Active Perception in Autonomous Fruit Harvesting

Viewpoint Optimization with Deep Reinforcement Learning

Dimitrios Klaoudatos

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Active Perception in Autonomous Fruit Harvesting

Viewpoint Optimization with Deep Reinforcement Learning

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Dimitrios Klaoudatos

March 16, 2021

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc), Cognitive Robotics
(CoR)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Active Perception in Autonomous Fruit Harvesting

by

Dimitrios Klaoudatos

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: March 16, 2021

Supervisor(s):

Prof.dr. R. Babuska

Reader(s):

dr.ing. R. Van de Plas

dr. P.M. Esfahani

dr. G.W. Kootstra

Summary

Fruit robotic harvesting is a promising solution to the increasing food demand and the labour shortage. However, the harvesting performance is limited by various factors. The occlusions of the fruits by leaves, branches, and other fruits are significant constraints for the harvesting tasks. This MSc thesis presents the development of a viewpoint optimization framework to face the problem of detecting occluded fruits in autonomous harvesting. This approach is chosen after studying in the literature about the advantages and the drawbacks of the fruit detection methods. A Deep Reinforcement Learning (DRL) algorithm is developed in order to train a robotic manipulator to navigate to occlusion-free viewpoints of the tomato-target.

A vision-based reward, which requires visual information, is formulated for the DRL algorithm. Two Convolutional Neural Network (CNN)s, You Only Look Once (YOLO)v3 and Mask Region-Convolutional Neural Network (Mask R-CNN), are examined, trained, and evaluated. YOLOv3 results in high Average Precision (AP), equal to 0.9394, and detection confidence scores, with an average value over 90%, even in major occlusions. However, the reward function requires explicit pixel information, so the instance segmentation Mask R-CNN is preferred over the object detection YOLOv3. The highest achieved AP in Mask R-CNN is 0.7403, and the network detects the tomatoes under occlusions and separates the adjacent tomatoes with a high confidence score. Only some small parts of the tomato masks can be missing, but this does not affect the visual reward effectiveness considerably. Moreover, the inpainting method and some extra image processing are implemented to obtain the total number of pixels of the occluded tomatoes. The fraction of the visible over the total pixels is used as the occlusion modelling metric and is closely related to the visual reward.

The DRL active perception algorithm is trained and evaluated in a simulation environment by mounting a camera on a robotic manipulator. The use of different reward scheme and exploration strategy during training shows their effect on training performance. The assessment of the training takes place by illustrating the maximum visible tomato-target fraction per episode, the steps needed per episode, the trajectory of the robot's end-effector, the initial and final tomato-target viewpoint in each episode. The evaluation results show a satisfactory performance, which depends on the reward and exploration strategy, as in the majority of the cases the robot can fully see the tomato after a few steps. Generally, the trained CNNs sur-

pass other fruit detection methods, and the DRL framework shows a promising performance that can be further improved with suitable modifications.

Table of Contents

Acknowledgements

I would like to thank my supervisor Professor R. Babuska for the chance that he gave me to continue my research in robotic harvesting and his advice during the MSc thesis project.

Also, I would like to thank my family and my friends for their support during my MSc studies.

Moreover, I would like to thank the Bodossaki Foundation for its financial support through the scholarship that it provided me for my MSc studies.

Finally, I would like to thank the Centre for Genetic Resources, the Netherlands (CGN) in Wageningen University and Research for the dataset of tomato images that provided to me for the conduct of experiments.

Delft, University of Technology
March 16, 2021

Dimitrios Klaoudatos

Chapter 1

Introduction

1-1 Problem Description

Nowadays, there is a huge need for food production due to the global population growth. Fruit harvesting is an important part of providing food to people. However, it is a difficult, repetitive task for the laborers working in the fields and greenhouses due to the aggravating body movements and the exposure to the sun. This tedious work can cause serious musculoskeletal and mental problems to the fruit pickers. Moreover, there is a significant shortage of laborers, as they are discouraged from working in these conditions. The automation of the harvesting process can give a solution to this situation. Furthermore, this can contribute to higher quality and quantity in production, which is vital for the increasing food demand.

Despite all these advantages that come from the automation of the harvesting task, the cost and performance indices of the already developed harvesting robotic technology do not yet allow a widespread commercial and profitable adoption of harvesting robots [1]. Such a robotic system executes operations that are related to the detection, localization, approach, grasping, and cutting of the fruit. All these tasks face some difficulties in their successful implementation by a robotic harvester. These limitations have to be handled to achieve efficient harvesting performance with the developed robotic system.

The harvesting performance is associated with the harvesting success rate, the quality of the harvested fruit, and the harvesting speed. The first two depend on many factors [2]. The first important factor is the correct detection and localization of the target fruit. This is related to the perception system, which must not be affected by the varying lighting conditions, by the occlusion of the target fruit by leaves, branches, and other non-target fruits, and by the variation in the fruit size and shape. After the perception of the surrounding environment, the success of the harvesting process depends on an efficient path planning of the robotic system from the start position to the target position. Then, a suitable type of gripper that grasps the fruit correctly, without pushing it away before and damaging it after grasping, is required [3]. The harvesting speed is dependent on the methods used for the mentioned sub-tasks, the image processing software, the number of trials per successful harvest, and the speed of the robotic system's hardware.

Furthermore, the harvesting performance is also affected by the environmental conditions. The high-level occlusion of the target fruit by other fruits, branches, and leaves is a serious challenge imposed by the environment. Occlusion makes fruit detection and localization difficult. Even state-of-the-art techniques, which perform very well in various computer vision benchmarks, do not give a high detection accuracy in the case of cluttered scenes. The development of detection and localization methods under severely occluded scenes is very challenging. Some researches adapted the cultivation in a way that dense occlusions are not present, such as making the fruits to hang free in an open structure [2] or using a robot that removes the leaves firstly [4]. These approaches can facilitate the autonomous harvesting job as the fruits are more approachable and differentiated from the leaves. However, this strategy is not always efficient, as it depends on the kind of crop to be harvested, and the final solution can be more complex and expensive.

Inadequate visual information is the main limitation created by the dense occlusions to the robotic harvester. This is the main problem that the thesis addresses. One solution to this problem, without adapting the cultivation to the convenience of the robots, is the active camera navigation to the optimal pose, which obtains an occlusion-free perspective of the fruit. The approach of my thesis to the mentioned problem is the development of a viewpoint optimization method. More specifically, a control policy that moves the camera to occlusion-free viewpoints for acquiring the fruit's scene will be developed using Deep Reinforcement Learning (DRL). In this way, many computer vision methods' limitations can be circumvented, and the viewpoint optimization method can be used in other researches before the fruit detection, localization, approach, and grasp, to increase the harvesting speed and accuracy, and generally the whole performance.

1-2 Related Work

1-2-1 Fruit Detection and Localization Sensors

Image acquisition is the first step in fruit detection and localization procedure. The optical techniques for image acquisition are divided into passive and active methods. The first methods use the reflection of the natural light on a given target to measure its shape. The second methods enhance the shape acquisition by using an external lighting source that provides additional information. The three main technologies of the 3D cameras are Time of Flight (ToF), structured-light sensors, and active stereoscopy sensors. ToF sensors measure depth by estimating the time delay from light emission to light detection. Structured-light sensors combine the projection of a light pattern with a standard 2D camera and measure depth through triangulation. Active stereoscopy sensors look for artificially projected features from multiple cameras to reconstruct a 3D shape using triangulation and epipolar geometry. An extensive review of the sensors used in fruit detection and localization, with their potentials and limitations, is presented in my literature survey [5].

Giancola et al. [6] examined twenty 3D cameras based on the abovementioned three main technologies plus the passive stereoscopy sensors, and they dealt with indoor metric evaluations. They showed that the uncertainty in the depth measurement of a ToF camera scales linearly with the depth. In this way, reliable measurements can be given in long ranges. On the other hand, sensors based on the triangulation principle, such as Orbbec Astra S, are

more suitable for short ranges, as the uncertainty in the depth measurement grows quadratically. Vit and Shani [7] combined four different low-cost RGB-D sensors and evaluated their performance for close-range outdoor agricultural phenotyping. Phenotyping is the task of measuring plant attributes for analyzing the current state of the plant. The evaluated sensors are the following: Microsoft Kinect 2, Orbbec Astra S, Intel SR300, and Intel D435. The Intel SR300 is based on the structured-light technology and is designed for short-range, while the Intel D435 is based on the infrared active stereoscopy technology. They assessed the sensors' performance by computing the fill rate, i.e. the portion of the missing depth information over a Region of Interest (RoI), by using the obtained images for training deep learning segmentation models for object identification, and by estimating the size of known objects. The various experiments showed that Intel D435 outperformed the others in all cases, while the others had good performance only in some categories according to the distance to the target, the lighting intensity, and the size of the target object.

1-2-2 Fruit Detection Methods

A suitable vision system in a fruit harvesting robot must include fruit detection and localization methods. The detection relates to the identification of a ripe fruit target without diseases and damages. The subsequent localization computes the coordinates of the target fruit in the three-dimensional space with respect to a chosen reference frame. Many techniques have been developed to identify a fruit under occlusion. Some of these methods include the image segmentation based on a colour model [3], [8], the feature detection [9], [10] , the Artificial Neural Network (ANN)s, and the Convolutional Neural Network (CNN)s [11], [12], [13].

Also, some researchers applied visual servoing [14] in order to detect and localize fruits under occlusions [15], [16] [17]. In this method, the pose of the end-effector relative to the target is controlled continuously by using the visual information for the target object and matching it with the object's geometry and image features. Researches related to the above-mentioned methods are presented extensively in my literature survey [5].

1-2-3 Viewpoint Optimization in Fruit Detection

In active perception, action and perception are connected through the collection of sensory data by a moving robot. The action of the robot causes either the change of its camera perspective or the manipulation of other objects. One application of active perception in autonomous harvesting is the finding of a clear view of fruit according to the visual information, but without using strict image patterns such as in visual servoing. In this way, the derived control policy by the viewpoint optimization approach can be smoother and easier to stabilize.

The goal of viewpoint optimization is to find the camera's position with the best fruit view among other positions. Lehnert et al. [18] presented a viewpoint optimization approach in harvesting robotics, referred to as 3D Move to See (3DMTS), which finds the next best view using a 3D camera array and a robotic manipulator to obtain multiple samples of the scene from different perspectives. This method uses semantic vision and an objective function applied to each perspective to sample a gradient representing the direction of the next best view. This approach can be considered as an alternative to the visual servoing method in

which a template image is known in advance. Moreover, they created the camera array in order to be able to capture multiple images of the same scene from slightly different positions without having to move the camera. A score is assigned to each view in order to form a gradient. Then, the camera moves in the gradient ascent direction that maximizes the objective function. In the end, this method managed to lead to a locally optimal view of the object of interest even in highly occluded scenes and unstructured environments. The authors recognized some developments referring to some parameters of the camera array and proposed the implementation of this method to dynamic environments where the target fruit can move.

Moreover, Zapotezny and Lehnert [19] extended the work in [18] in a deep learning framework by using a CNN. The CNN is trained using as input the reference image of the camera array, which is located in the center of this structure, and as output the gradient of the objective function, which shows the movement of the arm position. Then, the evaluation pipeline takes place by putting the reference image as input to the already trained network and having the update of the arm position as output. Thus, there is a need for only one camera during the evaluation of the algorithm in the harvesting process, but the need for the camera array remains for training purposes. Finally, they concluded that the performance of the deep learning 3DMTS method is almost the same as the standard 3DMTS method, as it increases the fruit size in the image by the same factor. Nonetheless, the deep learning method outweighs the baseline as it is more time-efficient due to the reduced hardware and data processing.

Cheng et al. [20] proposed learning manipulation policies under occlusions by coordinating active perception to keep the target object in the field of view of the camera and push it to the desired location. In this way, they applied active perception for action as opposed to the recognition that other researchers have done. In particular, they trained the hand-eye controller by using Reinforcement Learning (RL) and particularly the Deep Deterministic Policy Gradient (DDPG) algorithm [21]. Also, they exploited the input images of an RGB camera in a trained object detector, whose architecture is Region Based Convolutional Neural Network (R-CNN). The state consists of the frame appearance, the three-dimensional centroid of the object of interest, and the gripper location. The actions are the camera and the gripper movements. The reward consists of achieving a manipulation task, and it was found that a visibility reward for successful object detection cases did not improve the performance of the active vision policies for manipulation. They concluded that this combination of active vision and learning manipulation policies has better performance than other static camera methods in cluttered scenes. Finally, they identified that the absence of memory state estimation in their work is a limitation, as the object detectors do not have information on the visual scene in time and act independently at every frame.

Furthermore, Sather and Zhang [22] dealt with the viewpoint optimization problem in autonomous strawberry harvesting. They developed a DDPG algorithm, which exploits a pre-trained object detector You Only Look Once (YOLO)v2 that provides visual reward feedback. More specifically, they used the confidence level of the detected, with YOLOv2, strawberries as a reward. However, the statement that the high probability of ripe strawberries detection corresponds to favorable viewpoints is a weak assumption. Also, the reward is negative when the actions lead the robot to unreachable positions, and it has a small penalty when no ripe strawberries are detected. The robot position is given by two angles of the end-effector around the plant. The state space is the set of all reachable end-effector positions and all possible

camera images. The action space consists of updates on the two defined robot positions. They compared the developed policy with some random and heuristic policies. They found that the heuristic policy, which combines the learned policy from DDPG with some hard-code rules, had the largest average return and that the learned policy needed the fewest time steps till obtaining a reward. The state that they used faces problems in heavily occluded environments due to the partial observability at each timestep. For this reason, they proposed, such as the researchers in [20], that the agent could propagate the relevant state information through time. The information inclusion from previous states to the current state would be a solution, but this would be limited by the available memory. One other more difficult solution, but probably successful, would be to make the agent learn which information to keep and propagate through time.

1-3 Thesis Approach

This thesis deals with the problem of facing the occlusions around fruit, in order to have an accurate fruit detection and subsequently a high harvesting performance. After studying in literature the methods for fruit detection under occlusions and analyzing their advantages and bottlenecks, I ended up in the viewpoint optimization technique. In this way, the goal of this thesis is to develop a control policy, which leads the camera to positions where each fruit, in particular tomato, is fully seen or is shown almost complete. The approach is the use of a DRL algorithm, and more specifically the DDPG algorithm, in order to learn the active perception policies. A CNN is trained to detect tomatoes, and another CNN is trained to segment tomatoes in instances, as the visual information is important for the validity of the learned policy. The final CNN type is selected according to the characteristics, the training, and the evaluation of some networks like YOLOv3 and Mask Region-Convolutional Neural Network (Mask R-CNN). As regards the details of the RL framework, the pose of the robot's end-effector and the tomato-target Cartesian coordinates are the observations, while incremental updates in the end-effector's position and orientation are the actions. The output of the CNN, which is chosen finally, is exploited in the formulation of the reward function. The visible and occluded areas of each tomato are computed using the trained CNN and an occlusion reasoning model. These two types of areas are used in the reward formulation. Also, invalid poses of the robot's end-effector are considered penalties to the reward calculation. The proposed method is implemented in a Gazebo simulation environment using a KUKA robotic arm and a tomato plant. The derived policy is evaluated using different versions and positions of a tomato plant in order to check the generalization attribute of the developed method.

The contribution of this work is the combination of a DRL algorithm with a state-of-the-art instance segmentation CNN, which is Mask R-CNN, in order to find an unoccluded view of fruits in high occlusions. At the work [22], the assumption about exploiting the visual information for the reward formulation is not enough strong, and also the instance segmentation of my case provides more accurate information about a target fruit than an object detection, which outputs only a bounding box and not an extra segmentation mask. The work of Cheng et al. [20] coordinates the active perception with manipulation, which is not the case of the current project. Finally, Zapotezny and Lehnert [19] may also use the combination of DRL with CNN, but their CNN does not perform instance segmentation, and also its training is

based on a camera array created in a previous work [18], which requires time to make this hardware and also more data processing.

The thesis is organized in the following way: The Chapter 2 describes the theory around CNNs, and especially their basic architecture and some recent types, which are examined to be used for the obtainment of the visual information. Then, the Chapter 3 presents the RL concept and its extension to the DRL. Also, there is an explanation of the value-based and policy gradient methods and the actor-critic architecture, which the finally implemented algorithm DDPG exploits. Moreover, the Chapter 4 illustrates the setup and the methods that are going to be used and provides all the necessary implementation details. Then, the experimental results are shown in Chapter 5. Finally, the thesis ends with the conclusions and the proposals for future work.

Chapter 2

Perception

The perception capability is crucial for an agent to obtain information about its environment and then to process it and act accordingly. The field of computer vision has been developed significantly. Machine learning, especially deep learning, is a reason for this. The Convolutional Neural Network (CNN) is one important method that is widely used. These networks are suitable for the two-dimensional detection and segmentation of the target object, but they can not compute the position of the object in the three-dimensional space. The 3D localization methods can provide information about the distance of the target from the camera. The object detection, segmentation, and localization concepts can be implemented using suitable computer vision software tools.

This chapter presents some CNN architectures that are used in image processing for object detection and segmentation. The algorithms for the three-dimensional localization of an object are described in my already complete literature survey [5].

2-1 Convolutional Neural Network (CNN)

Many methods of image processing are mentioned in my literature survey [5]. The classical methods related to the image segmentation based on a colour model present many limitations due to the variation in colour, the background features, the lighting conditions, and the fruit clusters. The combination of some features, such as colour, texture, and shape, can face some of these challenges, but the existence of uniformly distributed features and the need for matching with a fruit pattern require a more general and robust to the fruit variations method. The image classification methods, like Artificial Neural Network (ANN) and CNN, use a sample of images in different conditions in order to be trained. The problem with the ANN is that the handling of an image in a pixel-wise lever requires a huge number of neurons in the input layer, which means more processing time and generally an ineffective system. The advantage of the CNN is the extraction of image features and their conversion in lower dimensions without losing their important characteristics.

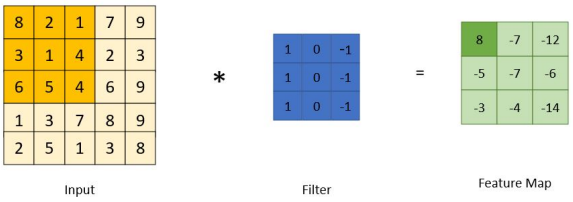


Figure 2-1: Convolution operation with stride=1

A CNN is structured in layers. The input layer has the input image, which is represented as a three-dimensional matrix. This layer is followed by a convolution layer that extracts features from an image by convolving the image with a kernel/filter. For this reason, it is also called feature extractor layer. More particularly, a part of the image is connected to the convolution layer to perform the convolution operation between this part, which has the same size as the filter, and the filter. Then the filter is moved to the next receptive field of the input image according to the stride size, and a convolution operation takes place again. This continues until the filter does the convolution operation with the whole area of the image. The output that is produced from the convolution of the input image with the feature detector is called convolved feature or feature map. The result of the convolution of one dimension of the image with a filter is shown in Figure 2-1. After the convolution, the input image's size has been reduced, depending on the stride length, so it becomes easier and faster to process it. As it is obvious there is some loss of information, but the position of the features to be detected remains recognizable, as there the featured map has a large value due to the match of the input image with the pattern that the filter represents. If there is the need to extract more than one feature from an image, then we use multiple filters, so there are multiple feature maps in the convolution layer. Images are non-linear themselves due to the different color and elements on these, and the convolution can make them linear. An activation function is used after the convolution in order to increase the non-linearity of the output. Common types of activation functions are ReLu and sigmoid functions.

The pooling layer is used in order to make the detected features more robust so that they can be detected even if the target object is presented in inconstant conditions, such as rotated, variable lighting conditions, etc. There are many different types of pooling, such as max, mean, and sum pooling, and all of them use a filter (F) and a stride (S) length. The achievement of robustness can be explained intuitively by considering a filter of huge size and a max pooling operation. In that case, the desired feature to be detected will be recognized even in the case that is rotated, as the rotated position can be included inside the filter and then there will be the same result after the max pooling operation. Also, pooling reduces more the size of the image and therefore of the parameters of the CNN, which prevents overfitting and speeds up the computation time. If the feature map given as an input to the pooling layer has dimension w x h, then the output of the pooling layer has dimensions w' x h', which are given by Eq. (2-1):

$$\begin{aligned} w' &= (w - F)/S + 1 \\ h' &= (h - F)/S + 1 \end{aligned} \tag{2-1}$$

A max pooling operation with a 2 x 2 filter and a stride of 2 is illustrated in Figure 2-2.

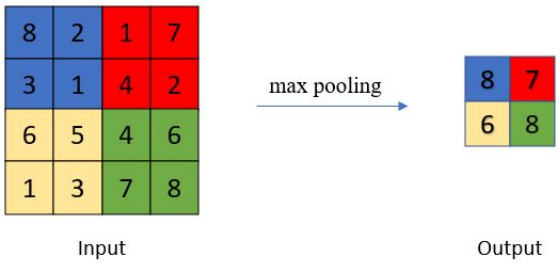


Figure 2-2: Max pooling operation

Figure 2-3: CNN layers

The step following pooling is the flattening, where the pooled feature map is put in one column vector.

A Fully Connected (FC) layer is at the final stage of a CNN. The vector created from the flattening is given as input to a new ANN. The ANN is used in order to combine the features into attributes that predict the output values better, as the learned high-level features are exploited in the FC layers. The FC layer is a hidden layer in the ANN, but it is fully connected with the previous and the next layers, even if they are input and hidden layers or hidden and output layers. The final FC layer performs the voting towards the predefined classes. Some common functions that are used in order to compute the output values of the classes are the softmax and the cross-entropy functions. The layers of a CNN are depicted in the Figure 2-3.

The CNN training takes place through the forward and back propagation process, and the network is well defined after a lot of iterations and epochs. In the CNN training, not only the weights of the network are adjusted but also the feature detectors through the gradient descent method, which allows us to end up with the most suitable feature detector maps. Therefore, the highly hierarchical structure and large learning capacity of CNN models allow them to perform classification and predictions particularly well, being flexible and adaptable in a wide variety of complex challenges.

2-2 You Only Look Once (YOLO)

YOLO is a type of CNN that is used for object detection in real-time processing thanks to its high speed of detection. This approach relies on the prediction of bounding boxes and the computation of probabilities directly from the images in one scan of them. Four major versions of YOLO have been developed with various accuracy and speed characteristics. The first three versions are described below.

2-2-1 YOLO version 1

YOLO divides the input image into a S x S grid, and each grid cell predicts only one object whose center is located in the grid cell. This one-object concept limits the number of near objects that can be detected. Generally, each grid cell:

- predicts B boundary boxes and each box has a confidence score
- detects only one object despite the number of the boundary boxes
- predicts C conditional class probabilities, which are one per class and refer to the likeness of the object class

Each boundary box in a grid cell contains 5 elements, 4 of them (x, y, w, h) are mentioned to the location of the box and the last to the confidence score. The confidence score shows how likely the box contains an object and how accurate is the bounding box. The conditional class probability is the probability that the detected object belongs to a certain class, so there is one probability per class for each grid cell. In this way, a YOLO prediction has a shape $(S, S, B \cdot 5 + C)$, and its main aim is to build a CNN to predict a tensor with shape $(S, S, B \cdot 5 + C)$. This happens by using a suitable number and kind of layers.

The class confidence score for each predicted boundary box is computed as in Eq. (2-2).

$$\begin{gathered} box\ confidence\ score = P_r(object) \cdot IoU \\ conditional\ class\ probability = P_r(class_i|object) \\ class\ confidence\ score = box\ confidence\ score \cdot conditional\ class\ probability \Rightarrow \\ class\ confidence\ score = P_r(object) \cdot IoU \cdot P_r(class_i|object) \Rightarrow \\ class\ confidence\ score = P_r(class_i) \cdot IoU \end{gathered} \tag{2-2}$$

where

- $P_r(object)$ is the probability the box contains an object.
- Intersection over Union (IoU) is the intersection between the predicted and the ground truth bounding box.
- $P_r(class_i|object)$ is the probability the object belongs to the $class_i$ given that there is an object.
- $P_r(class_i)$ is the probability the object belongs to the $class_i$.

As YOLO predicts multiple boundary boxes in each grid cell, only one is used in order to compute the loss, which is the box that has the higher IoU with the ground truth. The loss function uses sum-squared error between the predictions and the ground truth and consists of three kinds of losses at each cell:

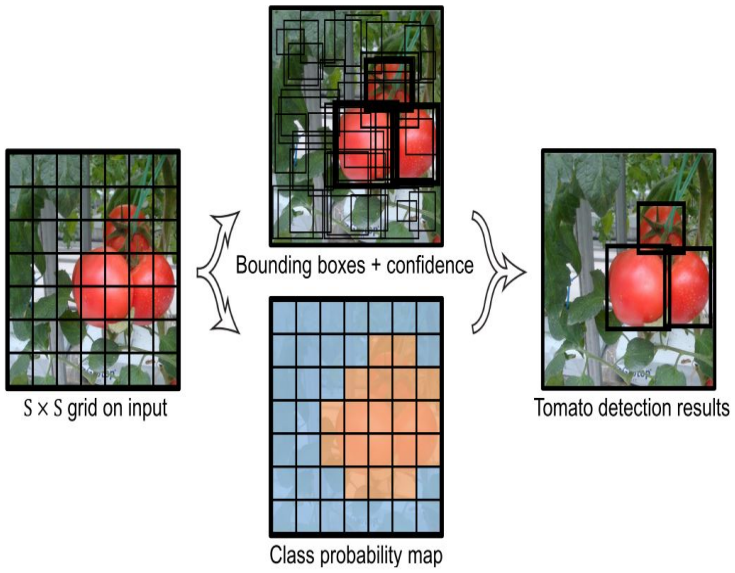


Figure 2-4: Applying YOLO detection in tomato images, [12]

- the classification loss, which is the squared error of the class conditional probabilities for each class.
- the localization loss, which computes the errors in the uniquely selected bounding box locations and sizes.
- the confidence loss, which computes the squared error of the box confidence score.

The formula for this loss can be found in [23]. Possible similar detections for the same object are faced using non-maximal suppression methods. The procedure of detecting tomatoes with a YOLO network is shown in Figure 2-4.

2-2-2 YOLO version 2

The second version of YOLO [24] provides higher accuracy and speed in comparison to the first version. This happens as a consequence of a series of improvements. Firstly, the convolution layers include batch normalization, which is used to normalize the activations of an input tensor before passing it into the next layer in the network. In other words, the output from the activation functions of a selected layer is normalized in order to prevent the output from being very large and causing instability through cascading to the neural network.

Moreover, YOLOv2 uses anchor boxes with an aspect ratio fitting to common real objects, so the boundary boxes have these shapes instead of random shapes. Anchors are a set of boxes with predefined locations and scales relative to images. In this way, the steep changes of gradients, due to the arbitrary guesses on the boundary boxes at the beginning of the training, are avoided. The need for predicting only the offsets of the boundary boxes makes the training more stable, and it is realized with only some modifications to the network

architecture. Furthermore, this YOLO version applies multi-scale training, as it chooses a different image size to do the training. This implies a data augmentation, so the network can predict robustly different image dimensions.

2-2-3 YOLO version 3

The third version of YOLO performs some improvements in comparison to the previous versions in order to increase accuracy and speed detection. These improvements refer to the class prediction, the boundary box prediction, the loss function calculation, and the network parts for the feature extraction. More details about these improvements can be found at [25].

2-3 Mask R-CNN

Mask Region-Convolutional Neural Network (Mask R-CNN) is a network that does instance segmentation, which means that it separates different objects in an image. More particularly, it combines object detection, where different objects are classified and localized, and semantic segmentation, where each pixel is classified in each class. Thus, instance segmentation detects and segments each object individually even they belong to the same class. The Mask R-CNN extends the Faster R-CNN [26], which is an object detection network, by adding a branch for predicting segmenting masks.

Mask R-CNN consists of two stages. The first stage proposes regions of interest where an object may be located. The second stage predicts the class of the object, refines the bounding box, and generates a mask for each object at a pixel level based on the proposals of the first stage. Both stages are connected to a backbone structure which is a Feature Pyramid Network (FPN) [27] type deep neural network consisting of a down-top, a top-down architecture, and connections between the layers of them. Generally, each CNN can create a multi-scale representation of the features. A feature pyramid can be built using the features that are extracted by the various levels of the CNN. The convolutional layers with high spatial resolution allow the network to detect features of small objects, while these with low spatial resolution contribute to the detection of strong features, which play an important role in the classification. The combination and connection of a down-top with a top-down architecture can lead the strong features, which are located to the end of a CNN, to high-resolution feature maps, so there are semantically strong features at all levels of the total network.

The first stage is called Region Proposal Network (RPN), and it scans the output feature maps of the backbone network and proposes regions that may contain objects. RPN uses a sliding window method to get anchor boxes from the feature maps. Then, it does a binary classification to figure out if the anchor contains an object or not, and bounding box regression to specify the size of the bounding box. In particular, an anchor is classified as positive, i.e. it contains an object, if its IoU with the ground-truth object is higher than a threshold. In this way, there are two losses in the first stage, which are the boundary box classification L_{cl1} and boundary box regression L_{bbox1} losses. The top positive anchor boxes are the output of this stage and are given to the second stage as the proposed regions.

The second stage takes the proposed Region of Interest (RoI) from the first stage, extracts feature maps from them using a RoI pooling layer, and performs classification and bounding

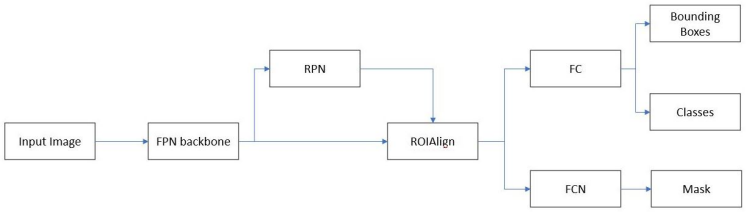


Figure 2-5: Mask R-CNN structure

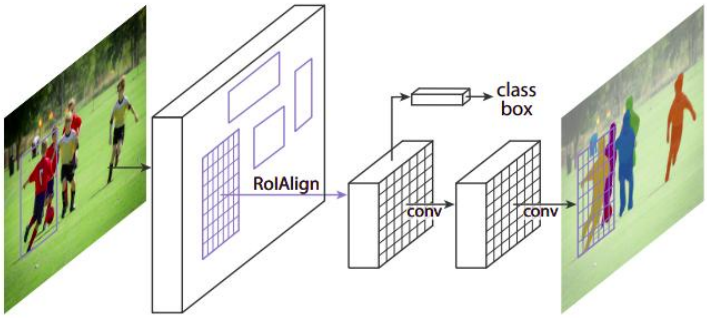


Figure 2-6: Mask R-CNN representation in the original paper [28]

box regression. The RoI pooling layer converts the part of a feature map, corresponding to a RoI, into a fixed size by performing a pooling operation (e.g. max), which then is fed to a fully connected layer. Then, the fully connected layer performs softmax classification to find the class of the object and also bounding box regression to find the suitable bounding box location and size. Thus, at the second stage, there are again two losses, which are about classification L_{cl2} and bounding box regression L_{bbox2}. A RoIAlign method is used in order to make the RoI perfectly aligned with the feature map grids during the pooling operation. This stage also has a branch for the generation of a binary mask for each object at a pixel level. This branch takes the proposed RoI and predicts the masks using a Fully Convolutional Network (FCN).

The structure of the Mask R-CNN is illustrated in Figure 2-5, while a representation of it in the original paper [28] is shown in Figure 2-6.

The loss function of Mask R-CNN is given in Eq. (2-3).

$$L = L_{cl1} + L_{cl2} + L_{bbox1} + L_{bbox2} + L_{mask} \tag{2-3}$$

where L_{mask} is the loss for the mask prediction, and it is calculated by taking the binary cross-entropy between the predicted mask and the ground truth.

Chapter 3

Deep Reinforcement Learning

3-1 Introduction to Reinforcement Learning

The Reinforcement Learning (RL) framework is used in order to solve problems related to control [29] and robotics. RL enables an agent to autonomously discover an optimal behavior through trial-and-error interactions with its environment [30]. The agent/robot and its environment are modeled by being in a state $s \in S$, and the agent performs actions $a \in A$ where each of them can be discrete or continuous. The goal of RL is to find a mapping from states to actions, called policy π, that picks actions a in the given states s maximizing the cumulative expected reward. The policy π can be deterministic or probabilistic. The first one always uses the exact same action for a given state in the form $a = \pi(s)$, while the second draws a sample from a distribution over actions for the given state, i.e., $a \sim \pi(s, a) = P(a|s)$. The policy also can be classified as stationary or non-stationary. The former is used for infinite horizons, where the cumulative rewards are not limited to a finite number of future time steps. The latter depends on the time step and is useful for the finite-horizon model in Eq. (3-1).

Thus, the RL concept is that the agent starts in an initial state $s_0 \in S$ in its environments and gathers an initial observation $o_0 \in O$. At each time step t, it takes an action $a_t \in A$ according to the policy. Then, the agent earns a reward $r_t \in R$ showing the suitability of the taken action, the state transitions to a new state s_{t+1}, and the agent receives an observation $o_{t+1} \in O$. This concept is illustrated in Figure 3-1.

The transitions between states s caused by actions a are modeled as $T(s_t, a, s_{t+1}) = P(s_{t+1}|s, a)$. The reward concept is used as a metric for finding the optimal policy. The optimal behavior can be expressed with different models, so there are many expressions about the expected reward J. A finite-horizon model only tries to maximize the expected reward for the horizon T.

$$J = E\left\{\sum_{t=0}^{T} R_t\right\} \tag{3-1}$$

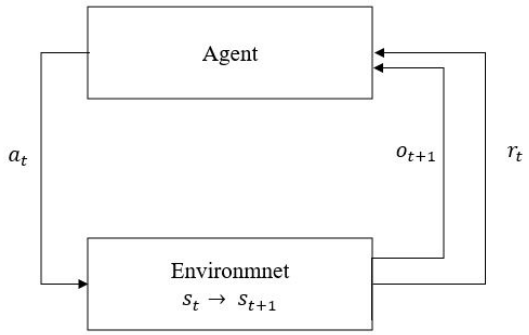


Figure 3-1: Reinforcement Learning Framework

On the other hand, future rewards can be discounted by a discount factor γ, which takes value from 0 to 1 and expresses how much the future is taken into account, and its value has to be selected manually:

$$J = E\left\{\sum_{t=0}^{\infty} \gamma^t R_t\right\} \tag{3-2}$$

The performance of the control law will be bad when a small γ is used, and there is a need for considering the long term rewards. Also, the performance may be unstable. If γ approaches 1, then a transient behavior will not have a significant role in the problem, as the agent will evaluate each of its actions based on the sum of all of its future rewards, which can lead to a huge variance. Then, the expected reward becomes an infinite horizon sum, such as in Eq. (3-3), which is not a good optimization criterion as the sum does not converge.

$$J = \lim_{H\to\infty} E\left\{\frac{1}{H}\sum_{h=0}^{H} R_h\right\} \tag{3-3}$$

The expected reward for an agent being in a certain state s and following a policy π is given also by Eq. (3-4) for the V-value function, $V^{\pi}(s) : S \rightarrow \mathbb{R}$

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, \pi\right] \tag{3-4}$$

where k expresses the time steps of the horizon. So, the optimal V-value function is given by Eq. (3-5).

$$V^*(s) = \max_{\pi\in\Pi} V^{\pi}(s) \tag{3-5}$$

There is another value function which is called Q-value function, $Q^{\pi}(s,a) : SXA \rightarrow \mathbb{R}$. This function is a measurement of quality of the action and is given by Eq. (3-6).

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a, \pi\right] \tag{3-6}$$

The optimal Q-value function is given by Eq. (3-7).

$$Q^*(s,a) = \max_{\pi \in \Pi} Q^{\pi}(s,a) \tag{3-7}$$

In that case, the optimal policy can easily be obtained:

$$\pi^*(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q^*(s,a) \tag{3-8}$$

Another useful function is the advantage function $A^{\pi}(s,a)$ which shows how good the action a is, as compared to the expected return when following the policy π.

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{3-9}$$

According to [30], many methods have been developed that compute $V^*(s)$ and $Q^*(s,a)$ and can be split into three categories: (a) dynamic programming-based optimal control approaches such as policy iteration or value iteration, (b) Monte Carlo methods, and (c) temporal difference methods such as TD(λ) (Temporal Difference learning), Q-learning, and SARSA (State-Action-Reward-State-Action).

Generally, policy search has been considered the harder problem for a long time as the optimal solution cannot directly be determined from equations like in Eq. (3-8). However, policy search, and especially policy gradient, methods have recently become an important alternative to value function based methods in robotics because of better scalability, as well as the convergence problems of approximate value function methods. Policy search methods are configured with a set of policy parameters θ. These methods optimize locally around existing policies by computing the change of the parameters in the policy that increases the expected cumulated reward, which is used as a performance objective. Usually, gradient-based approaches are used, so the update of the policy parameters becomes by following the gradient of the expected return J for a step size α:

$$\theta_{i+1} = \theta_i + \alpha \nabla_{\theta} J \tag{3-10}$$

The robotics problems are usually characterized by high-dimensional continuous states and actions, which create the curse of dimensionality problem. Also, the states are not observable completely and noise-free, so the exact states can be replaced by the environment's observations, which are called information or belief states, and thus a notion of uncertainty in the estimation must be taken into account. Another challenge is that the experience obtained by training is difficult and costly to be acquired in real-world conditions, so simulations have to be used. This creates the need for developing robust and generalizing algorithms. Moreover, the creation of a suitable reward function is another challenging issue in the use of RL in robotics and requires an engineering approach. One of the methods to solve the reward

shaping problem is the inverse reinforcement learning, where the reward function is updated continuously, and the appropriate policy is generated in the end.

The notion of the RL problem indicates that the agent has to discover new unused actions and their possible higher future rewards. The problem of choosing between following the safe strategy and discovering new strategies is called the exploitation-exploration trade-off. As regards the way of learning the policy from the data, there are offline and online methods, as well as off-policy and on-policy methods. In the offline method, the agent has to learn from data without interacting with the environment, while in the online method, it can gain experience from the environment during the learning. The off-policy methods learn independently of the employed policy, so an exploratory strategy different from the desired final policy can be employed during the learning process. On-policy methods collect sample information about the environment using the current policy. In this way, the off-policy methods are sample efficient as they are capable of using any experience.

3-2 Deep Reinforcement Learning (DRL)

The state space is high-dimensional and usually continuous in many real-world problems. DRL can learn complex policies with high-dimensional observations as inputs such as images, because of the neural networks, and especially the Convolutional Neural Network (CNN)s, which can handle high dimensional inputs without needing a huge increase in data when adding extra dimensions to the state or action state [31]. Actually, DRL combines RL with deep learning in the way that the action taken at every time step of the RL is determined by deep learning.

Deep learning relies on a function $f : X \rightarrow Y$ parameterized with $\theta \in \mathbb{R}^n (n \in \mathbb{N})$:

$$y = f(x; \theta) \tag{3-11}$$

This function can be implemented by a deep neural network. The deep neural networks consist of a large number of processing layers, where in each layer the values are transformed in a non-linear way and transmitted to the next layer. The connections between the network layers are characterized by weights, which are the parameters θ of the network. These parameters change during training in order to minimize a loss function. There are some different types of deep neural networks, such as deep feedforward neural networks, CNNs, Recurrent Neural Networks (RNN)s, Long Short Term Memory (LSTM), Deep Residual Network (DRN)s, and others.

3-3 Value-Based Methods in DRL

The value-based, policy-based, and model-based RL methods can scale to deep reinforcement learning methods in order to learn decision-making strategies from high-dimensional inputs. The value-based methods use a value-function to predict how good a state or a state/action pair is and subsequently to define a policy. The most common and simple value-based algorithm is the Q-learning [32] and its modified version, the fitted Q-learning [33]. The

Q-learning algorithm uses a lookup table for the Q-value of each state-action pair, which is difficult to implement in high-dimensional and continuous state-action spaces. In fitted Q-learning, the initial Q-value function is randomly initialized, and the approximation of the Q-value $Q\left(s,a;\theta_k\right)$ in each iteration is updated using the target value:

$$Y_k^Q = r + \gamma \max_{a' \in \mathcal{A}} Q\left(s', a'; \theta_k\right) \tag{3-12}$$

where k is the number of the iteration and s' is obtained by a dataset with experience tuple $< s, a, r, s' >$.

In neural fitted Q-learning [34], the Q-values are parameterized by a neural network $Q(s,a;\theta_k)$ which uses the state as an input and predicts a Q-value according to the different actions. The parameters θ_k are updated by a stochastic gradient descent method by minimizing the square loss between the predicted Q-value and the target value Y_k^Q. The approximation of the Q-value can be estimated conveniently as the second term of Eq. (3-12) can be computed in a single forward pass of the neural network. The problem is that when the weights are updated, then the target value also changes. This can create large errors that propagate with this update rule, and finally, the convergence to the optimal solution is slow or unstable. Another problem related to the use of function approximators is that the Q-values may be overestimated due to the max operator in the computations. The risk for instabilities and overestimations needs careful use of fitted Q-learning.

The Deep Q-Network (DQN) algorithm [35] is efficient in a variety of ATARI games by learning directly from image pixels. This network limits the instabilities of the neural fitted Q-learning by introducing the following improvements:

- The target Q-network in Eq. (3-12) is replaced by $Q\left(s', a'; {\theta_k}^-\right)$ where its parameters ${\theta_k}^-$ are updated every K iterations in this way $\theta_k^- = \theta_k$. This technique prevents the existence of instabilities that propagate quickly due to the change of the target value, as the target values are fixed for K iterations in this case.
- This algorithm uses an online setting with a replay memory where all the collected experience with an ϵ-greedy algorithm is kept for the last N time-steps. Then the updates become on a tuple set (s, a, r, s'), which is called mini-batch, and are selected randomly inside the replay memory. This allows for updates that cover a wide range of the state-action space, and one mini-batch update has less variance in comparison to a single tuple update.

As referred to previously, the max operation in Eq. (3-12) uses the same value to select and evaluate an action. This makes it more probable to select overestimated values in case of inaccuracies and noise, which results in an upward bias. The Double Deep Q Network (DDQN) algorithm uses two separate Q estimators where the selection of the estimator and its value are uncoupled. The use of two Q-estimators creates unbiased Q-value estimates of the actions, and thus the upward bias in action value estimation is avoided when there are errors in the estimated Q-values. In DDQN the target value Y_k^{DDQN} is computed in Eq. (3-13)

$$Y_k^{DDQN} = r + \gamma Q\left(s', \underset{a \in \mathcal{A}}{\text{argmax}} Q\left(s', a; \theta_k\right); \theta_k^-\right) \tag{3-13}$$

which leads to a smaller overestimation of the Q-values and thus better stability.

The dueling network architecture [36] uses the advantage function Eq. (3-9), which has the benefit that its values have a lower variance that leads to easy convergence, and the policy does not change discontinuously with changing values.

Generally, the DQN algorithm and all the following variations of it that are described can have a high performance and exceed the human performance in some applications like the Atari games. On the other hand, there are some limitations regarding the DQN algorithm and its modifications. These types of algorithms are not suitable to deal with high and/or continuous action spaces, as they are based on finding an action that maximizes the Q-value function, so it is impossible to apply it directly to continuous action spaces where there are infinity actions. Even though a discretization of the action space could be a solution to this problem, this would create the curse of dimensionality and a possible loss of information. Also, this kind of algorithms can not learn probabilistic policies explicitly, which can be solved by policy gradient methods.

3-4 Policy Gradient Methods in DRL

3-4-1 Policy Gradient Theorem

The policy gradient approaches are part of the policy-based methods in RL. These techniques optimize a performance objective, which is usually the expected cumulative reward, by finding a policy with the gradient ascent with respect to the policy parameters. The gradients on the policy parameters are found either with stochastic or deterministic policy gradient theorems.

The stochastic policy $\pi(s, a)$ is the probability $P(a|s)$ of taking the action a given a state s. The objective is to find a policy π_θ, which creates a trajectory τ $(s_1, a_1, s_2, a_2, \ldots, s_H, a_H)$ and maximizes the expected rewards:

$$J(\theta) = E_{a\sim\pi_\theta(s,a)}\left[\sum_t r(s_t, a_t)\right] = E_{\tau\sim\pi_\theta(\tau)}\left[\sum_t r(\tau_t)\right] \tag{3-14}$$

The expected rewards in the continuous space can be also written as:

$$J(\theta) = \int \pi_\theta(\tau) r(\tau) d\tau \tag{3-15}$$

The policy gradient of the objective function is:

$$\nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta(\tau) r(\tau) d\tau \tag{3-16}$$

The policy gradient of the policy can be written as:

$$\nabla_\theta \pi_\theta(\tau) = \pi_\theta(\tau)\ \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} = \pi_\theta(\tau)\ \nabla_\theta \log \pi_\theta(\tau) \tag{3-17}$$

In this way, Eq. (3-16) becomes:

$$\nabla_\theta J(\theta) = \int \pi_\theta(\tau)\ \nabla_\theta \log \pi_\theta(\tau) r(\tau) d\tau = E_{\tau\sim\pi_\theta(\tau)} \left[\nabla_\theta \log \pi_\theta(\tau) r(\tau)\right] \tag{3-18}$$

which shows that the policy gradient is described by an expectation, and it can be approximated using sampling. Also, the integral, which is not convenient in computations, is avoided. The policy $\pi_\theta(\tau)$ is computed in the following way:

$$\pi_\theta(\tau) = p\left(s_0\right) \Pi_{t=1}^{T} [\pi_\theta\left(s_t, a_t\right) p\left(s_{t+1} \mid s_t, a_t\right)] \tag{3-19}$$

where $p(s_0)$ shows the probability to start in a state s_0. The product rule of the probability is applied as each action probability is independent of the previous one due to the Markov property. At each time step t an action is taken according to the policy $\pi_\theta\left(s_t, a_t\right)$ and the environment dynamics $p\left(s_{t+1} \mid s_t, a_t\right)$ show in which state the agent transits. The logarithm of the policy is:

$$\log \pi_\theta(\tau) = \log p\left(s_0\right) + \sum_{t=1}^{T} [\log \pi_\theta\left(s_t, a_t\right) + \log p\left(s_{t+1} \mid s_t, a_t\right)] \tag{3-20}$$

and the policy gradient of the logarithm of the policy is:

$$\nabla_\theta \log \pi_\theta(\tau) = \nabla_\theta \sum_{t=1}^{T} \log \pi_\theta\left(s_t, a_t\right) \tag{3-21}$$

Using Eq. (3-21) and sampling, Eq. (3-18) is approximated as:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta\left(s_{i,t}, a_{i,t}\right)\right) \left(\sum_{t=1}^{T} r\left(s_{i,t}, a_{i,t}\right)\right) \tag{3-22}$$

This result shows that the knowledge of the initial distribution, as well as the environment dynamics, is not necessary, which is very useful as it is difficult to have a model in many applications. This shows the model-free feature of the policy gradient methods. The policy update happens like Eq. (3-10). Before improving the policy by using the policy gradients method, there is the need for evaluating the policy, which corresponds to the computation of the reward in Eq. (3-22). There are many ways to compute the reward term, such as using the Monte-Carlo estimator, which estimates the rewards by using rollouts, which means that the whole episode runs in order to compute the expected rewards. The problem is that there is the need for on-policy rollouts and there is a lot of variance due to the contribution of the reward in every time step t. An increase of the batch size would decrease the variance for the sampled rewards but would deteriorate the sample efficiency. In order to solve the problem of the high variance in the sampled trajectories, a baseline variable b is introduced in a way that the optimization takes place for the difference of the rewards. The baseline value is independent of the policy parameters so that the gradient estimation remains unbiased. Now, the gradient of the objective function is given by Eq. (3-23).

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta\left(s_{i,t}, a_{i,t}\right)\right) \left(\sum_{t=1}^{T} r\left(s_{i,t}, a_{i,t}\right) - b\right) \tag{3-23}$$

The use of a suitable baseline value is an important challenge. Considering the formula of the advantage function in Eq. (3-9), the values of the advantage function are lower than these of the Q-value function. The term $\sum_{t=1}^{T} r\left(s_{i,t}, a_{i,t}\right)$ is related to the Q-value function $Q(s,a)$, which means that a good choice of the baseline value can be the V-value function $V(s)$.

3-4-2 Actor-Critic Algorithms

As it is shown previously, the policy gradient method requires the reward estimation in the t time steps, which is related to the estimation of a value function. One approach to this estimation is an actor-critic network which consists of two parts: the actor, which refers to the policy, and the critic, which refers to the estimation of the value function [37]. In the DRL approach, both actor and critic can be represented by non-linear neural network function approximators. The actor uses gradients that are derived from the policy gradient theorem and adjusts the policy parameters θ in a way suggested by the critic. The critic is parameterized with w and estimates the value function for the current policy π_θ.

An off-policy approach, which has a set of tuples (s, a, r, s') from a replay memory, to estimate the critic is to use a bootstrapping temporal difference algorithm where at each iteration k the Q-value $Q(s, a; w)$ is updated using the following target value:

$$Y_k^Q = r + \gamma Q\left(s', a = \pi\left(s'\right); w\right) \tag{3-24}$$

This approach takes the immediate reward and a bootstrapped Q-value estimation of the next pair s, a in the trajectory. Bootstrapping refers to the recursive use of the own Q-value estimates. The approach described by Eq. (3-24) is simple and can be sample efficient because of the replay memory, but it is not computationally efficient as it uses a bootstrapping technique, which creates instabilities, due to the recursive estimation of the own value at each next step, and propagates slowly the rewards backward in time. An on-policy may be more computationally efficient, as the use of samples collected by the on-policy behavior policies can solve the two mentioned disadvantages. Thus, methods that combine off-policy and on-policy data for policy evaluation can be used like algorithm *Retrace*(λ) [38].

To put the actor and the critic together, the policy parameters are updated in each time step according to Eq. (3-25),

$$\theta \leftarrow \theta + \alpha_\theta Q(s, a; w) \nabla_\theta \log \pi_\theta(a \mid s) \tag{3-25}$$

where the α_θ is the learning rate of the actor network. Then, a correction term, which shows the error between the Q-value and its target value, is computed in Eq. (3-26).

$$\delta_t = Y_k^Q - Q(s, a; w) \tag{3-26}$$

This term is used to update the parameters of the Q-value function:

$$w \leftarrow w + \alpha_w \delta_t \nabla_w Q(s, a; w) \tag{3-27}$$

where the α_w is the learning rate of the critic network.

3-4-3 Advantage Actor-Critic

As it is mentioned in 3-4-1, the use of the advantage function reduces the high variances. The advantage function tells the improvement of the action a taken in a state s compared to the average value for actions taken in state s, as the $Q(s,a)$ shows how good the action a taken in state s is and the $V(s)$ shows the average value of the state s. So, it shows the extra reward obtained, if the action a is taken, in comparison to the average. The use of the advantage function as a critic network has the disadvantage that there will be the need for representing both value functions $Q(s,a)$ and $V(s)$ with neural networks. The Q-value function can be written according to the Bellman optimality equation as:

$$Q\left(s_t, a_t\right) = \mathbb{E}\left[r_{t+1} + \gamma V\left(s_{t+1}\right)\right] \tag{3-28}$$

so the advantage function becomes

$$A(s_t, a_t) = r_{t+1} + \gamma V\left(s_{t+1}\right) - V\left(s_t\right) \tag{3-29}$$

which shows that there is the need for only one neural network, which is for the V function.

Asynchronous Advantage Actor-Critic (A3C)

In A3C the experience replay is not used as it requires a lot of memory. On the contrary, multiple agents are executed asynchronously in parallel for different instances of the environment. In this way, they explore a bigger part of the state-action pairs in less time. The agents are trained in parallel and update a global network asynchronously. Due to this asynchronous concept, it is possible that some agents will use older versions of the parameters, so the aggregating update of the global parameters will not be optimal.

Synchronous Advantage Actor-Critic (A2C)

On the other hand in A2C, we wait for all the agents to finish their training and then synchronously update the parameters of the global network. Also, due to the synchronous nature, there is not the need for different agents but for the same agent working in different environments, as each agent will have the same set of weights as they are updated in parallel. The different environments can be connected between them by considering two networks; the step and the train network. The step network interacts with all the environments for n time steps in parallel and outputs a batch of experiences. Then, the train network is trained with these experiences. Then the step network is updated with the new weights.

3-5 Deep Deterministic Policy Gradient (DDPG)

In robotic applications, the actions usually are not stochastic, as the output action is determined for each state, and there is not uncertainty in the action selection. Moreover, the stochastic policy gradient performs worse as the dimensionality of the controller increases. Let $\mu(s)$ be the deterministic policy: $\mu(s) \ : \ S \rightarrow A$.

An approach to find the optimal policy in each time step k would be the following:

$$\mu^{k+1}(s) = \underset{a}{\text{argmax}} Q^{\mu^k}(s, a) \tag{3-30}$$

In a continuous space, the maximization operation would be computationally infeasible as we would have to search in the whole state-action space. If we define a differentiable deterministic policy $a = \mu_\theta(s)$, then we can lead it in the gradient direction of a performance objective. The performance objective is:

$$J(\theta) = \mathbb{E}_{s\sim\rho^{\mu_\theta}}\left[r\left(s, a\right)\right] = \mathbb{E}_{s\sim\rho^{\mu_\theta}}\left[Q\left(s, \mu_\theta(s)\right)\right] \tag{3-31}$$

where we consider as reward function the Q-value function, and ρ^{μ_θ} is the state distribution following the policy $\mu(s)$. Eq. (3-31) can be written as:

$$J(\theta) = \int_S \rho^{\mu}(s) Q\left(s, \mu_\theta(s)\right) ds \tag{3-32}$$

and by taking its gradient and considering the chain rule, i.e. taking the derivative of Q w.r.t the action a and then the derivative of the policy $\mu_\theta(s)$ w.r.t the parameters θ, we have:

$$\begin{aligned} \nabla_\theta J(\theta) &= \int_S \rho^{\mu}(s) \nabla_a Q^{\mu}(s, a) \nabla_\theta \mu_\theta(s) \Big|_{a=\mu_\theta(s)} ds \\ &= \mathbb{E}_{s\sim\rho^{\mu}}\left[\nabla_a Q^{\mu}(s, a) \nabla_\theta \mu_\theta(s)|_{a=\mu_\theta(s)}\right] \end{aligned} \tag{3-33}$$

This equation uses $\nabla_a Q^{\mu}(s, a)$ and $\nabla_\theta \mu_\theta(s)$ which shows a relation with actor critic methods. The expectation can be estimated using sampling, and the policy parameters are updated according to Eq. (3-10).

DDPG is an off-policy actor-critic algorithm, which learns a deterministic policy $\mu_\theta(s)$ in a continuous space. This algorithm uses an experience replay and a target network, such as the DQN that used them to stabilize the learning of the Q-function. In contrast to DQN, the parameter updates become smoothly as the parameters do not stay the same for every K iterations, but they change in the following way:

$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta' \tag{3-34}$$

where $\tau \ll 1$. Due to the off-policy character of the algorithm, there is the need for exploration. An exploration policy $\mu'(s)$ is derived by adding noise N in order to achieve a better exploration:

$$\mu'(s) = \mu_\theta(s) + N \tag{3-35}$$

In robotics applications, may there is the case that the states are positions and velocities, which have different units, so there is the possibility of varying statistics. This is faced using batch normalization, where every dimension across the samples of a minibatch is normalized.

As it can be derived from above, the DDPG algorithm uses four neural networks: a Q network, a deterministic policy network, a target Q network, and a policy target network. The two-target networks are copies of the original two networks (Q network and policy network),

and they are used in order to improve the stability of the learning. The actor-network is represented like $\mu(s;\theta^{\mu})$ where θ^{μ} are the weights of the actor-network. The output of the actor is the best-believed action at any given state s. The critic-network is represented like $Q(s,a;\theta^{Q})$ where θ^{Q} are the weights of the critic-network. The critic-network evaluates the Q-value function by using the actor's best-believed action. The target network for the actor is represented like $\mu(s;\theta^{\mu'})$ where $\theta^{\mu'}$ are the weights of the actor target network. The target network for the critic is represented like $Q(s,a;\theta^{Q'})$ where $\theta^{Q'}$ are the weights of the critic target network. At every step of an episode, an action is chosen according to the output of the actor-network and the additive noise N. According to this action, the agent receives a reward r and transits to the next state s'. This transition information is stored in an experience replay buffer. Then we sample N transitions from the replay buffer, and then we calculate the target Q value in Eq. (3-36).

$$y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'\left(s_{i+1} \mid \theta^{\mu'}\right) \mid \theta^{Q'}\right) \tag{3-36}$$

Then, the squared error between the Q and the target Q value is computed, and it is used in order to update the critic network's weights by minimizing this error. After that, the actor network's weights are updated using a policy gradient method and sampling. Finally, the target networks' weights are updated using the soft update technique, which is described in Eq. (3-34). The DDPG algorithm is presented in the Algorithm 1, as it is obtained by the original paper [21].

Algorithm 1: DDPG algorithm

Randomly initialize critic network $Q\left(s, a \mid \theta^{Q}\right)$ and actor $\mu\left(s \mid \theta^{\mu}\right)$ with weights θ^{Q} and θ^{μ}.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^{Q}, \theta^{\mu'} \leftarrow \theta^{\mu}$.
Initialize replay buffer R.
for *episode* $= 1, M$ **do**
  Initialize a random process $\mathcal{N}$ for action exploration.
  Receive initial observation state s_1.
  for $t = 1, T$ **do**
    Select action $a_t = \mu\left(s_t \mid \theta^{\mu}\right) + \mathcal{N}_t$ according to the current policy and exploration noise.
    Execute action a_t and observe reward r_t and observe new state s_{t+1}.
    Store transition (s_t, a_t, r_t, s_{t+1}) in R.
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R.
    Set $y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'\left(s_{i+1} \mid \theta^{\mu'}\right) \mid \theta^{Q'}\right)$.
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i \left(y_i - Q\left(s_i, a_i \mid \theta^{Q}\right)\right)^2$.
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_i \nabla_a Q\left(s, a \mid \theta^{Q}\right)\Big|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu\left(s \mid \theta^{\mu}\right)\Big|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$$

  end
end

Chapter 4

Setup and Methods

4-1 Simulation Setup

The experimental setup is simulated using Gazebo. The software for the implementation of the active perception algorithm is written using Robot Operating System (ROS). The integration between ROS and Gazebo is achieved by the Gazebo plugins, which have a similar message interface with the ROS components. This compatibility is an important reason for using Gazebo and ROS. The 7 Degrees of Freedom (DoF) robotic manipulator KUKA LBR IIWA 14kg is used. The robot is simulated in Gazebo using a ROS metapackage [39].

Intel D435 sensor seems to be the best choice in a harvesting robot operating in the field or a greenhouse according to [7]. Even though the experiments of the thesis are carried out in a simulation environment and not in a real greenhouse, the thesis aim for future work in real greenhouses justifies its selection. The sensor is used by importing its Unified Robot Description Format (URDF) file and the suitable Gazebo plugins which make the sensor functional.

A model of a tomato plant is used for the simulation experiments. The tomato plant includes some tomatoes which are highly occluded by leaves, which is important for testing the detection algorithms in dense occlusions. Also, some tomatoes can be seen by the robot as adjacent. The tomato plant, with the robot and the camera, in the simulation environment, are shown in Figure 4-1.

4-2 Computer Vision Methods

4-2-1 Image Dataset

The initial image dataset consisted of tomato images from the Fruits 360 dataset. This dataset contains images from many fruits and vegetables. Also, different varieties of tomatoes are included. The dimensions of each image in this dataset are 100x100 pixels. An example of

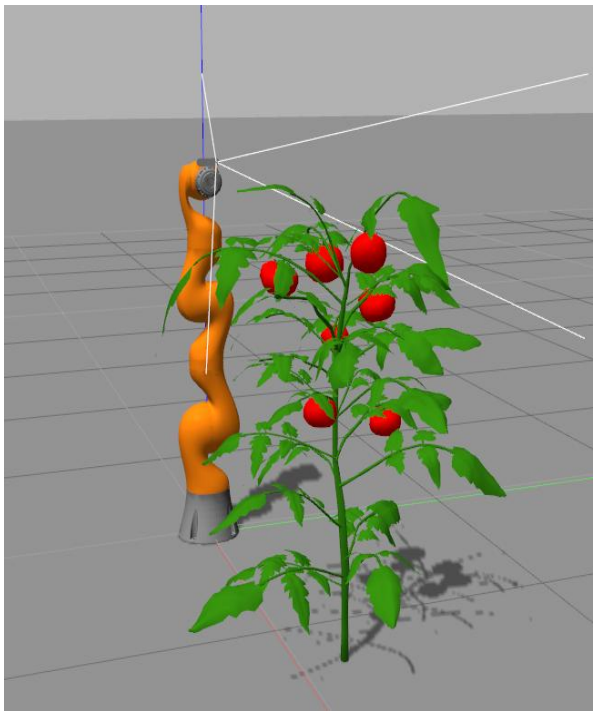


Figure 4-1: Simulation environment

a tomato image is shown in Figure 4-2. However, the fact that the whole image is covered by the fruit makes this image more suitable for classification purposes than object detection and localization. The final used image dataset consists of images with real and simulated tomatoes. An example of real and simulated tomato images is shown in Figure 4-3. The real images were provided by the Centre for Genetic Resources, the Netherlands (CGN) in Wageningen University and Research. The real images contain ripe, ripening, and unripe tomatoes under natural daylight and shading conditions. Also, there are many variations, such as occlusions, overlaps, size variations, and different mature levels, which is useful in order to make the tomato detector more robust. The simulated images are collected by mounting the camera on the robotic manipulator in the Gazebo simulation environment and moving the arm in various positions and angles around the tomato plant. Also, the tomato plant is located in various positions and angles with respect to the world coordinate frame. The reason for combining these two kinds of images is that the use of only simulated images can lead to learning only the simulated plant and overfitting. Moreover, the use of real images can contribute to the generalization of the learned policy to other simulated tomato plants.

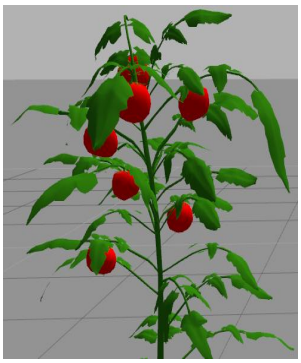
4-2-2 You Only Look Once (YOLO) Training and Evaluation

Dataset and Annotation

The YOLO network is trained and validated using 150 images from the CGN dataset, where the 120 images constitute the training dataset and the rest 30 images form the validation dataset. The annotation procedure takes place manually using the annotation tool LabelImg. The annotations have a Visual Object Classes (VOC) format. During the annotation, the

Figure 4-2: Example of tomato image in Fruits 360 dataset

(a) Real tomato image



(b) Simulated tomato image

Figure 4-3: Examples of real and simulated tomato images

following are not labelled as tomatoes: the unripe tomatoes, the severely occluded ripe tomatoes, the tomatoes with enough green marks on the fruit, the tomatoes occluded by tags, the tomatoes cut at the side of the image, and the tomatoes that do not look healthy. Following these rules, 476 tomato labels are included in the 150 tomato images. An example of the annotation process is shown in Figure 4-4.

Training Implementation and Parameters

The 3rd version of YOLO (subsection 2-2-3) is used for training thanks to its higher detection speed and accuracy in comparison to the other two previous versions. The implementation is based on this GitHub repository. Before the training, prior anchors, that suit to the bounding boxes of the ground truth tomatoes, are generated. In this way, the scales and the aspect ratios of the bounding boxes are adjusted in order to fit the size and the shape of the fruits.

Unless there is a huge Random Access Memory (RAM) capacity, there is not the capability to pass in once all the training data to the Convolutional Neural Network (CNN) in training. So, there is a need to split the data into segments/batches. Batch size is the number of training images that are used in a single batch. Learning rate is the amount that the weights are updated during the backpropagation. A large learning rate can lead the model to converge quickly to a suboptimal solution, while a small learning rate requires more training epochs and slows down the training process. The training parameters are shown in Table 4-1. The batch size is set to 1 due to the GPU limitations. Also, the learning rate is divided by

Figure 4-4: Example of annotation a tomato image for YOLO training

10 when two epochs pass, and the validation loss does not improve. The learning rate was decreased to 10^{-9} at the 40th epoch. Training stops when the validation loss is not improved in three consecutive epochs, and the learning rate has changed. Finally, training stopped at the 41st epoch. Moreover, no pre-trained weights are used. The object threshold, i.e. the confidence used to distinguish between object and non-object, is 0.5. The non-maximal suppression threshold, i.e. the threshold for determining whether two detections are the same, is 0.45. The Intersection over Union (IoU) threshold, i.e the threshold to determine whether a detection matches with the ground truth, is 0.5

Description	Symbol	Value
number of epochs	N_e	100
batch size	N	1
initial learning rate	lr	0.0001
max image input size	I_{max}	448
min image input size	I_{min}	288

Table 4-1: YOLOv3 training parameters

Training Evaluation

Precision P and recall R are two performance metrics that are used in the evaluation of the detection accuracy in object detection. Precision measures how accurate the predictions are, i.e. the ratio of the correct predictions to the total predictions. A correct prediction of a ground-truth bounding box is called True Positive (TP). An incorrect prediction of a non-existing ground-truth bounding box is called False Positive (FP). Precision is defined as in Eq. (4-1).

$$Precision = \frac{TP}{TP + FP} \tag{4-1}$$

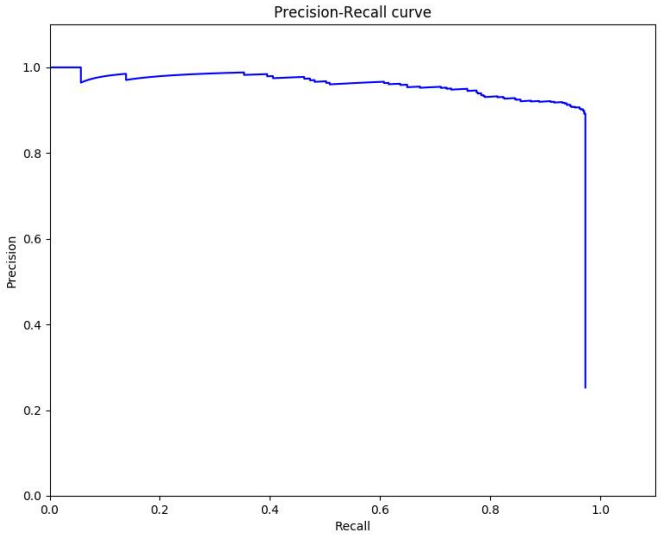


Figure 4-5: PR curve for test dataset using YOLOv3 training

Recall measures how well all the ground-truth bounding boxes are found, i.e. the ratio of the correct predictions to the total number of the object in the dataset. A ground-truth bounding box that is not detected is called False Negative (FN). Recall is defined as in Eq. (4-2).

$$Recall = \frac{TP}{TP + FN} \tag{4-2}$$

In other words, high precision shows that the object detector identifies only relevant objects, and high recall shows that the object detector finds all the ground-truth objects. The relationship between precision and recall is illustrated by the precision-recall curve. Average Precision (AP) is a common performance metric in object detection, which exploits both precision and recall metrics. AP is related to the area under the precision-recall curve. The higher is the AP, the better is the detection accuracy. AP is computed by interpolating the precision in each recall level. This is shown in Eq. (4-3)

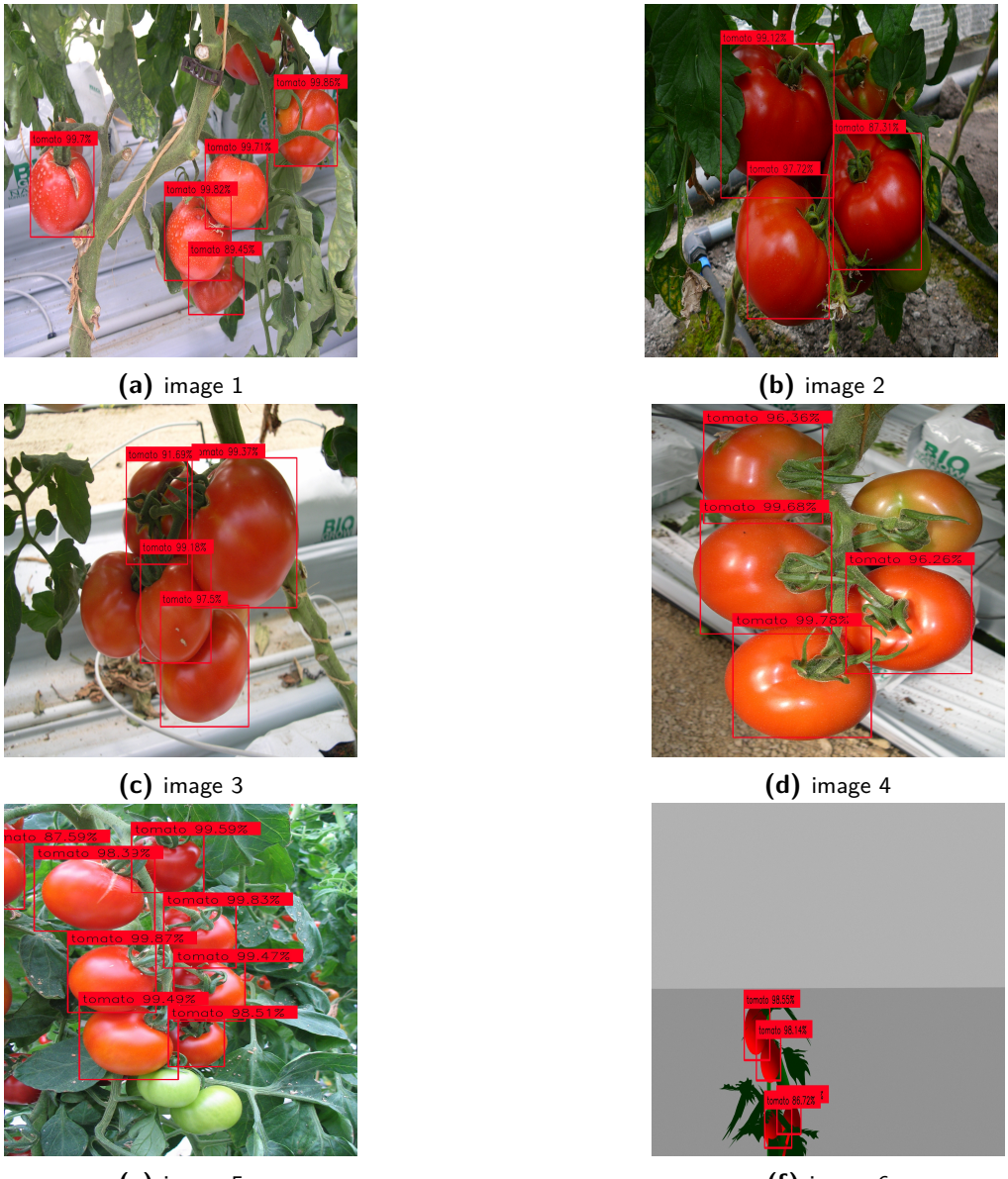
$$\begin{aligned} \mathrm{AP} &= \sum_{n=0} \left(R_{n+1} - R_n\right) P_{\mathrm{env}}\ \left(R_{n+1}\right) \\ P_{\mathrm{env}}\left(R_{n+1}\right) &= \max_{\bar{R}:\bar{R}\geq R_{n+1}} P(\bar{R}) \end{aligned} \tag{4-3}$$

where $P_{\mathrm{env}}\ \left(R_{n+1}\right)$ is the interpolated precision.

The AP of the training in the validation set is 0.9477. A test dataset of 50 unseen images is used. The precision-recall curve for the test dataset is shown in Figure 4-5. The AP for this test dataset is 0.9394. This detection performance is very good given the small training dataset of 120 images and the absence of image augmentation.

The results of detection in unseen real and simulated images are shown in Figure 4-6.

The results show that the detection confidence is very high, and it reaches 99 % when the tomatoes are fully observed. Also, it is high in the case of occlusions by other fruits and



(a) image 1 **(b)** image 2

(c) image 3 **(d)** image 4

(e) image 5 **(f)** image 6

Figure 4-6: Detection results of trained YOLOv3 in unseen real and simulated images

branches. Only one not severely occluded tomato in Figure 4-6-(c), is not detected. The unripe and ripening tomatoes are not recognized as ripe tomatoes. Moreover, the tomatoes with green marks on them are not recognized as a tomato, which is according to the annotation rules. Generally, the results are very good, the implemented CNN can even detect the tomatoes, behind the leaves and the branches, with a large detection confidence score, and the trained model generalizes very well. A larger dataset with data augmentation would lead to even better detection results.

4-2-3 Mask Region-Convolutional Neural Network (Mask R-CNN) Training and Evaluation

Dataset and Annotation

The training and evaluation of the Mask R-CNN take place using both real and simulated tomato images. The annotation procedure happens manually using the annotation tool VGG Image Annotator [40]. Each object in the image is annotated in a polygon format in order to extract a mask for it. Two labels are used: tomato and robot. The following tomatoes are not annotated as tomatoes: the unripe, the ripening, and the unhealthy tomatoes. A label for the leaves and branches is not used as it requires a lot of annotation time. Even the application of pre-augmentation techniques on the simulated images does not reduce significantly the annotation time. The bounding boxes are generated automatically according to the mask to boost up the annotation time. The annotation procedure of the real and simulated tomato images is shown in Figure 4-7.

Data augmentation is applied to the images during the training. In this way, the training dataset images are augmented, so more images are given as input to the Mask R-CNN. The applied types of data augmentation are horizontal flip, change of brightness, change of contrast, and gaussian blurring. The change of the brightness occurs by multiplying all the pixels of the image with a specific value. The contrast is adjusted by scaling each pixel to $127 + \alpha * (v - 127)$, where v is the pixel value and α is sampled uniformly in a specified interval. The gaussian blurring happens by blurring the image with a gaussian kernel of specified standard deviation. The augmentation is implemented using the library imgaug [41]. The annotation of the original image adjusts for all the used image augmentation types as it is shown in Figure 4-8.

Training Implementation and Parameters

The implementation of the Mask R-CNN is based on [42]. This is an implementation of the original Mask R-CNN paper [28] using TensorFlow and Keras. The theory behind the Mask R-CNN is explained in section 2-3. Two different training strategies are applied in order to see the improvement of the training performance. The common parameters on these training strategies are shown in Table 4-2.

The backbone network architecture is Resnet 101 [43]. The batch size is limited to 1 due to my 4GB Graphics Processing Unit (GPU). The weight decay is used in order to help the training model to generalize. This small number parameter is multiplied with the sum of the squares of the weights at the loss function in order to prevent the model from getting too

(a) Real image annotation **(b)** Simulated image annotation

Figure 4-7: Polygon annotation in real and simulated images

Description	Value
batch size	1
learning momentum	0.9
weight decay	0.0001
max image input size	1024
min image input size	800
min detection confidence	0.5

Table 4-2: Mask R-CNN common parameters for all the training strategies

complex and overfitting. The input images are resized in a square mode, such that multiple images can be set in one batch. This happens by scaling the images such that the small side is equal to 800 pixels, and the large side does not exceed 1024 pixels. Then, zero pixels are added to the small size to obtain a square image of dimensions $1024x1024$ pixels. The minimum detection confidence is the threshold for accepting a detected instance. Moreover, transfer learning is performed in every training strategy by using pre-trained weights, which have been trained in Common Objects in Context (COCO) dataset. Even though COCO does not include the labels tomato and robot, it contains a lot of other classes, so the trained weights have already learned a lot of features common to my dataset images.

The dataset in the first training strategy consists of 153 training images, 103 simulated and 50 real, and 98 validation images, 68 simulated and 30 real. The following three types of augmentation are applied: horizontal flip, gaussian blurring with a standard deviation equal to 2.5, and change of brightness by multiplying every image pixel with a value in the interval $(0.8, 1.5)$. The number of training epochs is 80. The number of training steps per training epoch is 600, while the number of the validation steps at the end of every training epoch is 100. One of the three augmentation types is applied to the input image in each training step with a probability equal to 0.75. The side of the anchors, which are used from the Region Proposal Network (RPN), can have a length equal to one of $32, 64, 128, 256, 512$ pixels. The non-max suppression threshold to filter RPN is 0.7. The learning rate lr is 0.001. The training procedure lasts 18:30 hours. The parameters are summarized in Table 4-3. The training results are mentioned in the next subsection.

(a) Original image

(b) Horizontal flip

(c) Change of brightness

(d) Change of contrast

(e) Gaussian blurring

Figure 4-8: Annotation adjustment after augmentation

Description	Value
total images	251
training images	153
validation images	98
number of epoches	80
number of augmentation types	3
training steps per epoch	600
validation steps at each epoch	100
anchors' size	32,64,128,256,512
non-max suppression threshold	0.7
learning rate	0.001

Table 4-3: Mask R-CNN first training strategy parameters

The parameters of the second training strategy are chosen to improve the results of the first training strategy. The dataset in the second training strategy consists of 220 training images, 144 simulated and 76 real, and 125 validation images. This dataset contains all the images of the first training dataset, and extra images are added. The annotation of the already existed images is refined. The change of contrast with an *alpha* value in the interval $(0.6, 1.4)$ is an extra augmentation type. The change of brightness takes place by multiplying every image pixel with a specific value in the interval $(0.7, 1.3)$. The number of training epochs is 65. The number of training steps per training epoch is 1000, while the number of the validation steps at the end of every training epoch is 120. One of the four augmentation types is applied to the input image in each training step with a probability equal to 0.8. The side of the anchors, which are used from the RPN, can have a length equal to one of $16, 64, 128, 256, 512$ pixels. This happens in order to be able to fit the smallest parts of the tomatoes. The non-max suppression threshold to filter RPN is 0.7. The learning rate lr for the first 20 epochs is 0.001, while for the rest 45 epochs is 0.0001. A dropout regularization is used in order to face the problem of overfitting the training data. The dropout method prevents from overfitting by dropping out different random nodes during training. The dropout is placed after the Fully Connected (FC) layers in the Feature Pyramid Network (FPN). The dropout rate is 0.3, which means that the 0.3 of the nodes are dropped out during a backpropagation. The training procedure lasts 24 hours. The parameters are summarized in Table 4-4. The training results are mentioned in the next subsection.

Training Evaluation

The training and validation losses of the two training strategies are shown in the Figures 4-9, 4-10, 4-11, 4-12, 4-13, 4-14, 4-15, 4-16, 4-17, 4-18, and 4-19 . The orange line corresponds to the first training effort, while the blue line corresponds to the second training effort. The thick line corresponds to smoothed values, while the transparent line shows the real values.

The loss graphs show that the training loss is smooth and gets small after some epochs in both cases. However, the validation loss is not sufficient and shows that the trained model overfits after few epoches. The validation loss is smaller in the second case, and there is not such a huge overfitting as in the first case. This is explained by the changes in the second

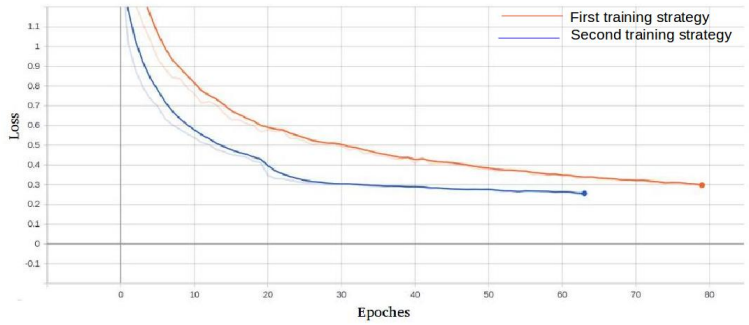


Figure 4-9: Training loss

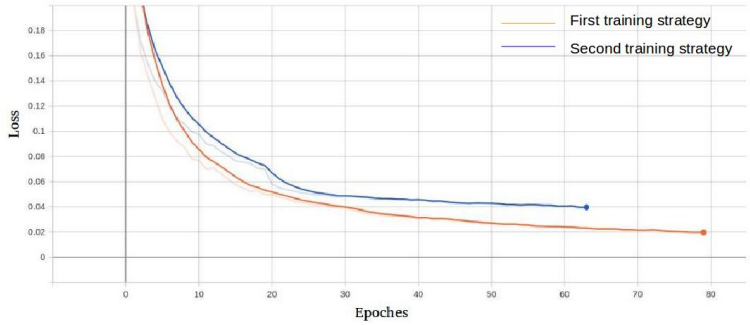


Figure 4-10: Training loss for Mask R-CNN bounding box refinement

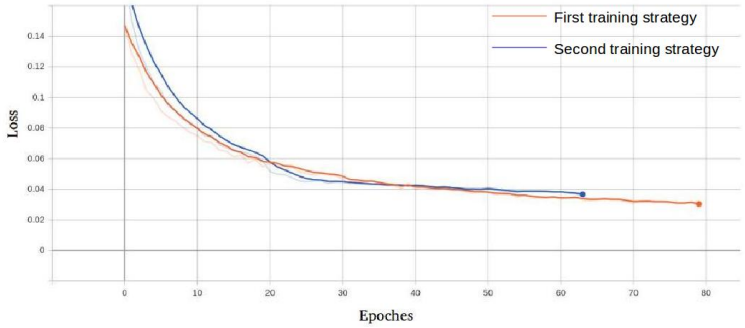


Figure 4-11: Training loss for the classifier head of the Mask R-CNN

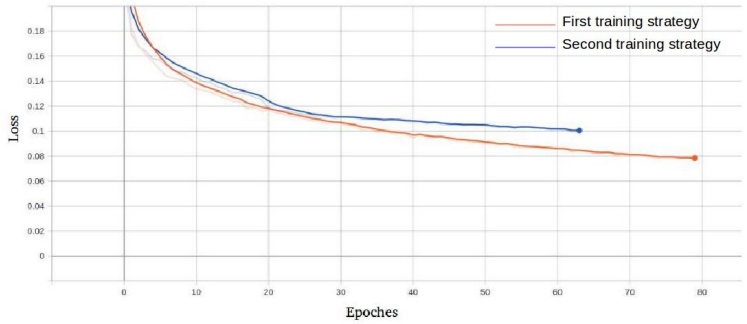


Figure 4-12: Masks binary cross-entropy training loss for the masks head

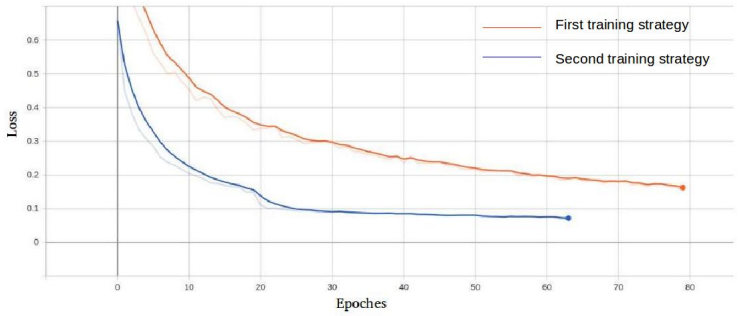


Figure 4-13: RPN bounding box training loss

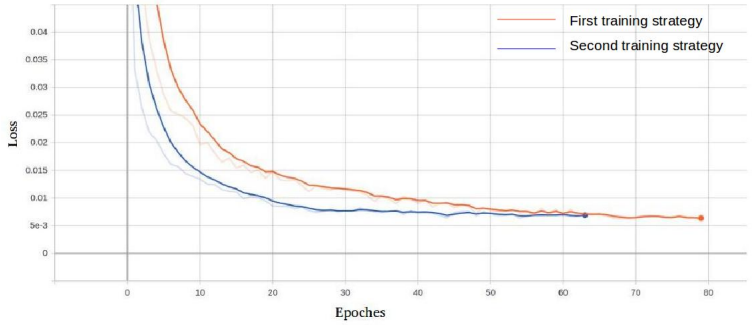


Figure 4-14: RPN anchor classifier training loss

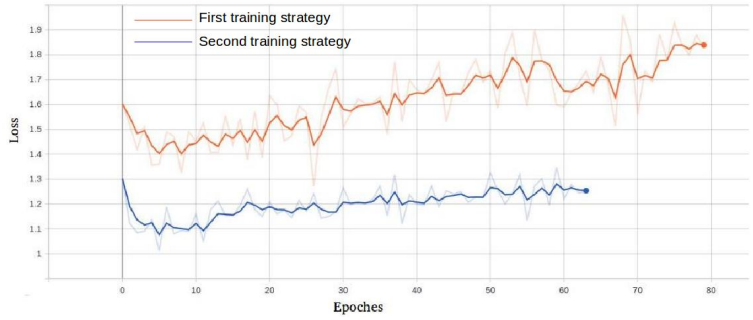


Figure 4-15: Validation loss

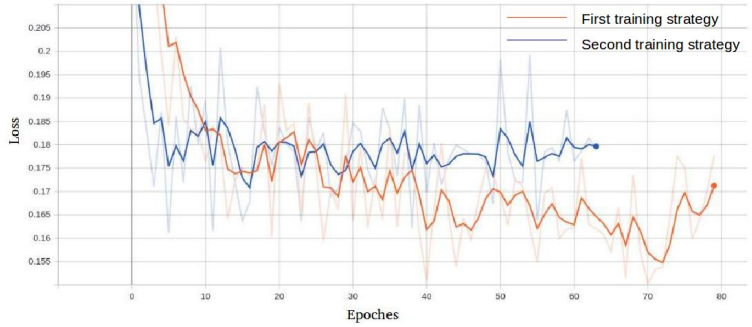


Figure 4-16: Validation loss for Mask R-CNN bounding box refinement

Figure 4-17: Validation loss for the classifier head of the Mask R-CNN

Figure 4-18: RPN bounding box validation loss

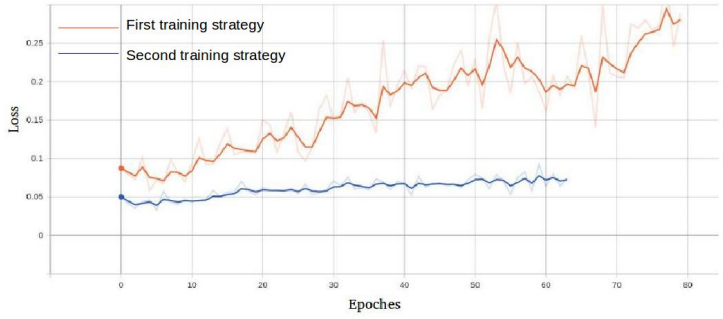


Figure 4-19: RPN anchor classifier validation loss

Description	Value
total images	345
training images	220
validation images	125
number of epoches	65
number of augmentation types	4
training steps per epoch	1000
validation steps at each epoch	120
anchors' size	16,64,128,256,512
non-max suppression threshold	0.7
learning rate	0.001, 0.0001
dropout rate	0.3

Table 4-4: Mask R-CNN second training strategy parameters

training strategy, which are made to prevent the model from overfitting. The main difference between the two training strategies is the RPN bounding box loss. The change of anchors' size to include smaller tomato parts may help. In this direction, the decrease of the non-max suppression threshold and the increase of the proposed training Region of Interest (RoI)s per image in a future training strategy can decrease more this kind of loss, which is the majority of the total validation loss. The increased validation loss for the classifier head is due to the confidence score of some overlapping detected parts of mostly the real tomatoes and robots, which have confidence score lower than 0.7.

The best weights for each training strategy are selected in order to evaluate the detection performance in a test dataset of 40 images. For the first training strategy, the weights of the 27th epoch are chosen, as the training loss is 0.5047, and the validation loss in 1.271. For the second training strategy, the weights of the 39th epoch are selected, as the training loss is 0.2918, and the validation loss is 1.12.

The precision-recall curve for the test dataset for the first training strategy is shown in Figure 4-20. The achieved average precision is 0.6725.

The instance segmentation results in images of the train, validation and test dataset are shown in the Figures 4-21, 4-22, and 4-23 correspondingly. At the first image of Figure 4-21, all the parts of the ground-truth tomatoes are detected with a high confidence. Only the bottom right tomato is falsely detected, as it is not annotated as a tomato in the dataset because it is considered not healthy. However, the confidence score for this is only 0.621, which can be excluded with a detection threshold of 0.65 instead of 0.5. Also, the masks are quite accurate. Moreover, the weights have been selected according to a good combination between the training and the validation loss. At Figure 4-22, all the tomatoes are detected with a high confidence, even though some small masks' parts are missing. At Figure 4-23 almost all the tomato parts, except one in Figure 4-23-(c), are detected with a high confidence. Even in adjacent tomatoes, the instance segmentation is correct with high confidence. Also, in this dataset some parts of the masks are missed.

The precision-recall curve for the test dataset for the second training strategy is shown in Figure 4-24. The achieved average precision is 0.7403. This AP is higher than the corresponding of the first training strategy, and it verifies the improved performance of the second training

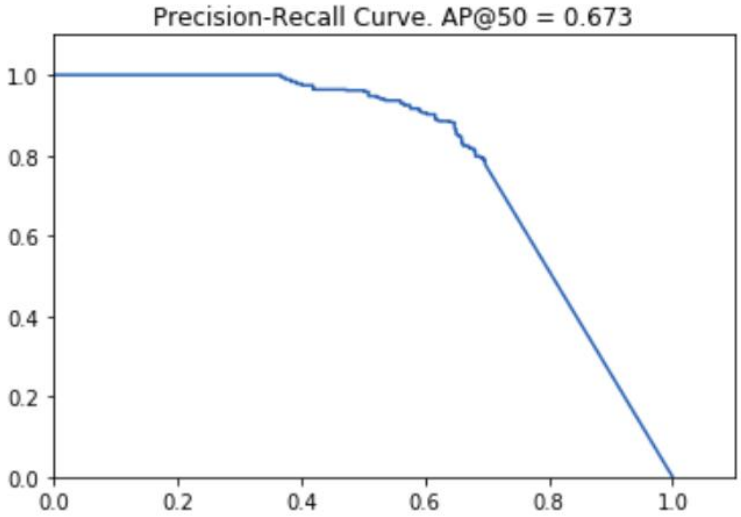


Figure 4-20: PR curve for test dataset using the first training strategy for Mask R-CNN

(a) Image 1 **(b)** Image 2

Figure 4-21: Instance segmentation results in images of the train dataset using the first training strategy

(a) Image 1 **(b)** Image 2

Figure 4-22: Instance segmentation results in images of the validation dataset using the first training strategy

strategy.

The instance segmentation results in images of the train, validation and test dataset are shown in the Figures 4-25, 4-26, and 4-27 correspondingly. The masks are quite accurate, and only some small parts are missing. Moreover, the weights have been selected according to a good combination between the training and the validation loss. At Figure 4-26-(a), only a shadowed part of tomato is not detected. The other parts are detected with high confidence. At Figure 4-27, almost all the tomato parts are detected with high confidence. Even in adjacent tomatoes, the instance segmentation is correct with high confidence. The instance segmentation of adjacent tomatoes is one important reason for training the Mask R-CNN. Also, in this dataset, some parts of the masks are missing, and there are some overlappings with low confidence though. Moreover, a small part of a tomato, that is missed in Figure 4-23-(c), is detected in Figure 4-27-(e).

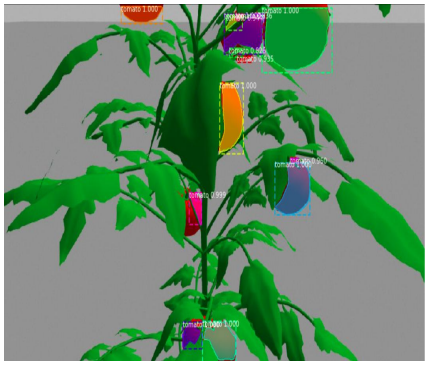
4-2-4 Occlusion Modelling

The viewpoint optimization method aims to find a position for the camera where a specific tomato is visible as more as possible. This technique is implemented by a Deep Reinforcement Learning (DRL) framework, which uses a reward that has to be maximized through the agent's actions. This reward must be related to the visibility of the tomato target, which can be occluded by leaves, branches, and other tomatoes. The number of visible pixels is a good indication of the tomato's visibility. However, this metric is not completely representative, as the size of the tomatoes differs, or the tomatoes are located at a different distance from the camera. In this way, a fraction of the visible pixels over the total pixels is used as the visual reward of the DRL framework. The visible fraction V_f is shown in Eq. (4-4).

$$V_f = \frac{V}{T} \tag{4-4}$$

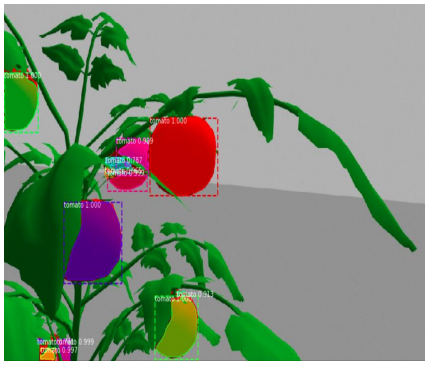
(a) Image 1

(b) Ground-truth masks for image 1

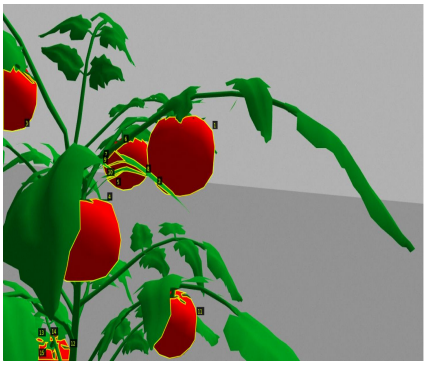


(c) Image 2

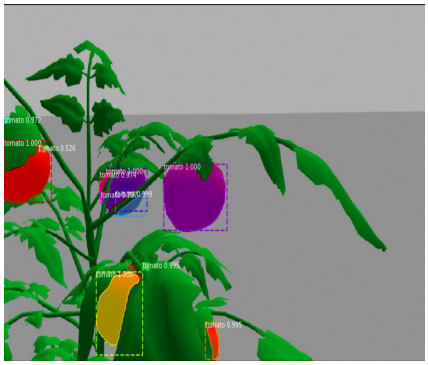
(d) Ground-truth masks for image 2



(e) Image 3



(f) Ground-truth masks for image 3



(g) Image 4

(h) Groundtruth masks for image 4

Figure 4-23: Ground-truth masks and instance segmentation results in images of the test dataset using the first training strategy

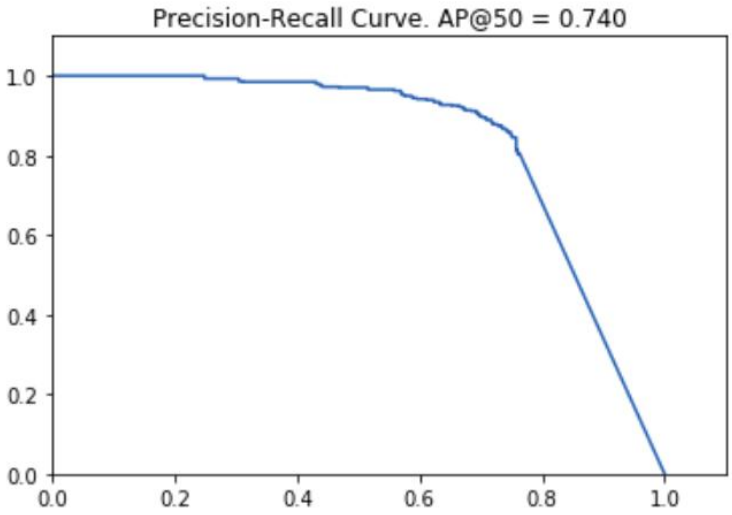
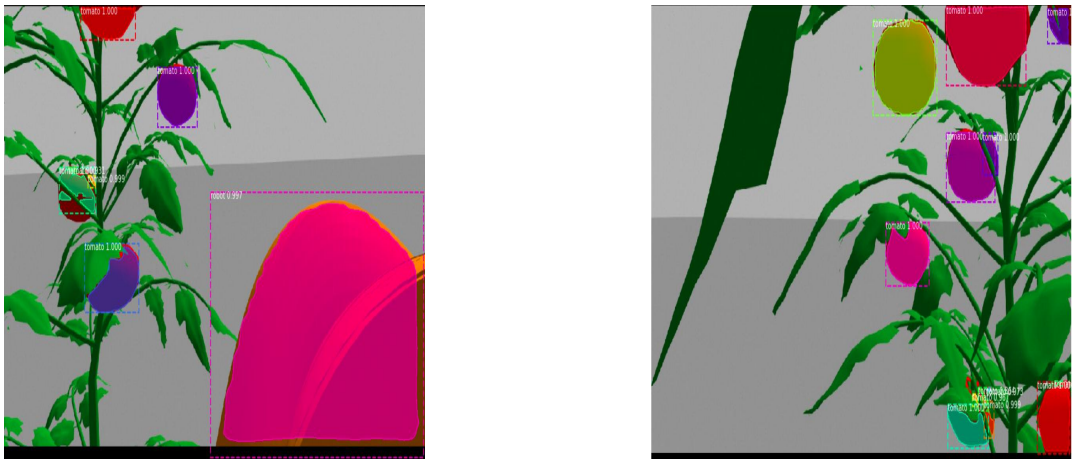


Figure 4-24: PR curve for test dataset using the second training strategy for Mask R-CNN

(a) Image 1 **(b)** Image 2

Figure 4-25: Instance segmentation results in images of the train dataset using the second training strategy



(a) Image 1 **(b)** Image 2

Figure 4-26: Instance segmentation results in images of the validation dataset using the second training strategy

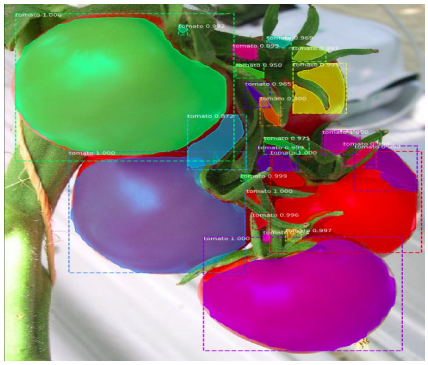
where V is the number of the visible pixels of a tomato, and T is the number of the total pixels of that tomato. The problem arises in the computation of T. This subsection deals with the whole tomato's 2D surface approximation, using only a single view image and the depth information instead of multiple view images, and then the computation of the visual reward.

The extraction of a mask for each visible tomato part is one reason for using the Mask R-CNN. In this way, the number of the visible pixels of a tomato can be easily computed using the detected masks. The following method is developed for the shape completion of the detected tomatoes and the computation of their visual reward. Some of the method's functions are implemented using OpenCV libraries.

- An RGB image I_{rgb} is obtained by the camera mounted on the robot's end-effector. This image is shown in Figure 4-28-(a).
- The masks at the I_{rgb} are detected using the inference mode of the trained Mask R-CNN.
- The detected tomato masks are combined in a binary image I_{tom_masks}. This image is shown in Figure 4-28-(b).
- A binary image I_{leaves} is created by obtaining the positive difference between the green and the red channels of I_{rgb}. In this way, a binary image, which contains the leaves, is created and is shown in Figure 4-28-(c).
- The non-zero pixels at I_{leaves} correspond to the leaves, and they are the mask which is to be inpainted in I_{rgb}. The inpainting [44] result is the image $I_{inpaint}$ which is shown in Figure 4-28-(d). This method helps to combine small parts of the same tomato, and approximate the tomato area behind the leaves.
- A binary image I_{bin} is obtained by setting the pixel values to 255 where the value of the red channel is larger than the value of the green channel of $I_{inpaint}$. Then, morphological

(a) Image 1

(b) Ground-truth masks for image 1



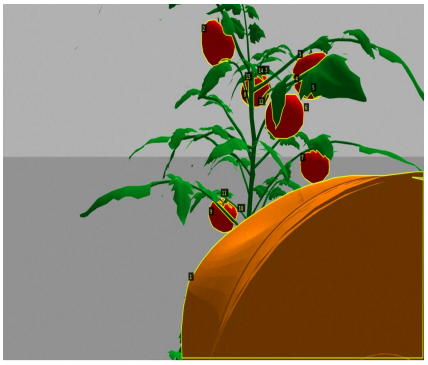
(c) Image 2

(d) Ground-truth masks for image 2

(e) Image 3

(f) Ground-truth masks for image 3

(g) Image 4



(h) Ground-truth masks for image 4

Figure 4-27: Ground-truth masks and instance segmentation results in images of the test dataset using the second training strategy

operations, such as erosion and dilation, are applied to remove the noisy pixels and fill holes. Possible background pixels, which are created by dilation and remain after the last erosion operation, are removed using the depth information. I_{bin} is shown in Figure 4-28-(e)

- The contours of I_{bin} are found, and they are drawn on I_{rgb} like it is shown in Figure 4-28-(f). These contours are not representative of the tomato shape. Thus, an ellipse is fitted on each contour using a least-square approach [45]. The fitted ellipses on I_{rgb} are shown in Figure 4-28-(g).
- The fitted ellipses contain also background pixels. The contours cnt_{ell} of the fitted ellipses are found. Then, the background pixels, which are inside these contours, are filtered out using the depth information.
- New contours cnt_{new} are found, and they are drawn on I_{rgb}, as it is shown in Figure 4-28-(h).
- T is the number of white pixels for each of the cnt_{new}. V is the number of pixels that are both white in cnt_{new} and I_{tom_masks}. The visible fraction of a tomato is computed according to Eq. (4-4), and plotted on the I_{rgb} in Figure 4-28-(h).

4-3 Thesis Method

4-3-1 Task Description

The goal of the MSc Thesis's experiments is to train a robot to obtain a pose in which the camera, mounted on the robot's end-effector, has almost full visibility of a tomato target. Two training strategies are implemented whose details are presented in Chapter 5. The position Cartesian coordinates of the tomato target are determined at the beginning of each episode. In case that no tomato is detected at the first step, then the tomato target is selected at the next steps of the episode until one is found. At the beginning of each episode, the position of the robot's end-effector is given by some ranges. Moreover, the orientation of the robot's end-effector is chosen in such a way that the camera, mounted on the robot's end-effector, looks at the tomato plant. The position and orientation ranges of the robot's end-effector for each training strategy are mentioned in Section 5-1 and Section 5-2.

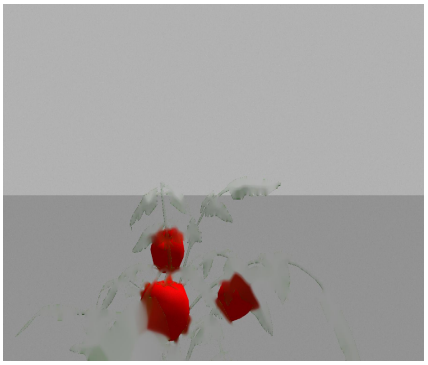
The Deep Deterministic Policy Gradient (DDPG) (subsection 3-5) is used for the training of the robot. The observations of the Reinforcement Learning (RL) framework are the position (x_{ee}, y_{ee}, and z_{ee}) and orientation ($roll_{ee}$, $pitch_{ee}$, and yaw_{ee}) of the robot's end-effector, and the position of the tomato target (x_t, y_t, and z_t). All of these coordinates are with respect to the world coordinate frame. The pose of the robot's end-effector is used as observations contrary to the robot's joints so that the trained policy is not dependant on the robot's configuration and can be used on other autonomous agents such as drones. Moreover, there can be singularity problems in the case of using the robot's joints as observations. The actions are incremental values for the position and the orientation of the robot's end-effector.

The reward consists of the visual reward R_v and penalties P_p due to the robot's end-effector position Eq. (4-5). The visual reward R_v is related to the visible fraction of the tomato as it

(a) I_{rgb} image

(b) I_{tom_masks} image

(c) I_{leaves} image



(d) $I_{inpaint}$ image

(e) I_{bin} image

(f) Contours of I_{bin} drawn on I_{rgb}

(g) Fitted ellipses drawn on I_{rgb}

(h) Final contours after removing background pixels of ellipses

Figure 4-28: Images created at each step of the occlusion modelling algorithm

is given in Eq. (4-4). The formulation of the visual reward by using the number of the visible pixels of a tomato mask is the reason for using Mask R-CNN in contrast to YOLOv3. The visual reward R_v and the position penalties P_p are defined differently in each of the training strategies, and are described in detail in Section 5-1 and Section 5-2.

$$R = R_v + P_p \tag{4-5}$$

4-3-2 Actor-Critic Network Architecture

The critic-network has two hidden densely-connected neural network layers, with 100 units in each layer, a RELU activation function is used, and a He initializer for the kernel weights matrix [46]. The input of the first hidden layer is only the state. Then, the output of this hidden layer is merged with the action. This concatenation is the input to the second hidden layer. The output of the critic network has a linear activation function, and the layer's initial weights are generated with a uniform distribution. An Adam optimizer is used.

The actor-network has also two hidden densely-connected neural network layers, with 100 units in each layer, a RELU activation function is used, and a He initializer for the kernel weights matrix [46], such as the hidden layers of the critic-network. The output of the actor network has a tanh activation function, and a Glorot initializer for the kernel weights matrix [47]. One reason for using the tanh activation function is that the actions are incremental values, so there is not the desire for changes over 1m and 1 rad correspondingly.

4-3-3 Training Algorithm

The developed training algorithm for the tomato viewpoint optimization is based on 1 and is presented in Algorithm 2. The initialization ranges, the exploration method, the reward function, and the full visibility threshold differ in the two training strategies. The first strategy uses an ϵ-greedy exploration, while the second uses an additive Gaussian noise to the current policy. In the first case, the threshold to consider a tomato fully visible is 0.85, and 0.8 in the other as the tomato plant is more cluttered.

4-3-4 ROS Implementation

The software for the MSc Thesis project is developed using ROS. ROS is a meta-operating system for robots, which means that is built on top of another operating system and provides the services, tools, and libraries for the development of autonomous robots' functionalities. Packages are used to organize ROS software. Two already existed and two newly created packages are used for this project. One package is responsible for the simulation environment in Gazebo, as it contains the URDF files of the camera and the robot, the camera plugins and meshes, and the world file. The second package is the KUKA ROS meta-package [39], which contains tools to simulate and interact with the robot. The third package executes the instance segmentation with Mask R-CNN, which is based on this Github repository. The fourth package implements the DRL algorithm and the interaction of the agent with Gazebo and ROS.

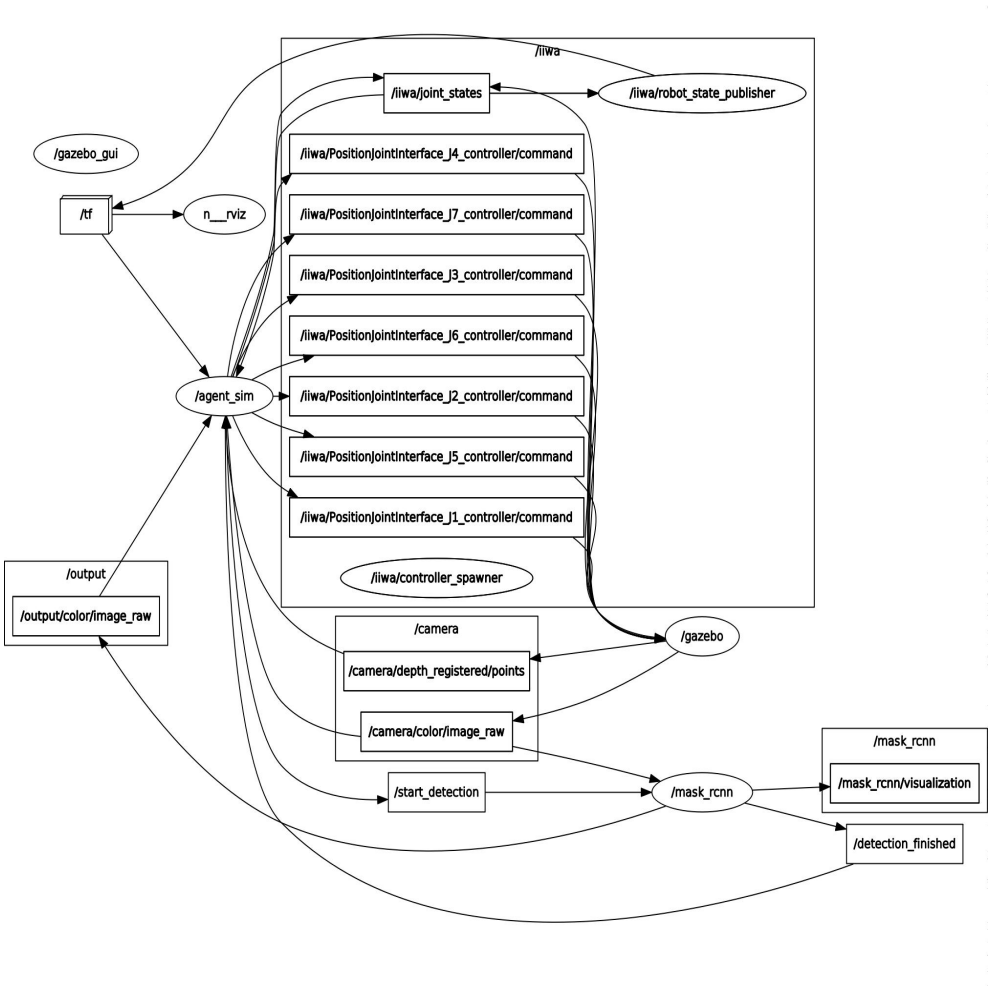


Figure 4-29: ROS nodes and topics that are developed for the thesis project

A running ROS program is called a node. ROS nodes communicate with ROS messages. Topics are used in order to transfer message information between the nodes. More specifically, a node can provide information by publishing messages on the appropriate topics. Another node, which wants to obtain information, subscribes to the suitable topics. The developed nodes and topics are shown in Figure 4-29. The ellipsoid schemes represent the nodes, while the rectangular schemes illustrate the topics.

The "/mask_rcnn" node subscribes to the topic "/camera/colour/image_raw" to obtain the RGB image message of the sensor. Also, it subscribes to the topic "/start_detection" to receive a signal about starting the instance segmentation of the obtained image. After the detection finishes, the node "/mask_rcnn" publishes the instance segmentation result on the topics "/output/colour/image_raw", and "/mask_rcnn/visualization" to visualize the result in Rviz. Also, this node publishes a string message on the topic "/detection_finished" to inform the node "/agent_sim" that the detection finished.

The "agent_sim" node subscribes to five topics. It subscribes to the topic "/iiwa/joint_states" to obtain the current joints' angle values. Also, it subscribes to the topic "/camera/colour/image_raw" to obtain the current RGB image seen by the sensor. Moreover, it subscribes to

the topic "/camera/depth_registered/points" to create the point cloud for the RGB image. Furthermore, it subscribes to the topic "/output/colour/image_raw" to take the detection result of the "/mask_rcnn" node. Finally, it subscribes to the topic "/detection_finished" to start the occlusion modelling algorithm when the detection finishes. On the other hand, the "agent_sim" node publishes on the topic "start_detection" to inform the node "/mask_rcnn" to start the detection when the robot's movement for the step has been completed. This node also publishes on the topics for each of the seven robot joints to move them with the appropriate angle and also publishes on the topic "/iiwa/joint_states" to update the robot state. Then, the node "/iiwa/robot_state_publisher" subscribes to the topic "/iiwa/joint_states" to take the robot state and publishes it to "/tf", which contains all the information about the transformations between the robot's frames. The node "agent_sim" uses "/tf" to obtain the current pose of the end-effector.

At the beginning of each episode, the robot's end-effector pose is determined by the position and orientation values. Moreover, the robot's end-effector pose is calculated at each step using the taken actions. However, the robot moves by publishing values on the topics of its seven joints. In this way, there is a need for an inverse kinematics solution. This is achieved by the package "trac_ik" [48], which provides an alternative inverse kinematics solver to the inverse Jacobian methods in Kinematics and Dynamics Library (KDL). More specifically, "trac_ik" outperforms KDL in the presence of joint limits and provides the capability to set independent error bounds for each Cartesian coordinate.

4-3-5 Thesis Algorithm Overview

The flowchart of the developed algorithm for the first training strategy is shown in Figure 4-30. The flowchart for the second training strategy is not presented as there are a few changes that are already discussed in the corresponding sections. Some algorithm blocks are illustrated separately because it is not feasible to fit the whole algorithm in an image.

The instance segmentation block implements instance segmentation on the current image using the already trained Mask R-CNN weights. The instance segmentation result includes for each detected mask: the bounding box in pixels, the class name, the detection confidence, an identity number, and the mask in an image form. The occlusion modelling block is shown in Figure 4-31, and the step block is shown in Figure 4-32.

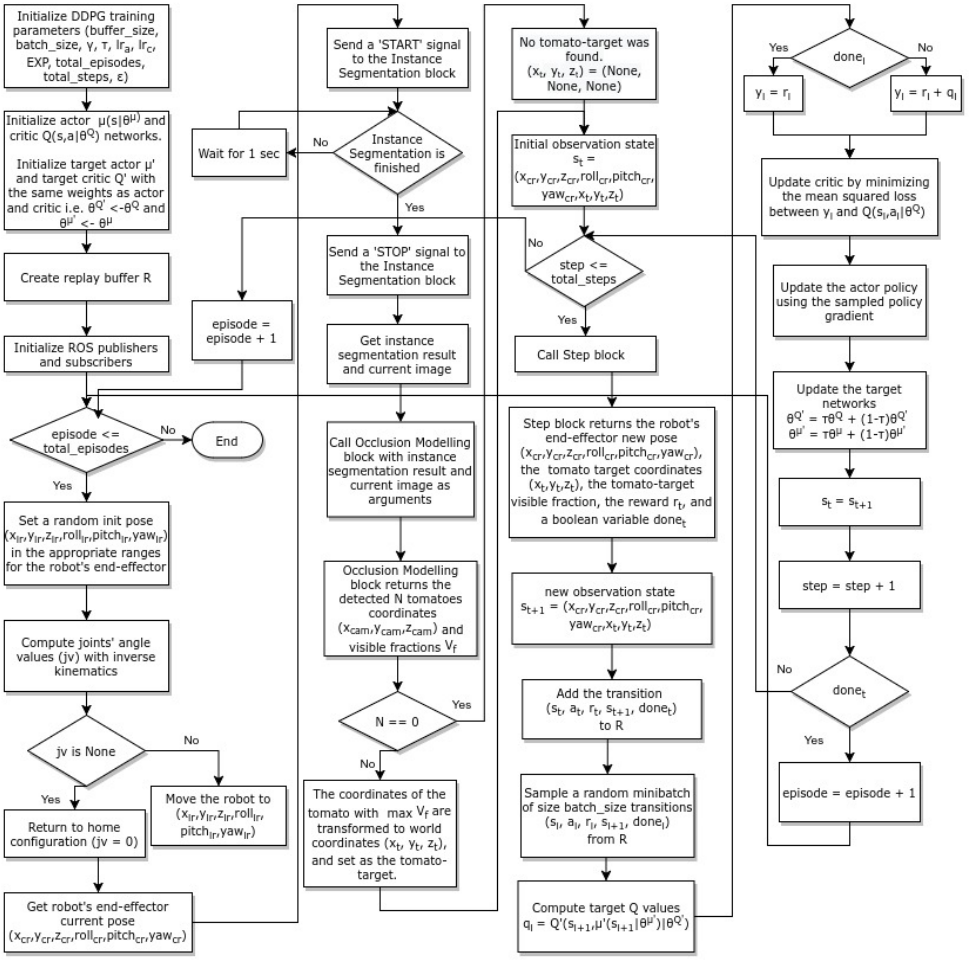


Figure 4-30: Main algorithm flowchart

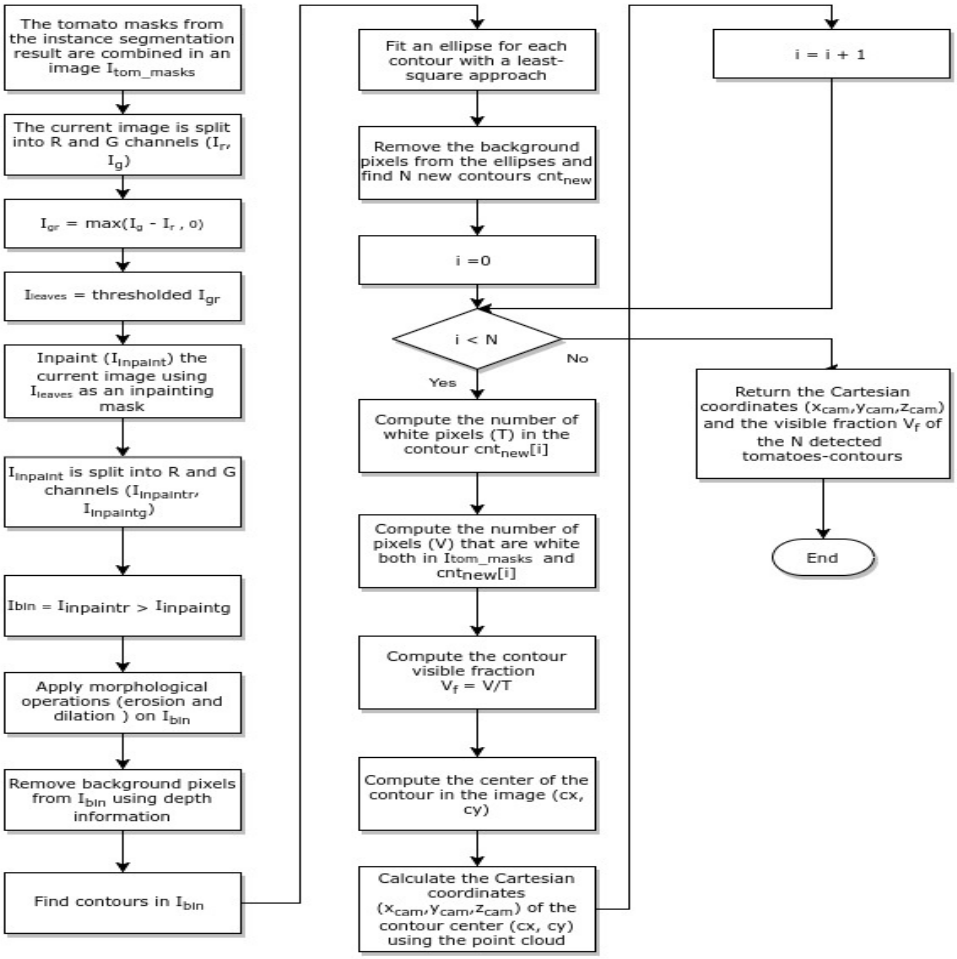


Figure 4-31: Occlusion modelling block flowchart

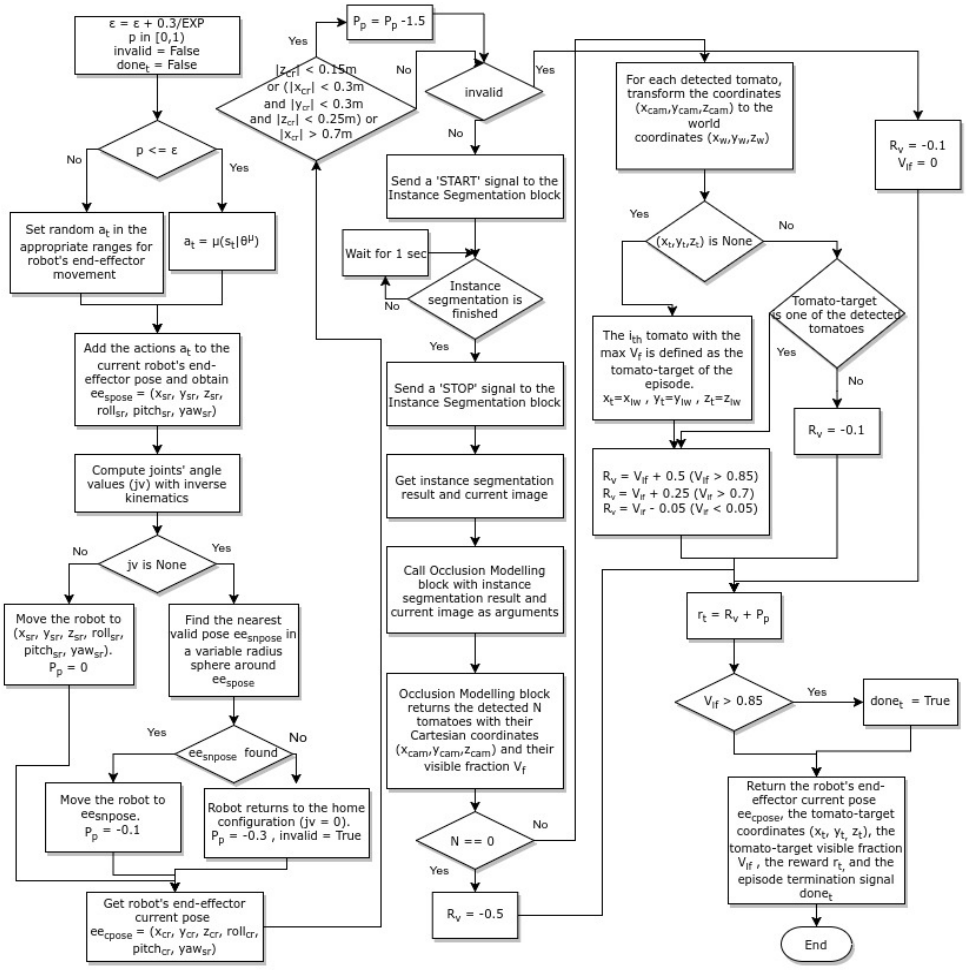


Figure 4-32: Step block flowchart

Algorithm 2: DDPG training for active perception

Initialize the ROS/Gazebo environment with the suitable nodes, publishers (joints' movements and state update, signals for the detection state, instance segmentation result, and visualization), and subscribers (current joints' angle values, signals for detection state, camera image, point cloud, and instance segmentation result).
Initialize training parameters.
Randomly initialize critic network $Q\left(s, a \mid \theta^{Q}\right)$ and actor $\mu\left(s \mid \theta^{\mu}\right)$ with weights θ^{Q} and θ^{μ}.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^{Q}, \theta^{\mu'} \leftarrow \theta^{\mu}$.
Initialize replay buffer R.
for *episode* $= 1, M$ **do**
  Move randomly the robot's end-effector according to (Eq. (5-1),Eq. (5-2)) or (Eq. (5-7),Eq. (5-8)).
  When the movement is done, obtain the robot's end-effector pose, and send a signal to start instance segmentation.
  Determine the position of the tomato target.
  Store the robot's end-effector pose and the tomato's position as the initial observation state s_1.
  done = False
  for $t = 1, T$ **do**
    Select action a_t according to either exploitation or exploration, based on either a ϵ-greedy algorithm or an additive Gaussian noise.
    Execute action a_t, obtain the robot's end-effector new pose, and send a signal to start instance segmentation.
    Calculate reward r_t according to (Eq. (4-5), Eq. (5-3), Eq. (5-4)) or (Eq. (4-5), Eq. (5-9), Eq. (5-10)).
    if *visible fraction* $>$ *full visibility threshold* **then**
      done = True
    Store the robot's end-effector pose and the tomato's position as the new observation s_{t+1}.
    Store transition $(s_t, a_t, r_t, s_{t+1}, done)$ in R.
    Sample a random minibatch of N transitions $(s_i, a_i, r_i, s_{i+1}, done_i)$ from R.
    if $done_i$ **then**
      $y_i = r_i$
    else
      $y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'\left(s_{i+1} \mid \theta^{\mu'}\right) \mid \theta^{Q'}\right)$
    end
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i \left(y_i - Q\left(s_i, a_i \mid \theta^{Q}\right)\right)^2$.
    Update the actor policy using the sampled policy gradient:
    $\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_i \nabla_a Q\left(s, a \mid \theta^{Q}\right)\Big|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu\left(s \mid \theta^{\mu}\right)\Big|_{s_i}$
    Update the target networks: $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'} \quad \theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$
    if *done* **then**
      break
  end
end

Chapter 5

Experimental Results

5-1 Guided-Reward Training and Evaluation

5-1-1 Training

Training Setup

The tomato plant in Figure 4-1 is modified using https://www.blender.org/, so one tomato with the overlapping leaves is obtained. This modified plant is used for training and some evaluations. At the beginning of each episode, the position of the robot's end-effector is given by the ranges in Eq. (5-1). The orientation is given by the ranges in Eq. (5-2). These values are with respect to the world coordinate frame and follow the right-hand rule. The robot returns to its home configuration, i.e. all joint angle values equal to 0, when an action leads the robot out of its workspace.

$$\begin{aligned} x_{ee} &\in [-0.1\ m, 0.2\ m] \\ y_{ee} &\in [-0.35\ m, 0.1\ m] \\ z_{ee} &\in [0.65\ m, 1.15\ m] \end{aligned} \tag{5-1}$$

$$\begin{aligned} roll_{ee} &= 0 \\ pitch_{ee} &\in [0.45\pi\ rad, 0.7\pi\ rad] \\ yaw_{ee} &\in [0\ rad, 0.25\pi\ rad] \end{aligned} \tag{5-2}$$

The visual reward R_v is highly related to the visible fraction of the tomato and gains some extra rewards or penalties as it is shown in Eq. (5-3). Some extra penalties P_p are added to the R_v according to the robot's end-effector position, and they are shown in Eq. (5-4). The reward values are not too high to not cause numerical instability during training the critic network. Moreover, this reward shaping contributes to a guided exploration during training.

$$R_v = \begin{cases} V_f & \text{if the tomato target is detected} \\ V_f + 0.5 & \text{if the tomato target is detected and is fully visible}(V_f > 0.85) \\ V_f + 0.25 & \text{if the tomato target is detected and is highly visible}(V_f > 0.7) \\ V_f - 0.05 & \text{if the tomato target is detected, but is highly occluded}(V_f < 0.15) \\ -0.5 & \text{if no tomato is found} \\ -0.1 & \text{if the tomato target is not detected in the step} \\ -0.1 & \text{if the taken action leads the robot out of workspace, so the camera is not at the desired position} \end{cases} \tag{5-3}$$

$$P_p = \begin{cases} -0.3 & \text{if the pose is out of the workspace} \\ -0.1 & \text{if the pose is out of the workspace, but the robot went to a neighbour pose} \\ -1.5 & \text{if } (z_{ee} < 0.15\ m) \textit{ or } (|x_{ee}| < 0.3\ m \textit{ and } |y_{ee}| < 0.3\ m \textit{ and } z_{ee} < 0.25\ m\ ) \textit{ or } (x_{ee} > 0.7\ m) \end{cases} \tag{5-4}$$

The training constant parameters are shown in Table 5-1:

Description	Symbol	Value
discount factor	γ	0.99
buffer size	$\|R\|$	1000
batch size	N	32
target network hyperparameter	τ	0.001
actor learning rate	lr_a	0.0001
critic learning rate	lr_c	0.001
number of episodes	M	240
number of steps	T	30
exploration parameter	EXP	1200

Table 5-1: First DRL training constant parameters

The choice of action according to exploitation, in contrast to exploration, is based on a ϵ-greedy algorithm. A variable ϵ has an initial value of 0.3 and increases in each step according to Eq. (5-5).

$$\epsilon = \epsilon + \frac{0.3}{EXP} \tag{5-5}$$

Thus, at the beginning of the training, it is more probable to choose the action with exploration, while in the end with exploitation, i.e. the policy $\mu_\theta(s)$. In the case that the action is chosen with exploration, then the actions related to the position and orientation of the end-effector obtain values in the range given by Eq. (5-6). This shows a more guided exploration.

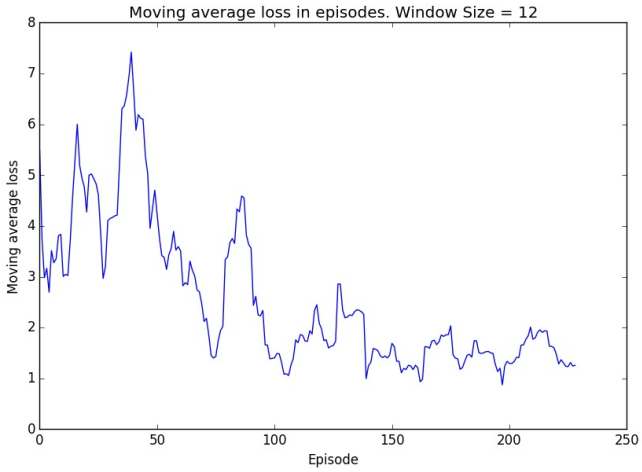


Figure 5-1: Moving average episode loss for the first DRL training strategy

$$\begin{aligned} \Delta x &\in [-0.07\ m, 0.1\ m] \\ \Delta y &\in [-0.2\ m, 0.1\ m] \\ \Delta z &\in [-0.1\ m, 0.1\ m] \\ \Delta pitch &\in [-0.17\ rad, 0.17\ rad] \\ \Delta roll &\in [-0.17\ rad, 0.43\ rad] \\ \Delta yaw &\in [-0.17\ rad, 0.26\ rad] \end{aligned} \tag{5-6}$$

Training Plots

The training procedure lasts around 9 hours. The final number of steps is 1278, and the value of ϵ is almost 0.6195. The training performance is presented by plotting the episode moving average rewards and losses, and the step moving average rewards and tomato visible fractions. The reward refers to the total reward as given by Eq. (4-5), and the loss refers to L loss in Algorithm 2. The episode loss is the sum of losses at all the steps of the episode. The moving average loss for each episode, using a window's size equal to 12, is shown in Figure 5-1. The moving average reward for each episode, using a window's size equal to 12, is shown in Figure 5-2. The moving average reward for each step, using a window's size equal to 75, is shown in Figure 5-3. The moving average visible tomato fraction for each step, using a window's size equal to 75, is shown in Figure 5-4. Figure 5-1 shows that the moving average episode loss is decreasing to a value close to 1. The moving average episode reward has values around 1, which are explained by the equivalent step reward when an episode terminates because of a fully visible tomato target. Finally, the moving average step reward and tomato visible fraction increase considerably in a similar way, as they are highly related.

Figure 5-2: Moving average episode reward for the first DRL training strategy

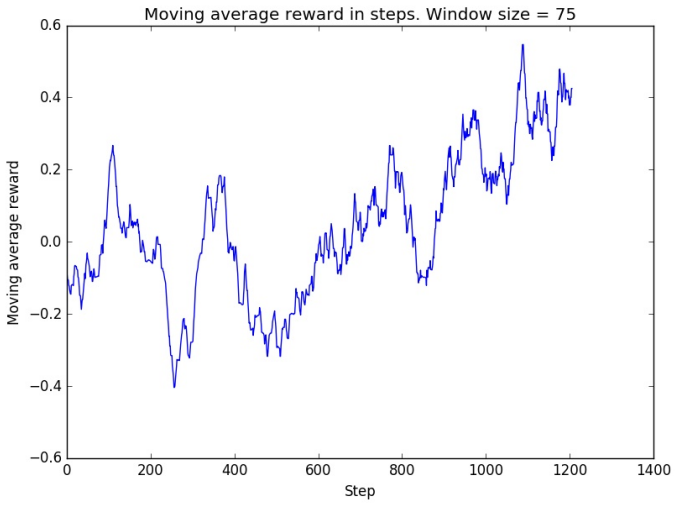


Figure 5-3: Moving average step reward for the first DRL training strategy

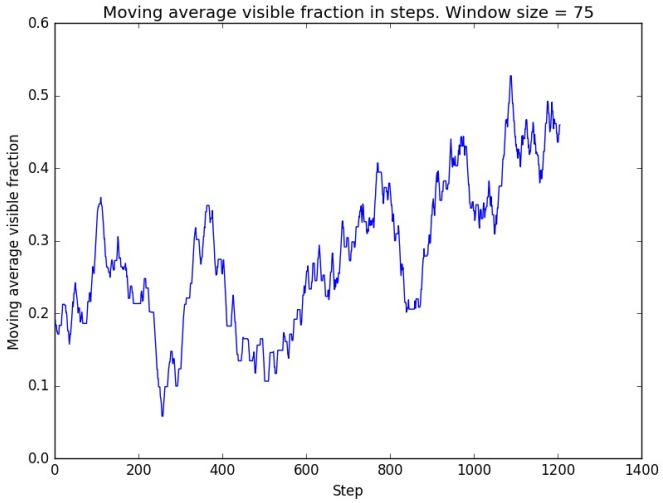


Figure 5-4: Moving average step visible tomato fraction for the first DRL training strategy

5-1-2 Evaluation

The evaluation of the training procedure occurs by running the training algorithm without exploration noise and network update. The trajectories of the robot's end-effector, the steps needed per episode, the maximum visible fraction per episode, and the initial and final tomato-target viewpoint are shown. The plant in Figure 4-1, with one and three tomatoes, is used for evaluation. The length of the evaluation is 6 episodes. An episode consists of 30 steps in case that it does not terminate earlier. Moreover, the plant is placed to a different pose than the training pose.

Evaluation in One-Tomato Plant

The position of the tomato plant is shown in Figure 5-5. Also, the steps, that an episode needs to terminate, are mentioned in Figure 5-6. Moreover, the maximum visible tomato-target fraction in each episode is presented in Figure 5-7. Furthermore, the trajectories of the robot's end-effector for three episodes are shown with three different colours in Figure 5-8. The opaque camera corresponds to the pose at the beginning of the episode, the transparent cameras to the intermediate steps, and the semi-transparent camera to the final pose where the tomato is fully visible. Finally, the viewpoint of a tomato-target at the beginning and the end of an episode is illustrated in Figure 5-9. The figures indicate that the robot sees the tomato-target quickly with a high visible tomato fraction. Only in the last two episodes, it needs more steps as it reaches poses out of its workspace.

Evaluation in Three-Tomatoes Plant

The position of the tomato plant is shown in Figure 5-10. The steps, that an episode needs to finish, are plotted in Figure 5-11. Moreover, the maximum visible tomato fraction in each episode is presented in Figure 5-12. Finally, the viewpoint of a tomato-target at the beginning

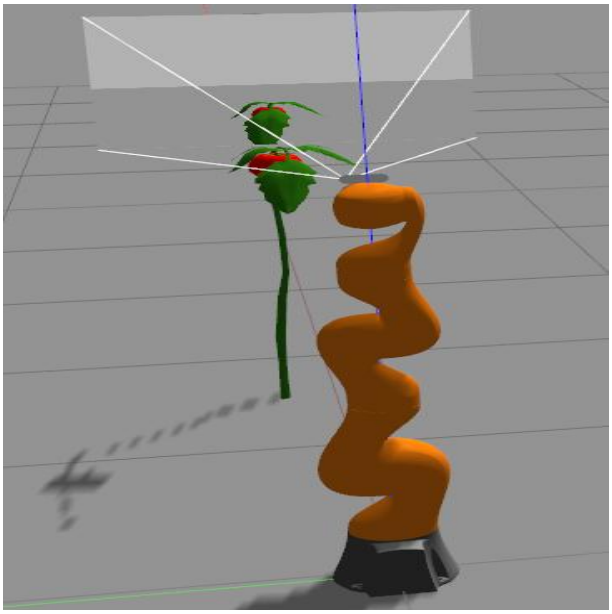


Figure 5-5: Position of the one-tomato plant for the training evaluation of the first DRL training strategy

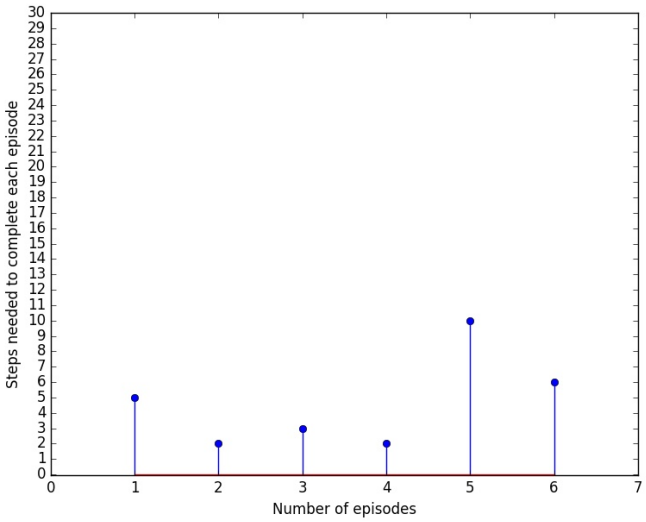


Figure 5-6: Number of steps needed for the termination of each episode in one-tomato plant first DRL training strategy evaluation

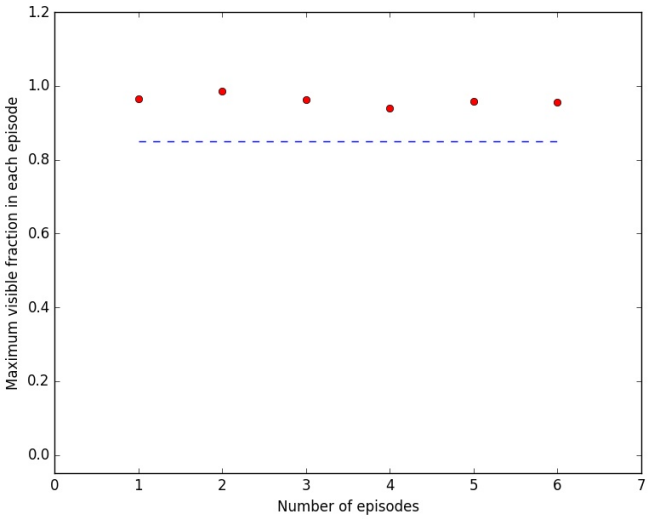


Figure 5-7: Maximum visible tomato-target fraction in each episode in one-tomato plant first DRL training strategy evaluation

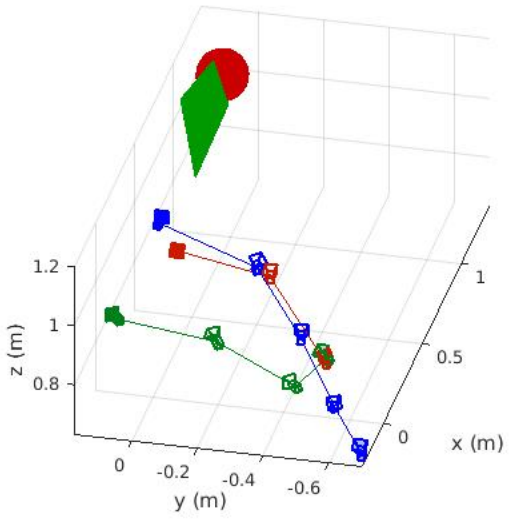
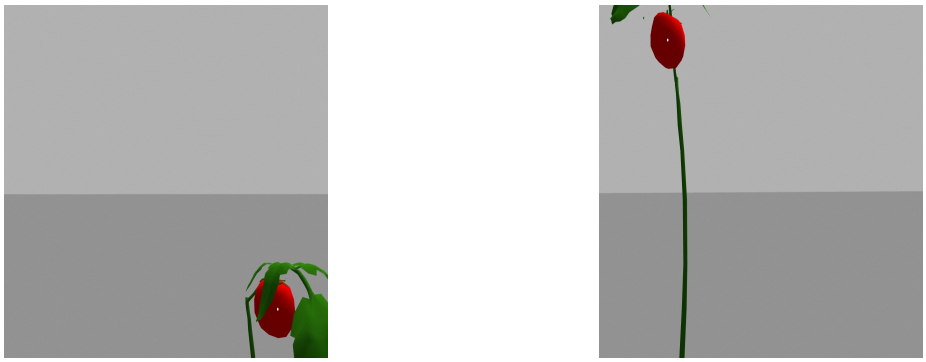


Figure 5-8: Robot end-effector's trajectories for three episodes in one-tomato plant first DRL training strategy evaluation



(a) Beginning of the episode **(b)** End of the episode

Figure 5-9: Tomato-target viewpoint at the beginning and the end of an episode in one-tomato plant first DRL training strategy evaluation

and the end of the episode is illustrated in Figure 5-13. The figures demonstrate that the robot can also fully see the tomato-target fast with a high visible tomato fraction in a plant with more tomatoes.

5-2 Sparse-Reward Training and Evaluation

5-2-1 Training

Training Setup

The tomato plant in Figure 4-1 is modified, and three tomatoes with the overlapping leaves are obtained. The modified plant is used for training and evaluation. The use of the three-tomatoes plant makes the training environment less deterministic in contrary to the first training strategy. At the beginning of each episode, the position of the robot's end-effector is given by the ranges in Eq. (5-7). The orientation is given by the ranges in Eq. (5-8). These initial values are selected in a more uniform range in comparison to the first training strategy. Moreover, the episode stops also when an action leads the robot out of the workspace.

$$\begin{aligned} x_{ee} &\in [-0.1\ m, 0.2\ m] \\ y_{ee} &\in [-0.25\ m, 0.25\ m] \\ z_{ee} &\in [0.65\ m, 1.15\ m] \end{aligned} \tag{5-7}$$

$$\begin{aligned} roll_{ee} &= 0 \\ pitch_{ee} &\in [0.45\pi\ rad, 0.7\pi\ rad] \\ yaw_{ee} &\in [-\frac{\pi}{8}\ rad, \frac{\pi}{8}\ rad] \end{aligned} \tag{5-8}$$

The visual reward R_v obtains more sparsely positive values at this training strategy, and it is shown in Eq. (5-9). Some extra penalties P_p are added to the R_v according to the robot's end-effector position, and they are shown in Eq. (5-10). The total reward R does not take

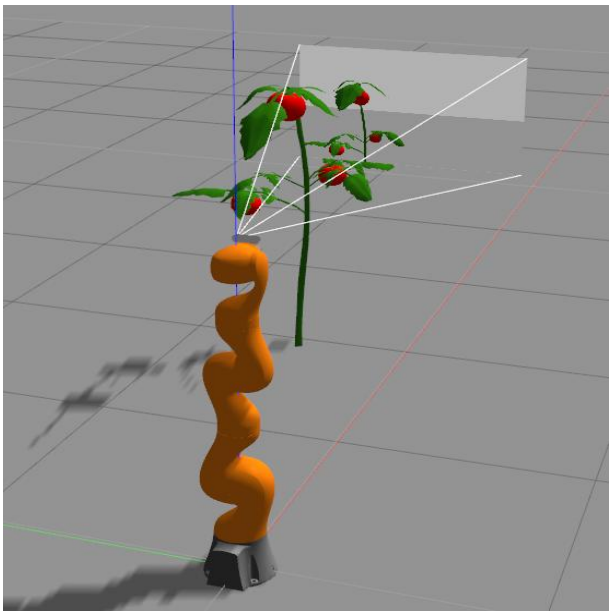


Figure 5-10: Position of the three-tomatoes plant for the training evaluation of the first DRL training strategy

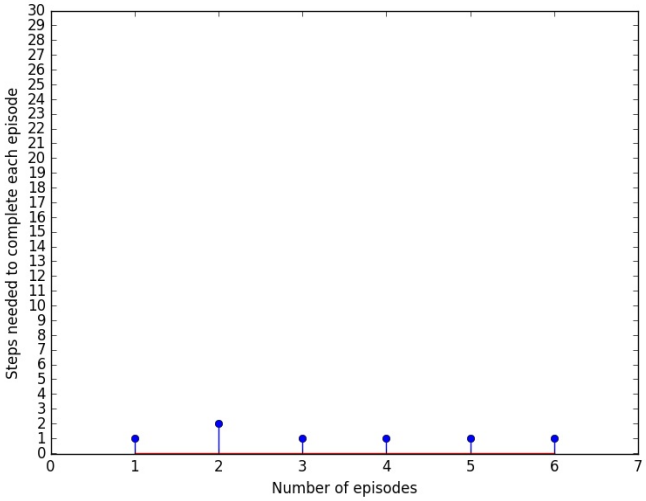


Figure 5-11: Number of steps needed for the termination of each episode in three-tomatoes plant first DRL training strategy evaluation

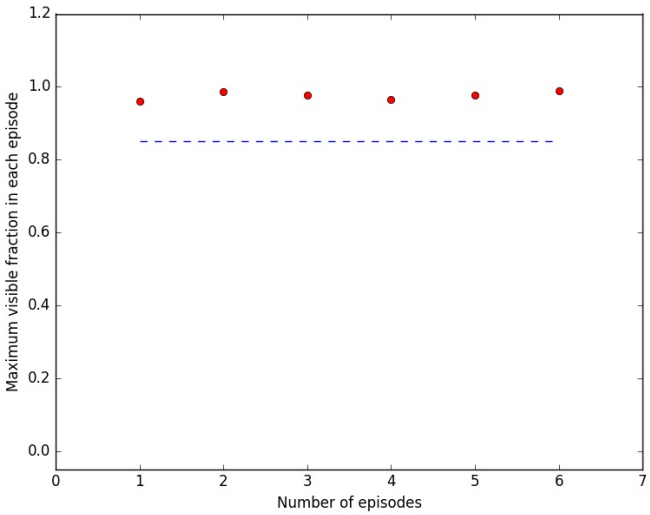
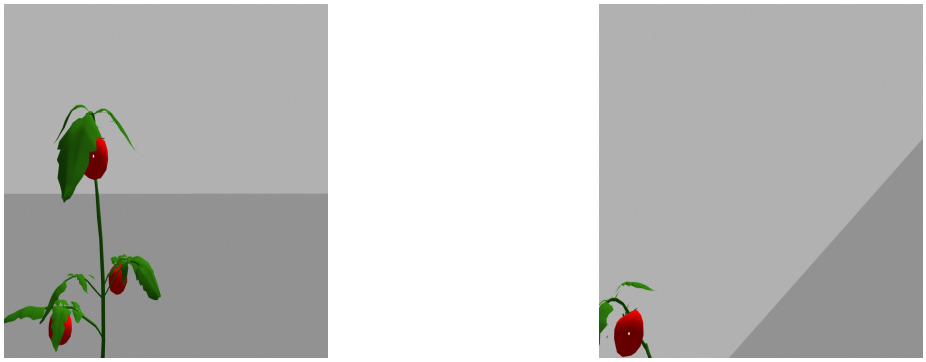


Figure 5-12: Maximum visible tomato-target fraction in each episode in three-tomatoes plant first DRL training strategy evaluation



(a) Beginning of the episode **(b)** End of the episode

Figure 5-13: Tomato-target viewpoint at the beginning and the end of an episode in three-tomato plant first DRL training strategy evaluation

values less than -1 when the robot reaches one of the positions in the second case in Eq. (5-10). This happens in order to clip rewards over |1|. Moreover, this reward formulation contributes to a less guided exploration during training.

$$R_v = \begin{cases} 1 & \text{if the tomato target is detected and is fully visible}(V_f > 0.8) \\ 0.1 & \text{if the tomato target is detected and is highly visible}(0.6 < V_f < 0.8) \\ -0.1 & \text{if no tomato is found} \end{cases} \tag{5-9}$$

$$P_p = \begin{cases} -1 & \text{if the pose is out of the workspace} \\ -1 & \text{if } (z_{ee} < 0.15\ m) \textit{ or } (|x_{ee}| < 0.3\ m \textit{ and } |y_{ee}| < 0.3\ m \textit{ and } z_{ee} < 0.25\ m\ ) \textit{ or} \\ & (x_{ee} > 0.7\ m) \end{cases} \tag{5-10}$$

The training constant parameters are shown in Table 5-2:

Description	Symbol	Value
discount factor	γ	0.99
buffer size	$\|R\|$	1500
batch size	N	32
target network hyperparameter	τ	0.001
actor learning rate	lr_a	0.0001
critic learning rate	lr_c	0.001
number of episodes	M	500
number of steps	T	30
exploration parameter	EXP	1800

Table 5-2: Second DRL training constant parameters

An additive Gaussian noise $N(0, 0.1)$ to each action, derived by the policy $\mu_\theta(s)$, is used for exploration purposes. The Ornstein-Uhlenbeck noise, which the original Deep Deterministic Policy Gradient (DDPG) paper [21] uses, is not preferred due to the complication and the absence of a significant effect on performance as the authors in [49] and [50] mention. The generated Gaussian noise is multiplied with a discount factor ϵ before the addition with the policy action, i.e. the final action is given by $\max(\epsilon, 0) \cdot N(0, 0.1) + \mu_\theta(s)$. The noise discount factor ϵ has an initial value equal to 1 and decreases using Eq. (5-11). In this way, there is not any exploration after some training episodes.

$$\epsilon = \epsilon - \frac{1.0}{EXP} \tag{5-11}$$

Training plots

The training procedure lasts around 22 hours. The exploration parameter ϵ becomes 0 in episode 238, so there is only exploitation after this episode. Also, the exploration terminates

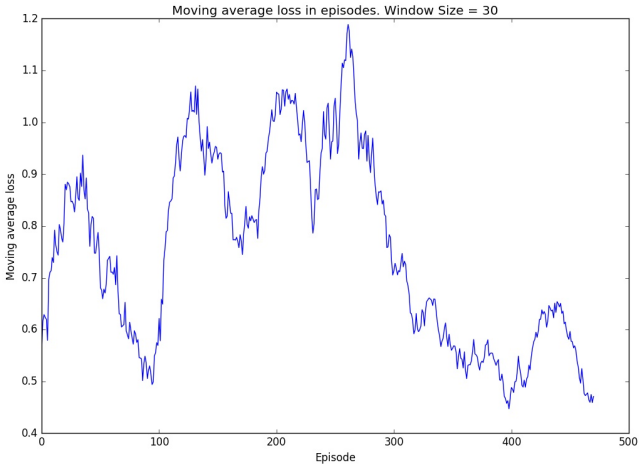


Figure 5-14: Moving average episode loss for the second DRL training strategy

at step 1800. The training performance is presented by plotting the episode moving average rewards and losses, and the step moving average rewards and tomato visible fractions. The reward refers to the total reward as given by Eq. (4-5), and the loss refers to L loss in Algorithm 2. The episode loss is the sum of losses at all the steps of the episode. The moving average loss for each episode, using a window's size equal to 30, is shown in Figure 5-14. The moving average reward for each episode, using a window's size equal to 30, is shown in Figure 5-15. The moving average reward for each step, using a window's size equal to 180, is shown in Figure 5-16. The moving average visible tomato fraction for each step, using a window's size equal to 180, is shown in Figure 5-17. Figure 5-14 shows that the moving average episode loss is ending up to values between 0.5-0.6. The moving average episode reward obtains its minimum value when the exploration terminates, i.e. at around episode 238, and then it increases without acquiring stable values. Furthermore, the moving average step tomato visible fraction behaves in the same way as the moving average episode reward after the end of the exploration at episode 1800. Finally, the moving average step reward presents unstable values, probably due to a lot of out of workspace poses of the robot. These plots show that the training performance is not stable. Also, the increase of the loss and the decrease of the reward after the 400th episode indicates that the agent does not learn anything useful after it.

5-2-2 Evaluation

The evaluation of the training procedure occurs by running the training algorithm without exploration noise and network update. The steps needed per episode, the maximum visible fraction per episode, and the initial and final tomato-target viewpoint are illustrated. The plant in Figure 4-1, with three tomatoes, is used for evaluation.

The length of this evaluation is 6 episodes. An episode consists of 30 steps in case that it does not terminate earlier. The position of the tomato plant is shown in Figure 5-18 and is different from the training position. Moreover, the steps, that an episode needs to terminate,

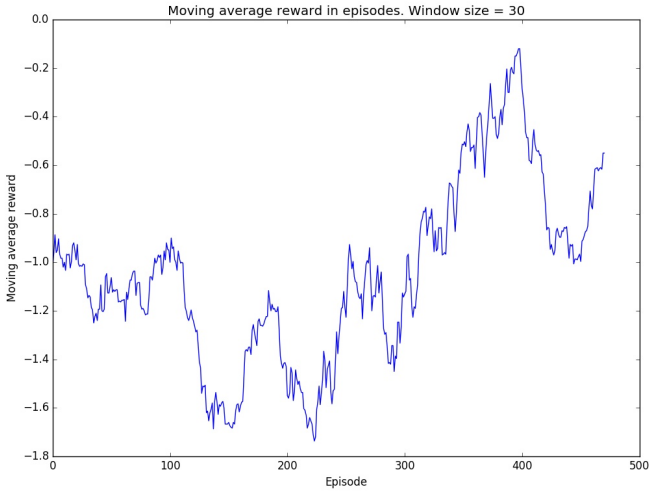


Figure 5-15: Moving average episode reward for the second DRL training strategy

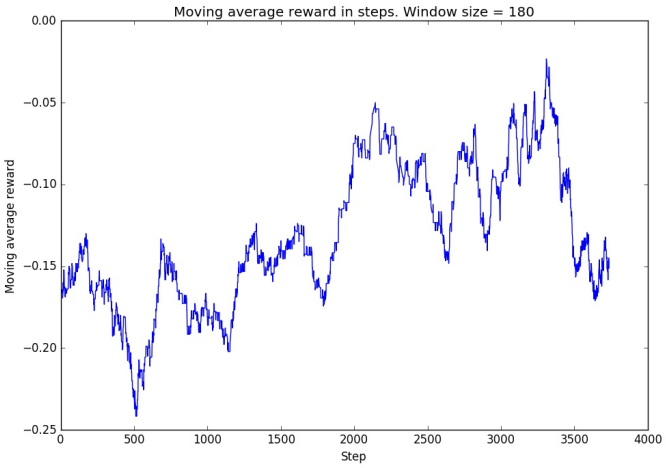


Figure 5-16: Moving average step reward for the second DRL training strategy

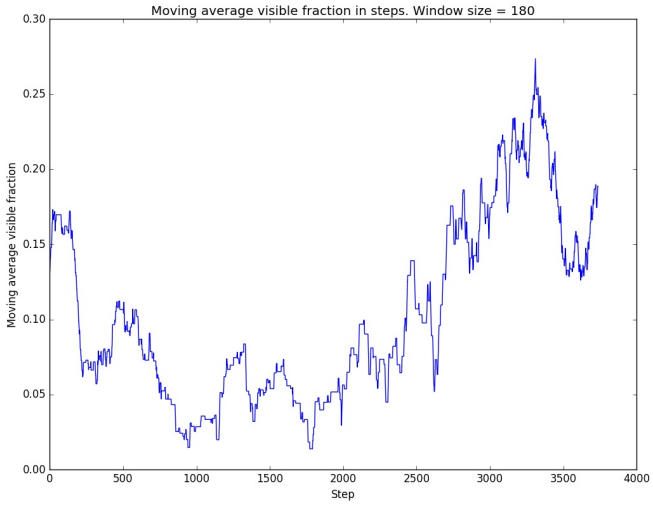


Figure 5-17: Moving average step visible tomato fraction for the second DRL training strategy

are presented in Figure 5-19. Furthermore, the maximum visible tomato fraction in each episode is demonstrated in Figure 5-20. Finally, the viewpoint of a tomato-target at the beginning and the end of the episode is illustrated in Figure 5-21. In the majority of the episodes, the robot sees the tomato-target after a small number of steps with a visible tomato fraction a little bit over the threshold. However, it fails to see the middle and top height tomatoes in Figure 5-18 at the second and third episode correspondingly, and the episodes terminate due to an out of robot's workspace pose. The reason for failing to fully see the middle height tomato is that the agent has not learned a complete policy to see this tomato.

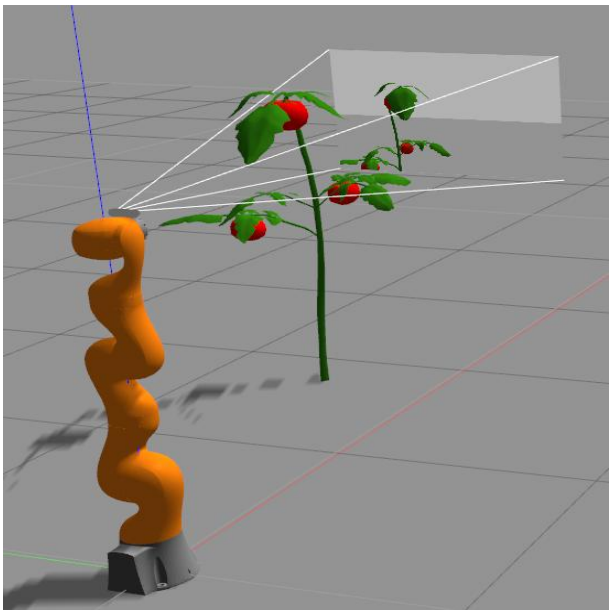


Figure 5-18: Position of the three-tomatoes plant for the training evaluation of the second DRL training strategy

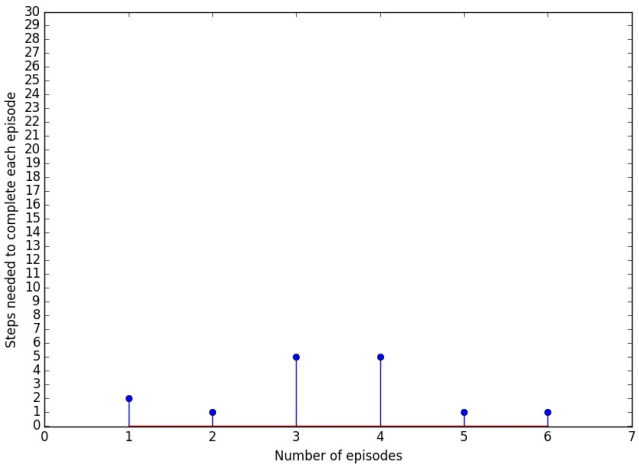


Figure 5-19: Number of steps needed for the termination of each episode in three-tomatoes plant second DRL training strategy evaluation

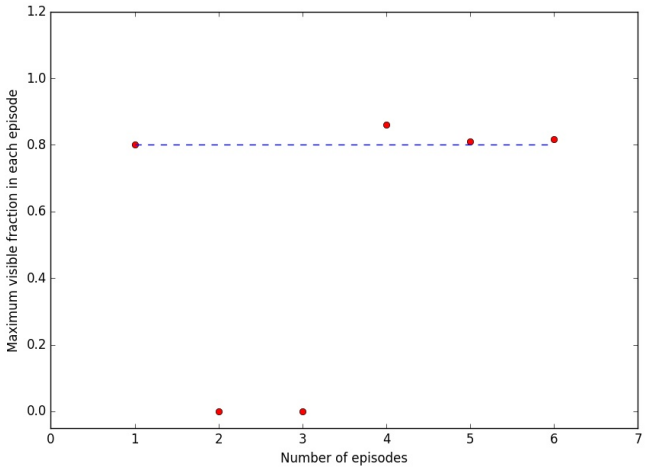
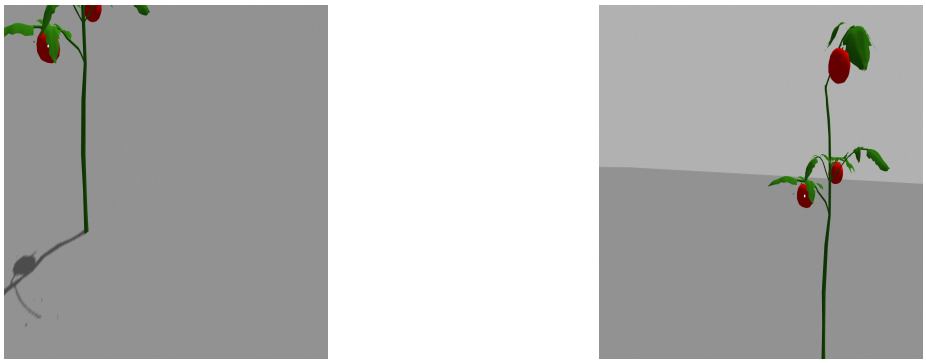


Figure 5-20: Maximum visible tomato-target fraction in each episode in three-tomatoes plant second DRL training strategy evaluation



(a) Beginning of the episode **(b)** End of the episode

Figure 5-21: Tomato-target viewpoint at the beginning and the end of an episode in three-tomatoes plant second DRL training strategy evaluation

Chapter 6

Conclusion

6-1 Discussion

This MSc thesis faces the problem of detecting fruits surrounded by occlusions. High detection accuracy is an important requirement for a successful harvesting performance. The study of the fruit detection techniques and their limitations led to the address of a fruit viewpoint optimization method. A Deep Reinforcement Learning (DRL) framework, using also state-of-the-art Convolutional Neural Network (CNN), was developed to solve the problem of fruit viewpoint optimization in autonomous tomato harvesting. Two CNN architectures, You Only Look Once (YOLO)v3 and Mask Region-Convolutional Neural Network (Mask R-CNN), were examined. A YOLOv3 was trained using real and simulated images. The achieved Average Precision (AP) was equal to 0.9477 in the validation dataset and 0.9394 in the test dataset. Also, the detection confidence score was huge and exceeded 90%, even in the presence of dense occlusions. This high performance was achieved without using a large dataset and data augmentation. Thus, the trained YOLOv3 is suitable for being used for the acquirement of densely occluded tomatoes' visual information. However, the reward formulation in DRL requires pixel information for the tomatoes.

In this way, a Mask R-CNN was trained to perform instance segmentation on the tomatoes. The used dataset was larger than the corresponding of YOLOv3, and also data augmentation was applied. Two training strategies were implemented. Both training strategies presented a smooth and decreasing training loss. However, the first one did not present a decreasing validation loss, which shows that the model overfitted. The overfitting was decreased at the second training case due to the applied changes but still, it did not have a decreasing trend. The performance of both training strategies was limited by the batch size, as the computer's GPU RAM can not handle larger than 1. The AP of the first training strategy in the test dataset was 0.673, while the second was 0.740. The tomatoes' parts were detected with a high confidence score even though they were adjacent. The second strategy managed to detect some small parts that were missing at the first strategy. Generally, the Mask R-CNN performance is quite good and can provide the majority of the required visual information. Some changes that are discussed in future work can boost the performance even higher.

The visibility of the tomato-target is exploited in the visual reward formulation in DRL. A representative metric requires information about the total number of pixels of a tomato. In particular, the fraction of the visible over the total pixels of tomato was used as an occlusion modelling metric. A simple and fast way of calculating the total number of pixels was developed. The inpainting method, in combination with some image processing, was used for this purpose.

The experiments were carried out in the Gazebo simulation environment, using a robotic manipulator and a tomato plant. A survey for a suitable camera was also performed. Deep Deterministic Policy Gradient (DDPG) was used as the DRL algorithm. The reward consisted of the visual reward, related to the occlusion modelling metric, and position penalties. Two different training strategies were applied. The first one used a more guiding reward formulation and exploration in comparison to the second one. The second strategy used infrequent positive rewards. The first training strategy achieved a sufficient training performance, and the agent was able to see the tomato-target after a few steps in each evaluation episode with a high visible fraction, except in some cases that it reached out of its workspace, so it needed more steps. Even though the second training strategy presented a decreasing loss, the moving average episode reward was unstable after reaching its peak. An early stopping would help the training performance. Moreover, the second strategy failed to see some tomatoes due to the non-optimal learned policy.

To sum up, the viewpoint optimization method performs quite well, as it can fully see the occluded tomatoes after a small number of steps in the majority of the cases. The trained CNNs are very effective and satisfy the requirements of this project. The active perception algorithm can face occlusions by plants and branches, but not by adjacent tomatoes due to the occlusion modelling method. Also, it fails at the very highly-dense plant occlusions, as probably the desired policy of the robot's end-effector movement has not been learned to face these challenging cases. Finally, the exploration way and the reward function design showed that they affect the efficiency of the learned policy.

6-2 Future Work

The developed viewpoint optimization method has a satisfactory performance even though there is space for a lot of improvements which are the following:

- The CNN can be trained to segment not only ripe tomatoes but also the surrounding branches and leaves in order to have more information about the environment. This was the initial idea, but the annotation of branches and leaves in the tomato plant required a vast of time.
- Implement a third training strategy for the Mask R-CNN to improve the two training performances. The dataset will be larger. The augmentation types and the anchors' size will be the same as the second Mask R-CNN training strategy. The number of training epochs will be 80 or larger. The number of training steps per training epoch will be 1000, while the number of the validation steps at the end of each training epoch will be 200. The non-max suppression threshold to filter Region Proposal Network (RPN) will be 0.5, and the training Region of Interest (RoI)s per image will be 300 instead of

the current 200. In this way, the RPN will generate more proposals in order to reduce the validation RPN bounding box loss. The learning rate lr for the first 10 epochs will be 0.001, for the next 30 epochs will be 0.0001, and for the rest epochs will be 0.00001. The dropout rate will be 0.5. The use of a larger batch size will increase significantly the training performance. However, this requires a more powerful GPU.

- Sometimes the same value of erosion and dilation kernels does not work for all the cases in the occlusion modelling. Thus, the adjacent tomatoes, from the beginning or due to inpainting, are not always separated. A more robust way to specify the kernel values has to be found.

- In occlusion modelling, the pixels of the deeper leaves, that are not obstacles, can be removed by the dilated contours. So, the number of the total pixels will be even more precise.

- Adjacent tomatoes can be separated with either a watershed algorithm or a second algorithm. The second algorithm, which has been validated, but is not included in the final occlusion modelling method due to speed cost, is the following: At each contour of the I_{bin} image, I find the groups of pixels which have the same identity number at the instance segmentation result. Then these groups are separated with a linear or polynomial classifier.

- DRL training can take place with more than one plant to improve the generalization of the learned policy. A diverse tomato plant in a different position must be generated in each episode.

- Include Hindsight Experience Replay (HER) in DDPG. HER improves the training performance when the rewards are sparse. In particular, the concept is to replay failed episodes, i.e. episodes which did not complete the initial goal, with an alternative goal that would be successful with the sequence of the taken failed actions. [20] showed that HER outperformed policy learning with DDPG.

- The definition of auxiliary tasks [51] can improve the performance of training with sparse rewards. These are tasks that are learned simultaneously, so some extra reward functions have to be maximized. In this way, there is a more constant reward, and the training speed is improved. An example of an auxiliary task would be the computation of the distance between the end-effector and the tomato-target. This specific auxiliary task could also affect the learned policy to approach more frequently the tomato-target while searching for the optimal viewpoint.

- The difficulty of designing the reward function can be faced by learning the reward from human demonstrations using imitation learning.

- The dense occlusion of the tomato-target can make the observation related to the Cartesian coordinates of the tomato-target partially observable. This can affect the returned reward. The instance segmentation of leaves and the integration of information from previous states to the current state can solve the partial observability problem.

- The training performance can be compared to that of other DRL algorithms such as Synchronous Advantage Actor-Critic (A2C), Asynchronous Advantage Actor-Critic

(A3C), Trust Region Policy Optimization (TRPO), and Distributed Distributioanl Deterministic Policy Gradient (D4PG) [50]. Moreover, the learned policy can be compared to a visual servoing or another classical robotics vision-based method. This comparison will show if the complexity of the DRL is worthy to implement in the viewpoint optimization problem.

- Evaluate the trained algorithm on a real robot.

Bibliography

[1] S. G. Vougioukas, "Agricultural robotics," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, no. 1, pp. 365–392, 2019.

[2] E. Van Henten, B. Van Tuijl, J. Hemming, J. Kornet, J. Bontsema, and E. Van Os, "Field test of an autonomous cucumber picking robot," *Biosystems Engineering*, vol. 86, no. 3, pp. 305 – 313, 2003.

[3] D. S. Klaoudatos., V. C. Moulianitis., and N. A. Aspragathos., "Development of an experimental strawberry harvesting robotic system," in *Proceedings of the 16th International Conference on Informatics in Control, Automation and Robotics - Volume 2: ICINCO,*, pp. 437–445, INSTICC, SciTePress, 2019.

[4] E. Van Henten, B. Van Tuijl, G.-J. Hoogakker, M. Van Der Weerd, J. Hemming, J. Kornet, and J. Bontsema, "An autonomous robot for de-leafing cucumber plants grown in a high-wire cultivation system," *Biosystems Engineering*, vol. 94, no. 3, pp. 317 – 323, 2006.

[5] D. Klaoudatos and R. Babuska, "Path planning and control of a robotic manipulator for agricultural applications," 2020.

[6] S. Giancola, M. Valenti, and R. Sala, *A Survey on 3D Cameras: Metrological Comparison of Time-of-Flight, Structured-Light and Active Stereoscopy Technologies*. SpringerBriefs in Computer Science, Springer International Publishing, 2018.

[7] A. Vit and G. Shani, "Comparing rgb-d sensors for close range outdoor agricultural phenotyping," *Sensors*, vol. 18, p. 4413, 12 2018.

[8] T.-T. Nguyen, J. Keresztes, K. Vandevoorde, E. Kayacan, J. De Baerdemaeker, and W. Saeys, "Apple detection algorithm for robotic harvesting using a rgb-d camera," 07 2014.

[9] M. Hannan, T. Burks, and D. Bulanon, "A machine vision algorithm combining adaptive segmentation and shape analysis for orange fruit detection," *Agricultural Engineering International : The CIGR e-journal*, vol. 0, 05 2010.

[10] C. McCool, I. Sa, F. Dayoub, C. Lehnert, T. Perez, and B. Upcroft, "Visual detection of occluded crop: For automated harvesting," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2506–2512, 2016.

[11] 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*, vol. 163, p. 104846, 2019.

[12] G. Liu, J. Nouaze, P. Touko, and J. Kim, "Yolo-tomato: A robust algorithm for tomato detection based on yolov3," *Sensors*, vol. 20, 04 2020.

[13] Y. Tian, G. Yang, Z. Wang, H. Wang, E. Li, and Z. Liang, "Apple detection during different growth stages in orchards using the improved yolo-v3 model," *Computers and Electronics in Agriculture*, vol. 157, pp. 417 – 426, 2019.

[14] P. Corke, *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*. Springer Publishing Company, Incorporated, 1st ed., 2013.

[15] S. Mehta and T. Burks, "Vision-based control of robotic manipulator for citrus harvesting," *Computers and Electronics in Agriculture*, vol. 102, pp. 146 – 158, 2014.

[16] Q. Bateux, É. Marchand, J. Leitner, and F. Chaumette, "Visual servoing from deep neural networks," *CoRR*, vol. abs/1705.08940, 2017.

[17] H. Cuevas-Velasquez, N. Li, R. Tylecek, M. Saval-Calvo, and R. B. Fisher, "Hybrid multi-camera visual servoing to moving target," *CoRR*, vol. abs/1803.02285, 2018.

[18] C. F. Lehnert, D. Tsai, A. P. Eriksson, and C. McCool, "3d move to see: Multi-perspective visual servoing for improving object views with semantic segmentation," *CoRR*, vol. abs/1809.07896, 2018.

[19] P. Zapotezny-Anderson and C. Lehnert, "Towards active robotic vision in agriculture: A deep learning approach to visual servoing in occluded and unstructured protected cropping environments," *IFAC-PapersOnLine*, vol. 52, pp. 120–125, 01 2019.

[20] R. Cheng, A. Agarwal, and K. Fragkiadaki, "Reinforcement learning of active vision formanipulating objects under occlusions," *CoRR*, vol. abs/1811.08067, 2018.

[21] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2016.

[22] J. Sather, "Viewpoint optimization for autonomous strawberry harvesting with deep reinforcement learning," *CoRR*, vol. abs/1903.02074, 2019.

[23] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015.

[24] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016.

[25] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018.

[26] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015.

[27] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," *CoRR*, vol. abs/1612.03144, 2016.

[28] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017.

[29] K. G. Vamvoudakis and N.-M. T. Kokolakis, "Synchronous reinforcement learning-based control for cognitive autonomy," *Foundations and Trends® in Systems and Control*, vol. 8, no. 1–2, pp. 1–175, 2020.

[30] J. Kober, J. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, pp. 1238–1274, 09 2013.

[31] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *CoRR*, vol. abs/1811.12560, 2018.

[32] C. J. C. H. Watkins and P. Dayan, "Q-learning," in *Machine Learning*, pp. 279–292, 1992.

[33] G. J. Gordon, "Stable fitted reinforcement learning," in *Proceedings of the 8th International Conference on Neural Information Processing Systems*, NIPS'95, (Cambridge, MA, USA), p. 1052–1058, MIT Press, 1995.

[34] M. Riedmiller, "Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method," in *Proceedings of the 16th European Conference on Machine Learning*, ECML'05, (Berlin, Heidelberg), p. 317–328, Springer-Verlag, 2005.

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–33, 02 2015.

[36] Z. Wang, N. de Freitas, and M. Lanctot, "Dueling network architectures for deep reinforcement learning," *CoRR*, vol. abs/1511.06581, 2015.

[37] V. Konda and J. N. Tsitsiklis, *Actor-Critic Algorithms*. PhD thesis, USA, 2002. AAI0804543.

[38] R. Munos, T. Stepleton, A. Harutyunyan, and M. G. Bellemare, "Safe and efficient off-policy reinforcement learning," *CoRR*, vol. abs/1606.02647, 2016.

[39] C. Hennersperger, B. Fuerst, S. Virga, O. Zettinig, B. Frisch, T. Neff, and N. Navab, "Towards mri-based autonomous robotic us acquisitions: a first feasibility study," *IEEE transactions on medical imaging*, vol. 36, no. 2, pp. 538–548, 2017.

[40] A. Dutta and A. Zisserman, "The VIA annotation software for images, audio and video," in *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, (New York, NY, USA), ACM, 2019.

[41] A. B. Jung, K. Wada, J. Crall, S. Tanaka, J. Graving, C. Reinders, S. Yadav, J. Banerjee, G. Vecsei, A. Kraft, Z. Rui, J. Borovec, C. Vallentin, S. Zhydenko, K. Pfeiffer, B. Cook, I. Fernández, F.-M. De Rainville, C.-H. Weng, A. Ayala-Acevedo, R. Meudec, M. Laporte, *et al.*, "imgaug." https://github.com/aleju/imgaug, 2020. Online; accessed 01-Feb-2020.

[42] W. Abdulla, "Mask r-cnn for object detection and instance segmentation on keras and tensorflow." https://github.com/matterport/Mask_RCNN, 2017.

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[44] A. Telea, "An image inpainting technique based on the fast marching method," *Journal of Graphics Tools*, vol. 9, 01 2004.

[45] A. Fitzgibbon, M. Pilu, and R. Fisher, "Direct least-squares fitting of ellipses," vol. 21, pp. 253 – 257 vol.1, 09 1996.

[46] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015.

[47] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 249–256, 01 2010.

[48] P. Beeson and B. Ames, "Trac-ik: An open-source library for improved solving of generic inverse kinematics," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pp. 928–935, 2015.

[49] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *CoRR*, vol. abs/1802.09477, 2018.

[50] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. TB, A. Muldal, N. Heess, and T. P. Lillicrap, "Distributed distributional deterministic policy gradients," *CoRR*, vol. abs/1804.08617, 2018.

[51] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *CoRR*, vol. abs/1611.05397, 2016.

Glossary

List of Acronyms

3DMTS	3D Move to See
A2C	Synchronous Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
ANN	Artificial Neural Network
AP	Average Precision
CGN	Centre for Genetic Resources, the Netherlands
CNN	Convolutional Neural Network
COCO	Common Objects in Context
D4PG	Distributed Distributioanl Deterministic Policy Gradient
DoF	Degrees of Freedom
DDPG	Deep Deterministic Policy Gradient
DDQN	Double Deep Q Network
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
DRN	Deep Residual Network
FC	Fully Connected
FCN	Fully Convolutional Network
FP	False Positive
FPN	Feature Pyramid Network

FN False Negative

GPU Graphics Processing Unit

HER Hindsight Experience Replay

IoU Intersection over Union

KDL Kinematics and Dynamics Library

LSTM Long Short Term Memory

Mask R-CNN Mask Region-Convolutional Neural Network

RAM Random Access Memory

R-CNN Region Based Convolutional Neural Network

RL Reinforcement Learning

RoI Region of Interest

ROS Robot Operating System

RPN Region Proposal Network

RNN Recurrent Neural Networks

ToF Time of Flight

TP True Positive

TRPO Trust Region Policy Optimization

URDF Unified Robot Description Format

VOC Visual Object Classes

YOLO You Only Look Once