POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Autonomous robotic strawberry harvesting: from perception to trajectory planning

Tesi di Laurea Magistrale in
Mechanical Engineering - Ingegneria Meccanica

Author: **Alessandra Tafuro**

Student ID: 944017
Advisor: Prof. Andrea Maria Zanchettin
Co-advisors: Prof. Amir M. Ghalamzan
Academic Year: 2020-21

"No limits. Just horizons."

A te, nonna.

Abstract

Nowadays, the agricultural chain's dependency on human labor turned out to be risky. The worldwide demand for agricultural products is rapidly increasing due to the growing population, but, in parallel, the number of rural laborers is declining chronic everywhere or can be compromised by exceptional events, like the recent pandemic. Automation of agriculture can be an important solution to tackle the increasing load on farming businesses forced to deal with the aforementioned issues. Strawberry is among high-value crops with considerable harvesting costs highly dependent on human labour, but a fully viable commercial robotic system for autonomous picking has yet to be established. This thesis deals with some main problems for the development of a successful strawberry harvesting technology: ready-to-be-picked strawberry detection, key-points localisation for grasping and picking actions, fruit weight estimation before picking, and path planning from visual information to reach the target fruit. Strawberries are subject to chaotic configurations from the organic cluster formations, causing obstructions to harvest-ready fruits and this poses a great problem for visual localisation. Moreover, picking and grasping points detection is a needed perception component for a successful selective harvesting robot. These problems have been addressed with Deep Learning (DL) and in particular, *Detectron-2* [133], a next-generation open-source object detection system from Facebook AI Research based on *Mask Region-based Convolutional Neural Network (Mask R-CNN)* [52], has been trained to segment berries, classify them as ripe or unripe and to detect the key-points for picking and grasping action. Strawberry weight estimation has been achieved in the Machine Learning framework implementing a *Random Forest Model [20] with Decision Trees* [70, 92] which takes as input a vector including features extracted from RGB and depth data of the berries whose weight has to be estimated. This approach has been proven to outperform many other state-of-the-art DL approaches. Weight estimation of the strawberries before picking can help sort the fruits in the correct punnet right after picking, that meets the legislation for immediate packaging, which results in minimum fruit touch and in the subsequent decrease of the added packaging costs during post-harvesting processes. Once the fruit perception problem is addressed, the ultimate goal for the robotic system is the ability to efficiently and successfully reach to pick ripe strawberries. This

has been solved by proposing a new probabilistic neural network architecture, called *deep Probabilistic Movement Primitives (deep-ProMP)*, that generates successful reach-to-pick trajectories distributions from visual camera information only, trained in a Learning from Demonstrations (LfD) [118] setting. A few model architectures have been presented, namely, *Deep-ProMP-AE, Deep-ProMP-VAE,* and *Deep-ProMP-cVAE* which all have a two-fold design: from the input image to a latent space representation and from the latent representation to the desired trajectories distribution. This architecture design has been compared against the direct mapping from the RGB image to the trajectory space to show its superiority. A probabilistic approach has been preferred over a deterministic one for trajectory prediction since this setting can be exploited in future developments to sample from the predicted distribution to optimise the trajectory to meet secondary objectives, like collision avoidance. Finally, a novel domain-specific latent space training has been proposed to learn the latent space representation of the input image in a way that is relevant both for computer vision and for the specific robotic task.

Keywords: Autonomous robotic strawberry harvesting, Object detection trough keypoints, Fruit weight estimation, Learning from Demonstration, Path planning, Latent space learning.

Abstract in lingua italiana

Al giorno d'oggi, la dipendenza della filiera agricola dal lavoro umano si è rivelata essere rischiosa. La domanda mondiale di prodotti rurali è in rapida crescita a causa dell'aumento della popolazione, ma, parallelamente, il numero di lavoratori agrari è in calo cronico ovunque o può essere compromesso da eventi eccezionali, come la recente pandemia. L'automazione dell'agricoltura può costituire un'importante soluzione per affrontare il carico crescente sulle imprese agricole, costrette a far fronte ai suddetti problemi. Nonostante la fragola sia tra le colture di alto valore con costi di raccolta considerevoli e fortemente dipendenti dal lavoro umano, deve ancora essere sviluppato un sistema robotico commercializzabile per la raccolta autonoma. Questa tesi affronta alcuni tra i principali problemi per lo sviluppo di una tecnologia di raccolta autonoma delle fragole: il rilevamento delle fragole mature, la localizzazione dei punti chiave per l'azione di raccolta, la stima del peso dei frutti, e la pianificazione della traiettoria della mano robotica dalle informazioni visive per raggiungere il frutto da raccogliere. Le fragole sono soggette a configurazioni caotiche e a formazioni organiche a grappolo, che rendono poco visibili i frutti maturi e questo rappresenta un grosso problema per la localizzazione a carico della tecnologia di visione artificiale. Inoltre, anche la localizzazione dei punti di taglio del frutto è una componente di percezione necessaria. Questi problemi sono stati affrontato con tecniche di Deep Learning (DL) ed in particolare con *Detectron-2* [133], un sistema di rilevamento di oggetti open-source di nuova generazione creato da Facebook AI Research, basato su *Mask Region-based Convolutional Neural Network (Mask R-CNN)* [52], il quale è stato addestrato per eseguire la segmentazione delle fragole, la classificazione in mature o acerbe e a rilevare i punti chiave necessari per la raccolta del frutto. La stima del peso delle fragole è stata ottenuta nell'ambito del Machine Learning, implementando un *Random Forest Model [20] with Decision Trees* [70, 92], il cui input è un vettore che include le caratteristiche estratte dai dati RGB e di profondità delle fragole il cui peso deve essere stimato. Questo approccio è stato dimostrato essere più accurato di altre tecniche recenti basate sul DL. La stima del peso delle fragole può aiutare a smistare i frutti nei corretti contenitori subito dopo la raccolta, il che soddisfa la legislazione per il confezionamento immediato, il quale implica un tocco minimo del frutto e una

riduzione dei costi di confezionamento aggiunti durante lo stadio post-raccolta. Una volta affrontato il problema dell'individuazione del frutto da raccogliere, l'obiettivo finale del sistema robotico è la capacità di raggiungerlo in modo efficace. Questo problema è stato risolto proponendo un nuovo design probabilistico di rete neurale, chiamato *deep Probabilistic Movement Primitives (deep-ProMP)*, che genera distribuzioni di traiettorie dalle sole informazioni visive della telecamera, allenato in un'impostazione di Apprendimento tramite Dimostrazioni [118]. Sono state presentate diverse architetture per il modello proposto, vale a dire *Deep-ProMP-AE, Deep-ProMP-VAE* e *Deep-ProMP-cVAE* che hanno tutte un duplice design: dall'immagine di input alla rappresentazione nello spazio latente e dalla rappresentazione latente alla distribuzione delle traiettorie. Questo design è stato confrontato con la mappatura diretta dall'immagine RGB allo spazio delle traiettorie per mostrarne la superiorità. Inoltre, un approccio probabilistico è stato preferito rispetto ad uno deterministico poiché questa impostazione può essere sfruttata in sviluppi futuri per campionare dalla distribuzione di traiettorie prevista e ottimizzare rispetto ad obiettivi secondari (come evitare le collisioni). Infine, è stato sviluppato un nuovo metodo di regolazione dello spazio latente specifico al dominio di applicazione, per apprendere la rappresentazione dell'immagine di input in funzione sia della visione computerizzata che dello specifico compito robotico.

Parole chiave: Raccolta autonoma roboticizzata delle fragole, Rilevamento di oggetti attraverso punti chiave, Stima del peso dei frutti, Apprendimento tramite dimostrazioni, Pianificazione della traiettoria, Apprendimento dello spazio latente.

Contents

1 | Introduction

1.1. Robotic technologies for autonomous strawberry harvesting: general overview of the problem

Developing robotic technologies for selective harvesting of high-value crops, such as strawberries, has become highly demanded because of different social, political, and economical factors [29], such as labour shortage, recent COVID-19 pandemic, and the increase of global population.

According to the International Labor Organization (ILO), agricultural laborers, as a percentage of the workforce, declined from 81.0% to 48.2% in developing countries and from 35.0% to 4.2% in developed ones since 2014 [6]. The shortage of people working in farms is growing chronic everywhere. In the Asia Pacific, especially in Japan alone, the number of people working in farms dropped from 2.2 million in 2004 to 1.7 million in 2014. Such a huge decline in the workforce of about 12.8% is also observed in the European agriculture sector [6].

In Fig. 1.1 the world share of the labour force employed in agriculture is shown as a heat map, in the years 1991 and 2019. It is evident how the agricultural labour force is decreased over the years.

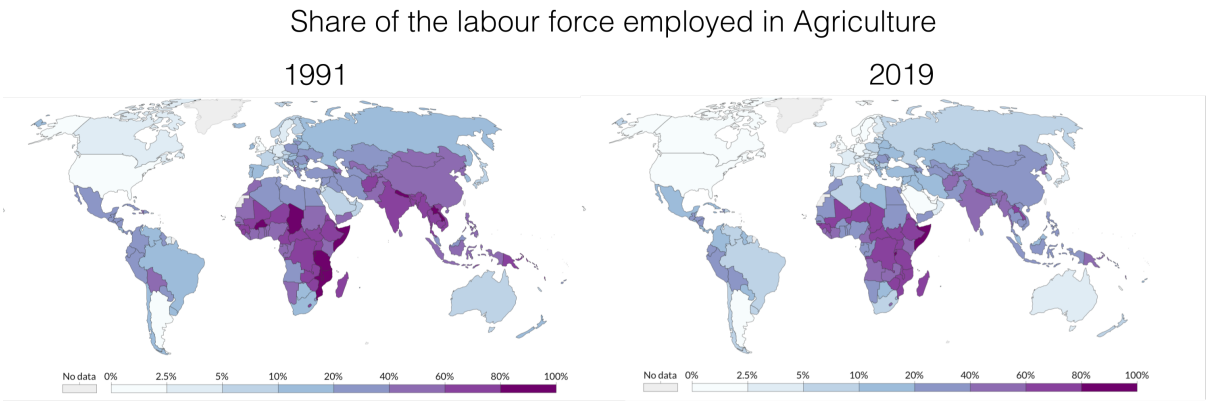


Figure 1.1: Share of the labor force employed in agriculture in 1991 vs 2019 [112].

Additionally, the open borders that had become nearly self-evident in most of the EU's territory, have largely been closed in the period of Covid-19 pandemic, as member states introduced travel restrictions to stem the further spread of the disease. This has left farmers in western Europe struggling to bring in the tens of thousands of seasonal workers on which they rely to pick their rapidly ripening fruits and vegetables. According to farming association Coldiretti, the more than 370,000 seasonal workers who travel to the country each year - primarily from Romania, Bulgaria, and Poland - produce more than a quarter of all Italian food [2]. The European Commission president said that Europe (and the rest of the world) must prepare all sectors to cope with an "era of pandemics" [38], meaning that this problem is not to be considered limited to the past Covid-19 pandemic.

Moreover, with a global population projection of 9.7 billion people by 2050, agricultural production will need to increase of at least 70% from current levels to serve nutritional trends. Now more than ever, the pressure on farmers to produce nutritious products is putting our planet's health under even more stress [65].

Also the climate change plays a role in this framework. Models vary, but the World Bank (2010) estimates that to meet the growing demand for food between 2005 and 2055, agricultural productivity will need to rise by 64% under the assumptions of the "business-as-usual" (no climate change) scenario and by a further 80% to offset the projected stresses arising from climate change (Fig. 1.2). However, the model indicates that if population remained constant at the 2005 level, agricultural productivity would need to rise only 25% under the "business-as-usual" scenario; that is, more of the required productivity increase under the "business-as-usual" scenario is necessitated by population growth than by increases in consumption per capita [27].

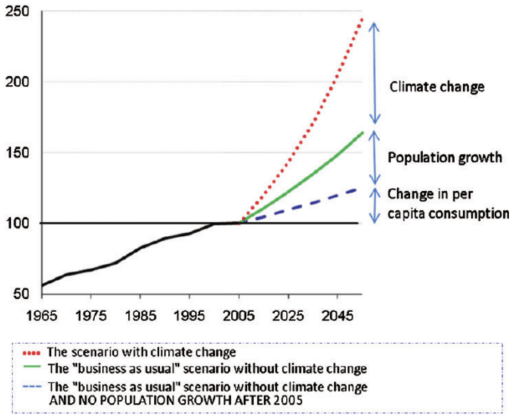


Figure 1.2: Required growth in agricultural productivity given estimated population Growth, increase in per capita consumption, and climate change [27].

Figure 1.3: Robotic technologies for autonomous strawberry harvesting

All these aspects underline that the dependency of the agricultural chain on human labor turns out to be very risky as the farming business is going to be pressurised and forced to deal with labor shortages due to different factors.

Robotic methods are therefore in development for soft fruits harvesting (Fig. 1.3) and other aspects of agriculture, as a possible solution for ensuring a more sustainable food chain, and as means of providing cheap alternative sources of labor globally. Agriculture is a perfect niche for innovations in the sphere of robotics: farmers usually have to deal with repetitive tasks in the field, and this work is primarily labor-intensive.

Smart farming is an evolving concept that refers to managing farms through incorporating current information and communication or advanced AI technologies to bring rapid growth in the quantity and quality of products while optimizing human labor [3]. As such, private and public sectors have invested across the globe to develop robotic harvesting and commercialise the corresponding technologies in the last few years (Fig. 1.4). Only EU invested above €5 million in BACHUS [30] project to develop robotic technology for grape and olive picking and above €4 million in SWEEPER [7] for Sweet pepper harvesting robotic technology. The global harvesting robot market size is expected to reach US$1,827.9 Million by 2027 [3].

Strawberries are farmed extensively in most parts of the world, growing either outdoors in open fields or in controlled environments, like greenhouses or polytunnels. It is among high-value crops with considerable selective harvesting costs in production. With a total retail value of $17 Billion globally, above $1 Billion is only the picking cost [5]. Strawberry production is heavily reliant on human labor, especially for harvesting [137]. It was reported that 25% of all working hours in Japan are consumed by harvesting operations

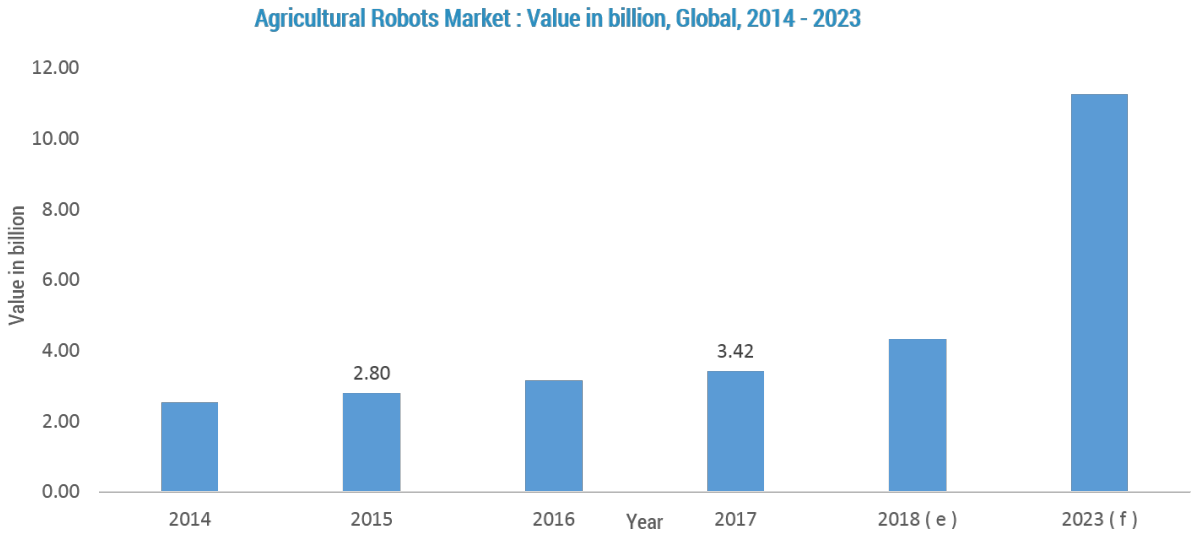


Figure 1.4: Agricultural robotics market trend from 2014 to 2023. The market is valued at USD 3.42 billion in 2017 and is expected to register a CAGR of 21.1% [3].

[139]. Despite several attempts to develop a robotic solution for harvesting strawberries and many other crops, a fully viable commercial system has yet to be established [122].

A major challenge for these robotic systems is that strawberries are subject to chaotic configurations from the organic cluster formations, causing obstructions to ripe and harvest-ready strawberries (Fig. 1.5), as well as complex non-linear interaction dynamics between the fruits, stems, leaves, and the robotic agents [68, 121]. Other factors such as delicate manipulations are also required to avoid bruising or damaging the fruits, to reduce waste.

Human intelligence and dexterity can easily accommodate this. However, robots require tightly integrated systems that incorporate perception, motion planning, and manipulation to be able to operate robustly with high efficiency in these settings [82, 136]. These robots are also required to operate at high speed, with high accuracy and robustness and at a low cost, all features that are especially challenging in unstructured environments, such as strawberry farms [135].

It is in the aforementioned setting that the author wants to contribute with the work presented in this thesis. In particular the problems that have been addressed are:

- **Perception problem (1):** ripe strawberries localization and picking point estimation;
- **Fruit weight estimation (2):** evaluation of the weight of the fruits before picking to directly sort berries into punnets of constant weight without additional handling;
- **Path planning (3)** to reach the ready to be picked fruit with the robotic hand.

Figure 1.5: Clusters of strawberries in a poly-tunnel farm.

1.2. Problem statement and thesis research aim

This thesis deals with three main problems related to a successful robotic technology for selective harvesting of strawberries.

The first one is a pure Computer Vision (CV) problem related to the **perceptive ability of the robotic system to detect the ready-to-be-picked fruits (1)**. Machine vision is an essential component for agricultural robots, enabling them to detect and localize the target crop [135]. Unstructured growing conditions, including variable clustering, occlusions, and varying lighting conditions, have been considered as the common challenges for fruit detection in farm environments [122]. Commercially available depth sensors, e.g Realsense *D435i*, also make the perception challenging as they are designed for large objects 3-D perception and controlled lighting conditions. For small fruits under outdoor lighting, the depth maps are not precise. Moreover, picking and grasping points localisation is a needed perception component for a successful selective harvesting robot. Consequently, the focus of much ongoing research, comprised of this one, is novel ways to resolve these situations.

The second proposal of this thesis regards an algorithm for **berries weight estimation before picking (2)**. Normally, after the picking action of the ripe strawberries from the plant, the harvested fruits are directly placed into punnets that are roughly required to be of the same weight. Human harvesters rely on their experience to estimate the strawberry weights. Clearly, this procedure can be imprecise, and subsequently, a lot of effort is required to re-sort the strawberries into punnets, increasing the production cost. Moreover, weighting strawberries after harvesting requires additional handling and maneuvering of the delicate fruits, which run the risk of being bruised. Developing an

AI technology able to estimate weight from the camera visual information constitutes a solution for the listed issues.

The last problem considered is the robot's ability to efficiently and successfully reach-to-pick harvest-ready strawberries regardless of obstructions and cluster configurations. The objective is to automatically **generate a path in the manipulator workspace to reach the ripe target fruit (3)** detected by the visual system using as input the camera visual information of the surrounding environment.

These three topics are individually considered in the following chapters and the relative work logical scheme is shown in Fig. 1.6.

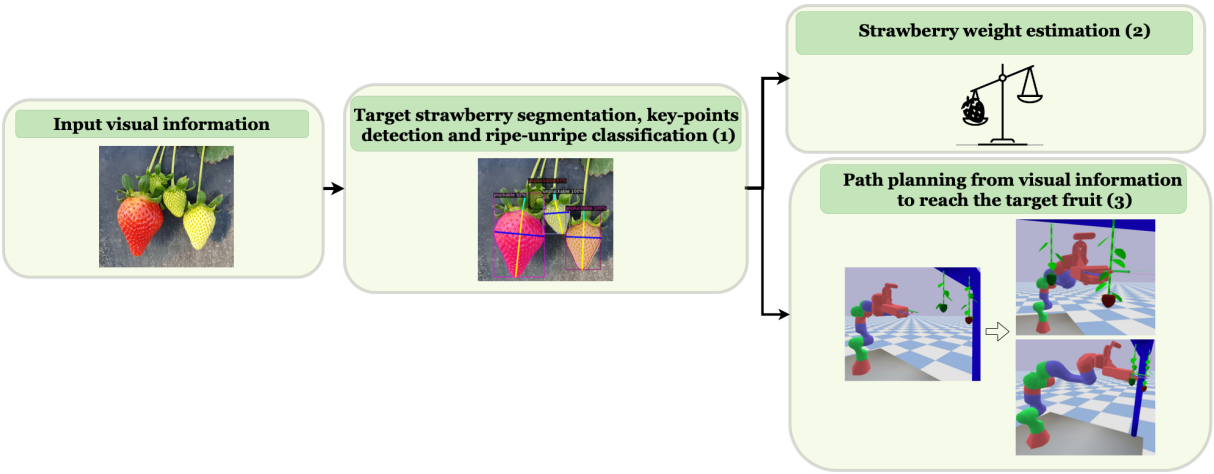


Figure 1.6: Thesis logical scheme.

1.3. Thesis structure

The remaining part of the thesis is structured into 5 chapters and a conclusive section (Fig. 1.7). These are:

- **Chapter 2:** current state-of-the-art works related to the problem of fruit detection, weight estimation, and trajectory planning through Learning from Demonstrations (LfD) are discussed;
- **Chapter 3:** the solution to the problem of strawberry segmentation, key-points detection and classification is detailed;
- **Chapter 4:** it includes the methods developed for strawberry weight estimation from visual sensors and the experimental results;
- **Chapter 5:** deals with the path planning problem to generate successful reach-to-

pick trajectories from visual camera information only. The mathematical formulation of the problem and the details of the implementations are provided;

- **Chapter 6:** it includes the experimental results both in simulation and with the real robotic arm for the proposed path planning approach;
- **Conclusion and future developments.**

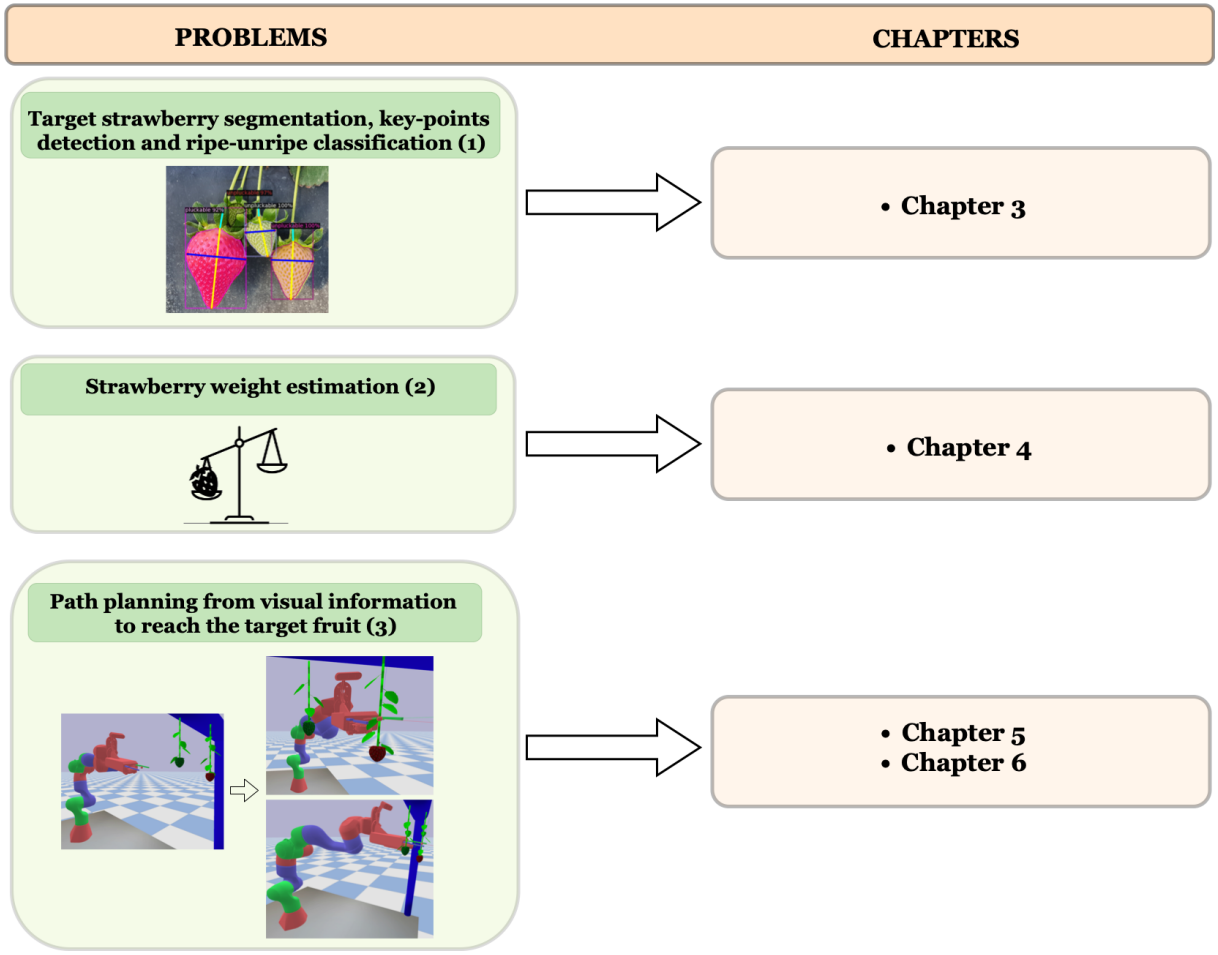


Figure 1.7: Thesis problems and relative chapters.

2 | State of the Art

In this chapter, current state-of-the-art works related to the problem of fruit detection, weight estimation, and trajectory planning through Learning from Demonstrations (LfD) are discussed to provide the baselines of the work included in this thesis.

2.1. Fruit detection, semantic segmentation, and key-points identification

Different fruit detection and localisation approaches exist for the successful application of machine vision systems for robotic fruit harvesting. These are mainly classic computer vision (CV)-based operations and modern state-of-the-art Convolutional Neural Networks (CNNs). CNNs are artificial Neural Network architectures able to deal with 2D-representations of data, like images, and for this reason they have become important in CV applications. While classical methods (including morphology, colour-based, thresholding, and geometrical approaches) provide good performance, other state-of-the-art approaches in CV and deep learning (DL) have been able to yield improved performance.

In the category of **classical colour-based methods** [22, 23, 47] (which select an optimal gray-level threshold value for separating objects of interest in an image from the background, based on their gray-level distribution), Rajendra et al. [105] converted strawberries images from RGB (red, green, blue) to HSI (hue, saturation, intensity) colour map to manually set a threshold for ripe berries. The authors also proposed diameter thresholding for peduncle detection of strawberries. Zhuang et al. [148] implemented automatic thresholding algorithms based on Otsu's thresholding method [91] for more robust thresholding. Arefi et al. [14] adopted colour-based segmentation to remove background and keep the fruit blob. Colour information can also be exploited with other features for a more robust approach. Tao et al. [126] employed colour with geometric features for apple classification with Genetic Algorithm and Support Vector Machine (GA-SVM). Lehner et al. [69] utilized colour-based segmentation with 3D parametric model-fitting for localisation of sweet peppers. Zhuang et al. [148] improved colour-based segmentation through

(a) Peduncle angle calculation from the geometry of strawberry [48].

(b) Apple recognition result in point cloud from [126].

Figure 2.1: Examples of traditional CV-based methods for fruit detection.

iterative-Retinex algorithm [67] followed by Otsu's thresholding. Arefi et al. [14] used the well-known watershed algorithm [111] (it treats the image it operates upon like a topographic map, with the brightness of each point representing its height, and finds the lines that run along the tops of ridges to detect different objects) to extract the morphology of tomatoes from binary images obtained by colour-thresholding. Huang et al. [57] applied erosion and dilatation operations (dilation adds pixels to the boundaries of objects in an image, while erosion removes pixels on object boundaries) to refine strawberry from the colour segmented binary mask. Li et al. [75] implemented morphological operations for twig detection to prune fruitless twigs for litchi harvesting. The connected component algorithm (it groups together pixels belonging to the same connected component, e.g. object) was utilized by Duran et al. to identify strawberry blobs [31]. Hayashi et al. [48] took advantage of the geometry of strawberry for calculating peduncle angle with respect to the vertical line for picking point localisation (Fig. 2.1a). [126] exploited a parameterised query of the spatial differences between a point and its adjacent area to form a Fast Point Feature Histogram (FPFH) descriptor (Fig. 2.1b). The FPFH descriptor formed a multidimensional histogram to describe the geometric properties within the K-neighborhood of a point. This offered the advantages of rotation invariance and good robustness under different sampling densities and noise levels [126].

However, the **inability to generalise and being prone to noise** are among the weaknesses of colour thresholding, geometry or morphology-based algorithms, and other traditional approaches. Moreover, CV engineers need to handcraft features and, as the variation in data increases, it becomes a cumbersome and infeasible task [93].

In recent years, DL has been demonstrated to be superior for tasks such as segmentation [53] and key-points detection [21]. Thus, authors in selective harvesting have begun to adopt some of the **DL techniques for fruit perception**. Lamb et al. [66] trained CNN for strawberry detection by optimising the network through input compression,

image tiling, colour masking, and network compression. Liu et al. [78] employed CNN in combination with depth data to calculate the relative 3-D location of fruit. Similarly, Zhang et al. [143] adopted CNN for tomato classification. [41] used spectral features together with CNN for strawberry quality or ripeness detection. CNN model has also been applied for pear bruise detection based on thermal images [141].

While CNN models perform well in image-specific tasks such as classification, for the pixel-wise understanding of images (semantic segmentation) Regional-CNNs (RCNNs) [45] (a special CNN architecture designed for segmentation and object detection) are more successful. Sa et al. [114] detect fruit using the fusion of faster-RCNN [109], RGB and Infrared (IR) images. Liu et al. [80] combined both YOLOv3 [108] and Mask-RCNN (MRCNN) [54] with ResNet-52 and ResNet-150 [51] as backbone for bounding box detection for citrus fruit harvesting. Among the three models tested, MRCNN with ResNet-150 as backbone provided the best performance. RCNN, MRCNN, faster-RCNN, and ResNets model architectures will be deepened in Sec. 3.2.1.

Researchers also started to combine the information extracted from RCNNs [45] with their algorithm to improve the estimation of picking points [42, 79]. Ge et al. [42] extracted strawberry pixels with MRCNN. Then, the extracted strawberry pixels were combined with depth data, density-based clustering, and Hough transformation to develop a more robust scene understanding. Similarly, Perez et al. [98] exploited MRCNN for strawberry segmentation for harvesting. Researchers have also employed RCNN in combination with other methods to improve the overall accuracy. For instance, Liu et al. [79] implemented MRCNN with the logical green operator to improve the overall performance for cucumber detection. Ganesh et al. [40] chose a combination of HSV and RGB images to improve the

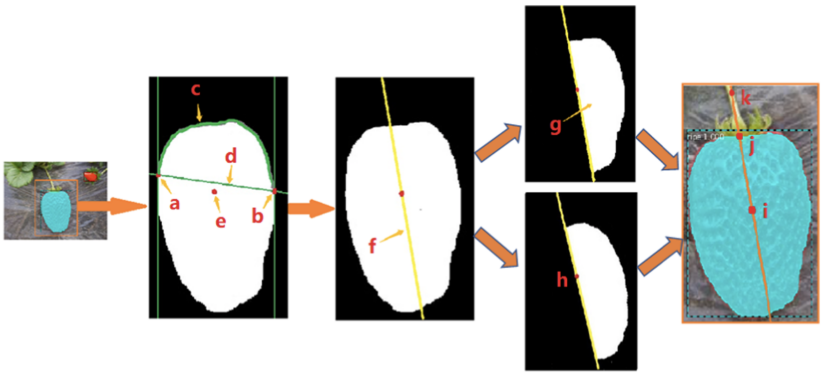


Figure 2.2: Summary of parameters for the localization of the fruit picking point from [140]: a. minimum of contour coordinate, b. maximum of contour coordinate, c. contour of interest, d. dividing line, e. barycenter, f. fruit axis, g. sample A, h. sample B, i. barycenter, j. vertex of contour, and k. picking point.

overall performance of MRCNN for orange detection. Yu et al. [140] first exploited MR-CNN to segment strawberry images and then applied geometrical calculations to localise the picking point (Fig. 2.2).

As seen, MRCNN has become the de facto standard for successful object detection, and this is the reason why it has been chosen also in the case of strawberry perception, as it will be seen in Chapter 3.

2.2. DL and CV-based fruit characteristics regression

As it is an important feature for robotic automation in the field of agriculture, several kinds of systems able to determine volume, surface area, and mass information of agricultural food products have been developed based on computer vision techniques. Computer vision systems mainly focus on features such as maturity, color, defects, and size of the fruit [59] for the automation of fruits and vegetable sorting. The size of a fruit corresponds to its volume, surface area, and mass [90]. Automating the fruit sorting operation can be very valuable in terms of manual labour cost reduction and to avoid excessive fruit handling and the usage of various tools that require dedicated human efforts. [84].

In 1996 the Agricultural Machinery Laboratory of Miyazaki University developed a machine vision computer software to separate strawberries into classes [87]. To have an accurate shape and size judgment it strictly requires that the strawberry is laid in a straight and forward direction with respect to the camera. At angles, more than 15 degrees to the right or left of the mentioned direction misjudgments result. To overcome these limits, Nagata et al. (1997) [86] investigated image processing and analysis to sort fresh strawberries based on size and shape independently from the fruit direction. Namely, the area ratio RA and area A extracted from the strawberry image were computed for this purpose, together with a series of geometrical assumptions. In 2013, Prabha et al. [100] developed a system able to estimate the maturity of bananas using traditional image-processing techniques based on the extraction of the fruit size and color from images. They categorized the maturity of bananas into three types (over-mature, mature, and under-mature). [33] implemented a color-grading method to estimate the maturity and the quality of date fruits using 2-D histograms of colors in the grading category to identify the co-occurrence frequency. In 2015, [46] introduced a method to determine the sweet lime maturity using RGB image processing. However, most of these methods are based on thresholds, in terms of shape, color, and size.

In 2014, Yamamoto, Kyosuke et al. [138] utilized **machine learning (ML)** (computer systems that are able to learn and adapt without following explicit instructions, by using

algorithms and statistical models to analyse and draw inferences from patterns in data) to identify tomato organic product development stages without a threshold value. Their system was composed of three parts: an "X" means clustering, pixel-based, and blob-based segmentation. In 2020, Faisal et al. [35] introduced the IHDS system, which consists of six DL systems to estimate seven maturity stages. The IHDS system was based on date fruit bunches in orchard datasets [12] and achieved accuracy of 99.4 %, F1 score of 99.4 %, 99.7 % recall, and 99.7 % precision.

Several approaches specifically developed for fruit weight estimation using image processing techniques have also been introduced. Some of the studies directly estimate the fruit weight using **traditional image processing techniques**. Eoh, C., and AR Mohd Syaifudin [125] determined the linear relation between the measured area and the actual weight of the mangoes with linear regression techniques. Other studies, as [76] and [74], started from 3-D images gathered by multiple cameras to estimate the weight of mango fruit. A more mathematical approach was introduced by Dang, Nhan T., et al. [26] in 2016. Here the volume is estimated using the 3D bounding box of the mangoes and the Monte Carlo.

Recently **CNN has been adopted for food volume estimation too**. In 2017, Liang et al. [49] introduced faster-RCNN based regression to estimate the food volume with an average error of 20%. In 2018, Li et al. [123] developed a CNN system for volume estimation using two phases. First, the fruit is recognized by a pre-trained detection net, then the fruit image passes into a ResNet-based regression relation between food image and volume estimation, with an average error of less than 15% for the various viewpoints. In 2020, Kalantar et al. [60] used RetinaNet deep convolutional neural network to estimate the number of melons and the weight of each melon using colored images acquired by a digital camera mounted on an unmanned aerial vehicle. This system included three phases: melon detection, geometric feature extraction, and individual melon yield estimation (Fig. 2.3).

In Chapter 4 an original strawberry weight estimation method is developed, exploiting both state-of-the-art machine learning and deep learning techniques.

2.3. DL and LfD for trajectory planning

Learning from demonstration (LfD) is a method exploited for training the robot to perform a certain task (in this case to reach a certain ripe target berry to be picked) with several demonstrations performed by an expert. The expert can be a human but this is not mandatory [18].

The goal is to derive from demonstrations a policy that maps from state to action, thus it is important to understand what to learn and how to learn. This approach makes teaching a new task easier and the policy learned can apply to never-seen-before environments. Thanks to these advantages, the adoption of LfD increased during the years (Fig. 2.4).

The demonstration can be performed in different ways: kinesthetic [34] in which the operator can move directly the robot by hand, teleoperation [9] where it is involved an external input to the robot, or observations [146] in which the task is shown directly with the body of the operator (Fig. 2.5).

LfD methods can build task models for trajectory planning/control with no extensive handcrafted programming. **Probabilistic LfD approaches** learn the distribution of demonstrated behaviour (e.g. Gaussian Mixture models (GMM) [43] or Gaussian Process [120]) and express them by a mean trajectory and its covariance. However, these

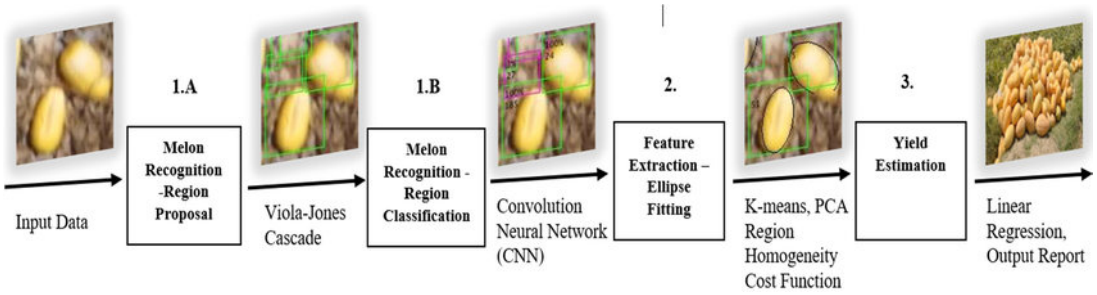


Figure 2.3: Algorithm pipeline for automated yield tracking by [60].

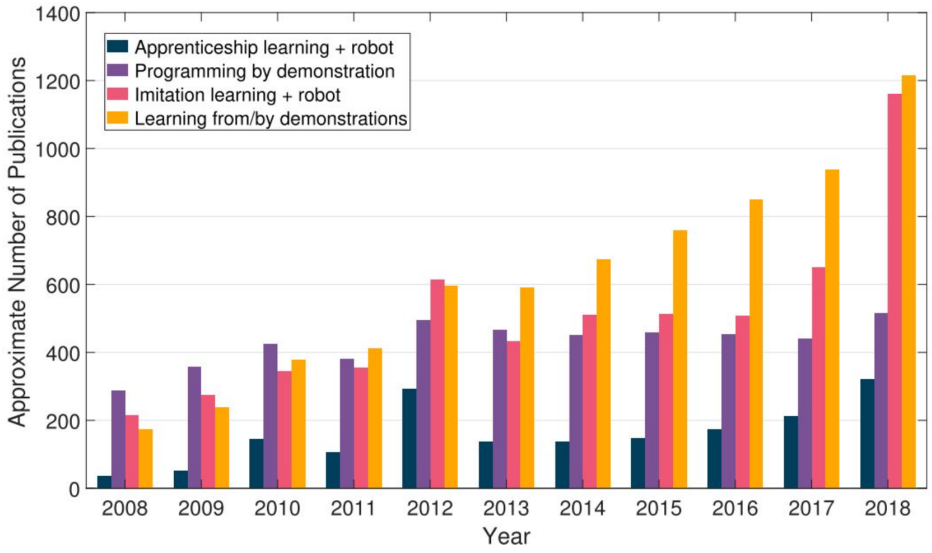


Figure 2.4: Consistent growth in the number of publications concerning LfD over the past decade, as reflected by the trend in the number of search results on Google Scholar that contain key phrases related to LfD [107].

Figure 2.5: Examples of the three categories of robot demonstrations [107].

models only encode the observed trajectory and need some input to be provided by human or CV module [106] for generalising to new scenes. In other words, they do not represent the mapping between some perceptive sensory information and the desired trajectory to perform the task.

End-to-end learning-based control approaches have been adopted to map pixel and depth images to the torque of quadruped to jump [81, 128]. Inverse Reinforcement Learning [71], vision-based Model Predictive Control [37], or Behaviour cloning [104] map directly visual information into robot actions; but they usually suffer from lack of generalisation in complex tasks. Moreover, they are only applied to a class of tasks that involve complicated motion control but not complex motion planning. End-to-end LfD methods enable generating control commands directly from raw image data for a task that requires complex motion control and simple motion planning, e.g. rolling a marble 1-2 centimeters on a table [127]. However, formalising the problem in this way for complicated tasks, e.g. fruit picking, makes it intractable. In this case it is hard to map a high-dimensional observation space into robot actions. Making combined motion/path planning and motion control using only raw images is intractable in such complex settings [11, 85].

Deep-time series (the prediction of a sequence of data, e.g. a trajectory, using a deep model) have been applied to learn the control policy of a robot to perform a specific task. For example, [72, 73] trained deep Neural Networks (NNs) in combination with Recurrent Neural Networks (RNNs) [51] with image data for learning a robots' control policy which may be effective for a limited number of (complex control) tasks.

Dynamic Movement Primitives (DMP) is a well-known LfD approach [119] able to encode the desired motion to be learned with a certain set of parameters (weights). Recently, deep learning has been applied to generate the DMP model parameters directly from an image of the environment in which the movement has to be executed [99, 110]. Ridge et al. [110] proposed a NN-based model that was trained to produce the DMP parameters [119]. Also, Pervez et al. [99] trained their deep NNs to enforce the DMP

model for visual servoing. Deep-MPs [115] utilise deep Neural Networks (NNs) for feature extraction and subsequent generation of the DMP weights defining a trajectory.

All the aforementioned models consider the prediction of a deterministic trajectory, but if there is some variability in the execution of a certain task (as can be the case of strawberry picking, where the fruit can be approached in multiple ways and with multiple orientations), it can be captured with the **Probabilistic Movement Primitives (ProMPs)** framework, able to represent the distribution of a set of demonstrations [94, 95]. However, ProMP lacks the relation between visual information and trajectory variations, hence, it lacks the points for generalisation [113]. In contrast to d-DMP [99, 110] and Deep-MPs [115], the approach that will be proposed in Ch. 5, called **Deep Probabilistic Movement Primitives (Deep-ProMP)**, maps the visual sensory information into a distribution of robot trajectories instead of a deterministic movement, filling the gap discussed above.

Deep-ProMP can be designed exploiting the architecture of **Autoencoder (AE)** [55], **Variational Autoencoder (VAE)** [13] or **conditional Variational Autoencoder (cVAE)** [61], as it will be seen in Ch. 5. The Encoder-Decoder architecture (common at AE, VAE and cVAE) is well-known in deep learning: it first compresses the information, e.g. image, in spatial resolution and then gradually decompress it [15]. In AE, VAE and cVAE architectures, the output of the decoder is the same as the input of the encoder. This compresses/embeds the feature representation into a lower-dimensional space (latent feature). This can be used for different tasks such as dimensionality reduction [130] and information retrieval [16]. AEs yield optimal reconstruction error but lack structure and interpretability in the latent space, i.e. it is not capable of generating new content. VAE [13] comes from the regularisation of AEs during training, expressing the latent space as probabilistic to avoid over-fitting. cVAE is introduced to learn the class-wise distribution of the data by inputting a condition (e.g. class label) to the latent space [61]. To improve the quality of latent space of VAE and cVAE regularisation is utilised [61].

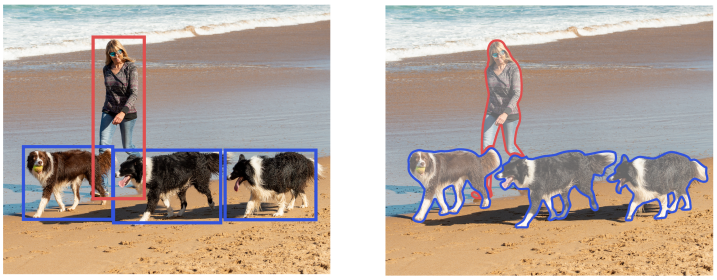
AE, VAE and cVAE architectures will be exploited in Ch. 5 to map the visual information of the environment in which the robot has to perform a task to the desired trajectory. The Encoder maps the input image to the latent space representation, and then, the latent space is mapped into the trajectory to be learnt with another trained deep model. A step more has been done with **Generative Adversarial Networks (GANs)**: VAE-GANs and cVAE-GANs additionally encode the ground truth trajectory back into the latent space [147]. In cLR-GAN [147], a random number (sampled from a known distribution) is input to the latent space that maps into the output and the latent vector is reconstructed from the output. Regularisation also helps to reconstruct the latent space from the output

in VAEs. For instance, Zu et al. [147] presented a BiCycle GAN model based on the same concepts in cVAE-GANs and cLR-GANs to improve the latent space. Kosaraju et al. [64] maps both (1) the latent noise to an output trajectory and (2) that trajectory back to the original latent space.

While GANs generate new content, the proposed Deep-ProMP aims to generate a probabilistic model of the demonstrated trajectories in the space of the observed dataset. This is different from content generation, hence, **GANs have not be implemented**. But, being inspired by [64] –that maps the output trajectories back to latent space to improve the latent space representation– regularisation and **domain-specific training** have been implemented to improve latent space representation (Sec. 5.4). This enforces the latent space to learn domain-specific information (i.e. on the reach-to-pick task for the hand of the robotic manipulator) to improve the proposed Deep-ProMP performance for path planning.

3 | Strawberry segmentation, key-points detection and classification

One key enabling technology towards automated harvesting is **accurate and robust strawberry detection**, which poses great challenges for the complex greenhouse environment that involves varying lighting conditions and foliage/branch occlusions. The first skill that an autonomous fruit-harvesting robotic system should acquire is the perceptive ability to detect the ready-to-be-picked fruits regardless of the complex and clustered configurations of the fruit plants. Greenhouse environments can be very challenging for computer vision problems, like fruit detection, since high variations in the colors or brightness of the fruits can be encountered over different plants of the same crop, overtime for the same plant, or over images of the same plant from different camera positions [10]. This chapter deals with the development of a deep learning (DL) based image analysis and computer vision method for the detection of berries able to handle these aspects. DL is a machine learning method, based on artificial Neural Networks (NNs), that uses multiple layers between the input and the output to extract higher-level features from the input data [39]. Object detection is the most informative instance of DL for the detection of fruits but it requires a lot of training data. This has led to the creation of **two novel datasets** useful for strawberry detection and ripeness classification through convolutional neural networks (CNNs) (Sec. 3.3). A CNN is a DL architecture that is most commonly used in computer vision to analyze visual data. While CNNs outperform humans in many cases, the complexity of the environments, such as multiple overlapping objects and different backgrounds, can pose several challenges. To this end, the current state-of-the-art object detection and instance segmentation method, Mask Region Convolutional Neural Network (Mask-RCNN) [54] has been chosen as a detection technique. The two proposed datasets have been used to **train a Mask-RCNN based model** (Sec. 3.2) with the objective of detecting individual berries, obtaining pixel-wise masks for each fruit, and classifying them accordingly to their ripeness level.



(a) Object detection. (b) Image segmentation.

Figure 3.1

3.1. Object detection and semantic segmentation

Before detailing the proposed strawberries detection system, let us first consider what the object detection and instance segmentation problems are in the framework of Computer Vision (CV).

Object detection consists of two separate tasks: classification and localization. The objective is to locate the presence of objects with a bounding box and types or classes of the located objects in an image (Fig. 3.1a).

- Input: An image with one or more objects.
- Output: One or more bounding boxes (e.g. defined by a point, width, and height) and a class label for each bounding box.

On the other hand, in digital image processing and computer vision, **image semantic segmentation** is the process of partitioning a digital image into multiple image segments (sets of pixels), also known as image regions or image objects. Image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics. Each of the pixels in a region is similar with respect to some characteristic or computed property [89], such as color, intensity, or texture (Fig. 3.1b).

3.1.1. Overview of fruit detection and localisation existing approaches

Different approaches exist for the problem of object detection and instance segmentation in the framework of fruit localisation. Among them, classic computer vision (CV)-based operations (including morphology, colour-based thresholding, and geometrical approaches) are widely used as a standard way to deal with the fruit perception problem and, in general, provide good performances.

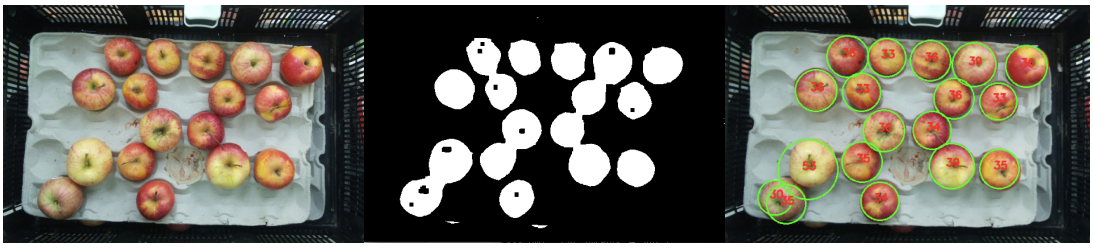


Figure 3.2: Color tresholding for fruit detection.

For example, the basic idea of **color thresholding** [22, 23, 47] is to automatically select an optimal gray-level threshold value for separating objects of interest in an image from the background based on their gray-level distribution (Fig. 3.2). The major problem with thresholding is that only the intensity is considered and not any relationships between the pixels. There is no guarantee that the pixels identified by the thresholding process are contiguous. Extraneous pixels that are not part of the desired region can be easily included, while isolated pixels within the region (especially near the boundaries of the region) can be easily excluded. These effects get worse as the noise gets higher, simply because it is more likely that a pixel's intensity does not represent the normal intensity in the region.

Morphology and geometrical approaches [25], instead, rely on the shape and geometry information of objects to detect them in an image. The problem of these approaches is the high variance among the possible shapes of fruit and so they lack the generalisation across different strawberries. Moreover, these algorithms often make several assumptions about the orientation of the fruit or the positioning of the peduncle or stem. However, strawberries and other fruits can grow with different positions and orientations and can be of any shape. This aspect limits the success of the approach presented. So, the inability to generalise and being prone to noise are among the weaknesses of these traditional approaches. Moreover, CV engineers need to handcraft features and, as the variation in data increases, it becomes a cumbersome and infeasible task [93].

Other state-of-the-art approaches in CV and DL have been able to yield improved performance. **Deep Learning** is a subfield of machine learning (computer systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyse and draw inferences from patterns in data) concerned with algorithms called artificial Neural Networks (NNs) inspired by the structure and function of the brain. NNs are comprised of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the

network. Otherwise, no data is passed along to the next layer of the network. In recent years DL has been demonstrated to be superior for tasks such as segmentation [54]. Thus, authors in selective harvesting have begun to adopt some of the DL techniques such as modern Convolutional Neural Networks (CNNs) for fruit perception. CNNs are specialized neural networks for processing data with an input shape like a 2D matrix (as images). But while CNN models perform well in image-specific tasks such as classification, for the pixel-wise understanding of images (semantic segmentation) Regional CNNs (RCNNs) are more successful [42]. RCNNs are widely described in section 3.2.1.

Another important need for autonomous fruit harvesting is the **localisation of the picking point**. Researchers have used information extracted from RCNNs with their own algorithm to improve the estimation of picking points [42, 79]. Yu et al. [140] first used a Mask-RCNN (MRCNN) to segment strawberry images and then used geometrical calculations to localise the picking point. The authors used two extreme points of a strawberry pixel to draw a horizontal line on the image frame. Then, the fruit axis was calculated based on the similarity of regions that would be divided by the fruit axis. The picking point was localised on the fruit axis based on statistics of strawberry shape. It is argued that, due to the widely varying shape of strawberries, such approaches for picking point localisation may fail in some cases and lacks the generalisation across different strawberries. Varying orientations of strawberry (upside-down, parallel to the gravity vector, or with an angle smaller than 90 degrees with respect to the gravity vector), occlusion, and other factors may limit the success of this presented approach.

3.1.2. DL-based key-points detection approaches

All the aforementioned aspects and possible failures for a precise strawberry detection and picking point localisation method led to consider **DL-based key-points detection approaches**, which has been very successfully applied in other domains, e.g. face landmark detection [144] or human-pose estimation [21] (Fig. 3.3a). These methods can be successfully applied to determine strawberry picking points, grasping points, and orientation (Fig. 3.3b). The idea of key-points detection is to detect interest points or key locations in an image, which are spatial locations or points that define what is interesting or what stands out in the image. They are invariant to image rotation, shrinkage, translation, distortion, and so on. However, this approach requires a **strawberry dataset with key-points annotations**.

(a) Key-points detection for human pose estimation.

(b) Key-points detection for strawberry perception .

Figure 3.3

As it will be shown in Table 3.1, the existing publicly available datasets do not present strawberry key-points or picking points, but only instance segmentation and detection annotations. So, two new datasets have been created containing key-points and picking points information, which are two essential attributes for selective harvesting (Sec. 3.3). Moreover, one of the two datasets also includes labeling for the weight of strawberries on the plant and it will be used for the development of a strawberry weight estimation algorithm in Ch. 4.

3.2. Proposed approach: Detectron-2 based model

The DL-based key-points estimation has never been explored for strawberry picking point localization. It has been demonstrated through the experimental results (Sec. 3.4.1) that the key-points regression approach based on MRCNN [54] works well for localizing the picking points of strawberries. Apart from localization of cutting points, the key-points are also pivotal in determining the orientation of strawberries which will be also used by the robot to approach the ripe strawberry. Similar to human pose estimation, the key-points are marked as visible or invisible. Thus, if the peduncle of a strawberry is occluded or facing away from the camera perspective, the MRCNN [54] algorithm can be trained to estimate the point as invisible. The proposed approach includes Detectron-2 [133] for segmentation and key-points estimation.

Detectron-2 [133] is a next-generation open-source object detection system from Facebook AI Research. The Detectron-2 model is based on MRCNN [54] and has become the de facto standard for instance segmentation. MRCNN [54], in turn, has been developed on top of RCNNs [45], fast-RCNNs [44], and faster-RCNNs [109].

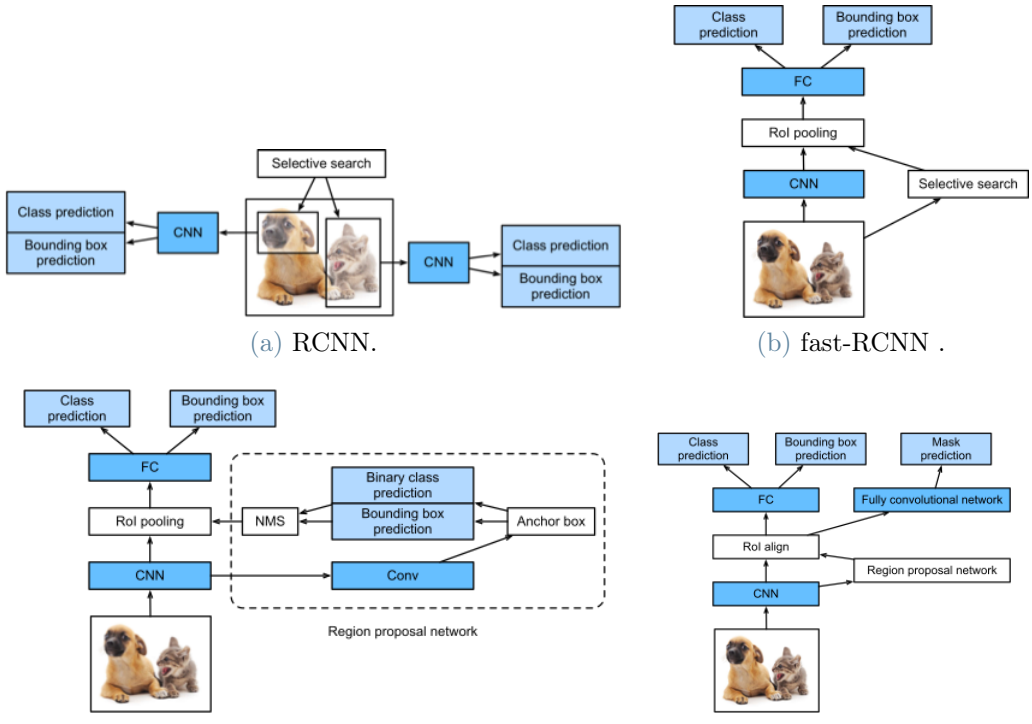
In the following sub-sections, the architecture of the aforementioned models is detailed.

3.2.1. RCNNs, fast-RCNNs, faster-RCNNs and MRCNN

RCNNs (Fig. 3.4a) are a family of deep learning models specially designed for object detection and developed by Girshick et al., from UC Berkeley in 2014 [45]. An input image given to the RCNN model goes through a mechanism called *selective search* to extract information about the multiple *regions of interest*. A *region of interest* is a rectangular portion of the input image and can be represented by its boundaries. Every *region of interest* goes through pre-trained CNNs to produce output features. These output features then go through multiple SVM (Support Vector Machine [24]) classifiers to classify the objects presented under a *region of interest*. In particular, the outputs of every single SVM are the bounding box coordinates and the class of the detected object in the proposed region.

Although the RCNN model uses pre-trained CNNs to effectively extract image features, it is slow. Imagining that thousands of region proposals are selected from a single input image, this requires thousands of CNN forward propagations to perform object detection.

This massive computing load makes it infeasible to widely use RCNNs in real-world ap-



(a) RCNN. (b) fast-RCNN .

(c) faster-RCNN . (d) Mask-RCNN .

Figure 3.4

plications. The main performance bottleneck of an RCNN lies in the independent CNN forward propagation for each region proposal, without sharing computation. Since these regions usually overlap, the independent feature extractions lead to much-repeated computation.

One of the major improvements of **fast-RCNN** [44] (Fig. 3.4b) from RCNN is that the CNN forward propagation is performed on the entire image, converting the whole image on the feature map, at once. Moreover, this CNN is trainable. The region proposals from the *selective search* mark *regions of interest* on the CNN output instead of the input image. Then, from these *regions of interest*, fast-RCNN further extracts features of the same shape to be easily concatenated. To achieve this, fast-RCNN introduces the *region of interest (RoI) pooling layer*: the inputs to this layer are the CNN output and the region proposals, while the outputs are the concatenated features further extracted from all the regions proposed on the CNN output.

So far no method for choosing *regions of interest* has been considered. This is the basic difference between fast-RCNN and **faster-RCNN** [109] (Fig. 3.4c). Faster-RCNN uses a region proposal method to create the sets of regions. Faster-RCNN possesses an extra CNN for gaining the regional proposals, called the *regional proposal network*. The rest of the model remains unchanged from fast-RCNN. It is worth noting that, being part of faster-RCNN, the region proposal network is jointly trained with the rest of the model. In other words, the objective function of faster-RCNN includes not only the class and bounding box prediction for object detection but also the binary class and bounding box prediction of anchor boxes in the region proposal network (Fig. 3.4c). As a result of the end-to-end training, the *region proposal network* learns how to generate high-quality region proposals and stay accurate in object detection with a reduced number of region proposals that are learned from data.

Mask-RCNN (MRCNN) [54] (Fig. 3.4d) was developed on top of faster-RCNN to perform image segmentation. While faster-RCNN has two outputs for each candidate object, a class label, and a bounding-box offset, MRCNN is the addition of a third branch that outputs the object mask from an additional fully convolutional network. The additional mask output is distinct from the class and box outputs, requiring the extraction of a much finer spatial layout of an object. In particular, the outputs of MRCNN are (Fig. 3.5):

- *Bounding Box*: described by the vertices of the rectangle around each detected object;
- *Class Label*: Class label assigned to each detected object;

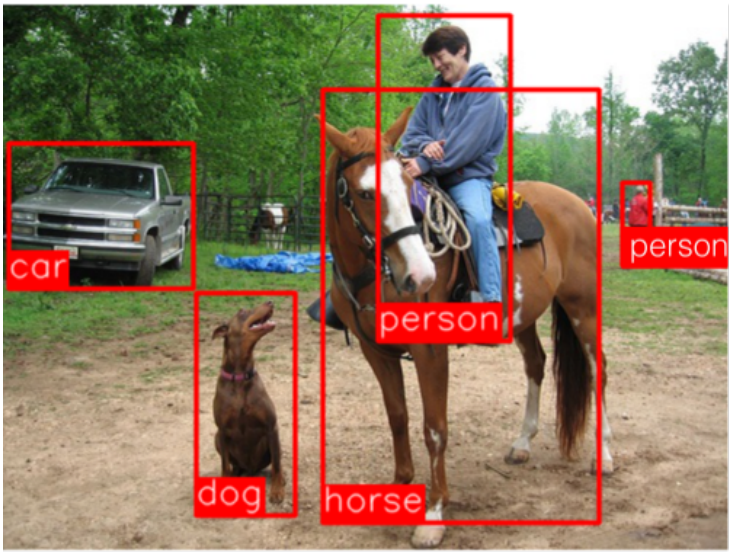


Figure 3.5: Output of Mask-RCNN.

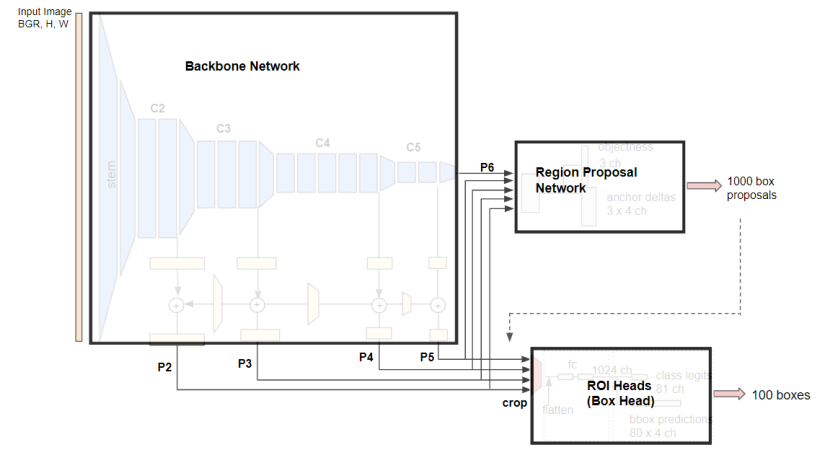
- *Prediction Confidence*: Confidence of class label prediction for each detected object;
- *Object Mask Outline*: Pixel values of the polygon outline for the mask of each detected object;
- *Object Mask*: pixels filling the the polygon for the mask of each detected object.

3.2.2. Proposed model based on Detectron-2

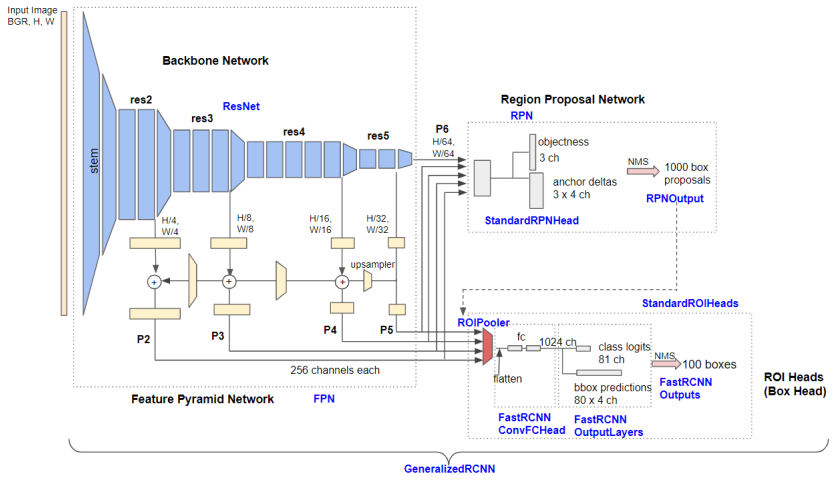
Detectron-2 [133] is Facebook AI Research (FAIR)'s next-generation library, based on MRCNN, that provides state-of-the-art detection and segmentation algorithms. Various state-of-the-art models for detection tasks such as bounding-box detection, semantic segmentation, and person key-point detection can be trained using this library, and this is the reason why it has been chosen for the purpose to estimate the strawberry key-points.

The schematic in Fig. 3.6 shows the meta-architecture of the Detectron-2 network, and its main components, namely:

- **Backbone Network or Feature Pyramid Network (FPN)**: it extracts feature maps from the input image at different scales.
- **Region Proposal Network**: it detects object regions (1000 box proposals by default) from the multi-scale features.
- **Box Head**: it crops and warps feature maps using proposal boxes into multiple fixed-size features, and obtains fine-tuned box locations and classification results via fully-connected layers.



(a)

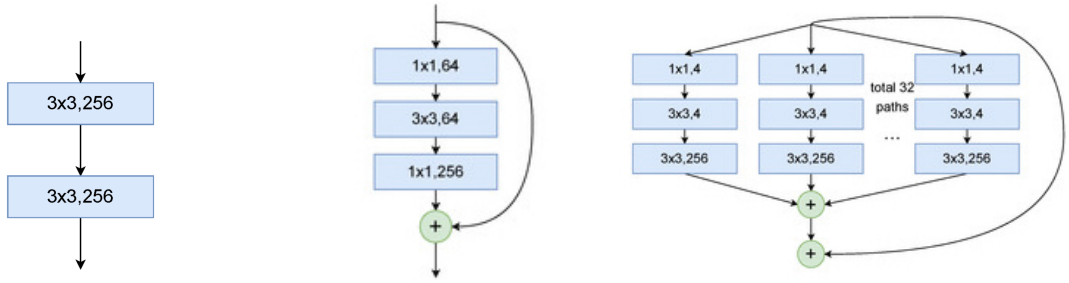


(b)

Figure 3.6: Detectron-2 meta architecture.

Experiments have been performed with three backbone networks for Detectron-2, R50-FPN, X101, and X101-FPN. These are all **Residual Neural Networks (ResNets)** [51]. Their peculiarity is that they use skip connections, or shortcuts to jump over some layers of the deep model (Fig. 3.7b). There are two main reasons to add skip connections among the hidden layers of a NN: to avoid the problem of vanishing gradients (when the partial derivative of the loss function with respect to the current NN weights becomes vanishingly small, preventing the weights from changing their values during training), or to mitigate the accuracy saturation problem (where increasing the depth of a network leads to a decrease in performance on both test and training data).

ResNeXt [134] (X101-FPN and X101) is more recent network architecture which was introduced as an improvement of **ResNet-50** [50] (R50-FPN), a convolutional residual neural network that is 50 layers deep. In particular, ResNeXt includes shortcuts from the



(a) CNN block (b) ResNet block (c) ResNeXt block

Figure 3.7: Three different network building blocks

previous block to next block, stacking layers and adapting split-transform-merge strategy (Fig. 3.7c). Section 3.4.1 discusses the results obtained with the different backbones in detail.

To train the Detectron-2 based models to estimate the bounding box around the detected berry together with the segmentation mask, the key-points, and the ripeness category, a dataset of images and relative annotations is needed. Since current publicly available strawberry datasets, as **Strawberry Digital Images (SDI)** [101] and **Dryad** [32] are not suitable for picking point localization and determination of the orientation through key-points, since they are not furnished with these annotations (see Table 3.1), two novel datasets (**Dataset-1** and **Dataset-2**) have been created and proposed with the annotations of the aforementioned features.

Strawberry datasets details

	SDI	Dryad	Dataset-1	Dataset-2
#Images	31200	35442	1588	3100
#Berries	17938	1611	2413	17938
Depth	✗	✓	✓	✗
Segmentation	✓	✗	✓	✓
Farm	✓	✗	✓	✓
Ripeness	✗	✗	✓	✓
Key-points	✗	✗	✓	✓
Weight	✗	✓	✓	✗
Dimensions	✗	✓	✓	✗

Table 3.1: Details of the already available and the new proposed strawberries datasets.

The datasets' key-points, segmentation masks and strawberry ripeness categories have been provided in MSCOCO JSON format [77] since this is the default format for feeding data into Detectron-2. A file in COCO JSON format is composed of five sections providing information for an entire dataset:

- *info* – general information about the dataset;
- *licenses* – license information for the images in the dataset;
- *images* – a list of images in the dataset;
- *annotations* – a list of annotations that are present in all images in the dataset;
- *categories* – a list of label categories.

3.2.3. Detectron-2 loss function for segmentation and key-points detection

The loss function to be minimized during training of Detectron-2-based models to get the bounding boxes and segmentation masks of the objects of interest, the classes of the detected objects, and the key-points location and type, is calculated as shown in Eq. 3.1.

$$L_{tot} = L_{cls} + L_{box} + L_{mask} + L_{kp} \tag{3.1}$$

L_{all} represents the total cost loss function of the Mask-RCNN model, L_{box} (Eq. 3.3) represents the regression loss of the predicted boxes, L_{cls} (Eq. 3.2) represents the classification loss of the predicted boxes, L_{mask} (Eq. 3.5) represents the loss on the predicted segmentation masks and L_{kp} represents the loss on the predicted key-points.

$$L_{cls}(p_i^k, p_i^{k*}) = -lb[p_i^k p_i^{k*} + (1 - p_i^k)(1 - p_i^{k*})] \tag{3.2}$$

In Eq. 3.2 the classification loss is shown, where p_i^k represents the predicted probability that pixel i belongs to a certain object class k, p_i^{k*} represents the corresponding real pixel for that class k label and lb represents the log loss function. k goes from 1 to K (total number of classes).

$$L_{box}(t^k, v) = \sum_{i \in x,y,w,h} [L_1^{smooth}(t_i^k - v_i)] \tag{3.3}$$

$$L_1^{smooth} = \begin{cases} 0.5x^2, & \text{if } |x| < 1, \\ |x| - 0.5, & \text{otherwise} \end{cases} \tag{3.4}$$

In Eq. 3.3 the bounding box regression loss is shown, where t^k and v represent the predicted bounding box for class k and ground truth bounding box, respectively. The term i represent bounding box coordinate offset (center coordinate (x, y), width w, and height h). These are the parameters to be correctly predicted and that describe a bounding box.

$$L_{mask} = -\frac{1}{m^2} \sum_{1 \leq i,j \leq m} [g_{ij} log(p_{ij}^k) + log(1 - p_{ij}^k)] \tag{3.5}$$

L_{mask} (Eq. 3.5) is computed as the average binary cross-entropy for the predicted masks associated with the ground truth. The mask head of the model has a dimensional output of Km^2 as it generates K (total number of classes) binary masks of size m x m (input image dimensions). g_{ij} is the class label probability of a pixel (i, j) in the ground truth mask, p_{ij} is the predicted label of the same pixel in the mask generated for class k.

L_{kp} has the same formulation of L_{mask} considering that the masks to be predicted are the one-hot binary mask representing the key-point locations (Fig. 3.8). So, for each key-point, during training, the target is a binary map of the image dimensions where only a single pixel is labeled as foreground.

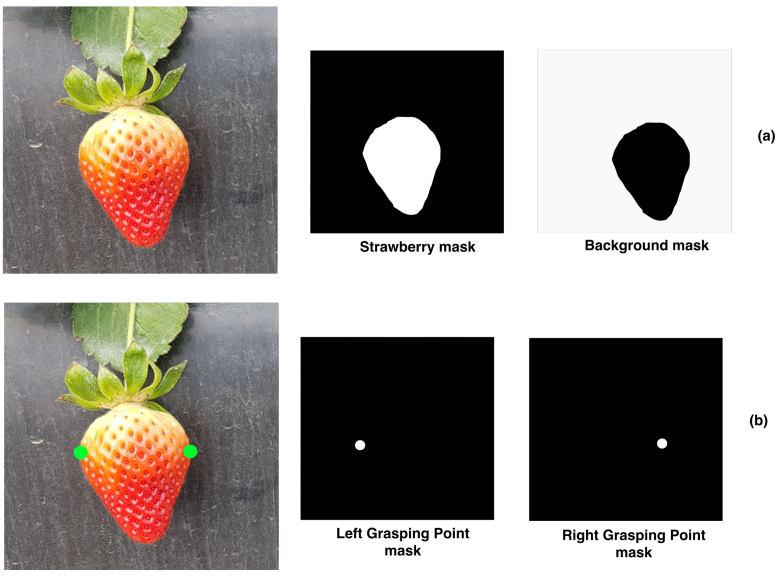


Figure 3.8: Segmentation masks (a) and key-points masks (b) to be predicted.

3.3. Datasets acquisition and annotation

The proposed datasets present the annotations of the bounding boxes, segmentation masks, key-points, ripeness category, and weight (Dataset-1) of each strawberry.

These novel features are not present in available datasets as SDI [101] or Dryad et al. [32] and are unique and very much needed for autonomous strawberry harvesting robots. In particular, the weight annotations have been exploited to develop a strawberry weight estimation algorithm (Ch. 4) useful for developing robotic systems that pick strawberries and place them directly into punnets according to their weights and sizes for delivering to supermarkets.

Dryad et al. [32] presents the weights of strawberries only under controlled laboratory conditions with the calyx separated from the strawberries. In contrast, the proposed Dataset-1 presents the **weights of strawberries under farm conditions** where it is difficult to obtain very accurate and consistent depth information, especially under sunlight. This is a more challenging and realistic perception scenario since the initial separation of berries into punnets happens directly after the picking action, according to the estimated strawberries' weight.

3.3.1. Dataset 1

Dataset-1 (Fig. 3.10a) has been collected at the new *15-acre* table-top strawberry glasshouse in Carrington, Lincolnshire, which is the latest addition to the state-of-the-art Dyson Farming's circular farming system [4]. Dataset-1 is a novel dataset that presents strawberries dimensions, weights, suitability for picking, instance segmentation, and key-points for grasping and picking. The main purpose of this dataset is to facilitate autonomous robotic strawberry picking action. For each strawberry, the dataset presents **five different key-points**: picking point (PP), top, bottom points of fruit, left grasping point (LGP), and the right grasping point (LGP) as shown in Fig.3.9. While the PP indicates the position on the stem where the cutting action has to be performed, the left and right grasping points can provide reference to the end effector for grasping action. Each key-point is then classified as "visible", if it is clearly visible in the image, "invisible-by-itself" if the key-point is not visible because covered by the berry itself due to its orientation with respect to the camera, or "invisible-by-other" if something different is hiding it. It is essential to accurately recalculate the bounding box once the key-points annotations are added. Without the key-points, the bounding box aligns to the extremities of the segmentation mask.

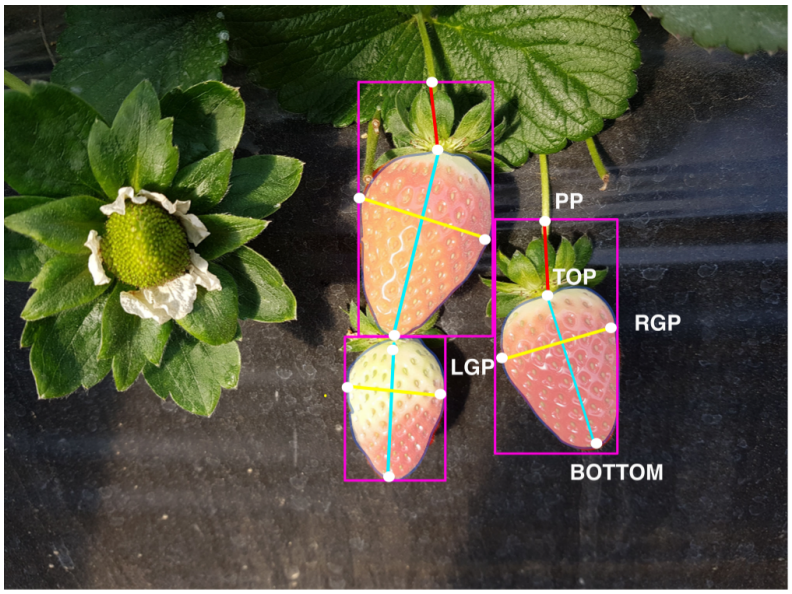


Figure 3.9: Strawberries 5 key-points.

However, the PP key-point lies outside the segmentation mask of the strawberries and thus outside the bounding box. Because of the nature of MRCNN, a key-point outside the bounding box is not detectable. Thus, the bounding boxes are expanded to accommodate all the key-points (see Fig. 3.9). In addition, the dataset also contains annotation for instance segmentation for each of the strawberries. To determine the suitability of strawberries for harvesting, each strawberry is labeled as **"pluckable"**–ready to be picked– or **"unpluckable"**–not to be picked–. "unpluckable" strawberries include unripe, semi-,

(a) Dataset-1.

(b) Datastet-2.

Figure 3.10: Images from Dataset-1 and Dataset-2

Strawberry datasets details

	#images	#"pluckable"	#"unpluckable"	#KPS	#Weight
Dataset-1	1588	1757	656	2413	1910
Dataset-2	3100	3659	14279	10999	NA

Table 3.2: Details of the new proposed strawberries datasets.

and over-ripe or rotten berries. The "pluckable" category includes strawberries that are nearly ripe and perfectly ripe.

The dataset contains 532 strawberry sets (Table 3.2). Each set has three color, depth, and point cloud data of the same strawberry cluster from three different distances. The farthest image captures the entire cluster whereas the nearest image focuses on one target strawberry in the cluster. In total, this dataset includes 1588 strawberries images (Table 3.2). All the images have been captured with Intel Realsense RGB-D sensor *D435i*.

3.3.2. Dataset 2

Dataset-2 (Fig. 3.10b) has been derived from the Strawberry Digital Images (SDI) [101] dataset and it is an enhancement of it. SDI dataset contains a total of 3100 images. These are dense strawberry clusters that contain an average of 5.8 strawberries per image. 10999 berries have been carefully annotated, each with the 5 different key-points.

Moreover, the "pluckability" (i.e. suitability to be picked) has been labeled for all the strawberries. The strawberries that are either severely occluded or are in an early flowering stage have not been annotated for key-points since a meaningful annotation was not possible.

3.4. Experimental results and discussion

3.4.1. Standard metrics for object detection and images segmentation

To evaluate the proposed models' performances for strawberry detection and segmentation the MSCOCO evaluation standard [8] has been adopted. Namely, the **Average Precision** (AP) is considered as it is the conventional metric to evaluate object detection performances. It is defined as the area under the precision-recall curve (PR curve). In equation 3.6 the definitions of **Precision** and **Recall** are shown, where ***tp*** is the number

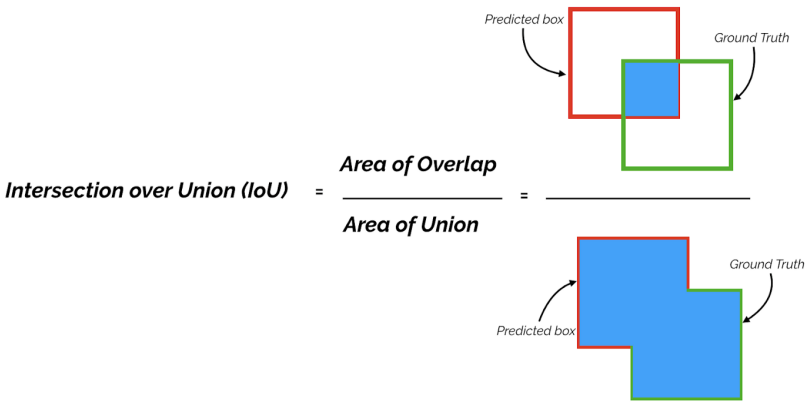


Figure 3.11: Intersection over Union score (IoU)

of true positives (number of outcomes that the model correctly predicts as positive class), ***fp*** is the number of false positives (number of outcomes that the model incorrectly predicts as positive class) and ***fn*** is the number of false negatives (number of outcomes that the model incorrectly predicts as negative class). Basically Precision is measuring the percentage of correct positive predictions among all predictions, and Recall is measuring the percentage of correct positive predictions among all real positive cases.

$$\begin{cases} \boldsymbol{Precision} = \boldsymbol{tp}/(\boldsymbol{tp} + \boldsymbol{fp}) & \text{(3.6a)} \\ \boldsymbol{Recall} = \boldsymbol{tp}/(\boldsymbol{tp} + \boldsymbol{fn}) & \text{(3.6b)} \end{cases}$$

In the case of object detection or instance segmentation, the **Intersection over Union (IoU)** ratio is used as a threshold for determining whether a predicted outcome is a true positive or a false positive. Given the predicted and ground-truth bounding boxes, the ratio between the area of overlap and the area of the union of the two bounding boxes is the IoU score (Fig. 3.11). IoU is a good way of measuring the amount of overlap between two bounding boxes or segmentation masks. If the prediction is perfect, IoU = 1, while if it completely misses, IoU = 0. Precision and Recall are calculated using the IoU value of the predicted and ground-truth bounding boxes, for a given IoU threshold. For example, if IoU threshold is 0.5, and the IoU value for a prediction is 0.7, then the prediction is classified as True Positive (***tp***). On the other hand, if IoU is 0.3, it is classified as False Positive (***fp***) as shown in Fig. 3.12. That also means that for a prediction, different binary true or false positives can be obtained, by changing the IoU threshold.

Average Precision (AP) is the precision average across all recall values between 0 and 1 at a certain IoU threshold. To have a more general view, the AP index has been computed with different IoU levels, namely 0.5, 0.7 and 0.9.

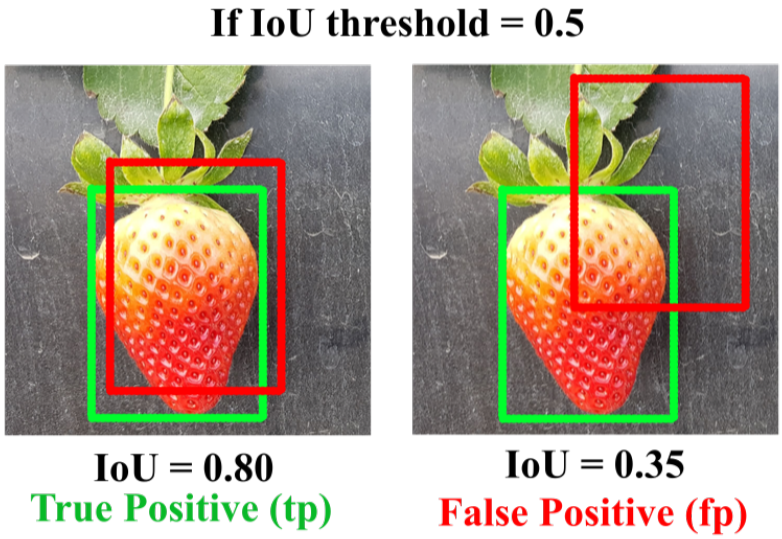


Figure 3.12: IoU for True Positive (tp) or False Positive (fp) classification.

To adopt AP for key-points detection, an analogous similarity measure has to be defined. This is done by defining the **Object Key-point Similarity (OKS)**, which plays the same role as the IoU. OKS [133] (Fig. 3.13) is the standard performance metric used by Detectron-2 [133] and MSCOCO [77] for key-point detection. It is calculated from the distance between predicted points and ground truth points normalized by the scale of the detected berry. Equation 3.7 shows the mathematical formulation of the OKS index, where d_i is the Euclidean distance between the detected key-point and the corresponding

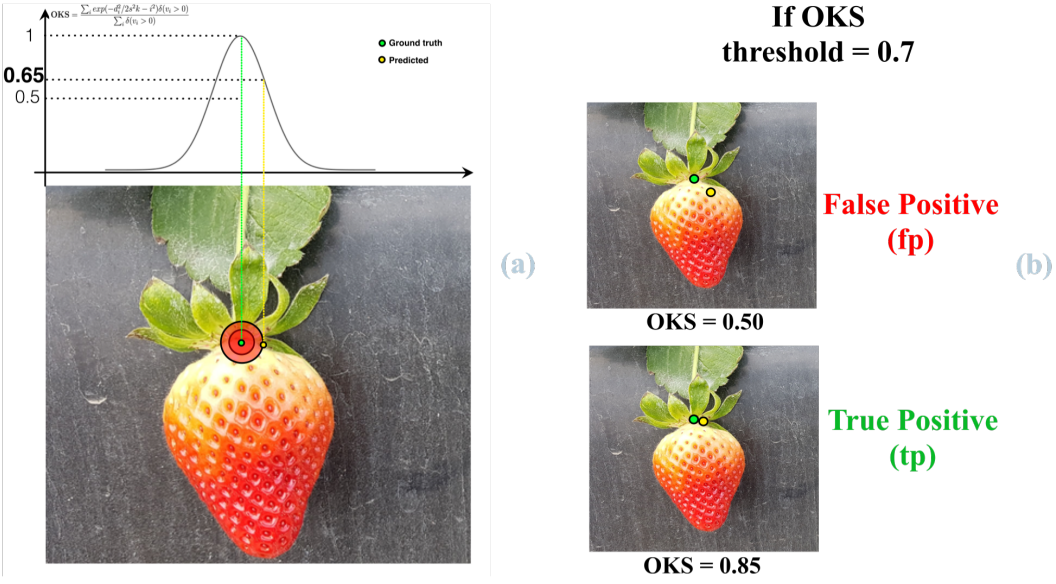


Figure 3.13: (a) Object Key-point Similarity (OKS). (b) OKS for True Positive (tp) or False Positive (fp) classification.

ground truth, v_i is the visibility flag of the ground truth point (annotated as visible or invisible), s is the strawberry scale, and k_i is the per-key-point constant that controls falloff. Perfect predictions will have OKS = 1 and predictions for which all key-points are off by more than a few standard deviations will have OKS = 0.

$$\textbf{OKS} = \frac{\sum_i exp(-d_i^2/2s^2k - i^2)\delta(v_i > 0)}{\sum_i \delta(v_i > 0)} \tag{3.7}$$

Also in this case the AP index has been computed with different OKS levels, namely 0.1, 0.3 and 0.5.

3.4.2. Implementation details

Detectron-2 [133] has been trained in Pytorch 1.8 [96] using separately first Dataset-1 and then Dataset-2 to have a comparison between the performances of the two datasets. For Dataset-1, the Detectron-2 model is trained for 2200 iterations, and for the larger Dataset-2, the model is trained for 20000 iterations. The Adam optimizer has been adopted with a learning rate of 0.0001.

3.4.3. Discussion of results

Table 3.3 summarises the results for segmentation and key-points detection of strawberries for both the datasets with Detectron-2 [133]. Although, the results demonstrate different backbones used in the experiments can produce consistent results across the dataset, **ResNeXt base model performs better than ResNet-50 base model** when used as backbone network for Detectron-2.

3.4.4. Segmentation results

The first two columns of Table 3.3 show segmentation Average Precision (AP) values for "pluckable" and "unpluckable" berries separately. The sub-columns show AP for Intersection over Union (IoU) thresholds of 0.5, 0.7, and 0.9. The standard practice is to consider IoU threshold 0.5 [54]; however, IoU thresholds up to 0.9 have been considered for completeness. Using Dataset-2, the proposed models yield decent AP values for both "pluckable" and "unpluckable" strawberries at IoU threshold 0.5. However, as shown in Table 3.3, the "unpluckable" berries in Dataset-2 significantly outnumber the "pluckable" berries. This results in better segmentation performance of "unpluckable" berries. The performance drops significantly for stricter IoU thresholds 0.7 and 0.9. This dataset

represents berries in very dense clusters and thus Dataset-2 is a very challenging dataset and has the potential to further advance the research in selective harvesting.

On the other hand, Dataset-1 shows very reliable AP values for "pluckable" strawberries for both the backbones across the different IoU thresholds. With IoU threshold of 0.5, Detectron-2 produces 93.32 (R50-FPN) and (X101-FPN) 94.19 AP values, while with a very strict IoU threshold of 0.9 it provides AP of 83.55 and 88.70 with R50-FPN and X101-FPN, respectively. This shows that for selective harvesting the dataset can be reliably used. For Dataset-1, the performance of the models on "unpluckable" berries is comparatively less reliable as there are fewer samples of "unpluckable" berries in this dataset. However, from a selective harvesting perspective, instance segmentation of "pluckable" berries is more essential.

3.4.5. Key-points detection results

The results of the key-points detection expressed in terms of AP at different OKS thresholds are similar to segmentation. At each OKS threshold, the AP with 0.5, 0.3, and 0.1 OKS has been taken. While the OKS threshold normally used is 0.5, 0.1 is a stricter threshold. The experimental results show that, similarly to segmentation, the results are consistent across the two backbones, although X101-FPN performs slightly better. Also, the key-points detection for "pluckable" berries is much better than "unpluckable"

Segmentation and key-points detection results

		Dataset-1		Dataset-2	
Backbone	IoU/OKS	R50-FPN	X101	X101-FPN	X101
Segmentation "Pluckable"	0.5	93.32	**94.19**	71.12	**72.12**
	0.7	90.97	**92.83**	64.70	**66.84**
	0.9	83.55	**88.70**	43.24	**47.86**
Segmentation "Unpluckable"	0.5	59.46	**61.12**	76.83	**78.09**
	0.7	53.61	**56.22**	74.52	**76.65**
	0.9	42.91	**45.64**	68.79	**70.30**
Key-Points "Pluckable"	0.1	91.27	**97.21**	**64.32**	59.29
	0.3	89.10	**91.40**	**58.93**	54.40
	0.5	81.90	**87.74**	39.92	**42.12**
Key-Points"Unpluckable"	0.1	51.36	**61.26**	73.26	**74.67**
	0.3	46.20	**56.52**	71.39	**71.45**
	0.5	37.30	**46.84**	**66.46**	65.30

Table 3.3: Details of the experimental results for strawberry segmentation and key-points detection.

berries for Dataset-1. The results for Dataset-2 obtained comparing X101-FPN and X101 networks, provide a good baseline for future research.

In Fig. 3.14 two segmented images are shown. In Fig. 3.14a an image from Dataset-1 is represented, while in Fig. 3.14b an image from Dataset-2 is shown. It can be seen that the berries are recognized, labeled as "pluckable" or "unpluckable" with a certain confidence, and the 5 key-points for picking and grasping action are predicted.

(a) Dataset-1. (b) Datastet-2.

Figure 3.14: Strawberry perception for images in Dataset-1 and Dataset-2

4 | Strawberry weight estimation

Normally, after the picking action of the ripe strawberries from the plant, the harvested fruits are directly placed into punnets (Fig. 4.1) that are required to be roughly of the same weight. Human harvesters rely on their experience and scales to estimate the strawberry weights. Clearly, this procedure can be imprecise, and subsequently, a lot of effort is required to re-sort the strawberries into punnets. This increases the production cost. Moreover, weighting strawberries after harvesting requires additional handling and maneuvering of the delicate fruits, which run the risk of being bruised. **AI-based weight estimation has the potential to reduce the production cost** by estimating the weight of the strawberries before detaching them from the plant [84, 142]. Robust and easy-to-use automatic machine vision-based systems can be used to increase the weighing efficiency of fruits and to decrease the overall costs of manual labour avoiding excessive fruit handling and the usage of various tools that require dedicated human efforts. [84].

Automatic image processing techniques are gaining popularity in the agriculture industry. The available image processing techniques for fruit weight estimation use mainly the intensity or color (RGB) values to identify and estimate weights.

Figure 4.1: Punnets of strawberries of fixed weight.

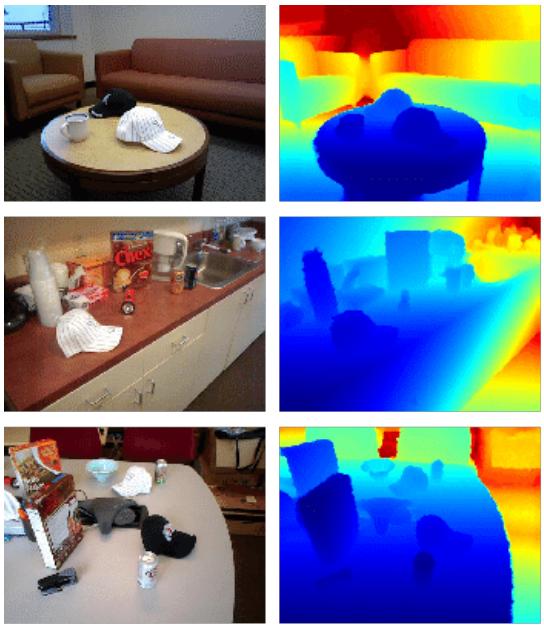


Figure 4.2: RGB-D image.

Recently, the availability of commercial depth sensors that are cheap and easy to use has opened the way to utilize the depth information along with intensity/color pixel values for automatic image processing methods [84]. An RGB-D image is simply a combination of an RGB image and its corresponding depth image (Fig. 4.2). A depth image is an image channel in which each pixel relates to a distance between the image plane and the corresponding object in the RGB image.

Various approaches for strawberry weight estimation have been developed and will be shown in the next sections. All the approaches belong to the category of deep (Sec. 4.2) or machine (Sec. 4.3) learning techniques, solving the problem of finding a mapping from the RGB-D information to the strawberry weight in different ways. In particular, the first approach (presented in Sec. 4.2.1) combines the RGB image and the raw depth image information using a neural network architecture inspired to the family of EfficientNet models [124]. The approach presented in Sec. 4.2.2 is a deep model which predicts the fruit weight starting from the reconstructed point cloud of the berry. In Sec. 4.2.3, a neural network able to handle a graph representation of the input data is presented. However, the machine learning-based approach explained in Sec. 4.3.1 turned out to be the most accurate, outperforming all the other deep model-based techniques.

The weight of strawberries is proportional to size, shape, and density. The instantaneous density of strawberries is also dependent on the sugar and water content of the strawberry which may not be feasible to be inferred from images alone. The higher the weight estimation accuracy, the better the performance of a robot picker in terms of meeting the weight and packaging legislation. Thus, estimating the weights of strawberries to a decent accuracy (80-90%) is the goal here so that they can be properly sorted into punnets while harvesting without the need for further manual processing [84].

4.1. Evaluation metrics

To measure the accuracy of weight estimation provided by the different implemented techniques, explained in the following sections, the **Percentage of Correct Weights (PCW)** protocol has been proposed as a metric. The goal is to have a unique way of measuring the weight estimation performances to compare the different approaches and to select the best or the most accurate one.

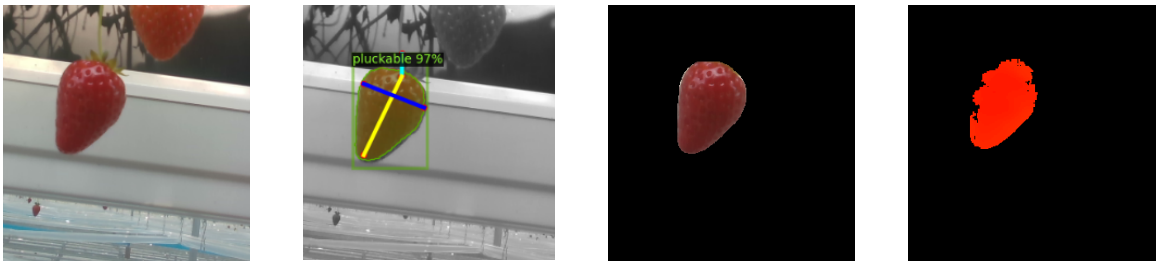
The regression error in percentage with respect to the ground truth is computed as shown in Eq. 4.1, where $\hat{x}$ is the predicted weight and x is the ground truth. If the error is within the desired tolerance, the inference is marked as accurate (score = 1, Eq. 4.1a); otherwise, it is marked as non-accurate (score = 0, Eq. 4.1b). The percentage of predictions within the tolerance values (marked as accurate) defines the accuracy of a model.

In the experiments, the tolerance levels are set at 0.05, 0.1, 0.15, 0.20, 0.25 and 0.30 which correspond to a range of 70-95% accuracy. The protocol is inspired by the Percentage of correct key-points (PCK) protocol used for human pose estimation [88].

$$\begin{cases} |\dfrac{\hat{x}-x}{x}| < tol \longrightarrow 1 & \text{(4.1a)} \\ |\dfrac{\hat{x}-x}{x}| > tol \longrightarrow 0 & \text{(4.1b)} \end{cases}$$

4.2. Deep learning-based approaches

The first set of techniques implemented for strawberry weight estimation includes several state-of-the-art neural networks. The vision-based weight estimation is dependent on the berries' apparent size in images and thus it is essential to use depth information together with the colour one. Dataset-1 (sec. 3.3.1) has been used to train the different models as it is composed of 1588 RGB-D images (captured with an Intel Real-Sense *D435i* camera) with 1910 strawberries, each annotated with the weight.



(a) Original RGB (b) Detection (c) Segmented RGB (d) Segmented depth

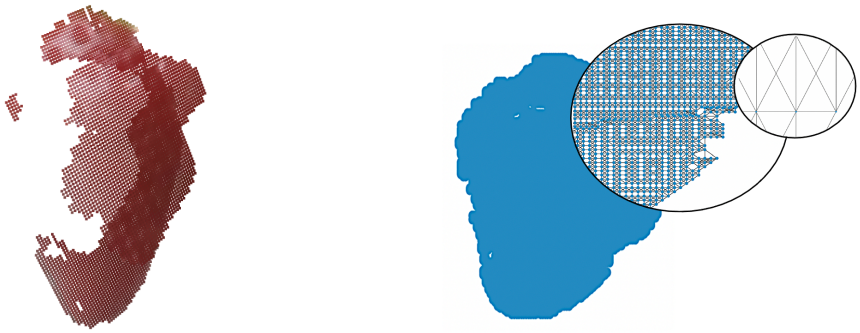
Figure 4.3: Strawberries instances extraction.

First, all the instances of strawberries from each RGB image (Fig. 4.3a) have been extracted with the help of segmentation through Detectron-2 [133] (Fig. 4.3b and 4.3c). The same segmentation mask is also applied to depth images (Fig. 4.3d).

These two segmented images (color and depth) have been used to train a model based on **EfficientNet** [124] (Sec. 4.2.1).

Moreover they have been combined with the help of the Realsense intrinsic camera calibration parameters to reconstruct the point cloud. The result is a segmented point cloud that only contains the strawberry points (Fig. 4.4a). The segmented point cloud has been used to train **PointConv** [132], **PointNet** [103] and **PointNet++** [102], which are well-known point cloud-based deep networks (Sec. 4.2.2).

Recently there has been an increased interest in graph-based neural networks since the introduction of graph neural networks by Kipf et al. [63]. Thus, also graph-based networks have been trained on the strawberry graphical representation (Fig. 4.4b), namely **Dynamic Graph CNN (DGCNN)** [131], **Graph Convolutional Network (GCN)** [63] and **Higher-order Graph Neural Network (HGNN)** [83].



(a) Strawberry segmented point cloud (b) Strawberry graph diagram

Figure 4.4: Strawberry segmented point cloud and relative graph.

4.2.1. Efficient-Net based model

The first proposed model capable to regress the weights of the berries is a Convolutional Neural Network (CNN). A **two-streams architecture** is used, where the first stream takes as input the segmented RGB image of a single strawberry, while the second stream is fed with the segmented depth image (Fig. 4.5). Thus, in both the inputs to the model, pixels belonging to the same strawberry retain their original value, whereas non-strawberry pixels are zeroed. The two branches are represented by a model taken from the **EfficientNet** family [124] with weights pre-trained on ImageNet. The final dense layer of the EfficientNet is removed and the outputs of both streams are concatenated. This concatenated layer is passed through a dense layer with linear activation to regress the berries' weights. EfficientNets [124] is a family of models obtained as improvement of standard Convolutional Neural Networks. Namely they originated from the observation that **balancing network depth, width, and resolution has been proven to lead to better performances**. Depth can be scaled up as well as scaled-down by adding-removing layers respectively; a deeper network can capture richer and more complex features, and generalizes well on new tasks. However, vanishing gradients is one of the most common problems that arise. In the second place, wider networks (with larger kernels/filters) tend to be able to capture more fine-grained features, but the accuracy saturates quickly with a larger width. Scaling resolution simply means that the resolution of the image passed to the CNN is scaled up or down. In high-resolution images, the features are more fine-grained, and hence high-resolution images should work better. Scaling up any dimension of a network (width, depth, or resolution) improves accuracy, but the accuracy gain diminishes for bigger models. M. Tan and Quoc V. proposed a simple yet very effective scaling technique that uses a compound coefficient ϕ to uniformly scale network width, depth, and resolution in a principled way as shown in Eq. 4.2.

$$\begin{cases} depth : d = \alpha^{\phi} \\ width : w = \beta^{\phi} \\ resolution : r = \gamma^{\phi} \\ \alpha\beta^2\gamma^2 \approx 2 \\ \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{cases}$$

In Eq. 4.2 ϕ is a user-specified coefficient that controls how many resources are available whereas α, β, and γ specify how to assign these resources to network depth, width, and resolution respectively. EfficientNets are networks obtained with the optimal values for those scaling parameters. For example EfficientNet-B0 has $\phi = 1$, $\alpha = 1.2$, $\beta =$

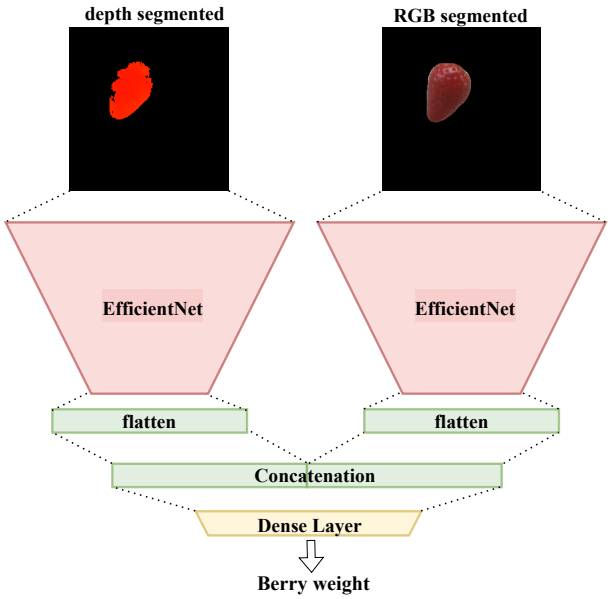


Figure 4.5: EfficientNet-based model architecture.

1.1 and $\gamma = 1.15$. EfficientNet-B7 achieves state-of-the-art 84.3% top-1 accuracy on ImageNet. Efficient-Net-B0 and B7 have been used as baselines for weight estimation. The ImageNet pre-trained EfficientNets have been implemented through Tensorflow 2.1 with Keras wrapper. In order to help the neural networks generalize better, data augmentation has been performed through random translations and rotations of the RGB and depth input images during training.

4.2.2. PointNet and PointNet++

Another baseline considered for strawberry weight estimation is represented by point cloud-based deep networks fed with the reconstructed segmented strawberry point cloud. Namely PointNet [103], PointNet++ [102] and PointConv [132] have been implemented. For each of these networks, the final layer in the network has been replaced with a dense layer containing only one unit with a linear activation function for regressing the strawberry weight.

PointNet [103] takes raw point cloud data as input. A point in a point cloud is fully described by its (x, y, z) coordinates together with other features that may be included, such as surface normal and intensity. The architecture of PointNet is surprisingly simple and quite intuitive. The network uses a shared multi-layer perceptron (MLP), i.e. a sequence of dense layers, to map each of the n points from three dimensions (x, y, z) to 64 dimensions. It is important to note that a single multi-layer perceptron is shared for

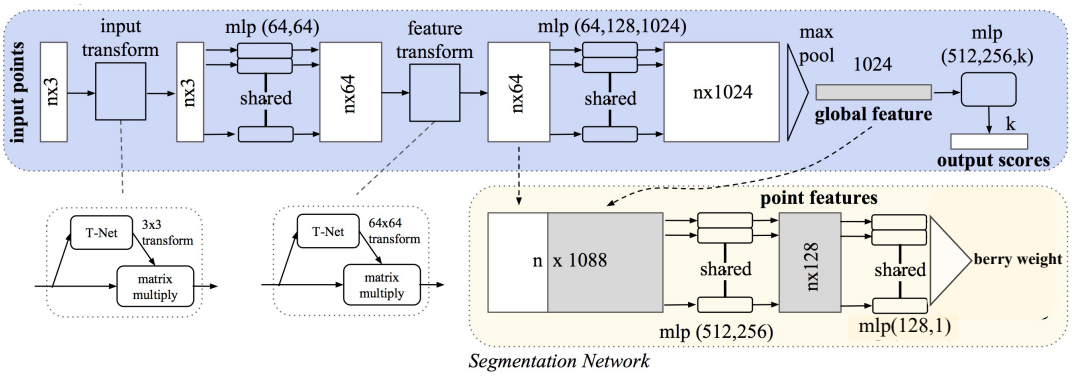


Figure 4.6: PointNet for weight estimation.

each of the n points (i.e., mapping is identical and independent on the n points). This procedure is repeated to map the n points from 64 dimensions to 1024 dimensions. With the points in higher-dimensional embedding space, max pooling is used to create a global feature vector. Finally, a three-layer fully-connected network is used to map the global feature vector to the regression output (weight of the berry). PointNet learns a spatial encoding of each point and then aggregates all individual point features to a global point cloud signature, so it does not capture local structures. Namely, it operates on each point independently and subsequently applies a symmetric function to accumulate features.

PointNet++ [102] is a hierarchical network that applies PointNet recursively on a nested portioning of the input point cloud. Similar to CNNs, PointNet++ extracts local features from a small neighborhood of points, and, further grouping them into larger units, produces higher-level features. This process is recursive until the feature of the whole point set is obtained. The ability to abstract local patterns along the hierarchy allows better generality to unseen cases with respect to PointNet.

PointConv [132] is the last model that is fed with a point cloud but it performs the regression operation differently from PointNet and PointNet++. While PointNet and PointNet++ consider each point in the point cloud as a set of coordinates to be processed through the units of the network, PointConv is based on an extension of the image convolution concept into the point cloud. PointConv trains multi-layer perceptrons on local point coordinate to approximate continuous weight and density functions in convolutional filters. This allows deep convolutional networks to be built directly on 3D point clouds.

Data augmentation has been performed also in this case, so the point cloud has been both translated and rotated randomly (the variations are within a region) before being used as input to the aforementioned models. A random translation has been applied in both dimensions perpendicular to the depth value that is kept fixed as the strawberry depth

location is a fundamental parameter for a correct weight regression. On the other hand, a random rotation has been applied around all three axes of the strawberry. After that, the coordinate values have been normalized and finally given as input to the different models for training.

4.2.3. Graph-based models

Experiments have also been carried out with graph-based networks, namely Graph Convolutional Network (GCN) [63], Higher-order Graph Neural Network (HGNN) [83] and Dynamic Graph CNN (DGCNN) [131].

Graph Neural Networks (GNNs) [116] are a class of deep learning methods designed to perform inference on data described by graphs. Graphs are simply one of the modalities which can be used to represent data. The difference is that in addition to the definition of the data points, the relationship that exists between them is also defined, and thus provides them with a structure. This means that graphs are a structured data representation. This additional information about the relation between data points can be very valuable, which is the reason why GNNs, in certain fields, have better performances compared to other solutions. A graph can be described as an entity composed of units (that are the data points we are working with), edges (that connect similar data points or express a relation between them), and features, that are additional information each node can carry on to better describe the node. The edges can be described with the Adjacency matrix, a binary square matrix which defines if there is a connection (with a 1) or not (with a 0) comparing each node with all the others and itself (Fig. 4.7). If instead of zeros and ones, float numbers are used to represent how powerful the connections are, the matrix is called the Weight matrix.

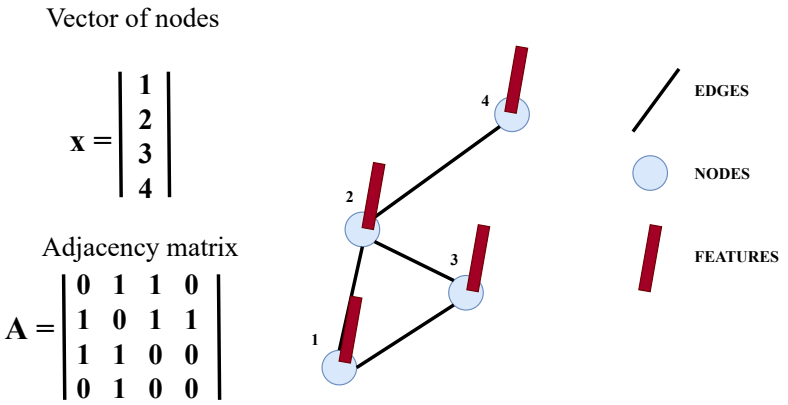


Figure 4.7: Graph and Adjacency matrix.

The graph dataset of strawberries is obtained by firstly applying surface mesh reconstruction to the segmented point clouds. Then, a fixed number of points have been sampled from the reconstructed surfaces. Finally, the k-nearest neighbor graph generation function has been exploited to get the final graphs, composed of nodes and edges (Fig. 4.4b). Each point in the point cloud represents a node. Given a point p, a directed edge links p to q whenever q is part of the k nearest neighbors of p, considering the Euclidean distance between p and q. The nodes' features are represented by the (x, y, z) coordinates of the point in the point cloud. In other words every node is represented by a point in the point cloud, while the edges are built on the basis of the Euclidean distance among the points. The obtained graphs are used as inputs for the different implemented GNNs, while the output is the strawberries' weight.

Graph Convolutional Network (GCN) [63] generalize classical Convolutional Neural Networks (CNN) to the case of graph-structured data. There are some challenges in applying convolution as it is applied in images or data sequences since in graphs both the number of neighboring nodes and the distance between nodes change. The concept of convolution can be extended to the graph framework considering that, on Euclidean domains, convolution is defined by taking the product of translated functions. In Eq. 4.3b, $\mathbf{x^{l+1}}$ is the representation of the input graph after the $l+1$ layer of the network. It is defined as the product of $\mathbf{x^l}$ for the Graph Convolution Filter ($\mathbf{G}$) plus a bias b passed trough a non linear function σ. The Graph Convolution Filter is defined as $\mathbf{G}$ (Eq. 4.3a) with w_k coefficients as weights pre-multiplied by S (adjacency matrix) to the power of the hyperparameter k. These w_k coefficients are the parameters that are updated and learned during training in order to minimize some loss function.

$$\begin{cases} \mathbf{G} = \sum_{k=0}^{K} w_k S^k & \text{(4.3a)} \\ \mathbf{x^{l+1}} = \sigma(\mathbf{G x^l} + b) & \text{(4.3b)} \end{cases}$$

In a nutshell, this is the equivalent formulation to perform a convolution step on graph data instead of 2D images.

The working principle of **Higher-order Graph Neural Network (HGNN)** [83] is the same as GCNs, but higher dimansional combinations of the input are utilized so that the input graph is combined in different ways before being fed to the GNN. The key insight in these higher-dimensional variants is that they perform message passing directly between subgraph structures, rather than individual nodes. This higher-order form of message passing can capture structural information that is not visible at the node-level [83].

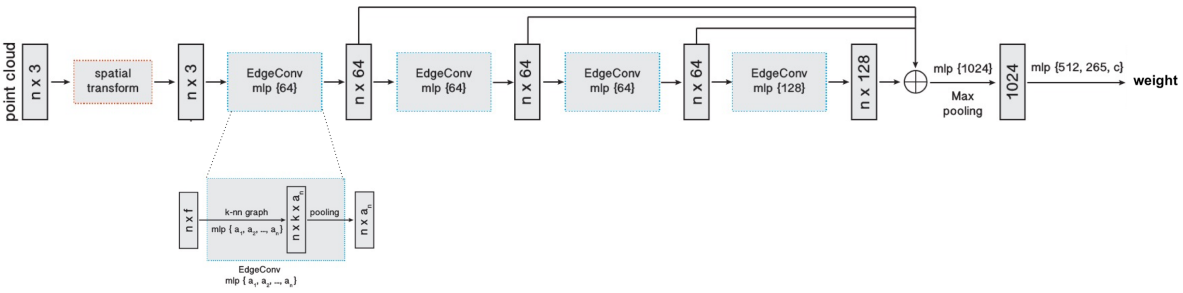


Figure 4.8: DGCNN model architecture for weight estimation. The EdgeConv block takes as input a tensor of shape n × f , computes edge features for each point by applying a multi-layer perceptron (mlp) with the number of layer neurons defined as $\{a_1, a_2, ..., a_n\}$, and generates a tensor of shape n × an after pooling among neighboring edge features.

Finally, **Dynamic Graph CNN (DGCNN)** [131] is a network that works slightly differently from the two previously described GNNs. It is fed with the original segmented point cloud, over which a dynamic graph representation is computed and updated at every layer of the network. This means that the computed graph, differently from the previous cases (in which the input graph was completely determined before training the models), is not fixed but rather is dynamically updated after each layer of the network. That is, the set of k-nearest neighbors of a point changes from layer to layer of the network and is computed from the sequence of embeddings. DGCNN is inspired by PointNet, but instead of working on individual points, it exploits local geometric structures by constructing a local neighborhood graph and applying convolution-like operations on the edges connecting neighboring pairs of points, in the spirit of graph neural networks, from which, on the other hand, it differs for the continuous update of the graph along the layers of the model (Fig. 4.8). It is able to do this using a neural network module called Edge-Conv suitable for CNN-based high-level tasks on point clouds. EdgeConv acts on the graphs dynamically computed in each layer of the network and, instead of nodes features, it generates edge features that describe the relationships between a point and its neighbors [131].

4.3. Machine learning-based approach

4.3.1. Random Forest Model with Decision Trees

Along with the above approaches, a novel, simple and effective baseline that outperforms the other state-of-the-art networks is presented. As shown in Figure 4.3, first the segmented images of individual strawberries are obtained through Detectron-2 [133]. The same segmentation mask is applied to the raw-depth image. As done for point cloud-based

deep networks these segmented images are combined with the help of Realsense intrinsic camera calibration parameters to create a segmented point cloud (Fig. 4.4a). The point cloud is not continuous and represents two depth layers (Fig. 4.9a). This is the case with original point clouds captured with Realsense. In realistic farm or outdoor scenes, the depth could range up to infinity. Thus, the resultant depth sensitivity (resolution) is low and unable to capture the relatively small depth gradient on the surface of a strawberry in a continuous manner. For this reason, the convex hull of the segmented point clouds (Fig. 4.9b) is reconstructed, which provides a more continuous surface. A principal component analysis (PCA) is performed on the convex hull to determine the direction of the largest extent (Fig. 4.9c). PCA is a technique used for identification of a smaller number of uncorrelated variables known as principal components from a larger set of data. The technique is widely used to emphasize variation and capture strong patterns in a data set. In this case it is used to find the 3 directions of largest extent of the point cloud of the strawberry.

A feature vector is created which contains the strawberry bounding box area obtained from the segmentation mask, the segmentation mask area, the histogram of depth values, and the primary principal components (Fig. 4.10). This feature vector is used to train a **Random Forest model with Decision Trees** [56]. Random forest is a flexible, easy-to-use machine learning algorithm that can produce, even without hyper-parameter tuning, great results both for classification and regression. It is a supervised learning algorithm. It builds multiple decision trees, so a combination of machine learning models, and merges them to get a more accurate and stable output, averaging the final predictions (the weights of the strawberries) from all the trees (Fig. 4.10). The trees are run in parallel and there is no interaction between them. The inner working of a Decision Tree can be thought of as a bunch of if-else binary conditions that leads to predicting the value of a target variable

(a) Discontinuous point cloud (b) Convex hull (c) PCA

Figure 4.9: Discontinuous point cloud, convex hull and PCA.

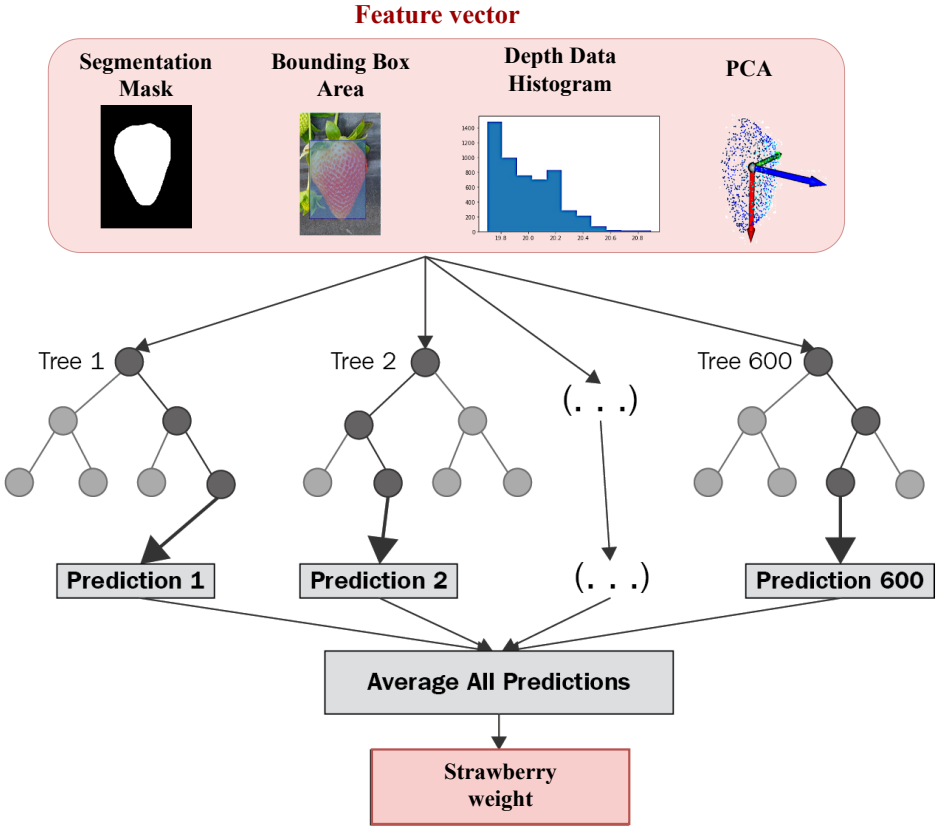


Figure 4.10: Random Forest model with Decision Trees to regress strawberries weight.

based on several input variables (the ones in the feature vector). It arrives at an estimate of the berry weight by asking a series of questions to the data, each question narrowing the possible values until the model gets confident enough to make a single prediction. The order of the question as well as their content is being determined by the model. In addition, the questions asked are all in a True/False form. During training, the model is fitted with the provided feature vectors and the true weight values. The model learns any relationships between the data and the target variable.

The histogram of the depth data of the strawberry with twelve bins has been considered to help the model learn about the extension of the fruit along the axis perpendicular to the camera. The bounding box area helps the model to learn the apparent size of strawberries. If a strawberry comes in different shapes and sizes, the area of the segmentation mask with respect to the bounding box helps the model to fine-tune the apparent size of the strawberry. This is straightforward if the orientation of the berry parallel to the camera view, but due to the different possible orientations of the strawberry, the bounding box and the apparent area of the strawberry can vary (Fig. 4.11). Thus, if the model is aware

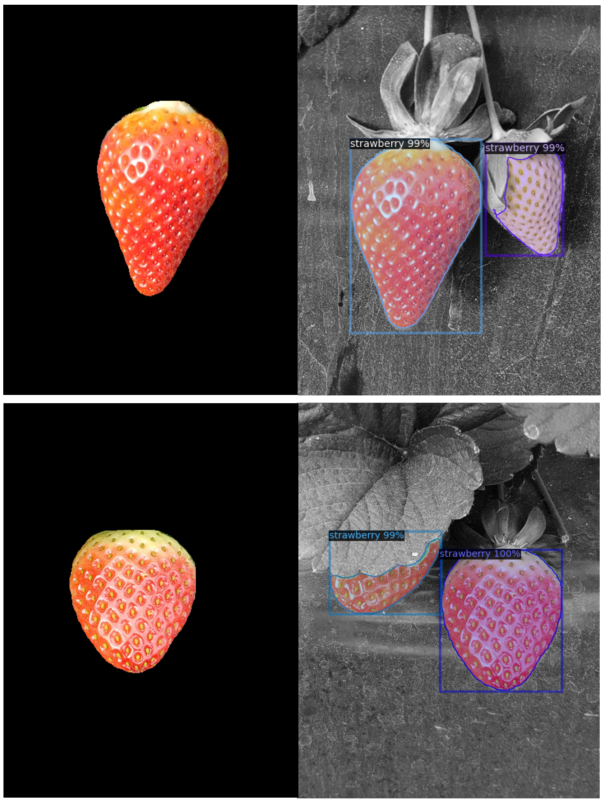


Figure 4.11: Due to the different possible orientations of the strawberry, the bounding box and the apparent area of the strawberry can vary.

of the 3-D orientation of the strawberry, it can learn the relationship of the weight and the variation of apparent size caused by the orientation of the fruit. And this is verified by the fact that **the inclusion of the primary principal components in the feature vector improves the performance of the Random Forest model with Decision Trees for weight estimation**. This last proposed model performs better than the deep models discussed earlier, as it can be seen from the graph in Fig. 4.12 and as discussed in Sec. 4.3.3.

4.3.2. Implementation details

The models and algorithms have been implemented in Python 3.7. The point cloud and graph-based neural networks have been implemented in Pytorch 1.8 [96]. For graph networks, Pytorch Geometric [36] version 1.7 has been used.

All models have been trained with a learning rate of 0.001 with Adam optimizer. For the Random Forest model with Decision Trees and to compute the PCA components of the strawberries point clouds, Scikit Learn [97] version 0.24.2 has been used.

Open3D [145] has been used for manipulating point clouds, computing the convex hulls and for the surface mesh reconstruction needed to create the graph representation of data.

4.3.3. Experimental results and Discussion

Figure 4.12 illustrates the result of strawberry weight estimation experiments. As discussed earlier the goal here is to achieve 80-90% accuracy using the Percentage of Correct Weights (PCW) protocol described in Sec. 4.1. With (PCW @x) the percentage of accurate predictions, considering a tolerance of x, is indicated. Point cloud, graph and RGB+depth-based state-of-the-art neural networks have been evaluated on a test set of RGB-D images of berries annotated with their ground truth weight value. Obviously this tasting dataset was never seen by the models during training.

The point cloud-based network, PointConv [132] provides only 15.5% (PCW @0.1) to 28.90% (PCW @0.2). Similarly, PointNet [103] gives only around 23.00% (PCW @0.1) to 41.90% (PCW @0.2) accuracy while PointNet++ [102] gives only 27.23% (PCW @0.1) to 48.30% (PCW @0.2) accuracy. From Figure 4.12, it can also be concluded that the graph-based networks achieve performances similar to point cloud-based networks. DGCN [131], GCN [63] and HGNN [83], provides 44.20% , 44.30% and 46.10% accuracy at PCW @0.20. Figure 4.12 shows that the performance of the two-stream model based on EfficientNet is

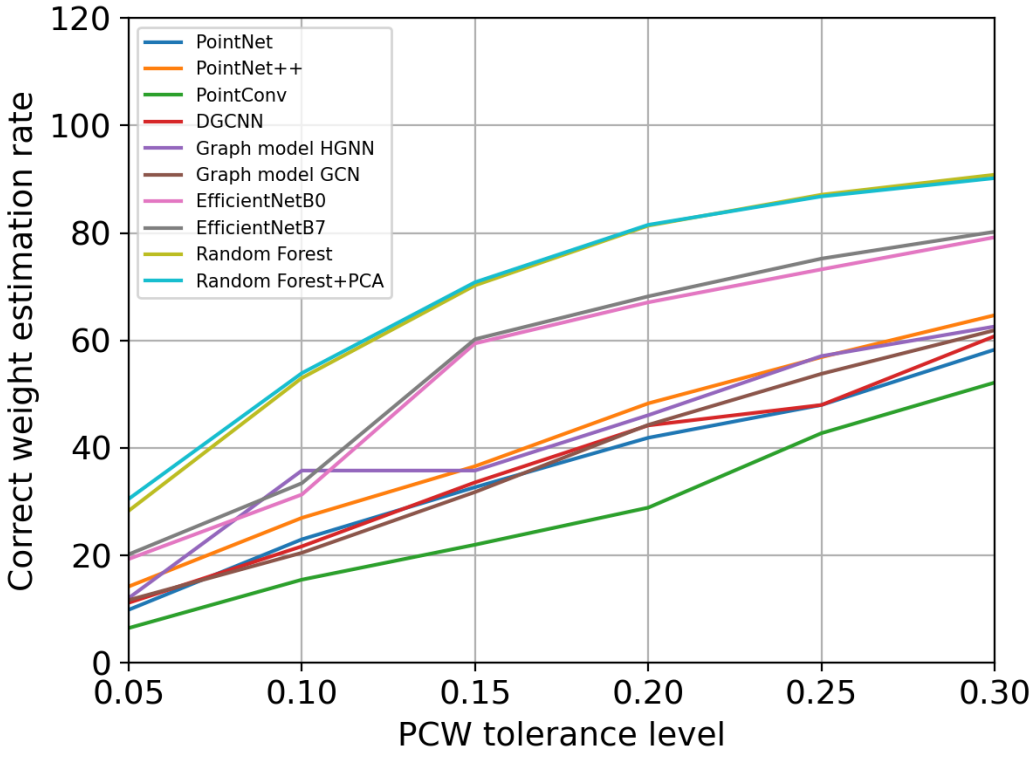


Figure 4.12: Weight estimation results.

more accurate than point cloud and graph-based networks. EfficientNet-B0 and B7 give much better results with 67.10 % and 68.21 % accuracy at PCW @0.2, but still far from suitability for selective harvesting.

This motivates the proposal of the **Random Forest-based method which performs much better than the other state-of-the-art deep methods**. The main novelty of the proposed algorithm is that by including the largest principal component using PCA, the method learns the orientation of the strawberry that helps with slightly more accurate weight estimation. The model performs 81.36 and 81.99% PCW @0.2 without and with PCA, respectively. By inclusion of PCA, the performance improves from 28.32% to 29.63% PCW @0.05. The graph in Fig 4.12 shows that **the inclusion of the PCA in the feature vector improves the performance across all the PCW levels.**

5 | Deep-ProMP: Path planning from visual information

5.1. Problem formulation: from perception to path planning

Robots for autonomous fruit picking require tightly integrated systems that incorporate perception, motion planning, and, in the last stage, manipulation. This section deals with the **motion planning problem for autonomous strawberry harvesting**. After the recognition of the target fruit to be picked at the perception stage, described in Ch. 3, the ultimate goal for the robotic systems is the ability to efficiently and successfully reach-to-pick harvest-ready strawberries regardless of obstructions and fruit cluster configurations. To achieve the reach-to-pick task, robot manipulators need to generate safe and smooth motions. The goal is to have a mapping between the detection information of the target ripe strawberry, furnished by the perceptive stage of the pipeline, and the robot end-effector trajectory towards the target berry (Fig. 5.1). This mapping is intended to be made through the segmented image (containing the detection information) and with the use of deep learning techniques. In other words, the objective is to find a non-linear mapping (represented by a trained neural network) from the segmented RGB image pixel values (the depth information is not exploited) to the robot end-effector trajectory for the reach-to-pick task. In this way also the path planning (together with the detection) part depends only on the visual information made available by a simple RGB camera.

Everything has been developed in the **Learning from Demonstrations (LfD) setting**. In the context of robotics and automation, LfD is the paradigm in which robots acquire new skills by learning to imitate an expert. This framework allows the learning of a huge number of possible tasks, and particularly it is useful when it is easier for an expert to demonstrate the desired behaviour rather than to hard code a policy able to perform it. Instead of requiring users to analytically decompose and manually program the desired behavior, LfD takes the view that an appropriate robot behaviour can be derived from

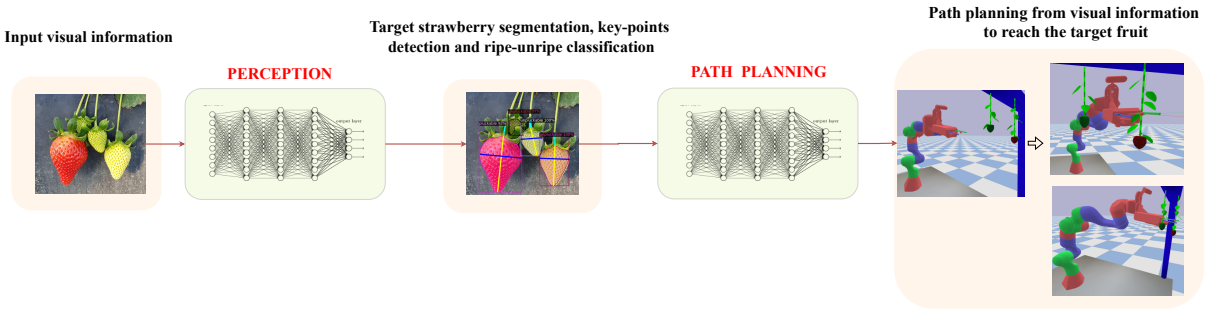


Figure 5.1: Pipeline from perception to path planning from the visual information.

observations of a human's performance thereof. The aim is for robot capabilities to be more easily extended and adapted to novel situations, even by users without programming ability.

There are two main phases in LfD:

- **Gathering examples**: the process of recording example data from the expert to derive a policy from.
- **Deriving policies**: analyzing examples to determine a policy.

In the case considered, in which the task to be performed is a target-reaching task, the human demonstrations are performed manually moving the robot end-effector towards the targeted berry in kinesthetic teaching mode, while the joint trajectories are recorded. The policy is then determined training a neural network to learn the mapping from the segmented input RGB image, in which the target fruit is detected and marked, to the joints trajectories to reach it, based on the data collected from the human expert demonstrations (Fig. 5.2). In Sec. 5.2 the complete mathematical formulation of the problem is provided.

A **probabilistic approach** has been preferred over a deterministic one in terms of trajectory prediction. This means that given the RGB image with the target berry, not only a single deterministic trajectory but a distribution of trajectories (assumed Gaussians) is predicted by the trained neural network. This way, the setting can be exploited in future developments to sample from the predicted distribution to optimise the trajectory to meet secondary objectives, like collision avoidance. Moreover, this leads to the exact reproduction of human behaviour, which is intrinsically stochastic when it deals with performing a task that has not a strictly unique way to be performed. Given a target fruit, there can be different possible ways of reaching it, and this stochasticity has been included in the dataset of recorded human demonstrations by repeating the reach-to-pick movement multiple times.

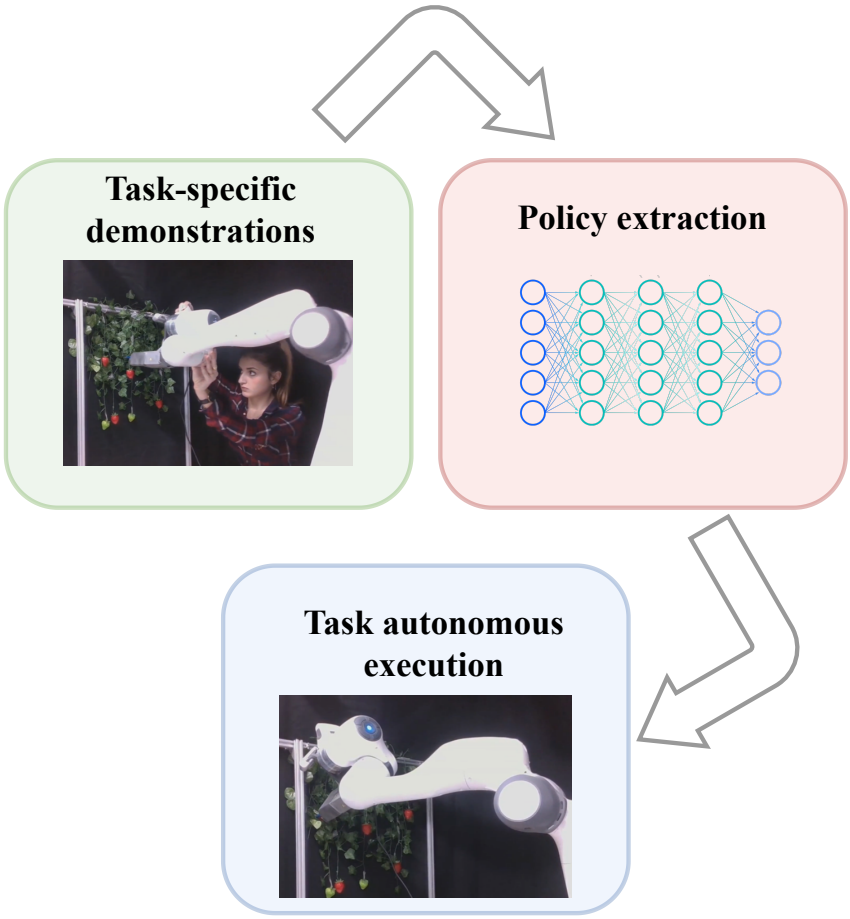
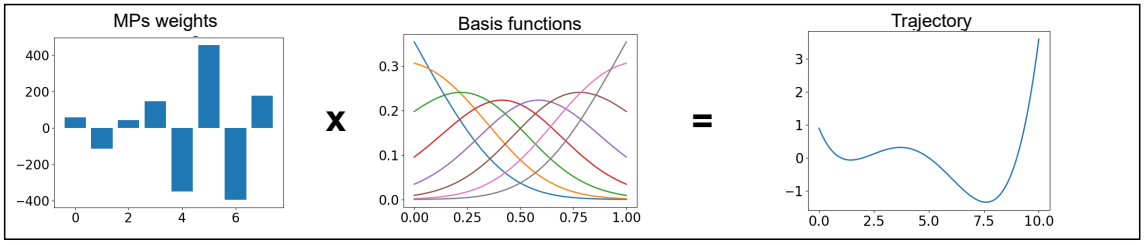


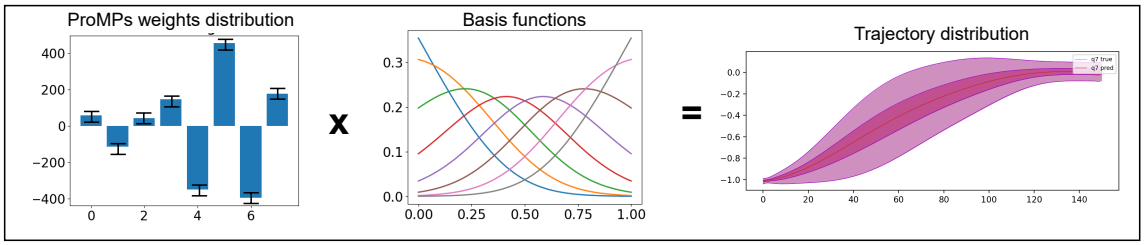
Figure 5.2: Learning from Demonstrations scheme (LfD).

Different neural network architectures have been implemented and compared to achieve the best performances in path prediction (Sec. 5.3). All the proposed models are composed of two parts: (1) part-1 encodes the high-dimensional input RGB image in a low-dimensional latent space vector, preserving all the relevant information for image reconstruction, using a set of convolutional layers, (2) part-2 maps the latent embedded vector to the relative trajectory distribution using a Multi-Layer Perceptron (MLP), i.e. a sequence of dense layers. The models differ in how they perform the encoding (1) or in how they predict the final trajectory distribution from the latent space (2).

Also, a **novel training method for learning domain-specific latent features** has been developed. This has been done so that the latent representation of the input RGB image does not only contain useful information for image reconstruction but also for the prediction of the reach-to-pick trajectory (Sec. 5.4).



(a) Movement Primitives (MPs)



(b) Probabilistic Movement Primitives (ProMPs)

Figure 5.3

Another important feature of the developed approach is the use of the **Probabilistic Movement Primitives (ProMPs)** [94] framework to represent the demonstrated-predicted trajectory distributions. ProMPs are the probabilistic extension of the Movement Primitives formulation (MP) [117], which is a compact reparameterization of a trajectory into a finite set of numbers, called weights. The concept behind it is that every kind of trajectory (considered as the variation of a quantity in time) can be represented as the sum of a set of standardized translated Gaussian functions, scaled by the relative weight, namely a number (Fig. 5.3a). So, just from a finite set of weights, it is possible to reconstruct a trajectory of any length. The number of weights and Gaussians needed to represent with an acceptable accuracy the trajectory is determined experimentally. For example, in Fig. 5.3a a set of 8 weighted Gaussians describes accurately the desired trajectory. To represent a distribution of trajectories instead of a single trajectory, it is possible to simply consider a set of probabilistic distributions of weights instead of a set of deterministic ones; so every weight is described by its mean and standard deviation values (Fig. 5.3b). This approach has been exploited to represent the predicted policy for the reach-to-pick task so that the neural network is trained to output a set of weights distributions (their mean and standard deviation values). In this way, the dimensionality of the problem is reduced, since the neural network is not required to predict a trajectory, considered as a value for every time instant, but it is only required to predict a finite set of weights (in number much lower than the one of time instants). In Sec. 5.3 the details of the proposed deep models are shown, while in Ch. 6 the experimental results obtained in simulation and with the real robotic arm are provided.

5.2. Mathematical formulation of the problem: from visual information to path prediction

Let us consider a set of N_{tr} demonstrations $\mathcal{T} := \{\{\mathbf{Q}^1, \mathbf{I^1}\}, \ldots, \{\mathbf{Q}^{N_{\text{tr}}}, \mathbf{I^{N_{\text{tr}}}}\}\}$ for the reach-to-pick task, where $\mathbf{Q}^n$ $(n = 1, \ldots, N_{\text{tr}})$ are the joint/task space sets of trajectories, and $\mathbf{I^n}$ is the raw RGB image taken from the corresponding robot's workspace that captures the target berry to be reached. A set of trajectories instead of a single one (for every joint or task space coordinate) is captured since the probabilistic face of the behaviour should be captured. In other words, for every single target berry, the reach-to-pick movement is performed S times (S trials compose a single set of trajectories) to extract a distribution of effective trajectories (for every joint/task space coordinate) in reaching the target berry. For a single joint (or task space coordinate) a set of trajectories relative to a certain target berry is defined as the ordered set $\mathbf{q_j}$ expressed in Eq. 5.1.

$$\mathbf{q_j} := \left\{\mathbf{q}_s^j\right\}_{s=1,\ldots,S} := \left\{q_{t,s}^j\right\}_{t=1,\ldots,T;s=1,\ldots,S} \tag{5.1}$$

$q_{t,s}^j \in \mathbb{R}$ is the joint (or task space coordinate) position during trial s at time instant t. Considering all the joints (or task space coordinates) together, $\mathbf{Q} := \{\mathbf{q}_1, ..., \mathbf{q}_{N_{\text{joint}}}\}$, where N_{joint} is the number of the joints (or Degree-of-freedoms (DOFs)) of the manipulator. In the following, the proposed approach is formulated in joint space, but it can be easily extended to obtain predictions in task space. The ProMPs framework is exploited to represent the demonstrated sets of trajectories. The deterministic Movement Primitives model is described in Eq. 5.2, where ψ_i are basis functions (usually Gaussians [19]) evaluated at $z(t)$. z is a phase function that allows time modulation. If no modulation is required, then $z(t) = t/f$, where f is the sampling frequency of the trajectory. $\theta_i \in \mathbb{R}$ are weights.

$$\mathbf{q}_s^j = \sum_{i=1}^{N_{\text{bas}}} \theta_i \psi_i(z(t)) \tag{5.2}$$

To pass to a probabilistic framework, a robot trajectory is described with an observation uncertainty added to the deterministic MPs model, as shown in Eq. 5.3 where $\epsilon_{\mathbf{q}_s^j}$ adds zero-mean Gaussian observation noise with variance $\Sigma_{\mathbf{q}_s^j}$.

$$\mathbf{q}_s^j = \sum_{i=1}^{N_{\text{bas}}} \theta_i \psi_i(z(t)) + \epsilon_{\mathbf{q}_s^j} \tag{5.3}$$

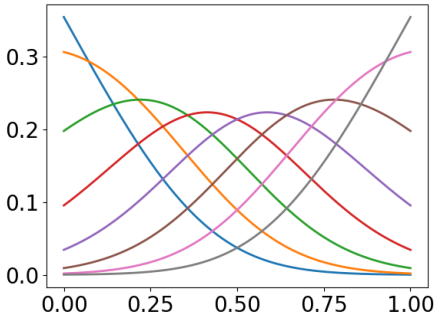


Figure 5.4: Gaussian basis functions.

For stroke-like movements, normalised Gaussian basis functions are used $\psi_i(t) := \frac{b_i(z(t))}{\sum_{j=1}^{N_{\text{bas}}} b_j(z(t))}$ where $b_i(z(t)) := \exp\left(-\frac{(z(t)-c_i)^2}{2h}\right)$. Some of them are represented in Fig. 5.4.

Eq. 5.3 can also be written in a matrix form (Eq. 5.4).

$$\mathbf{q}^j_{t,s} = \mathbf{\Psi}_t^T \mathbf{\Theta}^{\mathbf{s}}_{\mathbf{j}} + \epsilon_{q^j_{t,s}} \tag{5.4}$$

with: $\mathbf{\Psi}_t := (\psi_1(z(t), \ldots, \psi_{N_{\text{bas}}}(z(t)) \in \mathbb{R}^{N_{\text{bas}} \times 1}, \mathbf{\Theta}^{\mathbf{s}}_{\mathbf{j}} := (\theta_1, \ldots, \theta_{N_{\text{bas}}}) \in \mathbb{R}^{N_{\text{bas}} \times 1}$, $\mathbf{\Omega} := (\mathbf{\Theta^s}_1, \ldots, \mathbf{\Theta^s}_{N_{\text{joint}}}) \in \mathbb{R}^{N_{bas} N_{\text{joint}} \times 1}$ and $\mathbf{\Phi} := [\mathbf{\Psi}_1, \ldots, \mathbf{\Psi}_T]^T \in \mathbb{R}^{T \times N_{\text{bas}}}$.

The Probabilistic Movement Primitives (**ProMP**) [94] model directly follows from this formulation. From Eq. 5.3, it follows that the likelihood of observing $q^j_{t,s}$ is given by $p(q^j_{t,s}|\mathbf{\Theta}) = \mathcal{N}\left(q^j_{t,s} \,\middle|\, \mathbf{\Psi}_t^T \mathbf{\Theta}^{\mathbf{s}}_{\mathbf{j}}, \mathbf{\Sigma}_{q^j_{t,s}}\right)$. Since $\Sigma_{q^j_{t,s}}$ is the same for every time step t and every trial s ($\Sigma_{q^j_{t,s}} = \Sigma_{q_j}$), the values $q^j_{t,s}$ are taken from independent and identical distributions. Hence, the likelihood of observing a trajectory $\mathbf{q}^{\mathbf{j}}_{\mathbf{s}}$ is given by Eq. 5.5.

$$p(\mathbf{q}^{\mathbf{j}}_{\mathbf{s}}|\mathbf{\Theta}^{\mathbf{s}}_{\mathbf{j}}) := \prod_{t=1}^{T} p(q^j_{t,s}|\mathbf{\Theta}^{\mathbf{s}}_{\mathbf{j}}) \tag{5.5}$$

It can be assumed that the weight parameters are taken from a distribution. The distribution of $q^j_{t,s}$, which does not depend on $\mathbf{\Theta}^{\mathbf{s}}_{\mathbf{j}}$, but on $\rho := (\mathbf{\Theta}_{mean,j}, \mathbf{\Sigma}_{\Theta_j})$, can be estimated. This is done by marginalising $\mathbf{\Theta}^{\mathbf{s}}_{\mathbf{j}}$ out in the distribution as shown in Eq. 5.6 where $\mathbf{\Theta}^{\mathbf{s}}_{\mathbf{j}} \sim p(\mathbf{\Theta}^{\mathbf{s}}_{\mathbf{j}}|\rho) = \mathcal{N}(\mathbf{\Theta}^{\mathbf{s}}_{\mathbf{j}}|\mathbf{\Theta}_{mean,j}, \mathbf{\Sigma}_{\Theta_j})$.

$$\begin{aligned} p(q^j_{t,s}|\rho) &= \int \mathcal{N}\left(q^j_{t,s} \,\middle|\, \mathbf{\Psi}_t^T \mathbf{\Theta}^{\mathbf{s}}_{\mathbf{j}}, \Sigma_{q_j}\right) \mathcal{N}\left(\mathbf{\Theta}^{\mathbf{s}}_{\mathbf{j}} \,\middle|\, \mathbf{\Theta}_{mean,j}, \mathbf{\Sigma}_{\Theta_j}\right) \mathrm{d}\mathbf{\Theta} \\ &= \mathcal{N}\left(q^j_{t,s} \,\middle|\, \mathbf{\Psi}_t^T \mathbf{\Theta}^{\mathbf{s}}_{\mathbf{j}}, \Sigma_{q_j} + \mathbf{\Psi}_t^T \mathbf{\Sigma}_{\Theta_j} \mathbf{\Psi}_t\right) \end{aligned} \tag{5.6}$$

This means that the trajectory distribution can be represented by its mean and covariance values $(\mathbf{q}_{mean,j}, \mathbf{\Sigma}_{q_j})$, which in turn can be derived by the mean and covariance values of the ProMPs weights $(\mathbf{\Theta}_{mean,j}, \mathbf{\Sigma}_{\Theta_j})$, as described in Eq. 5.7.

$$\begin{aligned} \mathbf{q}_{mean,j} &= \mathbf{\Phi}^T \mathbf{\Theta}_{mean,j}, \\ \mathbf{\Sigma}_{q_j} &= \mathbf{\Phi}^T \mathbf{\Sigma}_{\Theta_j} \mathbf{\Phi} \end{aligned} \tag{5.7}$$

The reach-to-pick policy predicted by the trained neural network is determined by the predicted mean and covariance values of the ProMPs weights. The probabilistic neural network that maps the visual sensory information of a robot's workspace to the distribution of robot trajectories expressed in weights space (with $\mathbf{\Theta_{mean,j}}$ and $\mathbf{\Sigma_{\Theta_j}}$) is called Deep Probabilistic Movement Primitives (**Deep-ProMP**). Deep-ProMP learns the relation between $\mathbf{\Theta_{mean,j}}$ and $\mathbf{\Sigma_{\Theta_j}}$ and the input image.

$$\mathbf{\Theta_{mean,j}}, \mathbf{\Sigma_{\Theta_j}} = \mathrm{f}_j(\hat{\boldsymbol{W}}_j, \mathbf{I}^n, \hat{\sigma}_j) \tag{5.8}$$

Eq. 5.8 shows that $\mathbf{\Theta_{mean,j}}$, $\mathbf{\Sigma_{\Theta_j}}$ are equivalent to a nonlinear function, denoted by $\mathrm{f_j}$, of the input image $\mathbf{I^n}$, the weight parameter $\hat{\boldsymbol{W}}_j$ and the node activation $\hat{\sigma}_\mathrm{j}$. $\mathrm{f_j}$ is a non-linear deep model mapping the image $\mathbf{I^n}$, taken by robot's camera at a home position, to the ProMPs weights distributions. The weights distribution generates the corresponding trajectory distribution as per Eq. (5.7). Predicting $\mathbf{\Theta_{mean,j}}$, $\mathbf{\Sigma_{\Theta_j}}$ instead of $\mathbf{q}_{mean,j}, \mathbf{\Sigma}_{q_j}$ means reducing the dimensionality of the problem. For example, a trajectory of 150 time steps can be represented with a set of just 8 weights. Every joint (or task space coordinate) trajectory is considered decoupled from the other joints (or task space coordinates) trajectories, so the mean vector $\mathbf{q}_{mean,j}$ is 150-dimensional and the covariance matrix $\mathbf{\Sigma}_{q_j}$ is a 150 by 150 matrix, expressing the correlation of the trajectory at a certain time instant with the trajectory values at the other time instants. This identical information can be expressed with the weights mean vector $\mathbf{\Theta_{mean,j}}$ (8-dimensional) and covariance matrix $\mathbf{\Sigma_{\Theta_j}}$ (8 by 8) (Fig. 5.5).

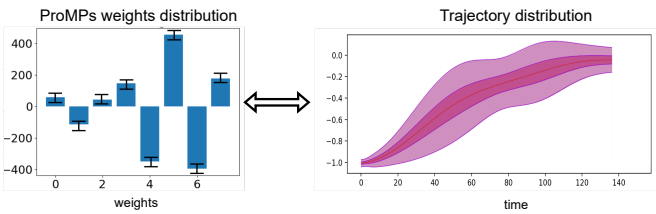


Figure 5.5: From ProMPs weights distribution to trajectory distribution. Shaded purple areas represent $\pm\sigma$ and $\pm 2\sigma$.

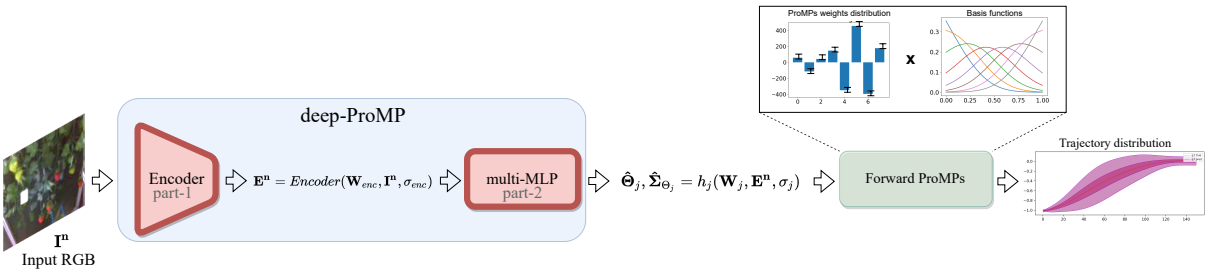


Figure 5.6: Two-fold design of Deep-ProMP.

5.3. Proposed Approach

Different baselines for the Deep-ProMP model architecture have been tested to gradually improve the accuracy in the prediction of the demonstrated behaviour. All the implemented architectures have in common a **twofold design** (Fig. 5.6).
They are indeed composed by two parts: (1) part-1 encodes the high-dimensional input RGB image $\mathbf{I^n}$ in a low-dimensional latent space vector, preserving all the relevant information for image reconstruction, using a set of convolutional layers (which constitute the *Encoder*) as per Eq. 5.9, where $\mathbf{W}_{enc}$ and σ_{enc} are the *Encoder* weights and activation functions;

$$\mathbf{E^n} = Encoder(\mathbf{W}_{enc}, \mathbf{I^n}, \sigma_{enc}) \tag{5.9}$$

(2) part-2 maps the latent embedded vector $\mathbf{E^n}$ to the relative ProMPs weights distribution $(\hat{\mathbf{\Theta}}_j, \hat{\mathbf{\Sigma}}_{\Theta_j})$, using a Multi Layer Perceptron (MLP) as per Eq. 5.10, where $\mathbf{W}_j$ and σ_j are the MLP weights and activation functions.

$$\hat{\mathbf{\Theta}}_j, \hat{\mathbf{\Sigma}}_{\Theta_j} = h_j(\mathbf{W}_j, \mathbf{E^n}, \sigma_j) \tag{5.10}$$

This twofold design of the model has an advantage over a direct mapping of the image to the trajectories distributions as it yields higher prediction accuracy, as it is demonstrated in Sec. 6.2.

All the details of the proposed model architectures are shown in Appendix A.

5.3.1. AE-deep-ProMP

The first baseline proposed, called **AE-deep-ProMP**, makes use of the encoding layers (*Encoder*) of an **autoencoder (AE) network** as part-1 of Deep-ProMP to reduce the input dimensionality while preserving the important information for image reconstruction.

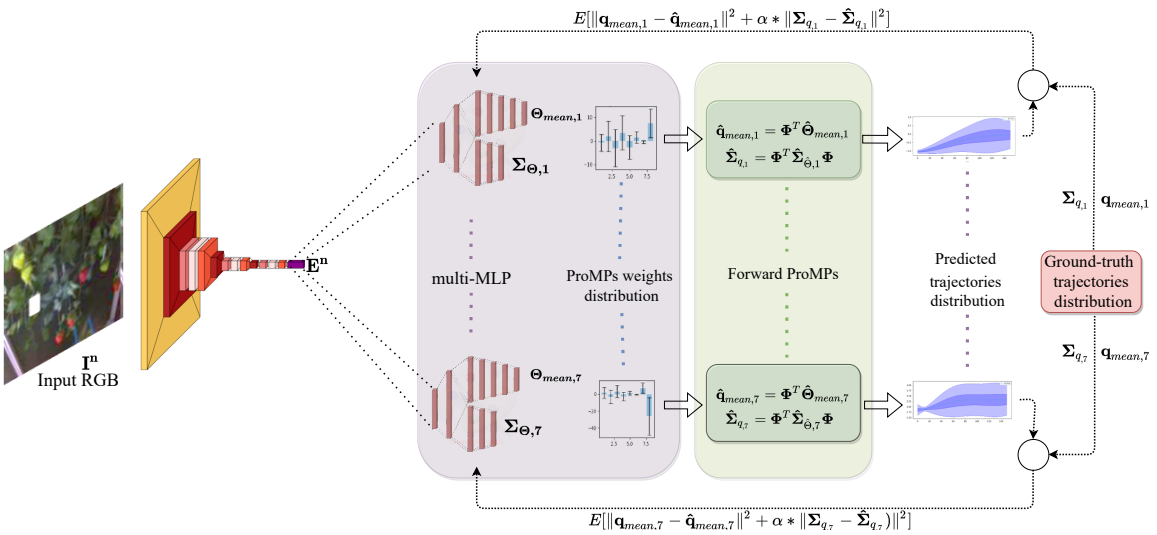


Figure 5.7: AE-deep-ProMP architecture and training.

An autoencoder [55] is a neural network composed of two main parts: an *Encoder* that maps the input into a lower-dimensional vector, and a *Decoder* that maps the latent vector back to a reconstruction of the input. In this case both the *Encoder* and *Decoder* are composed by a sequence of convolutional and dense layers. This means that, at first, an AE is trained to reconstruct the RGB images captured during the human expert demonstrations collection, as per Eq. 5.11, where $\hat{\mathbf{I}}^\mathbf{n}$ is the reconstructed image.

$$\begin{aligned} \mathbf{E^n} &= Encoder(\mathbf{W}_{enc}, \mathbf{I^n}, \sigma_{enc}) \\ \hat{\mathbf{I}}^\mathbf{n} &= Decoder(\mathbf{W}_{dec}, \mathbf{E^n}, \sigma_{dec}) \end{aligned} \tag{5.11}$$

The reconstruction loss used to train the autoencoder is the Mean Squared Error (MSE) expressed in Eq. 5.12.

$$Loss = E\left[\|\mathbf{I^n} - \hat{\mathbf{I}}^\mathbf{n}\|^2\right] \tag{5.12}$$

Once the AE is trained, the *Encoder* is able to extract a low-dimensional latent space representation of the input image, which still contains all the important information for image reconstruction. Algorithm 5.1 expresses the exact procedure for the AE training. Part-2 of AE-Deep-ProMP is composed of a sequence of dense layers that map the deterministic latent representation of the input image $\mathbf{E^n}$, to the ProMPs weights distribution, as per Eq. (5.10). A single MLP (for every joint or task space coordinate) is trained to predict the mean and covariance matrix $(\hat{\mathbf{\Theta}}_j, \hat{\mathbf{\Sigma}}_{\Theta_j})$ of the ProMPs weights. This is done minimizing the loss in Eq. 5.13 which is back-propagated to update the weights of the

MLPs, while the *Encoder* weights are kept fixed (Fig. 5.7).

$$L_j = E\left[\|\mathbf{q}_{mean,j} - \hat{\mathbf{q}}_{mean,j}\|^2 + \alpha\|\mathbf{\Sigma}_{q,j} - \hat{\mathbf{\Sigma}}_{q,j}\|^2\right] \tag{5.13}$$

The loss used to train the MLPs is the MSE computed in the trajectory space, and not in the ProMPs weights space. This means that, after the prediction of $(\hat{\mathbf{\Theta}}_j, \hat{\mathbf{\Sigma}}_{\Theta_j})$, $(\mathbf{q}_{mean,j}, \mathbf{\Sigma}_{q_j})$ are reconstructed using Eq. 5.7. The parameter α is determined experimentally as a tuning weight for the two components of the loss. The exact procedure to train the MLPs of AE-deep-ProMP model is expressed in Algorithm 5.2. Summing up, the training procedure for all deep-ProMP models (including deep-ProMP-AE) is composed of two subsequent and decoupled parts: (tr-1) considers the training of part-1 (the *Encoder*) and (tr-2) considers the training of part-2 (the MLPs).

Algorithm 5.1 Autoencoder training (tr-1).

Input: Encoder architecture *Encoder*, Decoder architecture *Decoder*, image $\mathbf{I^n}$, activation functions σ_{enc} and σ_{dec}.
Outputs: Encoder weights $\mathbf{W}_{enc}$, Decoder weights $\mathbf{W}_{dec}$ and reconstructed image $\hat{\mathbf{I}}^{\mathbf{n}}$

RGB Images Dataset:
$\mathcal{I} = \{\mathbf{I^n}\}_{n=\{1,\dots,N_{\text{tr}}\}}$
Initialise Autoencoder :
$\mathbf{E^n} = \mathit{Encoder}(\mathbf{W}_{enc}, \mathbf{I^n}, \sigma_{enc})$
$\hat{\mathbf{I}}^{\mathbf{n}} = \mathit{Decoder}(\mathbf{W}_{dec}, \mathbf{E^n}, \sigma_{dec})$
MSE Loss : Eq. 5.12
while $(e > \epsilon)$ **do**
  for all $\mathbf{I^n}$ *in* $\mathcal{I}$ **do**
    Forward Propagation:
    $\mathbf{E^n} = \mathit{Encoder}(\mathbf{W}_{enc,j}, \mathbf{I^n}, \sigma_{enc,j})$
    $\hat{\mathbf{I}}^{\mathbf{n}} = \mathit{Decoder}(\mathbf{W}_{dec}, \mathbf{E^n}, \sigma_{dec})$
    MSE Loss:
    $L_k = E\left[\|\mathbf{I^n} - \hat{\mathbf{I}}^{\mathbf{n}}\|^2\right]$
  end for
  Back Propagation:
  $\mathbf{W}_{enc}^{k+1} \leftarrow \{\mathbf{W}_{enc}^{k}, \frac{\partial L_k}{\partial \mathbf{W}_{enc}^{k}}\}$
  $\mathbf{W}_{dec}^{k+1} \leftarrow \{\mathbf{W}_{dec}^{k}, \frac{\partial L_k}{\partial \mathbf{W}_{dec}^{k}}\}$
end while
Encoder − Decoder:
$\mathbf{E^n} = \mathit{Encoder}(\mathbf{W}_{enc}, \mathbf{I^n}, \sigma_{enc})$
$\hat{\mathbf{I}}^{\mathbf{n}} = \mathit{Decoder}(\mathbf{W}_{dec}, \mathbf{E^n}, \sigma_{dec})$
end

In (tr-1) the weights of the MLPs are kept fixed, while in (tr-2) the weights of the *Encoder* are kept fixed. Algorithm 5.1 expresses (tr-1), while Algorithm 5.2 expresses (tr-2).

Algorithm 5.2 MLP training (tr-2).

Note: This pseudo code is for joint trajectory prediction without conditioning. Generalising it to the task space and/or with conditioning is straightforward.
Input: MLPs architecture h_j, encoder architecture *Encoder*, ProMP basis functions $\mathbf{\Phi}$ image $\mathbf{I^n}$, training set trajectories $\mathbf{q}_j$, activation functions σ_j and σ_{enc}.
Outputs: Encoder and MLP weights $\mathbf{W}_{enc}$, $\mathbf{W}_j$, mean trajectory $\hat{\mathbf{q}}_{mean,j}$ and covariance matrix $\hat{\mathbf{\Sigma}}_{q,j}$.

Probabilistic Dataset:
$\mathcal{T} = \{\mathrm{q_j}, \mathbf{I^n}\}_{n=\{1,...,N_{\mathrm{tr}}\},j=\{1,...,N_{joints}\}}$
Trajectories Mean and Covariance Extraction :
$\{\mathbf{q}_{mean,j}\ ,\ \mathbf{\Sigma}_{q_j}\}_{j=1...N_{joints}}$
Encoder-Decoder training for image reconstruction
Initialise DeepModel :
$\mathbf{E^n} = \mathit{Encoder}(\mathbf{W}_{enc}, \mathbf{I^n}, \sigma_{enc})$
$\hat{\mathbf{\Theta}}_{mean,j}, \hat{\mathbf{\Sigma}}_{\hat{\Theta}_j} = h_j(\mathbf{W}_j, \mathbf{E^n}, \sigma_j)$
Initialise ProMP:
$\hat{\mathbf{q}}_{mean,j} = \mathbf{\Phi}^T \hat{\mathbf{\Theta}}_{mean,j}$
$\hat{\mathbf{\Sigma}}_{q,j} = \mathbf{\Phi}^T \hat{\mathbf{\Sigma}}_{\hat{\Theta}_j} \mathbf{\Phi}$
MSE Loss : Eq. 5.13
while $(e > \epsilon)$ **do**
  for all $\{\mathbf{q}_{mean,j}, \mathbf{\Sigma}_{q,j}, \mathrm{I^n}\} \in \mathcal{T}$ **do**
    Forward Propagation:
    $\mathbf{E^n} = \mathit{Encoder}(\mathbf{W}_{enc,j}, \mathbf{I^n}, \sigma_{enc,j})$
    $\hat{\mathbf{\Theta}}_j, \hat{\mathbf{\Sigma}}_{\Theta_j} = h_j(\mathbf{W}_j, \mathbf{E^n}, \sigma_j)$
    Forward ProMP:
    $\hat{\mathbf{q}}_{mean,j} = \mathbf{\Phi}^T \hat{\mathbf{\Theta}}_{mean,j}$
    $\hat{\mathbf{\Sigma}}_{q_j} = \mathbf{\Phi}^T \hat{\mathbf{\Sigma}}_{\hat{\Theta}_j} \mathbf{\Phi}$
    MSE Loss:
    $L_{k,j} = E\left[\|\mathbf{q}_{mean,j} - \hat{\mathbf{q}}_{mean,j}\|^2 + \alpha * \|\mathbf{\Sigma}_{q_j} - \hat{\mathbf{\Sigma}}_{q_j}\|^2\right]$
  end for
  Back Propagation: *Keep* $\mathbf{W}_{enc}$ *Fixed & Train* $\mathbf{W}_j$ *with Loss in Eq.* **??**
  $\mathbf{W}_j^{k+1} \leftarrow \{\mathbf{W}_j^k, \frac{\partial L_k}{\partial \mathbf{W}_j^k}\}$
end while
Deep − ProMP:
$\hat{\mathbf{q}}_{mean,j} = \mathbf{\Phi}^T \hat{\mathbf{\Theta}}_{mean,j}$
$\hat{\mathbf{\Sigma}}_{q_j} = \mathbf{\Phi}^T \hat{\mathbf{\Sigma}}_{\hat{\Theta}_j} \mathbf{\Phi}$
end

5.3.2. VAE-deep-ProMP

The second baseline proposed is called **VAE-deep-ProMP**. The architecture is similar to the previous one, but now the latent representation is stochastic and not deterministic as per Eq. (5.14). This has been made possible by using a **Variational Autoencoder (VAE)** [62] instead of a simple AE.

$$\begin{aligned} \mathbf{E^n} &\sim \mathcal{N}(\mu_{\boldsymbol{E^n}}, \Sigma_{\boldsymbol{E^n}}) \\ \mu_{\boldsymbol{E^n}}, \Sigma_{\boldsymbol{E^n}} &= Encoder(\mathbf{W}_{enc}, \mathbf{I^n}, \sigma_{enc}) \end{aligned} \tag{5.14}$$

Just as a standard autoencoder, a variational autoencoder is an architecture composed of both an *Encoder* and a *Decoder* that is trained to minimise the reconstruction error between the encoded-decoded data and the initial data (Fig. 5.8).

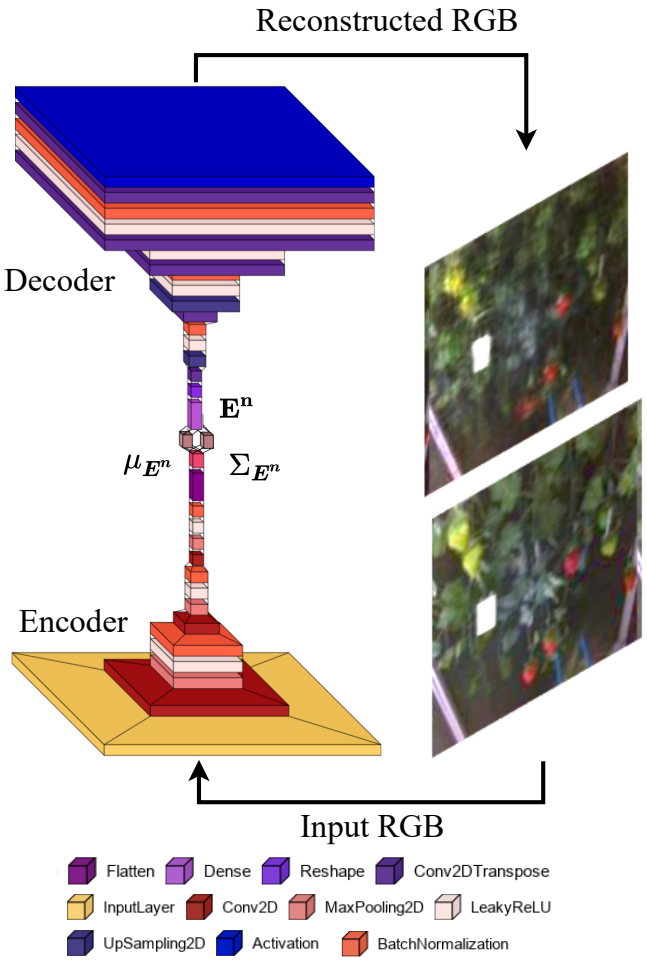


Figure 5.8: Variational autoencoder.

The *Decoder* reconstructs the input image after sampling from the latent space distribution predicted by the *Encoder*. However, to introduce some regularisation of the latent space, the encoding-decoding process is slightly modified: instead of encoding an input as a single vector, it is encoded as a distribution over the latent space.

A regular latent space is continuous and complete, meaning that every point in the latent space is a meaningful representation of data belonging to the input distribution and vice-versa. The encoded distributions are chosen to be normal so that the *Encoder* can be trained to return the mean vector and the covariance matrix that describe these Gaussians. Thus, the loss function that is minimised during the training of a VAE is composed of a "reconstruction term" (on the final layer), that tends to make the encoding-decoding scheme as performant as possible, and a "regularisation term" (on the latent layer), that tends to regularise the organisation of the latent space by making the distributions returned by the encoder close to a standard normal distribution. That regularisation term is expressed as the Kulback-Leibler divergence between the returned distribution and a standard Gaussian. This leads to the expression of the loss used to train the VAE in Eq. 5.15.

$$L = E\left[\|\mathbf{I^n} - \hat{\mathbf{I}}^{\mathbf{n}}\|^2 + KL\left(\mathcal{N}(\mu_{\boldsymbol{E^n}}, \Sigma_{\boldsymbol{E^n}}), \mathcal{N}(1, 0)\right)\right] \tag{5.15}$$

In Fig. 5.9 four input RGB and relative VAE reconstructed images are shown. The latent space representation is a 256-d vector, which is much smaller than the original (128 x 128 x 3) image.

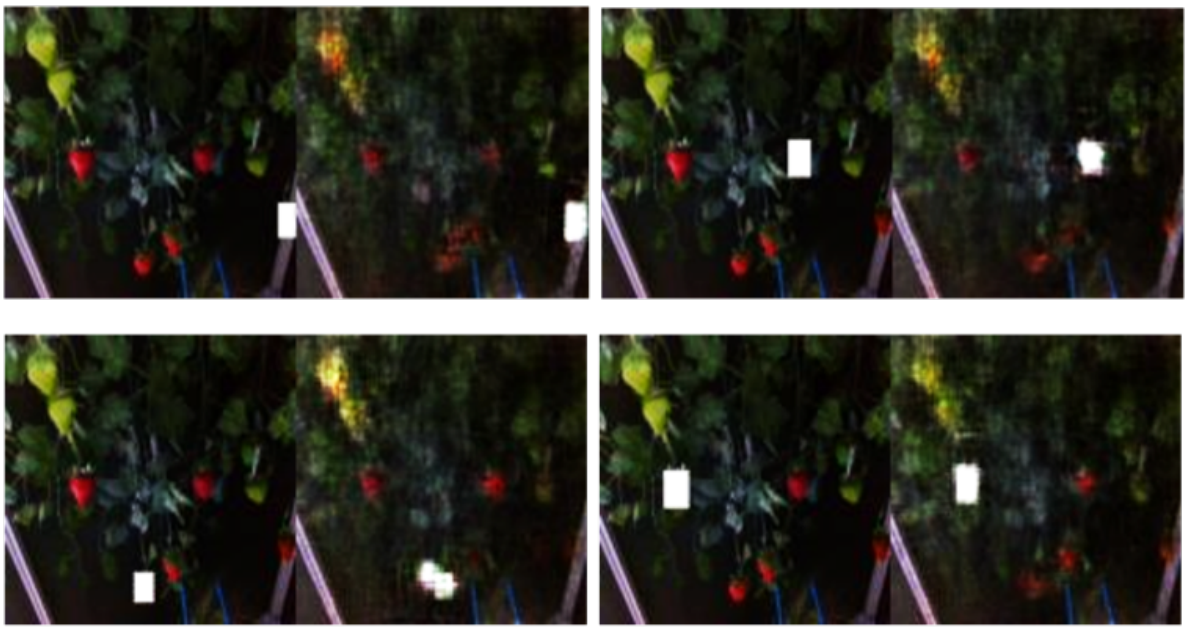


Figure 5.9: Variational autoencoder reconstructed images.

In Algorithm 5.3 the complete procedure to train the VAE is shown.

Part-2 of VAE-deep-ProMP is the same as part-2 of AE-deep-ProMP, and it is trained in the same way; after (tr-1) has been performed to train *Encoder* part of the VAE, (tr-2) is carried out using the same loss in Eq. 5.13.

Algorithm 5.3 Variational Autoencoder training.

Input: Encoder architecture *Encoder*, Decoder architecture *Decoder*, image $\mathbf{I^n}$, activation functions σ_{enc} and σ_{dec}.
Outputs: Encoder weights $\mathbf{W}_{enc}$, Decoder weights $\mathbf{W}_{dec}$ and reconstructed image $\hat{\mathbf{I}}^{\mathbf{n}}$

RGB Images Dataset:
$\mathcal{I} = \{\mathbf{I^n}\}_{n=\{1,\dots,N_{\text{tr}}\}}$
Initialise Autoencoder :
$\mu_{\boldsymbol{E^n}}, \Sigma_{\boldsymbol{E^n}} = Encoder(\mathbf{W}_{enc}, \mathbf{I^n}, \sigma_{enc})$
$\hat{\mathbf{I}}^{\mathbf{n}} = Decoder(\mathbf{W}_{dec}, \mathbf{E^n}, \sigma_{dec})$
MSE Loss : + *KL Loss* : Eq. 5.15
while $(e > \epsilon)$ **do**
  for all $\mathbf{I^n}$ *in* $\mathcal{I}$ **do**
    Forward Propagation:
    $\mu_{\boldsymbol{E^n}}, \Sigma_{\boldsymbol{E^n}} = Encoder(\mathbf{W}_{enc}, \mathbf{I^n}, \sigma_{enc})$
    $\mathbf{E^n} \sim \mathcal{N}(\mu_{\boldsymbol{E^n}}, \Sigma_{\boldsymbol{E^n}})$
    $\hat{\mathbf{I}}^{\mathbf{n}} = Decoder(\mathbf{W}_{dec}, \mathbf{E^n}, \sigma_{dec})$
    MSE Loss + KL Loss:
    $L_k = E\left[\|\mathbf{I^n} - \hat{\mathbf{I}}^{\mathbf{n}}\|^2 + KL\left(\mathcal{N}(\mu_{\boldsymbol{E^n}}, \Sigma_{\boldsymbol{E^n}}), \mathcal{N}(1, 0)\right)\right]$
  end for
  Back Propagation:
  $\mathbf{W}_{enc}^{k+1} \leftarrow \{\mathbf{W}_{enc}^{k}, \frac{\partial L_k}{\partial \mathbf{W}_{enc}^{k}}\}$
  $\mathbf{W}_{dec}^{k+1} \leftarrow \{\mathbf{W}_{dec}^{k}, \frac{\partial L_k}{\partial \mathbf{W}_{dec}^{k}}\}$
end while
Encoder − Decoder:
$\mu_{\boldsymbol{E^n}}, \Sigma_{\boldsymbol{E^n}} = Encoder(\mathbf{W}_{enc}, \mathbf{I^n}, \sigma_{enc})$
$\hat{\mathbf{I}}^{\mathbf{n}} = Decoder(\mathbf{W}_{dec}, \mathbf{E^n}, \sigma_{dec})$
end

5.3.3. cVAE-deep-ProMP

The third proposed model is called **cVAE-deep-ProMP**. It is very similar to VAE-deep-ProMP since it makes use of the encoding part of a trained VAE as part-1, but it also exploits the information of a conditional variable $\mathbf{c}$, concatenated with the latent vector, to predict the ProMPs weight distributions in part-2, as per Eq. (5.16).

$$\hat{\mathbf{\Theta}}_j, \hat{\mathbf{\Sigma}}_{\Theta_j} = h_j(\mathbf{W}_j, \mathbf{E^n}, \sigma_j, \mathbf{c}) \tag{5.16}$$

$\mathbf{c}$ helps the consequent MLP networks to tailor their behaviour according to some important information. In the case considered the information of the pixel coordinate of the center of the bounding box around the target berry has been considered as a conditional variable. The architecture is shown in Fig. 5.10.

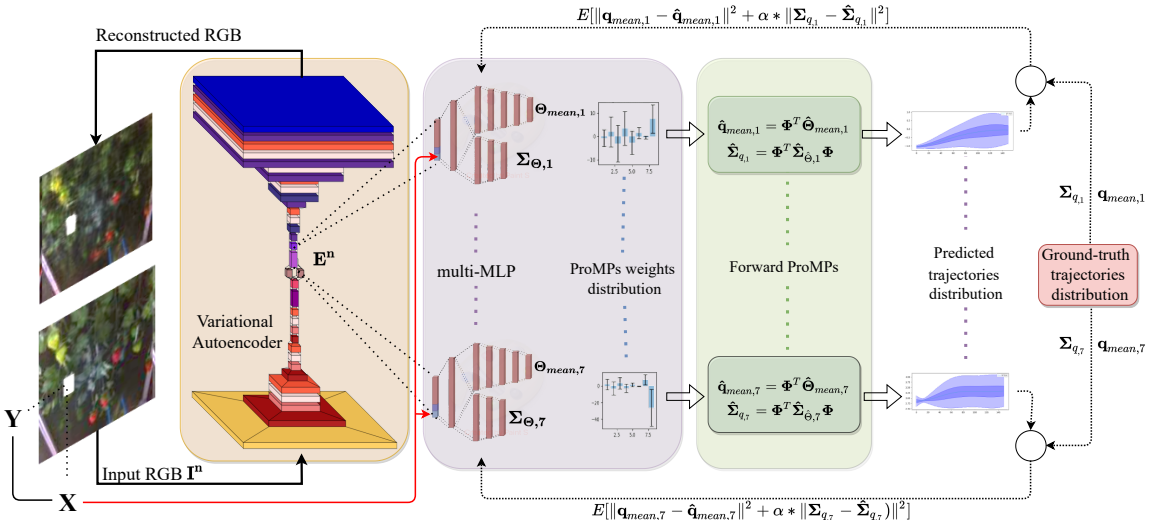


Figure 5.10: c-VAE-deep-ProMP architecture and training.

5.4. Domain-specific latent space learning

As described in the previous sub-sections, AE-deep-ProMP, VAE-deep-ProMP and cVAE-deep-ProMP have two ordered training stages: (tr-1) unsupervised training of part-1, so of AE or VAE (using image reconstruction loss and eventually KL loss), and (tr-2), supervised training of part-2 (using MSE error as per eq 5.13).

Tr-1 is done at first to train the encoder part; then, tr-2 is performed to train the MLPs. Hence, tr-1 and tr-2 are completely decoupled.

As such, the latent space is defined to maintain the information necessary for reconstructing the input RGB image, independently from the demonstrated trajectories distributions.

It can be argued that, while this non-domain-specific training tr-1 is useful for computer vision, it is not relevant for robotic tasks. Hence, it has been proposed to continue the training of the weights of *Encoder* using the loss in Eq. 5.17, while the MLP weights are kept fixed (Fig. 5.11).

$$e = E\left[\|\mathbf{q}_{mean,j}(t_{end}) - \hat{\mathbf{q}}_{mean,j}(t_{end})\|^2 + \alpha * \|\mathbf{\Sigma}_{q_j}(t_{end}) - \hat{\mathbf{\Sigma}}_{q_j}(t_{end})\|^2\right] \tag{5.17}$$

This is called **domain-specific latent space Deep-ProMP (l-Deep-ProMP)** and has been performed for all the three proposed architectures (AE/VAE/cVAE-deep-ProMP). In this way, there is a direct mapping of the latent space to the information useful both for image reconstruction and for trajectory prediction.

The domain-specific latent space learning Deep-ProMP is explained in detail in Algorithm 5.4.

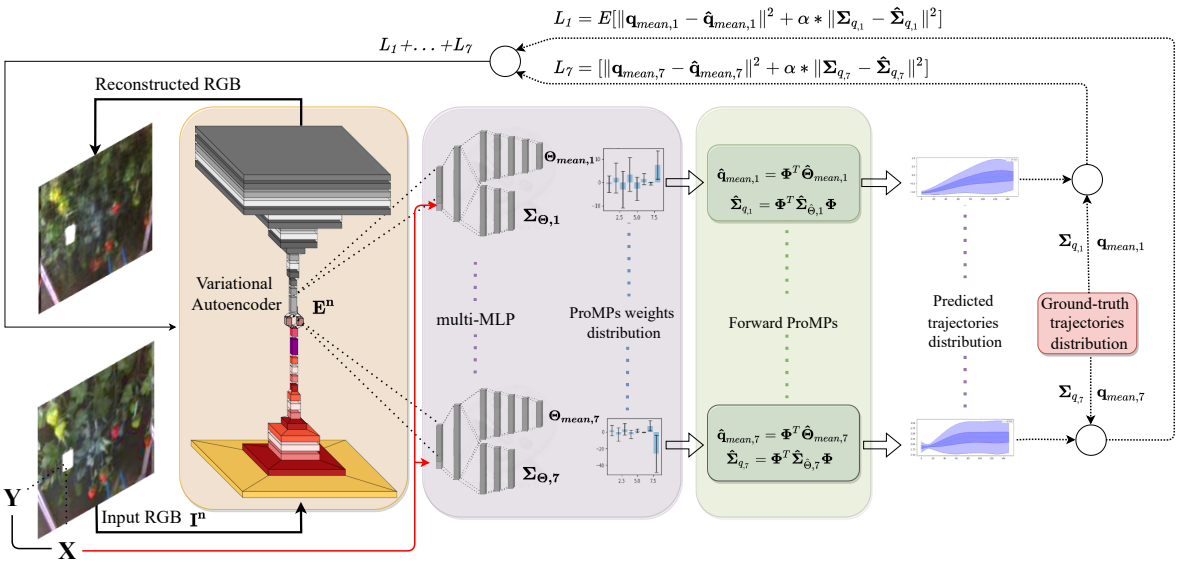


Figure 5.11: Latent space learning. In gray the non-trained parts of the model.

Algorithm 5.4 Domain-specific latent vector training Deep-ProMP.

Note: This pseudo code is for joint trajectory prediction. Generalising it to the task space is straightforward.
Input: MLPs architecture h_j, encoder architecture *Encoder*, ProMP basis functions $\mathbf{\Phi}$ image $\mathbf{I^n}$, training set trajectories $\mathbf{q}_j$, activation functions σ_j and σ_{enc}.
Outputs: Encoder and MLP weights $\mathbf{W}_{enc}$, $\mathbf{W}_j$, mean trajectory $\hat{\mathbf{q}}_{mean,j}$ and covariance matrix $\hat{\mathbf{\Sigma}}_{q_j}$.

Probabilistic Dataset:
$\mathcal{T} = \{\mathrm{q_j}, \mathbf{I^n}\}_{n=\{1,\dots,N_{\mathrm{tr}}\},j=\{1,\dots,N_{joints}\}}$
Trajectories Mean and Covariance Extraction :
$\{\mathbf{q}_{mean,j}\ ,\ \mathbf{\Sigma}_{q,j}\}_{j=1\dots N_{joints}}$
Encoder-Decoder training for image reconstruction
Initialise DeepModel :
$\mathbf{E^n} = Encoder(\mathbf{W}_{enc}, \mathrm{I^n}, \sigma_{enc})$
$\hat{\mathbf{\Theta}}_{mean,j}, \hat{\mathbf{\Sigma}}_{\hat{\Theta}_j} = h_j(\mathbf{W}_j, \mathbf{E^n}, \sigma_j)$
Initialise ProMP:
$\hat{\mathbf{q}}_{mean,j} = \mathbf{\Phi}^T \hat{\mathbf{\Theta}}_{mean,j}$
$\hat{\mathbf{\Sigma}}_{q_j} = \mathbf{\Phi}^T \hat{\mathbf{\Sigma}}_{\hat{\Theta}_j} \mathbf{\Phi}$
MSE Loss : Eq. 5.17
while $(e > \epsilon)$ **do**
  for all $\{\mathbf{q}_{mean,j}, \mathbf{\Sigma}_{q_j}, \mathbf{I^n}\} \in \mathcal{T}$ **do**
    Forward Propagation:
    $\mathbf{E^n} = Encoder(\mathbf{W}_{enc,j}, \mathbf{I^n}, \sigma_{enc,j})$
    $\hat{\mathbf{\Theta}}_j, \hat{\mathbf{\Sigma}}_{\Theta_j} = h_j(\mathbf{W}_j, \mathbf{E^n}, \sigma_j)$
    Forward ProMP:
    $\hat{\mathbf{q}}_{mean,j} = \mathbf{\Phi}^T \hat{\mathbf{\Theta}}_{mean,j}$
    $\hat{\mathbf{\Sigma}}_{q,j} = \mathbf{\Phi}^T \hat{\mathbf{\Sigma}}_{\hat{\Theta}_j} \mathbf{\Phi}$
    MSE Loss:
    $L_k = E\left[\|\mathbf{q}_{mean,j}(t_{end}) - \hat{\mathbf{q}}_{mean,j}(t_{end})\|^2 + \alpha * \|\mathbf{\Sigma}_{q_j}(t_{end}) - \hat{\mathbf{\Sigma}}_{q_j}(t_{end})\|^2\right]$
  end for
  Back Propagation: *Keep* $\mathbf{W}_j$ *Fixed & Train* $\mathbf{W}_{enc}$ *with Loss in Eq.* 5.17
  $\mathbf{W}^{k+1}_{enc} \leftarrow \{\mathbf{W}^k_{enc}, \frac{\partial L_k}{\partial \mathbf{W}^k_{enc}}\}$
end while
Deep − ProMP:
$\hat{\mathbf{q}}_{mean,j} = \mathbf{\Phi}^T \hat{\mathbf{\Theta}}_{mean,j}$
$\hat{\mathbf{\Sigma}}_{q_j} = \mathbf{\Phi}^T \hat{\mathbf{\Sigma}}_{\hat{\Theta}_j} \mathbf{\Phi}$
end

6 | Deep-ProMP: Experimental results

This chapter deals with the experimental results obtained implementing the proposed deep-ProMP architectures shown in Ch. 5 both in a simulated environment and with a real robotic arm.

In particular Sec. 6.1 is about the qualitative testing of the approach obtained with PyBullet, the open-source Python binding for the Bullet physics engine. A realistic environment with a 7-DoF robotic arm and a cluster of strawberries have been recreated and used to implement the approach from the demonstrations to the autonomous execution of the reach-to-pick task through the deep-ProMP models predictions.

In Sec. 6.2 the results obtained considering a series of real-robot tasks of picking strawberries with a mock setup in the laboratory of the University of Lincoln [1] are shown. In particular different types of analysis have been carried out. The performances of deep-ProMP-AE, deep-ProMP-VAE, and deep-ProMP-cVAE are compared before and after the domain-specific training. Furthermore, a comparison between predictions made in task and joint space is shown. Finally, a study on the latent space clustering level is reported.

6.1. Testing in simulation with PyBullet

The proposed approach has been first tested in a simulated environment and then applied to move a real robotic arm. The **Bullet physics engine** has been chosen for this purpose. Bullet is a free and open-source physics engine that simulates collision detection as well as soft and rigid body dynamics. In particular, the **PyBullet** Python binding for Bullet has been exploited. The robotic arm used in the real setup is a 7-DoF Panda manufactured by Franka Emika with a custom gripper specifically designed for strawberry picking and developed at the University of Lincoln [1], called UPH. The robotic arm with the UPH is shown in simulation in Fig. 6.1. The UPH has two separation fingers to manipulate leaves

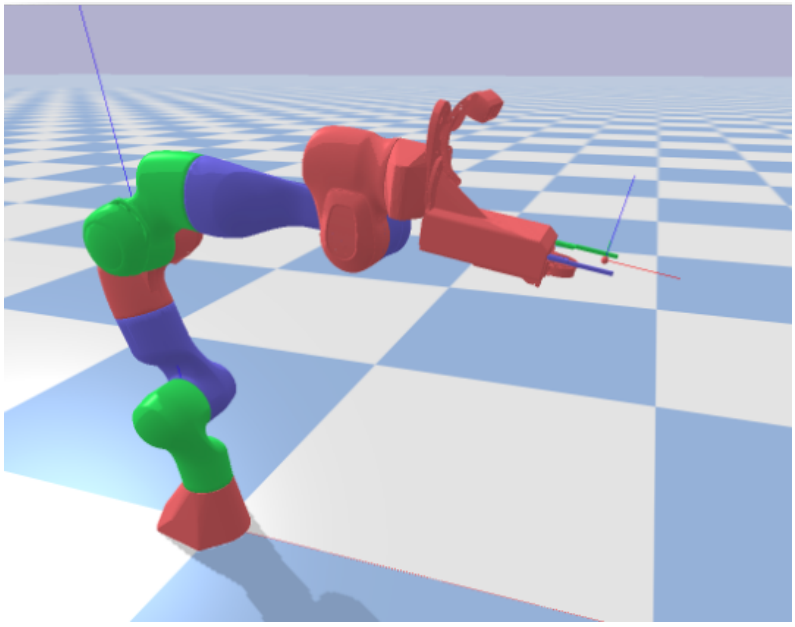


Figure 6.1: Robotic panda arm + UPH in PyBullet.

and branches during strawberry harvesting, and beneath them, there are two grasping fingers and a cutting blade. The objective of the reach-to-pick task is to arrive on top of the target berry with the two separation fingers open, as shown in Fig. 6.3.

The robotic arm is placed on a table, and a support for a cluster of strawberries is also recreated and placed in front of the robot. In Fig. 6.2 the simulated cluster of strawberries is shown. It has been designed so that there's always only a ripe strawberry, that is the one that the gripper of the UPH should reach, among multiple unripe strawberries and leaves in random number.

Once the simulated environment has been created, the demonstrations have been collected. Namely, for 25 different cluster configurations, the reach-to-pick trajectory has been performed and recorded. In particular, the joint trajectories, while reaching the

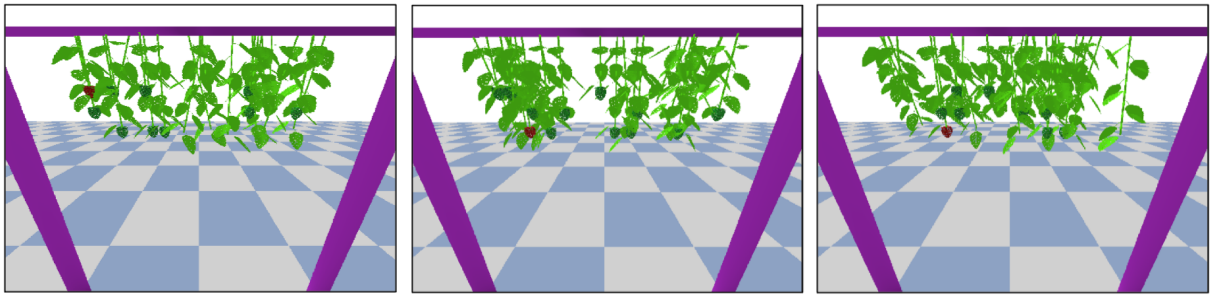


Figure 6.2: Simulated strawberries cluster in PyBullet.

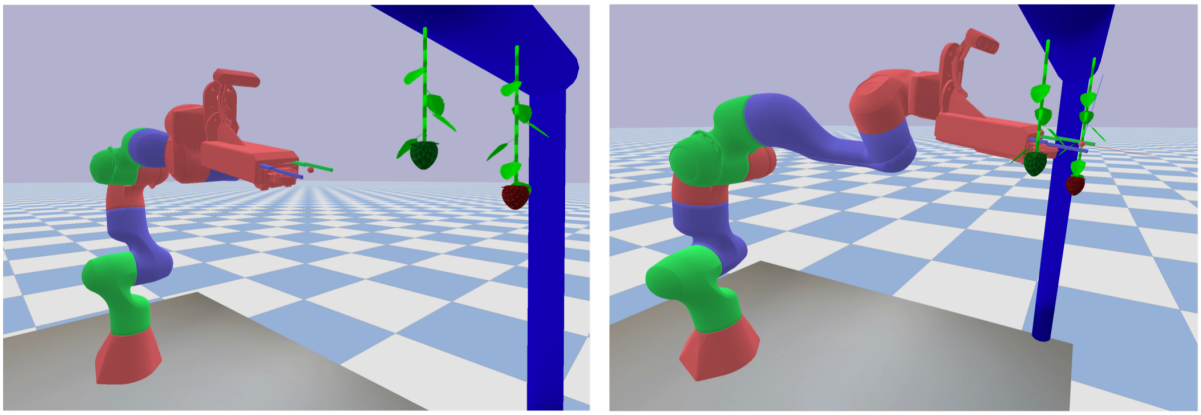


Figure 6.3: Reach-to-pick task in PyBullet. Home position and final position.

different target berries, have been captured. For every single configuration of the cluster, so for every different position of the target berry, the motion to reach the top of the berry with the UPH fingers open has been performed 10 times to extract the variability of the behaviour. This has been made possible since in simulation the position of the target berry is known and can be imposed as the final position of the end effector, computing the inverse kinematics to solve for the joints trajectories (Fig. 6.3). In Fig. 6.3 and 6.4 only the robotic arm (first in home position and then in target final position), the target berry and an unripe berry are rendered for clarity. In particular **IKFast** [129] has been used to solve the inverse kinematics. It is a powerful inverse kinematics solver provided within Rosen Diankov's OpenRAVE motion planning software. IKFast can analytically solve the kinematics equations of any complex kinematics chain, and generate language-specific files (like C++) for later use. The result is extremely stable solutions that can run as fast as 5 microseconds on recent CPUs. The variability in the trajectory execution has been obtained imposing different via-points and a different orientation of the end effector at the final position as shown in Fig. 6.4. Together with the recorded joint trajectories also the RGB images of the cluster from the camera on top of the UPH are collected (Fig. 6.2) to train the AE-deep-ProMP, VAE-deep-ProMP, and cVAE-deep-ProMP models to map the pixel values to the desired trajectories distributions. The simulated camera has been set up with the intrinsic and extrinsic parameters of an Intel RealSense D435i RGB-D camera since this is the model used in the real setting. At this stage, the testing of all the proposed architectures has been performed only in a qualitative way. Namely, it has only been verified that such approaches could be valid for the execution of a task of this type. After the training stages, the models were successfully able to predict the joint trajectories distributions suitable for reaching the target berry.

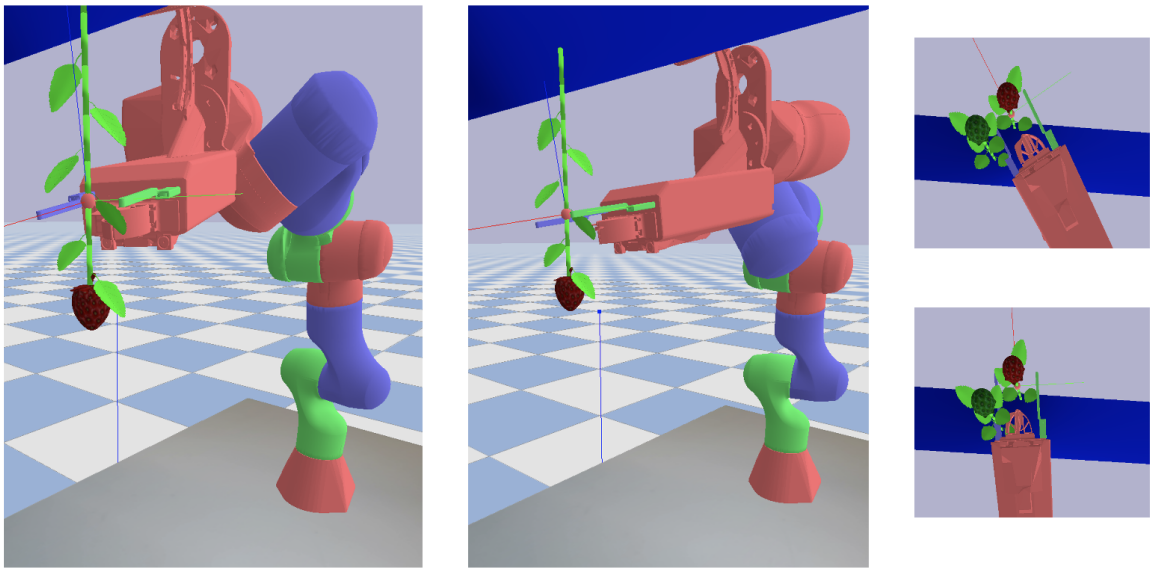


Figure 6.4: Reach-to-pick with final different orientations.

Since in this simplified setting, there is always only one ripe berry, clearly visible in the RGB images, no post-processing of the image has been performed before feeding it to the deep-ProMP models (the target berry detection and marking with the Detectron-2 [133] based model described in Ch. 3 is not needed).

In the experimental setup described in Sec. 6.2, instead, since multiple ripe berries are present in the RGB images, the targeted one has been detected and masked by its bounding box, before feeding the deep-ProMP models.

The simulation of the approach in PyBullet was successful to validate the idea and proceeding with the real setup since the models were able to predict valid trajectories distributions. No comparison among the models has been performed at this stage since it has been directly done during the experimentation with the real robotic arm.

6.2. Experimental results and discussion

To validate the proposed approach, a series of real-robot tasks of picking strawberries with a mock setup has been performed in the laboratory of the University of Lincoln [1]. In Sec. 6.2.1 the experimental setup is described as well as the process of human demonstrations collection. In Sec. 6.2.2 the implementation details are reported and finally, in Sec. 6.2.3 the quantitative results are shown to have a fair comparison among the proposed approaches.

6.2.1. Experimental set up and human demonstrations acquisition

The experimental setup consists of a 7-DoF Panda robotic arm manufactured by Franka Emika with a custom gripper (UPH) specifically designed for strawberry picking by the University of Lincoln (Fig. 6.5a). An Intel RealSense D435i RGB-D camera is mounted on the top of the gripper. The images are captured with VGA resolution (640,480). A mock set up with plastic strawberries to test the methods during the off-harvesting season has been created and it is shown in Fig. 6.6. It includes ripe and unripe strawberries together with leaves.

A dataset of 250 samples has been acquired, where each sample includes an RGB image of the scene and the robot joints trajectories, starting from a home configuration as shown in Fig. 6.5.

After taking an image, the robot is manually moved to reach the targeted berry in kinesthetic teaching mode (Fig. 6.7). The target berry in the input RGB image is masked with a white bounding box using the trained segmentation model based on Detectron-2 [133] and explained in Ch. 3. The movement is repeated 10 times for a single targeted strawberry to capture the demonstration variations and stochastic nature. Both the joint space and task space trajectories have been recorded. 5 different strawberry plant configurations have been created, each including 5 different target ripe berries.

(a) (b)

Figure 6.5: Panda robotic hand in home position.

Figure 6.6: Cluster of fake strawberries.

6.2.2. Implementation details

All the models have been trained and tested both for task and joint space predictions. Predicting in joint space means predicting the distributions of the 7 joints trajectories for the reach-to-pick task, while predicting in task space means predicting the end-effector position (x, y, z coordinates) and orientation (the four components of a quaternion) trajectories.

Working in task space is more general since the predictions are not bound to the very model of the robot; given a trajectory in task space, the relative trajectories of the joints can always be computed using the inverse kinematics of the specific robot. The trajectories have been represented with 8 ProMPs weights, since, as it can be seen from Fig. 6.8, this allows to have a negligible MSE between the original and ProMPs reconstructed trajectories.

Deep-ProMP-AE embeds the input image with the trained *Encoder* of AE. Deep-ProMP-

Figure 6.7: Human expert demonstrations.

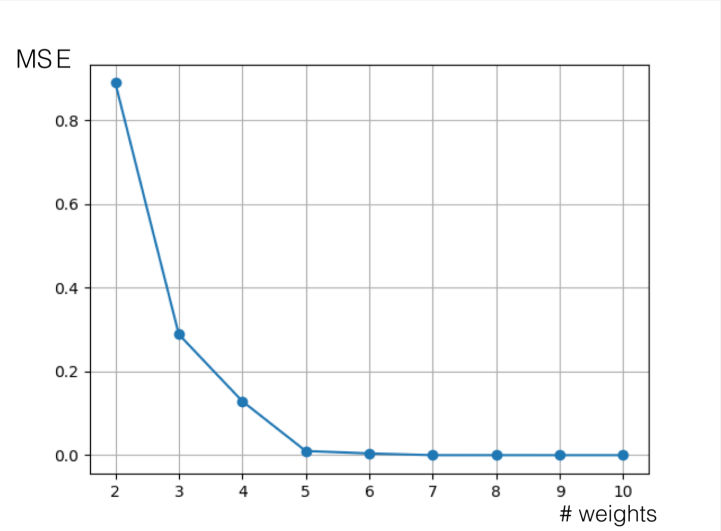


Figure 6.8: MSE vs #ProMP weights. Increasing the number of weights increases the accuracy in trajectory reproduction.

VAE utilises the trained *Encoder* of a VAE. Deep-ProMP-cVAE is conditioned by concatenating the normalized pixel coordinate of the target berry bounding box center, with the latent vector. The latent space representations are 256-dimensional vectors. For all the models, (tr-1) has been performed training for 250 epochs the AE or VAE to reconstruct the input images as described in Sec. 5.3. Then (tr-2) is performed using the loss in Eq. 5.13 to train the MLPs for 3000 epochs. In Fig. 6.9 the trend of the loss when training the MLP relative to a specific joint prediction is shown. It can be noticed that, at the final epochs, the loss becomes asymptotically horizontal, meaning that the learning

Figure 6.9: MLP training loss in logaritmic scale.

phase is finished. The prediction of the covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{\Theta}_j}$ is done exploiting the **LDL** decomposition for square positive definite matrices, as per Eq. 6.1. $\boldsymbol{L}$ is a lower triangular square matrix with unity diagonal elements, $\boldsymbol{D}$ is a diagonal matrix, and $\boldsymbol{L}^*$ is the complex conjugate transpose of $\boldsymbol{L}$.

$$\boldsymbol{\Sigma}_{\boldsymbol{\Theta}_j} = \boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^* \tag{6.1}$$

This means that instead of predicting the 56 components of $\boldsymbol{\Sigma}_{\boldsymbol{\Theta}_j}$, it is suffcient to predict the diagonal values of $\boldsymbol{D}$ (8) plus the non-zero values of $\boldsymbol{L}$ ($(8^2 - 8)/2 = 28$).

Moreover, the domain-specific latent space learning of these three models (l-Deep-ProMP-AE, l-Deep-ProMP-VAE, l-Deep-ProMP-cVAE) has been implemented. The latent space tuning loss used to train for 250 epochs more the encoder weights, is the prediction error at the last time instant, as per Eq. 5.17. This allows the tuning of the encoder weights according to specific task requirements. All the models have been implemented on Ubuntu 16 using TensorFlow 2.5. The learning rate has always been set to 0.0001 in all the training stages and the Adam optimizer has been used. In Fig. 6.10 the cVAE-deep-ProMP predictions on a test case (never seen during training) are shown. The ground truth and predicted joints trajectory distributions are respectively drawn in blue and red.

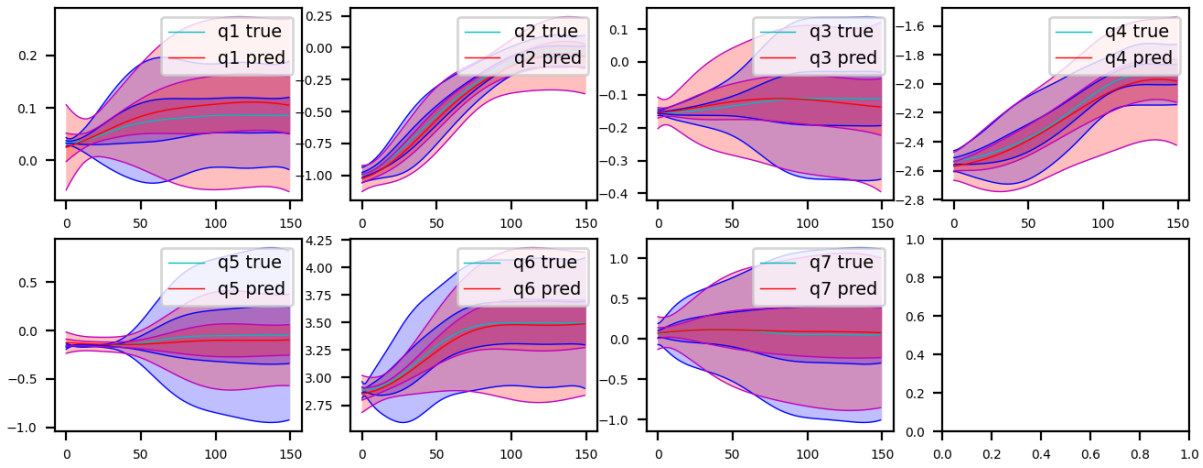


Figure 6.10: cVAE-deep-ProMP predictions in joint space.

6.2.3. Results

Three types of analysis have been carried out and the corresponding results are shown in Table 6.1, 6.2. and 6.3 and Fig. 6.12.

For the first experiment, the prediction performances of the 7 joint trajectories distributions with Deep-ProMP-AE, Deep-ProMP-VAE, and Deep-ProMP-cVAE are considered. Moreover, a model that maps directly the input image to the trajectories distributions with a series of convolutional and dense layers (Deep-ProMP-Direct) has been trained and tested.

Deep-ProMP-Direct has been used as a benchmark to demonstrate that the twofold design proposed (*Encoder* + MLPs) performs better than the direct mapping. The performances have been evaluated on a test set never seen in training or validation stages. The evaluation metric is the same as the loss used for the MLPs training, as shown in Eq. 5.13.

Table 6.1 shows that the prediction error is improving going from Deep-ProMP-Direct to Deep-ProMP-AE, Deep-ProMP-VAE, and finally to Deep-ProMP-cVAE. Hence, **Deep-ProMP-cVAE is outperforming all other models**. The same observation can be done for the case of predictions made in task space as shown in Table. 6.2.

Another experimentation has been conducted on the same test set to compare the model performances between task and joint space predictions.

The evaluation metric in Eq. 6.2 is used. It considers the error at the final point (when the target berry has to be reached) between the predicted and ground truth end-effector position and orientation. The error on position is simply the Euclidean distance between the predicted and ground-truth final points in the space.

Table 6.1: Joint space predictions.
deep-ProMp-cVAE is the most accurate model.

Joint	deep-ProMP -AE	deep-ProMP -VAE	drop	deep-ProMP -cVAE	drop
J1	5.4×10^{-4}	1.30×10^{-4}	-74%	$\mathbf{1.00 \times 10^{-4}}$	-28%
J2	48.9×10^{-4}	14.4×10^{-4}	-70%	$\mathbf{9.00 \times 10^{-4}}$	-38%
J3	93.0×10^{-4}	9.60×10^{-4}	-89%	$\mathbf{5.90 \times 10^{-4}}$	-38%
J4	37.1×10^{-4}	19.3×10^{-4}	-47%	$\mathbf{5.90 \times 10^{-4}}$	-71%
J5	24.2×10^{-4}	23.5×10^{-4}	-2.7%	$\mathbf{20.1 \times 10^{-4}}$	-13%
J6	29.0×10^{-4}	$\mathbf{15.4 \times 10^{-4}}$	-48%	$\mathbf{15.4 \times 10^{-4}}$	-0%
J7	21.4×10^{-4}	$\mathbf{5.40 \times 10^{-4}}$	-75%	$\mathbf{5.40 \times 10^{-4}}$	-0%

Table 6.2: Task space predictions.
deep-ProMp-cVAE is the most accurate model.

Task	deep-ProMP -AE	deep-ProMP -VAE	drop	deep-ProMP -cVAE	drop
X	1.40×10^{-4}	$\mathbf{1.40 \times 10^{-4}}$	-0%	$\mathbf{1.40 \times 10^{-4}}$	-0%
Y	15.31×10^{-4}	7.05×10^{-4}	-53%	$\mathbf{0.88 \times 10^{-4}}$	-88%
Z	0.92×10^{-4}	$\mathbf{0.12 \times 10^{-4}}$	-88%	$\mathbf{0.12 \times 10^{-4}}$	-0%
Q1	1.53×10^{-4}	$\mathbf{0.57 \times 10^{-4}}$	-68%	$\mathbf{0.57 \times 10^{-4}}$	-0%
Q2	24.2×10^{-4}	4.10×10^{-4}	-83%	$\mathbf{1.56 \times 10^{-4}}$	-75%
Q3	1.15×10^{-4}	0.37×10^{-4}	-80%	$\mathbf{0.21 \times 10^{-4}}$	-43%
Q4	0.78×10^{-4}	0.36×10^{-4}	-68%	$\mathbf{0.33 \times 10^{-4}}$	-8%

The error on orientation is defined as the minimum among the sum and the difference of the predicted and ground truth quaternions, since, according to [58], it is a parameter proportional to the distance between the two expressed orientations. A big value for this parameter means a big difference between the predicted and ground truth orientations, while a small value means a good prediction of the end effector orientation at the final point.

$$\begin{aligned} e_{position} &= \sqrt{(x-\hat{x})^2 + (y-\hat{y})^2 + (z-\hat{z})^2} \\ e_{orientation} &= min[\|q-\hat{q}\|, \|q+\hat{q}\|] \end{aligned} \tag{6.2}$$

In Eq. 6.2, (x, y, z) and $(\hat{x}, \hat{y}, \hat{z})$, and q and $\hat{q}$ represent the ground truth and predicted position and orientation (quaternions) at the final time step of the end effector, respectively. The trajectories that have been evaluated are the predicted mean trajectory and the one at a distance of 2σ from the mean. Table 6.3 illustrates that **task space predictions produce much more accurate performances at the final point**. Moreover, the most accurate model is the Deep-ProMP-cVAE.

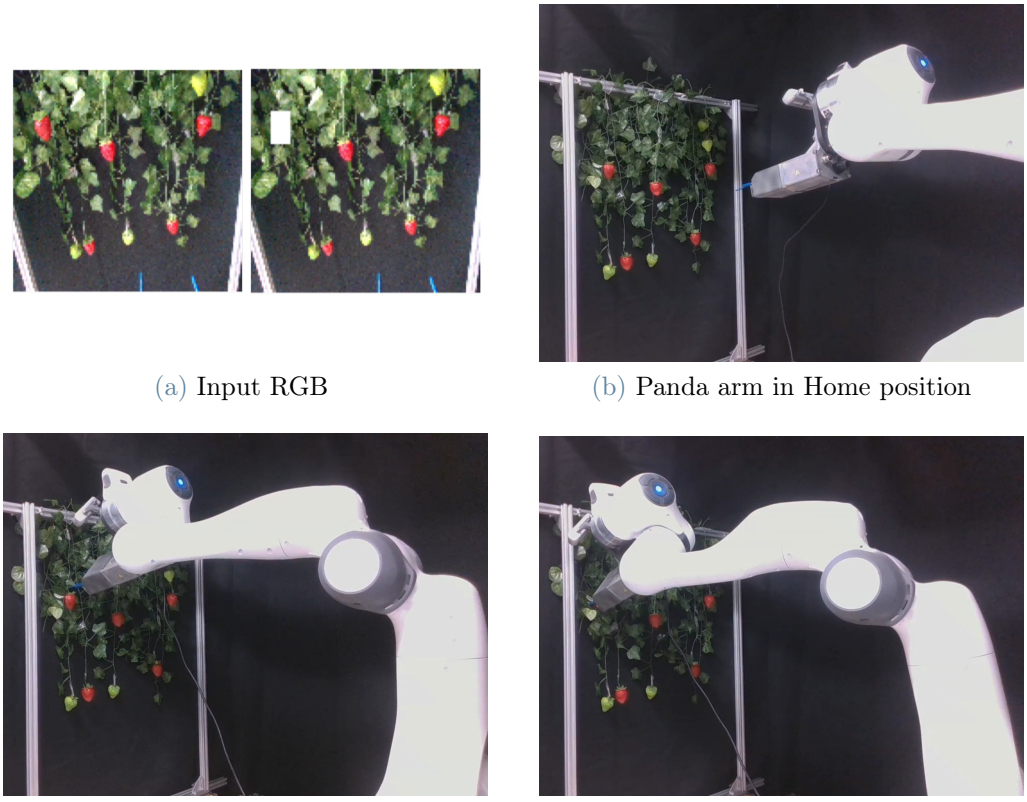
The third set of experiments has been performed implementing the models on the real robot instead of using a previously recorded dataset and testing the models' performances. 5 new (different) strawberry plant configurations are built, each including again 5 different target berries. This helps to demonstrate the generalisation ability of the models in predicting the reaching movement in unseen real-robot settings. The final position of the target strawberry has been used as a reference to evaluate the prediction performances. To capture this information the robot has been moved at first (kinesthetic teaching mode) to the desired final pose necessary for picking a target berry and the (x, y, z) position has been recorded. Deep-ProMP-AE, Deep-ProMP-VAE, Deep-ProMP-cVAE have been

Table 6.3: Joint Space vs Task Space prediction accuracy at the final point. Task space predictions are more accurate.

			Deep-ProMP -Direct	Deep-ProMP -AE	Deep-ProMP -VAE	Deep-ProMP -cVAE
JOINT	mean	pos	0.15	0.10	0.06	**0.04**
		ori	0.10	0.06	0.03	**0.02**
	2σ	pos	0.31	0.32	0.19	**0.18**
		ori	0.41	0.41	0.31	**0.28**
TASK	mean	pos	0.08	0.04	0.03	**0.02**
		ori	0.06	0.04	0.03	**0.02**
	2σ	pos	0.10	0.09	0.07	**0.06**
		ori	0.10	0.07	**0.06**	**0.06**

tested before and after the domain-specific latent space training. The predicted mean trajectory together with the trajectory at 2σ and at -2σ from the mean have been evaluated.

Eq.6.2 has been used as the metric to evaluate the position error at the final reaching



(a) Input RGB (b) Panda arm in Home position

(c) Deep-ProMP-AE (d) Deep-ProMP-cVAE-l

Figure 6.11: Test on real robot.

point.

In Fig. 6.11 the images of one test on the real robot are shown. In Fig. 6.11a the input RGB image before and after the marking with the white bounding box is shown. In Fig. 6.11b the robotic arm in Home position is captured while in Fig. 6.11c and Fig. 6.11d the robotic arm is captured at the final time instant when the predictions are made respectively with Deep-ProMP-AE and l-Deep-ProMP-cVAE (the mean trajectory was sampled from the predicted distributions).

Fig. 6.12 shows that **the model performances increase after the domain-specific training**. Furthermore, the most accurate model is l-Deep-ProMP-cVAE. The mean predicted trajectory performance is always better than the trajectories sampled at some σ from the mean.

The probabilistic framework can be exploited to perform the task in different ways sampling from the predictions of Deep-ProMP. For example, if the predictions are made in task space and some variability in the orientation of the end effector at the final point is desired, it can be simply achieved by sampling different trajectories from the predicted quaternion distribution (Fig. 6.13).

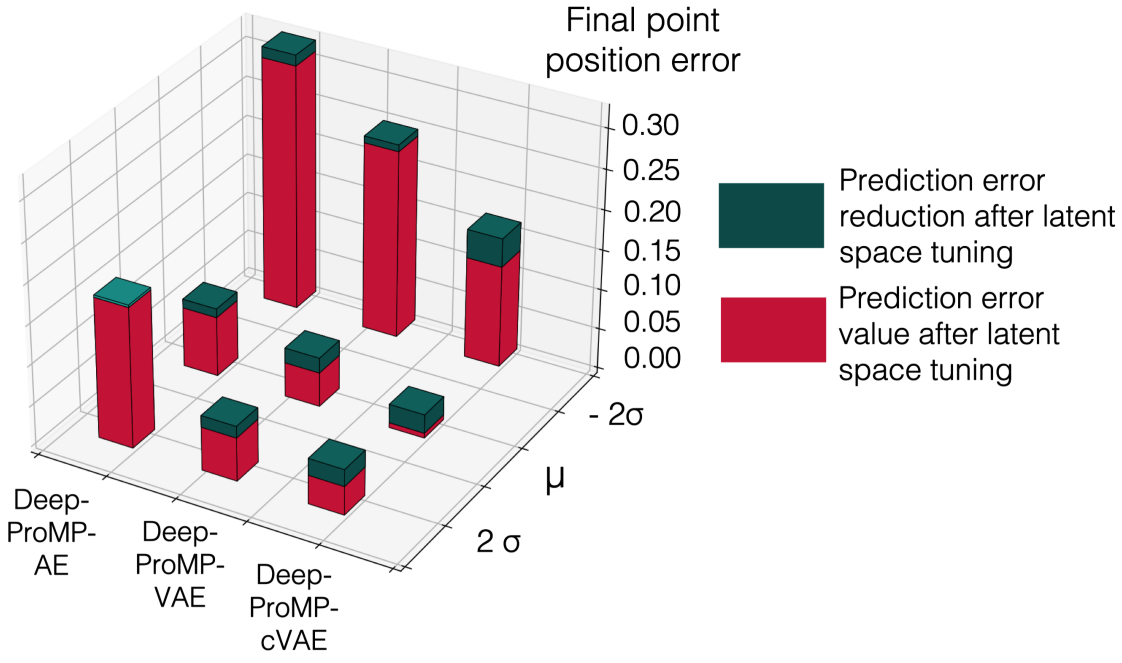


Figure 6.12: Experimental results before and after latent space tuning.

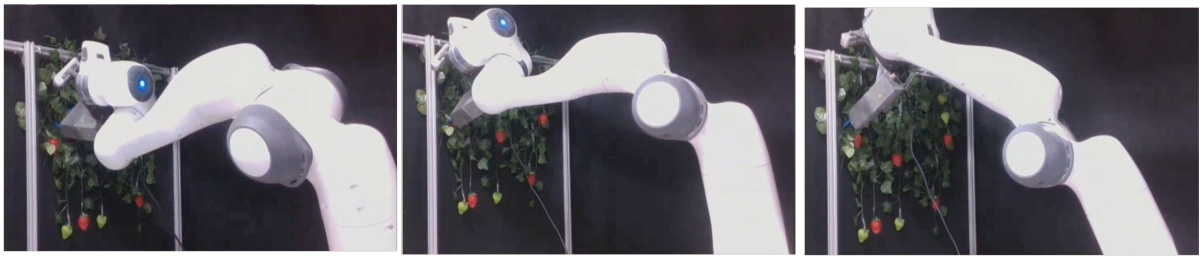


Figure 6.13: Reach-to-pick with different orientations.

The last type of analysis carried out is specific on the latent space morphology. The **clustering level of the latent space** before and after the domain-specific training has been explored. Fig. 6.14 visualises the 2-D representation of latent spaces of AE or VAE representing the input images (with a 256-d latent vector) before and after the domain-specific training. The T-distributed Stochastic Neighbour Embedding [17] has been used to reduce the dimensionality of the latent vectors from 256 to 2 to visualise latent spaces. It is a statistical method for visualizing high-dimensional data by giving each datapoint a location in a two or three-dimensional map. Specifically, it models each high-dimensional object by a two- or three-dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points with high probability. The t-SNE algorithm comprises two main stages. First, t-SNE constructs a probability distribution over pairs of high-dimensional objects in such a way that similar objects are assigned a higher probability while dissimilar points are assigned a lower probability. Second, t-SNE defines a similar probability distribution over the points in the low-dimensional map, and it minimizes the Kullback–Leibler divergence (KL divergence) between the two distributions with respect to the locations of the points in the map. Every point the representation in Fig. 6.14 represents an image taken from the dataset of 250 RGB images used for models training and testing.

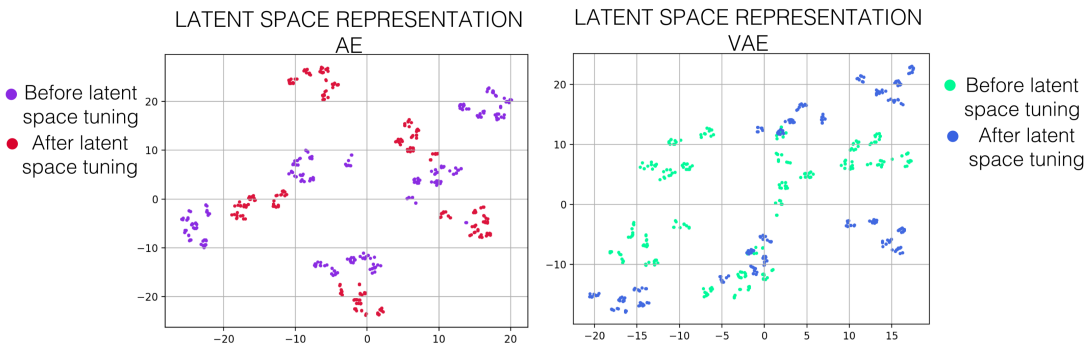


Figure 6.14: Latent space visualisation/representation through T-SNE embedding before and after domain specific tuning. After tuning, the clustering in the embedding space increases in both AE and VAE.

Table 6.4: Cluster separability of latent space representation through T-SNE using Davies-Bouldin Index [28] before and after latent space tuning with. A lower score indicates better clustering.

		Davies-Bouldin score
AE	base	0.616
	latent space tuned	**0.482**
VAE	base	0.424
	latent space tuned	**0.352**

Table 6.4 also shows the Davies-Bouldin [28] score used to evaluate the clustering level of the latent space with a numeric index. The score indicates the average similarity measure of each cluster with its most similar cluster, where similarity is the ratio of within-cluster distances to between-cluster distances. Thus, the lower the scores, the higher the level of clustering. This means that **the clustering level increases from AE to the VAE**. Moreover, **the domain-specific latent space learning further increases clustering level**.

A more clustered latent space is index of better representation of the input data in the embedding space. The regularity that is expected from the latent space can be expressed through two main properties: continuity (two close points in the latent space should not give two completely different contents once decoded) and completeness (for a chosen distribution, a point sampled from the latent space should give "meaningful" content once decoded). A more clustered latent space is more continuous and complete, meaning that the latent space has no points or regions which have no meaning, but it completely represents the distribution of the input data.

7 | Conclusions and future developments

This chapter summaries the major achievements of the research and suggests its further possible developments.

The worldwide demand for agricultural products is rapidly increasing due to the constantly growing population. However, rural human labor shortage due to different factors is becoming a limiting factor for agricultural production. Moreover, the recent Covid-19 pandemic has shown how possible travel restrictions can limit the affluence of seasonal farmworkers to the countries which rely on human power to pick rapidly ripening fruits and vegetables during harvesting season. All these aspects underline that the agricultural chain is strongly dependent on human labor, which turns out to be very risky in the current era.

Automation of agricultural activities can be an important solution to tackle these challenges and, as such, in the last few years private and public sectors have invested across the globe to develop and commercialise robotic fruit and vegetables harvesting technologies. Strawberry production is heavily reliant on human labor and despite several attempts to develop a robotic solution for harvesting strawberries and many other crops, a fully viable commercial system has yet to be established [122].

This thesis provides a solution for some main problems that need to be solved to develop a successful robotic technology for selective harvesting of strawberries, which are: ready-to-be-picked strawberries detection, key-points perception for picking and grasping actions, strawberries' weight estimation before picking, and path planning from visual information to reach the target fruit with the robotic hand.

The first two problems have been addressed with Detectron-2 [133], a next-generation open-source object detection system from Facebook AI Research based on Mask Region Convolutional Neural Network (Mask R-CNN) [52]. In particular, it has been trained to **segment berries, classify them as ripe or unripe and detect the key-points necessary for picking and grasping action** (Ch. 3). Moreover, two new datasets

useful for selective harvesting of strawberries (which include the strawberries RGB-D images under farm conditions from different perspectives, together with the annotations of key-points, instance segmentation, dimensions, and weights) have been presented and published. The results of the experiments show the effectiveness of the proposed approach and datasets to localize strawberries and key-points for selective harvesting regardless of the chaotic configurations which cause obstructions to ripe and harvest-ready strawberries .

Strawberry weight estimation of the fruits while on plant can help sort the fruits in the correct punnet right after picking. It has been achieved by implementing a Random Forest Model [20] with Decision Trees [70, 92] which takes as input a vector including features extracted from RGB and depth data of the berries whose weight has to be estimated (Ch. 4). This approach outperforms many state-of-the-art methods such as last generation EfficientNets or point cloud and graph-based neural networks.

Finally, a novel probabilistic framework, called **deep Probabilistic Movement Primitives (deep-ProMP)**, which maps the visual information of a robot workspace into the corresponding robot trajectories according to a set of human expert demonstrations, has been presented. The aim here is to correctly predict a distribution of trajectories effective in reaching the target ripe fruit using as input only the RGB information of the surrounding environment. A few model architectures have been presented, namely, Deep-ProMP-AE, Deep-ProMP-VAE, and Deep-ProMP-cVAE which all have a two-fold design: from the input image to latent representation and from latent representation to the desired trajectory. This architecture design has been compared against the direct mapping from RGB image to the trajectory space, (represented by Deep-ProMP-Direct) to show its superiority. The probabilistic nature of Deep-ProMP can be exploited in a series of real robot tests to reach the target point with different orientations. To further improve the performance of Deep-ProMP, a novel domain-specific latent space training has been proposed. This allows learning latent space representations of the input images in a way that is relevant both for computer vision and the specific robotic task. The results suggest that the deep-ProMP conditioning with a relevant feature (in this case the center of the bounding box of the target berry in pixel space) and domain-specific training of the latent space yields the best performances. Indeed, l-Deep-ProMP-cVAE outperforms other models with a relatively low error value suitable for real robot tasks.

Future works for sure include the testing of the approach in the real field. Moreover, the probabilistic framework can be exploited to sample from the deep-ProMP predicted distribution to optimise secondary objectives, e.g. avoid collisions at the reach point, eventually in a Reinforcement Learning setting.

Bibliography

[1] University of lincoln. https://www.lincoln.ac.uk.

[2] Article: Coronavirus triggers acute farm labour shortages in europe. https://ihsmarkit.com/research-analysis/article-coronavirus-triggers-acute-farm-labour-shortages-europe.html, 09.04.2020, Accessed 01.03.2022.

[3] Harvesting robot market share 2021 comprehensive growth, industry size-share, global trends, key players strategies, upcoming demand, business opportunities, revenue, gross margin and forecast 2030. https://znewsafrica.com/news/153047/harvesting-robot-market-share-2021-comprehensive-growth-industry-size/par-share-global-trends-key-players-strategies-upcoming-demand-business/par-opportunities-revenue-gross-margin-and-forecast-2030/, 23.02.2022,Accessed 01.03.2022.

[4] Dyson farming and state of the art strawberry glasshouse. https://dysonfarming.com/strawberries/, Accessed 01.03.2022.

[5] Strawberries in 2020 are a tale of supply and demand, not pandemic. https://www.producebluebook.com/2020/11/25/strawberries-in-2020-are-a-tale-of-supply-and-demand-not-pandemic, Accessed 01.03.2022.

[6] Agricultural robots market. https://www.mordorintelligence.com/industry-reports/agricultural-robots-market, Accessed 01.03.2022.

[7] Sweeper: Sweet pepper harvesting robot. https://cordis.europa.eu/project/id/644313, Accessed 01.03.2022.

[8] Detection evaluation. https://cocodataset.org/#detection-eval, Accessed 05.03.2022.

[9] P. Abbeel, A. Coates, and A. Y. Ng. Autonomous helicopter aerobatics through

apprenticeship learning. *The International Journal of Robotics Research*, 29(13): 1608–1639, 2010.

[10] M. Afonso, H. Fonteijn, F. Schadeck Fiorentin, D. Lensink, M. Mooij, N. Faber, G. Polder, and R. Wehrens. Tomato fruit detection and counting in greenhouses using deep learning. *Frontiers in Plant Science*, 11:571299, 11 2020. doi: 10.3389/fpls.2020.571299.

[11] B. Akgun, M. Cakmak, K. Jiang, and A. L. Thomaz. Keyframe-based learning from demonstration. *International Journal of Social Robotics*, 4(4):343–355, 2012.

[12] H. Altaheri, M. Alsulaiman, M. Faisal, and G. Muhammed. Date fruit dataset for automated harvesting and visual yield estimation, 2019. URL `https://dx.doi.org/10.21227/x46j-sk98`.

[13] J. An and S. Cho. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2(1):1–18, 2015.

[14] A. Arefi, A. M. Motlagh, K. Mollazade, R. F. Teimourlou, et al. Recognition and localization of ripen tomato based on machine vision. *Australian Journal of Crop Science*, 5(10):1144, 2011.

[15] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[16] P. Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49. JMLR Workshop and Conference Proceedings, 2012.

[17] A. C. Belkina, C. O. Ciccolella, R. Anno, R. Halpert, J. Spidlen, and J. E. Snyder-Cappione. Automated optimized parameters for t-distributed stochastic neighbor embedding improve visualization and analysis of large datasets. *Nature communications*, 10(1):1–12, 2019.

[18] E. A. Billing and T. Hellström. A formalism for learning from demonstration*. *Paladyn, Journal of Behavioral Robotics*, 1(1):1–13, 2010.

[19] C. M. Bishop. Linear models for regression. In M. Jordan, J. Kleinberg, and B. Schölkopf, editors, *Pattern recognition and machine learning*, chapter 3, pages 137–147. springer, New York, 2006.

[20] L. Breiman. Random forests. In *Machine Learning 45(1), 5-32*, 2001.

[21] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh. Openpose: realtime multi-person 2d pose estimation using part affinity fields. *IEEE transactions on pattern analysis and machine intelligence*, 43(1):172–186, 2019.

[22] S. B. Chaabane, M. Sayadi, F. Fnaiech, and E. Brassart. Color image segmentation using automatic thresholding and the fuzzy c-means techniques. In *MELECON 2008-The 14th IEEE Mediterranean Electrotechnical Conference*, pages 857–861. IEEE, 2008.

[23] H.-D. Cheng, X. Jiang, and J. Wang. Color image segmentation based on homogram thresholding and region merging. *Pattern recognition*, 35(2):373–393, 2002.

[24] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[25] Z. Cui, B. Zhang, C. Lian, C. Li, L. Yang, W. Wang, M. Zhu, and D. Shen. Hierarchical morphology-guided tooth instance segmentation from cbct images. In *International Conference on Information Processing in Medical Imaging*, pages 150–162. Springer, 2021.

[26] N. T. Dang, M.-T. Vo, T.-D. Nguyen, and S. V. Dao. Analysis on mangoes weight estimation problem using neural network. In *2019 19th International Symposium on Communications and Information Technologies (ISCIT)*, pages 559–562, 2019. doi: 10.1109/ISCIT.2019.8905118.

[27] M. Das Gupta. World bank res obs-2014-das gupta-population, poverty, and climate change, 07 2016.

[28] D. L. Davies and D. W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227, 1979. doi: 10.1109/TPAMI.1979.4766909.

[29] T. Duckett, S. Pearson, S. Blackmore, B. Grieve, P. Wilson, H. Gill, A. Hunter, and I. Georgilas. Agricultural robotics: the future of robotic agriculture. "UK-RAS White Papers", 2018.

[30] T. Duckett, S. Pearson, S. Blackmore, B. Grieve, P. Wilson, H. Gill, Hunter, A. J., and I. Georgilas. Bachus : Mobile robotic platforms for active inspection and harvesting in agricultural areas. `https://cordis.europa.eu/project/id/871704`, Accessed 01.03.2022.

[31] A. Durand-Petiteville, S. Vougioukas, and D. C. Slaughter. Real-time segmentation

of strawberry flesh and calyx from images of singulated strawberries during postharvest processing. *Computers and electronics in agriculture*, 142:298–313, 2017.

[32] A. Durand-Petiteville, D. Sadowski, and S. Vougioukas. A strawberry database: Geometric properties, images and 3d scans. `https://datadryad.org/stash/dataset/doi:10.25338/B8V308`, 2018.

[33] B. a. D.Zhang, D.-J.Lee. Date maturity and quality evaluation using color distribution analysis and back projection. *J. Food Eng.*, 131:161–169, 2014.

[34] S. Elliott, R. Toris, and M. Cakmak. Efficient programming of manipulation tasks by demonstration and adaptation. In *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 1146–1153. IEEE, 2017.

[35] M. Faisal, M. Alsulaiman, M. Arafah, and M. A. Mekhtiche. Ihds: Intelligent harvesting decision system for date fruit based on maturity stage using deep learning and computer vision. *IEEE Access*, 8:167985–167997, 2020. doi: 10.1109/ACCESS.2020.3023894.

[36] M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[37] C. Finn and S. Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793. IEEE, 2017.

[38] S. Fleming. Eu must prepare for 'era of pandemics', von der leyen says. `https://www.ft.com/content/fba558ff-94a5-4c6c-b848-c8fd91b13c16`, 2021.

[39] P. Ganesh, K. Volle, T. Burks, and S. Mehta. Deep orange: Mask r-cnn based orange detection and segmentation. *IFAC-PapersOnLine*, 52(30):70–75, 2019. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2019.12.499. URL `https://www.sciencedirect.com/science/article/pii/S2405896319324152`. 6th IFAC Conference on Sensing, Control and Automation Technologies for Agriculture AGRICONTROL 2019.

[40] P. Ganesh, K. Volle, T. Burks, and S. Mehta. Deep orange: Mask r-cnn based orange detection and segmentation. *IFAC-PapersOnLine*, 52(30):70–75, 2019.

[41] Z. Gao, Y. Shao, G. Xuan, Y. Wang, Y. Liu, and X. Han. Real-time hyperspec-

tral imaging for the in-field estimation of strawberry ripeness with deep learning. *Artificial Intelligence in Agriculture*, 2020.

[42] Y. Ge, Y. Xiong, G. L. Tenorio, and P. J. From. Fruit localization and environment perception for strawberry harvesting robots. *IEEE Access*, 7:147642–147652, 2019.

[43] H. Girgin, E. Pignat, N. Jaquier, and S. Calinon. Active improvement of control policies with bayesian gaussian mixture model. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5395–5401. IEEE, 2020.

[44] R. Girshick. Fast r-cnn, 2015.

[45] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014.

[46] P. R. Gokul, S. Raj, and P. Suriyamoorthi. Estimation of volume and maturity of sweet lime fruit using image processing algorithm. In *2015 International Conference on Communications and Signal Processing (ICCSP)*, pages 1227–1229, 2015. doi: 10.1109/ICCSP.2015.7322703.

[47] R. Harrabi and E. B. Braiek. Color image segmentation using automatic thresholding techniques. In *Eighth International Multi-Conference on Systems, Signals Devices*, pages 1–6, 2011. doi: 10.1109/SSD.2011.5993569.

[48] S. Hayashi, K. Shigematsu, S. Yamamoto, K. Kobayashi, Y. Kohno, J. Kamata, and M. Kurita. Evaluation of a strawberry-harvesting robot in a field test. *Biosystems engineering*, 105(2):160–171, 2010.

[49] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

[50] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[51] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. pages 770–778, 06 2016. doi: 10.1109/CVPR.2016.90.

[52] K. He, G. Gkioxari, P. Dollar, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[53] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[54] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[55] G. E. Hinton and R. S. Zemel. Autoencoders, minimum description length and helmholtz free energy. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, page 3–10, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[56] T. K. Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282 vol.1, 1995. doi: 10.1109/ICDAR.1995.598994.

[57] Z. Huang, S. Wane, and S. Parsons. Towards automated strawberry harvesting: Identifying the picking point. In *Annual Conference Towards Autonomous Robotic Systems*, pages 222–236. Springer, 2017.

[58] D. Huynh. Metrics for 3d rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35:155–164, 10 2009. doi: 10.1007/s10851-009-0161-2.

[59] T. Jadhav, K. Singh, and A. Abhyankar. Volumetric estimation using 3d reconstruction method for grading of fruits. *Multimedia Tools and Applications*, 78, 01 2019. doi: 10.1007/s11042-018-6271-3.

[60] A. Kalantar, Y. Edan, A. Gur, and I. Klapp. A deep learning system for single and overall weight estimation of melons using unmanned aerial vehicle images. *Computers and Electronics in Agriculture*, 178:105748, 11 2020. doi: 10.1016/j.compag.2020.105748.

[61] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[62] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[63] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[64] V. Kosaraju, A. Sadeghian, R. Martín-Martín, I. Reid, H. Rezatofighi, and S. Savarese. Social-bigat: Multimodal trajectory forecasting using bicycle-gan and graph attention networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[65] L. Ku. How automation is transforming the farming industry. https://www.plugandplaytechcenter.com/resources/how-automation-transforming-farming-industry/, 06.10.2021.

[66] N. Lamb and M. C. Chuah. A strawberry detection system using convolutional neural networks. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 2515–2520. IEEE, 2018.

[67] E. H. Land. The retinex. *American Scientist*, 52(2):247–264, 1964.

[68] C. Lehnert, A. English, C. McCool, A. W. Tow, and T. Perez. Autonomous sweet pepper harvesting for protected cropping systems. *IEEE Robotics and Automation Letters*, 2(2):872–879, 2017. doi: 10.1109/LRA.2017.2655622.

[69] C. Lehnert, A. English, C. McCool, A. W. Tow, and T. Perez. Autonomous sweet pepper harvesting for protected cropping systems. *IEEE Robotics and Automation Letters*, 2(2):872–879, 2017.

[70] F. P. Lepetit, V. Keypoint recognition using randomized trees. In *IEEE Trans. Pattern Anal. Mach. Intell. 28(9), 1465– 1479*, 2006.

[71] S. Levine, Z. Popovic, and V. Koltun. Nonlinear inverse reinforcement learning with gaussian processes. *Advances in neural information processing systems*, 24:19–27, 2011.

[72] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[73] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

[74] J. Li, Y. Tang, X. Zou, G. Lin, and H. Wang. Detection of fruit-bearing branches and localization of litchi clusters for vision-based harvesting robots. *IEEE Access*, 8:117746–117758, 2020. doi: 10.1109/ACCESS.2020.3005386.

[75] J. Li, Y. Tang, X. Zou, G. Lin, and H. Wang. Detection of fruit-bearing branches and localization of litchi clusters for vision-based harvesting robots. *IEEE Access*, 8:117746–117758, 2020.

[76] G. Lin, Y. Tang, X. Zou, J. Cheng, and J. Xiong. Fruit detection in natural envi-

ronment using partial shape matching and probabilistic hough transform. *Precision Agriculture*, 21:160–177, 2019.

[77] T.-Y. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2015.

[78] X. Liu, S. W. Chen, S. Aditya, N. Sivakumar, S. Dcunha, C. Qu, C. J. Taylor, J. Das, and V. Kumar. Robust fruit counting: Combining deep learning, tracking, and structure from motion. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1045–1052. IEEE, 2018.

[79] X. Liu, D. Zhao, W. Jia, W. Ji, C. Ruan, and Y. Sun. Cucumber fruits detection in greenhouses based on instance segmentation. *IEEE Access*, 7:139635–139642, 2019.

[80] Y.-P. Liu, C.-H. Yang, H. Ling, S. Mabu, and T. Kuremoto. A visual system of citrus picking robot using convolutional neural networks. In *2018 5th international conference on systems and informatics (ICSAI)*, pages 344–349. IEEE, 2018.

[81] G. B. Margolis, T. Chen, K. Paigwar, X. Fu, D. Kim, S. Kim, and P. Agrawal. Learning to jump from pixels. *arXiv preprint arXiv:2110.15344*, 2021.

[82] S. Mghames, M. Hanheide, and A. Ghalamzan. Interactive movement primitives: Planning to push occluding pieces for fruit picking, 06 2020.

[83] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[84] M. Mousavi and V. Surya Prasath. On the feasibility of estimating fruits weights using depth sensors. In *4 th International Congress of Developing Agriculture, Natural Resources, Environment and Tourism of IranAt: Tabriz Islamic Art University In cooperation with Shiraz University and Yasouj University, Iran*, 2019.

[85] K. Nagahama and K. Yamazaki. Learning from demonstration based on a mechanism to utilize an object's invisibility. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6120–6127. IEEE, 2019.

[86] M. NAGATA, P. M. BATO, M. MITARAI, Q. CAO, and T. KITAHARA. Study on sorting system for strawberry using machine vision (part 1). *Journal of the Japanese Society of Agricultural Machinery*, 62(1):100–110, 2000. doi: 10.11357/jsam1937.62.100.

[87] Q. C. M. M. T. F. Nagata, M. and O. Kinoshita. Study on grade judgment of fruit and vegetables using machine vision (part 2). *Journal of SHITA*, 8(4):140–145, 1996.

[88] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation. In *European conference on computer vision*, pages 483–499. Springer, 2016.

[89] F. Nielsen and R. Nock. On region merging: the statistical soundness of fast sorting, with applications. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 2, pages II–19, 2003. doi: 10.1109/CVPR.2003.1211447.

[90] M. Omid, M. Khojastehnazhand, and A. Tabatabaeefar. Estimating volume and mass of citrus fruits by image processing technique. *Journal of Food Engineering*, 100(2):315–321, 2010. ISSN 0260-8774. doi: https://doi.org/10.1016/j.jfoodeng.2010.04.015. URL `https://www.sciencedirect.com/science/article/pii/S0260877410002001`.

[91] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979. doi: 10.1109/TSMC.1979.4310076.

[92] F. P. L. V. Ozuysal, M. Fast keypoint recognition in ten lines of code. In *IEEE CVPR*, 2007.

[93] N. O'Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan, and J. Walsh. Deep learning vs. traditional computer vision. In *Science and Information Conference*, pages 128–144. Springer, 2019.

[94] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann. Probabilistic movement primitives. *Advances in Neural Information Processing Systems*, 26, 2013.

[95] A. Paraschos, C. Daniel, J. Peters, and G. Neumann. Using probabilistic movement primitives in robotics. *Autonomous Robots*, 42(3):529–551, 2018.

[96] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/`

9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[97] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[98] I. Pérez-Borrero, D. Marín-Santos, M. E. Gegúndez-Arias, and E. Cortés-Ancos. A fast and accurate deep learning method for strawberry instance segmentation. *Computers and Electronics in Agriculture*, 178:105736, 2020.

[99] A. Pervez, Y. Mao, and D. Lee. Learning deep movement primitives using convolutional neural networks. In *2017 IEEE-RAS 17th international conference on humanoid robotics (Humanoids)*, pages 191–197. IEEE, 2017.

[100] D. S. Prabha and J. S. Kumar. Assessment of banana fruit maturity by image processing technique. *Journal of Food Science and Technology*, 52:1316–1327, 2013.

[101] I. Pérez-Borrero, D. Marín-Santos, M. E. Gegúndez-Arias, and E. Cortés-Ancos. A fast and accurate deep learning method for strawberry instance segmentation. *Computers and Electronics in Agriculture*, 178:105736, 2020. ISSN 0168-1699. doi: https://doi.org/10.1016/j.compag.2020.105736. URL http://www.sciencedirect.com/science/article/pii/S0168169920300624.

[102] C. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. 2017.

[103] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[104] R. Rahmatizadeh, P. Abolghasemi, L. Bölöni, and S. Levine. Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3758–3765. IEEE, 2018.

[105] P. Rajendra, N. Kondo, K. Ninomiya, J. Kamata, M. Kurita, T. Shiigi, S. Hayashi, H. Yoshida, and Y. Kohno. Machine vision algorithm for robots to harvest strawberries in tabletop culture greenhouses. *Engineering in Agriculture, Environment and Food*, 2(1):24–30, 2009.

[106] M. Rana, M. Mukadam, S. R. Ahmadzadeh, S. Chernova, and B. Boots. Towards

robust skill generalization: Unifying learning from demonstration and motion planning. In *Intelligent robots and systems*, 2018.

[107] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard. Recent advances in robot learning from demonstration. *Annual Review of Control, Robotics, and Autonomous Systems*, 3(1):297–330, 2020. doi: 10.1146/annurev-control-100819-063206. URL `https://doi.org/10.1146/annurev-control-100819-063206`.

[108] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. 04 2018.

[109] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[110] B. Ridge, A. Gams, J. Morimoto, A. Ude, et al. Training of deep neural networks for the generation of dynamic movement primitives. *Neural Networks*, 127:121–131, 2020.

[111] J. B. T. M. Roerdink and A. Meijster. The watershed transform: Definitions, algorithms and parallelization strategies. *Fundam. Inf.*, 41(1–2):187–228, jan 2000. ISSN 0169-2968.

[112] M. Roser. Article:employment in agriculture. `https://ourworldindata.org/employment-in-agriculture`, Accessed 28.03.2022.

[113] E. Rueckert, J. Mundo, A. Paraschos, J. Peters, and G. Neumann. Extracting low-dimensional control variables for movement primitives. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1511–1518. IEEE, 2015.

[114] I. Sa, Z. Ge, F. Dayoub, B. Upcroft, T. Perez, and C. McCool. Deepfruits: A fruit detection system using deep neural networks. *Sensors*, 16(8):1222, 2016.

[115] O. Sanni, G. Bonvicini, M. A. Khan, P. Lopez, K. Nazari, and A. Ghalamzan. Deep movement primitives: towards breast cancer examination robot. In *36th AAAI Conference on AI*, page arxiv.org/abs/2202.09265. AAAI, 2022.

[116] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20:61–80, 2009.

[117] S. Schaal. Dynamic movement primitives–a framework for motor control in humans and humanoid robotics.

[118] S. Schaal. Learning from demonstration. NIPS'96, page 1040–1046, Cambridge, MA, USA, 1996. MIT Press.

[119] S. Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive motion of animals and machines*, pages 261–280. Springer, 2006.

[120] M. Schneider and W. Ertel. Robot learning by demonstration with local gaussian process regression. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 255–260. IEEE, 2010.

[121] A. Silwal, J. Davidson, M. Karkee, C. Mo, Q. Zhang, and K. Lewis. Design, integration, and field evaluation of a robotic apple harvester. *Journal of Field Robotics*, 34, 03 2017. doi: 10.1002/rob.21715.

[122] A. Silwal, J. R. Davidson, M. Karkee, C. Mo, Q. Zhang, and K. Lewis. Design, integration, and field evaluation of a robotic apple harvester. *Journal of Field Robotics*, 34(6):1140–1159, 2017.

[123] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015.

[124] M. Tan and Q. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[125] Y. Tang, M. Chen, C. Wang, L. Luo, J. Li, G. Lian, and X. Zou. Recognition and localization methods for vision-based fruit picking robots: A review. *Frontiers in Plant Science*, 11, 2020.

[126] Y. Tao and J. Zhou. Automatic apple recognition based on the fusion of color and 3d feature for robotic fruit picking. *Computers and electronics in agriculture*, 142: 388–396, 2017.

[127] S. Tian, F. Ebert, D. Jayaraman, M. Mudigonda, C. Finn, R. Calandra, and S. Levine. Manipulation by feel: Touch-based control with deep predictive models. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 818–824. IEEE, 2019.

[128] N. Wahlström, T. Schön, and M. Deisenroth. From pixels to torques: Policy learning with deep dynamical models. *arXiv preprint arXiv:1502.02251v3*, 2015.

[129] K. Wampler. Fast and reliable example-based mesh ik for stylized deformations. *ACM Trans. Graph.*, 35(6), nov 2016. ISSN 0730-0301. doi: 10.1145/2980179.2982433. URL `https://doi.org/10.1145/2980179.2982433`.

[130] W. Wang, Y. Huang, Y. Wang, and L. Wang. Generalized autoencoder: A neural network framework for dimensionality reduction. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 490–497, 2014.

[131] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019.

[132] W. Wu, Z. Qi, and L. Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9621–9630, 2019.

[133] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019.

[134] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[135] Y. Xiong, Y. Ge, L. Grimstad, and P. From. An autonomous strawberry-harvesting robot: Design, development, integration, and field evaluation. *Journal of Field Robotics*, 37, 08 2019. doi: 10.1002/rob.21889.

[136] Y. Xiong, C. Peng, L. Grimstad, P. J. From, and V. Isler. Development and field evaluation of a strawberry harvesting robot with a cable-driven gripper. *Computers and Electronics in Agriculture*, 157:392–402, 2019. ISSN 0168-1699. doi: https://doi.org/10.1016/j.compag.2019.01.009. URL `https://www.sciencedirect.com/science/article/pii/S0168169918312456`.

[137] Y. Xiong, C. Peng, L. Grimstad, P. J. From, and V. Isler. Development and field evaluation of a strawberry harvesting robot with a cable-driven gripper. *Computers and electronics in agriculture*, 157:392–402, 2019.

[138] K. Yamamoto, W. Guo, Y. Yoshioka, and S. Ninomiya. On plant detection of intact tomato fruits using image analysis and machine learning methods. *Sensors (Basel, Switzerland)*, 14:12191 – 12206, 2014.

[139] S. Yamamoto, S. Hayashi, H. Yoshida, and K. Kobayashi. Development of a stationary robotic strawberry harvester with a picking mechanism that approaches the target fruit from below. *Japan Agricultural Research Quarterly: JARQ*, 48(3): 261–269, 2014.

[140] Y. Yu, K. Zhang, L. Yang, and D. Zhang. Fruit detection for strawberry harvesting robot in non-structural environment based on mask-rcnn. *Computers and Electronics in Agriculture*, 163:104846, 2019.

[141] X. Zeng, Y. Miao, S. Ubaid, X. Gao, and S. Zhuang. Detection and classification of bruises of pears based on thermal images. *Postharvest Biology and Technology*, 161:111090, 2020.

[142] B. Zhang, N. Guo, J. Huang, B. Gu, and J. Zhou. Computer vision estimation of the volume and weight of apples by using 3d reconstruction and noncontact measuring methods. *Journal of Sensors*, 2020, 2020.

[143] L. Zhang, J. Jia, G. Gui, X. Hao, W. Gao, and M. Wang. Deep learning based improved classification system for designing tomato harvesting robot. *IEEE Access*, 6:67940–67950, 2018.

[144] Z. Zhang, P. Luo, C. C. Loy, and X. Tang. Facial landmark detection by deep multi-task learning. In *European conference on computer vision*, pages 94–108. Springer, 2014.

[145] Q.-Y. Zhou, J. Park, and V. Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

[146] R. Zhou, G. Yiming, F. He, and Z. Zhang. Multi-task learning from demonstration via embedded network. In *2017 24th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, pages 1–6, 2017. doi: 10.1109/M2VIP.2017.8211451.

[147] J.-Y. Zhu, R. Zhang, D. Pathak, T. Darrell, A. A. Efros, O. Wang, and E. Shechtman. Toward multimodal image-to-image translation. *Advances in neural information processing systems*, 30, 2017.

[148] J. Zhuang, C. Hou, Y. Tang, Y. He, Q. Guo, Z. Zhong, and S. Luo. Computer vision-based localisation of picking points for automatic litchi harvesting applications towards natural scenarios. *Biosystems Engineering*, 187:1–20, 2019.

A | Appendix A

A.1. Networks Architectures

In the next figures, the details per each model implemented in Ch. 5 are summarized.

Namely, both the Encoder and Decoder architectures for the Autoencoder (AE) and Variational Autoencoder (VAE) models are shown in A.1.1 and A.1.2 respectively.

In A.1.3 the architecture of the Multi-Layer Perceptrons (MLP) with and without conditioning is presented.

A.1.1. Autoencoder (AE)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
img (InputLayer)             [(None, 128, 128, 3)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 64, 64, 32)        2432
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 32)        0
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 8, 8, 64)          0
_________________________________________________________________
batch_normalization_1 (Batch (None, 8, 8, 64)          256
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 128)         73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 128)         0
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 2, 2, 128)         0
_________________________________________________________________
batch_normalization_2 (Batch (None, 2, 2, 128)         512
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
bottleneck (Dense)           (None, 256)               131328
=================================================================
Total params: 227,008
Trainable params: 226,560
Non-trainable params: 448
_________________________________________________________________
```

Figure A.1: AE-Encoder architecture.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
img (InputLayer)             [(None, 256)]             0
_________________________________________________________________
dense (Dense)                (None, 512)               131584
_________________________________________________________________
reshape (Reshape)            (None, 2, 2, 128)         0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 4, 4, 128)         147584
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 8, 8, 128)         0
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 8, 8, 128)         0
_________________________________________________________________
batch_normalization_3 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 16, 16, 64)        73792
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 32, 32, 64)        0
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 32, 32, 64)        0
_________________________________________________________________
batch_normalization_4 (Batch (None, 32, 32, 64)        256
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 64, 64, 32)        51232
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 64, 64, 32)        0
_________________________________________________________________
batch_normalization_5 (Batch (None, 64, 64, 32)        128
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 128, 128, 16)      12816
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 128, 128, 16)      0
_________________________________________________________________
batch_normalization_6 (Batch (None, 128, 128, 16)      64
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 128, 128, 3)       435
_________________________________________________________________
activation (Activation)      (None, 128, 128, 3)       0
=================================================================
Total params: 418,403
Trainable params: 417,923
Non-trainable params: 480
_________________________________________________________________
```

Figure A.2: AE-Decoder architecture.

A.1.2. Variational Autoencoder (VAE)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
img (InputLayer)             [(None, 128, 128, 3)      0
_________________________________________________________________
conv2d (Conv2D)              (None, 64, 64, 32)        2432
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 32)        0
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization (BatchNorma (None, 32, 32, 32)     128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2D) (None, 8, 8, 64)        0
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 8, 8, 64)          0
_________________________________________________________________
batch_normalization_1 (BatchNor (None, 8, 8, 64)       256
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 128)         73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2D) (None, 2, 2, 128)       0
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 2, 2, 128)         0
_________________________________________________________________
batch_normalization_2 (BatchNor (None, 2, 2, 128)      512
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
bottleneck_mean (Dense)      (None, 256)               131328
_________________________________________________________________
bottleneck_var (Dense)       (None, 256)               131328
_________________________________________________________________
sampling (Sampling)          (None, 256)               0

=================================================================
Total params: 358,336
Trainable params: 357,888
Non-trainable params: 448
_________________________________________________________________
```

Figure A.3: VAE-Encoder architecture.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
img (InputLayer)             [(None, 256)]             0
_________________________________________________________________
dense (Dense)                (None, 512)               131584
_________________________________________________________________
reshape (Reshape)            (None, 2, 2, 128)         0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 4, 4, 128)         147584
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 8, 8, 128)         0
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 8, 8, 128)         0
_________________________________________________________________
batch_normalization_3 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 16, 16, 64)        73792
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 32, 32, 64)        0
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 32, 32, 64)        0
_________________________________________________________________
batch_normalization_4 (Batch (None, 32, 32, 64)        256
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 64, 64, 32)        51232
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 64, 64, 32)        0
_________________________________________________________________
batch_normalization_5 (Batch (None, 64, 64, 32)        128
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 128, 128, 16)      12816
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 128, 128, 16)      0
_________________________________________________________________
batch_normalization_6 (Batch (None, 128, 128, 16)      64
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 128, 128, 3)       435
_________________________________________________________________
activation (Activation)      (None, 128, 128, 3)       0
=================================================================
Total params: 418,403
Trainable params: 417,923
Non-trainable params: 480
_________________________________________________________________
```

Figure A.4: VAE-Decoder architecture.

A.1.3. Multi-Layer Perceptron (MLP)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 256)]             0
_________________________________________________________________
dense (Dense)                (None, 512)               131584
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_3 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_7 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_5 (Dense)              (None, 16)                528
_________________________________________________________________
dense_8 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_6 (Dense)              (None, 8)                 136
_________________________________________________________________
dense_9 (Dense)              (None, 36)                2340
_________________________________________________________________
concatenate (Concatenate)    (None, 44)                0

=================================================================
Total params: 350,300
Trainable params: 350,300
Non-trainable params: 0
_________________________________________________________________
```

Figure A.5: MLP for singe joint or task space coordinate trajectory prediction without conditioning.

```
__________________________________________________________________
Layer (type)                 Output Shape              Param #
==================================================================
input_1 (InputLayer)         [(None, 256)]             0
__________________________________________________________________
dense_2 (Dense)              (None, 512)               131584
__________________________________________________________________
input_2 (InputLayer)         [(None, 1)]               0
__________________________________________________________________
input_3 (InputLayer)         [(None, 1)]               0
__________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 512)               0
__________________________________________________________________
dense (Dense)                (None, 1)                 2
__________________________________________________________________
dense_1 (Dense)              (None, 1)                 2
__________________________________________________________________
dense_3 (Dense)              (None, 256)               131328
__________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 1)                 0
__________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 1)                 0
__________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 256)               0
__________________________________________________________________
concatenate_1 (Concatenate)  (None, 258)               0


__________________________________________________________________
dense_4 (Dense)              (None, 128)               33152
__________________________________________________________________
dense_5 (Dense)              (None, 64)                8256
__________________________________________________________________
dense_6 (Dense)              (None, 32)                2080
__________________________________________________________________
dense_9 (Dense)              (None, 128)               33152
__________________________________________________________________
dense_7 (Dense)              (None, 16)                528
__________________________________________________________________
dense_10 (Dense)             (None, 64)                8256
__________________________________________________________________
dense_8 (Dense)              (None, 8)                 136
__________________________________________________________________
dense_11 (Dense)             (None, 36)                2340
__________________________________________________________________
concatenate (Concatenate)    (None, 44)                0

==================================================================
Total params: 350,816
Trainable params: 350,816
Non-trainable params: 0
__________________________________________________________________
```

Figure A.6: MLP for singe joint or task space coordinate trajectory prediction with conditioning.

List of Figures

List of Tables

Acknowledgements

Special thanks go to IntManLab, University of Lincoln, UK, for giving me the opportunity to work with a Franka Emika Panda robot. In particular, my gratitude goes towards prof. Amir for his patience and his precious advice. I would like also to thank all the members of the group for the forming experience and in particular Rick who supported me and gave me precious advice throughout the whole period.

My gratitude goes to Mattia and Giorgio who have been my Italian family during both hard and pleasant times in Lincoln.

The earnest acknowledgments also go towards prof. Zanchettin for the encouraging supervision and availability.

Finally, thanks to my family and friends for the constant support in all the decisions of my life.