to a CAN Bus, which includes CAN1H (high) and CAN1L (low) lines. This bus allows communication between the motors and a central controller.
- The motors receive control commands via the CAN Bus, and all are linked in a shared bus architecture, enabling coordinated control.

3. Arduino Portenta H7:

- The Arduino Portenta H7 serves as the master controller for the CAN Bus. It communicates with the motors via the CAN Bus (CAN1H and CAN1L).
- It also manages the Soft Stop button and a Limit Switch that is connected to the gripper mechanism. The Soft Stop button is a safety feature to stop the motors in a controlled manner. The Limit Switch of the gripper helps determine the end positions of the gripper for safety and control purposes.
- The Arduino is connected to the main PC via USB, which provides data communication and power supply.

4. Main PC:

- The main PC is responsible for managing high-level trajectory planning and interfacing with the Arduino.
- The PC also manages external cameras for the system: the Realsense camera and the Basler Camera are connected via USB and Ethernet respectively.

3.3 Sensors

For our kiwi harvesting robot, visual perception is a critical component that ensures precise detection, localization, and safe picking of the fruit. After careful evaluation, we selected two cameras: the Intel RealSense D435i (Figure 3.6a) and the Basler Ace (Figure 3.6b).

The Intel RealSense D435i is a depth camera that utilizes stereo vision technology to generate real-time 3D data, which is essential for the precise spatial positioning of kiwis in the environment. The D435i provides depth resolution of up to 1280x720 pixels and captures up to 90 frames per second, ensuring accurate depth perception at high speeds. This capability is crucial for our robot, which needs to make rapid decisions in a dynamic orchard setting.

Additionally, the RealSense D435i is highly portable and features a compact form factor, making it easy to integrate into the physical design of our robot. Its affordable cost also makes it an economical solution for scalability if multiple harvesting units are deployed.

This camera is mounted on the base link of the robot and it is used to retrieve a first guess of the kiwifruits positions in the workspace.

Complementing the depth perception capabilities of the Intel RealSense D435i is the Basler Ace camera. The Basler Ace is a high-resolution RGB camera known for its exceptional image quality and robust design. The Basler Ace provides crisp 2D imagery with resolutions up to 5 megapixels, allowing our system to distinguish between ripe and unripe kiwis based on their colour and texture.

The Basler Ace also comes with features that support low-light performance, which is essential for early morning or late evening harvesting operations. With its fast frame rates, the camera can capture quick movements of the robotic arm without motion blur, ensuring the precision required in the picking process. Moreover, the Basler Ace is compatible with a variety of lenses, which enables us to adjust the field of view depending on the specific configuration of the orchard.

(**a**) *Realsense D435i* (**b**) *Basler Ace*

Figure 3.6: *Cameras on the robot*

The choice of using both the Intel RealSense D435i and Basler Ace cameras comes from the fact that the RealSense alone is not able to achieve the required precision of kiwi position estimation, especially in an outdoor scenario with the sunlight disturbance. We chose to add a compact RGB camera like the Basler Ace rather than another stereo camera, as it needed to be mounted on the robot's end effector, requiring a lightweight and compact solution.

4

Control architecture

This chapter deals with the control architecture of the robot. The control system is essentially split between two main components: a micro-controller and a main PC. The micro-controller is responsible for implementing the motion control scheme, ensuring precise actuator movements to achieve desired positions and velocities. Meanwhile, the main PC runs ROS2 (Robot Operating System) along with MoveIt2 for trajectory planning and camera management. This chapter explains how these components interact to enable coordinated control of the robotic arm, from low-level actuator commands to high-level path planning and vision-based decision-making. The integration of ROS2 and MoveIt2 provides a flexible and powerful framework for executing complex motions, making it possible to navigate dynamic agricultural environments effectively.

4.1 General description

The control architecture of the robot is centered around the Arduino Portenta H7 board, which serves as the primary controller for low-level motion control. This board manages multiple critical subsystems, ensuring smooth coordination of all key robot functions. Specifically, the Arduino Portenta H7 handles CAN communication with the robot's motors (Cubemars AK series), enabling precise motor commands and feedback. Additionally, it oversees the gripper mechanism, including the gripper motor and limit switch sensor (Figure 3.5). The board is also responsible for implementing the soft stop function—a soft version of an emergency stop that, instead cutting off the power supply and letting the robot fall, it stops the robot with a gravity compensation that keeps it still.

The Arduino Portenta H7 board is connected to the main PC running ROS2 through a USB connection, allowing communication to be managed on the serial bus by a dedicated ROS node (more details in Chapter 5). On the PC, the high-level trajectory planning software is implemented. This software processes signals from the cameras and generates the trajectories required to move the robot in the analyzed environment.

We decided to split the computation between the Arduino and the main PC in

order to achieve better performances. The Arduino firmware runs at 1.25 kHz, ensuring minimal delay in the control loop for motion control, while the ROS software on the main PC publishes trajectory updates at a 50 Hz rate, managing updates from the cameras and enabling fast replanning. This division of tasks allows for optimal performance, balancing real-time control with high-level planning.

4.2 Low-level: motion control

The motion control scheme implemented on the Arduino board is presented in Figure 4.1, which has been specifically chosen over the traditional inverse dynamics approach (Figure 4.2, Siciliano et al., 2009). The main reason for this choice lies in the capabilities of the Cubemars motors used in the project, which operate in 'MIT mode'. These motors already internally close the linear feedback loop and also accept an additional torque command that combines different components, as in the structure shown in Figure 4.1:

$$u = K_P\tilde{q} + K_D\tilde{\dot{q}} + T_{ref}$$

By leveraging the internal linear control of the motors, we can close the linear loop with minimal delay, thus avoiding the communication lag introduced by the CAN bus and the processing time of the Arduino controller. This results in a more efficient and responsive control implementation.

It should be noted, however, that the feedback torque (T_{fb}) and the feedforward torque (T_{ff}) are updated at different frequencies, due to the separation between the inner and outer control loops. This design choice allows us to maximize the responsiveness of the linear control while maintaining the advantages of nonlinear compensation.

Furthermore, I decided to neglect the Coriolis term ($C(q, \dot{q})$) in the feedforward term. Given the relatively low velocities of our trajectories, the Coriolis effect remains limited, and its inclusion would add significant complexity to the control without considerable performance gain. Moreover, due to parametric model inaccuracies, the potential benefit of including the Coriolis term would likely be overshadowed by modeling errors. The modeling of the friction term (D) is discussed in detail in Chapter 7, as it plays a critical role in the overall performance of the control system. For completeness, the detailed calculations for the terms $B(q)$ and $g(q)$ are provided in the Appendix A.

To tune the control system, simulations were carried out using Simulink with the Robotics Toolbox (Corke, 2017), directly importing the URDF file (Unified Robot Description Format, provided in Appendix B) of the robot (Figure 4.3). In these simulations, the friction term (D) was not included, as it will be estimated and handled directly on the physical robot, as explained later in the text. Trapezoidal velocity profile trajectories were used as reference, as they provide greater excitation to the system compared to fifth-order polynomial trajectories, which will be used in the final im-

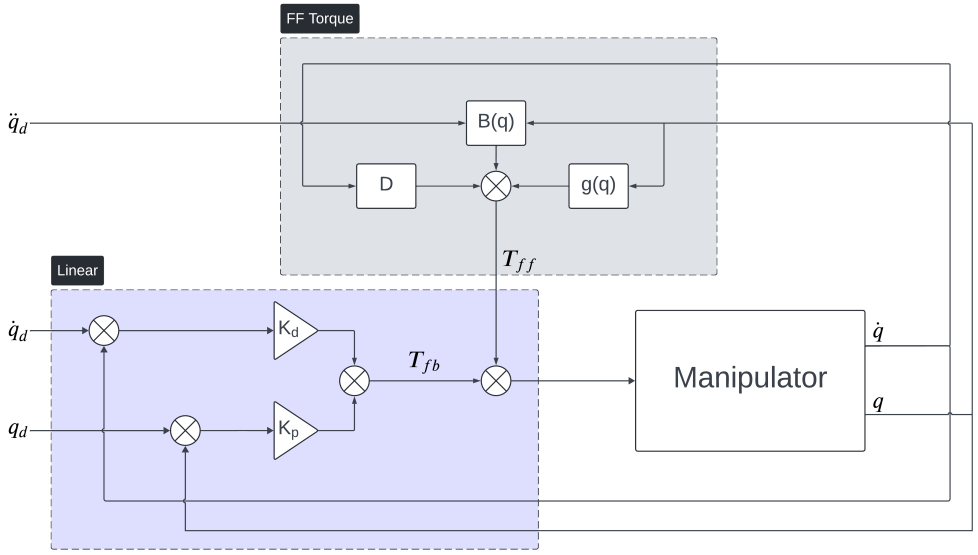


Figure 4.1: *Motion control scheme: the T_{ff} term is computed on the Arduino board, the T_{fb} stabilisation term is computed directly on the motor drives.*

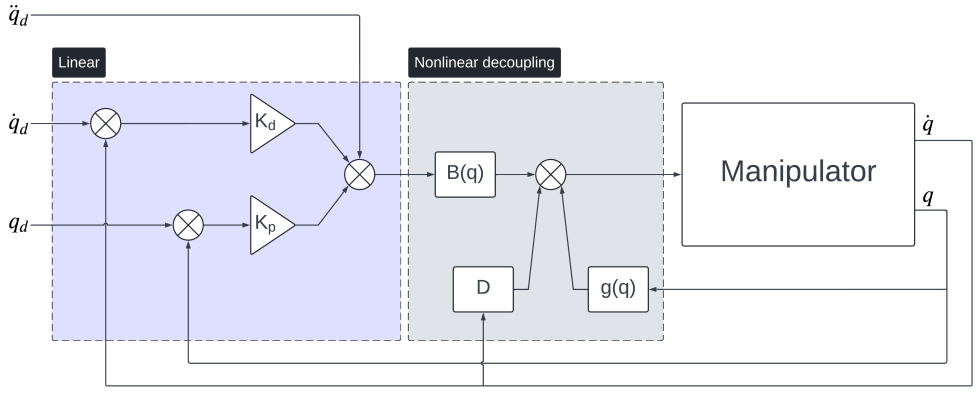


Figure 4.2: *Classical Inverse Dynamics scheme without the Coriolis term.*

plementation. Through this process, empirical tuning led to the selection of suitable values for control gains:

$$K_p = \begin{bmatrix} 15 & 0 & 0 & 0 \\ 0 & 15 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$

$$K_d = \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 15 & 0 \\ 0 & 0 & 0 & 5 \end{bmatrix}$$

These control gain values resulted in the trajectory tracking performance illustrated in Figure 4.4.

4.3 High-level: trajectory planning

Robotics today is evolving towards increasingly sophisticated solutions that demand flexibility, adaptability, and reliability. ROS2 (Macenski et al., 2022), has become a prominent middleware framework in the development of such advanced robotic systems, offering enhanced communication and control capabilities for complex robotic applications. Within ROS2, MoveIt2 (Coleman et al., 2014) stands out as a powerful motion planning framework, providing a comprehensive suite of tools to tackle motion planning, manipulation, and control tasks.

Particularly notable is the MoveIt2 Hybrid Planning pipeline, a recent advancement designed to seamlessly combine global and local motion planning techniques. The Hybrid Planning architecture is composed of several integral components that work in tandem to achieve adaptive and responsive motion planning: the Global Planner, the Local Planner, and the Hybrid Planning Manager. Together, these compo-

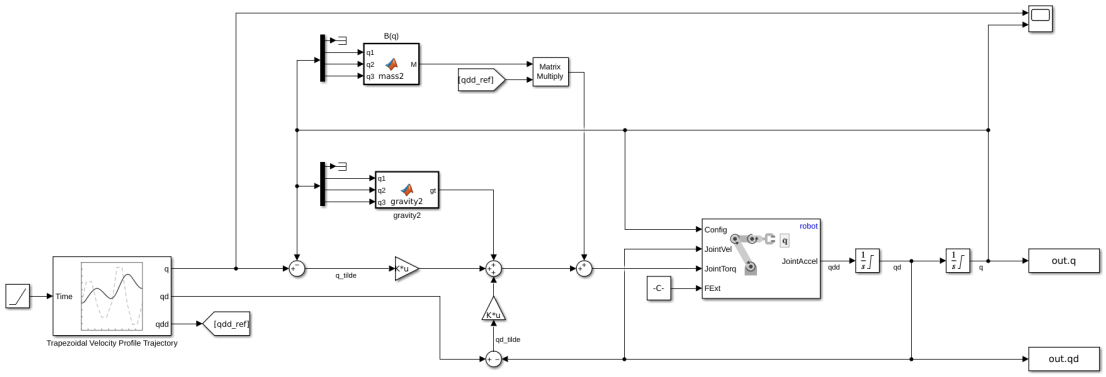


Figure 4.3: *Simulink scheme for control tuning*

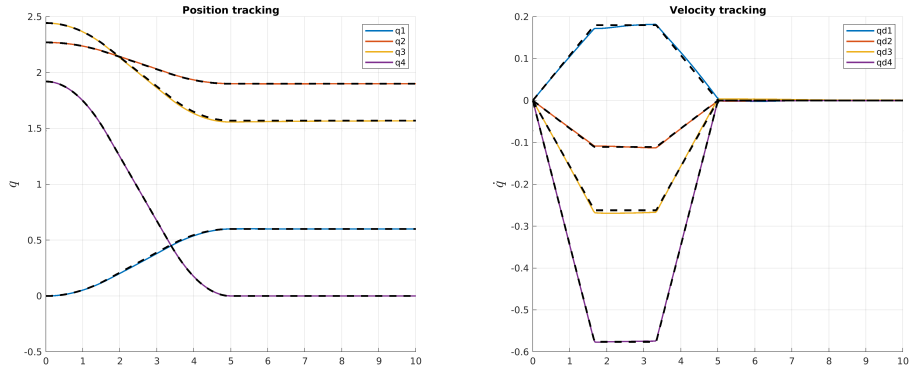


Figure 4.4: *Trapezoidal trajectory tracking*

nents enable a robust and flexible planning system that can handle dynamic and unpredictable environments.

The Global Planner is responsible for generating a high-level plan that guides the robot towards its goal. It operates on a more abstract level, taking into account the overall environment and creating a trajectory that is globally optimal. This planner ensures that the robot's movements are feasible on a larger scale, providing a complete path from the start to the goal while considering obstacles and environmental constraints. The Global Planner's output serves as the foundational guide for subsequent local adjustments.

In contrast, the Local Planner works on a finer scale, focusing on the real-time adaptation of the global trajectory. It allows the robot to react to changes and unforeseen obstacles that may arise during execution. By performing rapid, local adjustments, the Local Planner ensures that the robot can safely navigate around unexpected objects or handle minor deviations from the initial plan without compromising the overall goal. This responsiveness is crucial for dynamic environments, where static planning alone would be insufficient.

The Hybrid Planning Manager orchestrates the interaction between the Global and Local Planners. It acts as a central coordinator, ensuring that the plans generated by both the Global and Local Planners are consistent and complementary. The Planning Manager continuously monitors the execution progress and facilitates communication between the two planning modules. When discrepancies arise, the Hybrid Planning Manager decides whether to update the global plan or adjust the local plan to keep the robot on track. This coordination results in a harmonious balance between long-term strategic planning and short-term reactivity, providing a level of adaptability that is essential for real-world robotic applications.

Importantly, MoveIt2 treats the Global Planner, Local Planner, and Hybrid Planning Manager as plugins, allowing users to write and customize these components based on their specific needs. This modularity provides significant flexibility in designing planning strategies tailored to particular applications. As described before, I leveraged this feature to create custom implementations for the Global Planner, Lo-

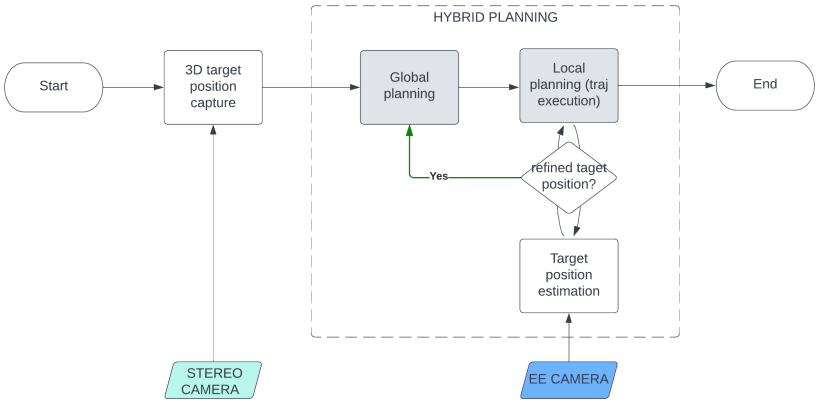


Figure 4.5: *Hybrid planning policy*

cal Planner, and Hybrid Planning Manager, enabling the use of cheap cameras that increasingly improve accuracy of the goal position estimate, during the trajectory execution.

4.3.1 Hybrid planning manager plugin

The Hybrid Planning Manager plugin I developed for my robot implements the control policy depicted in Figure 4.5. Initially, the 3D position of the target is captured using the base stereo camera. Based on this position, a trajectory is generated by the Global Planner and the execution begins through the Local Planner. While the robot is moving, the end-effector camera continuously refines the estimation of the target position. If the newly estimated position deviates significantly from the previous one (with a threshold of 2 cm), the Local Planner triggers the Hybrid Planning Manager. In response, the Hybrid Planning Manager calls the Global Planner to re-plan the trajectory towards the updated goal. This re-planning is performed online, without stopping the robot, allowing for a smooth and responsive adaptation to the updated target location.

The Hybrid Planning Manager is inspired by the 'single_plan_execution' plugin, with the addition of the following method:

```
1    ReactionResult ContinousPlanner::react(const std::string& event);
```

This addition allows the Hybrid Planning Manager to handle string-based events triggered by the Local Planner. The event of interest, marked as 'KIWI_POS_VALID,' signals the Global Planner to wake up and re-plan a new trajectory from the current position to the newly estimated target. This target position is read from a ROS topic published by the Local Planner.

4.3.2 Global planner plugin

The Global Planner plugin I developed is designed to support both the *OMPL* (Şucan et al., 2012) and the *Pilz Industrial Motion Planner* pipelines, which are two of the main trajectory planning pipelines available in MoveIt. The plugin takes as input a *MotionSequenceRequest* (Ioan Sucan, 2023) where multiple waypoints can be specified, along with the velocity constraints for each waypoint. The starting point of the trajectory is automatically determined from the current state of the robot, ensuring seamless integration with the robot's ongoing operations. In this way, the Global Planner can dynamically adapt to the current positioning, making it efficient and responsive.

For the final implementation, the OMPL pipeline was chosen, specifically using the RRTConnect algorithm (Kuffner et al., 2000) alongside motion adapters to improve trajectory planning efficiency. RRTConnect is well-suited for quickly finding valid paths in high-dimensional spaces, making it a strong choice for complex planning tasks. By combining OMPL with motion adapters, the plugin achieves a robust and flexible planning solution that efficiently navigates through multiple waypoints while respecting velocity constraints and the overall motion plan structure.

4.3.3 Local planner

As described in the MoveIt documentation, the Local Planner is composed by two plugins:

- Trajectory Operator plugin, that basically unfolds the global trajectory piecewise, giving to the Local Contraint Solver a much smaller trajectory to work on. In this project, this plugin has been implemented using the standard 'simple_sampler' example given by MoveIt.
- Local Contraint Solver plugin, that actually produces the robot commands based on the reference trajectory and the local constraints and integrates dynamic inputs coming from sensors. This plugin has been customized to dynamically integrate the estimate of the kiwifruit position coming from the end-effector camera.

Considering that this plugin receives the bounding box of the tracked kiwi by the Yolobot Node (more details in Chapter 5), it can compare the size of this bounding box with the one obtained by the base camera. This comparison is expressed by this ratios:

$$\begin{aligned} R_x &= f_x^R \frac{w_{ec}}{w_{bc}} \\ R_y &= f_y^R \frac{h_{ec}}{h_{bc}} \\ R_z &= (R_x + R_y)/2 \end{aligned} \tag{4.1}$$

where w_{ec}, w_{bc}, h_{ec}, h_{bc} are the width and the height of the kiwi bounding box measured by the end-effector camera and the base camera respectively; f_x^R and f_y^R are

normalization terms, introduced to compare dimensions of different bounding boxes coming from two different cameras. They are simply the ratio of the focal lengths:

$$f_x^R = \frac{f_x^{ec}}{f_x^{bc}}$$
$$f_y^R = \frac{f_y^{ec}}{f_y^{bc}}$$

Therefore, the R_z term is used to estimate the z position of the kiwi with respect to the end-effector camera. Then, also the x and y in that frame can be estimated:

$$\begin{aligned} \hat{z}^{ee} &= R_z z^{bl} \\ \hat{x}^{ee} &= \hat{z}^{ee} \frac{i_{bb} - c_x}{f_x^{ec}} \\ \hat{y}^{ee} &= \hat{z}^{ee} \frac{j_{bb} - c_y}{f_y^{ec}} \end{aligned} \tag{4.2}$$

where z^{bl} is the z coordinate of the kiwi estimated by the base camera and expressed in the base link frame, i_{bb} and j_{bb} are the end-effector camera image coordinates of the centre of the kiwi bounding box, c_x and c_y are the coordinates of the end-effector camera principal points.

The estimated coordinates are then transformed into the base link frame. The entire procedure is summarized in Figure 4.6: the block R_z implements equations (4.1), while the Projection block implements equations (4.2). The remaining steps involve transformations based on the kinematic model. The output consists of the estimated coordinates of the goal, expressed in the base link frame. As mentioned, if their distance from the previous valid estimate exceeds a certain threshold (for a certain amount of time to filter misleading estimates), the "KIWI_POS_VALID" message is sent to the Planning Manager, which triggers the Global Planner to plan a new trajectory toward the updated goal on-the-fly.

It should be noted that this procedure is activated only when the arm is positioned under the kiwi, providing a valid position in message (Listing 4.1). This is done to avoid corrections during other phases. When activated, the kiwi closest to the centre of the frame is chosen for tracking.

All this procedure is repeated at a 50 Hz rate, even if the Yolobot Node runs at a lower rate. The reason is that the Local Planner produces robot commands independently from the camera corrections, therefore it is necessary that it runs at a higher rate.

4.4 Changes to the Moveit Hybrid Planning source code

In addition to the development of the plugins I previously discussed, I also modified the Hybrid Planning Manager component, which is part of the MoveIt source code.

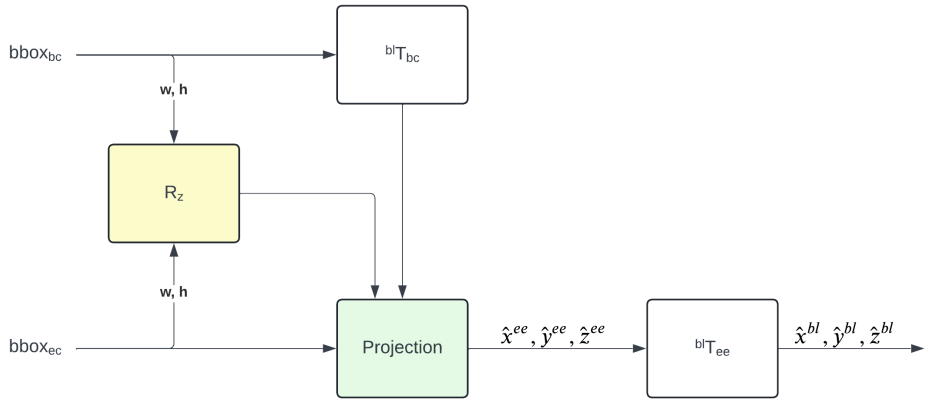


Figure 4.6: *Goal position estimation based on the end-effector camera, implemented in the Local Constraint Solver plug-in*

The goal of these modifications was to enhance the ability to transmit more information from my application to the Local Planner. Specifically, I made two key changes:

- I used the first two points of the 'reference_trajectories.cartesian_trajectory' field of the *MotionPlanRequest* to transmit the coordinates of the centre of the bounding box along with its dimensions (width and height) from the application node to the HP Manager.
- Using the message in the Listing 4.1, I enabled the transmission of not only the position of the bounding box centre but also its width and height in pixels, using the x and y of the size from the HP Manager to the Local Planner. This allows the Local Planner to compare the estimated size of the bounding box with the one obtained from the base camera and, consequently, adjust the position along the z-axis as explained previously.

```
# The 3D position and orientation of the bounding box center geometry_msgs/Pose center
Point position
    float64 x
    float64 y
    float64 z
Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w

# The total size of the bounding box, in meters, surrounding the object's center pose.
geometry_msgs/Vector3 size
    float64 x
    float64 y
    float64 z
```

Listing 4.1: *BoundingBox3D message from the vision_msgs ROS2 package Allevato, 2022*

5

Software structure and simulation

This chapter deals with the software structure of the robotic system, describing the developed ROS packages and their interactions. The chapter provides an in-depth explanation of each package, detailing their roles, functionalities, and how they work together to achieve the overall system objectives. Additionally, a significant part of the chapter is dedicated to the simulation environment, which has been heavily utilized to develop and test the application.

5.1 Modules architecture

The following is a list of the developed packages with a brief description. They are grouped by their functional role.

Moveit packages:

- *custom_robot_cr_arm_ikfast_plugin*: this package contains an inverse kinematics (IK) plugin specifically designed for use with the MoveIt motion planning pipeline. It was generated using the IKFast tool from OpenRAVE, following the steps outlined in the MoveIt IKFast tutorial (`https://moveit.picknik.ai/main/doc/examples/ikfast/ikfast_tutorial.html`). This plugin provides an efficient and fast solution for computing the inverse kinematics of the robot, enabling precise movement and positioning during robotic tasks. By leveraging IKFast, this plugin ensures real-time performance suitable for complex manipulations.
- *cr_moveit_config*: this package provides the MoveIt configuration for the custom robot, generated using the MoveIt Setup Assistant based on the URDF description of the robot. It includes essential configuration files for motion planning, joint limitations, and other parameters necessary for seamless integration with the MoveIt framework. Notable files include:
  - `joint_limits.yaml`: defines the upper and lower bounds for each joint, ensuring the robot operates within safe joint ranges (according to Table 2.3).
  - `ompl_planning.yaml`: contains the configuration for the OMPL (Open Motion Planning Library), specifying the planners and planning settings to be used for pathfinding.

  - `pilz_cartesian_limits.yaml`: holds configuration details for the Pilz Industrial Motion Library, specifying parameters for safe Cartesian movement.

- *manipulator_description*: this package contains the description files for the manipulator robot, primarily in the form of a URDF. The main URDF file has been crafted using the Xacro scripting language, which enhances readability, reusability, and maintainability by allowing the use of macros and parameterization. This makes it easier to manage complex robot models, modify components, and adjust parameters without significant rewrites. In the Appendix B, this description file is reported.
- *moveit_hybrid_planning*: this package is part of the native source code for the MoveIt Hybrid Planning pipeline, which integrates both global and local planning strategies for effective robotic motion planning. It has been included in this workspace to introduce modifications as described in Section 4.4, allowing for custom enhancements and specific adjustments needed for the project.
  Moreover, it contains also the Global Planner, Local Planner and Hybrid Planning Manager plugins described in Chapter 4.3.

Hardware management:

- *pylon-ros-camera*: official ROS2 driver for Basler cameras (Basler, 2022).
- *rs_set_param*: this package has been developed to assist in building the dataset described in Section 6.3.2. Rather than containing any direct robot control code, this node is responsible for automatically setting the exposure of two cameras used in data acquisition to different values. It has been used to collect the 'enhanced exposure' dataset that will be discussed in Section 6.3.2.

Communication:

- *hp_interfaces*: this package contains custom ROS interfaces developed specifically for the hybrid planning application. While an effort was made to utilize only standardized ROS messages and services, certain unique requirements necessitated the introduction of custom interfaces: *Move.srv* (Listing 5.1) and *Detection.action* (Listing 5.2). These messages are used for communication from the main application node and the hybrid planning and Yolobot node, as will be discussed in the next section and in Figure 5.2.
- *ROS-TCP-Endpoint*: this package developed by Unity-Technologies (Unity-Technologies, 2022) has been included to communicate with the simulated environment developed in Unity. More details in Section 5.3.
- *serial_comm*: this node is responsible for managing communication with the Arduino board that controls the robot's motors. It serves as a key link between ROS and the hardware, enabling both command and feedback functionalities:

  - Joint Trajectory Transmission: the node receives the desired joint trajectory commands from ROS and transmits them to the Arduino board over a serial

```
bool move_auto
geometry_msgs/Point position
float64 shoulder1_pos
float64 shoulder2_pos
float64 elbow_pos
float64 wrist_pos
---
bool success
```

Listing 5.1: *Move.srv custom message*

```
bool service
---
vision_msgs/Detection3DArray detection
---
```

Listing 5.2: *Detection.action custom message*

bus. This allows precise control over each motor to ensure the robot follows the planned trajectory.

  - Joint State Feedback: in the reverse direction, the node reads the current joint states—position and velocity—from the Arduino, and publishes this information to a ROS topic. This feedback is crucial for maintaining synchronization between the robot's physical movements and the rest of the ROS nodes.
  - Gripper Management: Additionally, the node, in coordination with the Arduino, controls the gripper motor. It also manages the limit switch sensor, ensuring safe operation by detecting and responding to the limits of the gripper's motion.

Main application:

- *yolobot_strongsort*: this package implements the computer vision pipeline responsible for detecting and tracking kiwifruit in the environment. It combines object detection using YOLO (Redmon et al., 2015) with object tracking via the DeepSort (Wojke et al., 2017) algorithm to provide robust real-time visual data. The pipeline interacts with both the Realsense and the Basler cameras to acquire the necessary visual input, and outputs a list of bounding boxes that indicate the locations of the detected kiwifruit. These bounding boxes are then utilized by other nodes to estimate the goal position for trajectory planning, as discussed in the previous chapter. More detailed information regarding the perception pipeline will be given in Chapter 6.
- *hybrid_planner*: this is the main application package and acts as the central supervisor for the entire system. The hybrid_planner node orchestrates all services, actions, and interactions between the different components in this workspace. It oversees the flow of data and ensures smooth coordination between motion planning, vision, hardware control, and other functionalities, making it the core of the

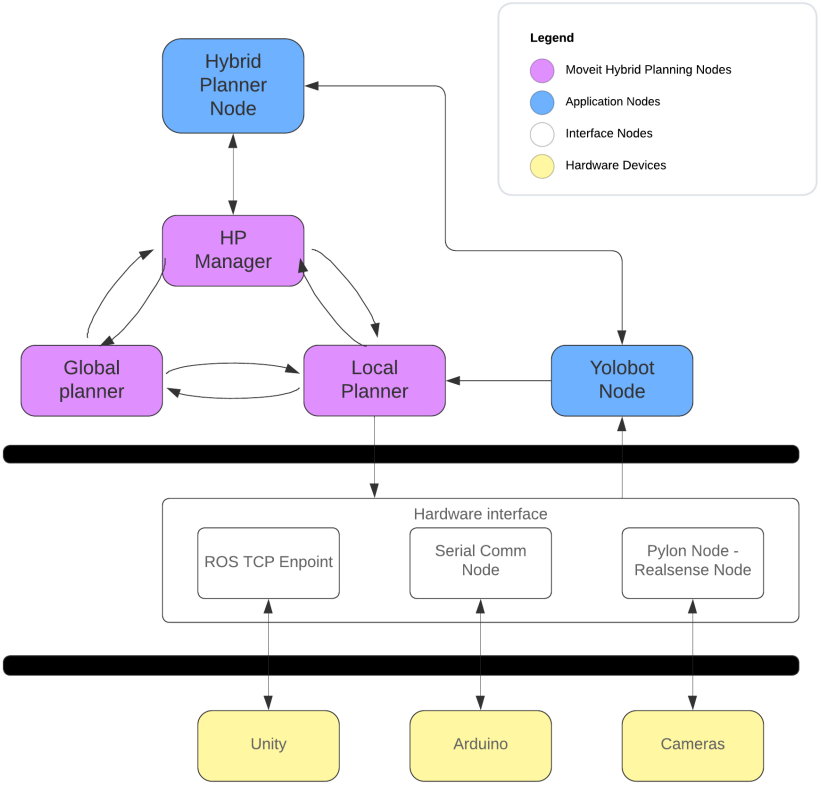


Figure 5.1: *Workspace architecture*

overall robotic process. Given its pivotal role in managing the hybrid planning approach, further in-depth details about the implementation and functionality of this package are provided in the next section.

The diagram in Figure 5.1 sums up how the main nodes interact within the system, depending on whether the real robot or a Unity simulation is being used. It can be notices how the application noded (Hybrid Planner Node and the Yolobot Node) and the MoveIt nodes are hardware agnostic, they interact with the actual hardware through the hardware interface nodes: using the ROS-TCP-Endpoint node if the simulated robot is running or with the Serial Com Node and the camera drivers if the real robot is running.

5.1.1 Arduino firmware

The Arduino firmware implements the motion control, integrating CAN communication for motor control and serial communication for reference trajectory input from the main PC. The system coordinates the movement of the four actuators (plus the gripper), each with specific limits on position, velocity, and torque, as discussed in Chapter 4. The setup function initializes the CAN bus communication to interact with the motors, as well as sets up PWM for handling the gripper.

The control logic resides in the loop function, where the desired positions, veloci-

```
void gravity(float q2, float q3, float q4)
{
  float t2 = G;
  float t3 = sin(q2);
  float t5 = -q3;
  float t4 = len_l[1]*t3;
  float t6 = q2+t5;
  float t7 = cos(t6);
  float t8 = q4+t6;
  float t9 = sin(t6);
  float t10 = cos(t8);
  float t11 = sin(t8);
  float t12 = a2*t7;
  float t13 = cog_len[2]*t9;
  float t14 = len_l[2]*t9;
  float t15 = a3*t10;
  float t16 = cog_len[3]*t11;
  g = {0.0, 0.6*(mass_l[3]*t2*(t4+t14+t15+t16)+m_m*t2*(t4+t14)+mass_l[2]*t2*(t4+t12+t1
  ↪  3)+cog_len[1]*mass_l[1]*t2*t3),
  ↪  -mass_l[2]*t2*(t12+t13)-m_m*t2*t14-mass_l[3]*t2*(t14+t15+t16),
  ↪  mass_l[3]*t2*(t15+t16)};
}
```

Listing 5.3: *Computation of the gravity compensation term*

ties, and torque values are updated in real-time based on feedback from the CAN bus, which receives sensor data from the robot's motors. Additionally, communication between the Arduino and a main PC through the serial interface allows the PC to provide trajectory references and control gains.

The code also includes functions for sending CAN messages to enter motor mode, exit motor mode, and set motor zeroing, ensuring robust communication with the motor drivers.

The calculations of the feed-forward torque mentioned in Chapter 4 are implemented exporting C code directly from the Matlab functions. In this way, all the calculations about the dynamic model have been performed symbolically in Matlab (and reported in the Appendix A) and exported directly in C code. In the Listing 5.3, the obtained function for gravity compensation term computation is reported as an example.

5.2 Application details

The diagram in Figure 5.2 shows the state machine of the Hybrid Planner Node, representing the implementation of the reference trajectory discussed in Chapter 3. The process is initiated through the */move* service, using the custom *Move.srv* message (Listing 5.1):

- If the field `move_auto` is True, the state machine starts and executes the full pick-

ing trajectory.
- If False, the user can specify a particular position in the workspace or a specific joint configuration as the goal, and a trajectory is generated towards that target.

It follows a detailed description of the internal states.

1. READY: a *MotionPlanRequest* is sent to the Hybrid Planning Manager to reach the Home position. This position has been chosen to be such that it does not obstruct the field of view of the base camera. Exploiting the action architecture of ROS2, the subsequent state is reached only when the action is completed and the robot has reached the desired position.
2. HOME: in this state, a call to the Yolobot Node using the *Detection.action* custom message is executed. As described in Listing 5.2, the returning values is a *Detection3DArray*: this message is filled with 2D bounding boxes of the detected kiwifruits (coordinates of the centres and dimensions) plus the z-coordinate of the centre of the bounding box measured by the stereo system. This is the reason why the Detection3D message has been used, instead of the simpler Detection2D message. After receiving this list, a *MotionPlanRequest* is sent to move the end effector under the first kiwifruit in the list.
3. APPROACH: here, the tracking service of the Yolobot Node is activated so that it receive the video signal from the Basler camera and process it accordingly. At the same time, a special *MotionPlanRequest* is sent to the HP Manager: it contains the coordinates of the goal plus the dimensions of the bounding box of the selected kiwifruit; it will be used by the Local Planner to refine the estimate of the goal, as discussed before. Therefore, the approach trajectory is executed with the corrections triggered by the Local Planner.
4. CLOSE GRIPPER: once the goal has been reached, the tracking service of the Yolobot Node is deactivate and 'close gripper' command action is sent to the Serial Com Node. It transmits in turn this command to the Arduino board that manages the gripper motor. The feedback of actual closing is sent back and triggers the next state.
5. DETACHMENT: even if it is not part of the reference task described in Chapter 3, it has been decided to add an additional downwards movement to ensure the detachment and to make sure to not damage the plant or other kiwis during the gripper reopening movement. This short movement is performed with a *MotionPlanRequest* to the HP Manager, choosing simply a point 10 centimetres below the picked kiwifruit as a goal.
6. OPEN GRIPPER: the same action as the CLOSE GRIPPER state is performed, this time opening the gripper e letting the picked kiwi fall into the tube.

This procedure is repeated until the last detected fruit is harvested, so the mobile platform can move forward to the next section.

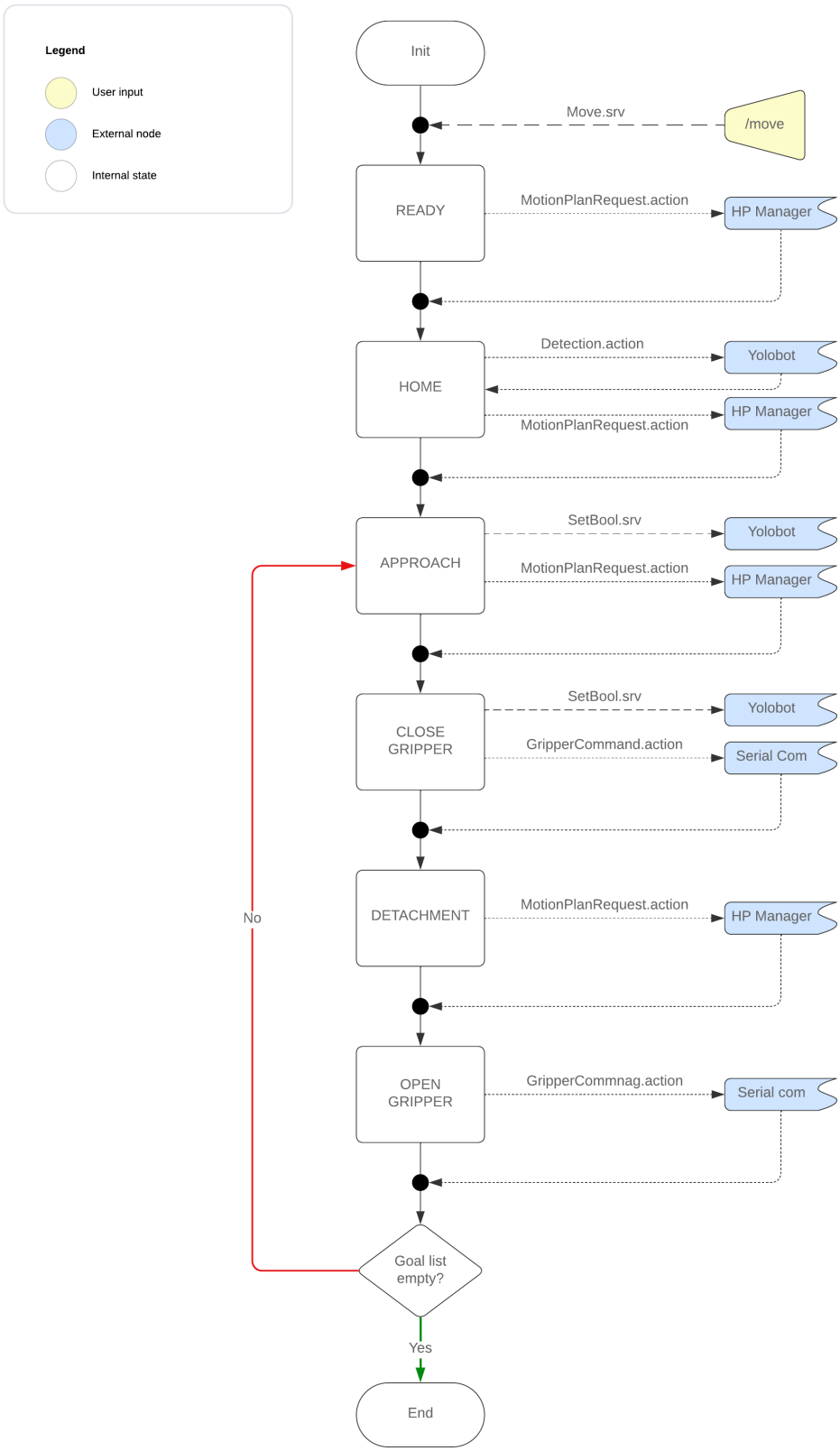
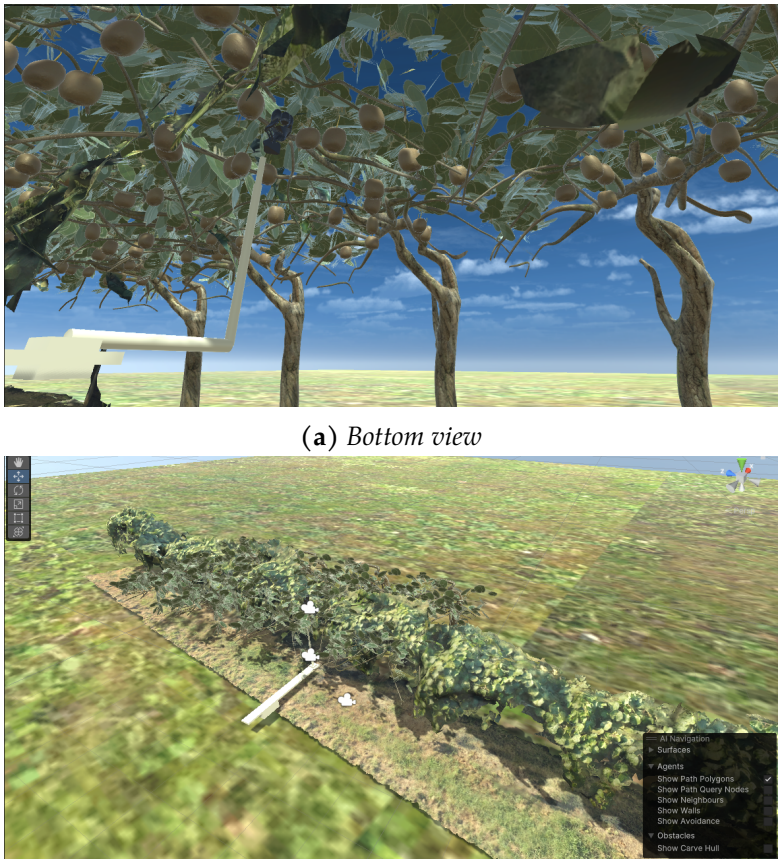


Figure 5.2: *State machine and interaction with other nodes of the hybrid_planner node.*



(**a**) *Bottom view*

(**b**) *Top view*

Figure 5.3: *Simulated orchard in Unity*

5.3 Simulation environment

Unity is a powerful cross-platform development environment, widely recognized for its versatility in creating interactive 3D applications, ranging from video games to simulations and virtual reality experiences. In the context of robotics, Unity offers a unique and valuable toolkit for creating complex, immersive environments that can be used to simulate robotic systems. By leveraging Unity's capabilities, engineers and researchers can develop, test, and refine robotic algorithms without needing access to costly hardware or physical spaces, ultimately accelerating the process of prototyping and deployment.

For my project, I chose Unity primarily due to its high configurability, its strong integration capabilities with ROS, and its ability to produce photo-realistic environments. Unity's flexibility allows users to create detailed virtual worlds with fine-tuned control over various elements, from object physics to lighting conditions. Additionally, interfacing Unity with ROS through the ROS-TCP-Endpoint and the Unity URDF Importer makes it easy to integrate the simulation with existing robotic frameworks, thereby creating a seamless bridge between virtual and physical robotics development.

An example of that can be seen in Figure 5.3, where a virtual kiwi orchard with the robotic harvester model is shown. The real interaction happens under the plant

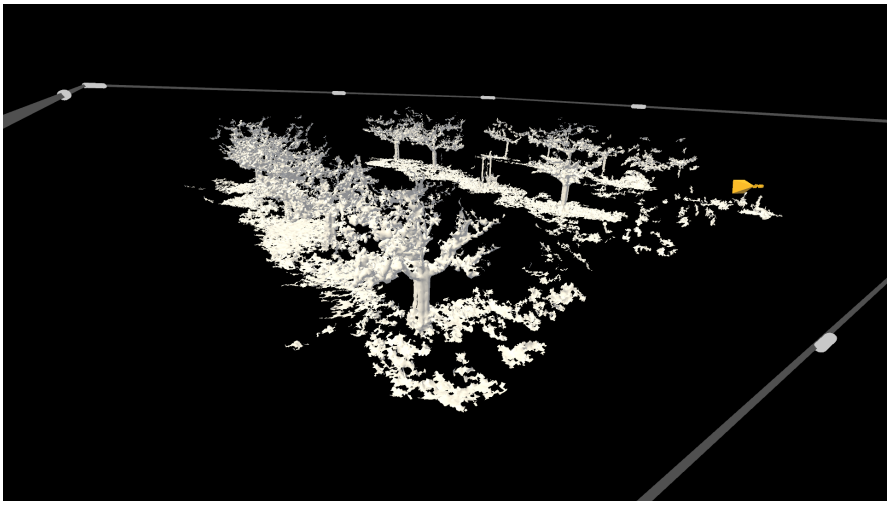


Figure 5.4: *Mesh of the mapped orchard*

(Figure 5.3a), where there is the robot equipped with cameras, interfaced with the ROS control software. The kiwifruits are spawned randomly, so that a different random configuration can be tested. In Figure 5.3b, it is shown the tree line from above. This model has been obtained by photogrammetry using a drone to shot the pictures. This shows that Unity allows to easily use 3D models coming from different contexts.

Another key feature that I particularly took advantage of was Unity's capability to produce photo-realistic scenes. This, combined with the Unity Perception Package (Unity Technologies, 2020), made it an excellent platform for generating synthetic datasets for computer vision tasks such as object detection. By using these tools, I was able to create high-quality, diverse datasets for training machine learning models for object detection, as described in the next chapter.

5.3.1 Application: mixed reality for orchards

Another application where I used Unity was the development of a mixed reality application for HoloLens 2, in collaboration with the Microsoft Mixed Reality & AI Lab. The project focused on creating an immersive experience that allowed users to interact seamlessly with the digital twin of an orchard. By leveraging the HoloLens 2's spatial computing capabilities, I built an application that visualized real-time data and offered intuitive interaction with the virtual representation of the orchard. The mapping and localization aspects were managed by a Microsoft proprietary service: in Figure 5.4, a mesh extracted by the reconstructed pointcloud collected by the HoloLens is shown. I implemented the database to manage the relational data of holograms and developed the mixed reality app to show and insert new data and interact with the databsae. This enabled users to explore environmental conditions, monitor tree health, and understand the impact of different variables on the orchard ecosystem.

The database structure I implemented consists of five main tables (Figure 5.5): FieldArea, Trees, FruitPos, Fruits, and TreeData. Each of these tables plays a crucial

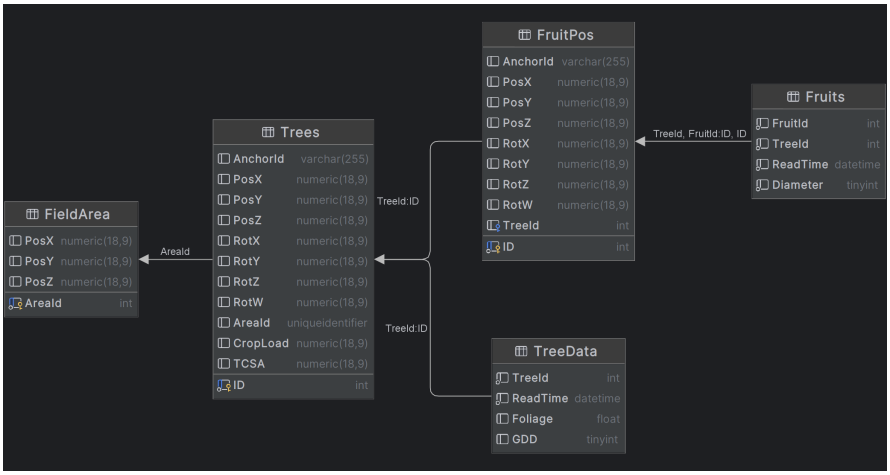


Figure 5.5: *Database structure of the digital twin*

role in organizing spatial and relational data about the orchard. The FieldArea table holds spatial data about different areas of the orchard, containing coordinates (PosX, PosY, PosZ) and a unique identifier (AreaId). The Trees table represents individual trees, containing both spatial positioning and rotational data to precisely define their location and orientation, and links each tree to a specific area using AreaId. The FruitPos table stores spatial positions and orientations of individual fruits, linking each fruit to a specific tree using TreeId. The Fruits table provides information about individual fruits, such as Diameter and the timestamp (ReadTime) of when data was collected. Lastly, the TreeData table contains time-series data for each tree, including attributes like Foliage and GDD (Growing Degree Days), providing a dynamic view of tree health over time.

This database structure allowed us to create a highly detailed and interconnected model of the orchard, which was essential for delivering an accurate mixed reality experience. The spatial and temporal data provided by the database, filled with information from sensors or robots inspecting the real orchard, enabled real-time interaction and visualization of various orchard conditions, making the digital twin a powerful tool for understanding and managing the agricultural environment.

This is exemplified in Figure 5.6, showing a screenshot taken from the HoloLens in the real orchard. The image shows important information about a selected tree, directly downloaded from the database. The displayed data includes the number of fruits, fruit size distribution, fruit size history, and detailed attributes like growth degree days (GDD), foliage volume, trunk cross-sectional area, and crop load. This kind of visualization helps users quickly assess the health and productivity of individual trees in the orchard, highlighting the value of integrating mixed reality with precise, sensor-driven data.

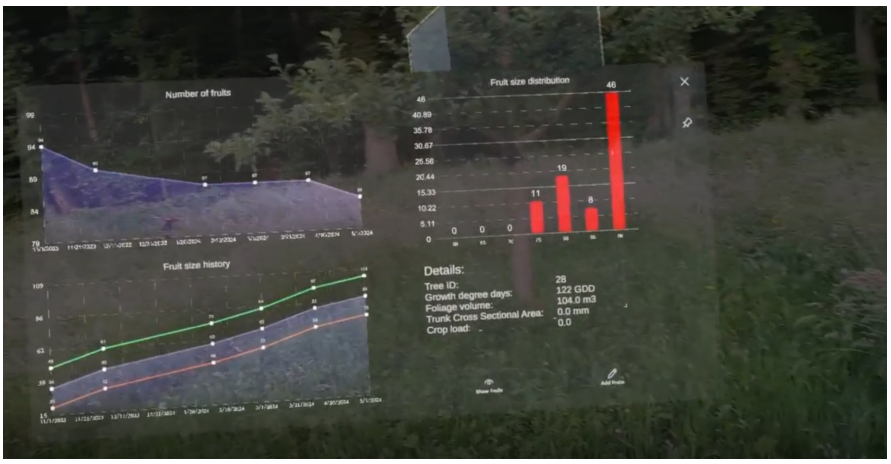


Figure 5.6: *Example datapanel in the mixed reality app*

6

Perception

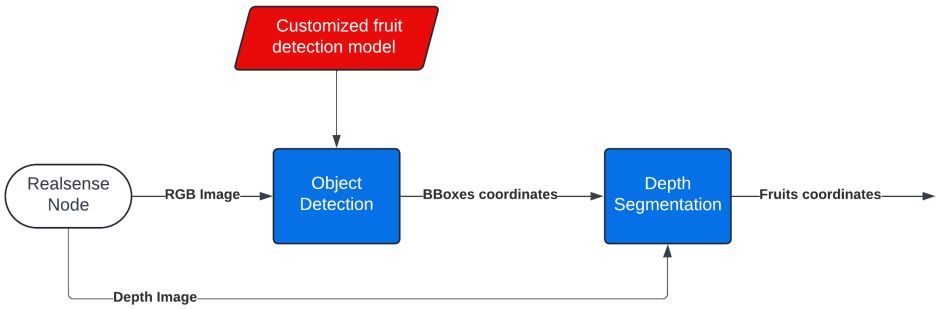
This chapter deals with the perception pipeline of the robotic system. It is shown how the images from the two cameras are processed in order to extract useful information about the position of kiwifruits. The chapter describes the steps involved in object detection and object tracking, leading to accurate localization of the fruits within the environment. The perception system plays a crucial role in enabling the robotic arm to make informed decisions, facilitating effective and precise fruit harvesting. Detailed explanations are provided on the algorithms and techniques used to interpret visual data, ensuring reliable detection and positioning of the target fruits.

6.1 Fruit detection and tracking

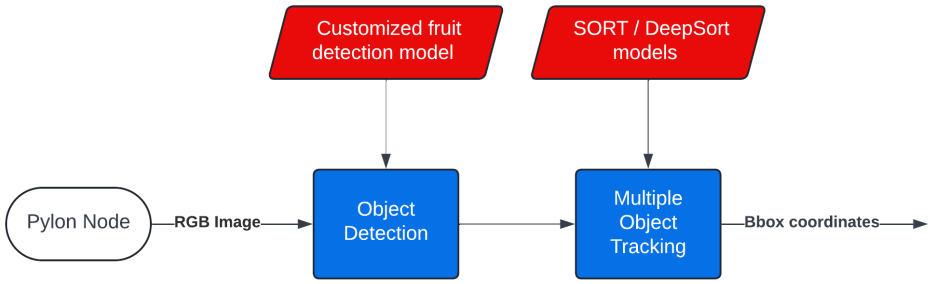
The perception system is a critical component of the robotic kiwi harvester, enabling it to accurately identify, localize, and interact with the fruit in a dynamic orchard environment. The perception architecture integrates two distinct types of cameras, each serving complementary roles in providing visual and depth information to guide the harvesting process. These cameras include an Intel RealSense D435i stereo camera and a Basler Ace RGB camera, introduced to maximize perception accuracy and flexibility.

The Intel RealSense D435i is mounted rigidly to the base link of the robot, providing a stable reference frame for capturing depth information. This stereo camera is capable of generating high-resolution depth maps by combining visual data from its dual lenses with an infrared projector, making it well-suited for detecting depth in an outdoor setting. By capturing a broader view of the environment, the RealSense camera helps in constructing an overall spatial map of the kiwi fruits in the area. This enables the robot to plan efficient paths to navigate and reach target fruits.

In contrast, the Basler Ace RGB camera is mounted on the end effector of the robot, allowing it to gather detailed visual data close to the target fruit. This positioning ensures that the camera has an unobstructed, high-resolution view of each kiwi during the final approach. The Basler Ace is employed for precise visual analysis, accurately estimating the pose of the kiwifruit. This close-up visual information is vital for refining the robot's grasp strategy, ensuring that the fruit is harvested without causing



(**a**) *Base camera perception pipeline*



(**b**) *End-effector camera perception pipeline*

Figure 6.1: *Perception pipelines for the two cameras*

damage to the delicate skin or stem.

The video signal coming from the cameras is processed by the Yolobot node, as mentioned in the previous chapter. In this node, a YOLO (You Only Look Once) (Redmon et al., 2015) model is used for object detection, and a DeepSort (Wojke et al., 2017) model is employed for object tracking (more details in the next paragraphs). The YOLO model enables the system to detect and classify kiwi fruits in real time, leveraging its powerful convolutional neural network to distinguish fruits even in cluttered or partially occluded environments. The DeepSort model, on the other hand, is responsible for tracking the detected kiwifruit across frames, providing robust temporal consistency for each target as the robot moves.

The signal coming from the RealSense camera is processed at the beginning of the harvesting action to identify all visible kiwi fruits, without requiring continuous tracking. As illustrated in Figure 6.1a, the perception pipeline begins by capturing an RGB image from the RealSense camera, which is then passed through a customized fruit detection model using YOLO. The output of this process is a set of bounding box coordinates that indicate the position of each detected kiwi. Once the bounding boxes are identified, the corresponding depth image is segmented to derive the 3D coordinates of the kiwifruit.

The signal from the Basler Ace camera, which is mounted on the end effector, is processed differently. As shown in Figure 6.1b, the RGB image captured by the Basler Ace is fed into the object detection model to identify kiwifruit. In this case, however, an

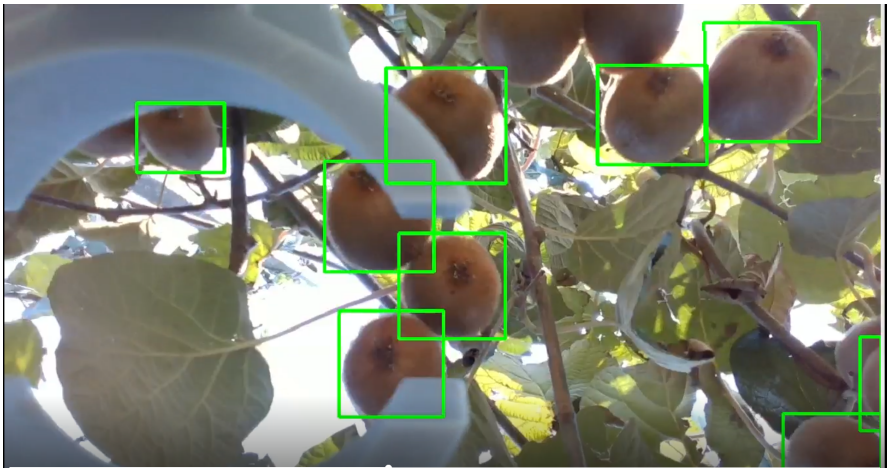


Figure 6.2: *Kiwi detection in real environment*

additional object tracking step is included to ensure that the robot consistently follows a single target kiwi at a time. The tracking process is managed by models such as SORT or DeepSort, which assign unique IDs to each detected fruit and track their positions across consecutive frames. This step is critical for maintaining focus on a specific kiwi during the approach and grasp phase, ensuring that only one kiwi is targeted and harvested at a time.

The output of this pipeline is the image coordinates of the bounding boxes for each detected kiwi, which are subsequently passed to the Local Planner node. The Local Planner selects the appropriate kiwi based on the assigned ID and uses these coordinates to estimate the 3D position of the fruit, as described in Chapter 4. This process allows the robotic arm to accurately plan and execute the harvesting motion, achieving precise alignment with the target fruit.

Figure 6.2 illustrates the output of the Yolobot node processing the end-effector camera, where kiwifruits are detected and highlighted with green bounding boxes. This visual feedback showcases the integration of object detection and tracking, allowing the robotic arm to keep track of multiple kiwifruits within its field of view

6.1.1 Object detection

Object detection in the scene was achieved using the YOLO (Redmon et al., 2015) convolutional neural network, which is a leading, real-time detection system. YOLO operates with a single-shot detection approach, effectively identifying multiple objects within an image with high accuracy by leveraging a Convolutional Neural Network (CNN) that directly processes images and outputs bounding box predictions. The primary reasons for choosing YOLO were its speed, flexibility, available tools, and open-source nature.

Before YOLO, conventional image classifier neural networks were limited to classifying a single image at a time. To detect multiple objects within a scene, the image

needed to be fragmented into smaller pieces, and classifiers would be applied individually to each part. YOLO revolutionized this process by enabling simultaneous detection of multiple objects in a single pass through the neural network, which inspired the name "You Only Look Once." Due to its effectiveness and rapid processing capabilities, YOLO quickly gained popularity. Over the years, the original YOLO model has been improved, resulting in numerous iterations that boast higher accuracy and faster processing speeds. For this particular project, YOLOv5 was chosen due to its speed and compatibility with Python code and the Torch framework at the time of initial implementation.

For this project, the YOLO network was specifically retrained to perform fruit recognition in-field and within tree rows. As YOLO is a supervised learning model, a substantial amount of labeled data was necessary to achieve satisfactory detection accuracy. To meet this requirement, an extensive labeling process was conducted.

Labeling is typically the most labor-intensive part of preparing for object detection, as it involves significant manual effort. Considering the labor-intensive nature of labeling, it was decided to mix real images with synthetic ones produced in Unity, which were automatically labeled by the software. More details will be provided in the Section 6.3. To streamline this process, for the real images Label Studio (Tkachenko et al., 2020-2022) was employed. Label Studio is an open-source, web-based tool that supports multi-user setups and allows for the creation of various labeling projects, whether for image detection or segmentation, as needed.

6.1.2 Object tracking

Fruit tracking across multiple frames is accomplished by employing existing state-of-the-art Multiple Object Tracking (MOT) algorithms. For this purpose, two MOT algorithms were chosen: SORT (Bewley et al., 2016) and DeepSort (Wojke et al., 2017), with DeepSort being an enhanced, deep-learning-based version of the SORT algorithm. The first technique, SORT, is purely algebraic and relies solely on bounding box parameters and confidence scores, while DeepSort includes an image feature matching layer that utilizes several pre-trained neural networks to improve accuracy in matching detections across frames.

The deep learning association network "`osnet_x0_25_msmt17.pt`" was selected for experiments using the DeepSort method. This association network is a community-provided, pre-trained model specifically designed for typical benchmarking scenarios involving people tracking. The selection of the type and dimension of the deep association metric network was influenced largely by the computational resources available for processing multiple fruit detections. Since speed is crucial in this context, a simpler and faster network was preferred over more complex, slower alternatives.

To prevent tracking of the wrong kiwi in subsequent frames, the tracking data generated by the MOT algorithm is utilized for position estimation update in the Local Planner. Each detected object is assigned a unique ID number, which allows it to be

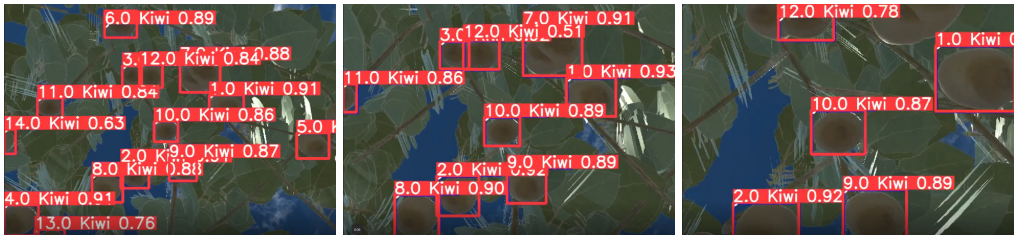


Figure 6.3: *Frame sequence from the approach manoeuvrer: the selected ID (10) stays always in the frame and never changes.*

tracked consistently across frames. Given that kiwis are very similar in appearance, and considering that the selected kiwi is remaining in the image frame during the approach, the selected ID number of the tracked kiwi is not changing during the approach and it is used by the Local Planner to filter the right bounding box (Figure 6.3).

6.2 Application: apple counting

The aforementioned MOT techniques have been employed also for an apple counting application, specifically addressing the differences between planar and spindle orchard types, as described in D. Mengoli et al., 2023. The planar type represents a robotic-friendly orchard structure, allowing all fruits to be exposed without occlusions, whereas the spindle type, being more traditional, often results in significant occlusions caused by the canopy. Experiments were conducted using the robotic platform prototype described in Section 1.3, which involved traversing orchard rows at a speed of approximately 0.6 m/s. This speed was selected based on the performance capabilities of the onboard computational hardware, consisting of a Cincoze DS-1301 automotive PC with an Nvidia RTX A2000 GPU. YOLO detection achieved a rate of 90 frames per second (fps), which decreased to 20 fps when incorporating the DeepSort algorithm. The system's computational resources had to be shared between the left and right cameras, so the cruising speed was configured to ensure proper overlap between consecutive frames at a rate of 10 fps.

Due to the differing cultivar types and expected levels of fruit occlusion, distinct confidence thresholds were set for each type: 0.46 for spindle cultivars and 0.54 for planar cultivars. Additionally, the Intersection over Union (IoU) threshold was set to 0.5. Each row of trees was scanned from both sides to obtain a comprehensive view. For the planar type, where canopy occlusion is minimal, the counts from both sides were averaged. In contrast, for the spindle type, where heavy occlusion is expected, the counts from both sides were summed to derive the total count.

Depending on the actual number of fruits detected, a calibration coefficient may be necessary to best align predictions with reality, particularly for the spindle architecture, where occlusions are more prevalent, or when detecting apples on adjacent rows in the planar type. The default parameters for the SORT algorithm were maintained,

Method	Spindle type			Planar type		
	Side 1	Side 2	Total (sum)	Side 1	Side 2	Total (avg)
SORT	1375	1108	2483	2983	2790	2886
DeepSORT	730	686	1416	1778	1880	1829
Manual	-	-	2298	-	-	1689

Table 6.1: *Comparison of different methods for Spindle and Planar types* (*D. Mengoli et al., 2023*)

while DeepSort parameters were adjusted to compensate for camera movement, requiring a minimum of 2 frames to initiate tracking, a maximum age of 30 frames, and a maximum distance of 0.8 for both matching and gating IoU thresholds.

The results of applying both SORT and DeepSort MOT algorithms in combination with YOLO object detection are presented in Table 6.1. As the performance of the MOT algorithms depends heavily on the detections from the YOLO network, identical parameters were used for both approaches to ensure consistent bounding boxes. The obtained results were subsequently compared with manual fruit counts, serving as a ground truth.

Figure 6.4 illustrates two example images of the tracked apples, showing the different orchard architectures: the spindle type on the left and the planar type on the right. In both images, a few blue bounding boxes are visible without an associated ID number, indicating YOLO detections that have not yet been correctly tracked by the MOT algorithm. It is important to note that during all field tests conducted, using the aforementioned configuration, every object detected was eventually tracked by the MOT algorithm as the video progressed. The MOT algorithms require a valid match for a specific number of consecutive frames before confirming an ID association, and due to the complexity of the scenario or motion blur caused by camera movement, some time may be needed to stabilize apple tracking.

As anticipated, DeepSort outperformed SORT in handling occlusions and reassigning IDs when apples were intermittently obscured by leaves or canopy, which caused the SORT algorithm to overestimate the total count. The spindle type, with its dense canopy, also tended to hide more apples, which resulted in the undercounting seen with DeepSort. To align the automatic counts with the actual number of apples present in the different orchard types, calibration coefficients were necessary in these cases.

The field tests demonstrated acceptable fruit counting performance, particularly in the planar orchard type, which is more favourable for robotic systems. Although some over-counting occurred, largely due to detection of apples in background rows, the overall error remained below 10. The YOLO detection performance played a critical role in achieving these results, as the MOT's accuracy depends on reliable initial detections. In scenarios involving DeepSort, manual verification on a limited set of apples indicated that the detection was highly accurate, with precision and recall exceeding 90, and that assigned IDs remained stable throughout each apple's traversal across the frame sequence.

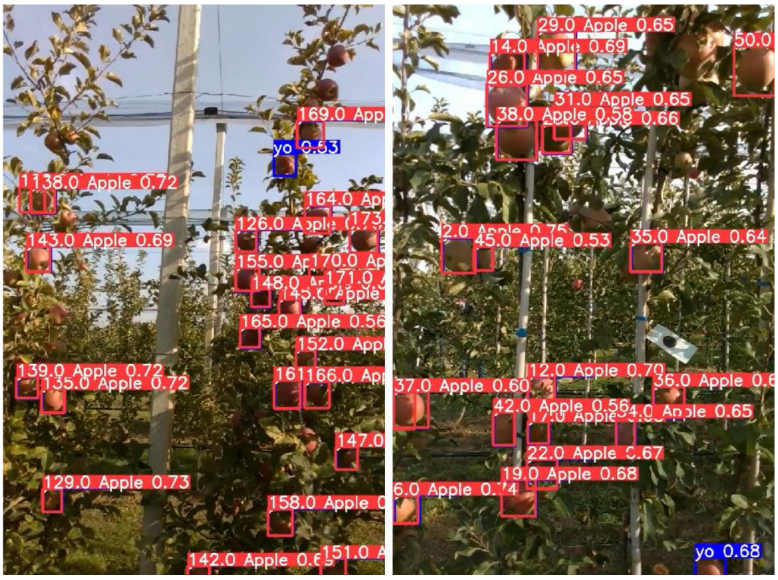


Figure 6.4: *Apples counting application in two different orchard types: spindle(left) and planar (right)*

6.3 Training dataset

6.3.1 Synthetic images

To enrich the object detection dataset for the YOLO detector, I employed Unity with the Unity Perception Package (Unity Technologies, 2020). The primary objective was to generate a large volume of synthetic images, each containing automatically labelled objects that are ready to be used for training the YOLO model (Figure 6.5). This approach significantly reduces the time and labour costs traditionally associated with dataset annotation, as the entire labelling process is automated, avoiding the challenges of manual image labelling.

Unity's capabilities as a 3D development environment provided several advantages for creating the object detection dataset. The Unity Editor allows users to construct virtual environments that simulate real-world settings, complete with custom lighting, textures, and backgrounds. Using these simulated environments, I was able to generate images featuring target objects from different angles, lighting conditions, and contexts, ensuring a high level of diversity in the dataset. The synthetic nature of this method also allows for precise control over object placement, movement, and visibility, which helps in capturing edge cases that would be difficult to gather in real-world scenarios.

The Unity Perception Package further amplified the value of this synthetic data creation process. The Perception Package, an open-source extension from Unity, is specifically designed to facilitate the collection of computer vision datasets. With features such as randomization and built-in labelling tools, I was able to generate diverse scenes where parameters such as object position, scale, and background light could be varied automatically. The package also comes equipped with a Labeller component that assigns annotation tags to all specified objects in a given scene, generating

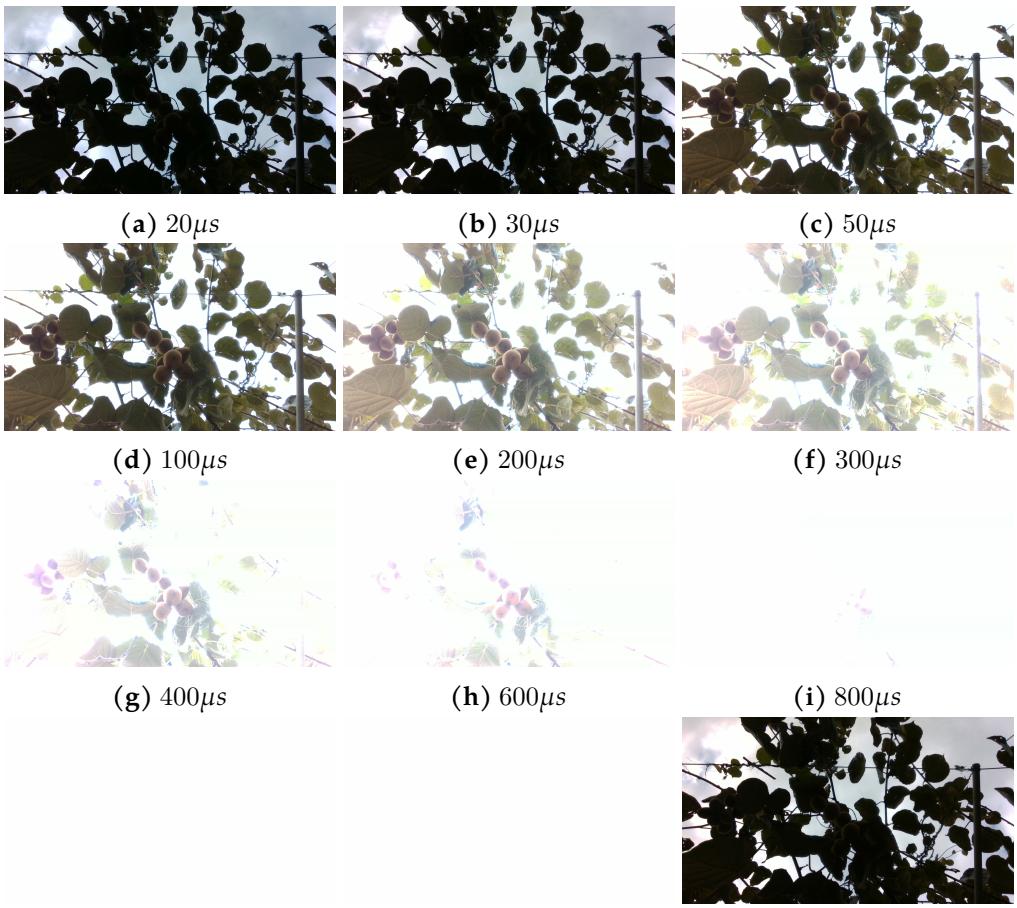
Figure 6.5: *Images captured from the Unity simulator for data augmentation with the highlighted auto-labelled objects*

metadata in a COCO-compatible JSON format. This automated labelling ensured that every frame generated through Unity came with high-quality, consistent annotations.

The advantage of using Unity and the Perception Package lies primarily in the reduction of human labour traditionally required for dataset labelling. Manual labelling is not only time-consuming but also prone to human error, particularly when dealing with large datasets involving numerous classes and images. By generating images within Unity, the labelling of object bounding boxes is done automatically, achieving pixel-perfect accuracy. Moreover, this automation makes it feasible to generate an enormous dataset in a fraction of the time it would take using manual methods. The resulting dataset is rich with variability, which helps prevent over-fitting and encourages the YOLO model to learn generalizable features.

6.3.2 Enhanced exposure images

The enhancement of the training dataset for the YOLO object detection model was critical to address issues arising from our specific imaging environment. The primary camera used for image acquisition was the Intel RealSense, which is optimized for indoor applications, while our initial dataset was gathered in outdoor conditions. This mismatch led to significant challenges in detection performance due to varying lighting conditions. Even minor fluctuations in weather or time of day caused drastic drops in model accuracy. To mitigate these issues, we expanded the dataset to include images captured at different exposure levels and across different times of the day. Specifically, images were collected with the following exposure times $[\mu s]$: 20, 30, 50, 100, 200, 300, 400, 600, 800, 1200, 1600, and with autoexposure enabled. However, it is important to note that not all the collected images were added to the dataset: some of them were too bright or too dark (as shown in Figure 6.6). Only the images where the kiwi were



(**a**) 20μs (**b**) 30μs (**c**) 50μs

(**d**) 100μs (**e**) 200μs (**f**) 300μs

(**g**) 400μs (**h**) 600μs (**i**) 800μs

(**j**) 1200μs (**k**) 1600μs (**l**) *Autoexposure*

Figure 6.6: *Same pictured taken at different exposure time, added to the training dataset*

distinguishable were included. This enhancement aimed to diversify the lighting scenarios and provide more robust training data for YOLO. An example can be observed in Figure 6.6l, where we demonstrate the inadequacy of the Realsense auto-exposure algorithm in this situation; the images tend to be underexposed as the brighter background overwhelms the dimmer foreground, since the camera is always pointing upwards and the kiwi tree covers the sky. By carefully augmenting the dataset in this way, we aimed to make the YOLO model more resilient to the dynamic lighting conditions typically encountered in our environment.

A very important future development is to write a custom autoexposure algorithm that works better for this situation. One possible approach for a custom autoexposure algorithm could be based on the values from the depth image. For instance, we could filter the RGB image by selecting only the pixels that fall within a certain depth range, effectively performing RGB segmentation based on depth. This filtered foreground could then be used to regulate the image exposure. In this way, even if the background becomes overexposed, the foreground containing the kiwis would remain properly exposed, improving detection performance.

7

Results

In this chapter, the results of the harvesting experiments are presented, including both simulations and real-world tests conducted in an actual orchard. The performance of the system is analysed, highlighting the successes and identifying areas of improvement. Additionally, the non-ideal characteristics of the built prototype are critically evaluated, such as limitations in accuracy, presence of non linear friction and cameras calibration process. This chapter provides valuable insights into the practical challenges encountered during implementation, and discusses the lessons learned that could inform future improvements.

7.1 Simulation

As introduced before, the robot and the environment are simulated in Unity while the results are shown in RViz (Figure 7.1). The simulation environment provides the state of the robot, the feedback from the cameras (Figure 7.2) and actuates the robot joints. In each test, a random number of fruits is spawned in random positions on the foliage of the tree to test the perception and planning algorithms in different situations. The results are summarized in Table 7.1: it shows the number of actual fruits present in the image frame captured by the base camera, the number of fruits detected, the ones that are in the reachable workspace, the number of the actually reached fruits using the tracking algorithm and the duration of the experiment.

Considering that the aim of these simulations is to validate the developed perception pipeline, the physics of the grasping and the detachment forces are not considered: a 10 seconds pause is introduced to take into account the opening and closing time of the real gripper.

From Table 7.1, the result is that the 80% of the fruits detected in the workspace are successfully reached with a 25.8s average time. This result can be easily improved considering that the robot is stationary during the gripper opening phase and the harvesting order of the fruit is not optimised to reduce the travelling time.

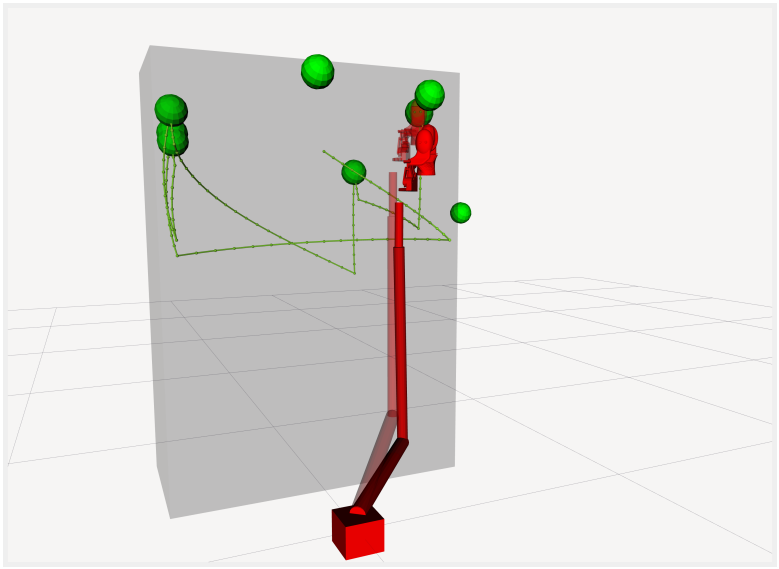
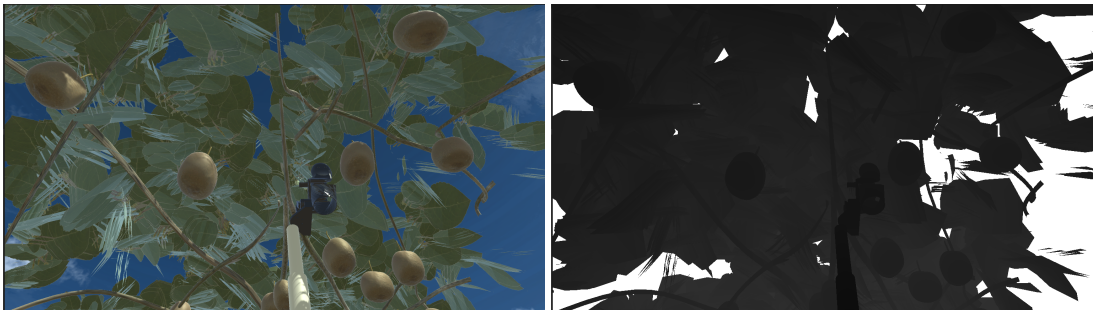


Figure 7.1: *Solid red: actual state of the robot. Transparent red: target state. Green balls: position of the detected fruits. Green line: planned trajectories. Gray box: workspace of the robot.*



(**a**) *RGB image* (**b**) *Depth image*

Figure 7.2: *Output of the simulated base camera in Unity*

	In frame	Detected	In workspace	Reached	Time [s]
1	13	12	5	5	112
2	13	11	4	3	83
3	10	9	3	2	53
4	12	11	6	5	121
5	13	11	6	4	107
6	9	6	4	3	83
7	10	8	1	1	27
8	12	10	3	3	83
9	9	9	3	2	53

Table 7.1: *Results of the simulated harvesting experiments: number of fruits present in the image frame captured by the base camera, fruits detected by the computer vision pipeline, fruits present in the reachable workspace, actually reached fruits using the tracking algorithm and the duration of the experiment*

7.2 Real robot

The robot shown in Figure 7.3 represents the finalized prototype designed for autonomous harvesting tasks, which was tested during a field campaign between late October and early November 2023. In Figure 7.3a, the robotic arm is shown in a typical working configuration inside the kiwi orchard. Figure 7.3b depicts the whole system (Hammerhead plus the robotic arm and the electrical box).

Despite our best efforts, numerous delays occurred during the mechanical construction phase, significantly affecting the overall schedule. As a result, we had approximately three weeks to develop and test the software component of the system, followed by a rushed one-week period for calibration and integration. These constraints left us with little time for thorough testing and refinement of the complete system in the field.

The field tests faced several practical challenges that limited the collection of meaningful data. One significant issue arose with the depth-sensing camera; the infrared-based RealSense sensor struggled under direct sunlight, leading to unreliable distance measurements and difficulty in the detection and localization of kiwifruit within the canopy.

Consequently, we gathered limited results. The most successful measurement recorded, involved a full sequence of operations:

1. the robot moved to the home position to avoid obstructing the field of view of the base camera;
2. the robot detected the kiwis in the area (using the base camera);
3. it started harvesting the detected kiwis one by one;
4. upon completing the list, it returned to the home position;
5. finally, the rover moved a step forward to a new harvesting area.

Since there was no connection between the rover navigation PC and the manipulator PC, the rover was manually activated once the manipulator emptied his workspace.

(**a**) *Harvesting manipulator* (**b**) *Entire robot*

Figure 7.3: *The real robot in a kiwi orchard*

The results of this sequence is summed up in Table 7.2. The average time has been measured not counting the rover activation and movement time because, as said before, it was manually activated.

7.2.1 Friction

During the development of our robotic prototype, we encountered significant challenges relating to non-linear friction, particularly in the first two joints of the robot. Unlike the standard viscous friction model, which typically assumes that friction is linearly proportional to velocity, our observations revealed substantial discrepancies between this classical model and the actual behaviour exhibited by these joints. This inconsistency led to considerable inaccuracies in motion control and precision, ultimately requiring an alternative approach for a more realistic representation of friction.

The friction present in the first two joints was especially problematic due to its complex nature, which was not adequately captured by a simple viscous friction model. Instead, a more nuanced approach was needed—one that combined viscous friction with both Coulomb friction and static friction to more accurately depict the observed dynamics. The non-linear characteristics were found to be especially pronounced during low-velocity motion or at the point where the joint transitioned from rest to movement, where static friction effects dominated.

No of harvested fruits	6
Total No of fruits	9
Not detected	3
Failed grasps	0
Avarage time per fruit	25.2 s

Table 7.2: *Results of the real harvesting experiment*

To address this, we adopted a friction model that integrated multiple components of friction: viscous, Coulomb, and static friction. This model, illustrated in Figure 7.4, provides a more comprehensive depiction of the friction forces acting on the joints. The model acknowledges the discontinuity at zero velocity, which represents the sudden change from static friction to kinetic friction as the velocity moves away from zero. This characteristic allowed us to better approximate the behaviour of the joints in various operating conditions.

The contribution of the viscous friction is represented by the matrix D used in control scheme in Figure 4.1, and this is its estimated value:

$$D = \begin{bmatrix} 0.2 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.1 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

The term incorporating the static and Coulomb friction is estimated to be as follows:

$$F_s = \begin{bmatrix} \phi_1 \\ \phi_1 \\ 0 \\ 0 \end{bmatrix} \tag{7.1}$$

where

$$\phi_1 = \begin{cases} 4\ \text{sign}(\omega) & \text{if } |\omega| < 0.01, \\ 2\ \text{sign}(\omega) & \text{otherwise} \end{cases}$$

As said before, this term contributes only to the joint 1 and 2, as expressed by the (7.1).

The combination of these frictional forces, particularly the role of static friction at low velocities and the switch to kinetic friction as velocity increased, was crucial in accurately modelling the non-linear effects observed in our prototype's joints. With this refined friction model in place, we were able to implement control strategies that compensated for the non-linearities more effectively, leading to improved precision and stability in the movement of the first two joints

7.2.2 Basler camera calibration

It was crucial to note that the Basler camera, in its default state, was not calibrated. Without proper calibration, the measurements obtained from the camera would have been inherently inaccurate due to distortions and incorrect mapping between the 2D camera image and real-world 3D coordinates. Therefore, to ensure the reliability of the fruit position measurements, a thorough camera calibration process was conducted.

Camera calibration is essential to correct lens distortions and determine the intrinsic and extrinsic parameters of the camera. Intrinsic parameters account for the internal characteristics of the camera, such as focal length, optical centre, and lens distortion

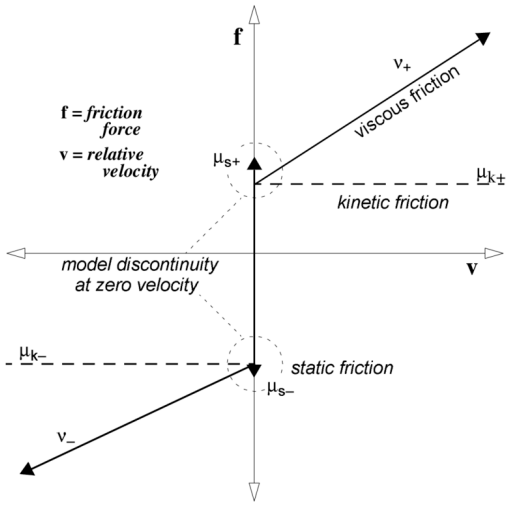


Figure 7.4: *Viscous friction combined with Coulomb friction and static friction* (*Marchi, n.d.*)

coefficients. Extrinsic parameters relate to the position and orientation of the camera in the physical space, allowing conversion between world and image coordinates (more details in the next subsection). The combination of these parameters enables precise mapping between the camera's 2D images and the 3D spatial environment, which is critical for accurately determining the position of objects within the camera's field of view.

The calibration process was performed using the OpenCV library (Bradski, 2000), a popular open-source computer vision toolkit. We employed a traditional method using a chessboard pattern as a calibration target. This approach involved capturing multiple images of the chessboard from different orientations, ensuring the calibration target was positioned at various angles and distances within the camera's field of view. The chessboard provides a set of known points that can be consistently identified across all captured images, which allows for the accurate calculation of camera parameters.

The OpenCV calibration process is based on Z. Zhang, 2000 and can be summarized as follows:

1. Image Acquisition: first, several images of the chessboard were captured using the Basler camera (Figure 7.5). These images were taken at different angles and distances to ensure that the entire field of view was well represented.
2. Feature Detection: the next step involved detecting the corners of the chessboard squares in each captured image. The OpenCV function `findChessboardCorners()` was used for this purpose, which identified the precise positions of the chessboard corners.
3. Corner Refinement: the initial corner detections were refined using the `cornerSubPix()` function in OpenCV to increase the accuracy of the detected points. This step ensures that the detected points are as precise as possible, which is critical for a

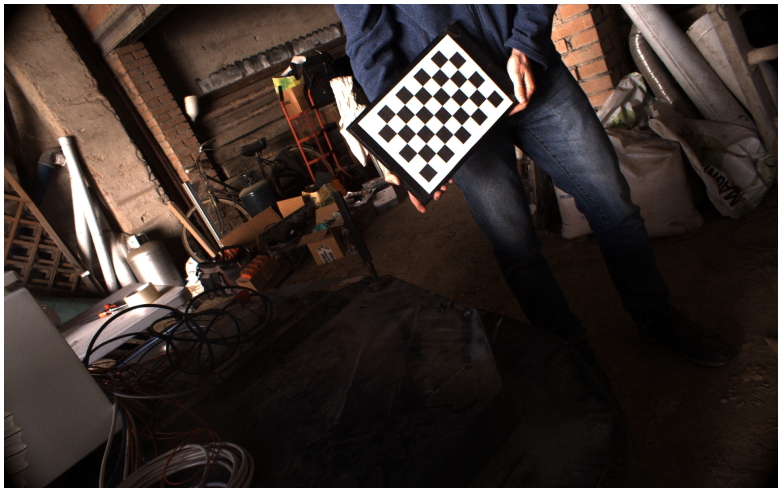


Figure 7.5: *Example image taken from the Basler camera with the calibration chessboard*

successful calibration.

4. Calibration Calculation: once all corner points were detected across the multiple images, the OpenCV function `calibrateCamera()` was used to compute the intrinsic and extrinsic parameters. This function applies optimization techniques to minimize the re-projection error, which represents the difference between the projected image points and the observed image points.
5. Undistortion: finally, the camera distortion coefficients obtained from the calibration were used to correct the images. Using the `undistort()` function, we were able to obtain distortion-free images, which were subsequently used to accurately determine the position of the fruits.

Through this calibration process, we successfully obtained reliable and accurate intrinsic parameters for the Basler camera, allowing us to mitigate the effects of lens distortion and ensure that the positions of fruits in the camera's field of view could be measured with a high degree of accuracy. The resulting parameters are summarized in Table 7.3.

Parameter	Value
f_x	1.416643262225107947e+03
f_y	1.423185284844755415e+03
c_x	9.738269421470386078e+02
c_y	5.960388940211554427e+02
K_1	-7.501987522029342215e-02
K_2	4.950160935461863504e-02
K_3	-6.346295618547725037e-04
K_4	-2.410154366788519280e-04
K_5	3.182615011319356868e-01

Table 7.3: *Basler estimated parameters after calibration*

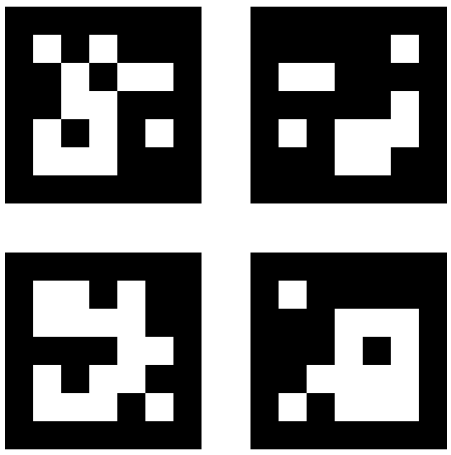


Figure 7.6: *ChAruco marker used for the kinematic calibration*

7.2.3 Kinematic calibration

Eye-in-hand refers to a specific camera configuration where the camera is attached to the end-effector of the robot. In this setup, the camera moves with the robotic arm, giving it a changing viewpoint as the arm moves. This is distinct from the "eye-to-hand" setup, where the camera is fixed in a position overlooking the workspace, while the robot arm moves independently.

The calibration aspect involves finding the transformation matrix between the camera's coordinate system and the end-effector's coordinate system. This transformation describes how the camera is positioned relative to the robot arm, including the rotational and translational components. The transformation is usually represented as a 4x4 homogeneous matrix that contains information about both the orientation and the location of the camera with respect to the robot.

The calibration process generally requires:

- Pose Data Acquisition: Gathering multiple images of a known calibration object (like the chAruco depicted in Figure 7.6) from different perspectives as the robot arm moves.
- Mathematical Solving: Using optimization and computer vision techniques to determine the geometric relationship between the camera and the end-effector. This is typically done by applying algorithms like Tsai et al., 1989, which involves using the camera's and robot's poses to estimate the transformation.

The resulting transformation between the end-effector camera frame and the end-

effector using this procedure is the following:

$$ {}^{ee}T_{ec} = \begin{bmatrix} 0.9596230 & 0.0055689 & -0.2812344 & 0.15761462 \\ -0.0081342 & 0.9999353 & -0.0079551 & 0.00137721 \\ 0.2811718 & 0.0099215 & 0.9596062 & 0.19915025 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Eye-to-hand calibration instead involves calibrating a vision system where the camera is fixed in position relative to the robotic workspace rather than being attached to the robot arm itself. In this setup, the camera ("eye") is usually mounted at a fixed location, such as above or to the side of the robot's workspace, while the robotic manipulator ("hand") moves within the field of view of the camera.

The goal of eye-to-hand calibration is to determine the relationship between the camera's coordinate system and the robot's coordinate system so that the robot can use visual information to accurately locate and interact with objects in the environment. This type of calibration involves finding the transformation that relates the camera's view to the coordinate system of the robot base, allowing the camera to observe the position of objects and convey that information to guide the robot's actions.

In this case, using the Shah, 2013 algorithm on collected corresponding robot configurations and pose data of the calibration object, the transformation between the base link of the robot and the base camera is the following.

$$ {}^{bl}T_{bc} = \begin{bmatrix} 0.9998381 & 0.0022618 & 0.0178503 & -0.02572462 \\ -0.0103003 & 0.8853767 & 0.4647601 & 0.01588668 \\ -0.0147531 & -0.4648688 & 0.8852567 & 0.30584433 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

7.2.4 Mechanical performances

To evaluate the repeatability and positioning accuracy of the developed prototype robot, a series of experiments were designed and conducted, as depicted in Figure 7.7. The experimental setup included mounting rigid bars in close proximity to the robot's working envelope. These rigid bars provided a fixed reference against which the actual displacement of the robot could be measured. The key objective was to investigate how accurately and consistently the robot could move along predefined paths, allowing us to quantify both systematic and random errors in the positioning system.

For these experiments, the robot was programmed to move along the path defined by the rigid bars. Several target positions were selected along the bars, and the robot's end-effector was commanded to move to these positions. After each commanded movement, manual measurements were taken using calibrated tools to compare the actual position of the robot's end-effector against the desired position. The differences between the desired and actual positions were recorded as the positioning errors.

Figure 7.7: *Setup for mechanical performance measurement*

Repeatability was determined by analysing the deviation of the robot's end-effector from the same target position over successive trials. The consistency of the results allowed us to understand the level of variability present in the robot's actuation system, highlighting potential mechanical or control issues that could influence the robot's precision. The results of this experiment are reported in Table 7.4: we let the robot move to that particular configuration and measured the error between the actual position and the theoretical position given by the forward kinematics function.

In the same way, we measured positioning accuracy starting from some know position and letting the end-effector move along a predefined axis. Then, we measured the positioning error tacking advantage of the metal bars. The results are reported in Table 7.5.

The resulting repeatability mean error is 15.85mm and the positioning accuracy mean error is 7.0mm. Of course, these experiments are not complete and should be repeated in the entire workspace to obtain a valid measure, but the goal was just to have an idea of the robot mechanical performances.

Joint configuration	Test number	Error [mm]
(-0.27443, 2.10868, 2.26933, 1.00903)	1	9
	2	10
	3	10
	4	15
	5	8
(-0.34653, 2.20634, 2.49402, 0.79617)	1	16
	2	22
	3	13
	4	17
	5	18
(-0.18593, 2.20939, 1.82529, -0.85028)	1	-18
	2	-11
	3	-5
	4	-4
	5	-8
(-0.13519, 2.03964, 1.73489, 0.45399)	1	-26
	2	-25
	3	-20
	4	-30
	5	-22

Table 7.4: *Repeatability measured starting from the home position and reaching a defined goal position (in the joint space) multiple times*

Starting position	Test	Movement [m]	Error [mm]
(0.019, 0.556, 0.993)	Cartesian z-axis movement	(0, 0, 0.10)	9
		(0, 0, -0.10)	0
		(0, 0, -0.05)	8
		(0, 0, 0.15)	9
		(0, 0, 0.20)	8
		(0, 0, -0.50)	3
(0.132, 0.542, 1.241)	Cartesian x-axis movement	(0.10, 0, 0)	5
		(-0.10, 0, 0)	14
		(-0.50, 0, 0)	10
		(0.15, 0, 0)	6
		(0.05, 0, 0)	5

Table 7.5: *Positional accuracy measured along predefined movement directions*

8

Conclusions

In this thesis, we explored the development of a robot for automatic kiwi harvesting, starting from scratch. The goal was ambitious: to create an autonomous system capable of operating in a complex agricultural environment like a kiwi orchard. The research began with defining the task requirements and proceeded to the design of the robot's layout and the sizing of the actuators. The control system was developed by dividing it between a microcontroller for motion control (low-level) and a PC with ROS for trajectory generation. We also addressed the perception component, illustrating the processing pipelines for the cameras dedicated to fruit detection and tracking. Finally, we presented the results obtained both in simulation and with the real prototype, which was tested in a kiwi orchard.

Despite the limited time available for field testing, the tests allowed us to highlight the weaknesses of the system and outline a path for future improvements. Currently, the data collected during the harvesting experiment with the real robot is insufficient to draw definitive conclusions about the system's effectiveness. However, several field trials revealed significant critical issues: the robot's mechanical construction requires substantial improvements, particularly regarding the joints, which showed non-linear friction, and the dynamic parameters, which did not perfectly match the CAD design. Additionally, sensor-related problems were identified, particularly the interference of sunlight with the Realsense infrared cameras and the inefficacy of the auto-exposure algorithms of the cameras used, which proved inadequate for a complex outdoor environment like a kiwi orchard.

Another aspect that deserves attention is the implementation of online correction for the Local Planner, which was tested only in simulation, as the real robot's movement was too imprecise to properly follow trajectory changes in real time. This represents a crucial point on which future work will focus, with the aim of making the robot's behaviour in the field smoother and more efficient.

Despite the challenges encountered and the areas needing improvement, the project has laid the foundation for a future, more advanced version of the robot, which can be further tested in the field once the identified issues are resolved. In the meantime, development in simulation can continue, contributing to reducing risks and improving

the performance of the next version of the prototype.

Finally, I would like to thank the FieldRobotics team for their support and contribution to the success of this project. Without their commitment and collaboration, the development of this robot would not have been possible.

This page intentionally left blank.

Bibliography

Allevato, Adam (2022). *vision_msgs*. URL: https://github.com/ros-perception/vision_msgs/tree/ros2.

Au, ChiKit et al. (2020). "Workspace analysis of Cartesian robot system for kiwifruit harvesting". English. In: *The Industrial Robot* 47.4. Copyright - © Emerald Publishing Limited 2020; Ultimo aggiornamento - 2023-11-25, pp. 503–510. URL: https://www.proquest.com/scholarly-journals/workspace-analysis-cartesian-robot-system/docview/2533911818/se-2.

Bac, C. W. et al. (2017). "Harvesting Robots for High-value Crops: State-of-the-art Review and Challenges Ahead". In: *Journal of Field Robotics* 34.6, pp. 1199–1220.

Basler (2022). *ROS2-Driver for Basler Cameras*. URL: https://github.com/basler/pylon-ros-camera.

Bechar, Avital and Claude Vigneault (2016). "Agricultural robots for field operations: Concepts and components". In: *Biosystems Engineering* 149, pp. 94–111.

Bewley, Alex et al. (Sept. 2016). "Simple online and realtime tracking". In: pp. 3464–3468. DOI: 10.1109/ICIP.2016.7533003.

Blackmore, Simon (2000). "Robotics and automation in agriculture". In: *IFAC Proceedings Volumes*. Vol. 33. 26. Elsevier, pp. 13–17.

Bradski, G. (2000). "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools*.

Coleman, David et al. (May 2014). "Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study". In: *Journal of Sofware Engineering for Robotics* 5.1. http://moveit.ros.org, pp. 3–16.

Corke, Peter I. (2017). *Robotics, Vision & Control: Fundamental Algorithms in MATLAB*. Second. ISBN 978-3-319-54413-7. Springer.

Gebbers, Robin and Viacheslav I Adamchuk (2010). "Precision agriculture and food security". In: *Science* 327.5967, pp. 828–831.

Giles, R (1998). "Agricultural robotics: the state of the art and future perspectives". In: *Computers and Electronics in Agriculture* 18.2-3, pp. 71–84.

Ioan Sucan, Sachin Chitta (2023). *moveit_msgs*. URL: https://github.com/moveit/moveit_msgs/tree/ros2.

Kootstra, G. et al. (2020). "Selective Harvesting of Fruits: Recent Advances in Robotic Harvesting Systems and Challenges for Future Developments". In: *Biosystems Engineering* 193, pp. 39–54.

Kuffner, J.J. and S.M. LaValle (2000). "RRT-connect: An efficient approach to single-query path planning". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 2, 995–1001 vol.2. DOI: 10.1109/ROBOT.2000.844730.

Lehnert, Christopher et al. (2017). "Autonomous sweet pepper harvesting for protected cropping systems". In: *IEEE Robotics and Automation Letters* 2.2, pp. 872–879.

Macenski, Steven et al. (2022). "Robot Operating System 2: Design, architecture, and uses in the wild". In: *Science Robotics* 7.66, eabm6074. DOI: 10.1126/scirobotics.abm6074. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

Marchi, Julian de (n.d.). "MODELING OF DYNAMIC FRICTION, IMPACT BACKLASH AND ELASTIC COMPLIANCE NONLINEARITIES IN MACHINE TOOLS, WITH APPLICATIONS TO ASYMMETRIC VISCOUS AND KINETIC FRICTION IDENTIFICATION". In: ().

Mcaneney, John, Joshua Judd, and Mike Trought (May 1984). "Wind damage to kiwifruit (Actinidia chinensis Planch.) in relation to windbreak performance". In: *New Zealand Journal of Agricultural Research - N Z J AGR RES* 27, pp. 255–263. DOI: 10.1080/00288233.1984.10430427.

Mengoli, D. et al. (2023). "An online fruit counting application in apple orchards". In: Leiden, The Netherlands: Wageningen Academic, pp. 459–465. ISBN: 9789086869473. DOI: 10.3920/978-90-8686-947-3_57. URL: https://brill.com/view/book/9789086869473/BP000058.xml.

Mengoli, Dario, Roberto Tazzari, and Lorenzo Marconi (Nov. 2020). "Autonomous Robotic Platform for Precision Orchard Management: Architecture and Software Perspective". In: pp. 303–308. DOI: 10.1109/MetroAgriFor50201.2020.9277555.

Pedersen, Søren Marcus et al. (2006). "Agricultural robots—system analysis and economic feasibility". In: *Precision Agriculture* 7.4, pp. 295–308.

PeK Automotive (n.d.). *Slope Helper Robot*. https://slopehelper.com/it/home-it/.

Redmon, Joseph et al. (June 2015). "You Only Look Once: Unified, Real-Time Object Detection". In: DOI: 10.48550/arXiv.1506.02640.

Santos, T. T. et al. (2021). "Sensor Fusion for Agricultural Robotics: An Overview and a New Approach". In: *Sensors* 21.10, p. 3366.

Scarfe, Alistair John (2012). "Development of an Autonomous Kiwifruit Harvester". PhD thesis. Massey University, Manawatu.

Shah, Mili (Aug. 2013). "Solving the Robot-World/Hand-Eye Calibration Problem Using the Kronecker Product". In: *Journal of Mechanisms and Robotics* 5, p. 031007. DOI: 10.1115/1.4024473.

Shamshiri, R. R. et al. (2018). "Research and Development in Agricultural Robotics: A Perspective of Digital Farming". In: *International Journal of Agricultural and Biological Engineering* 11.4, pp. 1–14.

Siciliano, Bruno et al. (Jan. 2009). "Robotics: Modelling, planning and control". In: pp. 1–623.

Şucan, Ioan A., Mark Moll, and Lydia E. Kavraki (Dec. 2012). "The Open Motion Planning Library". In: *IEEE Robotics & Automation Magazine* 19.4. https://ompl.kavrakilab.org, pp. 72–82. DOI: 10.1109/MRA.2012.2205651.

Tevel Aerobotics (n.d.). *Tevel Flying Robot*. https://www.tevel-tech.com/press-release-tevel-partners-with-unifrutti-expands-to-south-america/.

Tkachenko, Maxim et al. (2020-2022). *Label Studio: Data labeling software*. Open source software available from https://github.com/heartexlabs/label-studio. URL: https://github.com/heartexlabs/label-studio.

Tsai, R.Y. and R.K. Lenz (1989). "A new technique for fully autonomous and efficient 3D robotics hand/eye calibration". In: *IEEE Transactions on Robotics and Automation* 5.3, pp. 345–358. DOI: 10.1109/70.34770.

Unity Technologies (2020). *Unity Perception Package*. https://github.com/Unity-Technologies/com.unity.perception.

Unity-Technologies (2022). *ROS TCP Endpoint*. URL: https://github.com/Unity-Technologies/ROS-TCP-Endpoint.

Van Henten, E. J. et al. (2019). "The Evolution of Robotic Harvesters: From Vision to Field Deployment". In: *Robotics and Autonomous Systems* 114, pp. 1–16.

Van Henten, EJ et al. (2002). "An autonomous robot for harvesting cucumbers in greenhouses". In: *Autonomous Robots* 13.3, pp. 241–258.

Williams, Henry et al. (May 2019). "Robotic kiwifruit harvesting using machine vision, convolutional neural networks, and robotic arms". In: *Biosystems Engineering* 181, pp. 140–156. DOI: 10.1016/j.biosystemseng.2019.03.007.

Wojke, Nicolai, Alex Bewley, and Dietrich Paulus (2017). *Simple Online and Realtime Tracking with a Deep Association Metric*. arXiv: 1703.07402 [cs.CV]. URL: https://arxiv.org/abs/1703.07402.

Zhang, Chenghai and John M Kovacs (2012). "The application of small unmanned aerial systems for precision agriculture: a review". In: *Precision Agriculture* 13.6, pp. 693–712.

Zhang, Z. (2000). "A flexible new technique for camera calibration". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11, pp. 1330–1334. DOI: 10.1109/34.888718.

This page intentionally left blank.

Appendices

This page intentionally left blank.

A

Appendix A

Computation of the $\mathbf{B}(\mathbf{q})$ and $\mathbf{g}(\mathbf{q})$ terms of the dynamic model of the robot. All the computation is made using the Matlab Symbolic Toolbox.

A.1 Computation of the gravitational term

Let's start from the position of the centre of gravity of the link 1, link 2, link 3 and the wrist motor with respect to the base link, because these are the only elements that contribute to the gravitational term:

$$\mathbf{p_{l_1}} = \begin{pmatrix} 0 \\ -c_1 \sin(q_1) \\ c_1 \cos(q_1) \end{pmatrix} \tag{A.1}$$

$$\mathbf{p_{l_2}} = \begin{pmatrix} 0 \\ -l_1 \sin(q_1) - a_2 \cos(q_1 - q_2) - c_2 \sin(q_1 - q_2) \\ l_1 \cos(q_1) + c_2 \cos(q_1 - q_2) - a_2 \sin(q_1 - q_2) \end{pmatrix} \tag{A.2}$$

$$\mathbf{p_{l_3}} = \begin{pmatrix} 0 \\ a_3 \cos(q_1 - q_2 + q_3) - c_3 \sin(q_1 - q_2 + q_3) - l_1 \sin(q_1) - l_2 \sin(q_1 - q_2) \\ c_3 \cos(q_1 - q_2 + q_3) - a_3 \sin(q_1 - q_2 + q_3) + l_1 \cos(q_1) + l_2 \cos(q_1 - q_2) \end{pmatrix} \tag{A.3}$$

$$\mathbf{p_m} = \begin{pmatrix} 0 \\ -l_1 \sin(q_1) - l_2 \sin(q_1 - q_2) \\ l_1 \cos(q_1) + l_2 \cos(q_1 - q_2) \end{pmatrix} \tag{A.4}$$

where p_{l1}, p_{l2}, p_{l3} and p_m are the positions of the centre of gravity of link 1, link 2, link 3 and the wrist motor; c_1, c_2, c_3 are the component of distances of the centre of gravity of a link with respect to its extremity projected on its greatest principal axis of inertia; l_1, l_2, l_3 are the link lengths; a_2 and a_3 are the component of the distances of the centre of gravity of a link with respect to its extremity projected on the other principal axis of inertia.

The corresponding positional Jacobians (taken wrt to q_1, q_2 and q_3 only, because q_0 is not involved in the gravitational term) are the following:

$$\mathcal{J}_P^{(\ell_1)} = \begin{pmatrix} 0 & 0 & 0 \\ -c_1 \cos(q_1) & 0 & 0 \\ -c_1 \sin(q_1) & 0 & 0 \end{pmatrix}$$

$$\mathcal{J}_P^{(\ell_2)} = \begin{pmatrix} 0 & 0 & 0 \\ \sigma_4 - \sigma_1 - l_1 \cos(q_1) & \sigma_1 - \sigma_4 & 0 \\ -l_1 \sin(q_1) - \sigma_2 - \sigma_3 & \sigma_2 + \sigma_3 & 0 \end{pmatrix}$$

where

$$\begin{aligned} \sigma_1 &= c_2 \cos(q_1 - q_2) \\ \sigma_2 &= a_2 \cos(q_1 - q_2) \\ \sigma_3 &= c_2 \sin(q_1 - q_2) \\ \sigma_4 &= a_2 \sin(q_1 - q_2) \end{aligned}$$

$$\mathcal{J}_P^{(\ell_3)} = \begin{pmatrix} 0 & 0 & 0 \\ -\sigma_3 - \sigma_2 - l_1 \cos(q_1) - l_2 \cos(q_1 - q_2) & \sigma_3 + \sigma_2 + l_2 \cos(q_1 - q_2) & -\sigma_3 - \sigma_2 \\ -\sigma_4 - \sigma_1 - l_1 \sin(q_1) - l_2 \sin(q_1 - q_2) & \sigma_4 + \sigma_1 + l_2 \sin(q_1 - q_2) & -\sigma_4 - \sigma_1 \end{pmatrix}$$

where

$$\begin{aligned} \sigma_1 &= c_3 \sin(q_1 - q_2 + q_3) \\ \sigma_2 &= a_3 \sin(q_1 - q_2 + q_3) \\ \sigma_3 &= c_3 \cos(q_1 - q_2 + q_3) \\ \sigma_4 &= a_3 \cos(q_1 - q_2 + q_3) \end{aligned}$$

$$\mathcal{J}_P^{(m)} = \begin{pmatrix} 0 & 0 & 0 \\ -l_1 \cos(q_1) - l_2 \cos(q_1 - q_2) & l_2 \cos(q_1 - q_2) & 0 \\ -l_1 \sin(q_1) - l_2 \sin(q_1 - q_2) & l_2 \sin(q_1 - q_2) & 0 \end{pmatrix}$$

Considering that the gravity term can be computed as (Siciliano et al., 2009):

$$-\sum_{j=1}^{n} \left(m_{\ell_j} \mathbf{g}_0^T \mathcal{J}_{P_i}^{(\ell_j)}(\mathbf{q}) + m_{m_j} \mathbf{g}_0^T \mathcal{J}_{P_i}^{(m_j)}(\mathbf{q}) \right) = g_i(\mathbf{q})$$

where m_{l_j} is the mass of the link j, $\mathcal{J}_{P_i}^{(\ell_j)}(\mathbf{q})$ is the positional Jacobian of the of the centre of gravity of link j, m_{m_j} is the mass of the motor j, $\mathcal{J}_{P_i}^{(m_j)}(\mathbf{q})$ is the Jacobian of the position of the motor j, and $\mathbf{g}_0 = [0, 0, g]^T$ with g being the gravity acceleration.

Therefore, the complete gravity term is defined as follows:

$$
\begin{aligned}
g_1(q) &= 0 \\
g_2(q) &= m_3\, g\, (\sigma_2 + \sigma_1 + l_1\, \sin(q_1) + \sigma_3) + m_2\, g\, (l_1\, \sin(q_1) + \sigma_4 + \sigma_5) + m_m\, g\, (l_1\, \sin(q_1) + \sigma_3) + \\
&\quad + c_1\, m_1\, g\, \sin(q_1) \\
g_3(q) &= -m_3\, g\, (\sigma_2 + \sigma_1 + \sigma_3) - m_2\, g\, (\sigma_4 + \sigma_5) - l_2\, m_m\, \sin(q_1 - q_2)\, g \\
g_4(q) &= m_3\, g\, (\sigma_2 + \sigma_1)
\end{aligned}
$$

where:

$$
\begin{aligned}
\sigma_1 &= c_3\, \sin(q_1 - q_2 + q_3) \\
\sigma_2 &= a_3\, \cos(q_1 - q_2 + q_3) \\
\sigma_3 &= l_2\, \sin(q_1 - q_2) \\
\sigma_4 &= a_2\, \cos(q_1 - q_2) \\
\sigma_5 &= c_2\, \sin(q_1 - q_2)
\end{aligned}
$$

A.2 Computation of the inertial term

Considering also the rotational Jacobians:

$$
\mathcal{J}_O^{(\ell_1)} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}
$$

$$
\mathcal{J}_O^{(\ell_2)} = \begin{pmatrix} -1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}
$$

$$
\mathcal{J}_O^{(\ell_3)} = \begin{pmatrix} -1 & 1 & -1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}
$$

we can compute the the inertial matrix for the links 1, 2 and 3 as follows (Siciliano et al., 2009), considering that they are coplanar:

$$
\mathbf{B_{123}(q)} = \sum_{i=1}^{3} \left((\mathcal{J}_O^{(\ell_i)})^T \mathbf{I_i} \mathcal{J}_O^{(\ell_i)} + m_i (\mathcal{J}_P^{(m_i)})^T \mathcal{J}_P^{(m_i)} \right) \tag{A.5}
$$

$$
= \mathbf{B_1(q)} + \mathbf{B_2(q)} + \mathbf{B_3(q)} + \mathbf{B_m(q)} \tag{A.6}
$$

being I_i the moment of inertia of the link i.

The inertia of the robot seen by the joint 0 instead, can be computed as follows:

$$
b_{00} = I_1^{zz} + m_1 * (p_{l_1}^y)^2 + I_2^{zz} + m_2 * (p_{l_2}^y)^2 + I_3^{zz} + m_3 * (p_{l_3}^y)^2; \tag{A.7}
$$

Therefore the whole inertial matrix is the following:

$$\mathbf{B}(\mathbf{q}) = \begin{pmatrix} b_{00} & 0 & 0 & 0 \\ 0 & & & \\ 0 & & \mathbf{B_{123}}(\mathbf{q}) & \\ 0 & & & \end{pmatrix}$$

The elements of this matrix are:

$$\begin{aligned}
b_{00} &= I_1^{zz} + I_2^{zz} + I_3^{zz} + m_3 \left(c_3\, \sigma_{34} - a_3\, \sigma_{35} + l_1\, \sin\left(q_1\right) + \sigma_{32}\right)^2 + m_2\, {\sigma_{11}}^2 + {c_1}^2\, m_1 \sin\left(q_1\right)^2 \\
b_{11} &= I_1^{xx} + I_2^{xx} + I_3^{xx} + m_3\, \sigma_{10} \left(\sigma_{13} + \sigma_4 + \sigma_7 + \sigma_6\right) + m_3 \left(\sigma_{12} + \sigma_3 + \sigma_8 + \sigma_5\right) \sigma_9 + \\
&\quad + m_2 \left(\sigma_{15} - \sigma_{18} + \sigma_4\right) \sigma_{14} + m_2 \left(\sigma_{17} + \sigma_{16} + \sigma_3\right) \sigma_{11} + m_1\, \sin\left(q_1\right)\, {\left|c_1\right|}^2\, \sin\left(q_1\right) + \\
&\quad + m_1\, \cos\left(q_1\right)\, {\left|c_1\right|}^2\, \cos\left(q_1\right) + \frac{m_m \left(l_2\, \cos\left(q_1\right)\, {\left|l_1\right|}^2 + l_1\, \sigma_{30}\, {\left|l_2\right|}^2\right) \sigma_{19}}{l_1\, l_2} + \\
&\quad + \frac{m_m \left(l_2\, \sin\left(q_1\right)\, {\left|l_1\right|}^2 + l_1\, \sigma_{31}\, {\left|l_2\right|}^2\right) \sigma_{20}}{l_1\, l_2} \\
b_{12} &= -I_2^{xx} - I_3^{xx} - m_3\, \sigma_{22} \left(\sigma_2 + \sigma_{13} + \sigma_7 + \sigma_6\right) - m_3\, \sigma_{21} \left(\sigma_1 + \sigma_{12} + \sigma_8 + \sigma_5\right) + \\
&\quad - m_2 \left(\sigma_{28} - \sigma_{29}\right) \left(\sigma_2 + \sigma_{15} - \sigma_{18}\right) - m_2 \left(\sigma_{26} + \sigma_{27}\right) \left(\sigma_1 + \sigma_{17} + \sigma_{16}\right) + \\
&\quad - l_2\, m_m\, \cos\left(q_1 - q_2\right)\, \left(\sigma_2 + \sigma_{13}\right) - l_2\, m_m\, \sin\left(q_1 - q_2\right)\, \left(\sigma_1 + \sigma_{12}\right) \\
b_{13} &= I_3^{xx} + m_3\, \sigma_{23} \left(\sigma_2 + \sigma_{30}\, l_2 + \sigma_7 + \sigma_6\right) + m_3\, \sigma_{24} \left(l_2\, \sigma_{31} + \sigma_1 + \sigma_8 + \sigma_5\right) \\
b_{21} &= -I_2^{xx} - I_3^{xx} - m_2 \left(\sigma_{17} + \sigma_{16}\right) \sigma_{11} - m_3 \left(\sigma_{13} + \sigma_7 + \sigma_6\right) \sigma_{10} - m_3 \left(\sigma_{12} + \sigma_8 + \sigma_5\right) \sigma_9 + \\
&\quad - m_2 \left(\sigma_{15} - \sigma_{18}\right) \sigma_{14} - \frac{m_m\, \sigma_{30}\, {\left|l_2\right|}^2\, \sigma_{19}}{l_2} - \frac{m_m\, {\left|l_2\right|}^2\, \sigma_{31}\, \sigma_{20}}{l_2} \\
b_{22} &= I_2^{xx} + I_3^{xx} + m_2 \left(\sigma_{17} + \sigma_{16}\right) \left(\sigma_{26} + \sigma_{27}\right) + m_2 \left(\sigma_{15} - \sigma_{18}\right) \left(\sigma_{28} - \sigma_{29}\right) + \\
&\quad + m_3 \left(\sigma_{13} + \sigma_7 + \sigma_6\right) \sigma_{22} + m_3 \left(\sigma_{12} + \sigma_8 + \sigma_5\right) \sigma_{21} + m_m\, \cos\left(q_1 - q_2\right)\, \sigma_{30}\, {\left|l_2\right|}^2 + \\
&\quad + m_m\, \sin\left(q_1 - q_2\right)\, {\left|l_2\right|}^2\, \sigma_{31} \\
b_{23} &= -I_3^{xx} - m_3\, \sigma_{23} \left(\sigma_{30}\, l_2 + \sigma_7 + \sigma_6\right) - m_3\, \sigma_{24} \left(l_2\, \sigma_{31} + \sigma_8 + \sigma_5\right) \\
b_{31} &= I_3^{xx} + m_3 \left(\sigma_7 + \sigma_6\right) \sigma_{10} + m_3 \left(\sigma_8 + \sigma_5\right) \sigma_9 \\
b_{32} &= -I_3^{xx} - m_3 \left(\sigma_7 + \sigma_6\right) \sigma_{22} - m_3 \left(\sigma_8 + \sigma_5\right) \sigma_{21} \\
b_{33} &= I_3^{xx} + m_3 \left(\sigma_8 + \sigma_5\right) \sigma_{24} + m_3 \left(\sigma_7 + \sigma_6\right) \sigma_{23}
\end{aligned}$$

where

$$
\begin{aligned}
\sigma_1 &= \sin(q_1)\ l_1 \\
\sigma_2 &= \cos(q_1)\ l_1 \\
\sigma_3 &= \frac{\sin(q_1)\ |l_1|^2}{l_1} \\
\sigma_4 &= \frac{\cos(q_1)\ |l_1|^2}{l_1} \\
\sigma_5 &= \frac{\sin(\sigma_{25})\ |c_3|^2}{c_3} \\
\sigma_6 &= \frac{\sin(\sigma_{25})\ |a_3|^2}{a_3} \\
\sigma_7 &= \frac{\cos(\sigma_{25})\ |c_3|^2}{c_3} \\
\sigma_8 &= \frac{\cos(\sigma_{25})\ |a_3|^2}{a_3} \\
\sigma_9 &= a_3\,\sigma_{35} + c_3\,\sigma_{34} + l_1\ \sin(q_1) + \sigma_{32} \\
\sigma_{10} &= c_3\,\sigma_{35} + a_3\,\sigma_{34} + l_1\ \cos(q_1) + \sigma_{33} \\
\sigma_{11} &= l_1\ \sin(q_1) + \sigma_{26} + \sigma_{27} \\
\sigma_{12} &= \frac{|l_2|^2\,\sigma_{31}}{l_2} \\
\sigma_{13} &= \frac{\sigma_{30}\,|l_2|^2}{l_2} \\
\sigma_{14} &= l_1\ \cos(q_1) + \sigma_{28} - \sigma_{29} \\
\sigma_{15} &= \frac{|c_2|^2\,\sigma_{30}}{c_2} \\
\sigma_{16} &= \frac{|c_2|^2\,\sigma_{31}}{c_2} \\
\sigma_{17} &= \frac{|a_2|^2\,\sigma_{30}}{a_2} \\
\sigma_{18} &= \frac{|a_2|^2\,\sigma_{31}}{a_2} \\
\sigma_{19} &= l_1\ \cos(q_1) + \sigma_{33} \\
\sigma_{20} &= l_1\ \sin(q_1) + \sigma_{32} \\
\sigma_{21} &= a_3\,\sigma_{35} + c_3\,\sigma_{34} + \sigma_{32} \\
\sigma_{22} &= c_3\,\sigma_{35} + a_3\,\sigma_{34} + \sigma_{33} \\
\sigma_{23} &= c_3\,\sigma_{35} + a_3\,\sigma_{34} \\
\sigma_{24} &= a_3\,\sigma_{35} + c_3\,\sigma_{34} \\
\sigma_{25} &= q_1 - q_2 + q_3 \\
\sigma_{26} &= a_2\ \cos(q_1 - q_2)
\end{aligned}
$$

$$\sigma_{27} = c_2 \sin(q_1 - q_2)$$
$$\sigma_{28} = c_2 \cos(q_1 - q_2)$$
$$\sigma_{29} = a_2 \sin(q_1 - q_2)$$
$$\sigma_{30} = \cos(q_1 - q_2)$$
$$\sigma_{31} = \sin(q_1 - q_2)$$
$$\sigma_{32} = l_2 \sin(q_1 - q_2)$$
$$\sigma_{33} = l_2 \cos(q_1 - q_2)$$
$$\sigma_{34} = \sin(q_1 - q_2 + q_3)$$
$$\sigma_{35} = \cos(q_1 - q_2 + q_3)$$

B

Appendix B

URDF (Universal Robot Description Format) of the robot:

```
<?xml version="1.0"?>
<robot name="custom_robot" xmlns:xacro="http://www.ros.org/wiki/xacro">

  <xacro:property name="base_l" value="0.120" />
  <xacro:property name="base_w" value="0.120" />
  <xacro:property name="base_t" value="0.100" />

  <xacro:property name="rgb_to_depth" value="0.029" />
  <xacro:property name="camera_x" value="-0.02572462" />
  <xacro:property name="camera_y" value="0.01588668" />
  <xacro:property name="camera_z" value="0.30584433" />
  <xacro:property name="camera_slope_x" value="-0.48344732968236354" />
  <xacro:property name="camera_slope_y" value="0.017851267248094986" />
  <xacro:property name="camera_slope_z" value="-0.0022621326415571035" />

  <xacro:property name="arm_l" value="0.547" />
  <xacro:property name="arm_r" value="0.025" />

  <xacro:property name="forearm_l" value="0.733" />
  <xacro:property name="forearm_r" value="0.020" />

  <xacro:property name="hand_l" value="0.155" />
  <xacro:property name="hand_r" value="0.015" />

  <xacro:property name="hand_to_ee_x" value="0.0925" />
  <xacro:property name="hand_to_ee_z" value="0.358" />

  <xacro:property name="ee_camera_x" value="0.15761462" />
  <xacro:property name="ee_camera_y" value="0.00137721" />
  <xacro:property name="ee_camera_z" value="0.19915025" />
  <xacro:property name="ee_camera_r" value="0.008289796579141986" />
  <xacro:property name="ee_camera_p" value="-0.285080125737788" />
  <xacro:property name="ee_camera_y" value="-0.005803166768052965" />


```

```
36   <link name="base_link">
37     <visual>
38       <geometry>
39         <box size="${base_l} ${base_w} ${base_t}" />
40       </geometry>
41       <origin rpy="0 0 0" xyz="0 0 0"/>
42     </visual>
43     <collision>
44       <geometry>
45         <box size="${base_l} ${base_w} ${base_t}" />
46       </geometry>
47       <origin rpy="0 0 0" xyz="0 0 0" />
48     </collision>
49   </link>
50
51   <link name="camera_link"/>
52
53   <link name="base"/>
54
55   <joint name="joint_cam" type="fixed">
56     <origin rpy="${camera_slope_x} ${camera_slope_y} ${camera_slope_z}"
       ↪  xyz="${camera_x - rgb_to_depth} ${camera_y} ${camera_z}"/>
57     <parent link="base_link"/>
58     <child link="camera_link"/>
59   </joint>
60
61   <joint name="shoulder1" type="revolute">
62     <axis xyz="0 0 -1"/>
63     <limit effort="1000.0" lower="${-pi}" upper="${pi}" velocity="0.5"/>
64     <origin rpy="0 0 0" xyz="0 0 ${base_t/2}"/>
65     <parent link="base_link"/>
66     <child link="base"/>
67   </joint>
68
69   <link name="arm">
70     <visual>
71       <geometry>
72         <cylinder length="${arm_l}" radius="${arm_r}"/>
73       </geometry>
74       <origin rpy="0 0 0" xyz="0 0 ${arm_l/2}"/>
75     </visual>
76     <collision>
77       <geometry>
78         <cylinder length="${arm_l}" radius="${arm_r}" />
79       </geometry>
80       <origin rpy="0 0 0" xyz="0 0 ${arm_l/2}" />
81     </collision>
82     <inertial>
83       <mass value="1.3530714"/>
84       <inertia ixx="6.5081120e-2" ixy="0.0" ixz="0.0" iyy="6.7476140e-4"
         ↪  iyz="8.0337007e-4" izz="6.5145616e-2"/>
```

```
85      </inertial>
86    </link>
87
88    <joint name="shoulder2" type="revolute">
89      <axis xyz="-1 0 0"/>
90      <limit effort="1000.0" lower="${-pi}" upper="${pi}" velocity="0.5"/>
91      <origin rpy="0 0 0" xyz="0 0 0"/>
92      <parent link="base"/>
93      <child link="arm"/>
94    </joint>
95
96    <link name="forearm">
97      <visual>
98        <geometry>
99          <cylinder length="${forearm_l}" radius="${forearm_r}"/>
100       </geometry>
101       <origin rpy="0 0 0" xyz="0 0 ${forearm_l/2}"/>
102     </visual>
103     <collision>
104       <geometry>
105         <cylinder length="${forearm_l}" radius="${forearm_r}" />
106       </geometry>
107       <origin rpy="0 0 0" xyz="0 0 ${forearm_l/2}" />
108     </collision>
109     <inertial>
110       <mass value="1.711416"/>
111       <inertia ixx="1.5790694e-1" ixy="0.0" ixz="1.9228267e-3" iyy="1.5766998e-1"
        ↪  iyz="-1.5672527e-5" izz="7.4154764e-4"/>
112     </inertial>
113   </link>
114
115   <joint name="elbow" type="revolute">
116     <axis xyz="1 0 0"/>
117     <limit effort="1000.0" lower="${-pi}" upper="${pi}" velocity="0.5"/>
118     <origin rpy="0 0 0" xyz="0 0 ${arm_l}"/>
119     <parent link="arm"/>
120     <child link="forearm"/>
121   </joint>
122
123   <link name="hand">
124     <visual>
125       <geometry>
126         <cylinder length="${hand_l}" radius="${hand_r}"/>
127       </geometry>
128       <origin rpy="0 0 0" xyz="0 0 ${hand_l/2}"/>
129     </visual>
130     <collision>
131       <geometry>
132         <cylinder length="${hand_l}" radius="${hand_r}" />
133       </geometry>
134       <origin rpy="0 0 0" xyz="0 0 ${hand_l/2}" />
```

```
135     </collision>
136     <inertial>
137       <mass value="1.8"/>
138       <inertia ixx="3.0387029e-2" ixy="-2.7271686e-3" ixz="1.9228267e-3"
          ↪  iyy="1.2955984e-2" iyz="5.4334105e-4" izz="2.6339242e-2"/>
139     </inertial>
140   </link>
141
142   <joint name="wrist" type="revolute">
143     <axis xyz="-1 0 0"/>
144     <limit effort="1000.0" lower="${-pi}" upper="${pi}" velocity="0.5"/>
145     <origin rpy="0 0 0" xyz="0 0 ${forearm_l}"/>
146     <parent link="forearm"/>
147     <child link="hand"/>
148   </joint>
149
150   <link name="end_effector">
151     <visual>
152       <geometry>
153         <mesh filename="package://manipulator_description/meshes/custom_robot/assieme_⌋
          ↪  prototipo.stl" scale="0.001 0.001
          ↪  0.001"/>
154       </geometry>
155       <origin rpy="0 ${pi} ${pi/2}" xyz="-0.04 0.05 -0.18"/>
156     </visual>
157     <collision>
158       <geometry>
159         <box size="0.05 0.05 0.150" />
160       </geometry>
161       <origin rpy="0 ${pi} ${pi/2}" xyz="0 0 -0.075" />
162     </collision>
163   </link>
164
165   <joint name="joint1" type="fixed">
166     <origin rpy="0 0 0" xyz="${hand_to_ee_x} 0 ${hand_to_ee_z}"/>
167     <parent link="hand"/>
168     <child link="end_effector"/>
169   </joint>
170
171   <link name="camera_ee_link"/>
172
173   <joint name="ee_camera_joint" type="fixed">
174     <origin rpy="${ee_camera_r} ${ee_camera_p} ${ee_camera_y}" xyz="${ee_camera_x}
      ↪  ${ee_camera_y} ${ee_camera_z}"/>
175     <parent link="hand"/>
176     <child link="camera_ee_link"/>
177   </joint>
178
179   <link name="chess_link"/>
180
181   <joint name="chess_link_joint" type="fixed">
```

```
182     <origin rpy="0 ${pi/2-0.3665} ${-pi/2}" xyz="-0.166 0 0.064"/>
183     <parent link="hand"/>
184     <child link="chess_link"/>
185   </joint>
186
187 </robot>
```

Listing B.1: *URDF file of the robot, scripted with Xacro*