POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

3D Fruit Shape Reconstruction for Autonomous Harvesting

Supervisors

Prof. Marcello CHIABERGE

Dott. Alessandro NAVONE

Candidate

Greta TUMIATTI

March 2026

Abstract

This thesis presents the development of a method for automatic apple size estimation and orientation, needed in the context of automated harvesting. The reason for the increasing investment of time and resources in the design and creation of machines for autonomous agriculture is that their use reduces food and resources waste, improves productivity, lowers production costs. State of the art solutions examined during the first steps of the research focus on creating an accurate estimation of the shape of the original object with the use of deep learning models. However, these approaches are not suitable for real-time use in the fields, since their latency would not be helpful in increasing productivity. Additionally, the 3D mesh reconstruction process is computationally expensive and it is difficult to obtain good results in occluded scenes.

To address these limitations, this thesis presents a methodology based on a trade-off between real-time performance and geometric accuracy. The core approach involves approximating the shape of an apple to a sphere fitted on the fruit, whose diameter and pose will replicate the ones of the original apple. The proposed pipeline was created with the aim of being precise and fast. The developed system employs two distinct neural network models. The first is a neural network for instance segmentation to identify and isolate the apples, while the second is a pose estimation model to detect the critical key points, the base of the fruit and the stem. The intrinsic parameters and depth information obtained from a RGB-D camera are used to obtain the metric estimation of the radius of the apple, by mapping 2D pixel coordinates and depth data into the 3D space.

The performance of the trained models was evaluated using standard computer vision metrics, confirming that the system provides a reliable solution for the original purpose, even in occluded environments, while maintaining a reasonable inference speed. The correctness of the diameters calculated was then validated by comparing the results with measurements from a ground truth within a small tolerance, confirming precise results for the dimension estimation.

Acknowledgements

I would like to thank Professor Marcello Chiaberge for the opportunity to write my thesis at PIC4SeR, an environment where I could learn much about my field of study.

I would also like to thank Alessandro Navone and Luigi Mazzara for all the help and time they dedicated to my thesis. Their advice was fundamental in improving the quality of my work and will certainly be valuable for my professional future.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

This thesis is developed in the context of robotics and artificial intelligence for agriculture.[1] The reason for the increasing investment of time and resources in the design and creation of machines for autonomous agriculture is that their use reduces food and resource waste, improves productivity and reduces production costs.

Agricultural operations are often labor intensive and repetitive. To address this, autonomous platforms provide a way to maintain productivity without depending on manual human effort. Furthermore, robots can work for longer amounts of time compared to humans, allowing for the completion of urgent tasks meeting the required deadlines, for example, the harvesting of mature crops.

Common applications for robotics and artificial intelligence in agriculture include:

- **Weeding**: this is the most common application, since this task is repetitive and needs a considerable effort. It is considered one of the most challenging and costly tasks. This task can be addressed with two different methods:
  - **Mechanical Weeding**: consists of removing weed plants usually by plucking them out or cutting them.
  - **Chemical Weeding**: in this case, the weeds are sprayed with herbicides.
- **Seeding**: focus on accuracy of the placement of seeds, making sure they are planted at the ideal depth and distance to maximize yield.
- **Disease and insect detection**: with the use of sensors and artificial intelligence models, diseases and insect detection are quickly discovered and farmers can intervene and avoid relevant economic impact. The three major challenges in the detection of diseases are:

  - it is difficult to find databases containing images of the required diseases to train the detection models;
  - the speed of image processing is not optimal;
  - the varying lighting conditions in the field.

- **Crop scouting**: evaluation of the state of the plant. It can be divided in:
  - **Plant Vigor Monitoring**: checks the health and stress of the plant using visible or, with the help of sensors, also non-visible symptoms.
  - **Phenotyping**: checks the growth of the plant and the features that are helpful in the performance of the crop.

- **Spraying**: sensors select the specific areas that need spraying in real-time, minimizing the chemical exposure of humans and the environment.

- **Harvesting**: recognition and collection of crops. Robots used in this task can be bulk harvesters when every crop is collected and selective harvesters when the fruits need to meet some requirements to be harvested. This process is delicate and faces challenges when the view of the fruit is obstructed or because the fruits are sensitive and can easily be damaged.

- **Plant Management**: plant relocation, pruning and thinning.

- **Multi-Purpose Systems**: multi-purpose robots that are used for various tasks like seeding, weeding and others, in order to lower costs while maximizing production.

Robotic platforms are usually more compact and lighter than most farm machinery and this helps minimize damage to the soil integrity.

Robots can work on pre-programmed deterministic tasks. Artificial intelligence is needed to obtain reactive behaviors, handling unpredictable conditions in open fields, for example detecting obstacles.

1.2 Objectives of the Thesis

The task created and described in this thesis is needed in the context of autonomous apple harvesting. The pipeline that was envisioned at a high level to follow this structure:

- detection and isolation of individual apples;

- estimation of the apple's size;

- estimation of the apple's pose.

The system was intended to be fast, compact and precise. It was recommended for the method to be effective also with partially occluded apples, as well as with low resolution images resulting from the robot's movement and different lighting conditions. Finally, the solution had to be integrated in a ROS 2 Humble [2] pipeline, where the starting point would be the RGB-D streams published by an Intel RealSense [3] camera.

1.3 Thesis Outline

The thesis is structured in the following way:

- **State of the Art**: this chapter examines the options that were considered during the research for the purpose of the thesis. Photogrammetry is briefly described to assess its possible utility in the creation of a ground truth mesh. Additionally, four different deep learning models for 3D reconstruction are carefully described to highlight the research gaps that need to be addressed. Furthermore, different papers that use RGB-D based dimension estimation and YOLO models are listed and described to show their relevance and explain why these are the best choices to solve the main problem.

- **Theoretical Foundations**: this chapter describes the theoretical concepts necessary for the comprehension of neural networks for instance segmentation and pose estimation and provides a description of the evolution of different versions of YOLO. Finally, an overview of projective geometry is provided.

- **Instance Segmentation and Pose Estimation Tasks**: description of the datasets used to train the YOLO models and the augmentations used to improve generalization.

- **Methodology for Apples Radius Estimation**: this chapter describes in depth the most significant section of the code used in the proposed solution, for the estimation of the dimensions of the apple and the estimation of the pose of the fruit.

- **Experimental Results**: this chapter examines the performance of the YOLO models and of the radius estimation, with the help of computer vision metrics, graphs and output images.

- **ROS Package Design**: description of ROS 2 [2] and of the RGB-D Intel RealSense Camera [3], followed by a detailed analysis of the ROS 2 package developed, containing the logic useful for the main purpose.

- **Conclusion**: summary of the entire workflow, analysis of the results and description of possible improvements.

Chapter 2

State of the Art

2.1 Size and Shape Estimation of Objects

In this section a description of the methods studied in the first phases of the research is provided.

2.1.1 Photogrammetry and Structure from Motion

Initially, photogrammetry was considered for the creation of the ground truth point cloud of the apple.

Photogrammetry can be described as a technique used for the construction of 3D representations of an object by integrating photographs captured from multiple perspectives [4]. The workflow typically consists of three principal phases: the acquisition of photographs, the generation of the 3D model with the help of a specialized software and a post-processing refinement stage. This approach is considered valuable for its portability, cost-efficiency, and because it produces realistic textures in models.

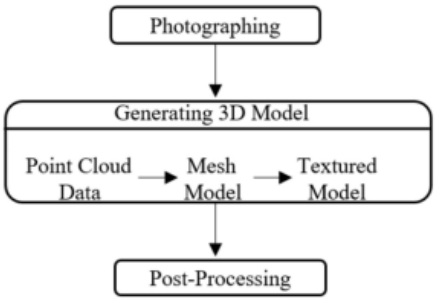


Figure 2.1: Typical Pipeline in Photogrammetry [4].

Structure from Motion is an important technique used in photogrammetry to derive a 3D geometry with the use of sequences of 2D images. This method reconstructs the 3D structure of the object and, at the same time, retrieves the movements of the camera. This is done with the analysis of a sequence of overlapping photographs. The process is equivalent whether the camera moves around the static object or the camera is stationary and the object rotates on a platform. In the paper, it is suggested to use the second method because it ensures that the direction of the light and the background do not change, avoiding unexpected shadows.

During the experiments, the overlap between images was significant, exceeding 80%. The object should be ideally close to the camera and a better resolution is proved to increase the number of faces and vertices of the reconstruction, ensuring a more detailed result. The efficiency of the software for the generation of the mesh is heavily influenced by lightning conditions in the first steps of the process, and can lead to a failure even before the modeling phase.

The study examined offers an analysis of 3D photogrammetry applied to small-scale objects, highlighting how different materials, geometries and photographic parameters affect the resulting 3D models. A platform was used for the rotation of the objects while the camera was fixed.

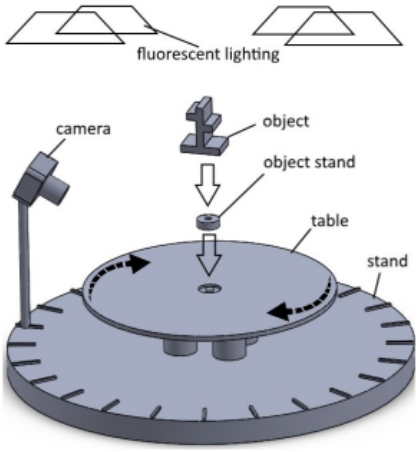


Figure 2.2: Setup for the rotation of objects [4].

The study was conducted in a room with the use of a fluorescent lighting and a dark, anti-reflective fabric background, to avoid shadows or reflections. The surface characteristics of the object were determinant in the quality of the reconstruction. Textiles, clay, ceramic and organic objects, for example, led to high-precision models.

Rough textures are, in fact, ideal for the task of 3D photogrammetry and their

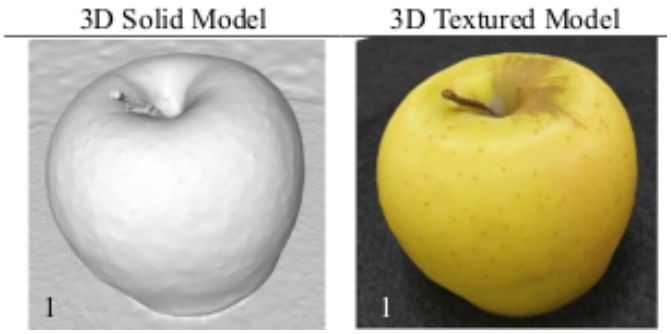


Figure 2.3: 3D reconstruction of an apple with 3D photogrammetry [4].

reconstruction can overcome the problems caused by complex shapes. Transparent, plastic and metal objects, in contrast, have been proven to cause problems in the reconstruction process unless some techniques were adopted, such as:

- **Coatings**: the application of matte powder or spray is necessary for transparent surfaces and improves geometric accuracy for the reconstruction of metal objects.
- **ISO Adjustment**: metal objects need a customized ISO value based on the intensity of the light reflectance.
- **Physical Labeling**: drawing lines on the edges of objects was helpful in improving the final result.

2.1.2 Deep Learning Models

The second stage of the analysis consists of the description of different Deep Learning Models for 3D reconstruction.

Amodal3R

Amodal3R[5] is a 3D generative model designed to reconstruct realistic 3D geometries of occluded objects.

The model is inspired by amodal completion, the ability of humans to infer the 3D structure of an object based on a single perspective, even when it is partially hidden.

Earlier simpler architectures consisted of 2D amodal completion and 3D reconstruction divided into two distinct and sequential phases, but this led to not optimal results. Amodal3R unifies the operations of reconstruction, completion and occlusion reasoning in the 3D latent space. This model processes the input image with the help of a visibility mask, which highlights the visible pixels of the

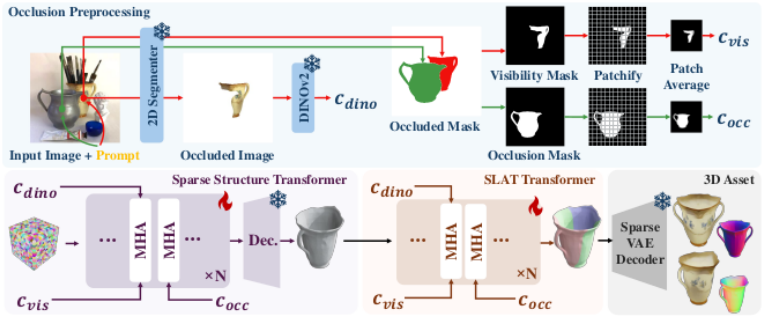


Figure 2.4: Amodal3R Pipeline : input images with the corresponding regions of interests are used to obtain the partially visible target object, the visibility mask and the occlusion masks with the help of a 2D segmenter. DINOv2 identifies the features c_{dino} of the unoccluded parts of the object. Mask-weighted cross-attention c_{vis} and occlusion-aware attention layer c_{occ} are necessary to successfully infer occluded regions by analyzing the visible parts. The regions of the image with a snowflake are not updated from the foundation model TRELLIS, the areas with fires are fine-tuned for the new specific task [5].

target object, and an occlusion mask, which identifies the pixels belonging to the objects blocking the view.

Amodal3R used the pre-existing architecture of the TRELLIS foundation 3D diffusion model and specialized it in amodal reconstruction. The system uses two primary components:

- **DINOv2 Encoder** : it is needed to find visual features in the initial image.
- **VAE Decoder** : it converts 3D mathematical representations into the required output representation, such as meshes.

The original internal parameters of these components are not modified to maintain the quality of 3D representations of the foundation model.

Amodal3R fine-tunes the Sparse Structure Transformer and the SLAT (Structured Latent Variables) Transformer to handle occlusions to predict the fundamental 3D core shape of the object and then refine the details in the geometry and visual appearance. By updating these layers, it is possible to process new information, such as the visibility mask and the occlusion mask.

The model's most important advancements are the Mask-Weighted Cross-Attention and the Occlusion-Aware Attention Layer.

Mask-Weighted Cross-Attention is a layer needed to focus the attention of the model on the observable sections of the object. This is done using the visibility mask that filters out irrelevant areas and identifies the details of the texture of the target object.

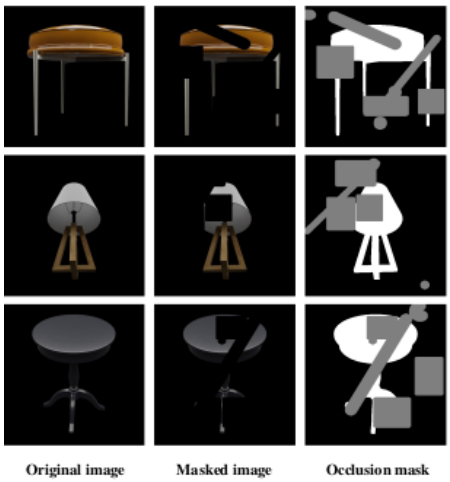


Figure 2.5: Example training images with occlusions in Amodal3R - Supplementary Material : the first column contains the original images, the second shows the occlusions and the third the visible areas of the target in white and the occlusions in gray, while the background remains black [5].

In contrast, the Occlusion-Aware Attention Layer is designed to infer the characteristics of occluded regions. It is necessary for the model to be able to separate the background and the occluder and to establish the parts of the target object where the completion operation needs to take place.

TripoSR

TripoSR[6] is a 3D reconstruction model designed for the reconstruction of high-quality 3D meshes from a single input RGB image.

TripoSR is capable of generating a 3D mesh in less than 0.5 seconds when deployed on an NVIDIA A100 GPU.

TripoSR is based on the Large Reconstruction Model (LRM) framework and is organized into three core components:

- **Image Encoder**: employs a pre-trained DINOv1 vision transformer that projects the input image into latent vectors encoding global and detailed local features.
- **Image-to-Triplane Decoder**: the latent vectors are converted into a triplane NeRF representation. This compact format is ideal for elaborate shapes and textures.
- **NeRF Model**: this is a sequence of multilayer perceptrons that infer the color and density of any 3D point in space.

Figure 2.6: Examples of Results of Amodal3R - Supplementary Material [5].

Figure 2.7: Example outputs TripoSR [6].

TripoSR introduces different key advancements that are not in the original Large Reconstruction Model:

- **Camera Parameter Estimation**: the model does not require exact camera metadata because it estimates intrinsic and extrinsic parameters. This feature

makes it effective for real-world photographs that are not associated with camera parameters.

- **Training Data**: the model was trained on a refined subset of the Objaverse dataset that reflects real-world image distributions thanks to different rendering techniques.

- **Optimization**: the triplane representation was optimized to 40 channels to provide high-quality reconstructions and a manageable GPU memory footprint.

- **Mask Loss**: a mask loss function is included in the training to substantially decrease the number of "floater" artifacts and increase the accuracy of the reconstruction.

- **Importance Sampling**: during the training phase, the model focuses on foreground regions, capturing more details of the surface.

The number of channels is a deciding factor in controlling the GPU memory consumption. The transformation of triplanes into a 3D structure is computationally expensive and a higher number of channels impacts the amount of memory used during the training and the inference.

Based on experimental results, it was determined that 40 channels provide the optimal compromise between performance and resources management so that the model can be trained with larger batch sizes and higher image resolutions with reasonable memory requirements. High-resolution rendering is essential for the model to provide an accurate shape reconstruction. At the same time, high resolution can be challenging for the GPU to manage. The identification of the optimal number of channels is necessary for the model to produce the final 3D mesh in under 0.5 seconds while maintaining accessible memory requirements.

TripoSR specifically incorporates a Mask Loss function designed with the aim to lower the number of "floater" artifacts. The mask loss compares the rendered mask against the ground-truth mask employing Binary Cross-Entropy, checking that the 3D model's silhouette aligns with the actual shape of the object.

ZeroShape

ZeroShape[7] is a deterministic, regression-based model engineered for the task of zero-shot 3D shape reconstruction from a single image. It was created to address the typical high computational costs in generative modeling while demonstrating that a regression-based approach achieves highly competitive performance.

ZeroShape estimates 3D shapes in a single forward pass. The architecture is built following these technical strategies:

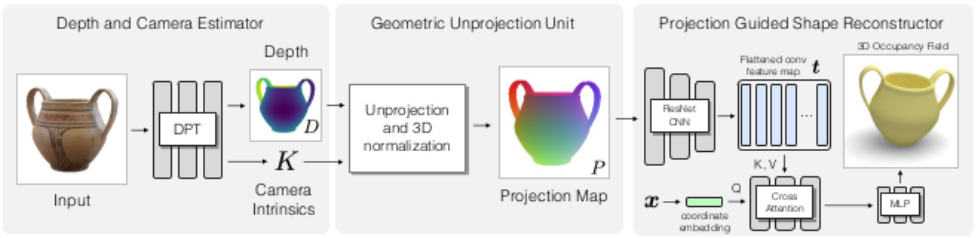


Figure 2.8: Pipeline ZeroShape : a depth and camera estimator is responsible of inferring the depth and camera intrinsics from the input photograph, a geometric unprojection unit transforms the previous estimations into a normalized 3D surface and, finally, a projection-guided shape reconstructor obtains information from the projection map using cross-attention and reconstructs the entire occupancy field [7].

- **Decomposition**: at the start the model estimates the geometry of the visible parts of the object. Based on the first estimate, it predicts the entire 3D shape.
- **Joint Modeling**: the model estimates at the same time depth and camera intrinsics. This is fundamental for higher accuracy.
- **Submodules**: the architecture is composed of a depth and camera, a geometric unprojection module, and a projection-guided reconstructor that synthesizes the final shape as described in the figure 2.8.
- **Representation**: an implicit function determines if one particular 3D point is placed inside or outside the object's surface.

The model has zero-shot ability, since it is trained exclusively on synthetic data from ShapeNet and Objaverse, but it effectively reconstructs a wide variety of real-world objects from categories it has never encountered before.

Pixel2Mesh

Pixel2Mesh[8] is a deep learning framework engineered to reconstruct 3D triangular meshes from a single RGB photograph. Pixel2Mesh utilizes a Graph Convolutional Neural Network (GCN) to represent the 3D geometry directly as a mesh.

Pixel2Mesh operates by progressively reshaping an initial ellipsoid mesh into the requested geometry. This transformation follows a "coarse-to-fine strategy". Initially, the network establishes a stable global structure that will be subsequently refined with local details. The process is organized into three cascaded mesh deformation blocks that progressively enhance the mesh resolution.

Figure 2.9: Example outputs ZeroShape [7].

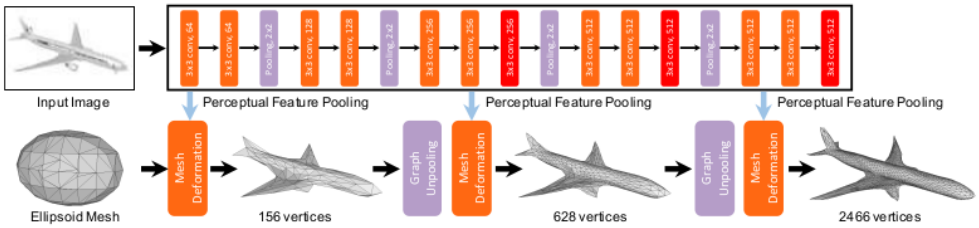


Figure 2.10: The three mesh deformation blocks in Pixel2Mesh are responsible for the improvement of the mesh resolution and for the estimate of the vertex position, necessary to obtain perceptual image features from the 2D CNN that are used as an input for the next block [8].

The key technical components in Pixel2Mesh are:

- **Perceptual Feature Pooling**: this component is useful to associate 2D image data with the 3D geometry. The model projects 3D vertices onto the 2D image place with the use of camera intrinsics and then pools features via bilinear interpolation to direct the deformation.

- **Graph Unpooling Layer**: these layers are used between deformation stages to insert new vertices and edges into the mesh. In this way, the model will capture more intricate surface details while preserving the triangular topology.

- **G-ResNet**: a deep network that integrates shortcut connections needed for an efficient information exchange across the graph of the mesh and successfully addresses the receptive fields limitations in standard graph convolutions.

The model uses the weighted loss of four specialized loss functions to make sure that the generated 3D meshes are physically accurate:

- **Chamfer Distance**: measures the distance from each point in the first set to its nearest neighbor in the second set.

$$l_c = \sum_p min_q ||p - q||_2^2 + \sum_q min_p ||p - q||_2^2$$

It is necessary for the location of the vertices to move as closely as possible to their ground-truth positions.

- **Surface Normal Loss**: ensures consistency by enforcing the edges between a vertex and its neighbors to be orthogonal to the ground-truth normals.

$$l_n = \sum_p \sum_{q=argmin_q(||p-q||_2^2)} ||\langle p - k, n_q \rangle||_2^2, s.t. k \in \mathcal{N}(p)$$

In this formula q represents the closest vertex to p obtained with the chamfer loss, k represents the neighbor pixel of p and n_q is the surface normal from the ground truth. Fundamentally, it requires the normal of a locally fitted tangent plane to align with the object, ensuring a smoother surface and capturing finer local details.

- **Laplacian Regularization**: this operator monitors the relative positioning between a vertex and its neighbors. During the training, it constrains vertices from moving independently, encouraging neighboring vertices to undergo the same displacements. At first, it is necessary to compute the laplacian coordinate for the vertex p

$$\delta_p = p - \sum_{k \in \mathcal{N}(p)} \frac{1}{\mathcal{N}(p)} k$$

Then the formula for laplacian regularization is

$$l_{lap} = \sum_p ||\delta'_p - \delta_p||_2^2$$

where δ'_p and δ_p represent the laplacian coordinate of the vertex after and before the deformation block, respectively. This is critical to avoid mesh self-intersection and to preserve the mesh's original topology during deformation, since the previous losses often remain stuck in a local minimum.

- **Edge Length Regularization**: computes and penalizes the distance between connected vertices in the mesh.

$$l_{loc} = \sum_p \sum_{k \in \mathcal{N}(p)} ||p - k||_2^2$$

It is needed to prevent outliers and encourage a uniform distribution of vertices, achieving a high recall.

The model employs a VGG-16 neural network architecture as its backbone to process the input 2D image and capture high-level perceptual features.

To link the 3D geometry with the 2D visual data, the system follows these steps:

- **3D to 2D Projection**: for every vertex in the 3D mesh, the model computes its corresponding coordinate on the 2D image plane with the use of camera intrinsics.
- **Bilinear Interpolation**: The model uses bilinear interpolation to obtain data from the four nearest pixels in the VGG-16 feature maps. This results in a more accurate sampling of the image data integrated in 3D shapes.

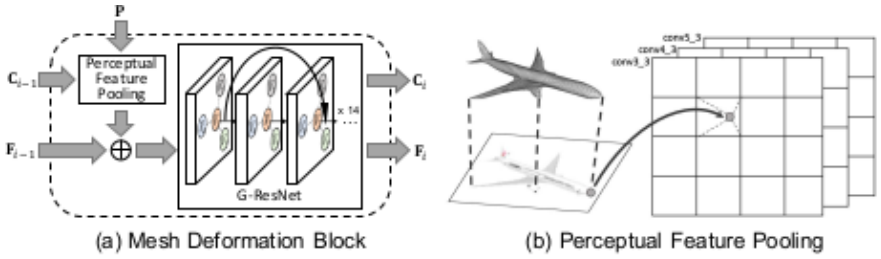


Figure 2.11: In Pixel2Mesh in the mesh deformation block the positions of vertex C_i are necessary for the extractions of the features in photographs, that consequently combined with the vertex features F_i and used as input in G-ResNet. In the Perceptual Feature Pooling the 3D vertices are mapped to the image plane with the help of camera intrinsics and bilinear interpolation is used to extract perceptual features from the 2D-CNN [8].

The image features that were collected are concatenated with the existing 3D features of the vertex. This combined information is then processed by a Graph Convolutional Neural Newtork (GCN). The image features are used as a set of instructions that provide the GCN with the necessary information on what the object looks like from that specific perspective. The network uses this guide to deform the initial 3D ellipsoid progressively until it aligns with the target geometry of the object in the image.

Final Comparison

Regression Models such as Pixel2Mesh[8], ZeroShape[7], and TripoSR[6] map 2D pixels to 3D space in one mathematical step.

The "one-shot" concept, or zero-shot generalization, is the defining goal of different modern 3D reconstruction models. It describes a model's ability to use as an input a single image of an entirely new object and instantaneously generate

Figure 2.12: Example outputs Pixel2Mesh [8].

Table 2.1: Deep learning models inference time

Model	Reported Inference Time
Amodal3R[5]	< 10.0 s
TripoSR[6]	< 0.5 s
ZeroShape[7]	< 5.0 s
Pixel2Mesh[8]	$\sim$15.58 ms

a complete 3D model without optimizations specific to an object or fine-tuning on a particular category. One-Shot Models use a feed-forward approach. These models, such as ZeroShape, TripoSR, and Pixel2Mesh, reconstruct the 3D shape of an object in a single forward pass through a neural network. They learn 3D shape priors from vast synthetic datasets such as Objaverse or ShapeNet. Once trained, the model knows the general principles of geometry, allowing it to generalize and reconstruct unfamiliar shapes in a single pass. One-shot or zero-shot regression models are engineered for maximum efficiency and instantaneous deployment.

Generative models such as Amodal3R[5] employ a different technical approach to the one-shot concept. While the experience for the user remains identical, with one input image and one 3D result, the internal process is generative. It is built on a foundation 3D generator (TRELLIS) and is fine-tuned to address occlusions. Earlier one-shot models assumed that objects were fully visible, while Amodal3R performs amodal 3D reconstruction, imagining and completing hidden parts of an object directly in the 3D latent space.

This fine-tuned regression involves adapting a robust, existing "foundation" model to a more specialized task. Rather than training a model from scratch, Amodal3R starts with a foundation 3D generator like TRELLIS. The strategy focuses on fine-tuning specific layers to handle new information, such as visibility

and occlusion masks. To preserve the foundation's extensive knowledge and high-quality 3D representation, other components such as the VAE Decoder and DINOv2 image encoder are not changed. The goal is to integrate specialized logic into the model for specialized tasks without extensively modifying the pre-trained model.

2.2 Research Gap

2.2.1 Trade-off between Speed and Precision

The previously analyzed methods show a significant challenge in finding a balance between computational speed and reconstruction accuracy. Specifically, the tested Deep learning models focus on the precision of the apple's geometry, resulting in a latency that is incompatible with our requirements.

Amodal3R[5] has an inference time of less than 10 seconds per instance, which is not suitable for high-speed real-time tasks. ZeroShape[7] processes images in less than 5 seconds, as shown in the figure 2.13.

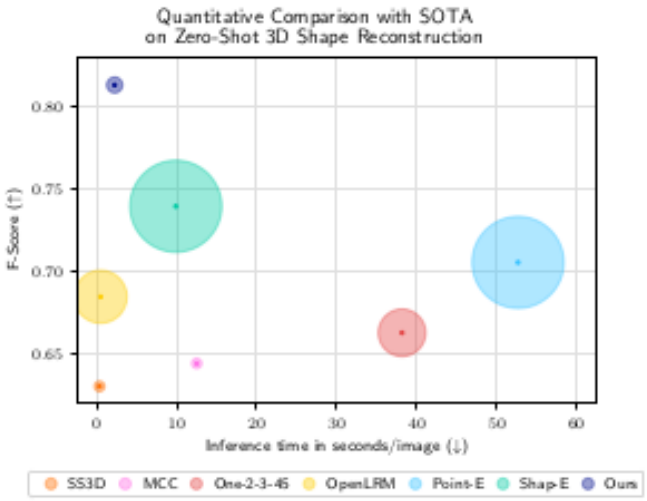


Figure 2.13: In this graph in the ZeroShape paper, it is possible to observe the inference time of the model, represented by the purple circle, described as "ours" [7].

Although it is much faster than Amodal3R, it remains impossible to use it in real-time contexts. TripoSR[6] and Pixel2Mesh[8] are the only instantaneous solutions. TripoSR generates a mesh in under 0.5 seconds and Pixel2Mesh is even faster, registering 15.58 ms for the inference during testing.

The research gap is the lack of a solution that combines a sufficiently accurate reconstruction for a safe harvest, while maintaining a real-time responsiveness.

2.2.2 Occlusion Challenge

For the specific task of reconstructing apples partially hidden by leaves, TripoSR[6] and Pixel2Mesh[8] face significant limitations. These models assume objects do not have a specialized amodal reasoning to complete the apple with the hidden parts. If an apple is occluded by a leaf, these models will likely create an incomplete mesh. In contrast, Amodal3R[5] and ZeroShape[7] have been designed considering the occlusion challenge. Amodal3R uses mask-weighted cross-attention and occlusion-aware attention layer, that were previously described. ZeroShape addresses the task with a decomposition strategy that consists of the estimation of the 3D visible surface followed by the prediction of the complete shape.

2.2.3 3D Mesh Creation

Another critical decision was whether to recreate the precise shape of the apple or simply use a fitted sphere.

Initially, obtaining the exact shape of the apple seems the best choice, but it requires a significant computational cost that could be saved by simplifying the geometry.

2.2.4 Dimension Estimation

It is essential that the models provide dimensions in centimeters of the apples for the harvesting task. Achieving this depends on how they handle camera intrinsics and how they resolve the scale ambiguity inherent in the 3D reconstruction from a single image.

Among the reviewed architectures, Pixel2Mesh[8] is the only model that explicitly references physical units of measurement in its internal configuration. The model deforms a predefined initial ellipsoid that is defined with specific metric dimensions: it is centered with a depth of 0.8 m facing the camera, with a radius of 0.2 m, 0.2 m and 0.4 m along its three principal axes. Since the model assumes that the correct intrinsic matrix of the camera is known, it can technically maintain a mathematical relationship between the pixels in the image and the 3D metric space.

ZeroShape[7] tries to calculate the camera's properties directly from the image. The model estimates both depth and camera intrinsics (K). During the training phase, it uses images with varying focal lengths to learn how different lenses affect the 2D projection of 3D shapes. However, the final output is not provided in absolute real-world centimeters.

TripoSR[6] is designed to estimate intrinsic and extrinsic camera parameters and the apple's shape will be correctly reconstructed, but the model cannot estimate the precise dimension in centimeters.

Amodal3R[5] does not provide reconstructions in absolute physical units, such as centimeters. Amodal3R represents 3D objects using Structured Latent Variables (SLAT) associated with voxels within a grid. The positions of these voxels are determined by the spatial resolution of the grid, which is a mathematical coordinate system rather than a metric one.

2.2.5 Pose Estimation

To prevent fruit damage during the harvesting process, it is recommended to estimate the pose of the apple. Here is how each model handles object pose:

TripoSR[6] estimates the camera parameters, which include both extrinsics and intrinsics, to place the 3D apple in a coordinate system.

ZeroShape[7] uses a view-centric coordinate system, which means that the apple's pose is always defined relative to the camera's eye. The camera is treated as the center of the world. The model estimates the distance of the apple and the intrinsics of the camera to ensure that the pose of the apple is not distorted. The apple is reconstructed exactly as it appears in front of the lens.

Amodal3R[5] is capable of determining the pose of the apple even when parts of the fruit are occluded, with the implementation of reconstruction and completion in the 3D latent space. It is not required a pre-defined camera pose to start. It uses 3D reasoning to figure out the orientation of the whole apple. Even if only one side of the apple is visible, the model uses its internal knowledge to determine the pose of the entire object.

Pixel2Mesh[8] treats the apple's pose as the final stage of the shape deformation process. It always starts with a standard ellipsoid with a depth of 0.8m facing the camera. The pose of the final apple is achieved by stretching the initial oval to match the target fruit in the image.

2.3 Use of RGB-D Based Estimation in Robotics for Agriculture

In the following section, RGB-D methods for size estimation in agriculture will be examined.

2.3.1 Apple Diameter Estimation

This study[9] introduces a model designed to estimate the maximum diameter of Red Fuji apples to support integrated picking and grading by agricultural robots. The primary objective is to create a real-time estimation method that works in the

unstructured environment of an orchard, where the distance of the object and the camera angles often change.

Initially, the ground truth for the study was established by manually measuring the actual maximum diameter of Red Fuji apples and using an Intel RealSense D435 RGB-D camera to capture images and collect depth data. After this step, it was necessary to synchronize the photos with the physical measurements. Because the camera has separate lenses for color and depth, a Python code was implemented to perfectly align the two data streams. A tool called LabelImg was necessary to manually annotate the apples in the images. This process provides the diagonal length of the bounding box that contains the apple in pixels and the distance from the camera.

The correlation between size of the minimum diagonal length of the bounding box in pixels (x), the distance of the fruit from the camera in centimeters (y) and the real diameter of the apple in millimeters (z) was studied to derive a specific predictive formula to obtain z.

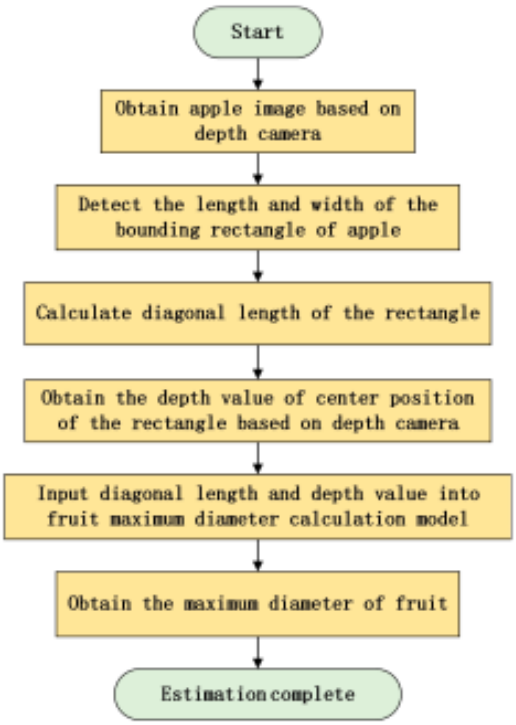


Figure 2.14: Pipeline of the method for Apple Diameter Estimation [9].

The final step involved using YOLOv5 for the detection of the apple. The system finds its distance, and uses the formula to display its diameter instantly. In this way, the robot can simultaneously recognize, locate, and grade the size of

apples in real-time as it moves through the orchard.

2.3.2 Pomegranates Sizing: Ellipse Fitting and 3D Projection

This research[10] employs an agricultural robot to automate the detection, shape estimation and size prediction of pomegranates in orchards.

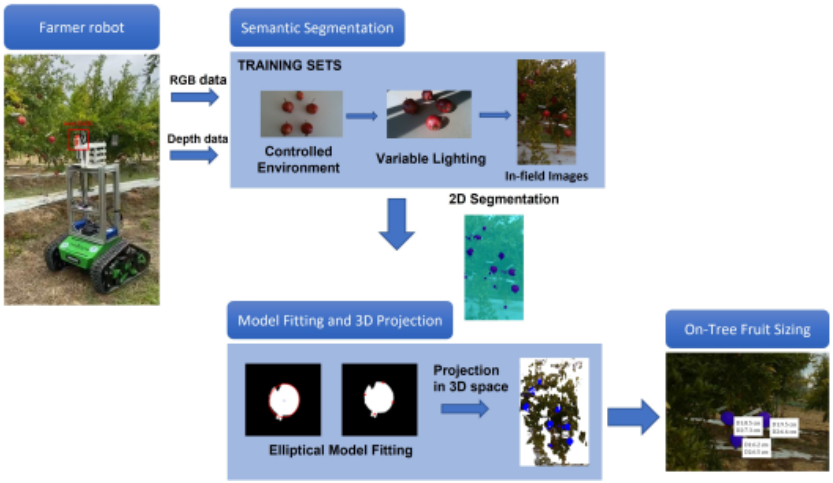


Figure 2.15: Pipeline of the method for Pomegranate Sizing, composed of data acquisition of RGB and depth data, use of a segmentation network for the creation of a 2D segmentation mask, fitting of elliptical models on each instance and projection into 3D space, final estimation of polar and equatorial diameters [10].

The process begins with the use of a semantic segmentation network that identifies fruit and non-fruit sections in RGB images. Connected Component Analysis is applied to identify individual fruits. The semantic segmentation model simply classifies pixels as belonging to a certain class, Connected Component Analysis separates those pixels into individual instances.

Following the identification of individual instances, they are enclosed in ellipses. It is now possible for the system to identify the principal axes, which represent the fruit's real world equatorial and polar diameters. The elongation of the fitted ellipse is evaluated and if it is considered excessive, the system assumes that, for example, the fruit is partially occluded or that presents anomalies in the segmentation. Highly elongated ellipses are filtered out to ensure that only fully visible, non-occluded fruits are processed, otherwise the sample is discarded to maintain data integrity.

The system detects the poles, or vertices, of the 2D ellipse and projects them into the 3D space using synchronized depth. The final result is determined by

computing the Euclidean distance between these two 3D points, that provides the diameter of the polar and equatorial diameters measured in centimeters.

2.3.3 Mango Fruit Size Estimation

This research[11] introduces a monitoring system that was developed to function on agricultural machinery traveling at speeds of 5 km/h.

The detection process begins with a cascade classifier integrated with Histogram of Oriented Gradients (HOG) features, which are effective in the identification of the curved edges of mangoes.

Otsu's method and color thresholding are used to remove background noise such as leaves and branches. Otsu's method is an unsupervised mathematical algorithm that autonomously identifies the optimal threshold for separating an image into two distinct categories and isolates the fruit contour.

Color thresholding was applied in the CIE L*a*b* color space. The system identifies specific numeric ranges in the a* and b* channels, which represent color scales, with the aim of removing pixels that match the typical colors associated with leaves and branches. Subsequently, the system employs a specialized one-dimensional filter designed to remove the fruit stalks. This process is necessary because any noise attached to the fruit's edge in the image would cause the calculation of an incorrect diameter, which will consequently result in an incorrect size estimation.

The model uses the Thin Lens Formula for the conversion of two-dimensional pixel data into three-dimensional physical measurements:

$$\frac{f}{D} = \frac{imagesize}{realsize}$$

where f represents the focal length, D represents the distance between the fruit and the camera, and the image size represents the physical size of the fruit. The image size is computed by multiplying the number of pixels the fruit occupies in the image and the size of the camera's physical sensor.

2.3.4 Observations and Comparison

RMSE, or Root Mean Square Error, is a standard evaluation metric used to quantify the deviation between predicted outcomes and the ground-truth values. It is necessary to express the magnitude of the error, a smaller RMSE value represents more accurate predicted results. It is obtained with the square root of the average of squared differences between actual and predicted values. For example, in the Apple Diameter Estimation paper[9] is defined in this way:

Table 2.2: Comparison of the performance between methods

Paper	Precision Reached	Velocity/Latency
Apple Diameter Estimation[9]	Orchard RMSE: 3.95 mm (±3.77% relative error)	Real-time processing
Pomegranate Sizing: Ellipse Fitting and 3D Projection[10]	RMSE: 1.35 cm (equatorial) and 1.31 cm (polar) and average error of ∼1.0 cm	0.15 s per frame
Mango Fruit Size Estimation[11]	RMSE: 4.9 mm (length) and 4.3 mm (width)	Real-time processing

$$RMSE = \sqrt{\frac{\sum \left(E_{Max_Diameter} - T_{Max_Diameter}\right)^2}{Num_{apple}}}$$

Relative Error quantifies the discrepancy between the estimated value and the ground-truth value and it is typically expressed as a percentage. In the Apple Diameter Estimation paper[9] the formula is as follows:

$$REE_{Max\_Diameter} = \frac{E_{Max\_Diameter} - T_{Max\_Diameter}}{T_{Max\_Diameter}} \cdot 100\%$$

The average error is calculated with Mean Absolute Error (MAE) in the Pomegranate Sizing paper[10] and it is expressed in absolute units (cm/mm). In the Apple Diameter Estimation paper[9], it is computed as the Average Estimation Relative Error (MREE) and it is expressed with percentages:

$$MREE_{Max\_Diameter} = \pm\frac{\sum |E_{Max\_Diameter} - T_{Max\_Diameter}|}{\sum Num_{apple}} \cdot 100\%$$

The results of the RGB-D based methods highlight their efficiency for real-time tasks while keeping a high quality performance in the size estimations. These observations were important for the choice to adopt this approach to solve the main challenge of the thesis.

2.4 Use of YOLO Models in Robotics for Agriculture

2.4.1 Comparing YOLOv8 and Mask R-CNN

This research[12] provides a comparison of the YOLOv8 one-stage detector with the Mask R-CNN two-stage detector for instance segmentation in orchards. YOLOv8

is designed as a one-stage detector that maximizes processing speed and efficiency, which is indispensable for real-time robotic agriculture tasks.

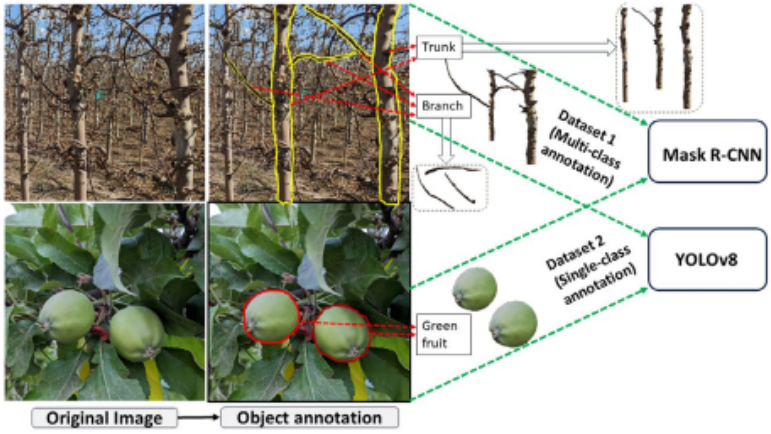


Figure 2.16: The two datasets that were annotated and used for the training of Mask R-CNN and YOLOv8 [12].

Traditional models, such as YOLOv5, used anchors, predefined boxes of various sizes that enclose objects. In contrast, YOLOv8 uses an anchor-free approach and predicts the center of an object directly. This approach shows more flexibility in the results and minimizes the complexity of the model.

Mask R-CNN uses the Region Proposal Network as the first stage. The network scans the entire image to localize Regions of Interest, candidate areas that could potentially contain the target object. Once these areas are identified, the model processes them using two separate concurrent paths or branches. One branch is responsible for bounding box detection, which identifies the object and draws a square around it to provide a general location. Simultaneously, the second branch is specialized in binary mask prediction, which draws the exact shape of the fruit.

YOLOv8 belongs to the "You Only Look Once" family of deep learning models created for tasks, such as object detection and instance segmentation. It is considered a one-stage detector, created to process images in a single pass, with high speed and efficiency.

The most important functionality of YOLOv8 is its ability to perform detection and segmentation in a single forward pass of the network. This end-to-end learning architecture creates a direct transition from raw input images to the final results and this is important for its application in real-time environments.

YOLOv8 includes several architectural advancements that differentiate it from its predecessors:

- **Anchor-Free Detection**: the previous versions of YOLO, such as YOLOv5

and YOLOv7, used traditional anchor-based methods. YOLOv8 has a significantly different approach for object detection, since it is an anchor-free and center-based. This improvement offers more flexibility and accuracy.

- **Decoupled Head**: a decoupled head divides the task of detecting the presence of an object and the task of classifying the object, ensuring more precise predictions and faster processing.
- **Feature Extraction**: YOLOv8 uses Darknet-53, a deep convolutional neural network with 53 layers specialized in feature extraction and object detection.
- **C2f Module**: the C2f module replaces the previous C3 module to improve accuracy and reduce processing times compared to earlier versions.
- **Efficiency Optimizations**: in the backbone 6×6 layers are replaced with 3×3 layers to reduce the total number of parameters and resulting in a "more compact, computationally efficient network".
- **Supervision**: during the training phase, YOLOv8 employs a technique in which multiple models with different configurations analyze the same dataset and generate predictions, increasing the accuracy of the final model.

YOLOv8 is available in multiple configurations, such as:

- **YOLOv8-Tiny**: a version suited for fast processing, that prioritizes a faster execution over accuracy. It is ideal for real-time applications on devices with constraints on resources.
- **YOLOv8-Small**: this configuration balances accuracy and processing speed, ensuring a higher quality of detections compared to YOLOv8-Tiny.
- **YOLOv8-Standard**: this version offers a reliable and consistent performance and can be used in many environmental conditions.
- **YOLOv8-Large**: this configuration prioritizes accuracy and precision and can be used when the detection of details is more important than the processing speed.

Different configurations are needed to balance computational speed and accuracy based on the available resources. YOLOv8 shows a superior inference speed and higher precision compared to many two stage models. However, it could occasionally result in the loss of resolution in segmentation masks.

The acquisition of RGB images was done using an Intel RealSense 435i camera. Each photograph was manually annotated. The training of both models integrated a warm-up phase, translation, scaling, mosaic and different augmentations to

ensure generalization and robustness. The evaluation of both models is based on Precision, Recall, mAP@0.5, and inference speed. YOLOv8 achieved better results in comparison to Mask R-CNN in both Single-Class Segmentation and Multi-Class Segmentation, in both accuracy and inference speed.

Single-Class Segmentation

This task involved the instance segmentation of immature green apples.

Metric	YOLOv8	Mask R-CNN
Precision	92%	84%
Recall	97%	88%
mAP@0.5	0.939	0.902
Inference Time	7.8 ms	12.8 ms
Speed (FPS)	128.21	78.13

Table 2.3: Evaluation of Single-Class Segmentation [12].

Multi-Class Segmentation

This task involved the instance segmentation of trunks and branches.

Metric	YOLOv8	Mask R-CNN
Precision (All Classes)	90.6%	81.3%
Recall (All Classes)	95%	83.7%
mAP@0.5 (Overall)	0.845	0.748
mAP@0.5 (Trunk Class)	0.971	0.828
mAP@0.5 (Branch Class)	0.719	0.673
Inference Time	10.9 ms	15.6 ms
Speed (FPS)	91.74	64.10

Table 2.4: Evaluation of Multi-Class Segmentation [12].

The study also highlights the fact that YOLOv8 is better at excluding background noise.

From the study of this research it is possible to deduce that for the original task of the thesis the YOLOv8-Small version is the best configuration because it balances high processing speed and accuracy.

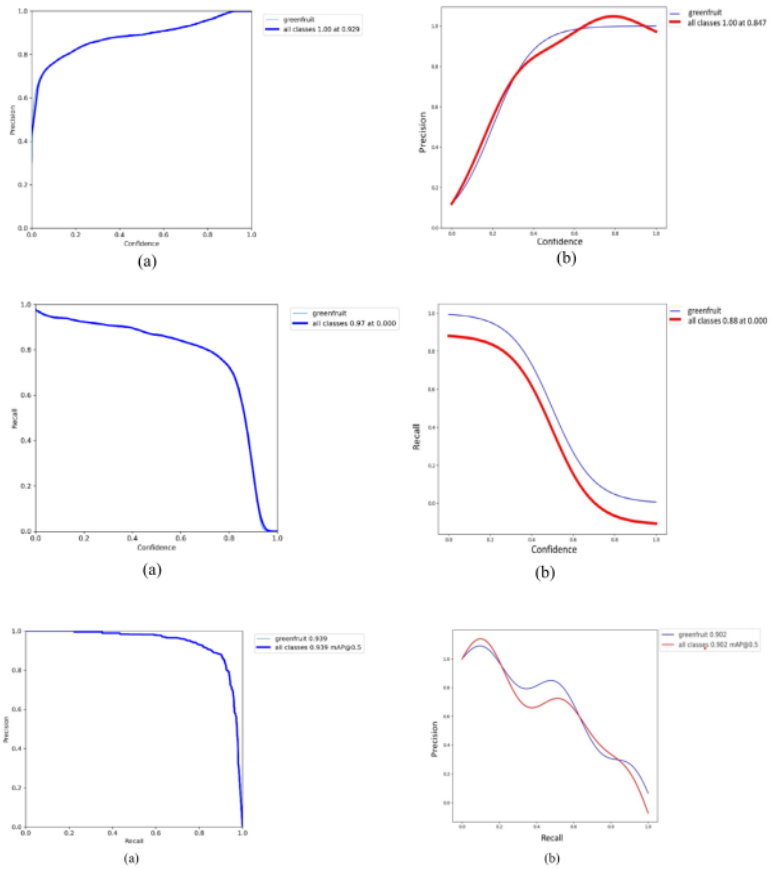


Figure 2.17: Graphs that describe the performance for the task of single-class segmentation. The graphs on the left describe YOLOv8 while the graphs on the right describe Mask R-CNN. In the first row it is shown the Precision-Confidence Curve, in the second row the Recall-Confidence curve and in the third row the Precision-Recall curve at mAP@0.5 [12].

2.4.2 YOLO11 and Vision Transformers for 3D Pose

This study[13] introduces a method for the estimation of the 3D orientation, or pose, of fruitlets, needed for the task of robotic thinning.

The keypoints of the fruit, specifically the calyx and the peduncle, were manually labeled using the Roboflow platform to define the pose of the fruit. By localizing the two distinct points, the model effectively defines a vector that represents the longitudinal axis of the apple.

Multiple configurations of YOLO11 and YOLOv8 were trained for this specific task. Both YOLOv8 and YOLOv11 are suited for real-time processing:

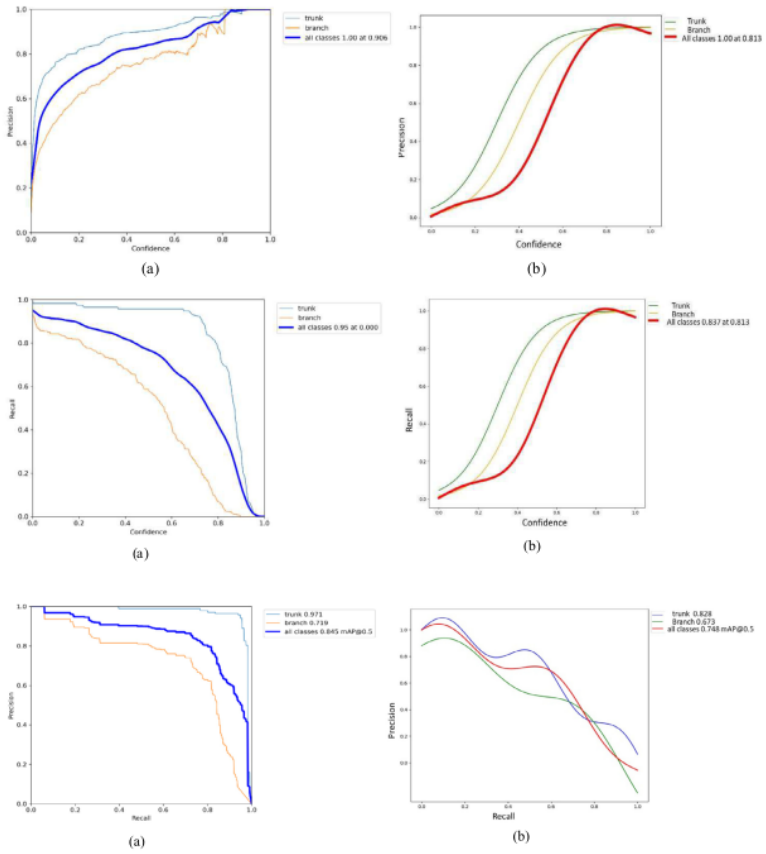


Figure 2.18: Graphs that describe the performance for the task of multi-class segmentation. The graphs on the left describe YOLOv8 while the graphs on the right describe Mask R-CNN. In the first row it is shown the Precision-Confidence Curve, in the second row the Recall-Confidence curve and in the third row the Precision-Recall curve at mAP@0.5 [12].

- **Three-Part Architecture**: both models employ a three-part structure comprising of a Backbone, a Neck and a Head.
- **Three-Phase Pipeline**: the pipeline is composed of three stages:
  - **Pre-processing**: resizing of the resolution of the input, normalization and, finally, augmentations.
  - **Inference**: execution of the object detection and keypoint localization.
  - **Post-Processing**: use of Non-Maximum Suppression for the refining of the results.
- **Keypoint Estimation**: Both architectures can be used for bounding box

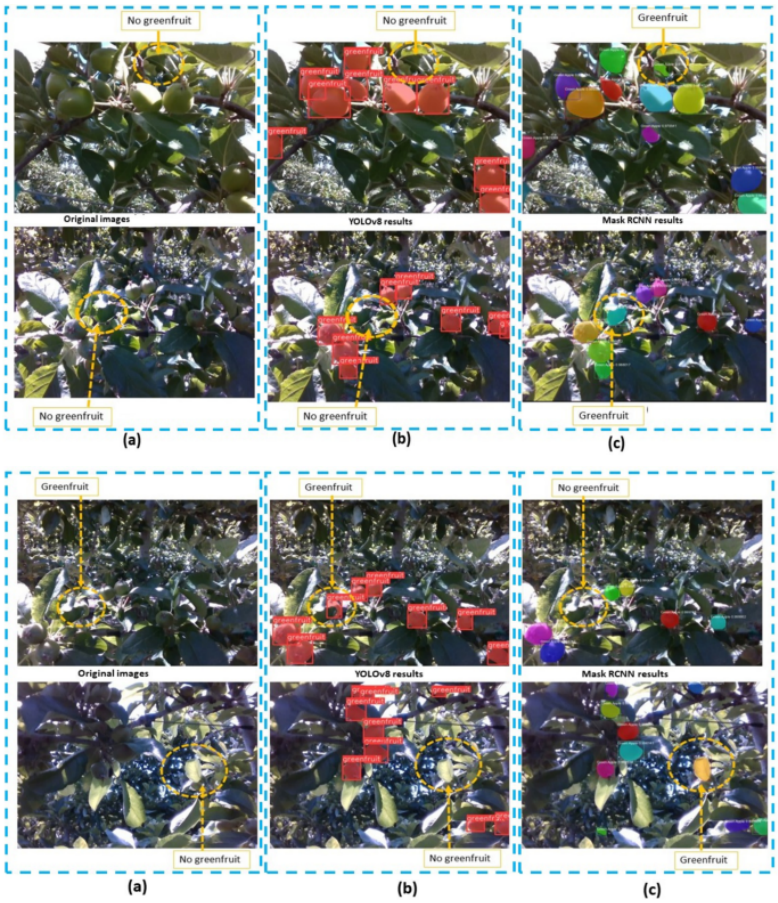


Figure 2.19: Wrong detections of Mask R-CNN that were not replicated by YOLOv8 [12].

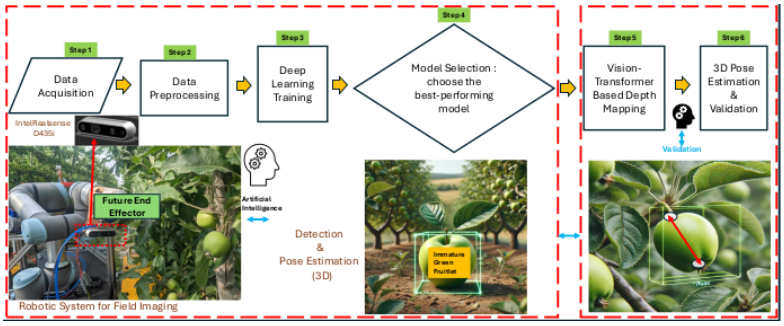


Figure 2.20: Pipeline of the method for the estimation of the 3D pose of fruitlets from that consists of data collection, training of the model and pose estimation [13].

detection and keypoint estimation. A pose vector connects the keypoints and determines orientation of the fruit.

The 2D coordinates of the calyx and peduncle are projected onto the 3D point cloud generated with the use of Vision Transformers.

Vision Transformers are advanced deep-learning models suited for this type of image analysis tasks. Depth Anything V2 is a monocular depth estimation model relying on transformer technology, used for the creation of accurate 3D representations starting from the RGB-D images. In this way, the system can derive the precise 3D spatial coordinates of the keypoints in physical units without the need for a physical depth sensor.

Vision Transformers are used to simulate depth data from monocular images. Subsequently, the system is able to create 3D point clouds without the help of additional information derived from physical depth sensors. After obtaining the 3D spatial context required to validate the poses detected by YOLO, it will be possible to compute the major axis length of a fruitlet in millimeters.

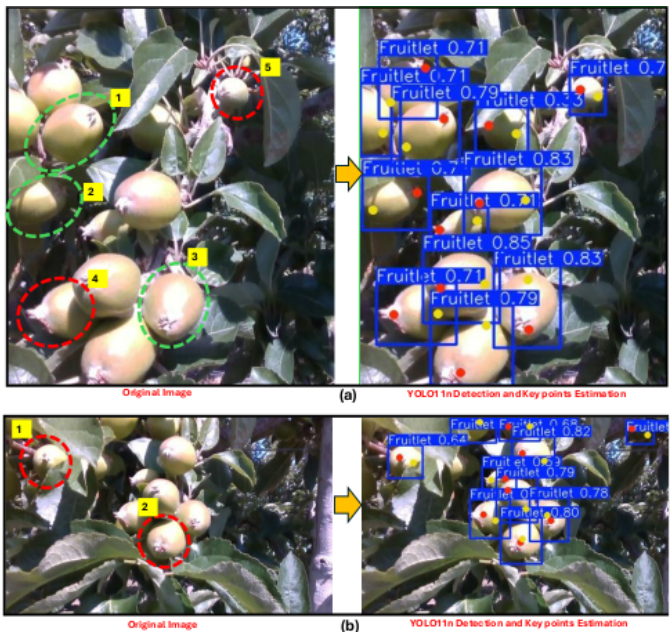


Figure 2.21: Examples images that show that the method performs successfully in occluded environments [13].

In the following section, only the results of YOLOv8s and YOLO11s configuration will be displayed, since in the previous parts of the thesis it was highlighted that the small versions of YOLO are able to obtain a balance between accuracy and

inference speed. The inference speed is calculated as

$$Inference\ Speed = \frac{Total\ Processing\ Time}{Number\ Of\ Images\ Processed}$$

The YOLOv8s configuration showed the following results:

Metric	Result
Box Precision	0.883
Box Recall	0.875
Pose Precision	0.896
Pose Recall)	0.895
Box mAP@50	0.927
Pose mAP@50	0.952
Inference Speed (ms)	5.2

Table 2.5: Evaluation of YOLOv8s configuration [13].

The YOLO11s configuration showed the following results:

Metric	Result
Box Precision	0.891
Box Recall	0.877
Pose Precision	0.899
Pose Recall)	0.891
Box mAP@50	0.940
Pose mAP@50	0.950
inference Speed (ms)	5.3

Table 2.6: Evaluation of YOLO11s configuration [13].

2.4.3 YOLO11 vs. YOLOv8 for Occlusion

This technical note[14] offers a comparative analysis between YOLO11 and YOLOv8 for the specific task of instance segmentation for immature green apples, often hidden by foliage in complex orchard environments. The dataset was manually classified into two distinct categories based on visibility: Non-occluded fruitlets and Occluded fruitlets.

Subsequently, the models were trained using identical parameters for 300 epochs with a resolution of 640×640 pixels. In the following section, only the results of YOLOv8s and YOLO11s configuration will be displayed, since in the previous

parts of the thesis it was highlighted that the small versions of YOLO are able to obtain a balance between accuracy and inference speed.

Metric	YOLO11-seg	YOLOv8-seg
Box Precision (All)	0.832	0.815
Box mAP@50 (All)	0.864	0.855
Mask mAP@50 (All)	0.837	0.840
Parameters (Millions)	10.07	10.48
Inference Speed	6.0 ms	5.1 ms

Table 2.7: General comparison of YOLOv8-seg and YOLO11-seg [14].

Metric	Category	YOLO11-seg	YOLOv8-seg
Box Precision	Non-occluded	0.845	0.816
Box Precision	Occluded	0.819	0.814
Box mAP@50	Non-Occluded	0.893	0.877
Box mAP@50	Occluded	0.835	0.823
Mask Precision	Non-Occluded	0.836	0.812
Mask Precision	Occluded	0.798	0.789
Mask mAP@50	Non-Occluded	0.878	0.885
Mask mAP@50	Occluded	0.795	0.794

Table 2.8: Comparison of YOLOv8-seg and YOLO11-seg for Non-Occluded and Occluded categories [14].

From these results, it is evident that YOLO11s is the best choice when the task requires a higher box accuracy, while YOLOv8s is preferable when a faster inference speed is needed. It is also highlighted how the occlusions affect the quality of the results.

2.4.4 Comparison and Observations

The previous analyzed papers led to the choices of:

- YOLO over Mask R-CNN for segmentation due to its better performances in Precision, Recall, mAP@0.5 and inference time.
- YOLO pose for the task of orientation estimation for fruits.
- YOLOv8 over YOLO11 because in a real-time task such as harvesting, it is necessary to prioritize inference speed over precision.

Chapter 3

Theoretical Foundations

3.1 Definition and Purpose of Computer Vision

Computer Vision[15] can be described as the field of study that aims to determine what it means for a machine to visually perceive the world. We can observe different tasks in Computer Vision, such as[16] :

- **Image Classification** : the process of automatically identifying the category of the depicted object within an image.
- **Object Detection** : the task of identifying and determining the position of one or more objects in a scene and enclosing them in bounding boxes.
- **Semantic Segmentation** : the process of classifying every single pixel according to its category.
- **Instance Segmentation** : the process of classifying every single pixel according to its category and distinguishing between individual objects of the same class.

3.2 Artificial Neural Networks and Convolutional Neural Newtorks

Artificial Neural Networks

Artificial Neural Networks[16] (ANNs) are computational models designed to address complex machine learning problems by emulating the structure of biological neurons. The Perceptron, the basic ANN architecture, relies on a threshold logic unit (TLU) that calculates the weighted sum of its inputs and subsequently uses a step function to produce an output.

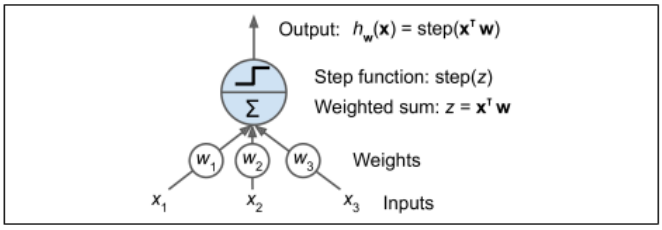


Figure 3.1: Threshold Logic Unit [16].

The TLU can work as a simple linear binary classifier. This is typically achieved through :

- **Heaviside Step Function**, which maps negative inputs to 0 and values greater than or equal to zero to 1.
- **Sign Function**, which returns -1 for negative inputs, 0 for zero and 1 for positive inputs.

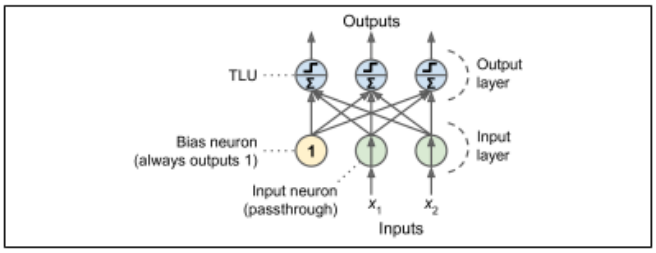


Figure 3.2: Perceptron [16].

A single Perceptron is adequate only for a simple binary classification, but addressing non-linear problems requires stacking multiple layers of neurons to form a Multilayer Perceptron (MLP). An MLP includes an input layer, one or more hidden layers and a final output layer.

If multiple linear transformations are stacked without the inclusion of a non-linear activation function, the entire architecture remains a simple linear model. Nonlinear activations are needed from the network to approximate any continuous function. This is necessary for the system to resolve complex problems, such as the XOR problem.

This is the list of nonsaturating activation functions for positive values:

- **ReLU (Rectified Linear Unit)**:

$$ReLU(z) = max(0, z)$$

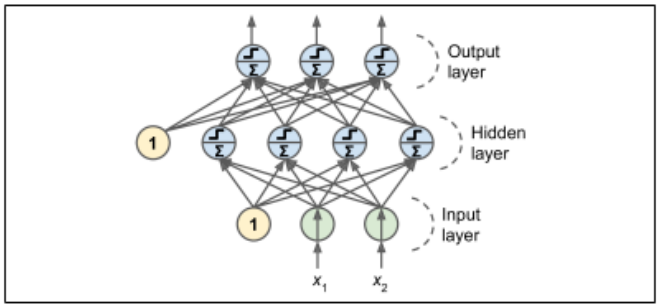


Figure 3.3: Multilayer Perceptron [16].

. It is usually the standard choice due to its efficiency and the fact that it does not saturate for positive values. However, it can suffer from Dying ReLUs, where neurons continue to output zero during training.

- **ELU (Exponential Linear Unit):**

$$ELU_\alpha(z) = \begin{cases} \alpha(exp(z) - 1) & if\ z < 0 \\ z & if z \geq 0 \end{cases}$$

this alternative handles negative values, allowing for an average output closer to zero. It mitigates the vanishing gradients problem and, while it needs more time for the computation, it accelerates convergence.

- **SELU (Scaled ELU):** A specialized variant designed for self-normalization in architectures with dense layers, mitigating the vanishing and exploding gradient problems.

An example of saturation occurs when the input of an activation function reaches extreme magnitudes in the logistic activation function. This effect is observable at both ends in the image 3.4: when the input is a large negative number, the output saturates at 0, and when the input is a large positive number, the output saturates at 1. Consequently, the function's derivative becomes extremely close to zero and during the backpropagation the gradient keeps getting diluted until there is nothing left.

In contrast, as previously mentioned, ReLU is a function that does not saturate for positive values since it does not reach a maximum value. The output continues to increase linearly in proportion to the input. The function remains a straight diagonal line for all positive values $f(z) = z$, its derivative remains constant at exactly 1 regardless of the magnitude of the input, as shown in the figure 3.5. The fundamental advantage of this property is that it helps solve the vanishing gradients problem.

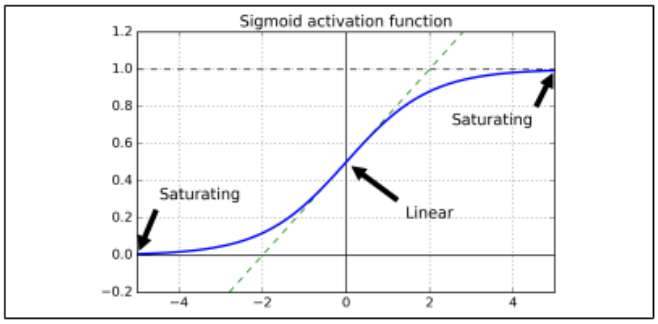


Figure 3.4: Saturation of a function [16].

In contrast, some examples of saturating functions, that are typically used in the output layer, are:

- **Sigmoid (Logistic)**:
$$\sigma(z) = \frac{1}{1 + exp(-z)}$$
outputs values between 0 and 1. It can be used for binary classification.

- **Tanh (Hyperbolic Tangent)**:
$$tanh(z) = 2\sigma(2z) - 1$$
outputs values between -1 and 1.

- **Softmax**:
$$\sigma(s(x))_k = \frac{exp(s_k(x))}{\sum_{j=1}^{K} exp(s_j(x))}$$
where K is the number of classes, $s(x)$ represents the vector of scores for each class for x and $\sigma(s(x))_k$ represents the probability that x is an instance of k. It is essential for multiclass classification and ensures all output probabilities sum to 1.

A very important advancement in the training of deep networks was the back-propagation algorithm, a technique reliant on Gradient Descent. Gradient Descent is an optimization algorithm employed to converge towards optimal parameters for a wide range of machine learning models by iteratively adjusting them to minimize a cost function.

A cost function quantifies the discrepancy between a model's predictions and the actual ground-truth values. The purpose of a learning algorithm is to minimize this

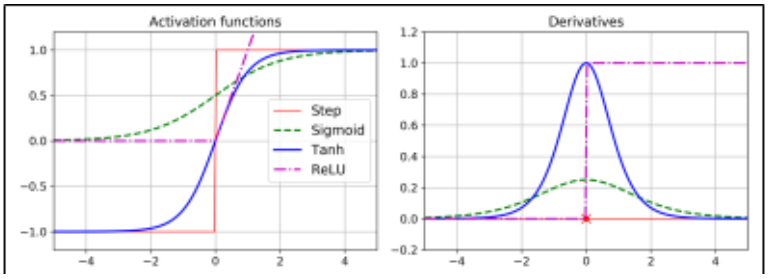


Figure 3.5: Some activation functions and their derivatives [16].

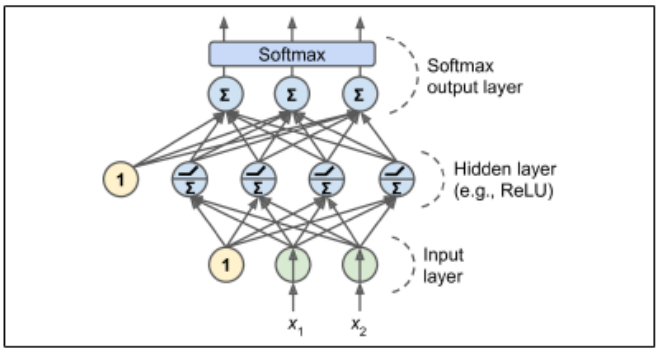


Figure 3.6: ReLU and Softmax used a in Multilayer Perceptron [16].

value. Several functions are used for training standard regression and classification models:

- **Mean Squared Error (MSE)**: a standard regression metric that calculates the average of the squares of the errors. For example, the MSE cost function for the Linear Regression Model is

$$MSE(X, h_\theta) = \frac{1}{m}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})^2$$

- **Mean Absolute Error (MAE)**:

$$MAE(X, h) = \frac{1}{m}\sum_{i=1}^{m}|h(x^{(i)}) - y^{(i)})|$$

  an alternative used for regression when there are many outliers in the dataset, as it is shows to be less sensitive to them than MSE.

- **Huber Loss**: A hybrid approach that combines the strengths of MSE and MAE. It is quadratic for small errors, but becomes linear when the error exceeds a threshold. It mitigates the influence of outliers while ensuring a fast convergence and more accuracy in comparison with the mean absolute error.

- **Log Loss**:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}log(\hat{p}^{(i)}) + (1-y^{(i)})log(1-\hat{p}^{(i)})\right]$$

  average cost between instances in the training set, heavily penalizes models that estimate high probability scores for the incorrect class.

- **Categorical Cross-Entropy**:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} log(\hat{p}_k^{(i)})$$

  where $y_k^{(i)}$ is the probability that the i^{th} element is an instance of k. This is the generalization of log loss designed for multi-class classification tasks.

Backpropagation involves two passes:

- **The Forward Pass**, where the model generates predictions and then quantifies the discrepancy between the output and the actual target value, using a loss function
- **The Reverse Pass**, where the algorithm propagates the error gradient backward from the output layer toward the input to assess the contribution to the error for each connection and determines the necessary adjustments for each weight

Finally, with the Gradient Descent step, the weights are changed as previously defined.

Autoenconders

Autoencoders[16] are artificial neural networks, created to learn dense representations of input data, called latent representations or codings, that do not require manual supervision.

The Encoder, or Recognition Network, and the decoder, or Generative Network, are the two essential components of the autoencoder. Their purpose is to perform the compression of data into an efficient representation and the subsequent reconstruction or transformation of the same data.

The encoder's main function is to process and convert input data into a dense and low-dimensional internal representation, known as codings or latent representation.

In specialized tasks such as Natural Language Processing, the encoder acts as a sequence-to-vector network that condenses a sequence, for example a sentence, into a single vector representation.

Since decoders can create new data from a vector, it is possible to use them as generative models to randomly produce new instances that resemble the training data.

Autoencoders can be described as a form of self-supervised learning because the model is trained to minimize the reconstruction loss, defined as the discrepancy between the input and the reconstructed output.

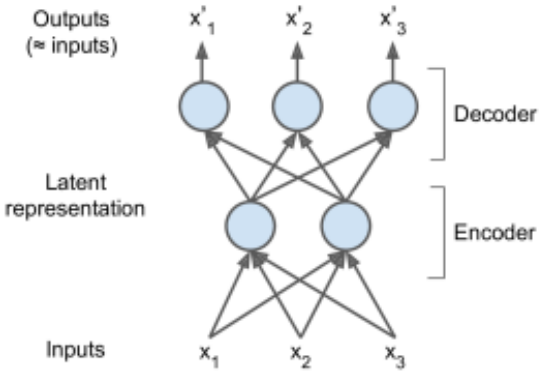


Figure 3.7: A simple autoencoder [16].

Several specialized architectures were developed, depending on the different data types and purposes, for example:

- **Undercomplete Autoencoders**: have a coding layer with a significantly lower dimensionality than the input that requires the network to capture only the most essential features.

- **Stacked Autoencoders**: include multiple hidden layers, needed by the network to learn more elaborate codings.

- **Denoising Autoencoders**: are expected to reconstruct the original and clean input from a version that has been intentionally corrupted by noise. The model is trained to learn robust feature extraction.

- **Variational Autoencoders**: are probabilistic and generative. The encoder outputs a mean coding and standard deviation. The coding is sampled from a Gaussian distribution with the previous mean and standard deviations. Finally, the coding is processed, generating new data instances that resemble the training set.

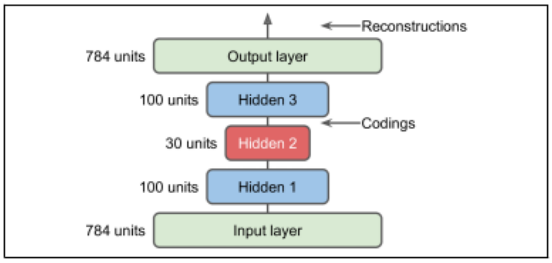


Figure 3.8: Denoising autoencoder, where the Gaussian noise and dropout introduce the noise [16].

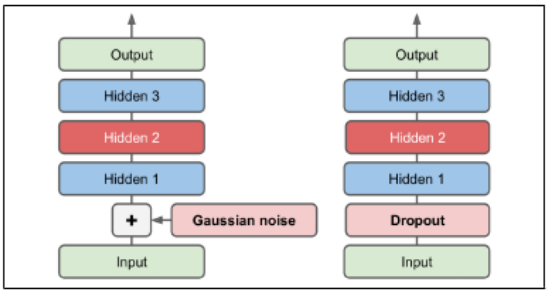


Figure 3.9: Stacked autoencoder [16].

Convolutional Neural Networks

Convolutional Neural Networks[16] are a specialized category of artificial neural networks based on the architecture of the biological visual cortex. Convolutional Neural Networks are designed to process inputs with a 2D or 3D structure, such as images or videos. They are considered an optimal choice for computer vision tasks because layers are not completely connected, reuse weights, and count fewer parameters compared to fully connected DNNs.

CNNs are characterized by two fundamental components:

- **Convolutional Layers**, The network detects low-level features in the initial layers, and assemble them into more complex structures in subsequent stages. Each neuron is only connected to pixels within the corresponding small local receptive field in the previous layer.
- **Pooling Layers**, which perform subsampling on the input to decrease computational load, memory usage, and number of parameters to prevent the model from overfitting. Max pooling, frequently used, offers a degree of translation invariance. The model ensures the recognition of an object even when its position in the image changes slightly. The model focuses on the detection of the feature rather than its precise location. Max pooling neurons do not need

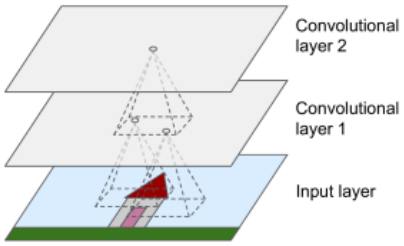


Figure 3.10: Layers in Convolutional Neural Networks and local receptive fields [16].

weights, they are simply stateless sliding windows that compute a specific aggregation function. For each receptive field, the layer finds the maximum input value and exclusively propagates that value to the next layer, discarding any other inputs.

The key technical concepts necessary to comprehend the functioning of CNNs are:

- **Filters (Kernels)**: the filter is slid across the image with the aim of producing a feature map that highlights the location of specific patterns.
- **Weight Sharing**: All neurons in a specific feature map use the same weights and bias. This significantly reduces the total number of parameters. When a CNN learns to identify a pattern in one location, it can recognize it regardless of its position in the image.
- **Zero Padding**: this technique involves adding zeros around the edges of the input to preserve the values of the height and width after a convolution.
- **Stride**: represents the distance crossed by the filter between each receptive field. A larger stride corresponds to the reduction of the dimensions of the output.

CNNs offer several advantages that are not provided by fully connected deep neural networks:

- **Efficiency**: since the layers are partially connected and weights are shared, they have fewer parameters. This results in a faster training and less data needed to generalize.
- **Spatial Awareness**: deep neural networks lack prior knowledge of pixels organization, while CNNs contain this prior information.

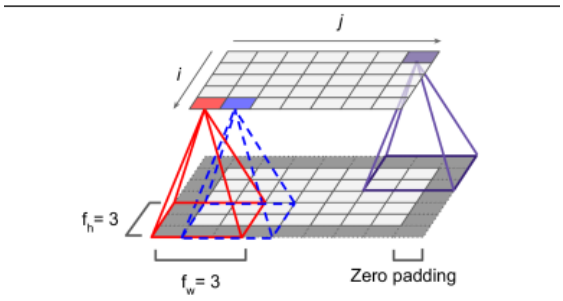


Figure 3.11: Connections between layers and zero padding in Convolutional Neural Network [16].

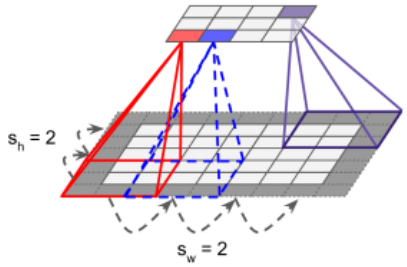


Figure 3.12: Dimensionality reduction with a stride of 2 [16].

3.3 Evolution in different versions of YOLO

The evolution from YOLOv5 to YOLO11[17] represents a significant architectural transformation, evolving from a specialized detection tool to a multi-task vision framework which includes detection, segmentation and pose estimation.

YOLOv5

YOLOv5 represented an important evolution from the original Darknet framework to a PyTorch-native implementation. This change prioritized developer ergonomics and modularity, making the model more accessible and easier to customize.

It is possible to list the characteristics of YOLOv5

- **Object Detection**: YOLOv5 used an anchor-based detection head. The model calculated the offsets relative to the predefined anchor priors. The architecture was built on backbones inspired by CSP networks and employed a PANet-style neck. PANet, or Path Aggregation Network, is a specialized architecture component designed to aggregate features of various scales extracted from the backbone of the neural network, causing a small latency overhead.

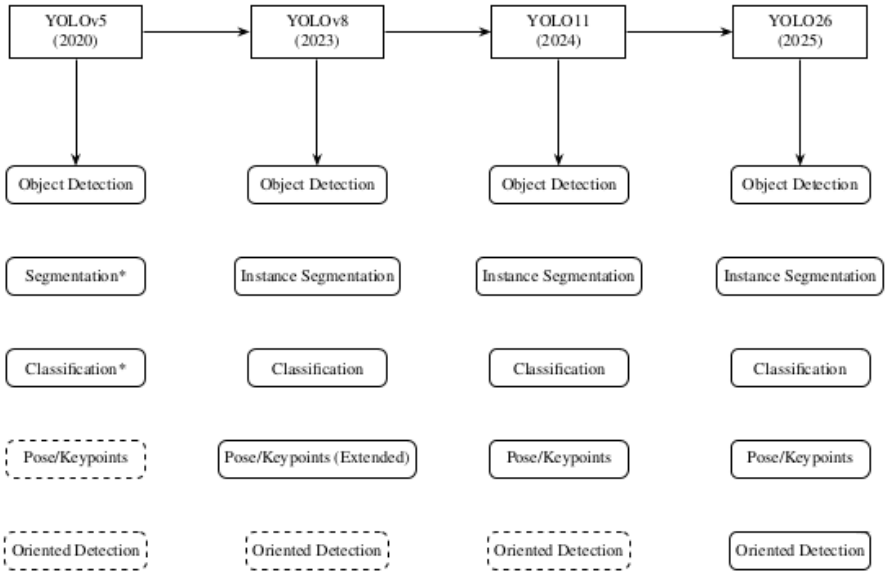


Figure 3.13: Graph that shows the evolution of the tasks supported by different versions of YOLO [17].

It works with the SiLU (Sigmoid Linear Unit), an activation function used to mitigate optimization pathologies, such as vanishing gradients.

- **Instance Segmentation and Pose Estimation** : were not supported in the initial release

YOLOv8

YOLOv8 represents a significant redesign that moved toward a more flexible, anchor-free technique.

- **Object Detection (Decoupled Head)** : an important theoretical advancement was the introduction of a decoupled detection head. YOLOv8 divided the classification and regression tasks into separate branches to mitigate gradient interference between the localization and recognition objectives. Providing each task with its own single branch leads the model to a more stable training process that can achieve its optimal performance steadily and obtain an improved calibration given an operating point.
- **Anchor-Free Design**: YOLOv8 no longer uses anchor boxes and estimates normalized bounding box parameters and confidences from feature map points that are considered promising candidates. As a consequence, hyperparameter tuning was simpler and the generalization was improved.

- **Native Multi-Tasking**: YOLOv8 integrates Instance Segmentation and Pose Estimation (Keypoints). It introduces a C2f backbone that offers a more lightweight representation while preserving receptive-field richness. With the C2f design, the model can reduce its total number of parameters. At the same time, the model maintains high processing speeds, allowing YOLOv8 to remain a real-time detector. Finally, the model preserves the detailed spatial information needed to identify items of reduced dimensions.

YOLO11

YOLO11 improved the multi-task approach by prioritizing device efficiency and focusing on the model's capacity to work in complex, cluttered scenes.

- **Detection (Hybrid Task Assignment)**: a significant theoretical advancement in YOLO11 is the improvement of the supervisory signal through hybrid task-aware assignment. This approach involves jointly tuning label assignment and loss weighting across classification, localization, segmentation and pose to minimize failure modes in crowded scenes. This technique results in a more robust model regardless of variations in training batch sizes.
- **Architectural Enhancements (C3k2 & C2PSA)**:
  - **C3k2 Bottlenecks**: C3k2 is characterized by a smaller kernel. This choice results in improved efficiency and feature reuse, avoiding increased latency. This model is capable of processing information with less redundancy. One of the main advancements was the improvement of the FLOPs-to-mAP ratio.
  - **C2PSA Module**: this module integrates CSP design and spatial attention mechanisms. With the use of spatial attention, the C2PSA module helps the model improve feature selectivity, ensuring that details of cluttered scenes with small objects are not lost.
- **Segmentation and Pose**: improvement of the FLOPs to mAP ratio. It is more accurate than YOLOv8 with a similar latency.

3.4 Projective Geometry

In order to understand the following sections of the thesis, it is important to know some concepts about 2D projective geometry[15], which is the study that focuses on geometric properties of the projective plane $\mathbb{P}^2$ that remain invariant even under a group of transformations known as projectivities. A projectivity is defined as a projection h from $\mathbb{P}^2$ to $\mathbb{P}^2$ that can be inverted and where the points x_1, x_2, x_3

are placed on the same line if and only if also $h(x_1), h(x_2), h(x_3)$ are. In Euclidean geometry, parallel lines are defined as never meeting, but the projective space includes points at infinity, also known as ideal points, where they actually intersect. A standard 2D point (x, y) is expressed as a 3D vector (x, y, w). In this system, only the ratio between elements is relevant. This notation is important because it allows the non-linear process of perspective projection to be expressed as linear matrix equations. A point at infinity is defined by setting the final coordinate $w = 0$, which ensures calculations with infinite entities to be as easy as finite ones.

A hierarchy of geometric transformations is necessary to comprehend how to move from a distorted camera view to real-world measurements:

- **Projective Geometry**: the most general level, preserves collinearity.
- **Affine Geometry**: a specialization of the previous geometry where the line (or plane) at infinity is identified. It preserves parallelism, ratios of lengths along parallel lines and ratios of areas.
- **Euclidean Geometry**: the most specific level, where the circular points and absolute conic are identified. It enables the measurement of angles and absolute length ratios.

Projective geometry provides the mathematical foundation for computer vision and is useful for mapping a three-dimensional world onto a two-dimensional image plane.

In single-view geometry, so the geometry of a single perspective camera, these are the fundamental tools for metric recovery:

- **Vanishing Points**: they represent the images of points located on the plane at infinity. It is found after intersecting the image plane with a ray that is parallel to the world line and meets the camera centre. A vanishing point is determined only by the direction of parallel lines in the scene, rather than their specific location. For this reason, parallel world lines share a vanishing point.
- **Vanishing Lines**: they represent the images of lines located on the plane at infinity. These occur when the image meets a plane parallel to the scene plane through the camera centre. A vanishing line is determined only by the orientation of the scene plane, rather than its specific location.

3.4.1 Perspective Projection and Intrinsic Parameters

The projection process[15] defines how a 3D point in space is mapped onto a 2D pixel. This occurs as a ray of light travels from a point, passes through the centre

of projection and finally meets the image plane. This intersection is the image of the point. This transformation involves two sets of parameters:

- **Intrinsic Parameters (The K matrix)**: they are stored in a calibration matrix and include:
  - **Focal Length**: that determines the degree of magnification of the scene
  - **The Principal Point**: it identifies the exact pixel coordinates where the principal axis intersects the image plane.
  - **Skew**: a factor that addresses whether the x and y axes are perpendicular, but they are rarely not.
- **External Parameters**: they define the location of the camera in the world and which way it is pointing.

In finite projective cameras, the camera matrix is

$$P = K[R|t]$$

, where K is the intrinsic calibration matrix, R is 1 3×3 rotation matrix and t is a translation vector.

The internal structure of K is:

$$K = \begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

where α_x, α_y represent focal length in pixel dimensions and x_0, y_0 are the coordinates of the principal point.

3.4.2 Back-Projection and 3D Inference

Back-Projection [15] is the process that involves identifying which set of points in the 3D space map to the selected point in the image. When analyzing a 2D image, the precise 3D position of a pixel is uncertain because all points along a specific ray starting from the camera map onto the same pixel.

To overcome this depth uncertainty and perform 3D Inference, two or more views are required. In the process known as triangulation, by identifying the same feature in two or more distinct views, the intersection of the back-projected rays in space can be computed to determine the point's coordinates in space.

If the camera's internal settings are unknown, the 3D model generated will likely suffer from projective ambiguity, appearing distorted compared to reality. To resolve this and achieve a reliable metric reconstruction it is necessary to find specific landmarks, such as the plane at infinity and the absolute conic.

The Absolute Conic (Ω_∞) is a special mathematical curve situated on the plane at infinity. Since it remains invariant under Euclidean transformations, it is possible to use it as a landmark to define absolute properties like angles and ratios of lengths.

The Image of the Absolute Conic (IAC), represented by the symbol (ω) is the 2D projection of the absolute conic onto the image plane.

Since the Absolute Conic lies on the plane at infinity, the translation and rotation of the camera have no effect on the shape of its image ω. The IAC can only be influenced by the intrinsic parameters K. This relationship is expressed as

$$\omega = (KK^T)^{-1} = K^{-T}K^{-1}$$

.

Chapter 4

Instance Segmentation and Pose Estimation Tasks

The developed pipeline employs two YOLOv8 models for the primary tasks : the instance detection and segmentation of the fruit and the keypoint regression to determine its pose.

4.1 Augmentation Techniques

During the training phase, data augmentation techniques[18] were used as regularization methods[16] to avoid overfitting so that the model can generalize to unseen data. In this way, the model learns general representations of objects rather than remembering specific training instances. Here are listed some of the most important [19] [20]:

- **Scaling**: applying a random scaling factor within a designated range is important for the model to recognize objects independently of their size and proximity to the camera.
- **Translation**: applying a horizontal and vertical translation by a predefined fraction of the size of the image is important for the model to recognize objects independently of their location.
- **Rotation**: rotating the image by a random degree in a predefined interval is necessary for the model to be more robust to changes in orientation.
- **Flipping**: it can be done vertically or horizontally so that the model recognizes objects regardless of their orientation.

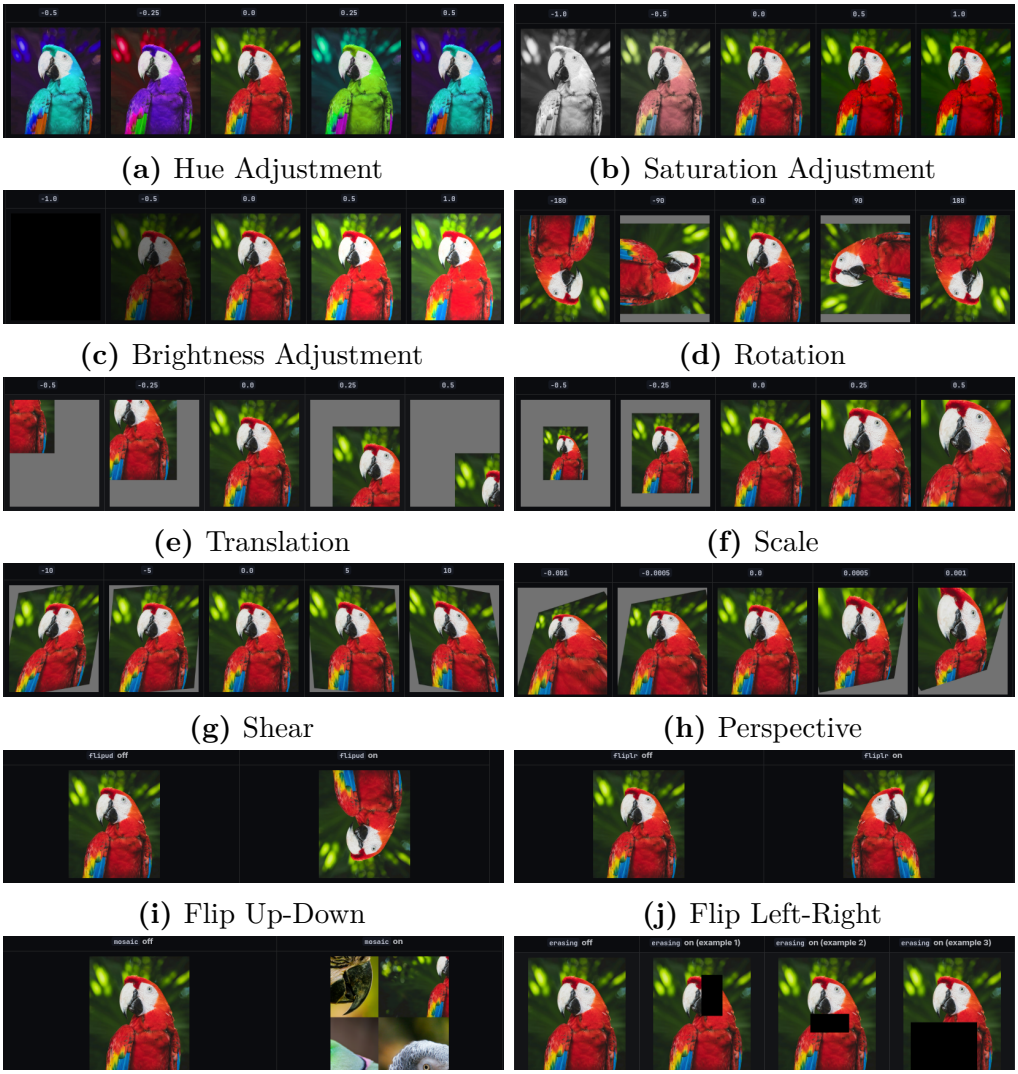
- **Shear**: shifts portions of the image in a direction, preserving the parallel lines to improve the model's robustness to different viewpoints.
- **Perspective**: simulates how the shape of the object changes from different distances and angles, performing a full transformation for the perspective.
- **HSV Adjustments**: random varying exposure and saturation of images by up to a factor of 1.5 is important to enhance the model's robustness to several lighting conditions and color changes. These parameters are typically Hue, Saturation and Exposure:
  - **Hue Adjustment**: changes the colors in the image, but maintains their relationships.
  - **Saturation Adjustment**: changes the intensity of colors in the image.
  - **Brightness Adjustment**: changes the brightness in the image.
- **Mosaic**: unites four images into one to help identify small objects and improve the comprehension of the context.
- **Mixup**: according to a predefined probability, it blends two images and their corresponding labels to improve the recognition of occluded objects.
- **Copy-Paste**: extracts objects and places them in new backgrounds. It is used in the segmentation tasks to increase the number of times that a rare instance is studied.
- **Random Erasing**: erases random parts of the image to make the model more robust to occlusions.

4.2 Segmentation Model for Apple Detection

4.2.1 Dataset Structure

The segmentation model was trained on a dataset focused on a single "apple" class. The dataset was partitioned as follows:

Subset	**Number of Images**
Training	319
Validation	208
Testing	105



(a) Hue Adjustment **(b)** Saturation Adjustment

(c) Brightness Adjustment **(d)** Rotation

(e) Translation **(f)** Scale

(g) Shear **(h)** Perspective

(i) Flip Up-Down **(j)** Flip Left-Right

(k) Mosaic **(l)** Random Erasing

Figure 4.1: Examples of Augmentation [19]

4.2.2 Training Procedures

The adopted input resolution was of 1088 x 1088 (imgsz). The standard for the YOLO architecture is typically a resolution of 640 x 640 [21], but the experimental data of the examined papers show the correlation between increased resolution and segmentation precision [22]. Instance segmentation demands larger inputs compared to simple object detection. A larger image ensures higher quality masks and better performance on smaller objects, which results in a more accurate dimension estimation in the next steps.

In the following section the augmentation settings[18], [16], [20], [19] used will be listed and explained:

- **mosaic=1.0**: all images are surely combined.
- **degrees=15.0, translate=0.1, scale=0.5** : the model recognizes objects regardless of their variations in location, orientation, and dimension.
- **fliplr=0.5**: horizontal flipping.
- **hsv_h=0.015, hsv_s=0.7, hsv_v=0.4**: these adjustments to hue, saturation and exposure ensure the model remains robust in varying lighting conditions.
- **copy_paste=0.3**: this technique is important in the instance segmentation task.
- **erasing=0.4**: erases parts of the image to become more robust to occlusions.
- **optimizer=AdamW**: [23] influences the stability and speed of the convergence.
- **warmup_epochs=5.0**: [23] incrementing the learning rate progressively in the predefined number of epochs to stabilize the training sooner.

4.3 Pose Estimation Model

4.3.1 Dataset Structure and Keypoints definition

The dataset for the pose estimation task focuses on a single class "apple" and 3 different keypoints. The topological structure of the apple is represented by three points drawn in the shape of a triangle as shown in the Figure 5.2. The upper point represents the stem, while the other two represent the base of the apple.

The dataset was partitioned as follows:

Subset	**Number of Images**
Training	933
Validation	265
Testing	134

Figure 4.2: Test Images for the Segmentation Model

4.3.2 Training Procedures

The input resolution was 1088 x 1088 (imgsz) since the performance in keypoint regression is significantly influenced by the input resolution[24]. The training step integrated augmentations[18], [16], [20], [19] to enhance the model's robustness:

- **mosaic=1.0**: all images are surely combined.
- **copy_paste=0.4**: overlaying objects instances onto new backgrounds.
- **degrees=20.0, scale=0.9**: these settings mitigate the impact of changes in orientation, size, and position.
- **perspective=0.0005**: creation of images that seem to have different angles.
- **shear=5.0**: simulates different points of view caused by a slight tilt.
- **hsv_h=0.03, hsv_s=0.9, hsv_v=0.5**: The model remains robust under different lighting conditions.

Figure 4.3: Annotation Examples for the Pose Model

- **optimizer=AdamW**: [23] influences the stability and speed of the convergence.
- **warmup_epochs=5.0**: [23] incrementing the learning rate progressively in the predefined number of epochs to stabilize the training sooner.

Figure 4.4: Test Images for Pose Model

Chapter 5

Methodology for Apples Radius Estimation

5.1 Algorithm Design

5.1.1 Theory Foundations

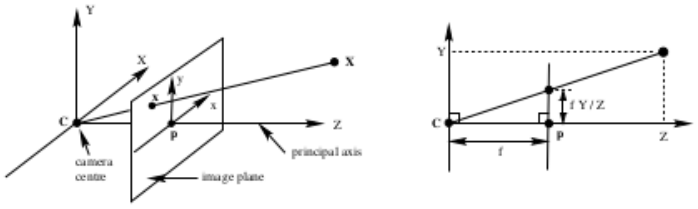


Figure 5.1: Pinhole camera geometry where C represents the camera centre, located on the origin, and p is the principal point [15].

The camera calibration matrix [15], K, represents the internal geometry of the camera. The matrix K acts as the affine transformation between a pixel x and the direction of the corresponding back-projected ray d within the camera's Euclidean coordinate system. To determine the actual trajectory of light entering the lens, it is necessary to use the inverse of the calibration matrix:

$$d = K^{-1}x$$

.

Projective geometry involves dimensionality reduction, where a three-dimensional environment is compressed into a two-dimensional image plane. In a central projection, every point that resides along a specific ray passing through the camera

center is mapped onto the same pixel. Consequently, the distance of an object in a single 2D image is indeterminate. To resolve this, the depth value Z is needed. Mathematically, the 3D point is calculated by scaling the ray direction d by the depth Z. This identifies the unique point X in the camera space:

$$(X, Y, Z)^T = ZK^{-1}x$$

.

The Basic Pinhole Model can be defined as the simplest representation of central projection. The model is built upon the geometric principle of similar triangles. A point in space with depth Z is projected onto the image place at a distance f. Since these triangles share a vertex at the camera center, it is possible to derive that

$$(X, Y, Z)^T \rightarrow (\frac{fX}{Z}, \frac{fY}{Z})$$

that shows the projection from the world to coordinates in the image. Consequently, in the code, the ratio of the physical size to the distance D_{metric}/Z is equal to the ratio of the pixel size to the focal length d_{px}/f.

The application of the Minimum Enclosing Circle[25] function from the OpenCV library is needed to find the smallest circle that can enclose the mask of the apple and is theoretically supported by the study of images of smooth surfaces. The outline of an object in an image is known as the apparent contour. For a sphere, the rays from the camera center that are tangent to the surface form a right-circular cone. The intersection of the cone with the image plane results in a conic section. If the sphere is centered on the principal axis, its projection appears as a perfect circle, otherwise, it becomes an ellipse. In the code, by calculating the diameter of this contour at a determined depth Z, the physical metric size is estimated.

5.1.2 Metric Radius and 3D Position Estimation

The code implements back-projection to map 2D image coordinates into the 3D Camera Coordinate System. To locate the apple in the 3D space, the system uses the camera calibration matrix K, which stores the camera's internal characteristics[15].

- **Utilization of Intrinsic Parameters**: K acts as the transformation between a 2D image point x and its corresponding 3D ray direction d in the camera's Euclidean coordinate frame, where $d = K^{-1}x$.
- **Resolving Scale Ambiguity**: Since a reconstruction from a single perspective is subject to scale ambiguity, the depth value Z is therefore essential to act as a scaling factor, since it establishes the precise location of the object along the back-projected ray.

- **Application of Similar Triangles**: The metric size estimation of the apple's size is based on the basic pinhole camera model, where the camera center acts as a vertex for two similar triangles.

  Due to the similarity of these triangles, the ratio of their bases is equal to the ratio of their heights $\frac{d_{px}}{f} = \frac{D_{metric}}{Z}$ and the physical diameter can be calculated as

  $$Dmetric = \frac{(dpx \cdot Z)}{f}$$

- **Spherical Approximation and Scale Invariance**: The diameter is estimated after using the Minimum Enclosing Circle [25] function applied to the segmentation mask. This choice is supported by the theory that the apparent contour of a sphere under perspective projection forms a conic. By computing the size with the use of the depth Z measured in the center of the apple, the estimation becomes scale-invariant. This ensures that when the apple moves further away and Z increases, its pixel size d_{px} decreases proportionally, maintaining the final calculated D_{metric} constant.

5.1.3 Stem-Base Vector and Angular Orientation

To determine the orientation angle θ in radians, the system employs a geometric pipeline that converts pixel displacements into a circular coordinate system. The process begins by identifying two critical points in the 2D image to create a directional vector:

- **The Stem (P_{stem})**: the top point of the apple's axis.
- **The Base Midpoint (P_{mid})**: calculated as the center between the two lower keypoints.

The displacement components are obtained by subtracting the coordinates of the base from the stem:

- d_x: measures the lateral tilt of the fruit.
- d_y: measures the vertical alignment.

To convert the d_x, d_y values into an angle, the code applies the function[26]:

$$\theta = atan2(dx, dy)$$

that determines exactly where the vector is pointing and since the sign of both x and y is known, it can output the corresponding quadrant for the angle.

Figure 5.2: Example of Keypoints Structure

Chapter 6

Experimental Results

6.1 Evaluation Metrics

The quality of the model is established through fundamental classification performance indicators[16]:

- **Accuracy**: the ratio of correct predictions relative to the total number of instances.
- **Precision**: a measure that quantifies the accuracy of positive predictions
- **Recall**: a measure that calculates the proportion of positive instances correctly detected.

Subsequently, the following metrics were used to evaluate instance segmentation models.

- **Average Precision (AP)**: determined by the Intersection over Union (IoU), which is the area of overlap between the predicted mask and the correct mask divided by the total area of their union.
- **mAP**: the mean of the Average Precision (AP) calculated across all distinct object classes.
- **Threshold-Specific AP**: models are evaluated at specific IoU thresholds, such as AP50 that requires a 50% overlap for the detection to be considered successful.

For pose estimation, the evaluation focuses on the localization of keypoints:
Object Keypoint Similarity (OKS) is the most important metric for pose estimation[24] and can be considered as the keypoint equivalent of Intersection over Union.

Given the requirements of robotic harvesting tasks, the model is also evaluated using speed and efficiency metrics because the goal is real-time performance. Models are evaluated using inference time in milliseconds.

6.2 Results of the Segmentation Model

Metric Type	Precision	Recall	mAP50	mAP50-95
Bounding Box(B)	0.685	0.848	0.732	0.632
Segmentation Mask (M)	0.687	0.863	0.738	0.560

Table 6.1: Results Segmentation Model

The results in the table 6.1 reflect a high-performing segmentation model. The model shows a significant gap between Recall ($\sim$0.85-0.86) and Precision ($\sim$0.68). A high recall indicates that the model can successfully identify most target instances within the dataset. Since the precision is lower than the recall, the model is more likely to produce a higher number of false positives.

The mAP50 value (0.732 for boxes and 0.738 for masks) is quite high. This means that the model seems successful at locating the general area of the apples with at least 50% overlap.

The mAP50-95 value (0.632 for boxes and 0.560 for masks) indicates that the model faces greater difficulty with stricter Intersection over Union thresholds, while at the lower threshold mAP50, the Segmentation Mask outperforms the Bounding Box. In contrast, at the mAP50-95 level, the results for bounding boxes are significantly more accurate than the results for the masks. Although the model excels at capturing the shape of the object, the precise pixel boundaries required are more difficult to regress than the four coordinates of a box. These competitive results are likely supported by the choice of a high input resolution.

FPS	Average inference time
21.12 FPS	0.0473

Table 6.2: Inference Time Segmentation Model

The industry standard[22] for real-time instance segmentation is typically considered to be above 30 FPS.

The segmentation model processes data at 47 ms, as shown in the table 6.2, and this latency represents a trade-off for the high input resolution chosen, since a higher resolution of images results in a significant decrease of the speed. At the same time, an inferior resolution would cause a wrong estimation of the size of the apples.

YOLO Segmentation Graphs

In the figure 6.1, the BoxPR and MaskPR curves represent the results in the table. The graphs explicitly show an mAP@0.5 of 0.732 for Bounding Boxes and 0.738 for Segmentation Masks. This confirms that at a 50% IoU threshold, the model's masks are slightly more accurate than the boxes.

Both the BoxF1 and MaskF1 curves reach an identical peak at a score of 0.76. The F1 score is "the harmonic mean of precision and recall".[17] A peak of 0.76 at confidence thresholds of 0.544 (Box) and 0.529 (Mask) represents the fact that the model is most effective when filtering out detections with less than ∼53% confidence. The curves show a broad plateau between 0.2 and 0.7 confidence, suggest a stable performance that is not easily impacted by minor changes in the confidence threshold.

The Recall-Confidence curves show the model achieving recall of ∼0.95 at a 0.00 confidence threshold. This matches the high recall values expressing the model's capacity of finding most apples in the image.

The Precision-Confidence curves show that precision starts at ∼0.3 and only reaches high levels when the threshold increases. This explains the lower precision of the table (∼0.68), which indicates that the model initially identifies many false positives.

The Confusion Matrix provides a detailed description of the errors. The matrix shows 2134 background instances that were incorrectly predicted as "apple" and 3327 apples correctly identified. The normalized matrix indicates a 0.94 recall for apples, meaning that the model only overlooked 6% of the actual fruit.

6.3 Results of the Pose Model

Metric Type	Precision	Recall	mAP50	mAP50-95
Bounding Box(B)	0.871	0.817	0.890	0.702
Keypoint Pose (P)	0.857	0.803	0.876	0.782

Table 6.3: Results Pose Estimation Model

The results in the table 6.3 describe a high performance and stability. For both bounding boxes (∼0.87) and keypoints (∼0.86), the model shows high accuracy in its positive predictions while successfully identifying most instances. This balance suggests that the chosen training strategy, augmentations and a high input resolution of 1088x1088, effectively regularized the model, preventing it from producing excessive false positives or overlooking targets.

The mAP50 values (0.890 for boxes and 0.876 for keypoints) express that the model is highly reliable at a standard 50% similarity threshold.

The Keypoint Pose (0.782) significantly outperforms the Bounding Box (0.702) in the mAP50-95 metric. The high mAP for poses even at strict thresholds express the fact that the model is successfully handling occlusion.

FPS	**Average inference time**
23.88 FPS	0.0419

Table 6.4: Inference Time Pose Estimation Model

An inference time of approximately 42 milliseconds, shown in the table 6.4, is a competitive result for a simultaneous detection and pose estimation task, considering the use of a higher resolution as an input that negatively impacts the speed.

YOLO Pose Graphs

In the figure 6.2, the box detection maintains a high precision until a recall of approximately 0.7. The pose precision is slightly lower than the box precision but remains reliable.

For both Box and Pose, the F1 curve reaches the peak respectively at a confidence threshold of ∼0.538 and ∼0.537, producing scores of 0.84 (Box) and 0.83 (Pose). This peak represents the optimal trade-off between precision and recall for the model. The F1 graph represents a reliable model that performs consistently well in a wide range of confidence thresholds, from ∼0.2 to ∼0.7.

The Precision-Confidence curve shows that the model reaches a precision of 1.00 at a confidence of 0.963.

The Recall-Confidence Curve shows a high maximum recall of 0.95 at zero confidence.

6.4 Results of the Radius Estimation Methodology

The proposed methodology for the estimation of the radius was evaluated using ROS bags of showing apple trees. In a sample of 118 successfully detected apples, 83,05% instances were found with an estimated diameter within the range of 6.5 cm to 8.5 cm, the benchmark established for accuracy. In the same sample, the system calculated an average estimated diameter of 7.86 cm with a mean distance from the camera of 1.62 m.

The range for the diameter was decided after choosing a plastic apple as a ground truth, with the following dimensions in meters:

X	0.075
Y	0.075
Z	0.065

Table 6.5: Dimensions of the Ground Truth Apple

The images in the figure 6.3 show the mesh created by using the mobile application Polycam, a tool for creating meshes using photogrammetry and AI solutions[27].

With the use of the software Blender[28], for modeling meshes, it was possible to remove the unnecessary points in the mesh that did not belong to the apple and to correct the dimensions of the model as shown in the figure 6.4.

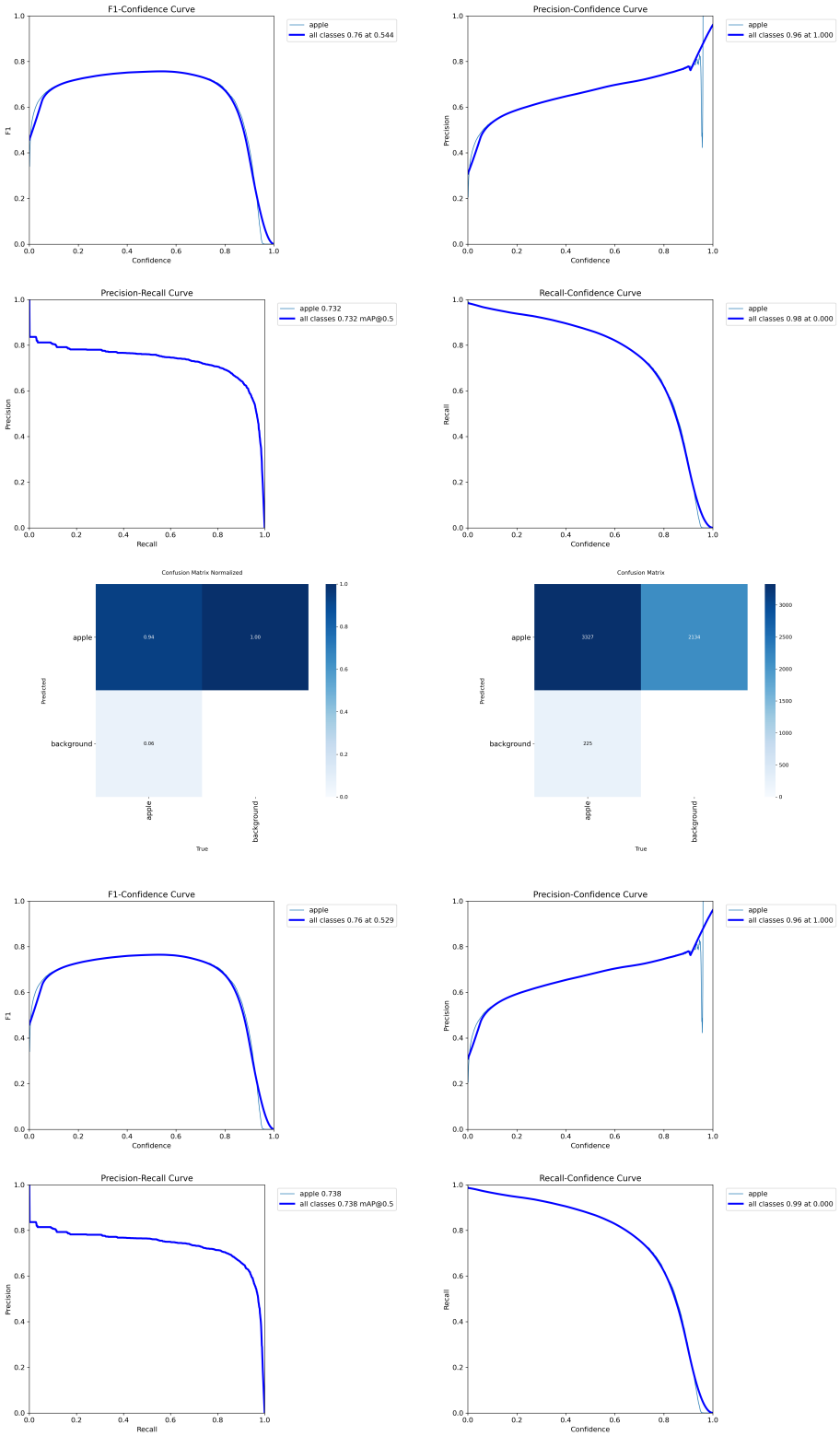


Figure 6.1: Results for the Segmentation Model

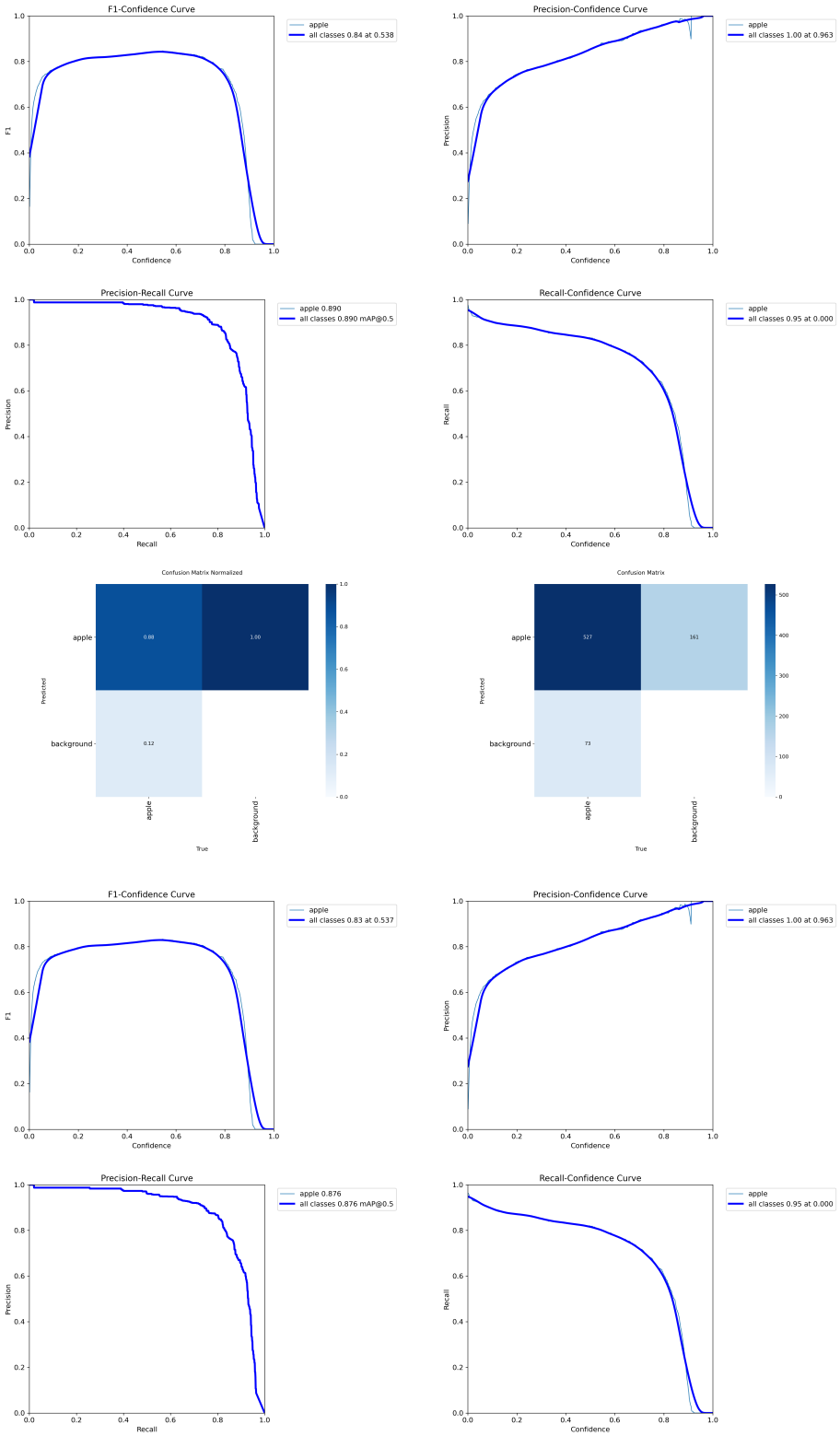


Figure 6.2: Results for Pose Estimation Model

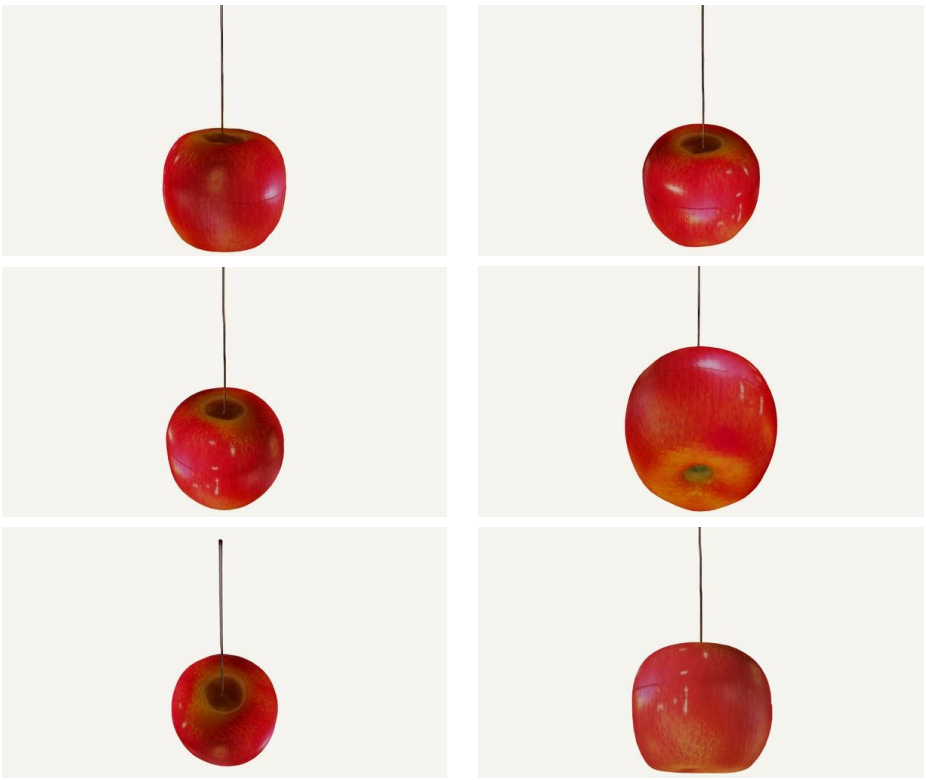


Figure 6.3: Starting Point Ground Truth Mesh

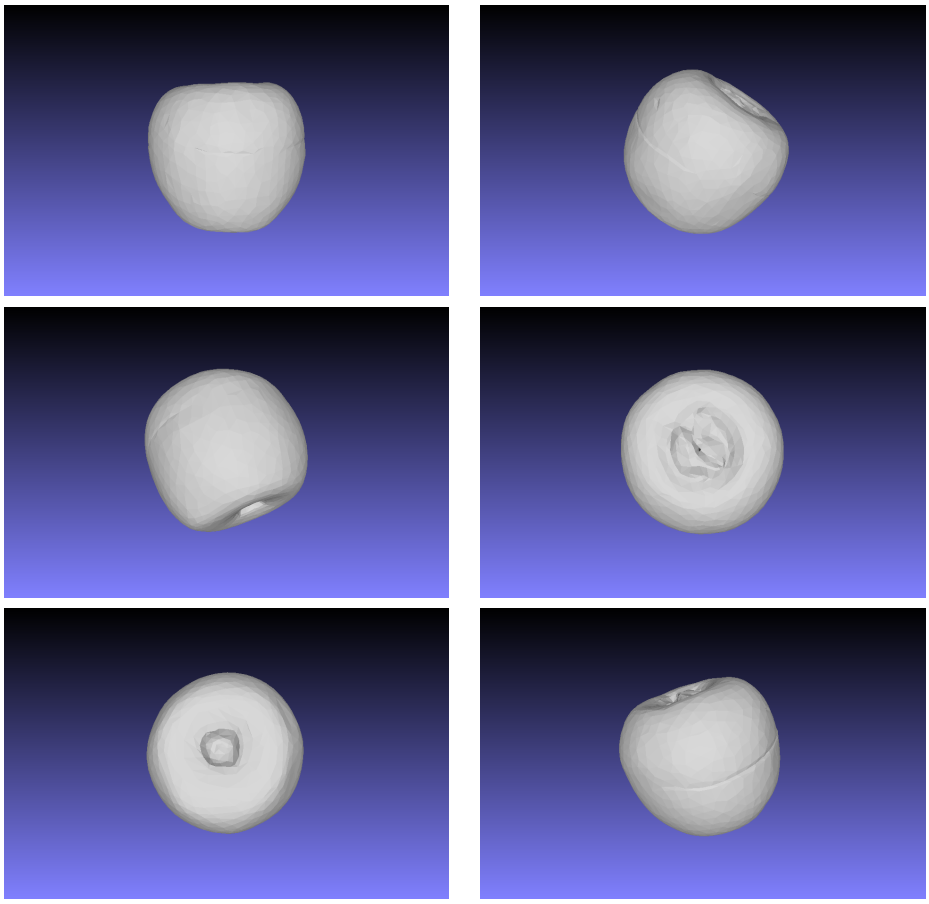


Figure 6.4: Ground Truth Mesh

Chapter 7

ROS Package Design

7.1 ROS Humble

The second generation of the Robot Operating System (ROS 2) is considered a complete redesign of the framework [29], created to transition from a platform suited for research tasks to one capable of supporting industrial applications[30] . ROS 1 was essential for accelerating robotics development but it still had important limitations regarding security, real-time support and reliability in environments with unstable networks. ROS 2 addresses these challenges by using the Data Distribution Service (DDS) as its communication backbone. The integration of DDS provides the following advancements:

- **Decentralized Discovery**: while ROS 1 requires a central main node, DDS uses peer-to-peer discovery and avoids the risk of having a single point of failure.
- **UDP Transport**: DDS primarily uses UDP to deliver data. UDP does not automatically re-transmit data and this helps DDS managing re-transmissions more efficiently in unreliable conditions.
- **Security**: DDS includes a built-in security standard that provides authentication, access control and encryption.

One of the most powerful features of DDS is the ability to tune communication via QoS settings to adapt to specific network constraints:

- **Reliability**: it could be set to "best-effort", where data is sent once, or "reliable", where data is re-transmitted until receipt is acknowledged.
- **Durability**: if it is set to "transient-local", the system saves messages for late-joining nodes that subscribe, if it is set to "volatile", the system forgets them immediately.

- **History**: when set to "keep-last", it saves a fixed number of recent messages. When set to "keep-all", it keeps all data.

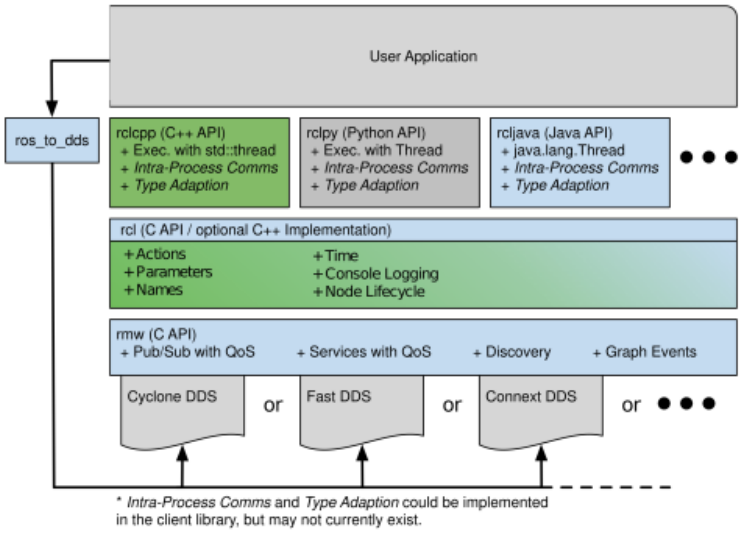


Figure 7.1: "ROS 2 Client Library API Stack"[29].

ROS2 is designed to move beyond the limitations of the first generation by implementing a standardized architecture:

- **The RMW Abstraction Layer**: In ROS 2, DDS is hidden behind an abstraction layer called the RMW (ROS MiddleWare). This design allows developers to swap implementations of different DDS vendors.
- **Unified Client Libraries**: A common C library, called rcl, provides core functionality to language-specific client libraries, such as rclcpp for C++ or rclpy for Python, as shown in the figure 7.1.

The architecture of ROS 2 is built following several core design principles, such as:

- **Distribution**: the system is decomposed into functionally independent components that share data through explicit, decentralized communication.
- **Abstraction**: establishing semantics for data exchange and ensuring that components remain interoperable and independent of specific hardware or software vendors.
- **Modularity**: Following the UNIX philosophy, the ecosystem is organized into federated packages.

These principles allow developers to reuse code, test software, isolate faults, and collaborate effectively.

From a communication perspective, ROS2 organizes software into Nodes, the fundamental organizational units, interacting through:

- **Topics**: a framework used to transmit asynchronous messages, that allows many-to-many data streaming.
- **Services**: a non-blocking pattern for request-response interactions.
- **Actions**: a non-blocking pattern for complex, long-running tasks that require periodic feedback and the ability to be canceled.

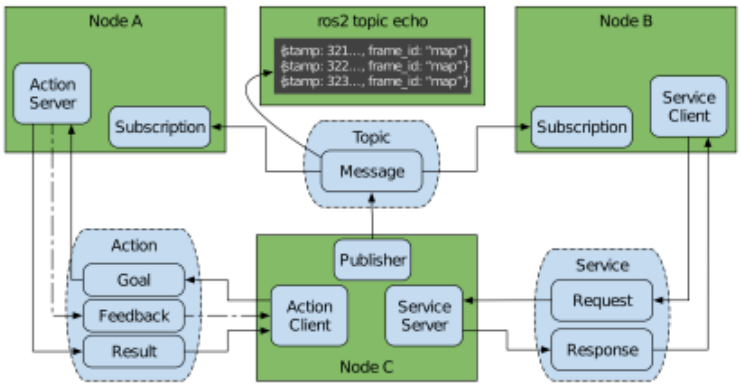


Figure 7.2: "Topics, services and actions" [29].

ROS 2 distributions are comprehensive Software Development Kits (SDKs) built on a federated ecosystem of packages. **Humble Hawksbill** is identified as a Long Term Support (LTS) distribution. This was the distribution used during the development of the solution of the original problem in the thesis.

Modern distributions incorporate advanced mechanisms for determinism and performance, such as:

- **Lifecycle Nodes**: a state machine to manages program states, such active or inactive, to coordinate complex distributed systems.
- **Node Composition**: a program can be located in a process at compile-time or run-time. Multiple nodes can share a single process, eliminate networking delays and drastically reduce resource usage.

Bag files (rosbag2) are an indispensable component of the ROS 2 developer toolkit. Their core functionalities include:

- **Data Collection**: rosbag2 is used to record raw data streams from all levels of a robotic system during runtime.

- **Logging and Metadata**: these tools capture runtime events, error logs, debug outputs, and system metadata, which are stored and subsequently analyzed and debugged.
- **Validation and Debugging**: logging is central to the validation process. The use of rosbag2 allows for a standardization of the logging system and a facilitation of the exchange of information.

7.2 Intel RealSense Camera

The Intel RealSense D435[31] is a stereoscopic depth camera often used in robotics tasks. Its primary technical features include[32]:

- **Global Shutter**: The D435 features a global shutter that is necessary to capture moving objects by reducing motion blur and perform in low-light conditions.
- **Wide Field of View (FOV)**: A wide FOV of 91°× 65°× 100°reduces "blind spots" in the depth image and makes the camera suitable for tasks where the general visual performance is more important than the accuracy.
- **Stereo Vision with Unstructured Light**: the RealSense is composed of a RGB camera, an infrared camera and an infrared laser projector, as shown in figure 7.5. All these lenses are necessary to obtain the depth, identifying the infrared light reflected from the object and subsequently creating the depth image, with the help of visual data. The camera obtains the depth with stereovision, using two sensors (one on the left and one on the right) and the infrared projector. The projector maps invisible infrared rays that improve the depth accuracy in difficult scenes. The processor, that can process 36 million depth points per second, receives the information from both the left and right lenses and computes the depth values for all pixels in the image correlating the points obtained from the images obtained with the two sensors. The depth value of the pixel is obtained in relation to the parallel plane of the camera and not to the actual camera as shown in the figure .

The D435 calculates depth using stereoscopic matching.

The internal processor converts this disparity into a metric depth value (z) with this formula:

$$z = \frac{f \cdot B}{d}$$

In this formula, f represents the focal length in pixels and B represents the baseline defined as the physical distance between the two IR lenses The error in the disparity space is usually constant for a stereo system.

Figure 7.3: Intel RealSense D435 [3].

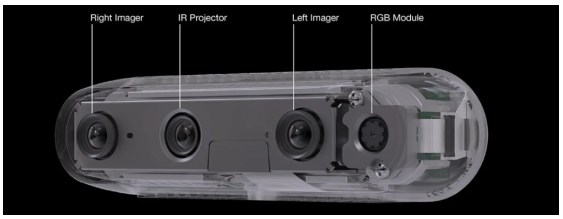


Figure 7.4: Intel RealSense D435 Structure [3].

To transform 3D spatial coordinates into 2D image pixels[15], the digital camera uses an internal calibration matrix (K).

$$K = \begin{bmatrix} \alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

The parameters contained by (K) are split into two components for the x and y axes:

- **Focal Lengths** f_x, f_y: a physical lens has a single focal length, but the digital representation in the K matrix uses two values, denoted as α_x, α_y:
  - **Non-Square Pixels**: These values represent the focal length scaled by the number of pixels per unit distance in the horizontal (m_x) and vertical (m_y) directions.
  - **Mathematical Scale**: $\alpha_x = f \cdot m_x, \alpha_y = f \cdot m_y$. If the sensor's pixels are not perfectly square f_x, f_y will be different to ensure the mapping maintains the correct geometric proportions.
- **Principal Point** x_0, y_0: the principal point is defined as the precise location where the camera's optical axis intersects the image plane.
  - **Origin Offset**: in a digital image, the coordinate origin (0,0) is usually located at the top-left corner or at the centre. The principal point is not always perfectly aligned with the origin of the coordinates of the pixel grid.

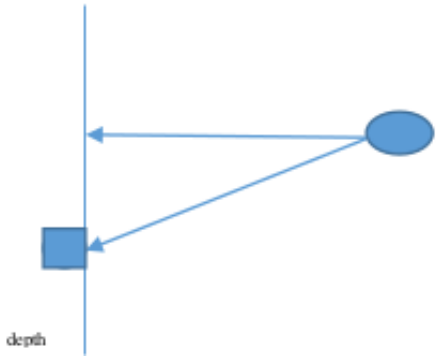


Figure 7.5: Image from that represents how the depth for each pixel is computed in the RealSense [31].

– **2D Coordinates**: consequently, two coordinates x_0, y_0, that represent the horizontal and vertical offset, are required to correctly define the center of projection relative to the image origin, where $x_0 = m_x p_x$ and $y_0 = m_y p_y$.

With the use of these precise intrinsics, which are factory calibrated for every individual unit, the D435 is capable of performing high-speed hardware undistortion and rectification and ensuring a real-time depth stream.

7.3 Node Implementation

This section describes the software architecture of the ROS 2 package developed for the main purpose of the thesis using the Humble Hawksbill distribution. In the figure 7.8 it is possible to see a rqt_graph[33], a feature that provides the visualization of nodes, topics and connections between them. In fact, as previously mentioned, the system is organized in nodes that use topics to send messages to each other. The first node is called *mcap_publisher_node* that publishes three topics

- $/camera/camera/color/camera\_info$: the camera parameters
- $/camera/camera/color/image\_raw$: the rgb image
- $/camera/camera/aligned\_depth\_to\_color/image\_raw$: depth data

The following $/yolo\_qc\_node$ subscribes to these topics and applies the segmentation and pose model to the original image and uses the back-projection algorithm to obtain the dimensions of the apple. The same node publishes messages containing information about the size and pose of the apple on the topic $/apple\_qc/full\_data$ that is subscribed by $/apple\_data\_receiver$ for debug purposes.

7.3.1 Data Retrieval Node

The Data Retrieval Node simulates the functioning of the Intel RealSense Camera by reading a ROS bag from which extracts raw data streams for Color, Depth and Camera Intrinsic Parameters. The streams are then broadcast over ROS 2 topics. The pipeline is compatible with a real-time use where this node is supposed to be replaced by the live streaming of the Intel RealSense camera.

7.3.2 Inference and Radius Estimation Node

This node is the central processing unit of the perception pipeline. The node subscribes to three main ROS 2 topics: Color Image, Depth Image and Camera Info. The RGB and Depth frames need to be synchronized to maintain data integrity. This is done with a filter that ensures that the processing starts only when there is a pair of frames with nearly identical timestamps. The input image is analyzed by:

- the segmentation model that isolates the pixels belonging to each apple, producing a binary mask that defines the precise shape of the apple
- the pose model that identifies keypoints on the apple, used to calculate Theta θ, that represents the orientation of the fruit.

The node computes the median depth of the selected pixels contained in the segmentation mask to exclude the depth inaccuracies typically found at the edges of objects. The camera's intrinsic parameters are used in the following equations to find the coordinates to map a pixel in the 3D space.

$$x = \frac{(u - c_x) \cdot z}{f_x}$$

$$y = \frac{(u - c_y) \cdot z}{f_y}$$

where:

- f_x, f_y represent the horizontal and vertical focal length, the magnification of the lens expressed in pixels units
- c_x, c_y the principal point indicating the pixel coordinates of the optical center of the camera
- u, v the 2D coordinates identifying the center of the apple in the image
- Z represent the distance, or depth, from the camera to the apple

The knowledge of the distance Z and the radius of the apple in pixels rad_{px} is used by the node to compute the physical size rad_{cm} using similar triangles. The code first computes the physical diameter by scaling the pixel width by the depth Z and the focal length f_x. This value is then converted into centimeters before being stores as the final radius.

$$rad_{cm} = \frac{rad_{px} \cdot 2 \cdot Z_m}{f_x} \cdot \frac{100}{2}$$

This formula is necessary for the system to tell the difference between a small apple close to the camera and a large apple far from the lens.

The orientation angle θ is derived from the Keypoints identified by the pose model. The model identifies the stem and the base of the apple. The system treats the apple as an object with a primary axis and defines a vector v in the 2D image plane using two specific keypoints:

- $P_1(u_1, v_1)$: The stem
- $P_2(u_2, v_2)$: The midpoint between the two points that represent the base of the apple

Digital imaging sensors[15] typically use a left-handed coordinate system. This means that the origin of computer images starts at the top-left corner and the Y-value increases when moving downward.

In contrast, the standard used for Euclidean geometry is right-handed systems where the Y-axis points upward.

The recommended practice is to negate the y coordinate of the image points:

$$d_y = -(y_{stem} - ybase)$$

.

This transformation restores a right-handed frame, which is required maintaining consistency during 3D projection. This inversion of coordinates ensures that a point located physically higher in the real world results in a positive d_y value.

Once the displacement components d_x, d_y have been corrected, the object's orientation is calculated using the $atan2$[26].

By swapping to the arguments $atan2(d_x, d_y)$, if an object is perfectly upright $d_x = 0, d_y > 0$, the result is 0 radians. Consequently, any value different from zero of θ directly represents the angular deviation from the ideal vertical axis [34].

Finally, the node publishes a custom AppleData message, that was created to encapsulate the processed information, with the following structure:

- **std_msgs/Header header** : timestamp of the original image capture.

- **geometry_msgs/Point center** : specifies the 3D coordinates of the apple, in meters.
  - **X**: the horizontal offset from the camera's center;
  - **Y**: the vertical offset from the camera's center;
  - **Z**: the distance from the lens to the fruit.
- **float64 radius** : the estimated size of the fruit in centimeters.
- **float64 theta** : the orientation of the apple in radians. If the stem or the base are hidden this value will correspond to 0.0, since the result would not be accurate.

7.3.3 Subscriber node

The final node of the pipeline is the subscriber used for debug purposes. The node subscribes to the topic that sends the custom AppleData.msg.

7.3.4 Debug of the package

In order to check the results and be able to write tests, the package contains debug functions. Specifically, it is useful to write a CSV file and for every message processed, corresponding to the unique id of the apple, it appends a row containing:

- **ID**
- **Timestamp**
- **X_m**
- **Y_m**
- **Z_m**
- **Radius_cm**
- **Theta_rad**

In the output image, the unique Apple ID is visible next to the fruit, to ensure that the same apple is not counted or picked twice, and the arrows associated with θ are drawn in different colors:

- **Green**: when both the stem and the base are visible.
- **Blue**: when only the stem is visible and the base is hidden.
- **Yellow**: when only the base is visible and the stem is hidden.

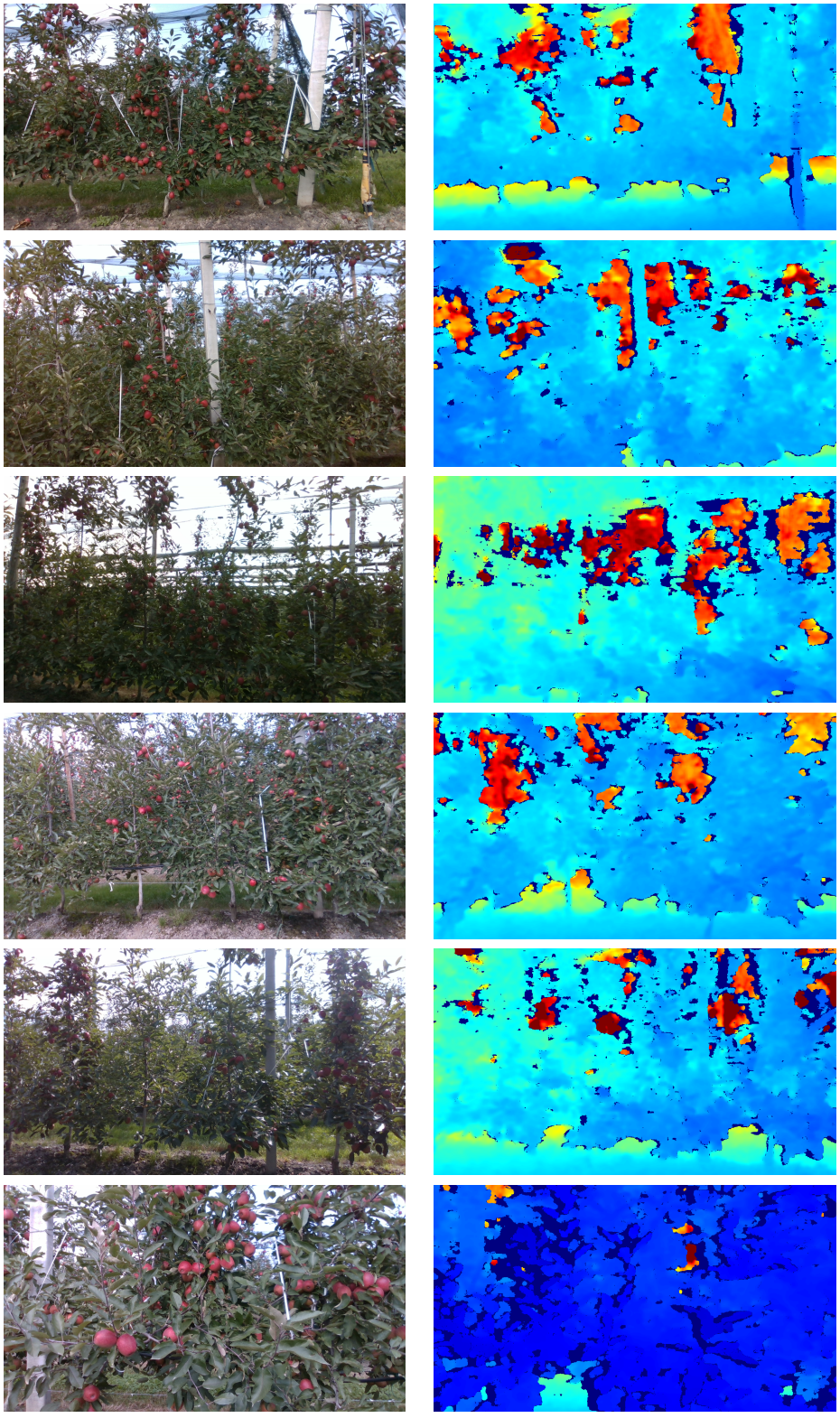


Figure 7.6: Examples of RGB and Depth from a Intel RealSense Camera

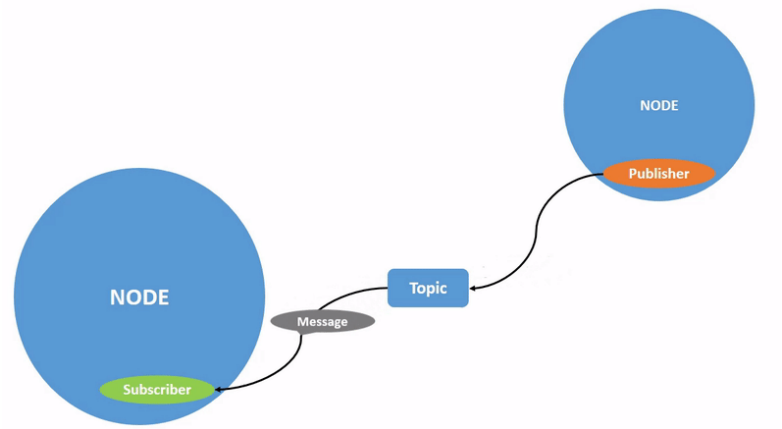


Figure 7.7: The exchange of messages between nodes in ROS2 [33].

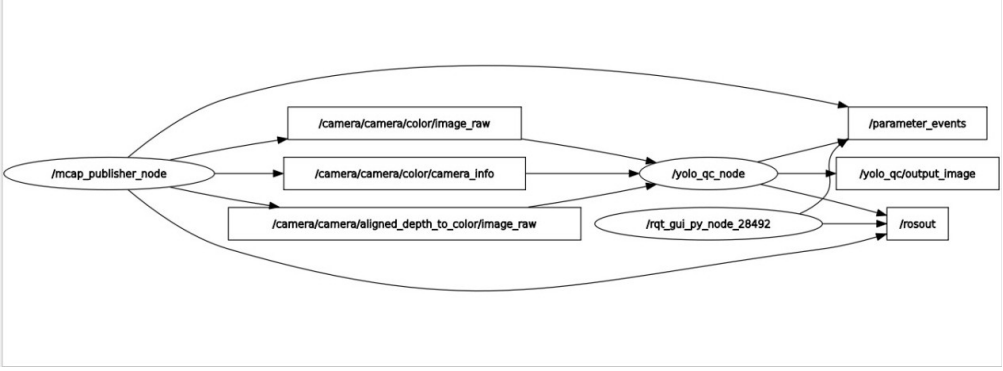


Figure 7.8: rqt_graph representing the pipeline of the program

Figure 7.9: Detail of output image, the green arrows are used when we see both stem and base, while the yellow arrow is for when only the base is visible

Figure 7.10: Examples of output images produced by the pipeline

Chapter 8

Conclusion

This research started with the identification of the steps needed before the harvesting task, that were found in the following pipeline:

- detection and isolation of individual apples;
- estimation of the apple's size;
- estimation of the apple's pose.

In order to understand what were the state-of-art solution in 3D reconstruction, different papers were examined. Photogrammetry solutions were analyzed for the creation of a ground truth mesh of the apple. Subsequently, several deep learning models were tested for the original problem of 3D reconstruction and resulted in an inefficient approach, due to their latency. Finally, different RGB-D approaches that used depth cameras and mathematical equations were analyzed and determined to be the best choice, because they would provide the best trade-off between speed and precision. It was also decided that the reconstruction of a mesh was not necessary, and that the creation of a simple sphere with a precise estimation of the diameter would be enough. In order to detect and identify the exact shape of the individual apple and its pose, some solutions were examined and highlighted that the best option would be the YOLO framework.

In fact, the next step was creating and manually labeling two datasets containing photographs of the orchards and, subsequently, training two YOLOv8 models: one for the segmentation task and the other for the pose task. It was decided to use YOLOv8 small for both models because, also in this case, it was a trade-off between speed and accuracy.

With the use of the Intel RealSense Camera, it is possible to obtain the RGB and depth streams that need to be synchronized. After finding the precise mask of the apple, it was enclosed in the smallest circle that could contain the entire

area of the segmentation. After this step, a back-projection algorithm was used to obtain the metric dimensions of the radius of the apple. The pose was used to draw an arrow from the base of the fruit to the stem that provided the tilt of the fruit.

Several tests regarding the performance of the segmentation model, the pose model and the radius estimation algorithm were conducted and provided good results. The two YOLO models were evaluated using as metrics Precision, Recall, mAP50, mAP50-95 for Bounding Boxes, Masks and Keypoints Pose. The inference time was also evaluated. The results for the estimation of the radius were compared to the measurement of an apple mesh used as ground truth.

This research successfully demonstrated the implementation of a real-time, RGB-D based pipeline for the dimension estimation and pose estimation of apples.

In the future, this approach could be the base for additional improvements in the precision of the shape enclosing the mask or for the latency.

Bibliography

[1] Spyros Fountas, Nikos Mylonas, Ioannis Malounas, Efthymios Rodias, Christoph Hellmann Santos, and Erik Pekkeriet. «Agricultural Robotics for Field Operations». In: *Sensors* 20.9 (2020). ISSN: 1424-8220. DOI: 10.3390/s20092672. URL: https://www.mdpi.com/1424-8220/20/9/2672 (cit. on p. 5).

[2] ROS2 Humble. *ROS 2 Documentation*. URL: https://docs.ros.org/en/humble/index.html (cit. on p. 7).

[3] RealSense. *D435 - RealSense*. URL: https://www.realsenseai.com/products/stereo-depth-camera-d435/ (cit. on pp. 7, 76).

[4] Hasan Surmen. «Photogrammetry for 3D Reconstruction of Objects: Effects of Geometry, Texture and Photographing». In: *Image Analysis & Stereology* 42 (July 2023), pp. 51–63. DOI: 10.5566/ias.2887 (cit. on pp. 9–11).

[5] Tianhao Wu, Chuanxia Zheng, Frank Guan, Andrea Vedaldi, and Tat-Jen Cham. *Amodal3R: Amodal 3D Reconstruction from Occluded 2D Images*. 2025. arXiv: 2503.13439 [cs.CV]. URL: https://arxiv.org/abs/2503.13439 (cit. on pp. 11–14, 20–23).

[6] Dmitry Tochilkin et al. *TripoSR: Fast 3D Object Reconstruction from a Single Image*. 2024. arXiv: 2403.02151 [cs.CV]. URL: https://arxiv.org/abs/2403.02151 (cit. on pp. 13, 14, 19–23).

[7] Zixuan Huang, Stefan Stojanov, Anh Thai, Varun Jampani, and James M. Rehg. *ZeroShape: Regression-based Zero-shot Shape Reconstruction*. 2024. arXiv: 2312.14198 [cs.CV]. URL: https://arxiv.org/abs/2312.14198 (cit. on pp. 15–17, 19–23).

[8] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. *Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images*. 2018. arXiv: 1804.01654 [cs.CV]. URL: https://arxiv.org/abs/1804.01654 (cit. on pp. 16, 17, 19–23).

[9] Bin Yan and Xiameng Li. «RGB-D Camera and Fractal-Geometry-Based Maximum Diameter Estimation Method of Apples for Robot Intelligent Selective Graded Harvesting». In: *Fractal and Fractional* 8.11 (2024). ISSN: 2504-3110. DOI: 10.3390/fractalfract8110649. URL: https://www.mdpi.com/2504-3110/8/11/649 (cit. on pp. 23, 24, 26, 27).

[10] Rosa Pia Devanna, Francesco Vicino, Simone Pietro Garofalo, Gaetano Alessandro Vivaldi, Simone Pascuzzi, Giulio Reina, and Annalisa Milella. «Automated On-Tree Detection and Size Estimation of Pomegranates by a Farmer Robot». In: *Robotics* 14.10 (2025). ISSN: 2218-6581. DOI: 10.3390/robotics14100131. URL: https://www.mdpi.com/2218-6581/14/10/131 (cit. on pp. 25, 27).

[11] Zhenglin Wang, Kerry B. Walsh, and Brijesh Verma. «On-Tree Mango Fruit Size Estimation Using RGB-D Images». In: *Sensors* 17.12 (2017). ISSN: 1424-8220. DOI: 10.3390/s17122738. URL: https://www.mdpi.com/1424-8220/17/12/2738 (cit. on pp. 26, 27).

[12] Ranjan Sapkota, Dawood Ahmed, and Manoj Karkee. «Comparing YOLOv8 and Mask R-CNN for instance segmentation in complex orchard environments». In: *Artificial Intelligence in Agriculture* 13 (2024), pp. 84–99. ISSN: 2589-7217. DOI: https://doi.org/10.1016/j.aiia.2024.07.001. URL: https://www.sciencedirect.com/science/article/pii/S258972172400028X (cit. on pp. 27, 28, 30–33).

[13] Ranjan Sapkota and Manoj Karkee. *YOLO11 and Vision Transformers based 3D Pose Estimation of Immature Green Fruits in Commercial Apple Orchards for Robotic Thinning*. 2025. arXiv: 2410.19846 [cs.CV]. URL: https://arxiv.org/abs/2410.19846 (cit. on pp. 31, 33–35).

[14] Ranjan Sapkota and Manoj Karkee. *Comparing YOLOv11 and YOLOv8 for instance segmentation of occluded and non-occluded immature green fruits in complex orchard environment*. 2025. arXiv: 2410.19869 [cs.CV]. URL: https://arxiv.org/abs/2410.19869 (cit. on pp. 35, 36).

[15] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004 (cit. on pp. 37, 48–50, 59, 60, 76, 79).

[16] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 2nd. O'Reilly Media, 2019 (cit. on pp. 37–46, 52, 55, 56, 63).

[17] Ranjan Sapkota and Manoj Karkee. *Ultralytics YOLO Evolution: An Overview of YOLO26, YOLO11, YOLOv8 and YOLOv5 Object Detectors for Computer Vision and Pattern Recognition*. 2025. arXiv: 2510.09653 [cs.CV]. URL: https://arxiv.org/abs/2510.09653 (cit. on pp. 46, 47, 65).

[18] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV]. URL: https://arxiv.org/abs/1506.02640 (cit. on pp. 52, 55, 56).

[19] Ultralytics. *Data Augmentation using Ultralytics YOLO*. URL: https://docs.ultralytics.com/guides/yolo-data-augmentation/ (cit. on pp. 52, 54–56).

[20] Teerath Kumar, Alessandra Mileo, Rob Brennan, and Malika Bendechache. *Image Data Augmentation Approaches: A Comprehensive Survey and Future directions*. 2023. arXiv: 2301.02830 [cs.CV]. URL: https://arxiv.org/abs/2301.02830 (cit. on pp. 52, 55, 56).

[21] Ultralytics. *Configuration*. URL: https://docs.ultralytics.com/usage/cfg/ (cit. on p. 54).

[22] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. *YOLACT: Real-time Instance Segmentation*. 2019. arXiv: 1904.02689 [cs.CV]. URL: https://arxiv.org/abs/1904.02689 (cit. on pp. 54, 64).

[23] Ultralytics. *Model Training using Ultralytics YOLO*. URL: https://docs.ultralytics.com/modes/train/ (cit. on pp. 55, 57).

[24] Debapriya Maji, Soyeb Nagori, Manu Mathew, and Deepak Poddar. *YOLO-Pose: Enhancing YOLO for Multi Person Pose Estimation Using Object Keypoint Similarity Loss*. 2022. arXiv: 2204.06806 [cs.CV]. URL: https://arxiv.org/abs/2204.06806 (cit. on pp. 56, 63).

[25] Open CV. *Structural Analysis and Shape Descriptors*. URL: https://docs.opencv.org/4.x/d3/dc0/group__imgproc__shape.html#ga8ce13c24081bbc7151e9326f412190f1 (cit. on pp. 60, 61).

[26] Python Documentation. *math - Mathematical Functions*. URL: https://docs.python.org/3/library/math.html#math.atan2 (cit. on pp. 61, 79).

[27] Polycam. *Polycam*. URL: https://poly.cam/tools/photogrammetry (cit. on p. 67).

[28] Blender. *Blender*. URL: https://www.blender.org/ (cit. on p. 67).

[29] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. «Robot Operating System 2: Design, architecture, and uses in the wild». In: *Science Robotics* 7.66 (May 2022). ISSN: 2470-9476. DOI: 10.1126/scirobotics.abm6074. URL: http://dx.doi.org/10.1126/scirobotics.abm6074 (cit. on pp. 72–74).

[30] Steve Macenski, Alberto Soragna, Michael Carroll, and Zhenpeng Ge. *Impact of ROS 2 Node Composition in Robotic Systems*. 2023. arXiv: 2305.09933 [cs.RO]. URL: https://arxiv.org/abs/2305.09933 (cit. on p. 72).

[31] Vladimir Tadic, Ákos Odry, Istvan Kecskes, Ervin Burkus, Zoltán Király, and Peter Odry. «Application of Intel RealSense Cameras for Depth Image Generation in Robotics». In: *WSEAS Transactions on Computers* 18 (Sept. 2019), pp. 107–112 (cit. on pp. 75, 77).

[32] Leonid Keselman, John Iselin Woodfill, Anders Grunnet-Jepsen, and Achintya Bhowmik. *Intel RealSense Stereoscopic Depth Cameras*. 2017. arXiv: 1705.05548 [cs.CV]. URL: https://arxiv.org/abs/1705.05548 (cit. on p. 75).

[33] ROS2 Humble. *Understanding Topics - ROS2 Documentation Humble*. URL: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html (cit. on pp. 77, 82).

[34] H. van der Marel. *Reference Systems for Surveying and Mapping*. Faculty of Civil Engineering and Geosciences Delft University of Technology, 2014 (cit. on p. 79).