POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

An improved Computer Vision based approach for automatic fruit picking with robots

Tesi di Laurea Magistrale in
Mechanical Engineering - Ingegneria Meccanica

Author: **Samuele Mara**

Student ID: 224527
Advisor: Prof. Francesco Braghin
Co-advisors: Prof. Loris Roveda
Academic Year: 2023-24

Abstract

The global agricultural sector is grappling with significant challenges, particularly acute labor shortages exacerbated by political factors, demographic shifts, and urbanization. An aging workforce and the declining interest of younger generations in agricultural labor further intensify the issue, making alternative harvesting solutions critical.

Robotic fruit picking has emerged as a promising response, driven by advancements in key enabling technologies, such as computer vision and AI. These robots can operate continuously and increase productivity, addressing labor shortages. However, a primary technical challenge remains: the complexity of fruit identification and picking in dynamic environments. Fruits are often obscured by leaves, branches, and other obstacles, creating occlusions that complicate detection and classification. Unlike human pickers, robots must use advanced computer vision techniques to navigate and perform effectively.

In this thesis work, innovative solutions are proposed to address the challenges of occlusions in robotic fruit harvesting, leveraging state-of-the-art techniques that allow the merging of 2D and 3D perception. A 3D Gaussian Splatting-based pipeline reconstructs the environment, extracting the point clouds of the plants and the fruits. Additionally, an improved custom You Only Look Once (YOLO)v11 network is tailored and trained to detect both visible and occluded fruits. These methods are tested in both a simulated ROS environment and a custom experimental benchmark, picking the detected fruits using collision-free motion planning algorithms for an Emika Franka Panda Robot.

Promising results are achieved, improving the methods based on state-of-the-art techniques in the majority of the metrics measuring the detection accuracy of occluded fruits. The continuous advancements in AI and computer vision research are essential drivers for future developments and further improvements, bridging the gap between robotic and human perception of complex picking environments.

Keywords: Automatic Fruit Picking, Computer Vision, 3D reconstruction, AI, YOLOv11, 3DGS, Franka Panda Robot, ROS Simulation, Experimental Benchmark

Abstract in lingua italiana

Il settore agricolo globale sta affrontando sfide significative a causa della carenza di manodopera, dell'invecchiamento della forza lavoro e del disinteresse dei giovani per il lavoro agricolo, cambiamenti aggravati da fattori politici, mutamenti demografici e urbanizzazione.

La raccolta robotizzata dei frutti si è affermata come una soluzione promettente, grazie ai recenti progressi in tecnologie chiave come la Computer Vision e l'IA. I robot possono operare senza sosta, aumentando la produttività e contrastando la carenza di manodopera. Tuttavia, una delle principali sfide tecniche è rappresentata dalla complessità nell'identificare e raccogliere i frutti: essi sono infatti spesso nascosti da foglie, rami e altri ostacoli, rendendo difficile la loro rilevazione e classificazione. A differenza dei raccoglitori umani, i robot devono impiegare tecniche avanzate di Computer Vision per navigare e operare in modo efficiente.

In questo lavoro di tesi, vengono proposte soluzioni innovative per affrontare le sfide delle occlusioni nella raccolta automatica dei frutti, sfruttando tecniche all'avanguardia che permettono di integrare la percezione 2D e 3D. Una pipeline basata sul 3D Gaussian Splatting ricostruisce l'ambiente, estraendo le nuvole di punti delle piante e dei frutti. Inoltre, una rete nuerale migliorata You Only Look Once (YOLO)v11 è apposiamente modificata e addestrata per rilevare sia i frutti visibili che quelli occlusi. Questi metodi sono testati sia in simulazione che in un banco-prova sperimentale , raccogliendo inoltre i frutti rilevati utilizzando un robot Emika Franka Panda e algoritmi di pianificazione cinematica avanzati in grado di evitare collisioni.

Sono stati ottenuti risultati promettenti, superando le tecniche relative allo stato dell'arte nella maggior parte delle metriche riguardanti l'accuratezza del riconoscimento dei frutti occlusi. I continui progressi nella ricerca sull'IA e sulla Computer Vision sono fattori essenziali per i futuri sviluppi e ulteriori miglioramenti, colmando il divario tra la percezione robotica e quella umana in ambienti di raccolta complessi.

Parole chiave: Raccolta automatica dei frutti, Computer Vision, Ricostruzione 3D, IA, YOLOv11, 3DGS, Robot Franka Panda, Simulazione ROS; Banco-prova sperimentale

Contents

Acronyms

3DGS 3D Gaussian Splattings. 35, 50

AP Average Precision. 36

CNNs Convolutional Neural Networks. v, 16–18, 117

DL Deep Learning. v, 16, 17, 117

DOFs Degrees Of Freedom. 9, 26

IoU Intersection over Union. 36

mAP Mean Average Precision. 36

MPC Model Predictive Control. 21, 22, 117

P Precision. 37, 38

PRM Probabilistic Road Map. 25

PSO Particle Swarm Optimization. 21

R Recall. 37, 38

R-CNNs Region-based Convolutional Neural Networks. 17, 19

ROS Robot Operating System. 53

RRT Rapidly exploring Random Tree. 21, 25

SAM Segment Anything Model. 35

YOLO You Only Look Once. 6, 16, 17, 19–21, 35, 36, 38, 39, 42, 118, 121

Introduction

Research on fruit-picking robots dates back to the 1980s, but commercialization has only recently begun to gain momentum, driven by a pressing market demand and advancements in key enabling technologies. This shift comes at a time when the global agricultural sector is facing mounting challenges, particularly an acute labor shortage [44].

The primary market driver behind the growing interest in robotic fruit-picking is labor shortages, which have been widely reported across major fruit-growing regions such as the USA, Australia, New Zealand, the UK and parts of continental Europe. These shortages are driven by both political factors (e.g., Brexit and U.S.-Mexico border policies) and broader demographic shifts.

Urbanization, an aging agricultural workforce and the declining willingness of younger generations to engage in physically demanding, low-wage farm labor have significantly reduced the available workforce. In many regions, younger workers are increasingly migrating to urban areas in search of less strenuous and more financially rewarding opportunities. Stricter immigration policies have further exacerbated these shortages, particularly in countries that have historically relied on migrant labor for seasonal harvesting. Given these challenges, the need for alternative harvesting solutions has become critical, with robotic systems emerging as a promising and increasingly necessary response.

Other crucial issues involve the diminishing interest of the workforce in agricultural occupations [14], as well as recent international travel restrictions, such as those imposed during the COVID-19 pandemic [30], which have severely limited the availability of migrant workers. These challenges have led to significant crop losses, with tons of fresh produce left unharvested and rotting in fields [54].

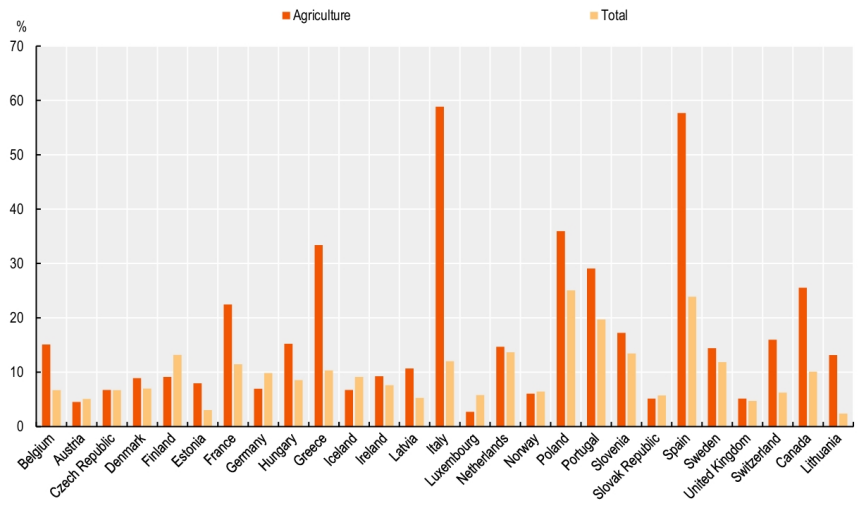


Figure 1: Temporary employment in agriculture compared to overall temporary employment, people aged 25-54, 2011-12, source:[44]

The socioeconomic implications are profound, as farmers struggle with rising labor costs, reduced productivity and financial instability. The impact of these shortages is significant. For example, in 2017, Santa Barbara County alone saw $13 million worth of strawberries, broccoli and other produce left to rot due to a lack of available workers [6].
Additionally, the Natural Resources Defense Council estimates that 20% of the produce grown in the U.S. never leaves the farm due to labor shortages or the high cost of manual harvesting. This situation presents both ethical and economic concerns, underscoring the urgent need for viable alternatives.
Beyond labor shortages, conventional fruit-picking practices tend to have a strong environmental impact. Intensive agricultural practices often contribute to soil degradation, water overuse and biodiversity loss. Additionally, manual harvesting can lead to inefficiencies, such as uneven picking and post-harvest losses, which further strain resources and reduce overall productivity.
The challenges posed by manual harvesting are further compounded by the effects of climate change, which include unpredictable weather patterns and extreme events, thus disrupting traditional growing seasons and exacerbating the aforementioned challenges.

Robotic fruit picking has emerged as a transformative solution to address labor shortages and the rising costs associated with manual harvesting. By automating the process, robots can operate continuously, increasing productivity while alleviating the time-sensitive pressures of seasonal work. Equipped with advanced sensors and data analysis capabilities, these systems can assess fruit ripeness, detect positioning and account for environmental

factors such as branch distribution, leaf occlusion and disease presence, leading to optimized resource use and improved decision-making.
The integration of artificial intelligence and machine learning further enhances the capabilities of robotic harvesters, allowing them to identify, locate and pick fruits with high precision [10]. These automated systems not only mitigate labor shortages but also ensure consistency in fruit selection while minimizing post-harvest losses. However, despite their potential benefits, robotic fruit harvesting still faces technical and economic challenges that must be addressed for widespread adoption in commercial agriculture.

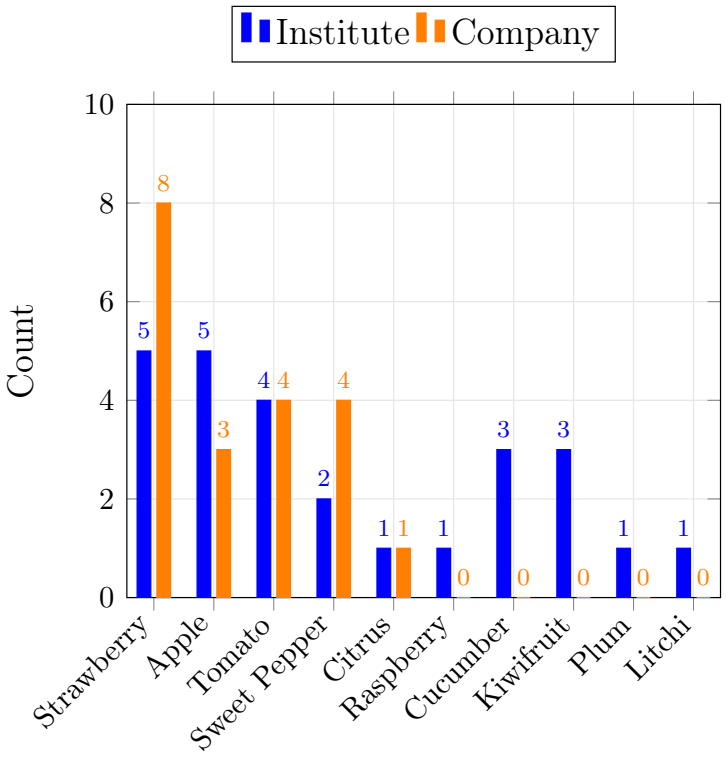


Figure 2: Comparison of Institute and Company research on fruit detection, source:[65]

One of the primary challenges in robotic fruit harvesting is the complexity of fruit identification and picking in dynamic environments. Fruits are often obscured by leaves, branches and other natural obstacles, creating occlusions that make it difficult for robotic vision systems to detect and classify them accurately. Unlike human pickers who rely on their experience and adaptive abilities to navigate occluded environments, robots must employ advanced computer vision techniques to achieve comparable performance. The development of robust perception systems capable of identifying and localizing fruit in real-time is crucial for the success of automated harvesting solutions.

In this thesis work, innovative solutions are proposed to address the challenges of occlusions in robotic fruit harvesting, focusing on both 3D reconstruction and occlusion detection. The proposed approach includes the use of state-of-the-art 3D environment reconstruction techniques that enable robots to generate a detailed spatial representation of the plants, allowing them to use a precise 3D representation of the picking space. Alongside point cloud processing, depth cameras are utilized to construct comparative models of fruit-bearing plants, improving picking strategies.
Furthermore, an innovative approach to the You Only Look Once (YOLO) object detection algorithm is introduced to improve occlusion detection, incorporating contextual information and increasing detection accuracy for severely occluded fruits.
The subsequent chapters of this thesis will explore these proposed solutions in detail:

- **Chapter 1**: Literature Review, will provide a comprehensive overview of the current state-of-the-art in robotic fruit harvesting, discussing existing methodologies and their limitations, including complete systems, object detection and localization, trajectory planning and 3D environment reconstruction;
- **Chapter 2**: Methodologies, will outline the proposed approach for improving occlusion detection, through both YOLOv11 detection and 3D environment reconstruction;
- **Chapter 3**: Simulation, will present the simulated picking environment, from the simulation of the hardware to the implementation of the proposed methods;
- **Chapter 4**: Experimental Tests, will describe the creation of the experimental picking environment, the composition of the used robotic system and the real-world implementation of the proposed methods;
- **Chapter 5**: Results, will present the outcomes of the 3D environment reconstruction, YOLOv11 detection and picking tasks;
- **Chapter 6**: Conclusions and Future Works, concludes the Thesis works with a discussion on the future research directions and potential applications in precision agriculture;

1 | Literature Review

Automatic fruit harvesting technology has been actively developed over the past three decades [3, 17], achieving significant advancements in both integrated systems and their constituent subsystems—such as vision, grippers and control mechanisms [5]. However, despite these technological improvements, the widespread adoption of harvesting robots in orchards remains limited. One key reason is the indispensable role of human labor in the harvesting process, which necessitates the integration of human-centric design principles into robotic systems. Moreover, the deployment of robotic solutions introduces social, legal and ethical considerations that warrant further exploration and assessment.
To understand the complexities of automatic fruit picking, it is essential to first analyze the various types of harvesting operations that are feasible. These operations can be categorized according to several criteria, including:

- **Type of picking operation**: Based on the type of sensing and harvesting hardware employed, automatic fruit picking operations can be classified into two subgroups, as highlighted by [62] and illustrated in Figure 1.1;
- **Harvested fruit variety and detaching method**: Considering the anatomical characteristics of the target fruits and the associated phenomenological requirements, both the sensing and picking solutions exhibit considerable variability. The principal categories of harvested fruits reported in the literature [65] are depicted in Figure 1.2, where also the main detaching methods are listed;
- **Type of picking environment**: Depending on whether the picking environment is structured (e.g., greenhouses, laboratory settings) or unstructured (e.g., orchards, cultivation fields), different operational strategies are employed.

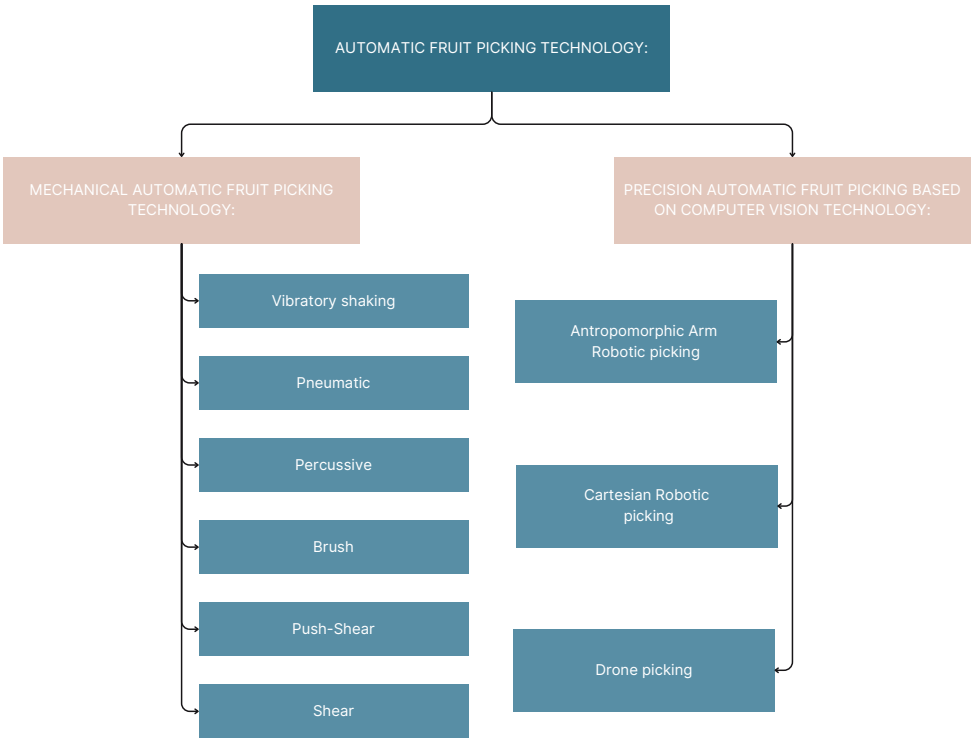
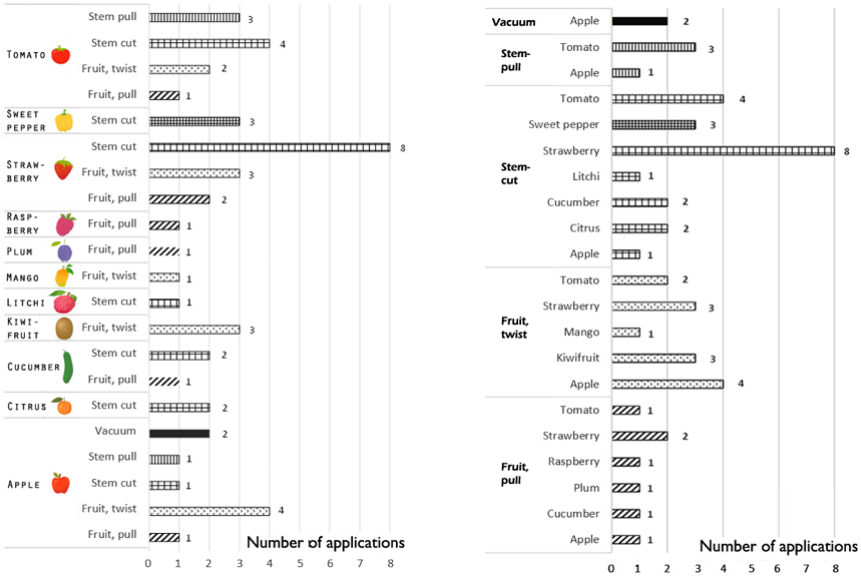


Figure 1.1: Automatic fruit picking technology classification



(a) Fruit variety. (b) Detachment method.

Figure 1.2: Fruit variety and detachment method classification [65]

1.1. Complete systems

To gain a comprehensive understanding of the state of the art in robotic fruit picking, this section highlights a selection of fully operational systems. Given that this thesis focuses on the automatic harvesting of fruits using a 6-Degrees Of Freedom (DOFs) anthropomorphic robotic arm, only the most relevant projects within this category are included.

1.1.1. Academic complete systems

Among the most innovative advancements in agricultural robotics, several cutting-edge projects from academic research teams and university spin-offs are driving the development of intelligent harvesting technologies, as shown in Figure 1.3:

- **NMBU and Saga Robotics**: Researchers at the Norwegian University of Life Sciences (NMBU) have developed an advanced robotic strawberry harvester equipped with innovative obstacle-separation and path-planning algorithms to enhance efficiency and effectiveness in high-value crop automation. Building on this research, Saga Robotics, founded by NMBU researchers, is pioneering the "Thorvald"[46] modular robotic platform, a versatile system configurable for various agricultural tasks, including strawberry harvesting, making it a valuable tool in precision agriculture.
- **University of Plymouth and Fieldwork Robotics**: At the University of Plymouth, Fieldwork Robotics[16], a university spin-out, is developing a raspberry-picking robot that utilizes 3D cameras, sensors and machine learning to identify and harvest ripe fruit. This project marks a significant step toward commercializing robotic harvesting technologies for soft fruits.
- **Central Queensland University**: Meanwhile, researchers at Central Queensland University[53] are advancing robotic solutions for mango harvesting. Their system integrates LED lighting, sensors and cameras to estimate fruit weight, position and ripeness, optimizing the harvesting process and minimizing waste.
- **Wageningen University & Research**: In the context of the European Union's Horizon 2020 research and innovation program, the SWEEPER[41] project was carried out. SWEEPER is a robotic system designed to harvest sweet peppers in greenhouses. It features a 6-DOFs industrial arm with a specialized end effector, RGB-D camera and other advanced technologies. The robot is mounted on a mobile platform that autonomously moves along pipe rails and concrete floors.

(a) Saga Robotics strawberry picker

(b) Fieldwork Robotics raspberry picker

(c) Central Queensland University mango picker

(d) Wageningen Peppers harvester

Figure 1.3: Main academic existing complete systems

1.1.2. Industrial complete systems

Also the private sector can offer very interesting solutions for fruit picking, bringing cutting-edge automation technologies to market at a commercial scale, as shown in Figure 1.4:

- **Octinion's Rubion**: Based in Belgium, Octinion[35] has developed the Rubion strawberry-picking robot, a sophisticated system utilizing an RGB camera paired with an advanced 3D vision system to accurately detect ripe strawberries. The Rubion features a patented soft gripper designed to delicately handle the fruit, minimizing damage during harvesting. Its adaptive design enables it to operate effectively in varied field conditions, making it a significant industrial advancement in automating high-value crop harvesting.
- **FFRobotics**: FFRobotics[15], based in Israel, is focused on the development of a robotic apple-picker with multiple robotic arms for precise fruit handling. With ad-

vanced machine vision and control algorithms, their system aims to automate apple harvesting with high efficiency and minimal fruit damage—an important challenge in the industrial automation of fruit picking.

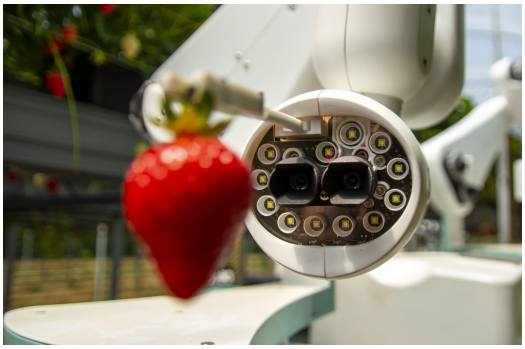
- **Harvest CROO Robotics**: Operating out of Florida, Harvest CROO[12] Robotics has introduced the Berry 5 robot, designed for large-scale strawberry harvesting. The Berry 5 can harvest up to eight acres per day, greatly reducing manual labor and operational costs. Combining a mobile platform, cutting-edge vision systems and specialized end-effectors, the Berry 5 is an industrial solution to optimizing the strawberry-picking process.
- **Advanced Farm Technologies (AFT)**: AFT[1], based in California, has commercialized the T-6 mobile strawberry-picking robot, designed for both greenhouse and open-field environments. The T-6's advanced navigation and picking technology streamline the harvesting process, offering an industrial solution to address labor shortages in the agricultural sector through automation.
- **Dogtooth Technologies**: This UK-based start-up[12] is developing a robotic strawberry-picking system for table-top growing systems. After conducting field trials in Australia, Dogtooth Technologies has refined its design to perform optimally in real-world conditions. Through the integration of advanced sensors and computer vision, the company is focused on achieving precise fruit detection and gentle handling—key requirements for industrial-scale robotic harvesting.
- **Abundant Robotics**: Abundant Robotics[43], based in California, has developed an autonomous, tractor-style apple-picking vehicle. This system uses a vacuum-based mechanism to remove apples from trees and incorporates advanced navigation and control systems, designed for large orchard environments. This innovative industrial solution addresses the labor-intensive nature of apple harvesting.
- **Tevel Aerobotics Technologies**: Tevel Aerobotics Technologies[50], based in Israel, integrates aerial robotics with agriculture by combining drone technology and AI algorithms for fruit picking. UAVs are employed to detect and harvest fruit in orchards, offering a scalable solution to challenges related to accessibility and labor shortages in the industry. This cutting-edge integration of robotics and AI is paving the way for precision agriculture on a large industrial scale.

(a) Octinion Rubion

(b) FFRobotics apples picker

(c) Harvest CROO strawberries picker

(d) Advanced Farm Technologies apples picker



(e) Dogtooth strawberries picker

(f) Abundant Robotics apples picker

(g) Tevel Aerobotics Technologies apples picker

Figure 1.4: Main industrial existing complete systems

1.2. Current technological limitations

By analizing in depth the literature, it is possible to understand the specific design choices adopted by each of the aforementioned projects. In particular, several limitations of the current technologies leave some partially, or even totally, unsolved problems. According to [65] the most notable still opened challenges are:

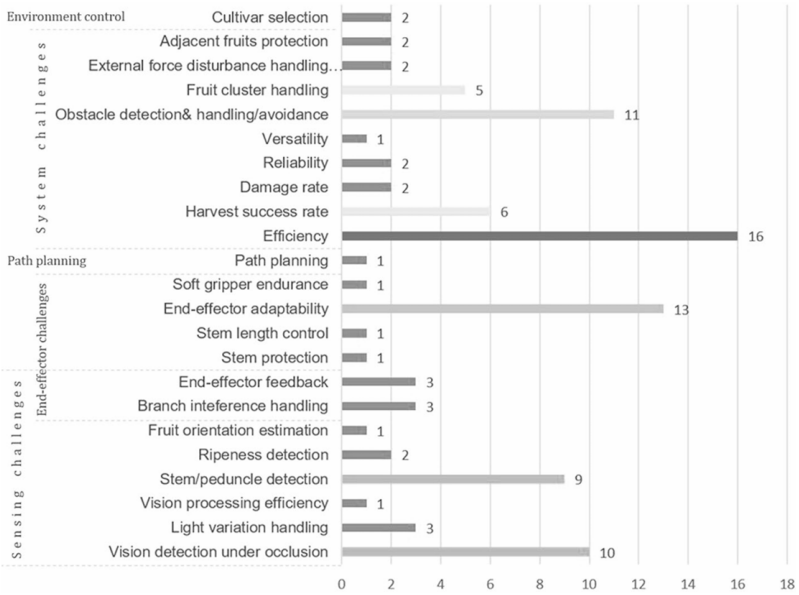


Figure 1.5: Main opened technological challenges [65]

As shown in Figure 1.5, the most crucial limitation of the robotic system present in the literature is the low overall efficiency. This lack of picking efficiency, both in terms of picking time and success rate, can be traced back to the other crucial factors extracted by [65]. In descending order of apperance,the first three opened challenges are:

- **End effector adaptability**
- **Obstacles detection, handling and avoidance**
- **Detection under occlusion**

The first issue can be improved by a tailored selection of the robot end effector, evermore if it is made of innovative materials, as explained by [34] in its review of the so called "soft robotics grippers". The other two are far more systematic challenges, that can be found in most of the fruit picking applications. For this reason, a deeper literature review is carried out to find the most recent solutions in these fields.

1.3. Object detection and localization

In the domain of agricultural automation and computer vision, the accurate detection and location of trees and fruits is of paramount importance. While the detection of trees in both structured and unstructured environments poses a substantial and intricate challenge in itself [20, 27], this thesis work focuses specifically on the problem of fruit detection. By narrowing the focus, the objective is to investigate and address the particular challenges associated with identifying fruits in diverse backgrounds and lighting conditions.

1.3.1. The object detection problem in agriculture

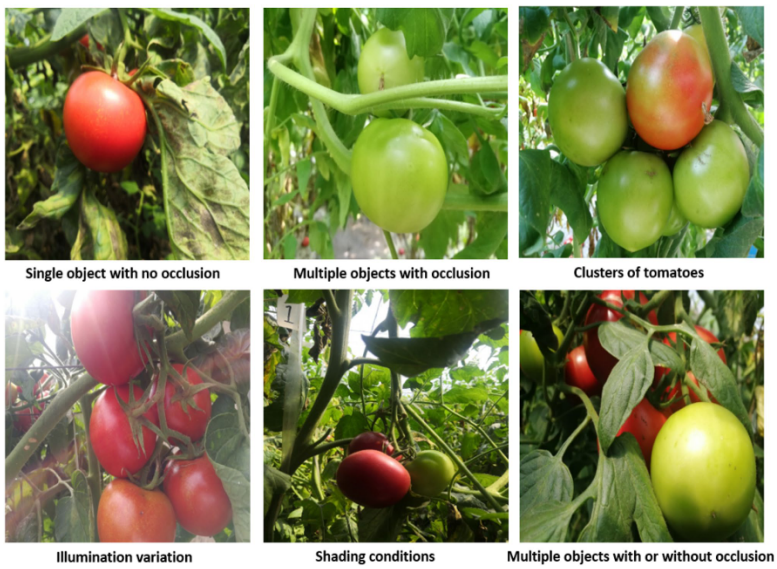
Identifying the target fruit is essential for the picking robot and significantly impacts its performance. In unstructured environments, various interfering conditions can affect fruit recognition, making automatic target identification one of the major challenges in visual control. To tackle this problem, researchers have developed numerous recognition methods, several of which have been implemented in picking robots. These methods can be broadly classified into three categories:

- **Single feature analysis**: relying on just one characteristic, mainly color or shape;
- **Multi feature fusion analysis** : exploring the application of more tailored techniques, such as Clustering and Dynamic thresholds;
- **Machine learning-based approaches**: exploiting state-of-the-art techniques from computer vision.

Early attempts at fruit detection primarily relied on color-based information to identify and locate fruits. However, this approach proved insufficient, especially in natural environments where lighting conditions, shadows, occlusions and variations in fruit color made accurate detection challenging. Recognizing these limitations, researchers shifted towards more robust methods. Kelman and Linker [23] demonstrated that shape analysis could effectively detect apples without depending on color.
Similarly, Jidong et al. [21] improved fruit detection by integrating Otsu's dynamic threshold segmentation, edge detection and the randomized Hough transform, enabling more reliable identification. Liu et al. [29] further advanced detection techniques by incorporating Simple Linear Iterative Clustering (SLIC) and Histogram of Oriented Gradient (HOG) features to detect partially visible apples.
While color and shape-based methods improved fruit detection, they still face significant challenges, particularly with varying light conditions and occlusions. Natural environments introduce inconsistencies in illumination due to shadows, reflections and changes

(a) Limiting light conditions [56]

(b) Foreground-Background affinity [4]



(c) Typologies of fruit occlusions [26]

Figure 1.6: Main issues in fruit detection with color and shape-based methods

in sunlight (Figure 1.6a), which can alter the perceived color of the fruit and lead to misclassification.
Additionally, fruits are often partially hidden by leaves, branches, or other fruits (Figure 1.6b), creating a critical issue where the background, such as green leaves, merges with the foreground (Figure 1.6c), i.e. the fruits, making it difficult for both color and shape-based methods to accurately distinguish the fruit from its surroundings. These challenges further complicate the detection and segmentation process. These limitations, as shown in Figure 1.6 [4], [26] and [56], reduce the reliability of traditional computer vision approaches, as they struggle to generalize across different environments.
Building on the early developments of color-based methods, recent years have seen the emergence of Convolutional Neural Networks (CNNs) and Deep Learning (DL) techniques, which address the limitations of traditional approaches. The aforementioned challenges prompted the development of CNNs and DL methods, which utilize more robust feature extraction techniques to overcome these obstacles and improve detection accuracy in complex, real-world environments.

1.3.2. DL and CNNs

As highlighted by Xiao et al. [55], the recent advancements in the powerful realm of CNNs and DL have quickly made their way into the agricultural sector. Innovative research, incorporating the application of convolutional neural networks (e.g., VGGNet, SSD, YOLO) and event-based vision systems, further refines the capabilities of fruit detection. Machine learning techniques, including clustering algorithms, Bayesian classifiers, KNN and SVM-based methods, also play a crucial role in boosting recognition accuracy under varying environmental conditions.
Deep learning, a subset of machine learning, works by training artificial neural networks on large datasets to automatically learn patterns and representations, allowing models to generalize across different conditions without relying on manually designed features. These networks are composed of interconnected layers, where each layer consists of multiple artificial neurons that process and transmit information to the next layer. Each neuron applies a mathematical operation to its inputs, transforming the data and passing it forward, allowing the network to progressively learn more abstract and high-level features from raw images, as summarized in Figure 1.7.

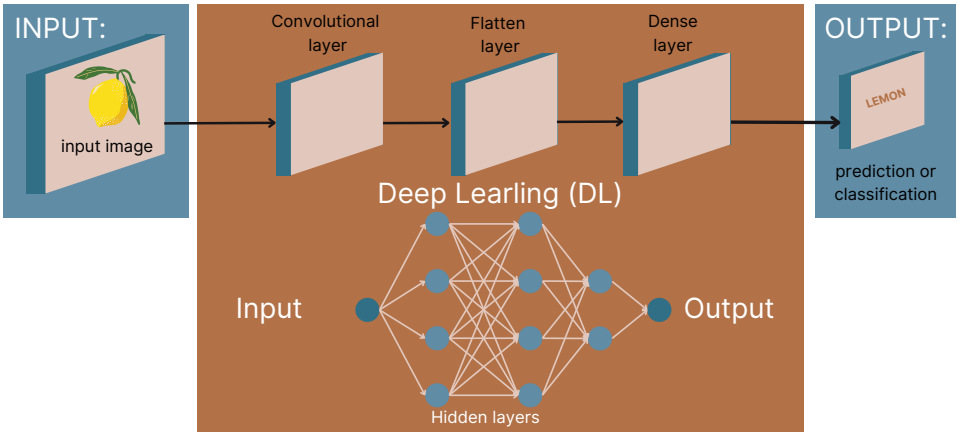


Figure 1.7: DL structure

On the other hand,CNNs have revolutionized object detection in agriculture by leveraging their hierarchical feature extraction capabilities, enabling models to identify fruits with high precision even in challenging environments. These networks consist of multiple layers, including convolutional, pooling and fully connected layers, each serving a specific function in detecting and classifying objects. The convolutional layers extract key spatial features from input images by applying a series of learnable filters, which help in recognizing textures, edges and shapes of fruits. Pooling layers reduce the spatial dimensions, enhancing computational efficiency while retaining essential information. Fully connected layers then process the extracted features to classify the detected objects, distinguishing fruits from leaves, branches and other background elements, as shown in Figure 1.8.

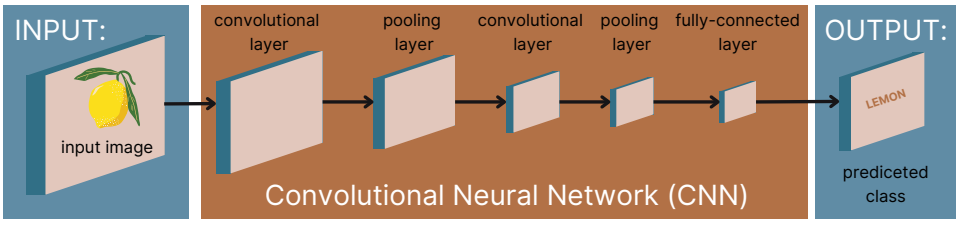


Figure 1.8: CNNs layered structure

Among CNN-based object detection frameworks, one-stage detectors such as SSD (Single Shot MultiBox Detector) and YOLO offer fast and efficient detection by predicting bounding boxes and class probabilities in a single forward pass, making them suitable for real-time applications in agricultural robotics. In contrast, two-stage detectors like Faster R-CNNs generate region proposals before classification, providing higher accuracy but at the cost of increased computational complexity. These deep learning models excel in han-

dling challenges such as occlusions, varying lighting conditions and complex backgrounds, which traditionally hindered color- and shape-based detection methods.
Additionally, advancements in transfer learning allow pre-trained models to be fine-tuned on agricultural datasets, reducing the need for extensive labeled data and improving performance in domain-specific tasks. The integration of CNNs with event-based vision, hyperspectral imaging and multimodal sensor fusion further enhances detection robustness, enabling real-time, high-accuracy fruit recognition for automated harvesting, yield estimation and quality assessment.

1.3.3. Implemented solutions

The majority of implemented solutions for fruit detection in agriculture have been based on Faster R-CNN and YOLO, two of the most widely used deep learning models for object detection.
Faster R-CNN, a two-stage detector, is known for its high accuracy, as it first generates region proposals and then classifies and refines bounding boxes, making it effective in complex agricultural environments where fruits may be occluded or partially hidden.
Pan and Ahamed [38] explored the use of a 3D stereo camera combined with Mask R-CNN for pear recognition in orchards, leveraging both depth and RGB images to improve detection accuracy. A dataset of 9054 RGBA images, collected under varying lighting conditions, was used for training and evaluation. Comparative analysis showed that Mask R-CNN outperformed Faster R-CNN, achieving a higher mean Average Precision (mAP) of 99.45% in testing, particularly excelling in detecting clustered pears. These findings emphasize the effectiveness of R-CNN-based object detection, particularly in handling fruit clustering. However, its computational cost is relatively high, making it less suitable for real-time applications.

Model	Validation Set	Testing Set
Faster R-CNN	87.90%	87.52
Mask R-CNN	95.22%	99.45

Table 1.1: mAP results for the testing and validation sets [38]

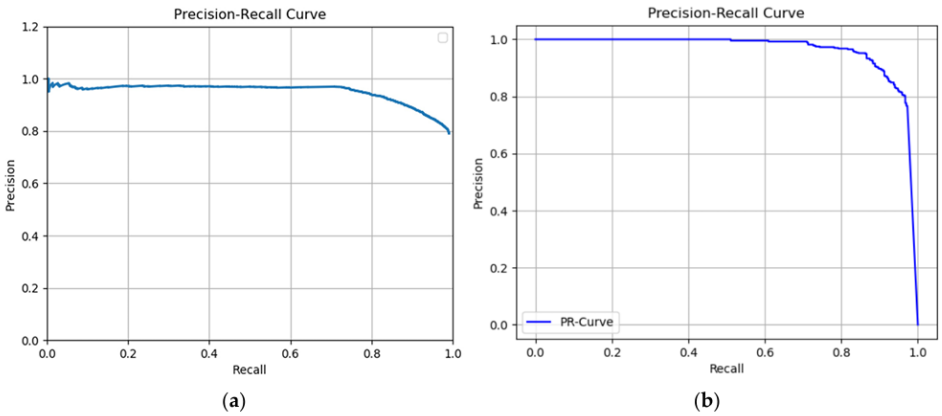


Figure 1.9: Precision-Recall curve of Faster R-CNN(a) and Mask R-CNN(b) at learning rate = 0.001 [38]

In recent years, YOLO has emerged as a strong competitor in fruit detection, gaining popularity due to its simplicity and high computational efficiency. Unlike two-stage detectors like Faster R-CNNs, YOLO follows a one-stage detection approach, predicting bounding boxes and class probabilities in a single forward pass. This streamlined process significantly enhances detection speed, making YOLO well-suited for real-time applications in precision agriculture, such as automated harvesting and crop monitoring. Its balance between accuracy and efficiency has led to its widespread adoption in recent studies, demonstrating its effectiveness in addressing the challenges of fruit recognition in dynamic and complex orchard environments. Various enhancements have been introduced across different YOLO versions to improve accuracy and compete with other advanced detection models.

Chen et al. [9] created an improved YOLOv4-based method for real-time citrus detection in orchard environments, addressing challenges such as occlusions and varying growth stages. RGB images were collected using a Kinect V2 camera and the Canopy and K-Means++ algorithms were applied to optimize the selection of prior frames. Enhancements to the YOLOv4 network focused on improving the detection of smaller citrus fruits against complex backgrounds. Additionally, sparse training techniques were used to prune unnecessary network layers while fine-tuning parameters to recover accuracy.

Experimental results demonstrated a 3.15% increase in detection accuracy, reaching 96.04%, outperforming YOLOv3, the original YOLOv4 and Faster R-CNN. The model achieved an average detection speed of 0.09 seconds per frame at 1920×1080 resolution, as shown in Table 1.2:

Model	Growth Stage	F1 Score (%)	Average Time (s)	Accuracy (%)
Faster-RCNN	Growing period	86.24	0.32	0.84
YOLOv3	Growing period	83.97	0.21	0.82
YOLOv4	Growing period	87.51	0.12	0.92
Improved YOLOv4	Growing period	91.95	0.10	0.95
Faster-RCNN	Maturity	86.48	0.32	0.86
YOLOv3	Maturity	84.17	0.21	0.82
YOLOv4	Maturity	87.58	0.11	0.92
Improved YOLOv4	Maturity	92.13	0.09	0.96

Table 1.2: Performance comparison of different models at different growth stages [9]

This improved YOLOv4 network showed also an impressive capability at handling occlusions, as reported in 1.3:

Occlusion Condition	Model	Citrus Count	Correctly Identified		Falsely Identified		Missed	
			Amount	Rate (%)	Amount	Rate (%)	Amount	Rate (%)
Less than 50%	Faster-RCNN	200	162	81.24	19	9.71	31	15.44
	YOLOv3	200	156	78.22	23	11.52	38	18.96
	YOLOv4	200	173	86.38	14	7.10	22	11.21
	Improved YOLOv4	200	187	93.58	12	5.98	16	8.15
More than 50%	Faster-RCNN	200	156	78.24	26	12.81	39	19.34
	YOLOv3	200	148	74.22	35	17.52	46	22.96
	YOLOv4	200	166	83.18	20	10.05	26	13.07
	Improved YOLOv4	200	182	90.82	14	7.18	21	10.36

Table 1.3: Identification parameters across different levels of occlusion [9]

Other researchers have utilized YOLO's capabilities to address various challenges, including accurate fruit geometry estimation [66] and the assessment of ripeness or defect levels [18]. Eventually, also tailored versions of YOLO networks has been developed to tackle the problem of fruit clusters, in particular for tomatoes [26, 61], showing good results in single fruit detection. A decisive comparison between YOLOv4 and Mask R-CNN has been carried out by [13], for a potato tubers detection and segmentation application. The study analyzes the outcomes obtained using the trained YOLOv4 model, revealing limitations in densely clustered scenarios due to difficulties distinguishing individual tubers from the background. Grayscale image processing did not significantly improve detection accuracy, underscoring the need for more sophisticated approaches. On the other hand, the Mask R-CNN model showcased superior segmentation quality with consistently high average mean intersection of union, which ranged between 73% and 80% depending on the density of the pile of potatoes.

This resulted in slower performance compared to YOLOv4, leading to the conclusion that Mask R-CNN is more suitable for tasks requiring high clustering and precise segmentation, such as quality grading. Meanwhile, YOLOv4 excels in real-time field operations where speed is crucial. Given these results and the ongoing advancements in the YOLO architecture by Ultralytics [52], YOLO stands out as the optimal balance between training and inference speed and detection accuracy.

1.4. Trajectory Planning and Control of robot

The second challenge is planning the trajectory from the initial pose to the picking position, while avoiding obstacles and controlling the robot.
Additional objectives include minimizing picking time and effort, as well as reducing potential damage to the fruit and plants during the robot's movement.

1.4.1. The Trajectory Planning problem in agriculture

To analyze the critical challenges arising from planning and executing a safe picking trajectory, it is essential to distinguish between structured environments (e.g., laboratories, indoor setups, greenhouses) and unstructured environments (e.g., open fields, orchards).

Structured environments provide controlled conditions, both in terms of plant characteristics (e.g., visible fruits, pruned leaves) and external factors, ensuring minimal environmental variability. In such settings, a priori knowledge of plant position and geometry is often sufficient to plan and execute collision-free trajectories. Consequently, trajectory planning can rely solely on fruit positions and detected obstacles, allowing state-of-the-art algorithms such as Model Predictive Control (MPC), Particle Swarm Optimization (PSO) and Rapidly exploring Random Tree (RRT) to generate optimal paths.
Notable implementations from the industrial sector include Avanzini et al. [2] for MPC-based approaches, Zeng and Wu [59] for PSO and Zhou et al. [67] for RRT-based algorithms.

In contrast, unstructured environments introduce a significantly higher degree of uncertainty. Dynamic obstacles, such as unexpected human intervention or wildlife intrusions, require continuous adaptation. Static trajectory planning methods must be extended to track and respond to environmental changes in real time, necessitating a continuous loop of perception, planning and control to ensure safe and collision-free motion.

To address these challenges, Zhu et al. [69] proposed a cascaded nonlinear MPC framework with a two-stage optimization approach. The high-level MPC integrates an artificial

potential field to generate predictive and smooth trajectories, while the low-level MPC functions as a trajectory tracker and safety enforcer, imposing hard constraints to prevent collisions and singularities. Additionally, a super-twisting observer enhances the accuracy of dynamic obstacle motion estimation. Experimental results demonstrate that this approach effectively ensures safe and smooth obstacle avoidance in real-world applications with moving obstacles.

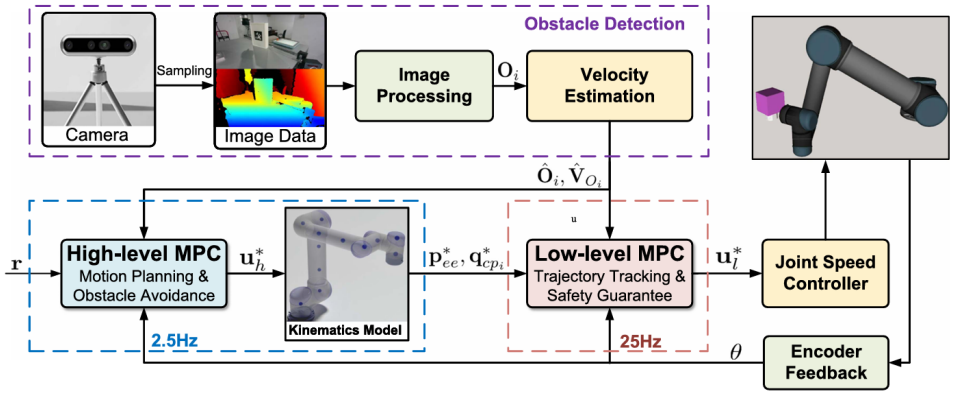


Figure 1.10: The cascaded MPC controller, where the optimal control sequences (u_l^*) is refined by the trajectory and obstacles position (O_i) and velocity ($\hat{V}_{O_i}$) update using control input (u_h^*) [69]

Palmieri and Scoccia [36] presented a motion planning and control framework for redundant manipulators, incorporating collision avoidance in dynamic environments. The study extends previously tested planar algorithms to full-mobility manipulators operating in three-dimensional workspaces. The proposed approach integrates offline path planning with online motion control. The path planning component utilizes the potential fields method combined with Bézier curve smoothing to generate obstacle-avoiding trajectories before the robot starts moving.

The online control strategy dynamically adjusts the manipulator's trajectory based on obstacle motion, employing a velocity control law using the null space method for redundancy resolution. An additional control term accounts for both the speed and position of obstacles, further enhancing collision avoidance.

The framework has been validated through simulations involving the KUKA LBR iiwa collaborative robot, demonstrating significant reductions in angular accelerations (up to 90%) while maintaining computational feasibility.

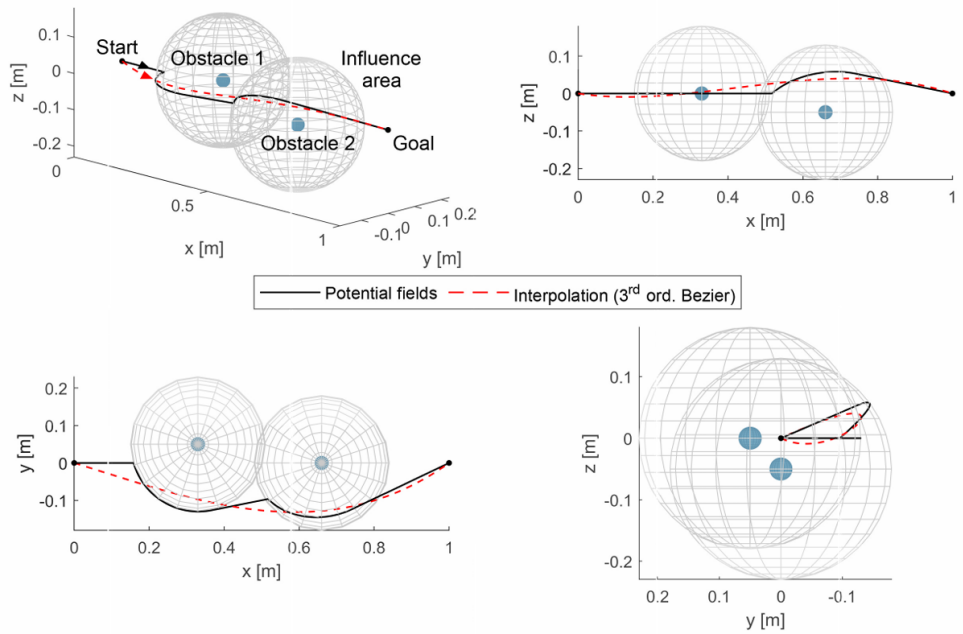


Figure 1.11: The 3D obstacles–aware path planning proposed by [36]

1.4.2. Implemented solutions

Among the implemented projects regarding trajectory planning and control of picking robots the first notable example that can be found is represented by [64]. This study focuses on enhancing the adaptability of robotic manipulators for automatic tomato harvesting in unstructured environments. A novel planting pattern and manipulator structure were designed based on the biological and cultivation characteristics of tomatoes.
By simplifying the fruit growth space from a cuboid to a rectangle, mathematical models for structural optimization were developed, maximizing workspace area while considering workspace perimeter constraints. Using MATLAB's optimization toolbox, the structure parameters were refined and a Monte Carlo simulation verified the feasibility of a 4-degree-of-freedom (DOF) manipulator for harvesting. Based on the optimized parameters, four fundamental working postures were proposed.
A collision-free, efficient picking trajectory was developed using a cycloid-based path planning algorithm, ensuring smooth transitions between picking, releasing and intermediate postures. MATLAB simulations (Figure 1.12) confirmed that the planned trajectory effectively avoids collisions with stems, leaves and other fruits, improving the efficiency and reliability of the tomato harvesting process.

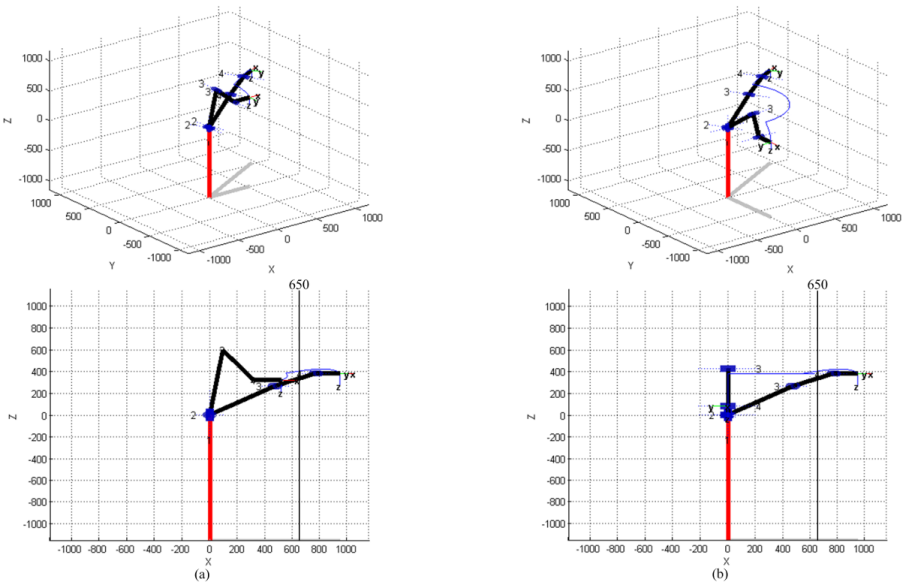


Figure 1.12: Matlab simulation results [64]: preparation position to picking position (a); picking position to fruit release position (b)

Ye et al. [57] proposed a two steps method based on binocular stereo vision. First, an improved adaptive weight particle swarm optimization (APSO) algorithm is used to solve the inverse kinematics of the robot and determine a collision-free picking posture. Second, to improve the efficiency of the Bi-RRT algorithm in high-dimensional environments, a modified version called AtBi-RRT is introduced that incorporates target gravity and adaptive coefficient adjustments for faster convergence. Simulations show that APSO quickly finds an optimal picking posture, while AtBi-RRT determines a collision-free path with an average computation time of 4.24 seconds and a 100% success rate in a laboratory setting, as reported in Table 1.4:

Senses	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Picking deviation in real experiment (mm)	4.06	5.49	4.72	1.87	2.97	5.41	1.47	3.46	12.11	7.18
Position error in simulation result of APSO (mm)	0.34	0.18	0.002	0.13	0.12	0.76	0.21	0.51	0.05	1.08
Orientation deviation in real experiment (°)	11	9	17	11	12	15	14	15	13	11
Orientation error in simulation result of APSO (°)	7.42	8.96	7.98	7.05	6.89	6.45	6.12	6.07	6.34	5.65

Table 1.4: Comparison of the deviations of the experiments and errors of the simulations.

Cao et al. [7] tackled the challenge of motion planning for robotic fruit-picking by developing a multi-objective trajectory model for a robotic manipulator. It introduces an improved multi-objective particle swarm optimization algorithm (GMOPSO) that increases population diversity, avoids local optima and accelerates convergence through mutation operators, annealing factors and a feedback mechanism. The performance of the algo-

rithm is validated using the ZDT1-ZDT3 benchmark functions and compared with other multi-objective evolutionary algorithms. Simulations optimise the manipulator's trajectory based on time, energy consumption and pulsation, achieving a near-Pareto optimal solution. Practical experiments show that the optimised trajectory enables smooth and efficient fruit picking, with an average picking time of 25.5 seconds and a success rate of 96.67%.

Other two notable improvements [22, 58] of the standard RRT algorithm tried to increase the overall path planning speed.
Yoshida et al. [58] assembled a dual-armed fruit harvesting system designed to automate fruit picking of pears in Japan. This robotic system boosts two Universal Robotic manipulators and is used to simultaneously carry out fruits searching and picking (Figure 1.13).

(a) The harvesting robot (b) Picking procedure

Figure 1.13: Dual-armed fruit harvesting system [58]

The robot operates in orchards with joint V-shaped trellises, using an installed Intel's Real sense D435 depth camera and computer vision to detect and estimate fruit positions. To prevent collisions with other fruits or its own arms, the system applies inverse kinematics and a fast path-planning method based on random sampling.
In this study several RRT-based algorithm are confronted (Table 1.5), showing that Transition-RRT (T-RRT) can perform very fast path planning compared to RRT*, Probabilistic Road Map (PRM) and PRM*. On the other hand, RRT is faster than T-RRT because the basic RRT method is not optimized. Kang et al. [22] introduced an improved bidirectional path planning method, EDDS-bi-RRT, to guide the tree Expansion Direction through a Dynamic Step (EDDS). Built on the rapidly exploring random tree (RRT) algorithm, it dynamically adjusts expansion direction and step size based on scene

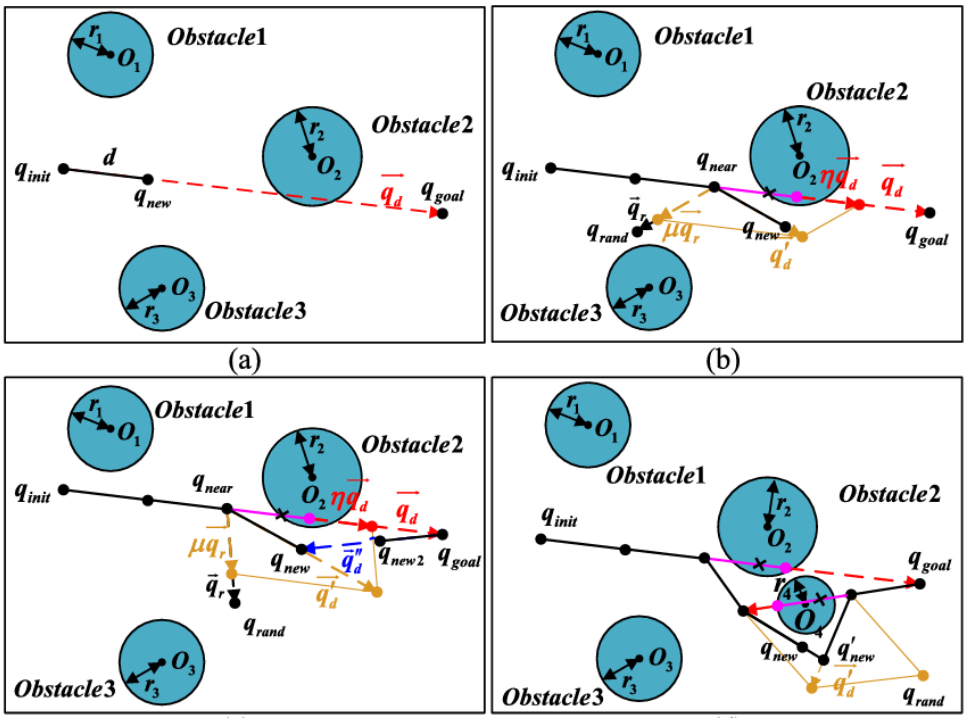
Target		RRT	T-RRT	RRT*	PRM	PRM*
1	Planning [sec]	0.097	0.236	10.115	10.083	10.111
	Harvest [sec]	9.793	9.744	9.785	9.907	9.739
	Sum [sec]	9.89	9.98	19.9	19.99	19.85
2	Planning [sec]	0.497	0.497	10.114	10.109	10.132
	Harvest [sec]	9.843	11.833	11.996	11.951	11.898
	Sum [sec]	10.34	12.33	22.11	22.06	22.03
3	Planning [sec]	0.432	0.268	10.078	10.179	10.104
	Harvest [sec]	13.088	11.962	11.842	13.041	11.986
	Sum [sec]	13.52	12.23	21.92	23.22	22.09
4	Planning [sec]	0.269	0.24	10.137	10.106	10.083
	Harvest [sec]	8.511	8.86	8.313	8.344	8.047
	Sum [sec]	8.78	9.10	18.45	18.45	18.13

Table 1.5: Performance comparison between RRT-based algorithms

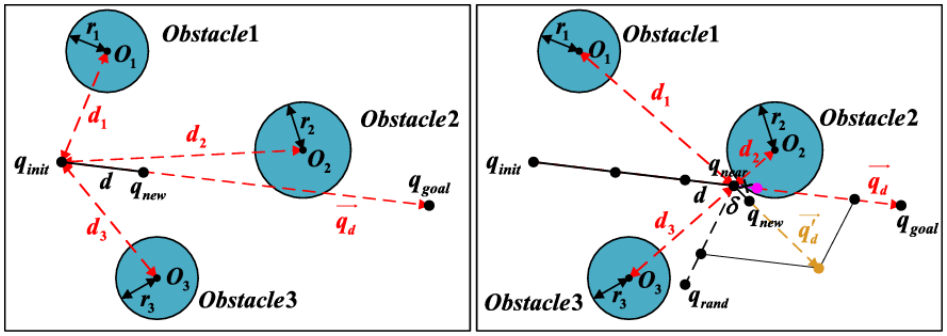
information. Key innovations include:

- adjusting tree growth direction using cosine and sine factors when obstacles are detected;
- converting the robot's maximum safe distance into an expansion step using matrix operator norms, optimizing path efficiency.

as shown in Figure 1.14.
Simulations and experiments with a 17-DOFs apple-harvesting robot demonstrate improved planning speed, shorter path length and effective collision-free navigation in unstructured environments.



(a) Sampling direction



(b) Dynamic step

Figure 1.14: EDDS-bi-RRT algorithm [58]

1.5. 3D Environment Reconstruction

There is a growing emphasis in the literature on reconstructing 3D picking environments. A 3D representation of the surroundings enhances perception by mimicking human spatial awareness, allowing for immediate differentiation between fruits, obstacles and occlusions while providing a complete tridimensional understanding of object positions.
Point clouds are currently the most effective tool for capturing 3D spatial information, integrating positional data with additional features such as color. Various methods have been developed to generate point clouds from 2D images, depending on the available data and the sensing technology used.
The state-of-the-art approach involves RGB-D point cloud reconstruction, which utilizes depth information from specialized sensors like lasers and infrared cameras.
More recently, advancements have enabled point cloud generation using only standard 2D RGB images. The following sections highlight some of the most notable projects employing both techniques.

1.5.1. Depth camera and Lidar based solutions

Barnea et al. [4] addressed the problem of unreliable fruits detection under unstable lighting conditions, where fruit color blends with the background. Color-based computer vision remains highly sensitive to illumination changes, making segmentation and detection challenging. While multispectral imaging could mitigate this issue, a simpler and more cost-effective alternative is desired.

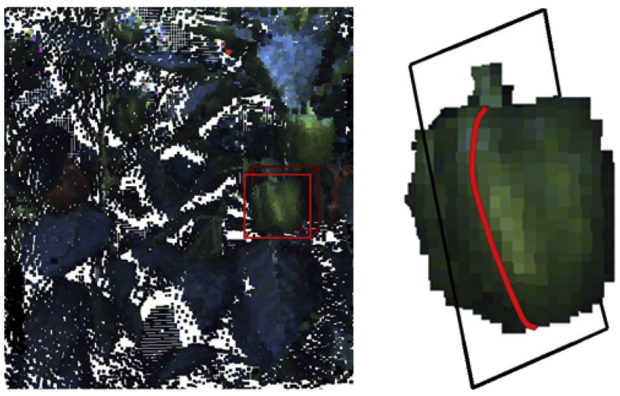


Figure 1.15: 3D point cloud provided by the depth camera [4]: preparation position to picking position (a); picking position to fruit release position (b)

This study proposed a shape-based detection method combining RGB and range data to analyze 3D surface normal features, reflective symmetry and elliptic surface highlights. The approach enables robust fruit detection in 3D space regardless of color (Figure 1.15), demonstrating effectiveness on a challenging sweet pepper dataset with significant occlusions.

A novel 3D descriptor, Color-FPFH, was introduced by Tao and Zhou [49], combining color and geometric features. The system classified objects into apples, branches and leaves (Figure 1.16), improving 3D perception.
A support vector machine (SVM) classifier, optimized with a genetic algorithm, was trained for recognition. Comparative analysis with other descriptors and classifiers demonstrated superior performance. Additionally, the method's ability to estimate occlusions was evaluated (Figure 1.17), confirming its effectiveness for accurate apple detection in robotic harvesting.

Figure 1.16: 3D point cloud of apples, branches and leaves provided by [4]

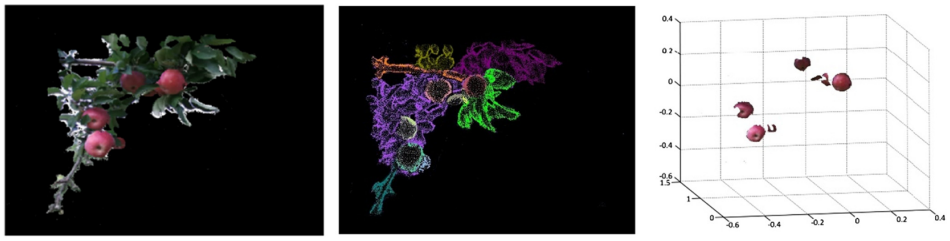


Figure 1.17: Pre–processed point cloud (a), segmented point cloud represented by different colors (b) and individual apple point clouds (c)

Lin et al. [28] developed a robust RGB-D-based algorithm for detecting and localizing citrus fruits in outdoor orchards for robotic harvesting. The method employs a depth filter and Bayes-classifier-based segmentation to remove background noise, followed by density clustering to group potential citrus fruits. A support vector machine classifier using color, gradient and geometry features eliminates false positives. Tested on 506 RGB-D images under varying lighting conditions, the algorithm achieved an F1 score of 0.9197, with minimal positioning and sizing errors, demonstrating its effectiveness in guiding citrus-harvesting robots.

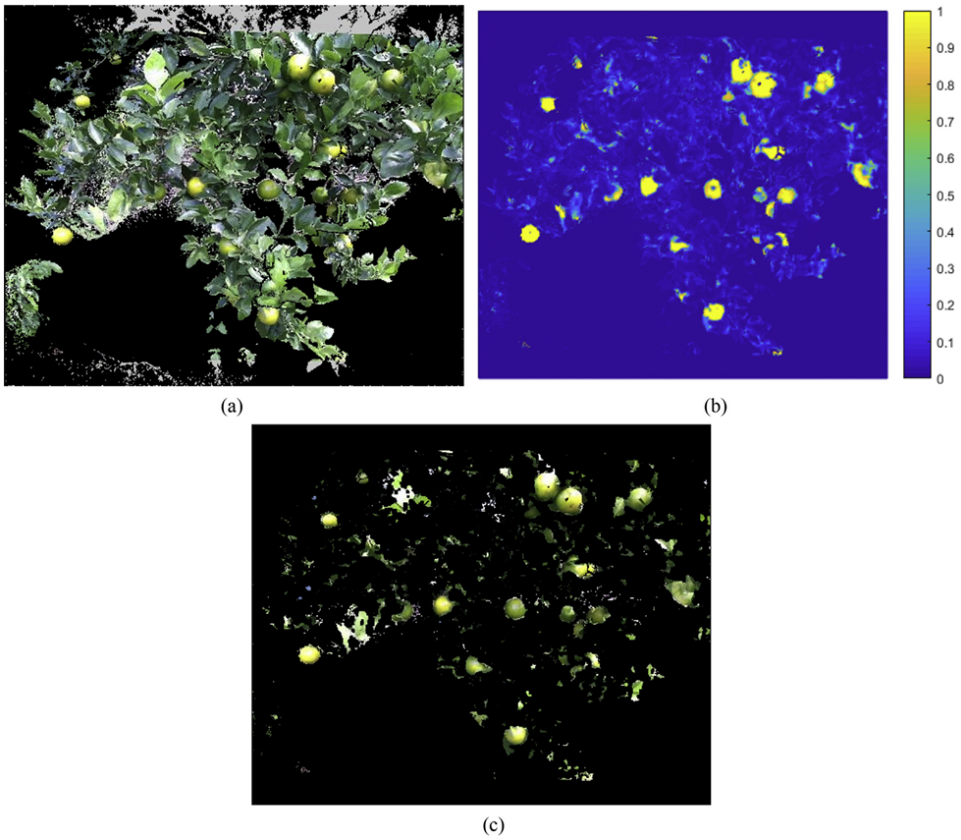


Figure 1.18: RGB-D image post depth filtering (a), generated probability map (b) and final segmentation outcome (c) by [28]

The Tomato Pose Method (TPM), proposed by Zhang et al. [60], is a 3D pose detection approach that integrates a priori geometric modeling, a cascaded multi-task network, and a 3D reconstruction process. The model utilizes prior agronomic knowledge to describe the spatial arrangement of tomato bunches and employs a deep learning-based network for keypoint and bounding box prediction.
Using a dataset of 1800 RGB-D images under varying conditions, TPM achieves a 94.02%

success rate in 2D keypoint detection and reconstructs multi-pose tomato bunches with 70.05% accuracy. Notably, it enables real-time 3D reconstruction in 1.0–2.0 seconds using a single RGB-D image, providing crucial data for robotic picking path planning.

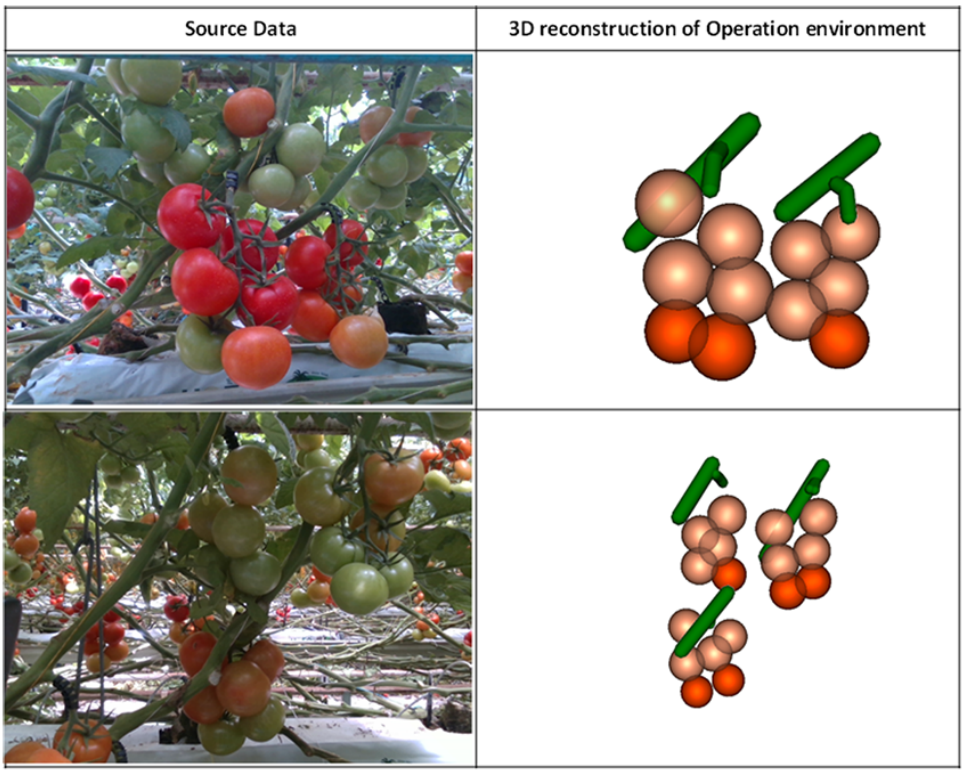


Figure 1.19: 3D reconstruction in multi-bunch scenarios results [60]

1.5.2. RGB camera-based solutions

In recent years, significant advancements have been made in photogrammetry, particularly in generating point clouds from standard 2D RGB images.
Structure from Motion (SfM) techniques, such as COLMAP [47], have greatly improved camera position estimation from image datasets. Building on these developments, Neural Radiance Field (NeRF) has transformed novel view synthesis and 3D reconstruction, finding applications in robotics, urban mapping, autonomous navigation, and VR/AR. Since its introduction by Mildenhall et al. [32], NeRF has gained widespread attention, resulting in over 250 preprints and 100+ accepted papers in leading Computer Vision conferences. This method achieves state-of-the-art results in novel view synthesis by optimizing a continuous volumetric scene representation using a sparse set of images. NeRF optimizes a deep fully-connected neural network without any convolutional layers (often referred to as a multilayer perceptron or MLP), taking a 5D input (spatial location and viewing direction) and outputting volume density and view-dependent radiance (Figure 1.20).

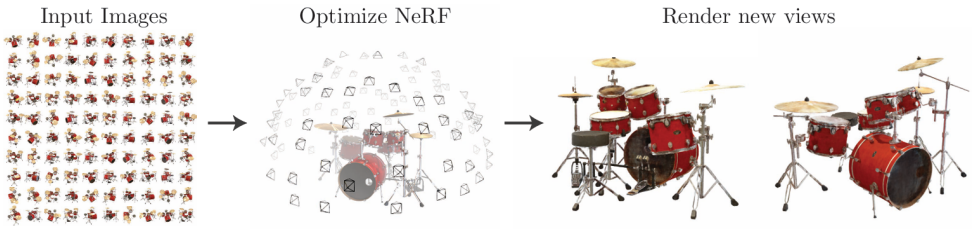


Figure 1.20: NeRf workflow [32]

This method generates photorealistic novel views, by querying 5D coordinates along camera rays and applying volume rendering (Figure 1.21).
Since volume rendering is differentiable, the approach requires only images with known camera poses for optimization. The results surpass previous neural rendering and view synthesis techniques, producing highly realistic scene reconstructions.

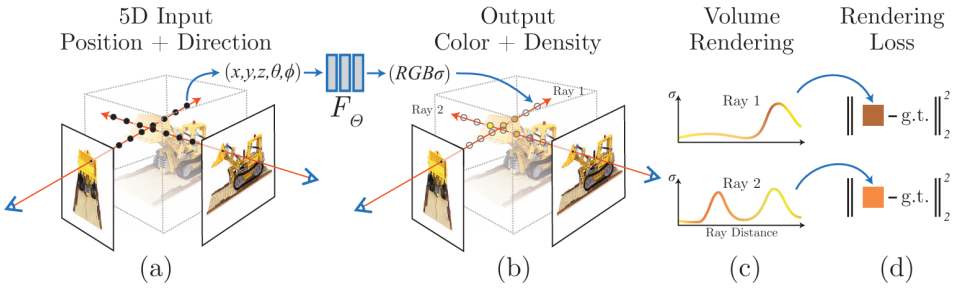


Figure 1.21: 5D coordinates sampling along camera rays (a), MLP (F_θ) produce a color and volume density (b), volume rendering techniques to compose the image (c) and scene optimization by minimizing the residual between synthesized and ground truth observed images (d) [32]

Several agricultural applications have been developed based on NeRF frameworks.
Hu et al. [19] focused on two key tasks in plant phenotyping: 2D novel-view synthesis and 3D reconstruction. This article evaluates two state-of-the-art NeRF methods: Instant-NGP for high-quality image generation and Instant-NSR, which improves geometry reconstruction by incorporating the Signed Distance Function (SDF).
The study presents a novel plant phenotype dataset featuring real images from production environments, marking the first comprehensive exploration of NeRF in agriculture. Results show that NeRF performs well in novel-view synthesis and achieves competitive 3D reconstruction compared to Reality Capture [8] (Figure 1.22).

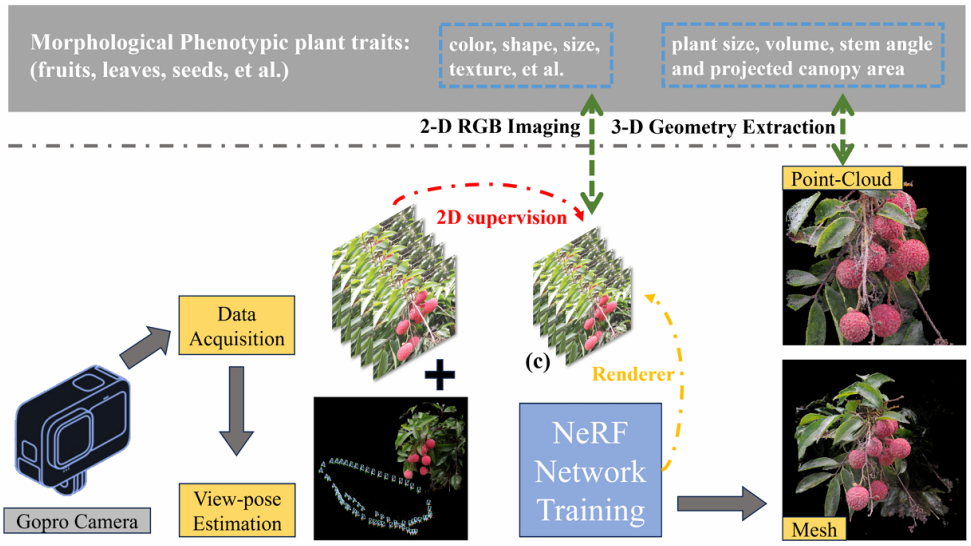


Figure 1.22: Phenotyping framework via NeRF [32]

However, challenges remain, such as slow training speeds, performance limitations with insufficient sampling, and difficulties in complex setups.

Zhang et al. [63] addressed the challenges of efficiently and realistically reconstructing large-scale 3D orchard scenes. Traditional methods struggle with modeling efficiency and computational costs. To overcome these limitations, the NeRF-Ag model, an extension of NeRF, is proposed. NeRF-Ag incorporates a multi-resolution latent feature encoding technique to improve training efficiency and modeling precision (Figure 1.23). It also embeds environmental factors for enhanced robustness. Experimental results show that NeRF-Ag achieves photo-realistic rendering, surpassing NeRF in PSNR, SSIM and LPIPS, with a 39x faster training speed. In 3D reconstruction, it provides better texture detail and accuracy than the COLMAP method. The study also explores free-viewpoint rendering and highlights the importance of training image quantity for 3D rendering precision, contributing to agricultural digital twin systems.

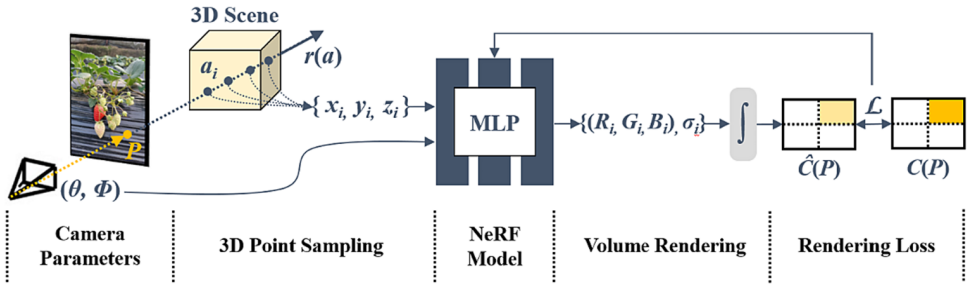


Figure 1.23: NeRF-Ag pipeline [63]

Saeed et al. [45] introduced PeanutNeRF, a deep learning-based 3D reconstruction method for accurate phenotypic analysis of peanut plants.
2D images were used for detailed feature capture, and neural radiance fields (NeRF) were optimized for implicit 3D plant representation. The trained NeRF model generated 3D point clouds for 360-degree and frontal-view reconstructions. Using the developed Frustum PVCNN (Figure 1.24), 3D detection of peanut pods was performed, achieving a chamfer distance below 4×10^{-4} and a precision of 0.7 at an IoU threshold of 0.5. This method enhances plant phenotyping and can be applied to other crops.

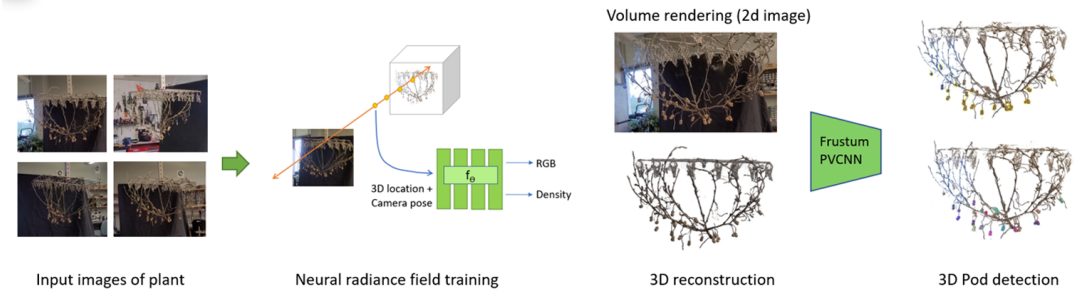


Figure 1.24: PeanutNeRF framework for peanut plants [45]

Eventually, FruitNeRF [31] is a novel 3D fruit counting framework that utilizes state-of-the-art view synthesis techniques to count any fruit type directly in 3D.
The system processes unordered monocular camera images, segments fruits using a foundation model, and trains a semantic neural radiance field (NeRF) with both RGB and segmentation data. By extracting fruit-only point clouds and applying cascaded clustering, it achieves accurate fruit counting while avoiding issues like double counting or misidentifying irrelevant objects (Figure 1.25). Evaluations on real-world and synthetic datasets demonstrate its effectiveness across various fruit types, outperforming traditional methods like object tracking and optical flow.

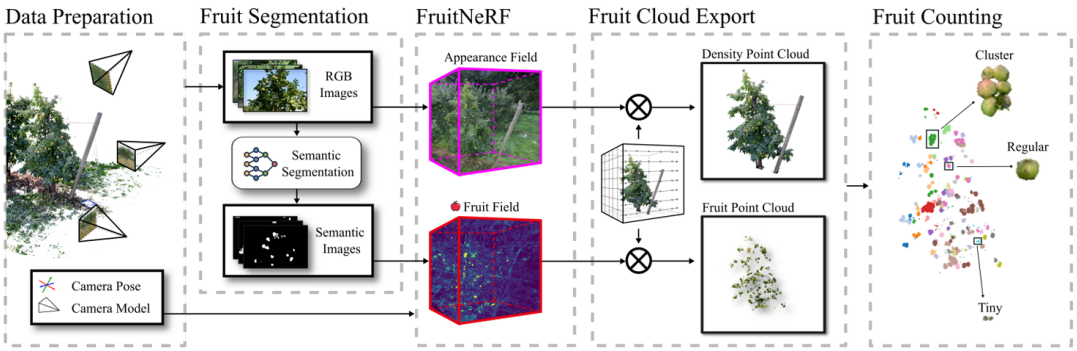


Figure 1.25: PeanutNeRF framework for peanut plants [31]

2 | Methodologies

From the analysis of the state-of-the-art, several limitations still present significant obstacles to the fast and efficient detection and robotic picking of fruits. In particular, resolving occlusions caused by leaves and stems—especially when the visible area is minimal—remains a challenge with a relatively low success rate.

As discussed earlier, a promising approach is to combine conventional 2D detection techniques with 3D environment reconstruction, utilizing the additional spatial information provided by point clouds. To achieve this, we propose to integrate the latest YOLOv11 network for fruit detection and classification with Segment Anything Model (SAM)[25] for precise instance segmentation.

By carefully designing the training dataset and object classes, occlusion issues can be addressed concurrently with object detection, leading to higher detection rates.
Additionally, by selecting the fastest available model, YOLO v11 Nano, we aim to achieve a new level of inference speed. These enhancements, combined with the latest advancements in real-time radiance field rendering 3D Gaussian Splattings (3DGS)[24], are expected to significantly improve both accuracy and processing speed in overcoming occlusion-related challenges (Table 2.1).

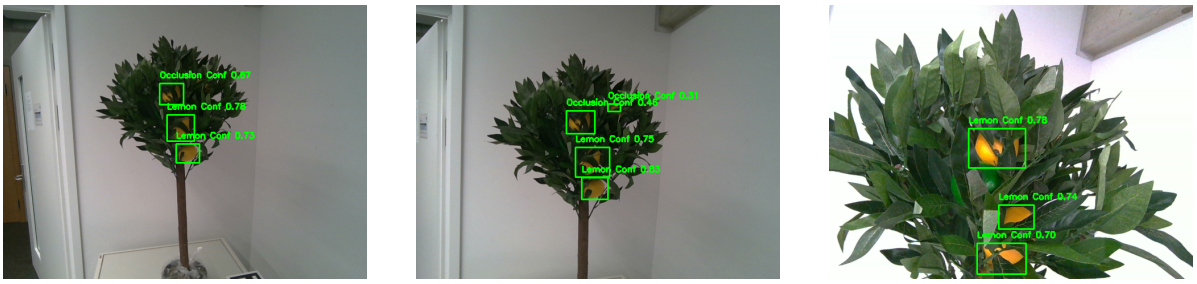
A lemon plant with both yellow and green fruits is chosen as the main benchmark to test this method, challenging the detections of a high variability of colors between the targets. The capability of solving occlusions of this proposed method was tested by choosing a fake plant with a lot of green leaves.

Target technique	**Actual limitations**	**Aimed improvements**	**Proposed solutions**
2D fruit detection	Slow speed	Faster speed	Use lightweight YOLOv11n model
	Sensitive to fruit occlusions	Improved detection rate	Detect both fruits and occlusions with separate classes
3D environment reconstruction	Slow training speed	Faster training speed	Use 3DGS [24] technique
	Imprecise Point Cloud	Improved Point Cloud	Use 3DGS-to-PointCloud [48] technique

Table 2.1: Proposed solutions for the occlusion problem

2.1. Improved YOLOv11 detection

The main limitations of the currently implemented fruit detection solutions are the sensitivity to occlusions and the overall speed.
A different approach is proposed to address the first problem: by detecting both the fruits and the occlusions, as separate entities, a better overall detection rate is expected. In fact, each class has to focus only on one target and the occlusions (2.1a) can be easily solved consequently by zooming in (2.1c) and focusing, navigating around the detected occlusion position (2.1b). A practical example of the proposed workflow is reported in Figure 2.1:



(a) Occluded fruits are detected as occlusions even far from the plant

(b) Occluded fruits start to be correctly detected as fruits

(c) All the occlusions have been solved

Figure 2.1: The working principle behind the proposed 2D occlusions solution: detect both visible and occluded fruits, solving occlusions by changing camera position

.

For the second problem, speed, a tailored selection of the fasted and lightest YOLO model is carried out.
To better analyze the different opportunities and select the best trade-off between the several parameters, a quick recall of the main accuracy and speed metrics for object detection is presented:

- **Intersection over Union (IoU):** Measures the overlap between a predicted bounding box and a ground truth bounding box.
- **Average Precision (AP):** Calculates the area under the precision-recall curve, summarizing the model's precision and recall.
- **Mean Average Precision (mAP):** Averages AP values across multiple object classes for a comprehensive evaluation.

- **Precision (P) and Recall (R):**
  - **Precision:** Proportion of true positives among all positive predictions.
  - **Recall:** Proportion of true positives among all actual positives.
- **F1 Score:** The harmonic mean of precision and recall.

Starting from Precision (P), Recall (R), number of True (T) and False (F) detections, True Positives (TP), False Positives(FP) and False Negatives (FN) several metrics can be computed, as reported in Figure 2.2, including the aforementioned IoU, AP and mAP.

$$P = \frac{TP}{TP + FP} \tag{2.1}$$

$$R = \frac{TP}{TP + FN} \tag{2.2}$$

$$AP = \int_0^1 P(R)\, dR \tag{2.3}$$

$$mAP = \frac{\sum_{i=1}^{n} AP_i}{n} \tag{2.4}$$

$$IoU = \frac{TP}{TP + FP + FN} \tag{2.5}$$

$$MIoU = \frac{\sum_{i=0}^{k} IoU_i}{k + 1} \tag{2.6}$$

$$DE = \frac{\sqrt{(\hat{x}_h - x_h)^2 + (\hat{y}_h - y_h)^2} + \sqrt{(\hat{x}_r - x_r)^2 + (\hat{y}_r - y_r)^2}}{2} \tag{2.7}$$

$$AE = \left| \arctan\left(\frac{y_h - y_r}{x_h - x_r}\right) - \arctan\left(\frac{\hat{y}_h - \hat{y}_r}{\hat{x}_h - \hat{x}_r}\right) \right| \tag{2.8}$$

$$CA = \frac{T}{T + F} \tag{2.9}$$

Figure 2.2: Mathematical formulations for the most used object detection metrics.

In particular, the metrics regarding the bounding boxes are of utmost interests. The most used and most effective at measuring the precision are:

- **Precision (P)**
- **Recall (R)**
- **mAP50:** Mean average precision at an IoU threshold of 0.50.
- **mAP50-95:** Average of mean average precision at varying IoU thresholds (0.50 to 0.95).

The final choice is represented by the YOLOv11-nano detection model, whose performances are compared with previous YOLO models in Figure 2.3 on the COCO (Common Objects in Context) dataset, as reported by the model distributor Ultralytics [52].

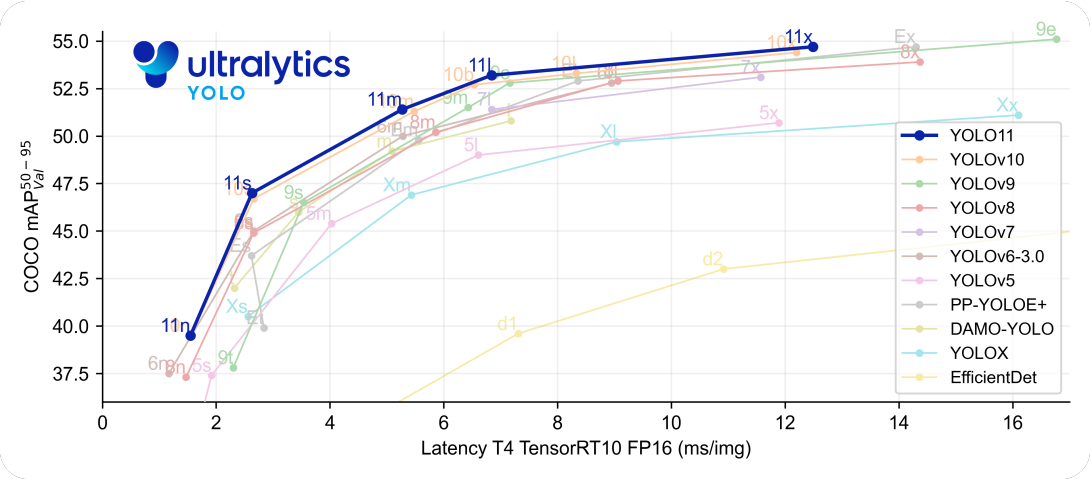


Figure 2.3: Speed-Precision comparison between different YOLO versions

As shown in Table 2.2, the comparison of training speed, inference speed, and precision across different YOLOv11 models highlights that the nano model is the fastest while maintaining a highly acceptable level of precision.

Model	Size (pixels)	mAPval 50-95	Speed CPU ONNX (ms)	Speed T4 TensorRT10 (ms)	Params (M)	FLOPs (B)
YOLO11n	640	39.5	56.1 ± 0.8	1.5 ± 0.0	2.6	6.5
YOLO11s	640	47.0	90.0 ± 1.2	2.5 ± 0.0	9.4	21.5
YOLO11m	640	51.5	183.2 ± 2.0	4.7 ± 0.1	20.1	68.0
YOLO11l	640	53.4	238.6 ± 1.4	6.2 ± 0.1	25.3	86.9
YOLO11x	640	54.7	462.8 ± 6.7	11.3 ± 0.2	56.9	194.9

Table 2.2: Performance comparison of YOLOv11 models

Training a YOLO11 model on a custom dataset involves the following steps:

1. **Preparing the Dataset**: Manually labelling the images, ensuring that the custom dataset follows the YOLO format;
2. **Loading the Model**: Using the Ultralytics YOLO library to load a pre-trained model or create a new model from a YAML configuration file;
3. **Training the Model**: training the model in Python and evaluating the results;

After selecting the most suitable YOLOv11 framework, a well-structured image dataset must be created.

2.1.1. Dataset creation

To construct a functional dataset and manually annotate it, Roboflow [42] was utilized. This online platform simplifies the annotation process by enabling users to draw bounding boxes, assign corresponding labels (Figure 2.4), apply pre-training augmentations and seamlessly convert the dataset into a YOLO-compatible format.
A total of 612 images were selected and labeled, primarily consisting of real-world images of lemon plants. The initial dataset, sourced online, was expanded with additional synthetic images and also images of the benchmark plant, both with and without lemons. This deliberate inclusion of images without targets aims to reduce false positives, particularly those related to color, by training the YOLO network to better distinguish between the environment and the target classes.
The final imposed classes were:

1. **Lemon**: to effectively detect yellow (ripe) lemons;
2. **Lemon_green**: to effectively detect green (unripe) lemons;
3. **Occlusion**: to detect occluded fruits, both ripe and unripe, for the following investigation and occlusions solution;

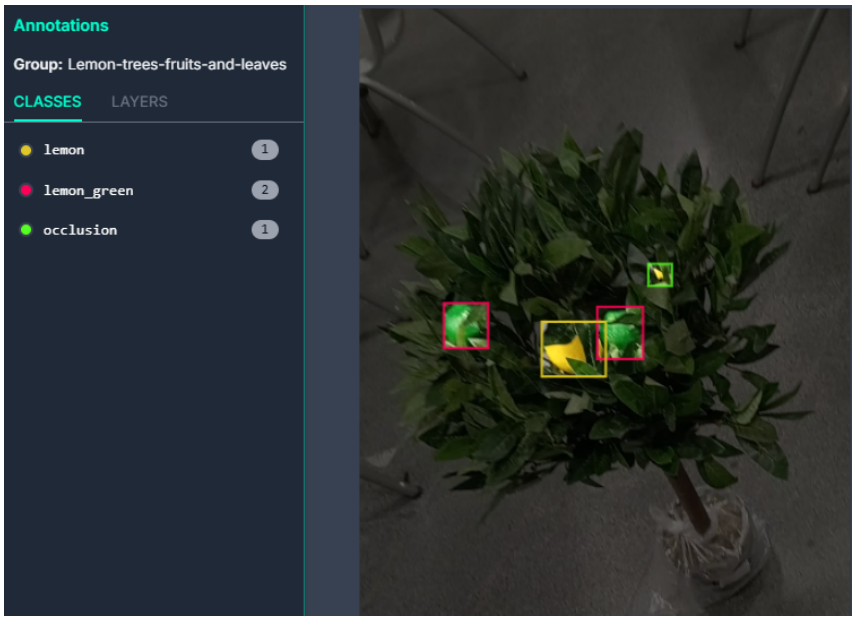


Figure 2.4: A snippet of the Roboflow [42] environment: bounding boxes are manually drawn and assigned to desired class.

Another dataset, containing the exact same images, has been labeled using only the "Lemon" class, which includes exclusively yellow (ripe) lemons, aligning with conventional practices reported in the literature.

This additional dataset serves as a baseline to compare the results of the newly proposed method against standard state-of-the-art approaches.

The next step involves determining the proportions for dividing the dataset into training, validation, and testing subsets. Given the limited number of images, priority was given to ensuring a robust training set. Therefore, a custom split, shown in Figure 2.5, was chosen instead of the conventional 70% training, 20% validation, and 10% testing distribution. Next, pre-processing and augmentation are carried out.

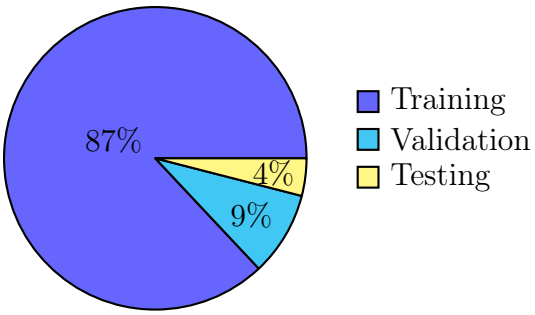
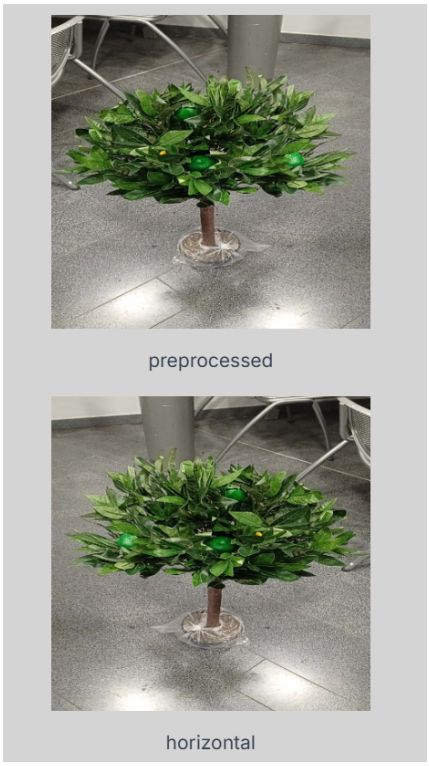


Figure 2.5: Dataset split for training, validation, and testing

(a) Saturation

(b) Rotation



(c) Flip

Figure 2.6: The augmentation procedure

.

Pre-processing procedure consists in:

1. **Auto-Orientation**: Discard EXIF rotations and standardize pixel ordering;
2. **Resize**: Downsize images for smaller file sizes and faster training to a resolution of 640x640 pixels, to be aligned with the standard YOLO training format;

Then, augmentation is carried out to both extend the dataset size and to artificially increase the image variability of dataset, with a particular focus orientation and light conditions. The following steps are thus executed:

1. **Saturation** : a random adjustment of the colors' vibrancy in the images (Figure 2.6a);
2. **Rotation**: a random rotation, between $-15°$ and $15°$, around the center of the image (Figure 2.6b);
3. **Flip**: a full $180°$ rotation around the vertical symmetry axis (Figure 2.6c);

After augmentation the dataset has tripled, reaching a total number of 1460 images.
This total number of images and conditions should be enough to achieve a satisfactory result.

2.1.2. Loading and training the model

The next step is to export the created database in YOLO compatible format.
Using the Robloflow API this can be easily done, both for detection and instance segmentation datasets, obtaining the following structure:

```
yolo-database/
├── test/
│   ├── images/
│   └── labels/
├── train/
│   ├── images/
│   └── labels/
├── valid/
│   ├── images/
│   └── labels/
└── data.yaml
```

The last remaining step is to train the model itself. To do so a Python code has been developed, leveraging the Ultralytics [52] Python package. The process begins with the

pre-trained “yolo11n.pt“ weights for object detection and “yolo11n-seg.pt“ for instance segmentation. The model is then fine-tuned on a custom dataset using specified training parameters, such as batch size, image size and number of epochs.

To optimize GPU memory usage, the model's memory is managed both before loading and after training, with a thorough cleanup to free up GPU resources. Additionally, a “patience“ mechanism is enabled: if the model stops improving and reaches a plateau for a predefined number of epochs (set by the “patience“ parameter), the training process is automatically stopped. Considering the specific set of training parameters and the hardware on which the training took place, the training time for both tasks is saved.

To sum up all the imposed training parameters with their description, Table 2.3 is reported.

Parameter	**Type**	**Description**
`Patience`	Int	Number of epochs to wait without improvement in validation metrics before early stopping.
`Epochs`	Int	Total number of training runs. Each epoch represents a full pass over the entire dataset.
`Batch`	Int	GPU utilization: set as an integer (e.g., `batch=16`), auto mode for 60% GPU memory utilization (`batch=-1`), or auto mode with a specified utilization fraction (`batch=0.70`).
`Image Size`	Int or List	Target image size for training: all images are resized to this dimension before being fed into the model.

Table 2.3: Training parameters and their descriptions

The training parameters for the proposed model, with the occlusion class, are shown in Table 2.4, while the training parameters for the standard model, without the occlusion class, are shown in Table 2.5.

Parameter	Detection	Segmentation
Patience	10	10
Epochs	100	100
Image Size	640x640	640x640
Batch Size	2	2
GPU Used	NVIDIA GeForce RTX 3060 , 6.144 GB	NVIDIA GeForce RTX 3060 , 6.144 GB
Training Time	**1 h 7 min**	**2 h 55 min**

Table 2.4: Training parameters for detection and segmentation of the proposed model with occlusion class

Parameter	Detection	Segmentation
Patience	10	10
Epochs	100	100
Image Size	640x640	640x640
Batch Size	2	2
GPU Used	NVIDIA GeForce RTX 3060 , 6.144 GB	NVIDIA GeForce RTX 3060 , 6.144 GB
Training Time	**39 min**	**1 h 19 min**

Table 2.5: Training parameters for detection and segmentation of the proposed model with occlusion class

2.1.3. Training, valuation and test results

Once the training is complete, several metrics can be extracted.
So the most common metrics are respectively reported in:

- **Figure 2.7** for the proposed detection model with the occlusion class;
- **Figure 2.8** for the proposed segmentation model with the occlusion class;
- **Figure 2.9** for the standard detection model without the occlusion class;
- **Figure 2.10** for the standard segmentation model without the occlusion class;

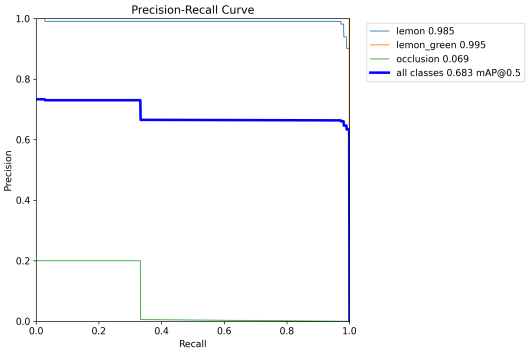
Several insightful metrics can be extracted also from the model validation process. During validation the model predictions are confronted with ground-truth annotated images coming from the "**valid**" dataset, obtaining the metrics reported in Table 2.6 for detection and in Table 2.7 for segmentation:

Model	Class	Images	Instances	Box(P)	R	mAP50	mAP50-95
Proposed	All	100	158	0.676	0.775	0.683	0.623
	Lemon	76	110	0.940	0.991	0.985	0.859
	Lemon Green	45	45	0.980	1.000	0.995	0.994
	Occlusion	3	3	0.109	0.333	0.0688	0.0148
Standard	All	55	89	1.000	0.964	0.993	0.804

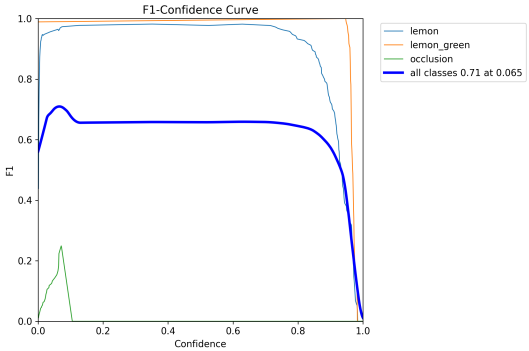
Table 2.6: Validation results for proposed and standard detection models

Model	Class	Images	Instances	Box				Mask			
				P	R	mAP50	mAP50-95	P	R	mAP50	mAP50-95
Proposed	All	100	158	0.990	0.663	0.672	0.624	0.990	0.663	0.672	0.617
	Lemon	76	110	0.982	0.988	0.987	0.870	0.982	0.988	0.987	0.850
	Lemon Green	45	45	0.987	1.000	0.995	0.989	0.987	1.000	0.995	0.991
	Occlusion	3	3	1.000	0.000	0.0345	0.0146	1.000	0.000	0.0345	0.00999
Standard	All	55	89	1.000	0.960	0.993	0.841	1.000	0.960	0.993	0.817

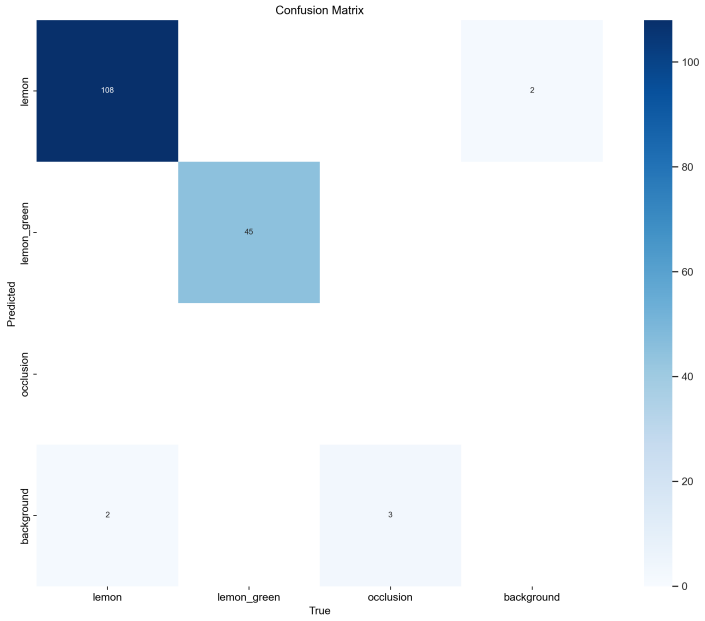
Table 2.7: Validation results for proposed and standard segmentation models



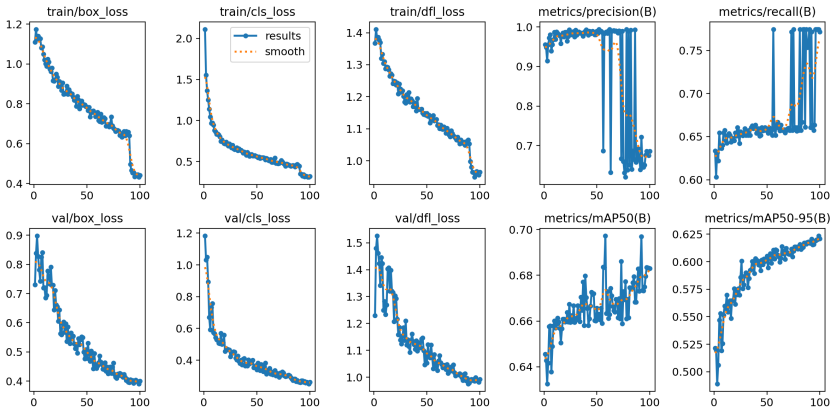
(a) Precision–Recall curve



(b) F1–Confidence curve



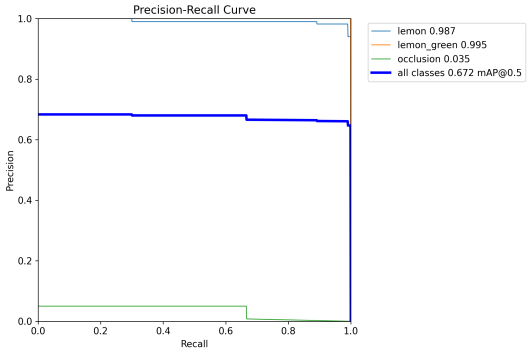
(c) Confusion Matrix



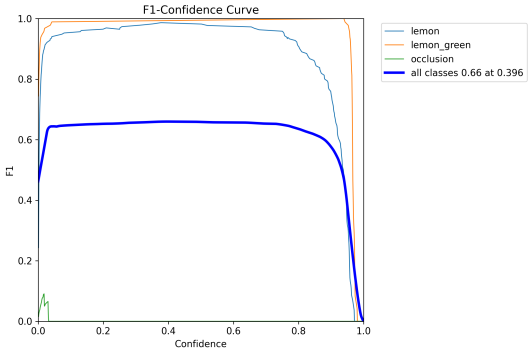
(d) Training metrics over training epochs

Figure 2.7: Training metrics for the detection model with the occlusion class

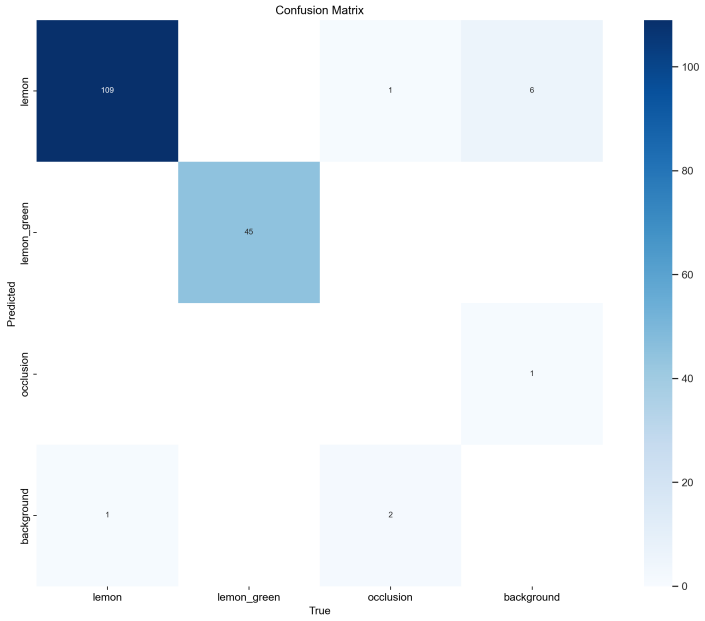
.



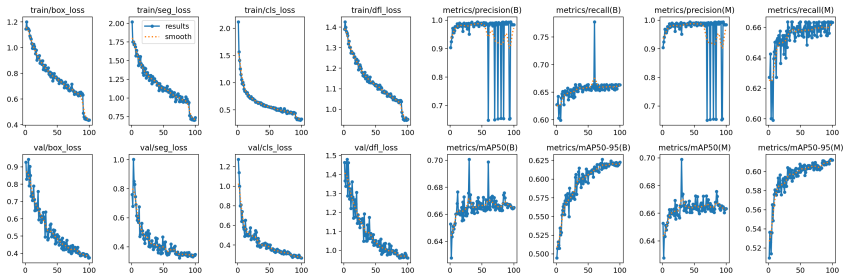
(a) Precision–Recall curve



(b) F1–Confidence curve



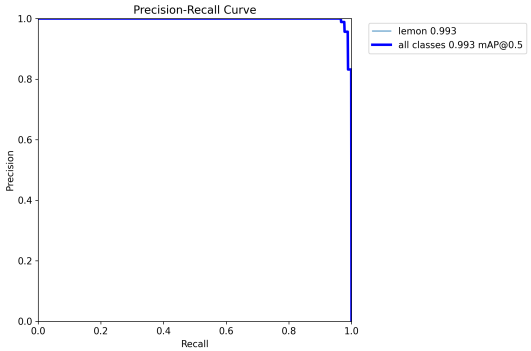
(c) Confusion Matrix



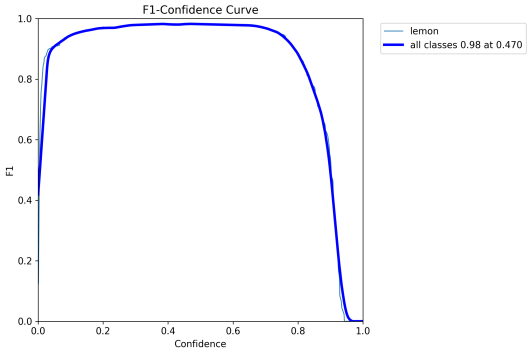
(d) Training metrics over training epochs

Figure 2.8: Training metrics for the segmentation model with the occlusion class

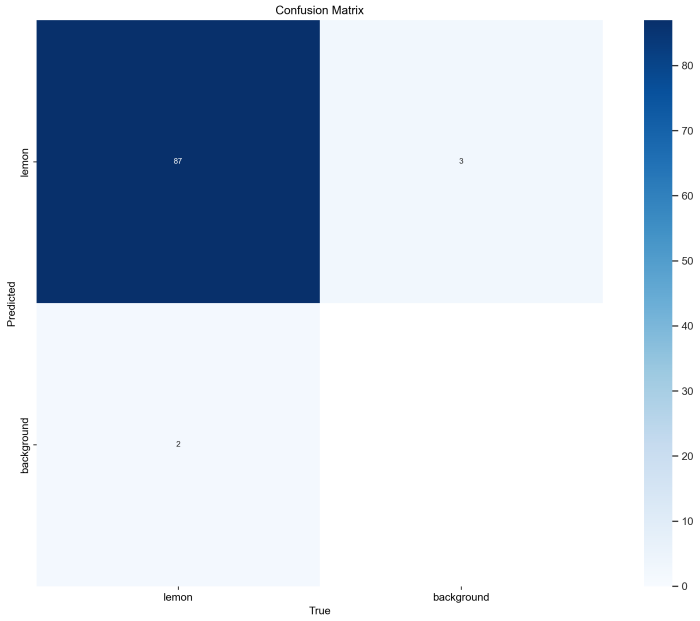
.



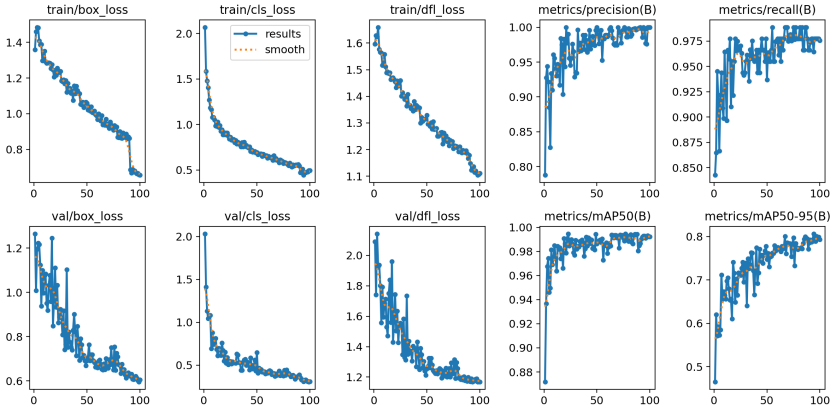
(a) Precision–Recall curve



(b) F1–Confidence curve



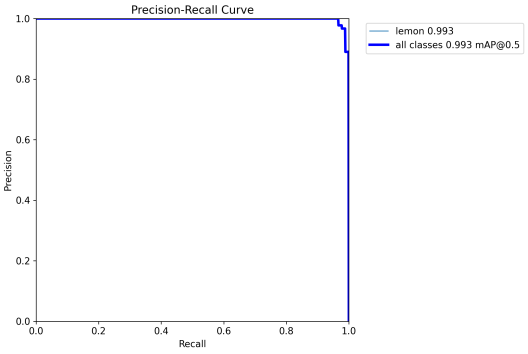
(c) Confusion Matrix



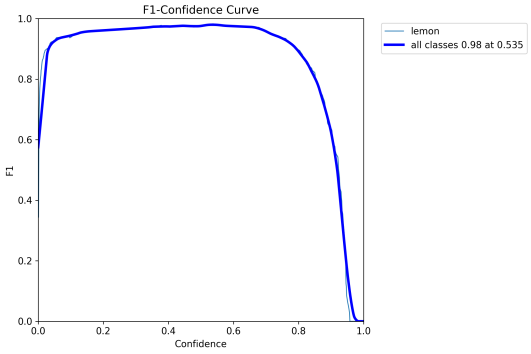
(d) Training metrics over training epochs

Figure 2.9: Training metrics for the detection model without the occlusion class

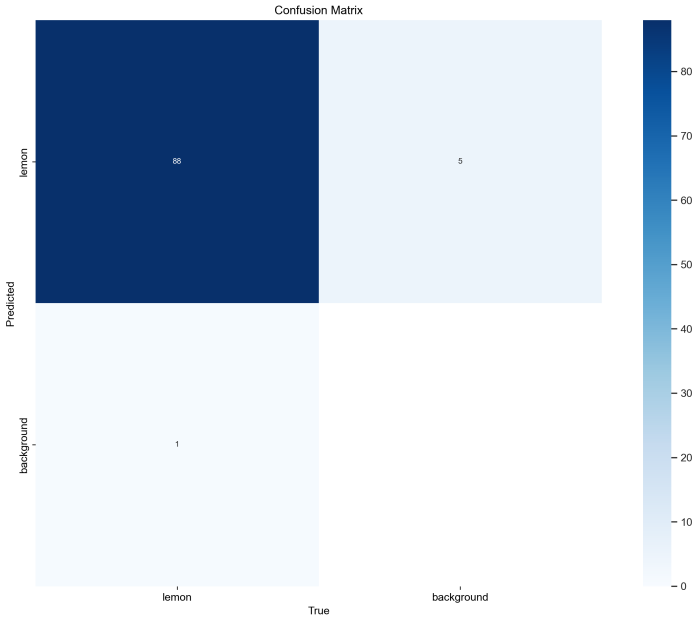
.



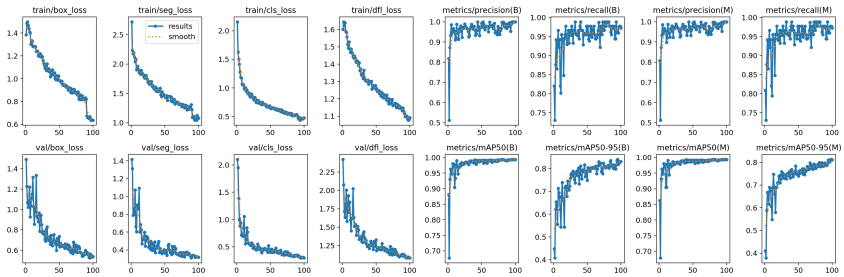
(a) Precision–Recall curve



(b) F1–Confidence curve



(c) Confusion Matrix



(d) Training metrics over training epochs

Figure 2.10: Training metrics for the segmentation model without the occlusion class

.

2.2. Improved 3D environment reconstruction

In the recent years NeRF(1.5.2) has been recognized as the best and utmost recent technique for 3D point cloud reconstruction of a picking enviroment starting from standard 2D RGB images.
As summarizied in Table 2.1, the main limitations of this currently implemented 3D environment reconstruction are the training speed and the computed point cloud precision.

As highlighted by the numerous reviews present in the literature[51, 68] more recently, a strong opponent has risen to challenge NeRF in both training speed and accuracy: 3D Gaussian Splattings (3DGS) [24].

This technique presents a novel approach to achieving real-time, high-quality novel-view synthesis using 3D Gaussian representations. The optimization process begins with a sparse SfM point cloud, from which a set of 3D Gaussians is generated. These Gaussians are then refined through optimization and adaptive density control.
A fast tile-based renderer is used during training, ensuring competitive training times compared to state-of-the-art radiance field methods. Once trained, the renderer enables real-time navigation across diverse scenes and to export a dense point cloud.
A very effective representation of the 3DGS workflow is shown in Figure 2.11:

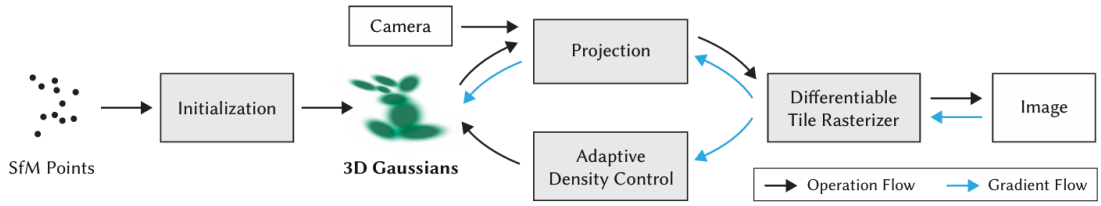


Figure 2.11: The 3DGS workflow [24]

Another important step has been achieved by Stuart and Pound [48] that developed an open source algorithm to export colored dense point clouds and even meshes from a 3DGs representation.

By leveraging these two great achievements, it is possible to improve the performance of the selected 3D reconstruction algorithm also for a picking application.
The proposed pipeline for the 3D reconstruction of the picking environment is thus composed by several open source algorithms, taking as input a simple RGB video and giving as output the colored point cloud of the picking environment, as shown in (Figure 2.12):

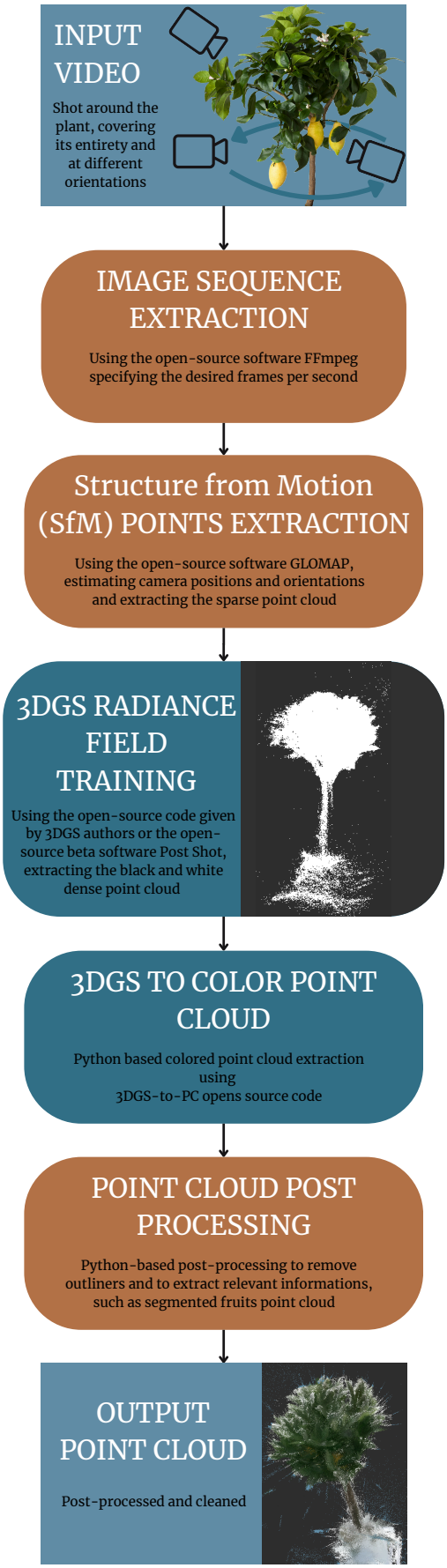


Figure 2.12: The proposed 3D reconstruction workflow

In particular:

1. **Image sequence extraction**: The image sequence is automatically extracted from the input video at the desired frame rate using the open-source software FFmpeg. Additional post-processing is performed if needed.

2. **SfM points extraction**: The extracted image sequence is processed using the open-source SfM software Glomap [37], a globalized version of the Colmap software. It estimates the camera positions and orientations, which are saved in the `cameras.bin` file, while the sparse point cloud is stored in the `points3D.bin` file. The final Glomap directory structure is as follows:

```
Glomap-database/
├── images/
│   ├── image_1
│   ├── image_2
│   └── ...
├── sparse/
│   └── 0/
│       ├── cameras.bin
│       ├── images.bin
│       └── points3D.bin
└── database.db
```

3. **3DGS radiance field training**: the radiance field is trained according to the open-source code provided by Kerbl et al. [24] or the open-source beta software "Post Shot" by Jawset(`https://www.jawset.com/`).
   The former allows to set very training parameter and to save the checkpoints during training, being the default number of iteration set to 30k itarations.
   The latter is easy to install and easier to use, giving also the possibility to post-process manually the computed point cloud. For both options, the resulting point cloud is given in black and white format.

4. **3DGS to pointcloud**: using the python-based code created by Stuart and Pound [48] is possible to densify the generated point cloud and to retrieve the color information for each point starting from the training image dataset.

5. **Point cloud post-processing**: additional post-processing is carried out according to the particular necessities, such as outliners removal and point cloud segmentation.

3 | Simulation

To evaluate the proposed methodologies, a simulated version of the workflow is implemented. This simulation replicates both the picking and detection routines of the robot intended for real-world experiments, ensuring a high degree of realism.

3.1. Building the simulated picking environment

The simulated picking environment is developed in Gazebo (`https://gazebosim.org/`), leveraging the advanced realism and simulation capabilities of Robot Operating System (ROS) (`https://www.ros.org`).

To construct a comprehensive and effective picking environment that addresses all key aspects of the proposed methodologies, the following requirements are established:

- **Occluded Fruits**: The simulated environment must accurately represent a realistic fruit distribution scenario, including both occluded and exposed fruits. Their arrangement on the plant should closely resemble real-world conditions. Furthermore, to maintain consistency between the simulated and physical environments, the lemon fruits must be modeled with a high level of realism.
- **Realistic Environment Physics**: The physical interactions within the simulated environment must be as realistic as possible, particularly in handling different collision outcomes between the robot and surrounding objects. Specifically, distinctions must be made between collisions involving the rigid tree trunk, deformable leaves and small branches, and the pickable fruits.
- **Robotic Arm Integration**: A robotic arm is required to simulate both the detection and picking tasks. To ensure high fidelity between the simulation and real-world experiments, the same robotic arm used in physical tests must be employed in the simulation. Additionally, the operational parameters in both environments should be identical.

- **Gripping Capability**: A gripper is necessary for the robotic arm's end-effector to facilitate the picking process. To maintain consistency, the same gripper used in real-world experiments must be replicated in the simulation.
- **Depth Camera Capability**: To enable object detection and position estimation for picking, a depth camera is required. To ensure coherence with the physical setup, a camera with similar specifications to the one used in real-world testing must be integrated into the simulation.
- **Customizable Python-Based Control**: The robot's control system should preferably be implemented in Python to ensure seamless integration with the Python-based detection algorithm. Furthermore, a high degree of customization must be supported, allowing for flexible selection of the motion planner.

To meet these requirements, specific implementation choices are made. The following sections provide a detailed explanation of each choice, outlining the solutions proposed for each requirement.

3.2. Picking environment generation

The first requirement to be addressed is the generation of the picking environment, focusing in particular on the generation of the tree and of the fruits.
To add an element in a custom Gazebo simulated environment, several parameters have to be set. Firstly, the object appearance and 3D geometry have to be set by adding its mesh to the `world.launch` Gazebo file. Several mesh formats are accepted, mainly `.dae` and `.stl`, given that the former is able to export both geometry and material properties in Gazebo (such as color and texture).

To achieve a realistic physical behavior of the tree and the fruits, each component's collision and physical properties have to be specified. In particular, by enabling the collision property the interactions between the selected object and the rest of the collision-enabled objects will be considered. By disabling it, these interactions will be neglected and the selected object could be penetrated by other rigid bodies.
The following collision properties have to be assigned to each particular element to mimic their real physical behavior (Table 3.1):

Element	Physical behavior	Implemented collision policy
Tree trunk	Hard and stiff	Assumed as an obstacle, collision is enabled
Leaves and small branches	Soft and compliant	Can be crossed, collision is disabled
Fruits	Hard and stiff	Assumed as the picking target, collision is enabled

Table 3.1: Assigned collision properties for each type of object

The original tree mesh, sourced from an open-access online repository, has been modified by separating it into two distinct components: the tree trunk, which retains the designated collision properties, and the tree canopy, where collision detection is disabled.

Additionally, the mesh has been resized to a more appropriate scale and refined in terms of color and texture to enhance realism.

Subsequently, the positioning, appearance, and collision properties of the fruits are defined. Given that the simulation focuses on the robotic picking of lemons, the fruits are represented as yellow spheres with a diameter of 10 cm. As previously mentioned, collision detection is enabled for the fruits, ensuring rigid interaction between the robotic gripper and the fruit, which is essential for successful picking.

Finally, the fruits placement and their attachment to the tree canopy are automated using a dedicated Python script. This script models the tree canopy as the union of a hollow cylinder and a hollow hemisphere, with their respective properties outlined in Table 3.2. It then procedurally generates the positions of the specified number of lemons.

Property	Lower hollow cylinder	Upper hollow sphere
Internal radius	37 cm	37 cm
External radius	40 cm	40 cm
Height	15 cm	–
Number of lemons	12	30

Table 3.2: Tree canopy modeling parameters

Consequently, the code automatically assembles the `fruit_picking_world.world` file that gathers the tree, both trunk and canopy with their specific properties, and the randomly generated lemons.
It is worthy noticing that by modifying the internal and external radius parameters it is possible to simulate different levels of occlusions since the lemons will be more inside or outside the tree canopy mesh.

The resulting lemon tree at increasing levels of occlusion is shown in Figure 3.1:

(a) Low level of occlusions

(b) Intermediate level of occlusions

(c) High level of occlusions

Figure 3.1: Resulting lemon tree in Gazebo at different imposed levels of fruits occlusion

.

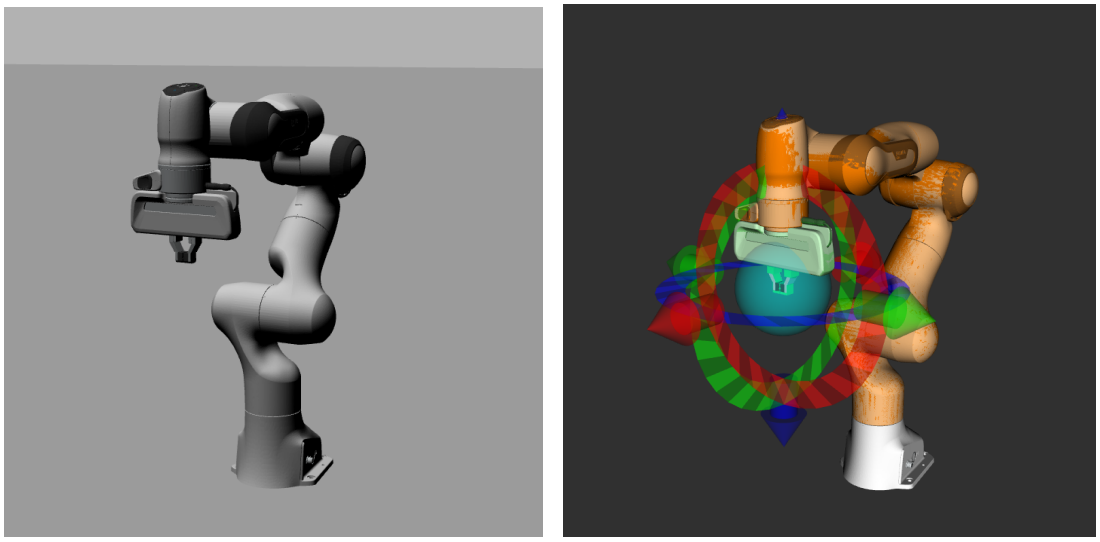
The grid and the reference systems are omitted in these images for a better look, but the tree is placed in the origin of the simulated Gazebo world.

3.3. Robot simulation and control

Next, the simulated robotic arm is integrated into the picking environment. To ensure maximum coherence with real-world testing, the `Emika Franka Panda` (`https://franka.de/`) robot is selected.

The simulation utilizes the open-source package `franka_description`, which provides all necessary informations, including the robot's geometric and kinematic properties. The essential details regarding the robot's structure are encapsulated in its `Unified Robotic Description Format (URDF)` file, which defines the robot's links along with their geometric and inertial properties, as well as the joints that establish kinematic connections between them.

The simulated environment is launched simultaneously in Gazebo and RViz (`http://wiki.ros.org/RViz`), allowing real-time visualization of the robot within the picking environment. Moreover, these tools enable comprehensive monitoring of the robot's state and grant access to the current planning scene. Figure 3.2 showcases the graphical user interface (GUI) outputs of these software tools, featuring the standard Franka Panda robot model.



(a) Gazebo GUI (b) RViz GUI

Figure 3.2: Franka panda robot in simulation, as imported from `franka_description`

To control the robot in the Gazebo, using a Python-based code as stated in the requirements, `MoveIt` and `franka_control` libraries are used.
`MoveIt` links the robot model, expressed by the "`.urdf`" file, with Ros, Gazebo and RViz

for easy visualization and simulation.
`Franka_control` is an open-source library that provides an effective low-level controller for the Franka Robots family.

The "`Move Group Python Interface`" is an extension of MoveIt that integrates the control of robots, both for real and simulated contexts, with a Python-accessible interface. Through this Python interface it is possible to command the robot through `franka_control`, accessing real-time informations coming from the robot itself.
The high-level command framework is represented by the `move_group node`, which is sum up in Figure 3.3:

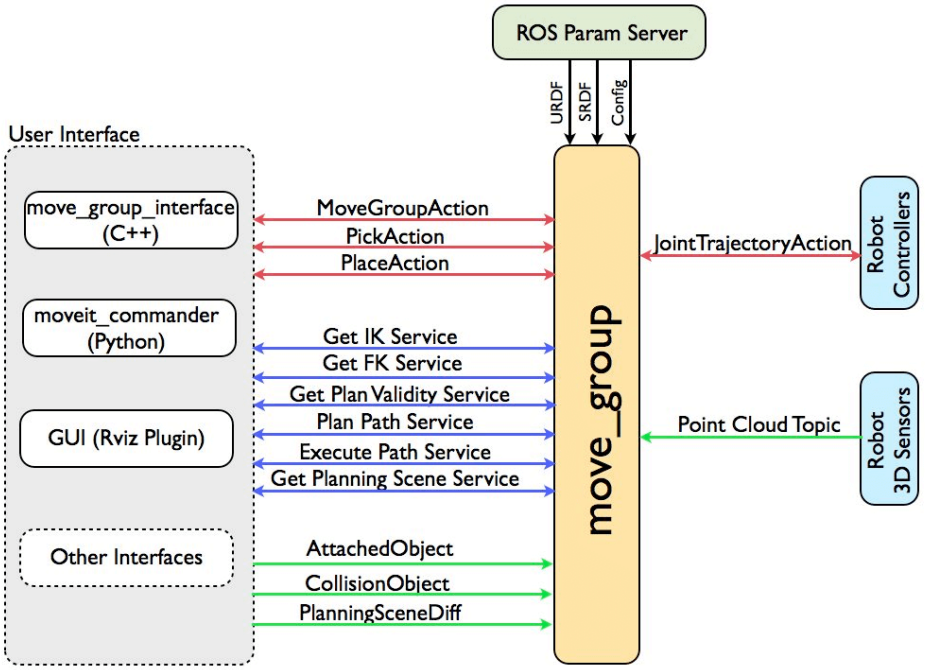


Figure 3.3: The `move_group node` high-level control scheme

.

The ability to command the robot groups in Python, particularly the robotic arm (`panda_arm` or `panda_manipulator`) and the gripper (`panda_hand`), provides the necessary level of customization while leveraging a well-established and actively maintained framework. The integration with the object detection and localization algorithms is thus fully achieved in the same coding language, granting the implementation of real–time pipelines.

Another significant advantage of this solution is the ability to integrate some of the most advanced trajectory planning algorithms referenced in the literature (Chapter 1.4).
Indeed, MoveIt allows trajectory planning using a diverse selection of open-source libraries,

such as:

- **Open Motion Planning Library (OMPL)**: an open-source library for sampling-based motion planning. These algorithms are probabilistically complete, meaning they will find a solution if one exists but cannot confirm its non-existence. Efficient and usually fast.
- **Covariant Hamiltonian Optimization for Motion Planning (CHOMP)**: Optimizes an initial trajectory by pulling it out of collisions but often produces jerky paths. Obstacle avoidance comes at the cost of velocity smoothness, while the `ridge_factor` must be ≥ 0.001 to avoid obstacles.
- **Stochastic Trajectory Optimization for Motion Planning (STOMP)**: Generates smooth, collision-free paths using noisy trajectories to explore alternatives and refine the initial path.
- **Pilz Industrial Motion Planner**: is a trajectory generator for planning standard robot motions, using the `MoveIt PlannerManager` plugin interface. It functions solely as a motion generator and does not account for obstacle avoidance. The intended trajectory is computed first and then checked for collisions: if a collision is detected, the entire trajectory is discarded.

The particular sets of algorithms available for each library are listed here:

- **Open Motion Planning Library (OMPL)**:
  - RRT*
  - PRM*
  - LazyPRM*
  - BFMT
  - FMT
  - Lower Bound Tree RRT (LBTRRT)
  - SPARS
  - SPARS2
  - Transition-based RRT (T-RRT)
  - Adaptively Informed Tree (AIT*)
  - Batched Informed Tree (BIT*)

  - Advanced Batched Informed Tree (ABIT*)

- **Covariant Hamiltonian optimization for motion planning (CHOMP)**:
  - CHOMP
- **Stochastic Trajectory Optimization for Motion Planning (STOMP)**:
  - STOMP
- **Pilz Industrial Motion Planner**:
  - Point-To-Point (PTP)
  - Linear Cartesian trajectory (LIN)
  - Circular arc Cartesian trajectory (CIRC)

Several algorithms performances are tested, and the final choices for the trajectory planning of the various robot groups are:

Planning Group	**Library**	**Chosen motion planner**
`panda_arm`	Pilz	PTP
`panda_manipulator`	Pilz	PTP
`panda_hand`	Franka_gripper	Cartesian

Table 3.3: Trajectory planners choices for each planning group

Using these planners and the full `Move Group Python Interface` capabilities, two main callable python functions are implemented:

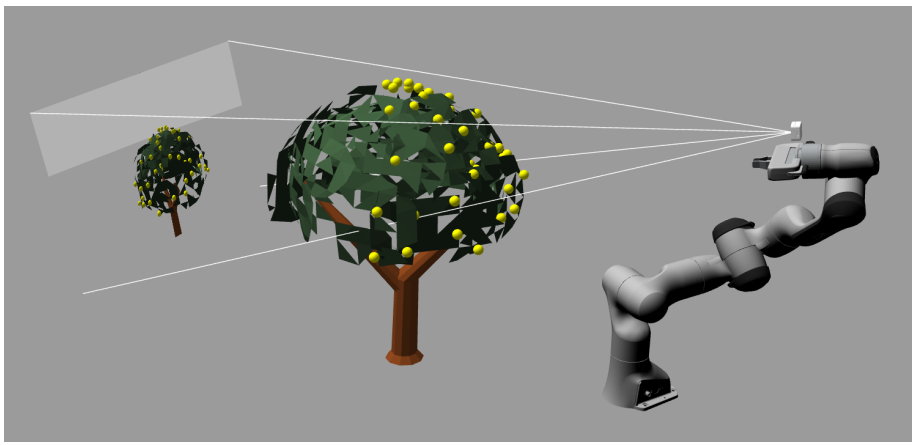
- **MoveArm**: takes as input the desired pose (x,y and z position plus a quaternion for orientation) of the end-effector and subsequently plan and execute a collision-free trajectory, starting from the current pose.
- **MoveHand**: takes as input the desired value of gripper's fingers positions, maximum force and speed, planning and executing the resulting trajectory, starting from the current opening values.

These two functions can be organically called in the main code of the detection and picking pipelines, granting both the control of the robotic arm and gripping capabilities.

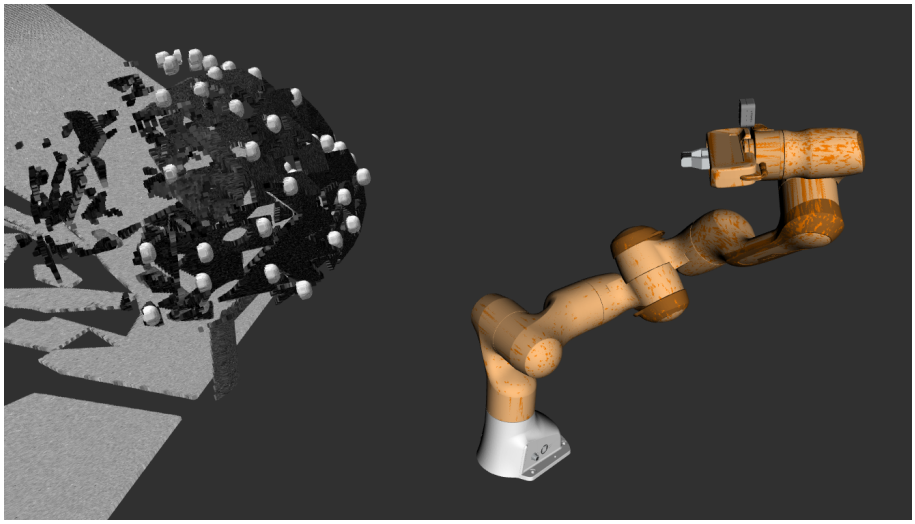
3.4. RealSense camera simulation

The following implemented feature is the integration of the depth camera.
To keep the highest fidelity, the same camera used in the real experiments is simulated. In particular, the `Intel Realsense d405` depth–stereo camera is chosen and simulated through the `realsense2_description` opens source package.

The camera model, though its geometry, inertia and collision properties, is added to the simulation, while the simulated images and depth streams are obtained through the ROS `librealsense_gazebo_plugin`. This integration with Gazeboand RViz (Figure 3.4) allows also to show the Point Cloud computed using the data coming from RGB and depth streams together (Figure 3.4b).



(a) Gazebo camera integration, the RGB stream is visualized



(b) RViz camera integration, the computed point cloud is visualized

Figure 3.4: The simulated output of the d405 realsense camera

Its outputs, with their particular settings and parameters, are summarized in Table 3.4:

Parameter	RGB stream	Depth stream
`Width`	1280	1280
`Height`	720	720
`frame per seconds(fps)`	90	90
`Data stream format`	RGB_INT8	L_INT8

Table 3.4: RGB and depth streams main properties and parameters

where:

- **RGB_INT8**: is a color image format with 8-bit depth per channel.
- **L_INT8**: is a grayscale image format with a single 8-bit intensity channel.

`rgb_callback`, `depth_callback` and `camerainfo_callback` callable functions are implemented to retrieve directly in python the RGB, depth and camera parameters topics.
An additional python code is implemented to compute the color point cloud, exploiting both depth and RGB streams.The code subscribes to multiple ROS topics to acquire synchronized depth and color images, as well as camera intrinsic parameters and an externally computed point cloud from RViz.

After acquiring the data streams through ROS topics, the code listens to depth images, RGB images, and intrinsic camera parameters, ensuring proper alignment between depth and RGB streams. The goal is to construct an accurate colored point cloud from the depth and RGB images, leveraging the simple pinhole model and using the camera intrinsic parameters. These parameters are typically represented by a 3x3 matrix, often denoted as K, and include:

- **Focal lengths** (f_x, f_y): These values define the scaling factors that relate pixel coordinates to real-world distances in the image. The focal length in the x-direction (f_x) and y-direction (f_y) determine how much the camera zooms in each axis. They are usually the same for most cameras but can vary slightly due to lens distortion.
- **Principal point** (c_x, c_y): These represent the coordinates of the optical center (or "center of projection") in the image plane. In an ideal camera, this is typically at the center of the image, but it can shift due to misalignment or sensor characteristics.

The intrinsic matrix K is structured as:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

Where the values f_x and f_y control the scale of the image, and c_x and c_y specify the position of the optical center. These parameters are essential for converting 2D pixel coordinates back into 3D real-world coordinates using depth information.
The conversion from depth to 3D space follows the pinhole camera model, where each pixel's depth value is used to compute its corresponding real-world coordinates (X, Y, Z) using the intrinsic parameters and pixel position in the camera reference system (u, v):

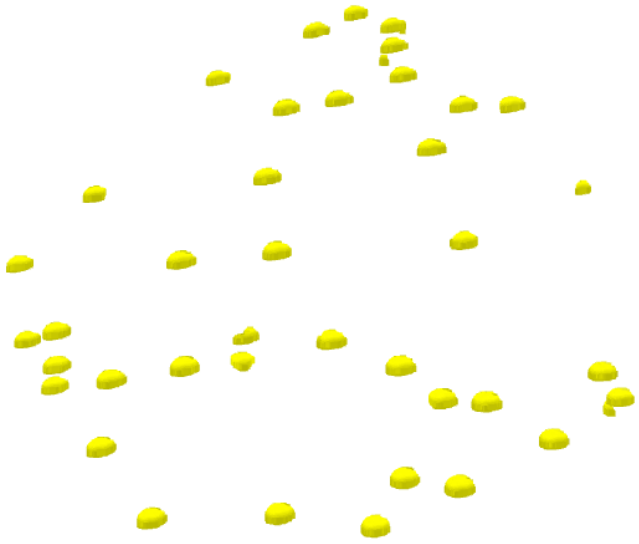
$$X = \frac{(u - c_x) \cdot Z}{f_x}, \quad Y = \frac{(v - c_y) \cdot Z}{f_y}, \quad Z = \text{depth}(u, v) \tag{3.2}$$

Simultaneously, each pixel's RGB value is mapped to its respective 3D point, producing a dense colored point cloud. To ensure data quality, invalid points (e.g., those with zero depth or exceeding a threshold of 3 meters) are filtered out.

Beyond computing the point cloud from depth data, the system also processes the point cloud received from RViz in the `Point2` topic, extracting its spatial coordinates and color attributes. Both point clouds are then filtered and stored in the `PLY format` for further analysis. As last step, the lemon are segmented using a color based clusterization, selecting points in the proper `HSV format` range.

Finally, visualization is handled using `Open3D`, allowing an interactive comparison between the computed and received point clouds. The final results are shown in Figure 3.5:

(a) Reconstructed colored point cloud



(b) Segmented lemons from reconstructed point cloud

Figure 3.5: The reconstructed point clouds from simulated RGBd realsense camera

3.5. Simulated Environment 3D reconstruction

The same pipeline, described in 2.2, is carried out also in the Gazebo simulated environment, to reconstruct the 3D environment, with particular interest for the tree and its fruits.
A `.mp4` video is recorded by the simulated camera, while the robot navigates around the plant. After this, the video is saved and exported to Windows for the following steps of the proposed 3D reconstruction pipeline. The following results are obtained:

Figure 3.6: Coarsely filtered colored 3DGS point cloud

Figure 3.7: Finely filtered colored 3DGS point cloud

Figure 3.8: Segmented lemons from colored 3DGS point cloud

3.6. Simulated Detection Routine

The last step was to test the effectiveness of the proposed detection pipeline.
To do so the robot has to explore the surroundings of the tree, covering various points of view.

Specifically, a circular trajectory around the plant, with the camera always pointed at the plant, is considered the best strategy. So, a Python–based detection routine is designed, covering several circular waypoints around the target, ensuring that, for each one of them, two rounds of inference are carried out.
This redundant feature is added to minimize the possible oscillation of YOLO detections during inference, given some small residual robot movements or even just small delays between the different part of the simulated system. Following this strategy, several `rospy(sleep)` checkpoints are insered, allowing the Python code and the ROS-dependent features to synchronize themselves, in order to avoid delays accumulations and communication errors.

The list of circular waypoints is created by imposing an equidistanced list of points, linking the x and y coordinates to a circumference of set radius and center (the estimated target center), while the z coordinates are imposed at a desired fixed height (Table 3.5). Starting from these values of position, also the desired end-effector orientation is computed for each waypoint.

Parameter	Number of waypoints	Radius	Height	Center
`Value`	5	90 cm	60 cm	[10 cm, -20 cm, 40 cm]

Table 3.5: Circular Trajectory parameters

As stated before, the camera mounted on the end effector has to always point at the plant, assuring the best image coverage of the target.
To do so a custom python function is created, leveraging the open source python module `Rotation`, from `scipy spatial transform` package.
The code computes the desired normal vector of the camera, starting from the camera position and ending in the center of the target. This vector is then used to compute the necessary rotation, expressed in quaternions, needed to align the z-axis of the camera to the target-pointing vector. This quaternion is eventually set as the orientation value of the desired end-effector pose. The overall structure of this detection-finalized circular trajectory is described in Figure 3.9:

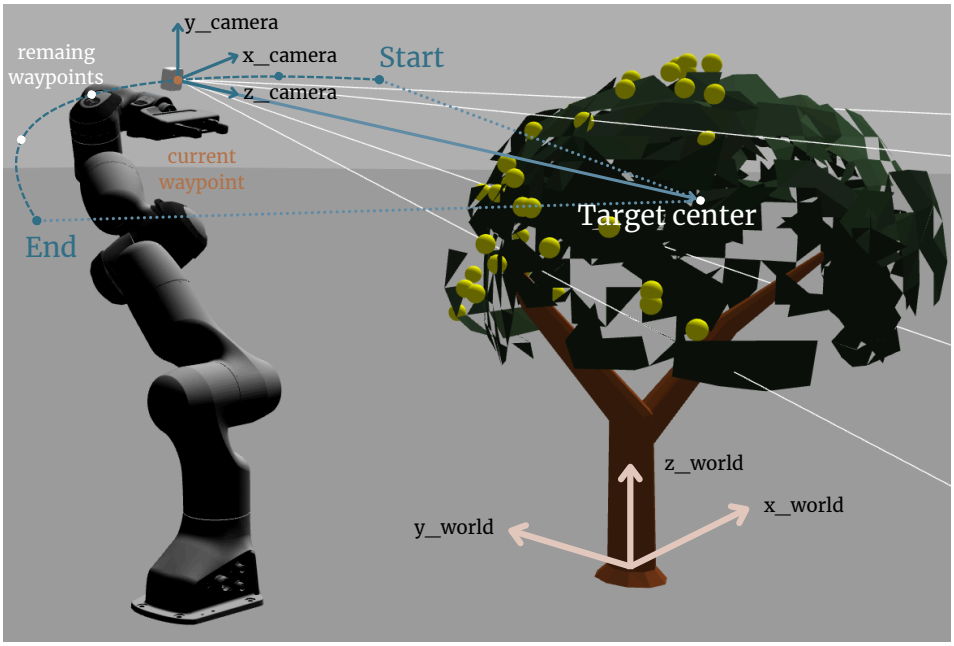


Figure 3.9: The circular trajectory used in the detection pipeline

This planned trajectory is thus executed using the aforementioned `MoveArm` function, taking the necessary buffer time to execute and save the detection results in a custom folder structure:

```
results_{date}_{time}/
├── detection/
│   ├── waypoint_1
│   │   ├── detections.png
│   │   └── detections_canva.png
│   ├── waypoint_2
│   ├── ...
│   └── waypoint_n
├── inference_time/
│   └── inference_time.txt
└── detections_with_positions.json
```

In fact, for each waypoint, the inference times are computed and saved for further evaluation.
The visual results of the inference process are saved in two `.png` images, the former (`detections.png`) reporting the detections with their local and global coordinates over the original RGB image, while the latter (`detections_canva.png`) adds also the detections over the depth image and the masked images. In the `detections_with_positions.json`

the detections are saved in a structured way, linking the information of both label and global position of the detected instance. This structured file is created contextually with the computation of the global cooridates of the detections, in the `TargetPositionsSim` callable python function.

This function estimates the global position of the detected instance, specifically of the center of its YOLO bounding box, by exploiting the depth information and the relative link between the camera and the robot, passing from the transformation that links the robot end effect to the world coordinates (Figure 3.10):

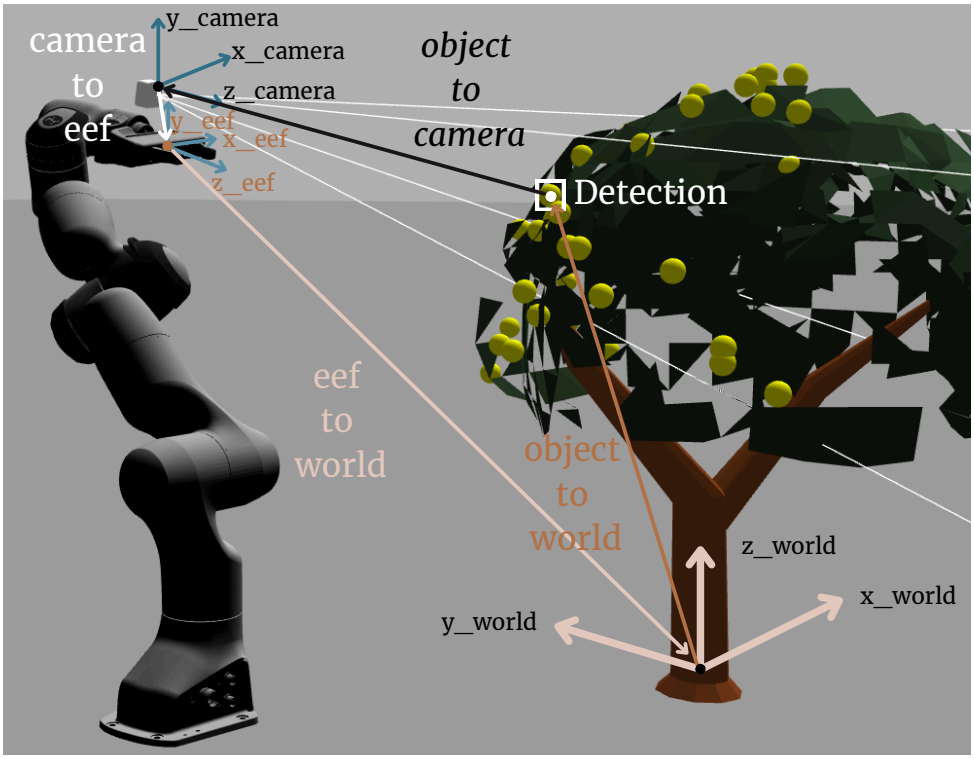


Figure 3.10: The transformations to retrieve coordinates of the target in the global reference system

Starting from the selected pixel position and depth in the image $(u, v, depth)$, the first transformation produces $(x_{camera}, y_{camera}, z_{camera})$ coordinates in the camera reference system, using the pinhole camera model (Eq. 3.2):

$$\begin{bmatrix} x_{camera} \\ y_{camera} \\ z_{camera} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{f_x} & 0 & -\frac{c_x}{f_x} & 0 \\ 0 & \frac{1}{f_y} & -\frac{c_y}{f_y} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ depth \\ 1 \end{bmatrix} \tag{3.3}$$

It can be rewritten as:

$$\begin{bmatrix} x_{camera} \\ y_{camera} \\ z_{camera} \\ 1 \end{bmatrix} = \begin{bmatrix} R_{obj\_to\_camera} & T_{obj\_to\_camera} \\ \underset{1\times3}{\mathbf{0}} & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ depth \\ 1 \end{bmatrix} \tag{3.4}$$

defining the equivalent rotation and translation matrices $R_{obj\_to\_camera}$ and $T_{obj\_to\_camera}$ that link the object pixel and depth coordinates with their transformed in the camera reference system.

The intrinsic camera parameters (f_x,f_y,c_x and c_y) are streamed by the simulated realsense camera as previously described.

Next the transformation between the camera reference system and the end effector (eef) reference system is carried out:

$$\begin{bmatrix} x_{eef} \\ y_{eef} \\ z_{eef} \\ 1 \end{bmatrix} = \begin{bmatrix} R_{camera\_to\_eef} & T_{camera\_to\_eef} \\ \underset{1\times3}{\mathbf{0}} & 1 \end{bmatrix} \begin{bmatrix} x_{camera} \\ y_{camera} \\ z_{camera} \\ 1 \end{bmatrix} \tag{3.5}$$

The Python implementation of this transformation is based on the `tf2_ros` and `tf2_geometry_msgs` package, which allows to retrieve the rotation and translation matrices $R_{camera\_to\_eef}$ and $T_{camera\_to\_eef}$ from the imposed robot description.

This first transformation is followed by the one between the end effector (eef) reference system and global reference system (world):

$$\begin{bmatrix} x_{world} \\ y_{world} \\ z_{world} \\ 1 \end{bmatrix} = \begin{bmatrix} R_{eef\_to\_world} & T_{eef\_to\_world} \\ \underset{1\times3}{\mathbf{0}} & 1 \end{bmatrix} \begin{bmatrix} x_{eef} \\ y_{eef} \\ z_{eef} \\ 1 \end{bmatrix} \tag{3.6}$$

In Python this transformation is carried out computing the rotation and translation matrices $R_{eef\_to\_world}$ and $T_{eef\_to\_world}$ from the streamed information coming from the `robot_state` ROS message.

Eventually, the final transformation between the object pixel and depth coordinates $(u, v, depth)$ and the global coordinates $(x_{world}, y_{world}, z_{world})$ of the detected object in the global reference system is recursively assembled:

$$\begin{bmatrix} x_{world} \\ y_{world} \\ z_{world} \\ 1 \end{bmatrix} = \begin{bmatrix} R_{eef\_to\_world} & T_{eef\_to\_world} \\ \underset{1\times3}{\mathbf{0}} & 1 \end{bmatrix} \begin{bmatrix} R_{camera\_to\_eef} & T_{camera\_to\_eef} \\ \underset{1\times3}{\mathbf{0}} & 1 \end{bmatrix} \begin{bmatrix} R_{obj\_to\_camera} & T_{obj\_to\_camera} \\ \underset{1\times3}{\mathbf{0}} & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ depth \\ 1 \end{bmatrix} \tag{3.7}$$

Which can be rewritten as:

$$\begin{bmatrix} x_{world} \\ y_{world} \\ z_{world} \\ 1 \end{bmatrix} = \begin{bmatrix} R_{obj\_to\_world} & T_{obj\_to\_world} \\ \underset{1\times3}{\mathbf{0}} & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ depth \\ 1 \end{bmatrix} \tag{3.8}$$

By using this transformation it is possible to estimate the global coordinates of the detected bounding box center.
This global detection is stored in the structured `detections_with_positions.json` file after being:

- **linked with the current Detection Label**: to keep the semantic information of the detection.
- **compared with the previous detections**: to avoid the presence of multiple, slightly different target positions for the same fruit. If the current detection Euclidean distance to each already stored detection is smaller than a set threshold, the last position is discarded. Otherwise, it is appended to the file, as considered a new instance.

Once all the waypoints have been covered, the final `detections_with_positions.json` file should contain all the detected lemons and occlusions.

3.7. Simulated Picking Routine

Once the complete `detections_with_positions.json` file is ready it is loaded to stream the picking target positions. The simulated picking routine is designed to maximize the picking success rate.
To do so, proper coordination between the Arm and Gripper motions has to be established. Furthermore, it would be beneficial to achieve a better success rate by monitoring the state of the picking target detection throughout the picking process. It is expected to find the detection again while approaching the estimated picking position, while the detection is expected to get lost after picking since the fruit is now in the gripper.

So the picking routine is divided into five phases:

1. **Approaching to Picking Position**: an intermediate position is placed before the estimated picking position. To achieve the additional detection control, two additional rounds of inference are carried out, confronting the updated detections with the ones stored in the target positions list. Once the picking target position has been confirmed, the gripper is opened.

2. **Reaching Picking Position**: once the gripper has been opened and the target confirmed, the gripper is moved to the picking position, waiting the synchronization of all the codes.

3. **Reaching Picked Position**: Once the end-effector reaches the desired position, the gripper is closed, after a quick pause to compensate small positioning errors.

4. **Approaching from Picking Position**: an intermediate position is placed after the picking position and before going to the place position. To achieve the second part of the additional detection control, two additional rounds of inference are carried out, checking if the detection has been removed.

5. **Reaching Placing Position**: once the end effector gets some distance from the plant and the actual picking of the fruit is confirmed, it can be moved to place position, still holding the fruit. Once placing position is reached, the gripper is commanded to be opened, releasing the fruit and becoming available for the next picking target in the list.

For each target position, its corresponding approach and picking positions are computed by radially adding some specific distances along the vector connecting the target position and the estimated plant center, after being projected on the global $(x-y)$ plane as shown in Figure 3.11:

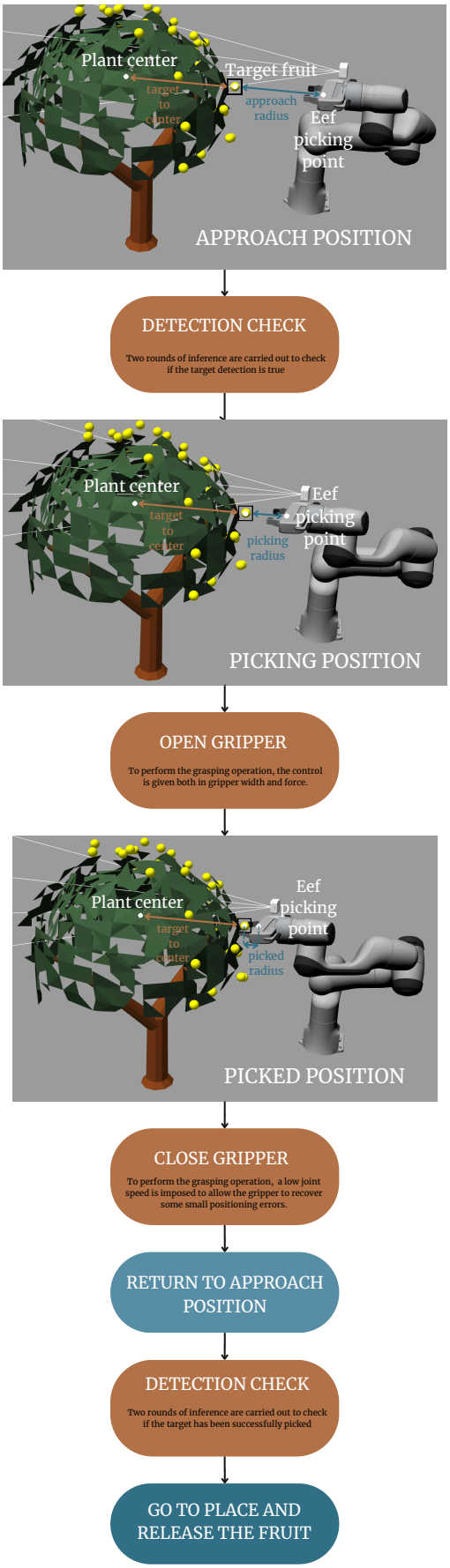


Figure 3.11: Simulated picking pipeline with all the main phases

The specific set of approach, picking and picked radius are reported in Table 3.6:

Parameter	Approach radius	Picking radius	Picked radius
`Value`	40 cm	10 cm	5 cm

Table 3.6: Approach, picking and picked radius

To save the results of the inferences carried out during the approaches to and from the picking position, the results folder is updated, adding the ''`picking`'' and ''`picked`'' folders, respectively.
The final structure is the following:

```
results_{date}_{time}/
├── detection/
│   ├── waypoint_1
│   │   ├── detections.png
│   │   └── detections_canva.png
│   ├── waypoint_2
│   ├── ...
│   └── waypoint_n
├── picking/
│   └── waypoint_1
│       ├── detections.png
│       └── detections_canva.png
├── picked/
│   └── waypoint_1
│       ├── detections.png
│       └── detections_canva.png
├── inference_time/
│   └── inference_time.txt
└── detections_with_positions.json
```

Singe the damage level on the fruit cannot be estimated in simulation or with fake fruits, the only remaining metric that can be compute to evaluate the performance of the picking process is the *SuccessRate*:

$$\text{Success Rate} = \frac{\text{number of picked fruits}}{\text{number of reachable fruits}} \tag{3.9}$$

4 | Experimental Tests

As stated before, the proposed methods are chosen to be tested in a realistic and challenging picking environment, recreating a situation where fruit occlusion is a strong issue.

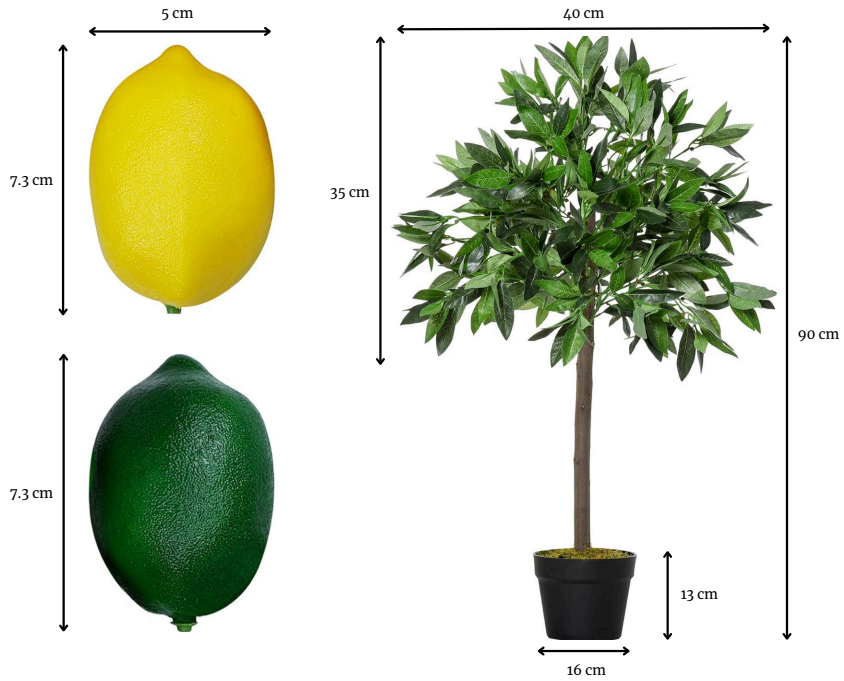
4.1. Creating the picking environment

To do so, the first focus is reserved to the proper choice of the plant and of the fruits. To allow the experiments to be carried out for a long period of time and most notably in winter, when most of the annual plants (like tomatoes for instance) are dead and even perennials don't have fruits on their canopies, both plant and lemon fruits are chosen among the most realistic fake plants available on the market.
The plant, in particular, is required to have a very leafy canopy, allowing to easily occlude fruits. The fruits themselves have to be fairly rigid, to be handled by a conventional gripper, and as realistic as possible in size and color.
To add possible layers of exploration, as explained in Methodologies, both yellow and green lemons are included.

The other requirements of the experimental environment involved the Robot manipulator, the sensing device and the gripper capabilities. Similarly as done for the simulated environment, the main requirements regarding sensing and control of the robot for real–world detection and picking are satisfied with the following choices:

- **Robotic Arm Integration**: the Emika Franka Panda present at the "Istituto Dalle Molle di Studi sull'Intelligenza Artificiale" (IDSIA) is used
- **Gripping Capability**: the standard two finger Panda Hand gripper of the Franka Panda is used.
- **Depth Camera Capability**: the real–world `Intel Realsense d405` depth–stereo camera is used.
- **Flexible Python-Based Control**: the implemented Python codes used in simulation are update and integrated with the real hardware.

The final selection of plant and fruits is reported in Figure 4.1:

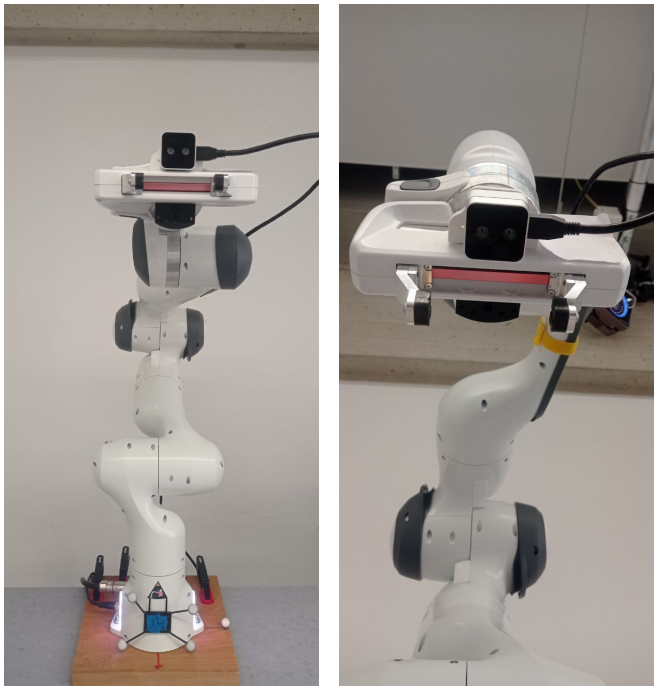


(a) Yellow and green lemons (b) Chosen plant

Figure 4.1: The chosen fake lemons and plant, with their dimensions

The fruits are positioned and attached to the plant in accordance with the specific routine being conducted.
To evaluate the detection performance, with a focus on occlusion detection, the fruits are as extensively concealed as possible within the plant canopy, without being rigidly attached to the plant.
Conversely, for the picking routine, the fruits are secured to the plant using rubber bands, enabling the gripper's limited fingers to access the fruit and perform the grasping operation.

The robotic system employed, comprising the `Intel Realsense d405` depth-stereo camera and gripper, is depicted in Figure 4.2, while the various fruit configurations are illustrated in Figure 4.3:



(a) Franka Panda Robot (b) Camera and gripper

Figure 4.2: The Franka Panda Robot, with the mounted `Panda Hand` gripper and the `Intel Realsense d405` camera

(a) Detection configuration (b) Picking configuration

Figure 4.3: The experimental assembly of plant and fruits, with different configurations

4.2. Real Franka Panda Robot control

The control framework of the real–world robot is exactly the same of the one used in simulation (3.3).
The only difference is in the designated launch file used to load both the robot description and the robot's real controller. In fact, to start simultaneously the real–world controller, its real position feedback from the hardware's sensors and the RViz real–time visualization of the robot state, it is necessary to add the Robot IP address to the basic Command Line Interface (CLI) launch command, after enabling the Franka Control Interface (FCI) on the robot server.
The launched RViz GUI allows immediate control of the robot. Leveraging the FCI, the developed Python-based control can also be used, exactly like in the simulation.

4.3. RealSense D405 Camera

Differently form the simulated version of the camera, the real RealSense D405 Camera has a unified Software Development Kit (SDK) called Intel RealSense SDK 2.0.
This open-source package offers several pre-built solutions to stream camera data on various platforms. Since the robot is going to need the information coming from the camera in a Python and ROS-compatible environment the following packages are used:

- **realsense-ros**: containing both the aforementioned (3.4) `realsense2_description` and the `realsense2_camera` packages. The former provides all the description files, such as visual meshes and simulated camera launch files, while the latter provides all the source codes to stream the real–world data from the camera into specific ROS topics.
  So, by launching the `rs_camera.launch` file, it is possible to publish several topics, from RGB, to depth and infrared images, depending on the camera capabilities.
- **Pyrealsense2**: the python wrapper for Intel RealSense SDK 2.0 provides the C++ to Python binding required to access the SDK. This library is of utmost importance considering the possibility to directly access the topics in Python, besides exploiting already implemented tasks, such as RGB and depth alignment or pixel projection in camera coordinates.

The capabilities of these two different declinations of the RealSense SDK are exploited differently, the former is used for the intrinsic and extrinsic calibration of the camera , while the latter is integrated in the Python-based fruits positions estimation, through the same detection pipeline carried out in simulation.

4.3.1. Calibration Procedures

The global calibration procedure of the camera is composed by two separate sub-tasks, the intrinsic camera calibration and the extrinsic camera calibration, that for this application consist of:

- **The intrinsic camera calibration**: this procedure allows to extract the real camera intrinsic parameters (3.1). This parameter are automatically confronted with the standard values that the camera possess when it is factory–new. If the present values perform bad, an automatic procedure is carried out to correct them and improve accuracy.
  This procedure is fundamental since, differently from the case of the simulated camera, the transformation between $(u, v, depth)$ image coordinates and $(x_{camera}, y_{camera}, z_{camera})$, object coordinates in the camera reference system, is carried automatically by the `rs2_deproject_pixel_to_point` Python fuction of the Pyrealsense2 library. So a precise estimation of the intrinsic parameters is of outmost importance for the overall precision of the targets positions estiamtion.

  The intrinsic calibration is carried out by directly using the `realsense-viewer` from the realsense-ros package. It automatically estimates and corrects the intrinsic camera parameter just using the RGB and depth streams of a close planar surface, like a wall. A particular attention has to be given to the light conditions, since scarce or very intense illumination can interfere with correct development of this process.

- **The extrinsic camera calibration**: this procedure allows to estimate the transformation between the camera reference system and the global reference system, through the transformation between the camera and an third intermediate reference system, in this case the end effector one, in the so called `hand-eye calibration`. The problem can be expressed as:

$$AX = ZB \tag{4.1}$$

  where A and B represent known transformations (e.g., between the robot base and a camera), and X and Z are unknown transformation matrices. A special case occurs when $X = Z$, reducing the problem to:

$$AX = XB \tag{4.2}$$

In fact, differently from the simulated environment where the relative transformation between the camera and end–effector reference systems is imposed in the robot URDF file, in the real world is extremely difficult to estimate the relative position and rotation of the camera mounting w.r.t. the end effector. Several state-of-the-art methods have been developed to perform indirectly this estimation, retrieving both the camera to end-effector and the camera to world transformations. The most common method is the one devolved by Daniilidis [11].

The extrinsic calibration is carried out with `MoveIt Calibration Plugin` on RViz. This plugin allows to exploit both the uploaded robot description and its robot state update, moving the camera (through the robot) in different positions and contextually saving the end effector pose for each position. To publish the image topic and the camera transformations frames (tf) the standard realsense2_camera launch file is used.
The calibration procedure is composed by the following steps:

1. **Aruco target Generation**: a standard Aruco Hand-Eye Target, whose set parameters are reported in Table 4.4, is generated (Figure 4.5) and printed in A4 format. This target is automatically detected by the plugin, estimating the target reference frame.

2. **Geometric Context definition**: the geometric context is defined by setting the type of extrinsic calibration (Hand-Eye calibration), the target reference frame (the one detected in the previous step),the camera reference frame (the one indicated by realsense2_camera node), the the robot end-effector reference frame and the robot base reference frame (both indicated in the launched robot description).

3. **Dataset collection**: several images of the Aruco target are taken from different points of view, saving the robot pose for each one of them. The indicated goal number of images is around 15, after this number usually the estimation error has already asymptotically converged, so any additional image isn't reflected in a precision improvement.

4. **Matrices computation**: eventually, the estimated matrices are computed using a wide range of methods. As stated before, in this case the Dual Quaternions method ([11]) is used.
   The computed end-effector to camera transformation matrix is thus printed in the CLI.

Parameter	Value
Number of markers, X	3
Number of markers, Y	4
Marker size	200 px
Marker separation	20 px
Marker border	1 bit
ArUco dictionary	DICT_5X5_250

Figure 4.4: Aruco target parameters

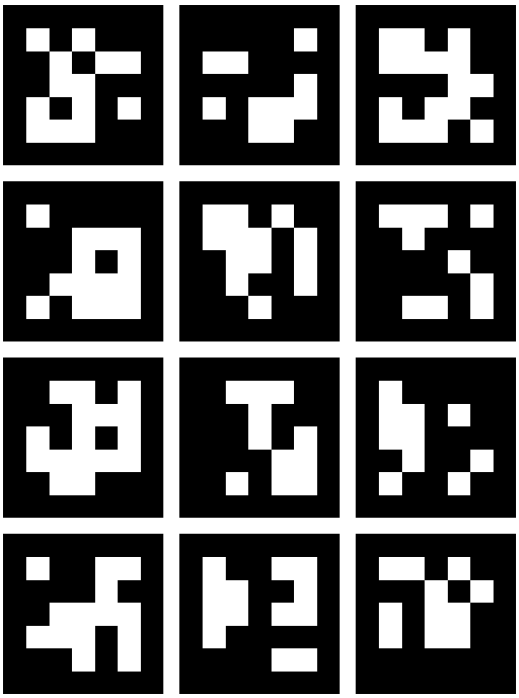


Figure 4.5: Generated Aruco Target

4.3.2. Python integration of the camera

The `Pyrealsense2` package is used to create the corresponding callable python function to what was done in simulation (3.4).
The implemented `get_realsense_images` function thus retrieve the same kind of information w.r.t. the simulated camera, with just different formats and encoding, since it relies on the Pyrealsense2 framework.

The information about currently-steamed RGB image, depth image and camera intrinsic parameters is then fed to the `TargetPosition` function, as for the simulated detection pipeline. This allows to tailor the same detection and picking pipelines proposed for the simulated environment for the real–world detection and picking operations.

The final significant difference, as previously anticipated, lies in the use of the real camera. In this case, the `rs2_deproject_pixel_to_point` function replaces the entire image-to-camera transformation, as it is already implemented within the `Pyrealsense2` package. In a similar way, the imported `align` function substitutes the alignment process done in the simulation, leveraging the information of RGB and depth frames given by the imported `wait_for_frames` function.

4.4. Real Environment 3D reconstruction

The same pipeline, described in 2.2 and already performed in 3.5, is carried out also in the real picking environment.
A `.mp4` video is recorded by the real mounted camera , while the robot navigates around the plant. After this, the video is saved and exported to Windows, as explained before, to undergo the proposed 3D reconstruction pipeline. The following results are obtained:

Figure 4.6: Coarsely filtered colored 3DGS point cloud

Figure 4.7: Finely filtered colored 3DGS point cloud

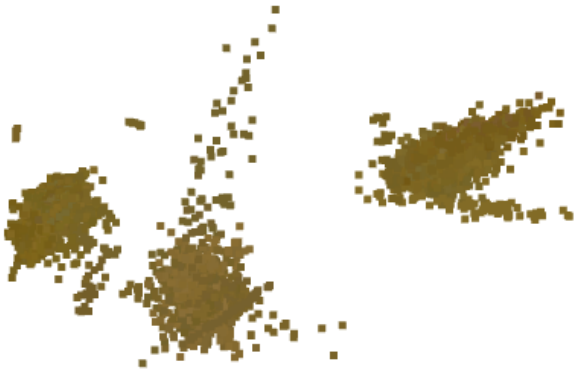


Figure 4.8: Segmented lemons from colored 3DGS point cloud

4.5. Real–world Detection and Picking routines

The core principles of both the detection and picking routines are also implemented in the real–world picking scenario. Thanks to the seamless integration of all hardware components into a Python-compatible framework, the developed software is fully adaptable for real–world testing.

Given that real picking operations are the ultimate goal of this research and thesis, an in-depth analysis is conducted on the progression of detection and picking phases throughout the entire operational pipeline.

Since picking time directly impacts operational costs, a key objective is to identify the optimal sequence of detection and picking phases that minimizes the total picking time per fruit. While this analysis remains at an initial stage, it is a crucial step toward making robotic picking increasingly competitive compared to fully manual labor.

When evaluating the picking process across an entire plant, two fundamental strategies emerge:

1. **Detect all – Pick all**: The detection routine is completed first, identifying all fruits. The picking routine then follows, processing the stored detections sequentially until all targets are harvested, as shown in Figure 4.9.
   This strategy focus on having the highest detection rate possible, sacrificing some time detecting and analyzing fruits that have been already seen.

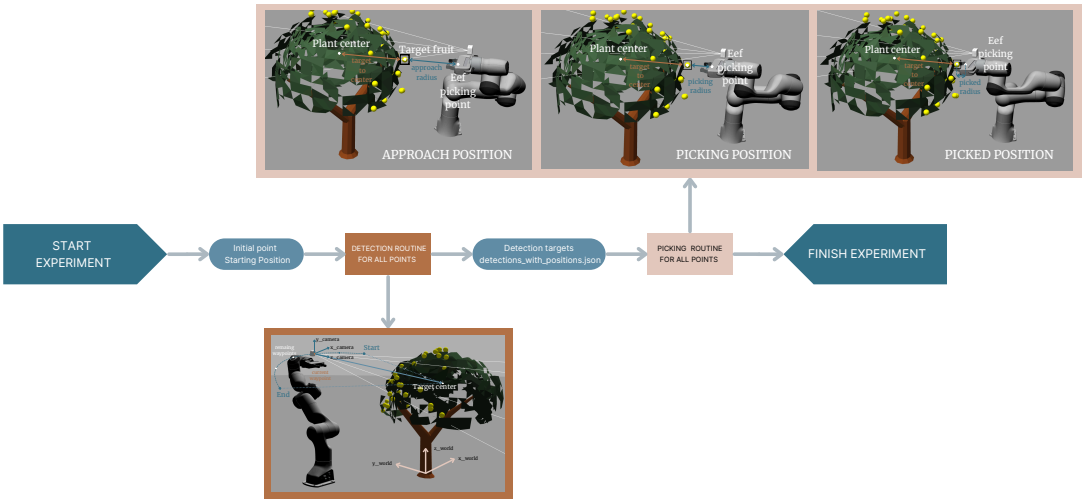


Figure 4.9: First proposed overall picking operational strategy

2. **Detect – Pick – Repeat**: Each detected fruit is immediately picked before moving to the next detection. This cycle continues until all waypoints in the routine are covered, as shown in Figure 4.10.
   This strategy focus on minimizing all the unnecessary repetitions of detection phases, picking the target when is first detected, sacrificing some accuracy but increasing the picking speed.

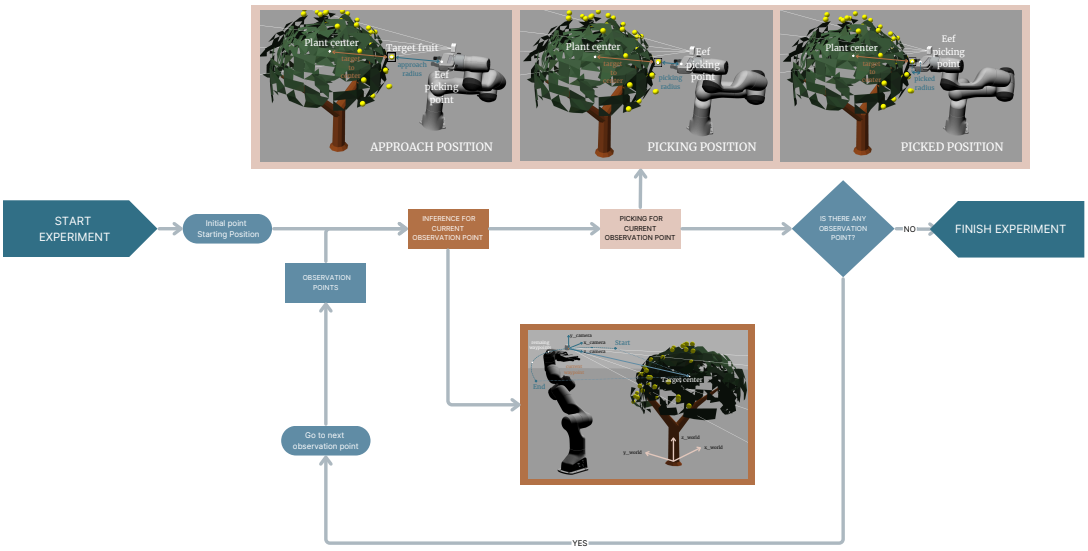


Figure 4.10: Second proposed overall picking operational strategy

To evaluate the performances of the two overall picking strategies in the real world, some additional metrics can be computed:

- **Total Picking time**: the total picking of the overall routine has to be computed and saved.
- **Effective Picking time per fruit**: the effective picking time per fruit has to be computed and saved, to understand what is the proportion w.r.t to the total picking time.
- **Success Rate**: as explained in 3.9, the total number of picked fruits over the number of reachable fruits is a good metric of the picking effectiveness.

5 | Results

In this chapter, the most meaningful results of this thesis work are presented and discussed.
Special attention is given to the most critical comparisons between the proposed methods and state-of-the-art techniques, evaluated using specific metrics for each method. The primary goal of this thesis is to enhance the detection of occluded fruits, thereby maximizing the picking success rate: consequently the results are presented highlighting the major effects on these two operations provided by the proposed methods.

5.1. 3D environment reconstruction

As reported in Sections 3.5 and 4.4, the results of the 3D reconstruction pipeline are presented in terms of filtered, unfiltered and extracted lemons clusters point clouds.
To compare them with other techniques, like NeRF, the same starting dataset and the same resources are used. In particular, the GPU used is the same reported in Table 2.4 and 2.5. The results are shown in groups according to the type of environment that has been reconstructed, either simulated or real.
To compute the NeRF point cloud, the open-source environment "Instant Neural Graphic Primitives (NGP)"[33] is used. The generated model is then meshed, exported, and converted to point cloud format to compare results.

To be consistent with the set of training parameters used for the different models, the following values are imposed:

Method	Number of simulated images	Number of real images	Maximum number of iterations
`NeRF`	90	228	30 000
`3DGS`	90	228	30 000

Table 5.1: Set of imposed training parameters for each model

Since the standard metrics for these models are just covering the rendering process, the visual results are presented in Figures 5.1 and 5.2, while the metrics related to the point clouds are reported in Tables 5.2 and 5.3.

(a) NeRF rendering

(b) 3DGS rendering

(c) NeRF filtered point cloud

(d) 3DGS filtered point cloud

(e) NeRF extracted lemons

(f) 3DGS extracted lemons

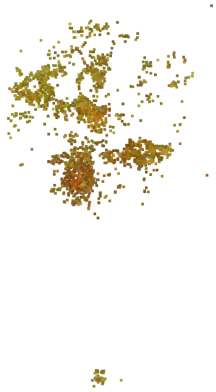
Figure 5.1: NeRF and 3DGS results of the simulated environment

(a) NeRF rendering

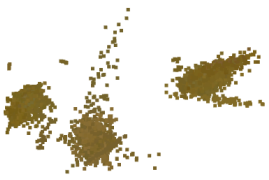
(b) 3DGS rendering

(c) NeRF filtered point cloud

(d) 3DGS filtered point cloud



(e) NeRF extracted lemons



(f) 3DGS extracted lemons

Figure 5.2: NeRF and 3DGS results of the real environment

Parameter	NeRF	3DGS
Training time	2 h 13 min	1 h 29 min
Training loss	0.0516	0.0135
Post processing time	12 min 32 s	1 min 24 s
Number of computed points	2 007 406	9 761 191
Number of clustered lemons points	2 619	45 864
Number of visible lemons (ground truth)	35	35
Number of detected lemons	25	32

Table 5.2: Metrics for the 3D reconstruction process of the simulated environment

Parameter	NeRF	3DGS
Training time	1 h 17 min	53 min
Training loss	0.0500	0.0407
Post processing time	1 min 41 s	1 min 30 s
Number of computed points	436 891	9 767 497
Number of clustered lemons points	3 547	7 789
Number of visible lemons (ground truth)	3	3
Number of detected lemons	1	3

Table 5.3: Metrics for the 3D reconstruction process of the real environment

5.2. YOLOv11 fruit detection pipeline

The results of the proposed YOLOv11-based detection pipeline are presented both in terms of visual detections and overall detection precision.

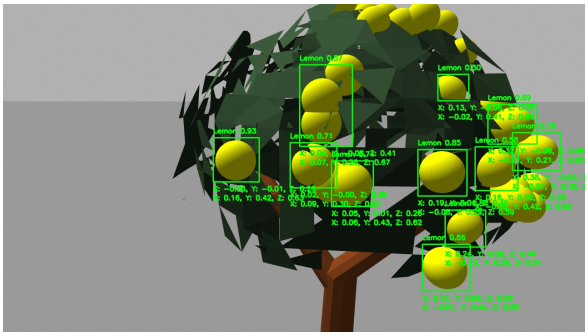
To assess the performance of the proposed methods, we also report the results of the standard YOLOv11 network, trained on the same dataset as the proposed one, as presented in Section 2.1. Only the detection model results are included, as the segmentation task is left for future developments.

The inference is carried out using minimum confidence thresholds: detections with a confidence under these values are not considered. The choice of these values is totally arbitrary, the particular choices for the simulated or experimental environments come from an empirical tuning process composed of several trials and errors, with consequent improvements. To obtain a fair comparison between the standard and the proposed detection models, the same minimum threshold for the "Lemon" class is set, while the additional classes are adjusted according to their behavior.
The final values are reported in Table 5.4:

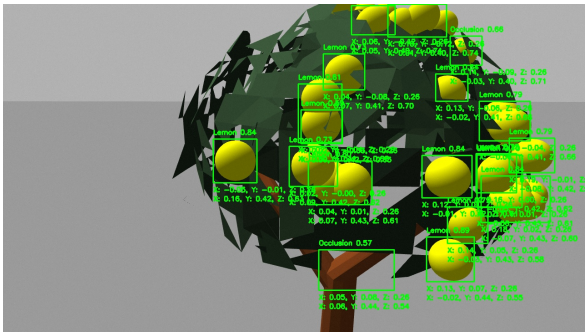
Standard Detection Model		
Class	Simulation	Experimental benchmark
`Lemon`	0.5	0.7
Proposed Detection Model		
Class	Simulation	Experimental benchmark
`Lemon`	0.5	0.7
`Lemon_green`	0.9	0.9
`Occlusion`	0.5	0.5

Table 5.4: Tailored minimum confidence values used in the inference process of the standard and proposed models for the simulated and the experimental environment

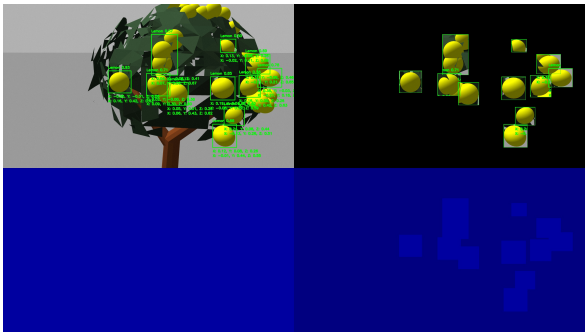
For the sake of brevity, only the most meaningful visual results are shown in Figure 5.3 and Figure 5.4. The reported metrics, however, are computed on the entire set of executed detection routines, both in simulation and the real benchmark. A more comprehensive overview of other notable visual results can be found in Appendix A.



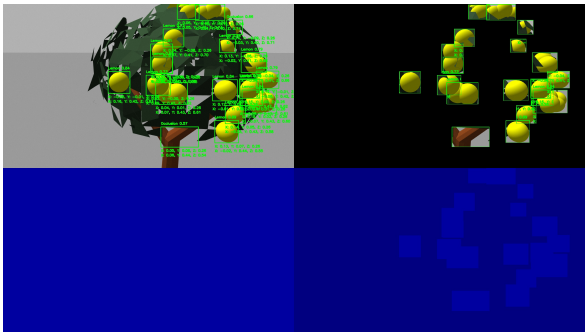
(a) Standard Model



(b) Proposed Model



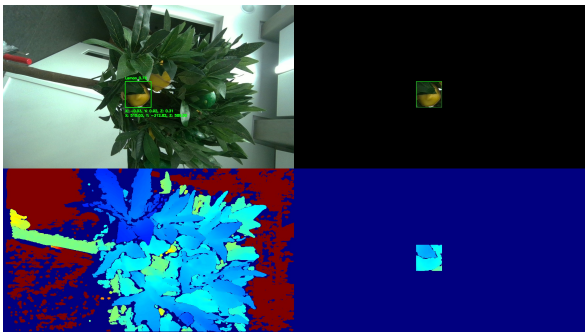
(c) Standard Model



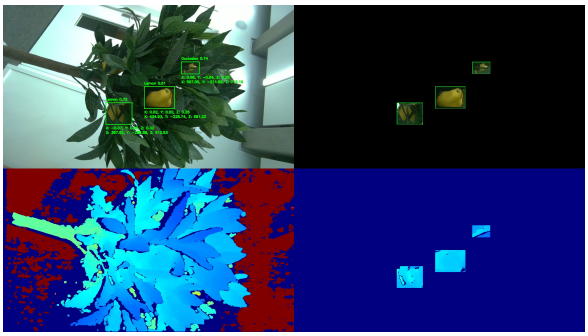
(d) Proposed Model

(e) Standard Model

(f) Proposed Model



(g) Standard Model



(h) Proposed Model

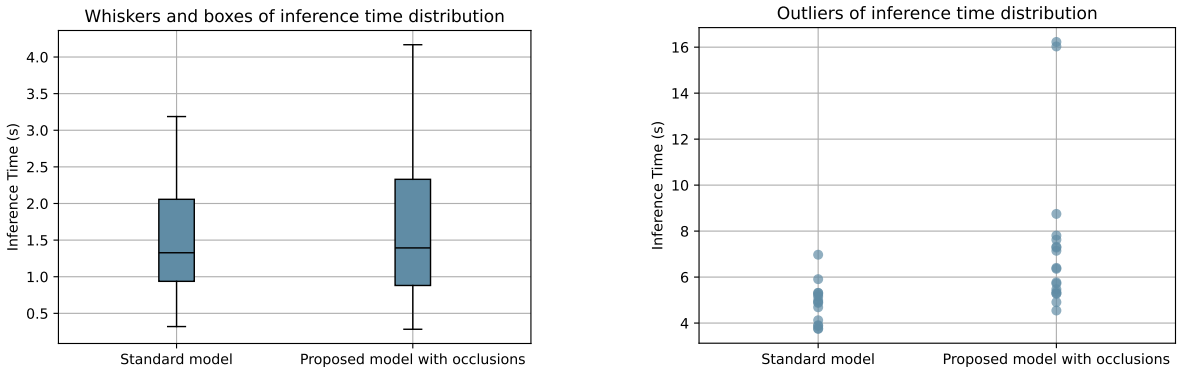
Figure 5.3: Visualized detections on the streamed RGB and depth images of simulated and experimental environment: comparison between the standard and the proposed model

Figure 5.4: Successful detections of severely occluded fruits

Among the most commonly used metrics in the literature, Precision (P)(2.1), Recall (R)(2.2), Intersection over Union (IoU)(2.5), and Mean Inference Time are chosen to be computed. These metrics are reported first for the simulated environment and then for the real benchmark.

Metric	Standard Model	Proposed Model
Number of images	24	24
TP	184	318
FP	24	26
FN	210	60
P	0.88	0.92
R	0.47	0.84
IoU	0.44	0.79
Mean Inference Time	1 793 ms	2 319 ms

Table 5.5: Metrics for the YOLOv11 detection process of the simulated environment

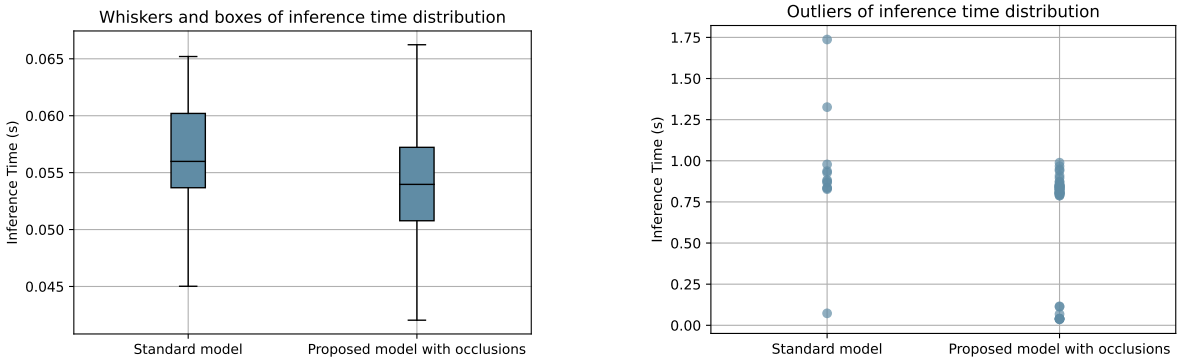


(a) Boxplots comparison (b) Outliners comparison

Figure 5.5: Boxplots and outliners comparison of total inference time distributions in the simulated environment

Metric	Standard Model	Proposed Model
Number of images	172	304
TP	100	636
FP	0	0
FN	372	320
P	1	1
R	0.211	0.6652
IoU	0.211	0.6652
Mean Inference Time	150.6 ms	105.9 ms

Table 5.6: Metrics for the YOLOv11 detection process of the real environment



(a) Boxplots comparison (b) Outliners comparison

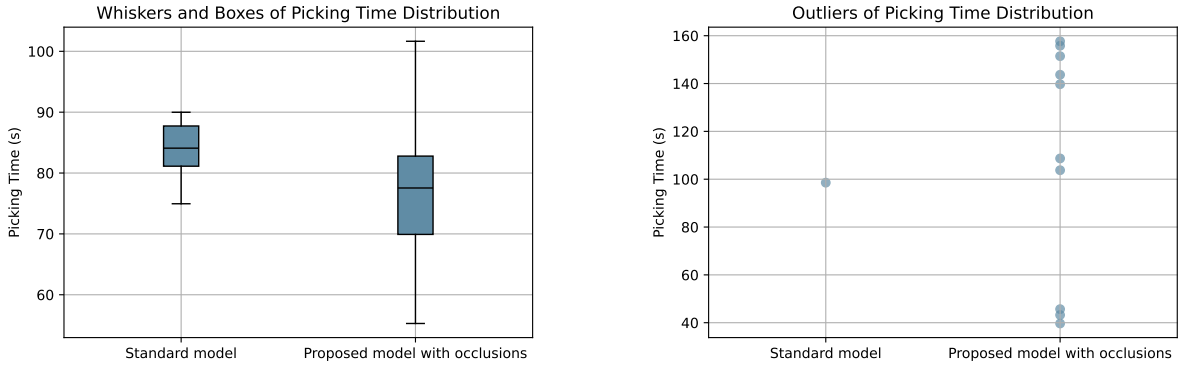
Figure 5.6: Boxplots and outliners comparison of total inference time distributions in the real environment

5.3. Picking pipeline

To evaluate the picking results in the experimental benchmark, two different strategies are highlighted (Section 4.5). Consequently, the aforementioned picking metrics are computed for each strategy and compared.

Due to challenging issues associated with the real hardware safety protocols and the limited availability of the robot, only individual fruit-picking routines are performed. As a result, the time required for a single fruit-picking operation is recorded and analyzed. The total picking time is subsequently calculated by summing the contributions of the best outcomes for each picking strategy.

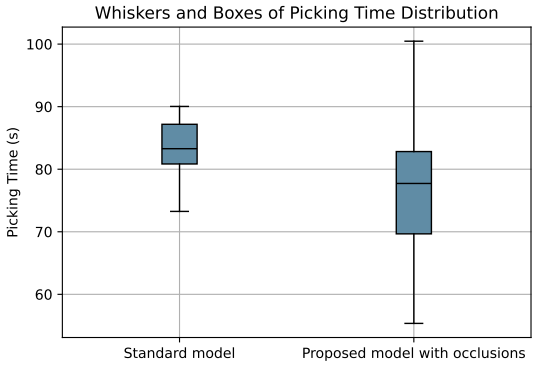
The results are shown according to the chosen picking strategy and used detection model. In Figure 5.7 the picking time distribution for the "detect all–pick all" strategy of the standard detection model is compared with the one of the proposed detection model. Similarly, the picking time distribution for the "detect and pick–all" strategy of the standard detection model is compared with the one of the proposed detection model in Figure 5.8.
Eventually, the best results in terms of success rate and total picking time for each picking strategy and detection model are reported in Table 5.7 and Table 5.8.



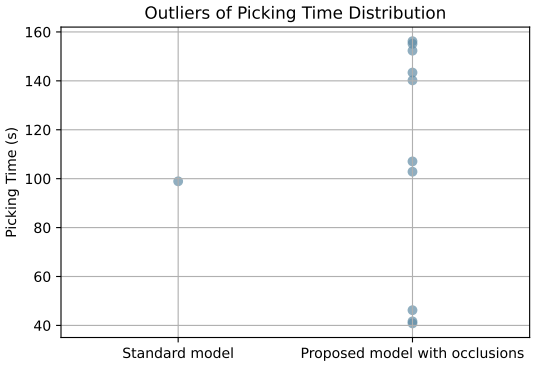
(a) Boxplots comparison

(b) Outliners comparison

Figure 5.7: Boxplots and outliners comparison of picking time distributions for the "detect all–pick all" strategy



(a) Boxplots comparison



(b) Outliners comparison

Figure 5.8: Boxplots and outliners comparison of picking time distributions for the "detect and pick–all" strategy

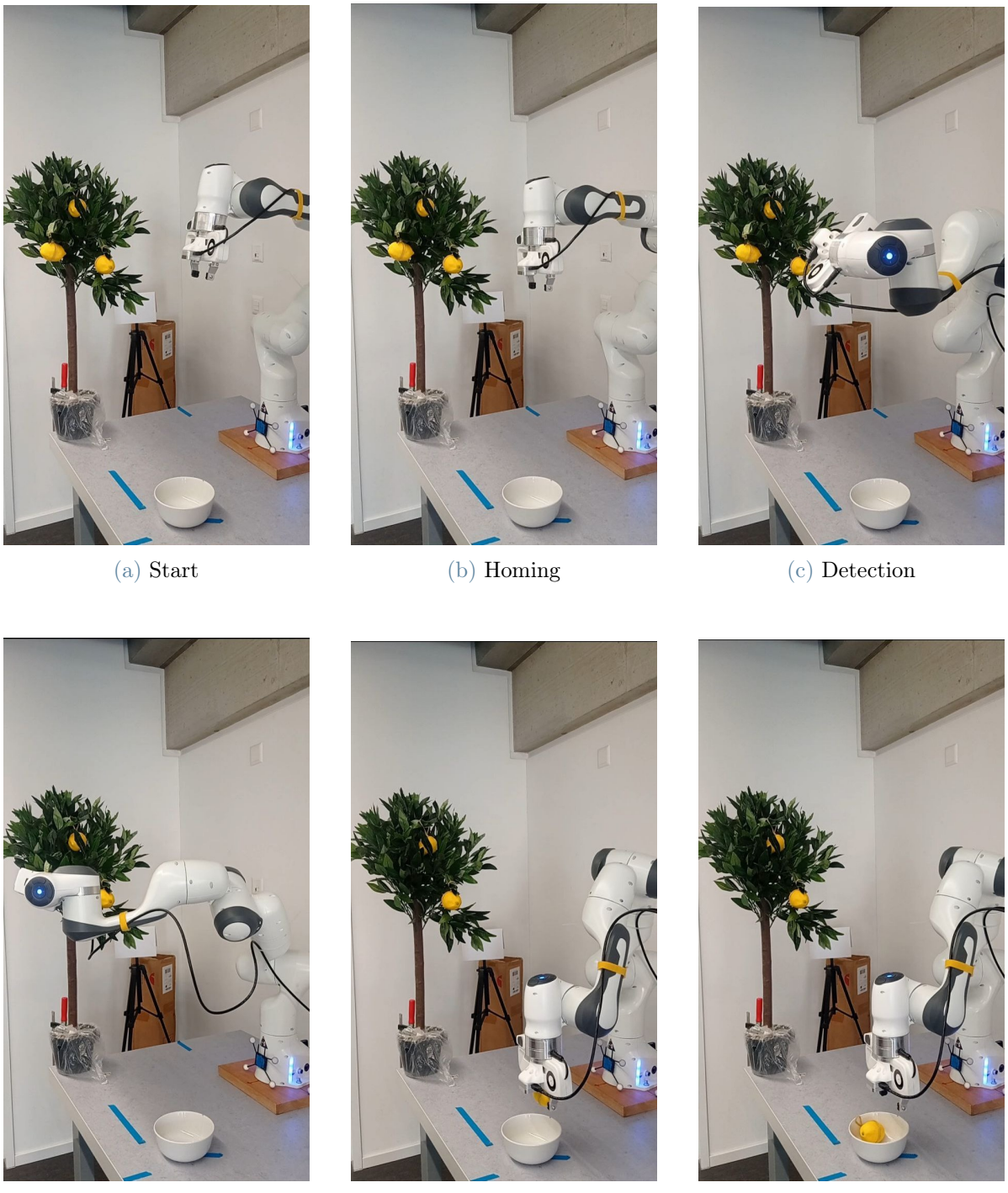
Metric	Standard Model	Proposed Model
Reachable lemons	5	5
Picked Lemons	3	4
Success Rate (%)	60	80
Total Picking Time	6 min 5 s	7 min 20 s
Average picking time	2.02 $\frac{\text{min}}{\text{lemon}}$	1.92 $\frac{\text{min}}{\text{lemon}}$

Table 5.7: Picking metrics for the "detect all–pick all" strategy

Metric	Standard Model	Proposed Model
Reachable lemons	5	5
Picked Lemons	4	5
Success Rate (%)	80	100
Total Picking Time	7 min 56 s	7 min 43 s
Average picking time	1.98 $\frac{\text{min}}{\text{lemon}}$	1.54 $\frac{\text{min}}{\text{lemon}}$

Table 5.8: Picking metrics for the "detect and pick–all" strategy

The most crucial phases of the “detect and pick–all” picking strategy are depicted while being executed in Figure 5.9:



(a) Start (b) Homing (c) Detection

(d) Picking (e) Placing (f) Placed

Figure 5.9: “detect and pick–all” strategy phases in the real environment

5.4. Results discussion

The presented results are then critically analyzed and discussed, with special attention to the comparison with state-of-the-art techniques.

5.4.1. 3D environment reconstruction

The results for the 3D reconstruction pipeline of the picking environment show interesting features.

Starting from the comparison between NeRF and 3DGS-derived point clouds, it is possible to notice a great improvement both in quality, testified by the lower values of training loss, and in point cloud density, proved by the higher number of resulting points in the cloud (Table 5.2 and Table 5.3).
This is achieved simultaneously with a significantly lower training and post-processing time. One key explanation for this great difference is that 3DGS outputs by default the reconstructed point cloud, while NeRF offers the possibility to export just a coarsely reconstructed mesh that has to be additionally processed into a point cloud with a dedicated code.

This greater flexibility and enhanced precision are also reflected in the processed and clustered point cloud of the fruits (Figure 5.1f and 5.1f), which appears far more accurate for the 3DGS-based pipeline.

5.4.2. YOLOv11 detection pipeline

The results for the proposed YOLOv11 detection pipeline show a critical improvement over the standard one when tested in the same environment and with the same parameters.

In particular, a drastic improvement in inference speed is achieved while dealing with comparable numbers of detections per image (Table 5.6).
Only when the number of detected instances for the proposed model is far greater than the standard one, the proposed model is slower than its counterpart (Table 5.5). On the other hand, this means that the proposed model is outperforming the standard one greatly in detections number and quality, as testified also by the visual results shown in Figure 5.3.

Another essential consideration is the balance between the number of False Positives and False Negatives. As shown in Tables 5.6 and 5.5, the overall number of False Positives is very low, while the number of False Negatives can sometimes be quite high. This result

can be attributed to the specific choice of minimum inference thresholds.
The chosen values have a more cautious effect, significantly reducing the False Positives, which are far more dangerous in robotics operations, while accepting more False Negatives as a trade-off. This high number of False Negatives, being computed on the images and not on the entire plant, might not be a significant problem since other views are likely to cover the missed fruits.

In fact, by knowing the exact number of fruits on the plant, it is possible to compute the Overall Accuracy of the detection process as:

$$\text{Overall Accuracy} = \frac{\text{number of detected fruits}}{\text{number of fruits}} \tag{5.1}$$

The overall accuracy results for the proposed detection pipeline in the real picking environment, with fruits occluded more than 50%, is confronted with the ones found in literature (Table 1.3):

Occlusion Condition	**Model**	**Citrus Count**	**Correctly Identified**	
			Amount	**Rate (%)**
More than 50%	Faster-RCNN	200	156	78.24
	YOLOv3	200	148	74.22
	YOLOv4	200	166	83.18
	Improved YOLOv4	200	182	90.82
	Proposed YOLOv11	164	160	97.56

Table 5.9: Identification parameters across different levels of occlusion [9]

The comparison in Table 5.9 illustrates how the combination of the proposed YOLOv11 detection network's precision and the high spatial coverage provided by the circular detection trajectory yields highly promising results, even in the presence of a high degree of occlusion (Figure 5.4).

This accuracy is also demonstrated by the high level of picking precision achieved in the picking pipeline. Without reliable detections, this level of precision would not have been possible.

5.4.3. Picking pipeline

The results of the experimental picking pipeline highlight a deep relation with the type of strategy used.

As reported in Table 5.7 and 5.8, the "detect and pick–all" strategy with the proposed detection model achieves the best results, with a stuggering 100% Success Rate at its best performance.

The real bottleneck of the picking procedure with real robot is represented by the intervention of the installed safety controls, especially regarding self–collisions between joints.

Considering the complicated motion that the robot is required to carry out during the various phases of the picking routine, especially regarding the end–effector orientation, the likelihood of approaching the limit set to avoid the collisions between robot links dramatically increases.

In these circumstances, the robot is immediately blocked and forced into safety mode, where it can only be moved manually in collaboration with the operator. To resume the automatic Python-based control of the robot, the operator needs to manually resolve the upcoming collision, moving the robot in a safe pose. By doing so, the overall picking routine is fragmented almost after each time a fruit is picked.

By overlooking the fragmentation of the various picking processes, the routine as a whole can nevertheless be considered very promising.

Speaking of overall picking time, while selecting the optimal combination of strategy and detection model can reduce it, there is still a need for significant improvement to make robotic picking economically feasible in agriculture.
A significant contributor to the overall picking time is the numerous pauses imposed in the code, along with the stringent velocity and acceleration scaling factors (Table 5.10) that have been added in this preliminary testing phase.

Scaling factor	**Velocity**	**Acceleration**
Value	0.1	0.1

Table 5.10: Assigned velocity and acceleration scaling factors

By reducing these constraints, the overall picking time can be significantly decreased while still maintaining a high level of safety.

Considering the picking time for each fruit, it is evident that it is almost unaffected by the choice of picking strategy, as shown in Figures 5.7 and 5.8.
The major difference lies in the varying durations of the inference procedures of the standard and proposed models. The additional inferences that are carried when approaching and leaving the picking position can be minimized or refined in the future to save extra time.

A very strong limitation of the current picking hardware is the grasping interface.
As reported in the Section 1.2, the customization of the grasping interface can help significantly at compensating small positioning error or to penetrate better the tree canopy when picking.

The Panda Hand gripper used in the interface offers a very limited picking reach and surface area with its standard short fingers. It is capable of picking the lemons only along one of their smallest dimensions, as shown in Figure 5.10:

Figure 5.10: The only possible grasping direction on the `Panda Hand` gripper

More tailored, customized, and possibly softer solutions can be designed and implemented to ease the grasping procedure and thus further increase the achievable Success Rate.

Another strategy could involve estimating the approximate principal dimensions of each fruit to be picked. By identifying the pose of the smallest dimension in space, the gripper could be properly aligned to always grasp the fruit using the optimal orientation.

Eventually, a notable gap in the overall realism of the experimental benchmark is the manner in which the lemons were hung on the plant.
In nature, fruits are directly attached to the tree branches, allowing for simple yet precise combinations of rotation and pulling motions to detach the fruit.
In contrast, the robot required significantly more force to pull the grasped fruit free from the branch due to the resistance of the entangled leaves.

6 | Conclusions and future developments

The results obtained in this thesis work demonstrate the power of computer vision and AI when applied to agricultural robotics.

The highly dynamic research in these fields has revolutionized robotic perception in complex environments, such as fields and orchards. The technological readiness of automatic fruit picking with robotic systems will make significant strides forward thanks to increasingly precise understanding and manipulation of the surroundings.

Specifically, detection has proved to be sufficiently ready and reliable to guide the picking process toward the target fruits.

Further efforts must be made to understand the surroundings, detect, and dynamically monitor obstacles to enhance resilience against external disturbances.

In this context, the simultaneous presence and collaboration of robotic and human labor in the fields will require ad-hoc safety protocols of the highest level. An active perception system is essential to maintain economically feasible operational picking times. Dynamically adjusted collision-free motion planners must be implemented to translate sensor data into feasible control. Additionally, optimality must be pursued through the proper design of the controller to minimize operational time.

Regarding 3D environment reconstruction, further improvements can be achieved in both the training and post-processing phases.

For instance, for each cluster of the lemons' point cloud, an approximate spheroid can be computed, utilizing its centroids during the picking phase as an additional layer of information for the 2D-based detection. Particular attention must be given to the reference system used in the 3D reconstruction and its relative position concerning the robot.

Furthermore, since the images are captured by a robot-mounted camera, the estimated camera position and orientation can be directly retrieved from the robot state, bypassing

the Structure from Motion (SfM) processing phase of the images. Careful attention must be given to recreating the expected COLMAP format, including the sparse point cloud and the correct linkage between images and associated camera positions.

Another potential area for advancement is the integration of semantic reconstruction alongside color-based reconstruction. As demonstrated with NeRF ([31]), the 3DGS method can also be expanded to achieve a semantic representation of the reconstructed point cloud. Several recent publications are exploring how to bridge the semantic layer with powerful segmentation tools such as SAM ([25]) and SAM2 ([39]).

These techniques, particularly zero-shot modifications like Grounded SAM ([40]), could simultaneously serve 2D and 3D detection and perception, generalizing the inference process input to a simple text prompt. Implementing more comprehensive semantic dictionaries could link specific picking targets to their physical properties, such as shape, color, or consistency, to tailor the grasping procedure.

In conclusion, the future of agricultural robotics is deeply connected to the advancement of key technologies such as computer vision and AI. The rapid and expansive progress in these fields will increasingly bridge the gap between research and the economic viability of agricultural robotics.

Bibliography

[1] advanced farm. Advanced farm apple harvester, 2025. URL https://advanced.farm/.

[2] G. B. Avanzini, A. M. Zanchettin, and P. Rocco. Constrained model predictive control for mobile robotic manipulators. *Robotica*, 36(1):19–38, 4 2017. doi: 10.1017/s0263574717000133. URL https://doi.org/10.1017/s0263574717000133.

[3] C. W. Bac, E. Van Henten, J. Hemming, and Y. Edan. Harvesting robots for high-value crops: State-of-the-art review and challenges ahead. *Journal of Field Robotics*, 31(7), 2014. doi: 10.1002/rob.21525.

[4] E. Barnea, R. Mairon, and O. Ben-Shahar. Colour-agnostic shape-based 3D fruit detection for crop harvesting robots. *Biosystems Engineering*, 146:57–70, 2 2016. doi: 10.1016/j.biosystemseng.2016.01.013. URL https://doi.org/10.1016/j.biosystemseng.2016.01.013.

[5] R. Bogue. Fruit picking robots: has their time come? *Industrial robot*, 47(2): 141–145, 1 2020. doi: 10.1108/ir-11-2019-0243. URL https://doi.org/10.1108/ir-11-2019-0243.

[6] K. Brugger. Labor shortage leaves $13 million in crops to rot in fields, 3 2017. URL https://www.independent.com/2017/06/22/labor-shortage-leaves-13-million-crops-rot-fields/.

[7] X. Cao, H. Yan, H. Zheng-Yan, S. Ai, Y. Xu, R. Fu, and X. Zou. A Multi-Objective Particle swarm optimization for trajectory planning of fruit picking manipulator. *Agronomy*, 11(11):2286, 11 2021. doi: 10.3390/agronomy11112286. URL https://doi.org/10.3390/agronomy11112286.

[8] Capturing Reality. Capturing reality - 3d reality capture software, 2025. URL https://www.capturingreality.com/. Accessed: 2025-02-13.

[9] W. Chen, S. Lu, B. Liu, G. Li, and T. Qian. Detecting citrus in orchard environment

by using improved YOLOV4. *Scientific Programming*, 2020:1–13, 11 2020. doi: 10.1155/2020/8859237. URL https://doi.org/10.1155/2020/8859237.

[10] Z. Chen, X. Lei, Q. Yuan, Y. Qi, Z. Ma, S. Qian, and X. Lyu. Key technologies for autonomous fruit- and vegetable-picking robots: A review. *Agronomy*, 14(10), 2024. doi: 10.3390/agronomy14102233. URL https://www.mdpi.com/2073-4395/14/10/2233.

[11] K. Daniilidis. Hand-Eye calibration using dual quaternions. *The International Journal of Robotics Research*, 18(3):286–298, 3 1999. doi: 10.1177/02783649922066213. URL https://doi.org/10.1177/02783649922066213.

[12] Dogtooth. Strawberry harvesting robots, 2025. URL https://dogtooth.tech/.

[13] I. Emwinghare and A. A. Al-Mallahi. Performance comparison between yolo v4 and mask r-cnn in detecting potato tubers running on post-harvest conveyors. *2023 14th International Conference on Information and Communication Systems (ICICS)*, pages 1–5, 2023. URL https://api.semanticscholar.org/CorpusID:265683764.

[14] C. L. Escalante and T. Luo. Sustaining a healthy farm labor force: Issues for policy consideration. *Choices*, 32(1):1–9, 2017. ISSN 08865558, 21622884. URL http://www.jstor.org/stable/90014639.

[15] FFRobotics. Ffrobotics apple harvester, 2025. URL https://www.ffrobotics.com/.

[16] fieldwork robotics. fieldwork robotics version 1.2 rasberry harvester, 2025. URL https://fieldworkrobotics.com/.

[17] S. Fountas, N. Mylonas, I. Malounas, E. Rodias, C. H. Santos, and E. Pekkeriet. Agricultural robotics for field operations. *Sensors*, 20(9):2672, 5 2020. doi: 10.3390/s20092672. URL https://doi.org/10.3390/s20092672.

[18] K. Goyal, P. Kumar, and K. Verma. AI-based fruit identification and quality detection system. *Multimedia Tools and Applications*, 82(16):24573–24604, 11 2022. doi: 10.1007/s11042-022-14188-x. URL https://doi.org/10.1007/s11042-022-14188-x.

[19] K. Hu, W. Ying, Y. Pan, H. Kang, and C. Chen. High-fidelity 3D reconstruction of plants using Neural Radiance Fields. *Computers and Electronics in Agriculture*, 220:108848, 3 2024. doi: 10.1016/j.compag.2024.108848. URL https://doi.org/10.1016/j.compag.2024.108848.

[20] S. Jiang, P. Qi, L. Han, L. Liu, Y. Li, Z. Huang, Y. Liu, and X. He. Navigation system for orchard spraying robot based on 3d lidar slam with ndt_icp point cloud registration. *Computers and Electronics in Agriculture*, 220:108870, 3 2024. doi: 10.1016/j.compag.2024.108870. URL https://doi.org/10.1016/j.compag.2024.108870.

[21] L. Jidong, Z. De-An, J. Wei, and D. Shihong. Recognition of apple fruit in natural environment. *Optik*, 127(3):1354–1362, 11 2015. doi: 10.1016/j.ijleo.2015.10.177. URL https://doi.org/10.1016/j.ijleo.2015.10.177.

[22] M. Kang, Q. Chen, Z. Fan, C. Yu, Y. Wang, and X. Yu. A RRT based path planning scheme for multi-DOF robots in unstructured environments. *Computers and Electronics in Agriculture*, 218:108707, 2 2024. doi: 10.1016/j.compag.2024.108707. URL https://doi.org/10.1016/j.compag.2024.108707.

[23] E. Kelman and R. Linker. Vision-based localisation of mature apples in tree images using convexity. *Biosystems Engineering*, 118:174–185, 1 2014. doi: 10.1016/j.biosystemseng.2013.11.007. URL https://doi.org/10.1016/j.biosystemseng.2013.11.007.

[24] B. Kerbl, G. Kopanas, T. Leimkuehler, and G. Drettakis. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics*, 42(4):1–14, 7 2023. doi: 10.1145/3592433. URL https://doi.org/10.1145/3592433.

[25] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick. Segment anything. *arXiv:2304.02643*, 2023.

[26] M. O. Lawal. Tomato detection based on modified YOLOv3 framework. *Scientific Reports*, 11(1), 1 2021. doi: 10.1038/s41598-021-81216-5. URL https://doi.org/10.1038/s41598-021-81216-5.

[27] Q. Li and H. Zhu. Performance evaluation of 2D LiDAR SLAM algorithms in simulated orchard environments. *Computers and Electronics in Agriculture*, 221:108994, 5 2024. doi: 10.1016/j.compag.2024.108994. URL https://doi.org/10.1016/j.compag.2024.108994.

[28] G. Lin, Y. Tang, X. Zou, J. Li, and J. Xiong. In-field citrus detection and localisation based on RGB-D image analysis. *Biosystems Engineering*, 186:34–44, 7 2019. doi: 10.1016/j.biosystemseng.2019.06.019. URL https://doi.org/10.1016/j.biosystemseng.2019.06.019.

[29] X. Liu, D. Zhao, W. Jia, W. Ji, and Y. Sun. A detection method for Apple fruits based on color and shape features. *IEEE Access*, 7:67923–67933, 1 2019. doi: 10.1109/access.2019.2918313. URL https://doi.org/10.1109/access.2019.2918313.

[30] J. Luckstead, R. M. Nayga, and H. A. Snell. Labor issues in the food supply chain amid the COVID-19 pandemic. *Applied Economic Perspectives and Policy*, 43(1): 382–400, 9 2020. doi: 10.1002/aepp.13090. URL https://doi.org/10.1002/aepp.13090.

[31] L. Meyer, A. Gilson, U. Schmidt, and M. Stamminger. FruitNeRF: A Unified Neural Radiance Field based Fruit Counting Framework. *arXiv (Cornell University)*, 8 2024. doi: 10.48550/arxiv.2408.06190. URL http://arxiv.org/abs/2408.06190.

[32] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. NERF: Representing scenes as neural radiance fields for view synthesis. *arXiv (Cornell University)*, 1 2020. doi: 10.48550/arxiv.2003.08934. URL https://arxiv.org/abs/2003.08934.

[33] T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022. doi: 10.1145/3528223.3530127. URL https://doi.org/10.1145/3528223.3530127.

[34] E. Navas, R. Fernández, D. Sepúlveda, M. Armada, and P. G. De Santos. Soft grippers for automatic crop harvesting: a review. *Sensors*, 21(8):2689, 4 2021. doi: 10.3390/s21082689. URL https://doi.org/10.3390/s21082689.

[35] Octinion. Rubion strawberry-picking robot, 2025. URL http://octinion.com/products/agricultural-robotics/rubion.

[36] G. Palmieri and C. Scoccia. Motion planning and control of redundant manipulators for dynamical obstacle avoidance. *Machines*, 9(6):121, 6 2021. doi: 10.3390/machines9060121. URL https://doi.org/10.3390/machines9060121.

[37] L. Pan, D. Barath, M. Pollefeys, and J. L. Schönberger. Global Structure-from-Motion Revisited. In *European Conference on Computer Vision (ECCV)*, 2024.

[38] S. Pan and T. Ahamed. Pear Recognition in an Orchard from 3D Stereo Camera Datasets to Develop a Fruit Picking Mechanism Using Mask R-CNN. *Sensors*, 22(11): 4187, 5 2022. doi: 10.3390/s22114187. URL https://doi.org/10.3390/s22114187.

[39] N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma, H. Khedr, R. Rädle, C. Rolland, L. Gustafson, E. Mintun, J. Pan, K. V. Alwala, N. Carion, C.-Y. Wu, R. Gir-

shick, P. Dollár, and C. Feichtenhofer. Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*, 2024. URL https://arxiv.org/abs/2408.00714.

[40] T. Ren, S. Liu, A. Zeng, J. Lin, K. Li, H. Cao, J. Chen, X. Huang, Y. Chen, F. Yan, Z. Zeng, H. Zhang, F. Li, J. Yang, H. Li, Q. Jiang, and L. Zhang. Grounded sam: Assembling open-world models for diverse visual tasks, 2024.

[41] W. . Research. Sweeper peppers harvester, 2025. URL https://www.wur.nl/en/project/sweeper-the-sweet-pepper-harvesting-robot.htm.

[42] Roboflow. Roboflow: Computer vision tools for developers and enterprises, 2025. URL https://roboflow.com/. Accessed: 2025-02-13.

[43] A. Robotics. Ffrobotics apple harvester, 2025. URL http://abundantrobots.com/.

[44] M. Ryan, M. Jansen, J. Nielsen, G. Gruère, J. Antón, and M. Asai. Labour and skills shortages in the agro-food sector. Technical report, 1 2023. URL https://doi.org/10.1787/ed758aab-en.

[45] F. Saeed, J. Sun, P. Ozias-Akins, Y. J. Chu, and C. C. Li. PeanutNERF: 3D radiance field for peanuts. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 6254–6263, 6 2023. doi: 10.1109/cvprw59228.2023.00665. URL https://doi.org/10.1109/cvprw59228.2023.00665.

[46] saga robotics. Thorvald platform, 2025. URL https://sagarobotics.com/.

[47] J. L. Schonberger and J.-M. Frahm. Structure-from-Motion revisited. 6 2016. doi: 10.1109/cvpr.2016.445. URL https://doi.org/10.1109/cvpr.2016.445.

[48] L. A. G. Stuart and M. P. Pound. 3dgs-to-pc: Convert a 3d gaussian splatting scene into a dense point cloud or mesh, 2025. URL https://arxiv.org/abs/2501.07478.

[49] Y. Tao and J. Zhou. Automatic apple recognition based on the fusion of color and 3D feature for robotic fruit picking. *Computers and Electronics in Agriculture*, 142: 388–396, 9 2017. doi: 10.1016/j.compag.2017.09.019. URL https://doi.org/10.1016/j.compag.2017.09.019.

[50] T. A. Technologies. Tevel apple harvester, 2025. URL https://www.tevel-tech.com/.

[51] F. Tosi, Y. Zhang, Z. Gong, E. Sandström, S. Mattoccia, M. R. Oswald, and M. Poggi. How NeRFs and 3D Gaussian Splatting are Reshaping SLAM: a Survey. *arXiv*

(Cornell University), 2 2024. doi: 10.48550/arxiv.2402.13255. URL https://arxiv.org/abs/2402.13255.

[52] Ultralytics. Ultralytics repository, 2025. URL https://docs.ultralytics.com/. Accessed: 2025-02-11.

[53] C. Q. University. Mango-picking robot, 2025. URL https://www.cqu.edu.au/research/impact/worlds-first-auto-harvester-targets-labour-force-issues.

[54] S. G. Vougioukas. Agricultural robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 2(Volume 2, 2019):365–392, 2019. doi: https://doi.org/10.1146/annurev-control-053018-023617. URL https://www.annualreviews.org/content/journals/10.1146/annurev-control-053018-023617.

[55] X. Xiao, Y. Jiang, and Y. Wang. Key Technologies for Machine Vision for Picking Robots: Review and Benchmarking. *Deleted Journal*, 22(1):2–16, 1 2025. doi: 10.1007/s11633-024-1517-1. URL https://doi.org/10.1007/s11633-024-1517-1.

[56] R. Yang, Y. He, Z. Hu, R. Gao, and H. Yang. CA-YOLOv5: A YOLO model for apple detection in the natural environment. *Systems science control engineering*, 12 (1), 1 2024. doi: 10.1080/21642583.2023.2278905. URL https://doi.org/10.1080/21642583.2023.2278905.

[57] L. Ye, J. Duan, Z. Yang, X. Zou, M. Chen, and S. Zhang. Collision-free motion planning for the litchi-picking robot. *Computers and Electronics in Agriculture*, 185: 106151, 4 2021. doi: 10.1016/j.compag.2021.106151. URL https://doi.org/10.1016/j.compag.2021.106151.

[58] T. Yoshida, Y. Onishi, T. Kawahara, and T. Fukao. Automated harvesting by a dual-arm fruit harvesting robot. *ROBOMECH Journal*, 9(1), 9 2022. doi: 10.1186/s40648-022-00233-9. URL https://doi.org/10.1186/s40648-022-00233-9.

[59] Y. Zeng and Z. Wu. Time-optimal trajectory planning based on particle swarm optimization. volume 1, pages 1794–1796, 6 2015. doi: 10.1109/iciea.2015.7334402. URL https://doi.org/10.1109/iciea.2015.7334402.

[60] F. Zhang, J. Gao, H. Zhou, J. Zhang, K. Zou, and T. Yuan. Three-dimensional pose detection method based on keypoints detection network for tomato bunch. *Computers and electronics in agriculture*, 195:106824, 4 2022. doi: 10.1016/j.compag.2022.106824. URL https://doi.org/10.1016/j.compag.2022.106824.

[61] F. Zhang, Z. Chen, S. Ali, N. Yang, S. Fu, and Y. Zhang. Multi-class detection of

cherry tomatoes using improved YOLOv4-Tiny. *International journal of agricultural and biological engineering*, 16(2):225–231, 1 2023. doi: 10.25165/j.ijabe.20231602.7744. URL https://doi.org/10.25165/j.ijabe.20231602.7744.

[62] J. Zhang, N. Kang, Q. Qu, L. Zhou, and H. Zhang. Automatic fruit picking technology: a comprehensive review of research advances. *Artificial Intelligence Review*, 57(3), 2 2024. doi: 10.1007/s10462-023-10674-2. URL https://doi.org/10.1007/s10462-023-10674-2.

[63] J. Zhang, X. Wang, X. Ni, F. Dong, L. Tang, J. Sun, and Y. Wang. Neural radiance fields for multi-scale constraint-free 3D reconstruction and rendering in orchard scenes. *Computers and Electronics in Agriculture*, 217:108629, 1 2024. doi: 10.1016/j.compag.2024.108629. URL https://doi.org/10.1016/j.compag.2024.108629.

[64] S. Zhang, T. Yuan, D. Wang, J. Zhang, and W. Li. Structure Optimization and Path Planning of Tomato Picking Manipulator. 12 2016. doi: 10.1109/iscid.2016.2091. URL https://doi.org/10.1109/iscid.2016.2091.

[65] H. Zhou, X. Wang, W. Au, H. Kang, and C. Chen. Intelligent robots for fruit harvesting: recent developments and future challenges. *Precision Agriculture*, 23(5):1856–1907, 6 2022. doi: 10.1007/s11119-022-09913-3. URL https://doi.org/10.1007/s11119-022-09913-3.

[66] J. Zhou, Y. Zhang, and J. Wang. A dragon fruit picking detection method based on YOLOV7 and PSP-Ellipse. *Sensors*, 23(8):3803, 4 2023. doi: 10.3390/s23083803. URL https://doi.org/10.3390/s23083803.

[67] X. Zhou, X. Wang, Z. Xie, F. Li, and X. Gu. Online obstacle avoidance path planning and application for arc welding robot. *Robotics and Computer-Integrated Manufacturing*, 78:102413, 7 2022. doi: 10.1016/j.rcim.2022.102413. URL https://doi.org/10.1016/j.rcim.2022.102413.

[68] Y. Zhou, Z. Zeng, A. Chen, X. Zhou, H. Ni, S. Zhang, P. Li, L. Liu, M. Zheng, and X. Chen. Evaluating modern approaches in 3d scene reconstruction: Nerf vs gaussian-based methods. *2024 6th International Conference on Data-driven Optimization of Complex Systems (DOCS)*, pages 926–931, 2024. URL https://api.semanticscholar.org/CorpusID:271768823.

[69] T. Zhu, J. Mao, L. Han, C. Zhang, and J. Yang. Real-Time dynamic obstacle avoidance for robot manipulators based on cascaded nonlinear MPC with artificial potential field. *IEEE Transactions on Industrial Electronics*, 71(7):7424–7434, 8

2023. doi: 10.1109/tie.2023.3306405. URL https://doi.org/10.1109/tie.2023.3306405.

A | Appendix

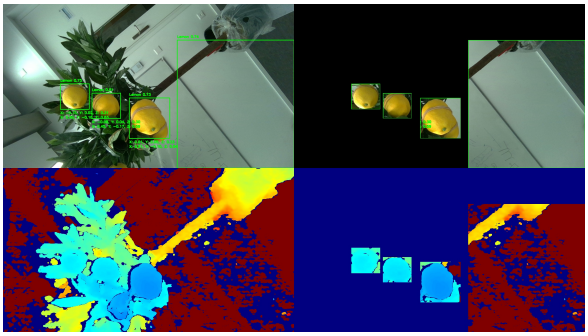
Here, some additional interesting insights regarding the results are reported:

- The reconstructed point clouds from the real–world d405 camera are shown in Figure A.2.
- The additional inference outputs before (Picking) and after (Picked) are shown in Figure A.2.
- The circular trajectory carried out by the robot during the detection pipeline and the "detect all–pick all" picking strategy is shown in Figure A.3.

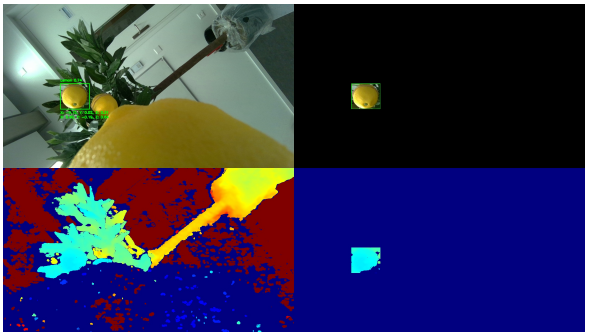
Figure A.1: Reconstructed point clouds from the real–world d405 camera streams

(a) Picking

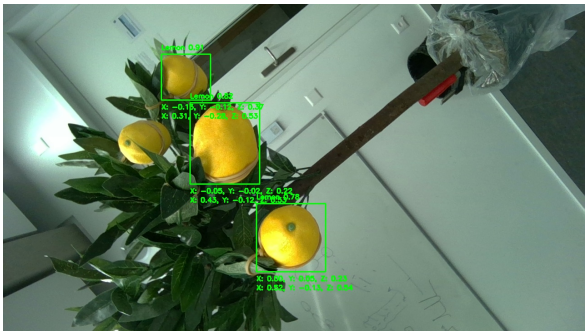
(b) Picked



(c) Picking

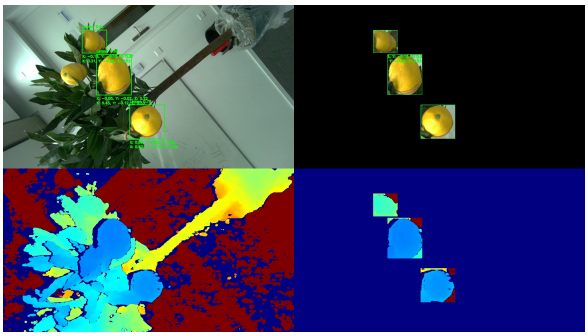


(d) Picked

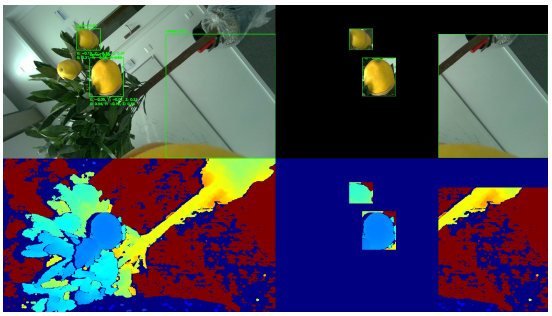


(e) Picking

(f) Picked

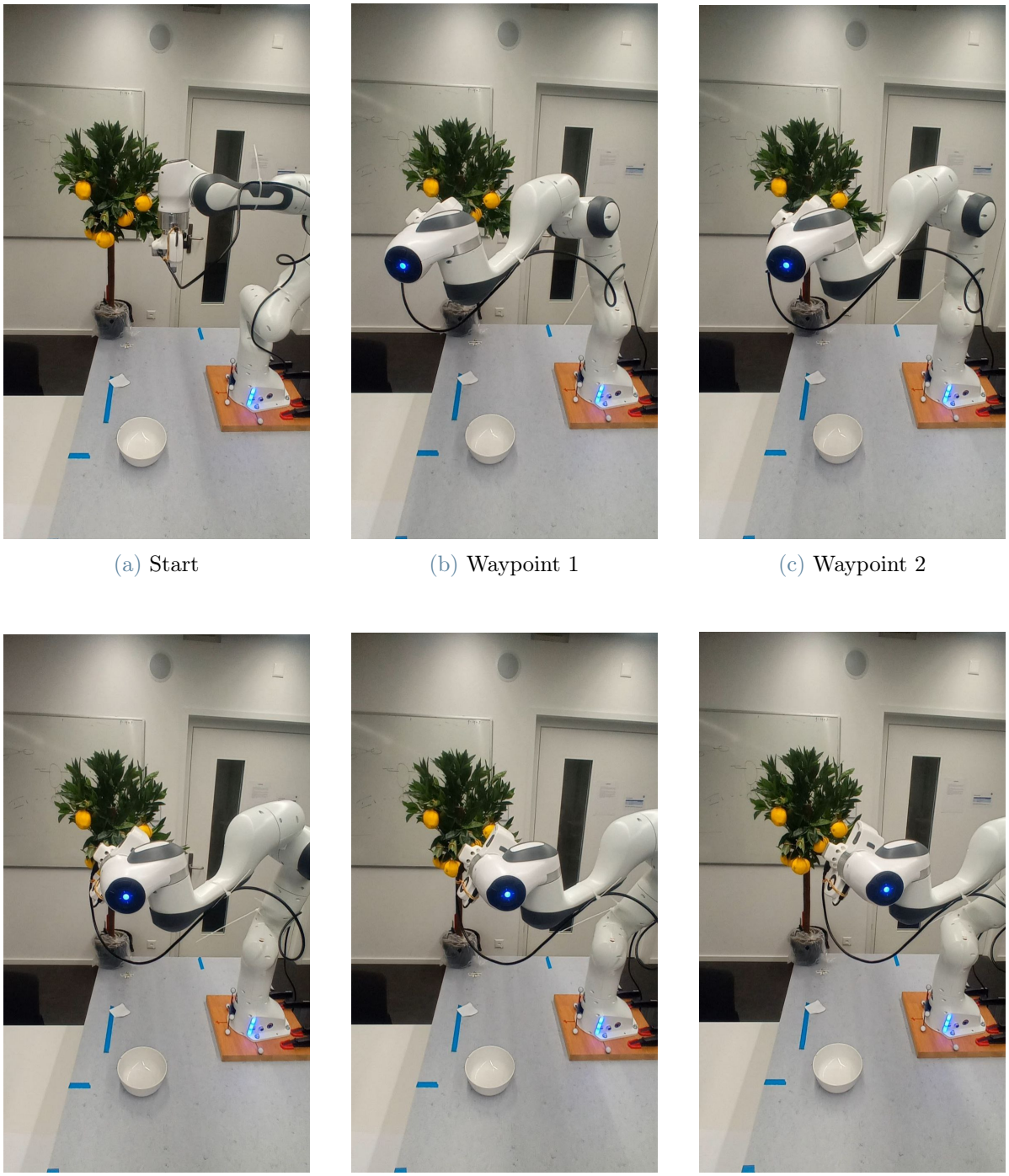


(g) Picking



(h) Picked

Figure A.2: Visualized detections on the streamed RGB and depth images of the proposed model before and after the grasping



(a) Start (b) Waypoint 1 (c) Waypoint 2

(d) Waypoint 3 (e) Waypoint 4 (f) Waypoint 5

Figure A.3: “detect all–pick all” strategy detection phases in the real environment

List of Figures

List of Tables

List of Symbols

Variable	Description	SI unit
f_x	focal length in the x-axis	m
f_y	focal length in the y-axis	m
c_x	principal point in the x-axis	px
c_y	principal point in the y-axis	px
K	camera intrinsic matrix	-