Interdisciplinary Project

Visual perception for autonomous harvesting of root vegetables growing in aeroponics

Niederhauser Thibault
Professor: Calinon Sylvain

École Polytechnique Fédérale de Lausanne
July 2020

Contents

1 Introduction **1**

2 Problem Description **2**
2.1 Context . . . 2
2.2 The Root System . . . 2
2.3 Goal of the Project . . . 2

3 Methods **5**
3.1 Hardware Solution . . . 5
3.2 Algorithm Architecture . . . 5
3.3 Object Detection . . . 7
3.3.1 Classical Methods . . . 7
3.3.2 Deep Learning Based Methods . . . 8
3.3.2.1 Region Proposal-Based Detection . . . 8
3.3.2.2 Single Shot Detection . . . 9
3.3.2.3 Feature Extraction . . . 10
3.3.2.4 Training . . . 11
3.4 Selection and Tracking . . . 11
3.5 Pose estimation . . . 13
3.5.1 3D-Localization . . . 13
3.5.2 Orientation . . . 13
3.5.2.1 Image Segmentation . . . 14
3.5.2.2 Orientation estimation . . . 15
3.6 Integration in global project . . . 15

4 Results **16**
4.1 Object Detection . . . 16
4.2 Pose Computation . . . 16
4.2.1 3D localization . . . 16
4.2.2 Orientation . . . 17

5 Discussion **19**
5.1 Selected solution 19
5.1.1 Object Detection 19
5.1.2 Pose computation 20
5.2 Improvement and further work possibilities 21

6 Conclusion **22**

A Intel RealSense D415 datasheet **27**

B Precision-Recall Curves **28**

C 3D-Localization Data **30**

D Object detection with different models **31**

1. Introduction

Along with the development of space research and technology, the idea of sending humans on foreign planets or satellites, such as mars or the moon, for long periods of time or even permanently is getting more on more concrete. One of the main concern about supporting human life away from Earth for extensive time-spans is the need to provide reliable and durable food supplies. The GrowBotHub project addresses this challenge by proposing fully autonomous vegetable growing and harvesting systems.
In order to cover a high percentage of human's daily energetic needs, a self-sufficient setup to grow potatoes was designed, as those root vegetables have a high calorie intake. The potatoes are grown in aeroponics to limit water needs and are autonomously harvested using a robotic arm. This is where cameras, machine vision and perception come into play. Indeed, automated vegetables picking requires some information about the vegetables' ripeness and positions. To tackle this problem, this project proposes a perception algorithm for potato detection and pose estimation involving RGB-D sensing, deep-learning algorithms and image processing techniques. Several state-of-the-art methods are explored, tested and compared in order to find the solution that best suits the specific task of harvesting aeroponics growing potatoes.
Even though, the development of this project is carried out in the scope of space technologies, robotic autonomous harvesting has a much wider field of application, as it could remove the need of harvesting human workforce on Earth. In regard to this, machine vision research has already shown satisfying results for perception of several sorts of vegetables and fruits [14], [15], [16]. In this sense, this project aims at developing a modular solution and proposes different approaches for the each process of the algorithm, so that the solution can be adapted for perception of other vegetables or fruits in other environments.

2. Problem Description

2.1 Context

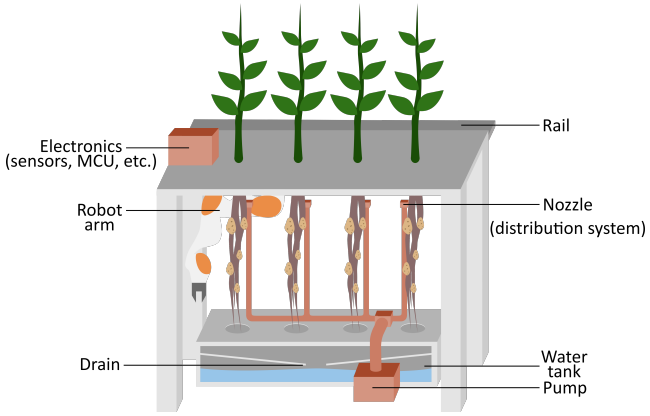
This semester project was conducted in the scope of GrowBotHub, an EPFL space-related interdisciplinary project that aims at developing autonomous and sustainable food supply for extreme environments such as the moon. This challenge is tackled by designing self-sufficient growing and harvesting systems, in which vegetables are grown in aeroponics and automatically harvested using cameras and a robotic arm. The GrowBotHub project is exploring different solutions for growing vegetables and designed different systems. This semester project was developed for the so-called Root-System, which aims at growing and harvesting root vegetables.
The GrowBotHub project takes part in IGLUNA 2020. IGLUNA is an international project coordinated by the Swiss Space enter, which aims at promoting space technologies development in universities and fosters international collaboration. The purpose of IGLUNA 2020 is to develop technologies to support human life in extreme habitats such as the moon or a foreign planet.

2.2 The Root System

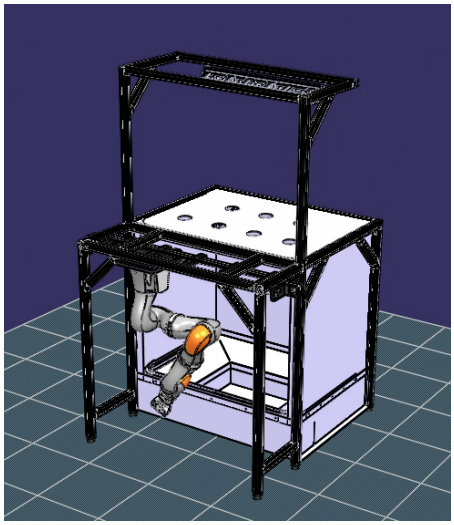
The Root System is designed to autonomously grow and harvest root vegetables. So far the focus has been specifically set on growing potatoes, since they have a huge calories intake compared to other root vegetables. The potatoes are grown using aeroponics: a mix of water and nutrients are pumped into nozzles that directly spray the liquid on the growing roots. Fig. 2.2 shows an example of potatoes growing in aeroponics: the roots are hanging directly in the air, there is no soil or mud. This way, growing potatoes can easily be accessed and harvested by a robotic arm.
The mechanical structure of the Root System consist of a metallic scaffold, on which a KUKA robotic arm (KUKA lbr iiwa 7 R800) is fixed. The potatoes are seized using a two finger gripper (Schunk EGH gripper), which is attached at the tip of the robotic arm. The arm is oriented downwards and can move horizontally on a rail. The potatoes' leaves grow above the mechanical structure under artificial light, whereas the roots hang in the lower part of the structure. The robotic arm is located at the same level as the roots and can therefore be used to harvest the growing potatoes (see Fig. 2.1). The rail and the kinematic redundancy of the KUKA robotic arm confer enough mobility to harvest any growing potato.

2.3 Goal of the Project

The aim of this project is to develop a solution for perception of the growing potatoes in the Root System. In order to achieve automated harvesting, the system needs to be able to detect and gather information about the growing potatoes. The required information to allow the robotic arm to achieve successful harvesting are potatoes' 3D coordinates, ripeness, size, and orientation.



(a) The Root System's units



(b) The Root System's mechanical structure

Figure 2.1: The Root System

Figure 2.2: Potatoes growing in aeroponics

(a)

(b)

(c)

Figure 2.3: Adaptation of the Root System

Due to the particular situation caused by Covid-19, access to the Root System's mechanical structure and aeroponics growing of potatoes were impossible. Nevertheless, the robotic arm and gripper could be taken out of campus and used by the robotic team. In order to work on the project, an adaptation of the Root System was built and the aeroponics growing potatoes were simulated by hanging potatoes on strings. This setup allowed to develop a solution and test it in conditions close to the targeted ones; however some minor tuning and adaption might be required when translating the developed solution to the real Root System.

3. Methods

3.1 Hardware Solution

A RGB-D camera (Intel RealSense Depth Camera D415, see Appendix A) is used to detect and locate the potatoes to be harvested. The camera is fixed on the SCHUNK gripper with a 3D-printed mount (see Table 3.1, this way the camera moves with the robotic arm. The Intel RealSense D415 is compact enough so that fixing it on the arm does not cause any problem.
Intel RealSense Depth Camera D415 camera has two channels:

- A RGB channel: this channel is used to detect potatoes and find their orientations.
- A depth channel: this channel is used to compute the potatoes' 3D-coordinates and sizes. Active IR stereo technology is used to gather the depth information. The camera has a high depth per degree accuracy, which allows to locate the potatoes with great reliability. The depth per degree accuracy comes to the cost of a tighter field of view. Since the camera is mobile (mounted on the robotic arm), the field of view is still more than satisfying for the task.
  The camera has a specified minimal depth distance of 16 cm. In practice, the depth data is not exploitable for objects nearer than around 20-25 cm. Therefore the potatoes' coordinates and sizes must be computed from a further distance than 25 cm and the control of the arm needs to be adapted in this sense.

3.2 Algorithm Architecture

The aim of the perception algorithm is to find ripe potatoes and to compute relevant information about their pose in space. This information is then communicated to the robotic arm's controller for harvesting. The

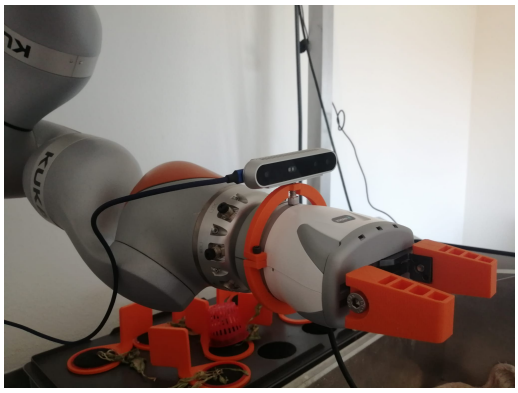


Figure 3.1: The RealSense camera mounted on the gripper

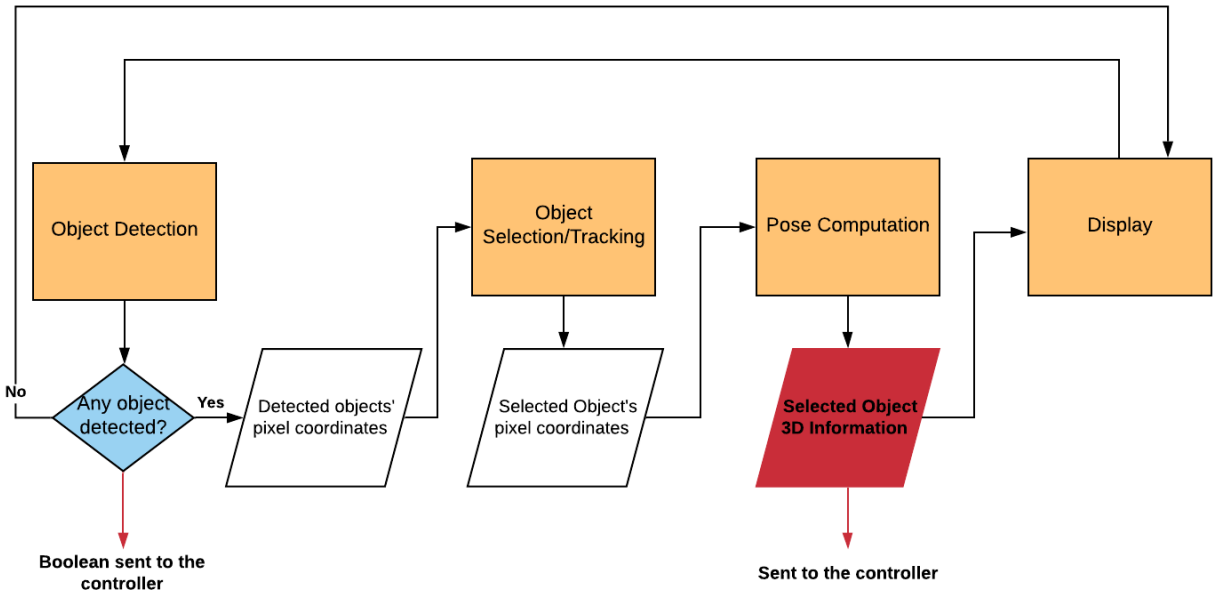


Figure 3.2: Algorithm Architecture

algorithm architecture is divided in four main blocks:.

- **Object-Detection:** The first task of the algorithm is to use the camera's information to identify growing potatoes. This is a well-known computer vision problem designated as *object-detection*. Many different approaches allow to process images to find the pixels' location of specific objects (see section 3.3).
- **Object Selection and Tracking**: Since the potatoes are picked one by one by the robotic arm, it is required to select only one potato to be harvested. Therefore, the Object Selection and Tracking algorithm chooses one potato as target (selection) and ensures that the same potato is selected at every camera frame (tracking). In addition, this module also makes sure that the selected potato is ready to be harvested by evaluating its ripeness. Some potatoes that were identified by the object-detector might not be ripe yet, therefore potatoes smaller than a fixed size threshold are discarded.
- **Pose Computation**: Once a potato is selected for harvesting, its relevant 3D information are computed, namely its orientation, 3D-coordinates and size.
- **Display** (see 3.3: The camera image flow and the processed information (detected objects, pose, sizes, etc.) are displayed for comprehension and debugging purposes (see Figure 3.3).

The code is developed in Python and can be found at the following link: The project's code can be found at the following link: `https://github.com/ThibaultNiederhauser/potato_harvesting_vision`

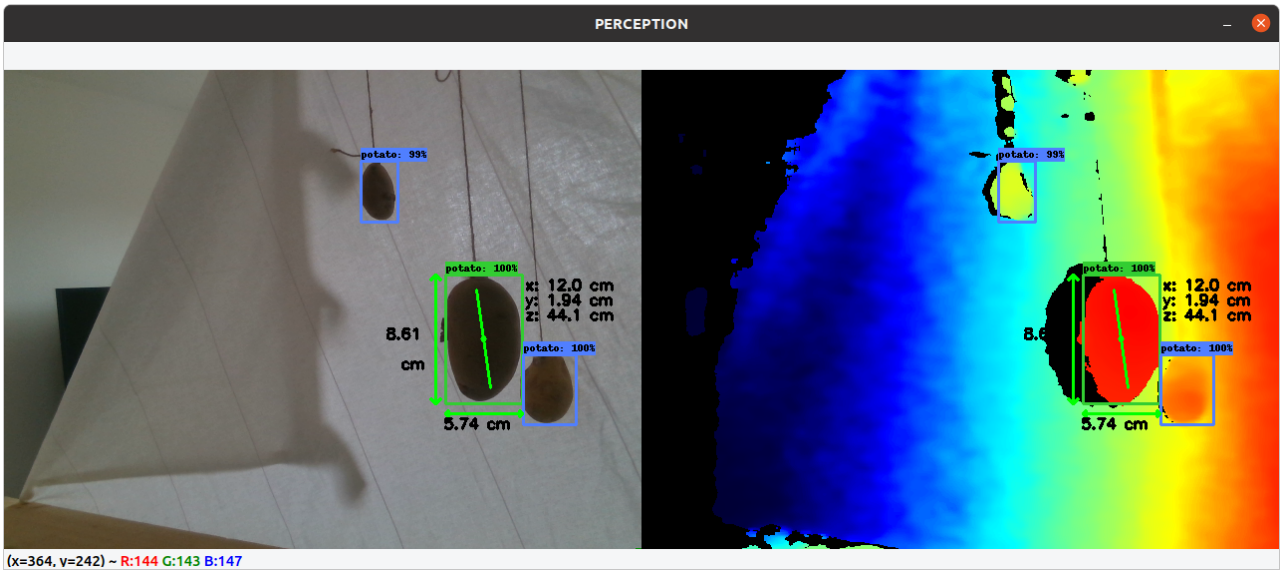


Figure 3.3: Display output of the perception algorithm: here three potatoes are detected. Then the nearest one is selected (in green) and its 3D position, orientation and size are computed.

3.3 Object Detection

Given an image or video stream and a set of objects, the task of object detection is to identify which of those objects might be present in the image and to compute information about their positions. Commonly, object detection models output rectangle bounding boxes around the detected objects.

3.3.1 Classical Methods

In this context, the term *classical methods* is to be understood as image processing and machine learning methods that do not involve deep learning. Such methods include among others image filtering, pattern recognition, edge detection, clustering machine learning algorithms such as K-Means, DBSCAN, Gaussian Mixture Models, etc. The design of the algorithm and the choice of the exploited visual features highly depends on the objects to be detected, the environment and the available information (RGB data, depth data, laser scan, etc.).
Classical methods have been widely used for crop's detection or inspection. Among others *Blasco et al.* [1] achieved fruits' peduncles detection using a Bayesian discriminant on RGB data, *Yamamoto et al* showed that a decision-tree-based model was able to identify tomatoes [5]. Grapes berries could be detected using radial symmetry transforms [6] or through segmentation, contour detection and curvature analysis [2]. Furthermore, algorithms based on conditional random fields (CRF) on RGB or texture data were able to classify the structure of grapevines into branches, leaves and fruits [3] or to detect sweet pepper crops [4].
Although classical image processing and machine learning methods have shown some success in crop's detection, they have been overtaken by the recent development of deep learning-based methods. One reason for this is that classical algorithms are usually task-specific and do not translate easily from one situation to another. As a matter of fact, clustering on RGB data might be able to detect red sweet pepper among leaves and branches, but probably would not be able to discriminate between potatoes and roots: new features need to be extracted for new situations. Furthermore, classical methods tend to have a longer computational time and lower accuracy and precision performances than deep learning-based methods. Therefore, DL-based methods

are nowadays commonly accepted as the state-of-the-art approach in matter of object-detection. Consequently, this project does not explore classical methods, but focuses on DL-based approaches.

3.3.2 Deep Learning Based Methods

Deep learning-based methods perform object detection using deep convolutional neural networks. Contrarily to classical machine learning techniques, the relevant features are learned by the network and do not need to be specified and extracted separately. Such methods have become the state-of-the-art approach for object detection.

The operation pipeline of object detection models can be divided into three main steps [7]:

- **1. Features extraction:** Image's pixels are fed into layers that extract relevant visual information for the task.
- **2. Region selection:** Different region of interests (or bounding boxes) that are likely to contain objects are selected on the input image. Since there might be several selected bounding boxes for the same object, a further processing step, such as non-maximum suppression, is sometimes performed on regions of interest with consequent overlap, in order to select the best one.
  Note that, depending on the network architecture, region selection is sometimes performed before features extraction, on raw image pixels.
- **3. Classification**: The selected regions of interest are fed into classification layers that determine to which category the objects belong. The output function of the classification sub-network is usually a single or multi-class softmax, so that the objects are labeled with a class and a confidence.

There exist different network architectures that perform those tasks in different ways. Based on how region selection is performed, object detection models can be divided into two main categories: region proposal-based frameworks and single-shot-detectors.

3.3.2.1 Region Proposal-Based Detection

Region proposal-based frameworks compute the regions of interest based on the image's information. Some architectures such as R-CNN (Regions Convolutional Neural Networks) and Fast-RCNN use selective search [8] to find the regions of interest. However, this approach is computationally heavy and time demanding, therefore, in this project, **Faster-RCNN**s [13] are considered. In such models, the regions of interest are computed by a fully connected convolutional network, called Region-Proposal-Network (RPN) (see Fig. 3.4). Previous works by *Ma et al* [14] and *Sa et al* [15] achieved successful fruit detection using Faster-RCNNs.

Another architecture implemented in this project is the Region-based Fully Convolutional Network (**R-FCN**) [19]. This model is a modification of the Faster-RCNN architecture that aims at accelerating the network. In Faster-RCNN, the RoI Pooling sub-network is inserted before the convolutional classification network (see Fig. 3.4), hence the classification layers perform unshared per-RoI computations, which slows down the process. The basic idea of R-FCN is to move all the fully connected convolutional layers before RoI pooling and to use them to generate a position sensitive score map (see Fig. 3.5): the image space is discretized in many small regions of the same size. For all $C + 1$ classes (all object classes plus the background class), each region is labeled with a score representing how likely it is that the region contains a part of the object. In parallel, the RoIs are computed using a RPN and RoI pooling is performed. After pooling, the score map is used to perform

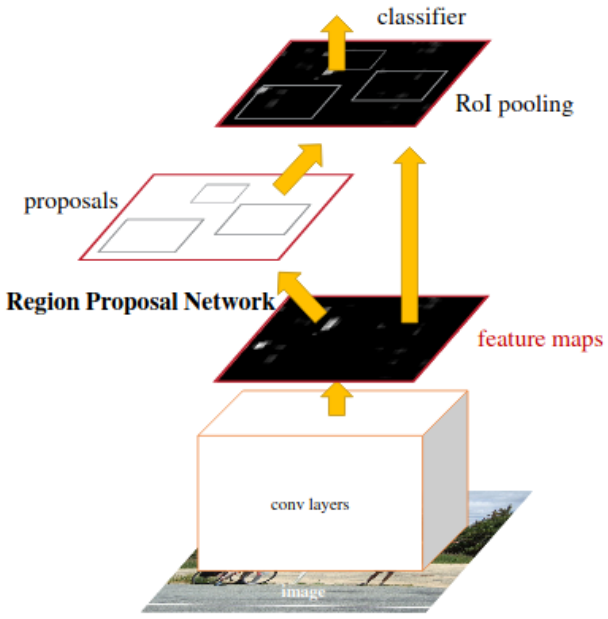


Figure 3.4: Faster-RCNN architecture: features are extracted from the image to construct feature maps. Then the feature map is fed into the RPN, which outputs the regions of interest (or region proposals). Subsequently, regions of interest pooling (RoI pooling) is applied on the proposals and feature maps; this allows all the image proposals to have the correct input size for the classification network. Eventually, the pooled information is fed into the classification network.

average voting on each RoI; this outputs a $C + 1$ dimensional vector which is eventually used to compute the softmax response across classes. Since there are no convolutional layers following RoI computing and pooling, the per-RoI computational cost is low (the only processes that are present after RoI pooling are average voting and softmax computation, which have a low computational cost). Thereby R-FCN are meant to be faster than Faster-RCNN, while maintaining similar accuracy performances.

3.3.2.2 Single Shot Detection

Contrarily to region proposal-based methods, single shot detectors do not compute the regions of interest based on the image features, but rely on a set of predetermined regions. To select the regions, a grid of default anchors is applied on the input image. At each anchor location, a list of default boxes of different sizes and shapes is set as the regions of interest. For each box, a prediction on whether the box contains an object or not is computed and additional tuning of the boxes cares for tighter fitting of the objects. Eventually non-maximum suppression is applied on boxes with significant overlap [7] . (see Fig. 3.6)
Whereas region proposal-based methods need to process the image's features map twice (once to compute the region proposals and once for classification), single shot detectors only require one processing step. Therefore, such methods are faster and can be used to perform real-time object-detection.
Two state-of-the-art architectures exist in the single shot detectors category: You Only Look Once (YOLO) [11], [10] and Single Shot Multibox Detector (SSD) [9]. Both models have similar speed performance and allow real-time object detection. In the scope of this project, SSD models are implemented and tested, since they show slightly better accuracy performances on COCO and PASCAL VOC data sets [7]. Note that there exist two types of SSDs, namely SSD300 and SSD512, in which the only difference is the size of the input (300x300

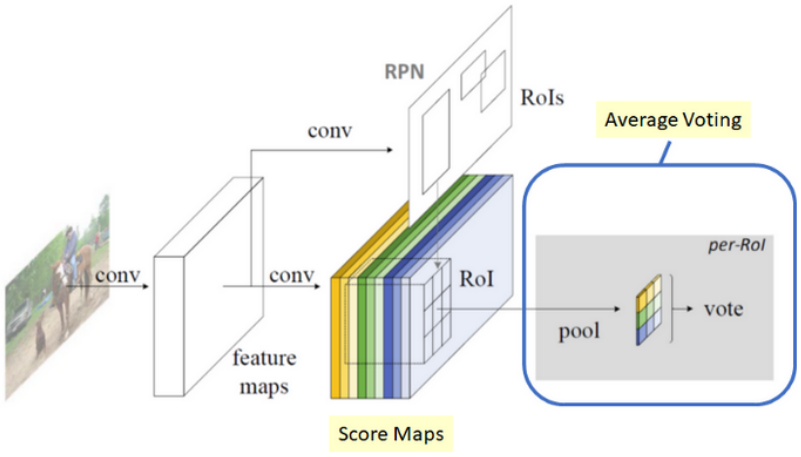


Figure 3.5: R-FCN architecture: a score map and RoIs are computed by two different sets of convolutional layers. The RoIs are then applied on the score map and average voting is performed. The per-RoI computational cost is low.

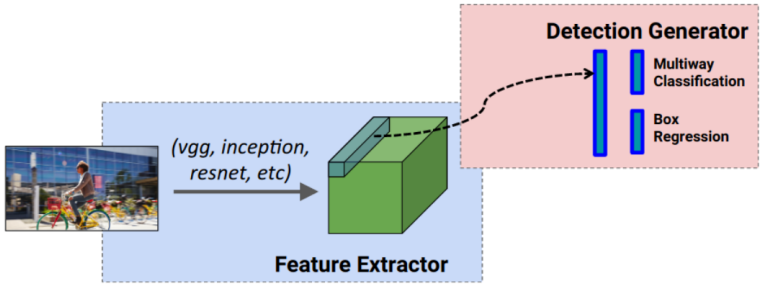


Figure 3.6: SSD architecture [23]

versus 512x512). Smaller input is meant to increase the speed performance of the network, but lowers the detection's accuracy.
In past research, SSDs have been successfully used for strawberries [17] and apples detection [16].

3.3.2.3 Feature Extraction

All deep learning-based object detection models make use of convolutional layers to extract relevant features from the image's pixels and build feature maps. The choice of the features extraction's network has an impact on the object detector's performances, therefor different architectures are implemented and tested in this project. The focus is set on the three following state-of-the-art feature extractors:

- **ResNet101** [22]: deep residual networks (short ResNet) focuses on accuracy performances.
- **Inception v2** [21]: this model aims at reducing the computational cost.
- **Mobilenet v2** [20]: Mobilenets are efficient models designed for embedded and mobile applications.

Table 3.1: Principal Training Parameters

Model type	Batch Size	Optimizer	Learning Rate
F-RCN	1	Momentum	0.0003
Faster-RCNN	1	Momentum	0.0002
SSD	24	RMSProp	0.004

3.3.2.4 Training

The first step to train object detection neural networks is to build a training data set. To do so 300 RGB photos of (simulated) aeroponics growing potatoes are stored. In order to obtain the best possible performances, the images need to be taken in conditions similar to the working conditions. Therefore, the photos are directly taken with the RealSense camera from positions where the camera could potentially be during harvesting.
Subsequently, the data-set is labeled manually: bounding boxes are drawn around all potatoes present in each image. There exist several software that allow to perform such labeling, in this case *Labelbox* was used. Then the labeled data-set is exported to the *RoboFlow* platform, which allows to generate XML files containing the labeling information (those files are required for training). Eventually, 10% of the data-set is kept aside for testing purposes; this leads to a training set of 270 images and a test set of 30 images.
In order to improve the object detectors' performances and robustness to environment variations, a data augmentation pre-processing step is performed on the training data-set before training. The data augmentation includes horizontal flips and random image cropping to improve the detection of potatoes of varying sizes and orientations, image resizing to speed-up training, as well as brightness modifications to increase robustness to changing luminosity conditions. Indeed, since potatoes root are actually meant to grow in the dark or under low luminosity, the light conditions of the adapted root system might differ from the ones of the real system; the last data augmentation step aims at maintaining the object detection performance when moving to the real Root System.
Rather than training the networks from scratch, fine-tuning of pre-trained-models on the COCO data-set is performed. This allows faster training and still shows satisfying performances. The pre-trained models are taken from the Tensorflow object detection model zoo[1]. Practically, training is performed using python's Tensorflow API. The neural networks are fine-tuned using Tensorflow's *train.py* function (which relies on back-propagation) and the 270 training images as input. The training parameters are chosen to be the same as the ones used for the training of the pre-trained models on the COCO dataset (see Table 3.1) and are retrieved from configuration files available on Tensorflow's platform[2]. Note that to improve computational time, the neural network training was performed using *Google Colab*'s GPUs.

3.4 Selection and Tracking

As soon as one or more ripe potatoes are detected, the perception algorithm needs to return information about one targeted potato, so that the robotic arm can harvest it. For each frame captured by the camera, two scenarios are possible. The first scenario occurs when no ripe potato was detected in the previous frame. In this case, a **selection** phase is performed: the nearest ripe potato is selected as the one to be harvested. Note that the ripeness evaluation is made based on the potatoes' sizes , therefore the dimensions are already calculated at this point even though it is part of the pose estimation algorithm (see section 3.5.1). Selecting a potato before applying the complete pose computation algorithm avoids the need to calculate the pose of all detected potatoes and improves the computational cost.

[1]https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md
[2]https://github.com/tensorflow/models/tree/master/research/object_detection/samples/configs

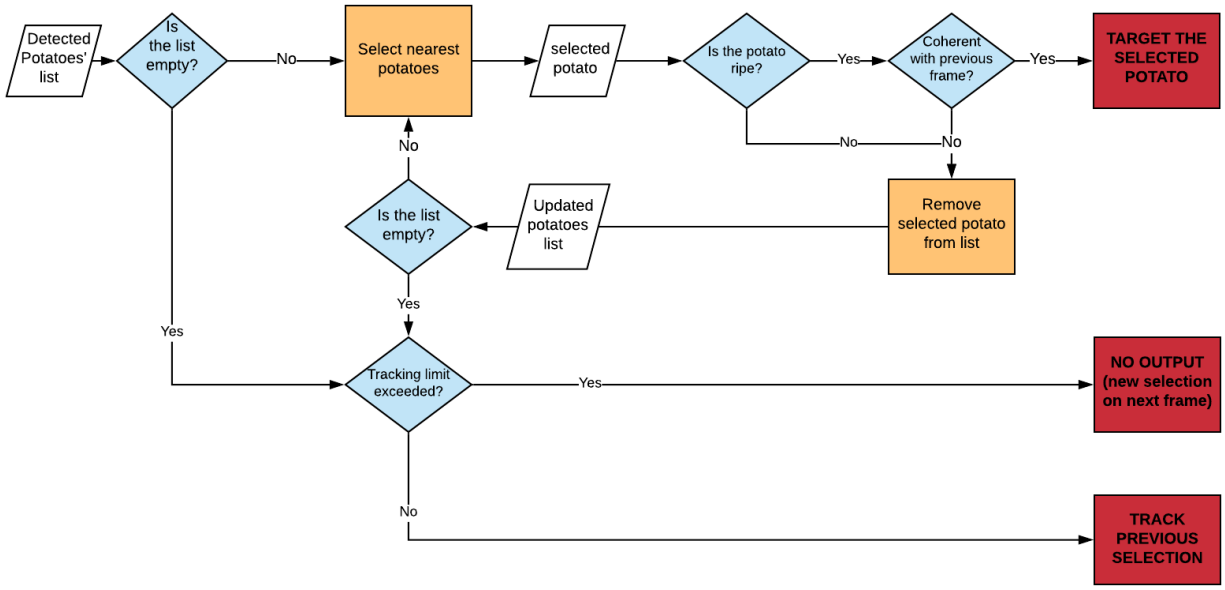
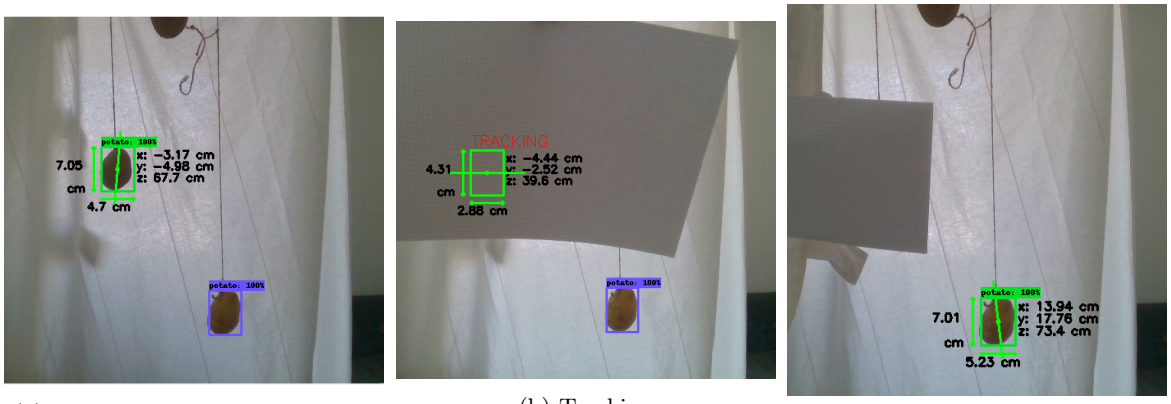


Figure 3.7: Structure of the tracking algorithm

In the second scenario, a potato was already targeted in the previous frame. In this situation, the algorithm needs to return information about that same targeted potato (**tracking**). Here, simply selecting the nearest potato as for the selection phase is not sufficient; indeed the object detector is not perfect and might fail to detect the targeted potato on one frame, and another potato would be selected instead. Furthermore, the control strategy of the robotic arm might lead the camera to get nearer to another non-targeted potato during motion, which would again result in a change of target. Such occurrences might stuck the controller into oscillations between two or more potatoes and need to be avoided.
To achieve successful tracking of a targeted potato, the algorithm starts by selecting the nearest potato to the camera. It then checks if this selection is coherent with the previous frame's selection; this is done by looking if the center of the currently selected bounding box is near enough the center of the previous frame's bounding box. A proximity threshold that defines if two selections are coherent (or *near enough*) is declared as an algorithm parameter. If the current frame selection is coherent with the previous one, it is stored as the targeted potato; if not, then the next nearest ripe potato is selected and the proximity check is performed again. This loop is executed until a selection coherent with the previous frame is found. If all detected objects are investigated and none of them matches the previous target, it means that, for some reasons, the object detector failed to identify the targeted potato on the current frame. In this case a tracking algorithm based of the previously selected bounding box is run to estimate the position of the targeted root vegetable (practically, the tracker is implemented using the python library *dlib*).
Finally, the algorithm keeps track of the number of consecutive frames on which tracking of a non-detected potato is performed. If a certain number of consecutive frames is reached (defined as a parameter), it probably means that the target potato is lost (this could happen for example because the initial selection was a false positive or because the targeted potato got occluded by too many roots). In this case, the targeted bounding box is discarded, and the algorithm returns the selection phase on the next frame.



(a) Selection of the closest potato

(b) Tracking

(c) Re-selection of the closest potato

Figure 3.8: Example of the selection/tracking algorithm: the closest potato is selected (a). If the potato is not detected on a frame, tracking is performed to estimate its position (b). After a certain number of consecutive tracking frames, the potato is considered as lost and selection of the closest potato is performed again (c).

3.5 Pose estimation

3.5.1 3D-Localization

The real world 3D-coordinates of a pixel captured by the camera can be computed using the distance to the pixel and the intrinsic and extrinsic parameters of the camera (focal length, lens distortion, etc.). Practically, this computation is made by looking at the central pixel of the potatoes bounding boxes and by using the built-in function *rs2_deproject_pixel_to_point*, which is provided in *pyrealsense2*, the python library for Intel RealSense cameras.
In order to seize a potato, not only the position needs to be known, but the potatoes dimensions are required as well. Therefore the algorithm also computes the widths and lengths of the potatoes' bounding boxes, which is done by looking at the x and y coordinates of the boxes' corners and computing the distance between each corner.
Note that the object detection models output vertical or horizontal bounding boxes (no tilt). Therefore, if a potato is tilted, the bounding box's dimension does not correspond to the potato's dimensions. However, before seizing a potato, the gripper first aligns with the target potato's orientation so that the dimensions match.

3.5.2 Orientation

In order to harvest a potato with the robotic arm, information about its orientation is required. Indeed, to ensure successful seizing, the potatoes need to be grabbed by the gripper along their widths. Therefore the 2D-orientations of the potatoes are computed. No information about the orientation can be deducted from the object detector, as it only outputs vertical or horizontal bounding boxes. Henceforth, an image segmentation algorithm is first applied inside the bounding boxes in order to extract pixel-wise knowledge about the potatoes. Then the potato's angle is computed using further image processing techniques.

(a) Detected potato (b) Cropped bounding box (c) Gray-scale filtered image (d) Canny edge detection

Figure 3.9: Canny Edge Detection pipeline: potato is detected (a) and its bounding box is cropped (b). The crop is then converted to a gray-scale image and filtered by a bilateral filter (c). Eventually Canny Edge Detection is performed (d). The edge points can then be used as input for PCA or ellipse fitting.

(a) Detected potato (b) Cropped bounding box (c) K-Means clustering

Figure 3.10: K-Means clustering pipeline: A potato is detected (a) and its bounding box is cropped (b). The pixels are then clustered in two classes using K-Means (c). The potato class points can eventually be used as input for PCA or ellipse fitting.

3.5.2.1 Image Segmentation

The image segmentation techniques are only applied in the bounding boxes of the detected objects (for this task the bounding boxes are actually extended by a few pixel to make sure that they contain the whole potato). This allows to save computational time and to simplify the algorithm, since it is known that only one potato is present in each box. In the scope of this project the following image segmentation techniques are considered:

- **Canny edge detection:**
  The rgb-image is first converted into a grayscale image. A bilateral filtered is then applied in order to reduce noise and Canny Edge Detection is performed on the image (by using the *canny* function provided in OpenCV). Finally, a grayscale threshold is applied to find the contour of the potato: pixels above the threshold are considered as contour points, others pixels are discarded. (See Fig. 3.9)

- **K-means Clustering:**
  K-Means clustering is performed on RGB data of the cropped bounding box. Practically, the k-means function provided by the *sklearn* library is used, with two clusters (to discriminate between potato and "non-potato"). (See Fig. 3.10)

Note that those image segmentation techniques are meant to be performed on RGB data, but can either be applied on the images provided by the camera's RGB channel or on the colorized depth images extracted from the camera's depth channel.

3.5.2.2 Orientation estimation

After image segmentation techniques are applied on the cropped bounding boxes, the potatoes' orientations need to be estimated. To do so, two approaches are considered:

- **PCA**:
  PCA can be applied on extracted contour point (in case of Canny Edge segmentation) or potato pixels (in case of K-Means clustering). The returned principal axis corresponds to the length axis of the potato, and the width axis is perpendicular.
- **Ellipse Fitting**:
  Another possibility to estimate the orientation of a potato is to extract its contour points and to fit an ellipse in a least-square sense around those points. The length axis of the ellipse approximates the length axis of the potato. In practice, ellipse fitting is performed using the *fitEllipse* function available in OpenCV.

3.6 Integration in global project

In the Root System, the communication between modules is handled by a ROS-master. The information computed by the developed perception algorithm is sent to a ROS-topic at a 3 Hz rate via a ROS-publisher. This information will then be accessed by the controller of the robotic arm and gripper and used to harvest a potato.
The ROS-message sent to the topic (see Table 3.2) contains the relevant information for successful harvesting discussed in the previous chapters, namely the 3D-coordinates and orientation of the potato, as well as the width and height of the bounding box. The following conventions are used regarding the positional data: the 3D-coordinates axis origin is the camera, the x axis is horizontal and increasing from left to right, the y axis is vertical and increasing from the top to the bottom and the z axis is increasing towards the field of view of the camera. The angle is communicated as a trigonometric angle.
Importantly, two supplementary variables are added to the ROS-message. The first one is a boolean that communicates if a potato is detected or not. This variable is used by the robot's controller during an initial scanning phase, in which the robotic arm moves the camera along the hanging roots in search of a potato. Once a potato is detected the boolean is set to 1, the controller stops the arm and initiates harvesting.
The second added variable is a boolean that informs if the depth data of the targeted potato is exploitable or not. In case the camera is too close to the selected potato, the depth data is incoherent and the 3D-localization of the potato is incorrect. In such case, the robotic arm needs to move backwards until exploitable depth data can be retrieved.

Table 3.2: ROS-message content

Variable	**Type**	**Description**
bool_detection	uint32	Boolean to communicate if a potato is detected or not
bool_depth	uint32	Boolean to communicate if the depth data of the camera is exploitable or not.
x	float32	x coordinate in meters
y	float32	y coordinate in meters
z	float32	z coordinate in meters
height	float32	height of the bounding box in meters
width	float32	width of the bounding box in meters
angle	float32	trigonometric angle of the potato's principal axis in radians.

4. Results

4.1 Object Detection

The object detection models performances are evaluated on the 30 images of the test data-set (see section 3.3.2.4). The performance's assessment relies on two parameters: the mean Average Precision (mAP) [18] and the computational time. The mAP is computed based on precision-recall curves (see Appendix B) and the computational time is calculated as the mean computational time per image over the test data-set (see Table 4.1). The tested models are trained over approximately 10'000 episodes.

Table 4.1: Performances of the object detection models

Object Detection Model	**Feature extractor**	**Mean run-time per frame** [s]	**mAP** [%]
R-FCN	ResNet101	0.98	93.15
Faster-RCNN	ResNet101	4.10	93.03
Faster-RCNN	Inception v2	0.64	93.01
SSD300	Mobilenet v2	0.07	92.68
SSD300	Inception v2	0.12	93.01
SSD512	Mobilenet v2	0.13	92.87
SSD512	Inception v2	0.20	91.75

4.2 Pose Computation

4.2.1 3D localization

To test the 3D-localization performance, a potato is placed on a setup where the x, y and z distances with respect to the camera can be measured and the difference between the computed positions and the measured positions is calculated. 2 different potatoes are placed at 16 different positions, which leads to the results listed in Table 4.2. The complete measurements can be found in Appendix C.

Table 4.2: 3D localization performances

Axis	**Error std** [deg]	**Mean error** [deg]	**Mean absolute error** [deg]
x	0.66	- 0.17	0.53
y	0.67	0.01	0.57
z	0.85	0.95	1.01

4.2.2 Orientation

The following combinations of image segmentation and orientation estimation techniques (see section 3.5.2) are tested:

- Canny Edge Detection (on RGB image) + PCA
- Canny Edge Detection (on colorized depth) + PCA
- Canny Edge Detection (on RGB image) + ellipse fitting
- Canny Edge Detection (on colorized depth) + ellipse fitting
- K-Means (on RGB image) + PCA
- K-Means (on colorized depth) + PCA
- K-Means (on RGB image) + ellipse fitting
- K-Means (on colorized depth) + ellipse fitting

To assess the accuracy of the orientation estimation methods, establishing a ground truth of the potatoes' angles is required . Since it is difficult to make accurate orientation measurements on the Root System, the methods are evaluated on another setup with similar conditions. The camera is fixed around 30 cm above a white background. The distance of 30 cm approximately corresponds to the distance between the crops and the camera during the scanning phase. Angles with respect to the camera are drawn on the white background with a protractor. Photos with potatoes placed at specific angles can then be taken with the camera. In order to have a representative sample of what could be seen on the Root System, the RGB images and colorized depth data of 5 different potatoes positioned at 4 different angles (0, 45, 90 and 135 degrees) are stored. Brown strings are laid on the set up to simulate crop's roots. In order to evaluate the segmentation techniques' sensitivity to the contrast, each photo is taken under two different light conditions, once under low and once under high luminosity. This results in 40 RGB and 40 colorized depth images that are used to test the different orientation computation techniques.

Table 4.3: Pose estimation performances

Method	Std on high luminosity data [deg]	Mean absolute error on high luminosity data [deg]	Std on low luminosity data [deg]	Mean absolute error on low luminosity data [deg]	Total std [deg]	Total mean absolute error [deg]	Mean computational time [s]
Canny Edge Detection (on RGB image) + PCA	29.8	22.3	35.4	27.8	33.8	25.05	0.013
Canny Edge Detection (on colorized depth) + PCA	13.74	7.2	23.2	9.3	19.9	8.3	0.008
Canny Edge Detection (on RGB image) + ellipse fitting	26.2	16.5	36.3	28.6	31.8	22.5	0.014
Canny Edge Detection (on colorized depth) + ellipse fitting	9.2	6.9	22.4	12.4	17.8	9.6	0.010
K-Means (on RGB image) + PCA	7.3	5.7	23.2	14.3	17.6	10.0	1.55
K-Means (on colorized depth) + PCA	8.1	5.1	6.3	4.5	7.2	4.8	1.50
K-Means (on RGB image) + ellipse fitting	21.9	14.6	35.6	28.2	30.13	21.4	2.10
K-Means (on colorized depth) + ellipse fitting	8.54	5.81	21.1	10.4	16.2	8.11	1.82

5. Discussion

5.1 Selected solution

5.1.1 Object Detection

As can be seen in Table 4.1, all tested architectures achieve similar (and high) mAP. Commonly, region-based models such as R-FCN and Faster-RCNN are expected to perform better than single shot detectors regarding accuracy [7] (they actually do perform slightly better in this case as well, but nothing significant). This can be explained as in this application all training images are very similar : the training images are taken with the same camera (i.e. same size, saturation, resolution, etc.), the background is always the same (a white sheet), the luminosity is the same, the object to detect are very similar (same sort of potatoes), etc.. Consequently, the object detection is easy to learn and even SSDs reach high mAPs. Training the networks on such specific and controlled environments allows to obtain great performances on that very same environment, however the drawback is that the object-detectors would probably perform very poorly in other environments. Therefore, for applications in which the environment is more heterogeneous or noisy, SSDs would probably show significantly lower accuracy.

Regarding the speed performance, the following observations can be made:

- Concerning features extractors, Mobilenet v2 result in slightly faster computations than Inception v2 when used with SSDs. When used with Faster-RCNNs, Inception v2 yields significantly better speed performances than ResNet101.
- SSDs show the best speed performances and allow real-time object-detection.
- SSD300 is slightly faster than SSD512, as the input data is smaller.

Since all tested models yield comparable accuracy results, the choice of the architecture is made based on the speed performances. The fastest model is selected, namely: **SSD300 with Mobilenet v2 as features extractor**.

One of the advantage of this implementation is that it allows real-time object detection. Real-time performances are not really required for the considered application, nevertheless it can simplify the control strategy, since the controller would not need to deal with latency. Furthermore, in case the object-detection or pose computation fail on one frame, it can be corrected much quicker if the algorithm works in real-time.

Finally, another advantage of using neural networks for object detection, is that this can easily be translated to other environments or to detect other vegetables. The only required adaptation is to retrain or fine-tune the object-detection network with new data.

5.1.2 Pose computation

First, as can be seen in Table 4.2, the mean absolute errors on the x, y and z coordinates do not exceed 1.01 cm, whereas the errors' standard deviations are all smaller than 1 cm. Consequently, the accuracy and precision of the 3D-localization is more than satisfying for the task of harvesting potatoes. Indeed, tests on the simulated Root System setup showed that if the gripper seizes a potato with a 1cm magnitude order offset on all three dimensions, the potato can still be successful harvested. This is due to the fact that 1 cm is not much compared to the potato size; for other smaller fruits or vegetables, more accurate and precise localization might be needed.

Concerning the results of the pose estimation methods (see Table 4.3), the following conclusions can be drawn:

- Concerning image segmentation methods:
  - Canny Edge Detection allows real-time image segmentation, whereas K-Means does not.
  - K-Means-based image segmentation results in better accuracy and precision than Canny Edge Detection.
- Concerning the orientation estimation technique:
  - When used after Canny Edge Detection, ellipse fitting and PCA show similar performances. Though, PCA tends to be slightly less accurate and slightly faster.
  - When used after K-Means, ellipse fitting leads to worse results than PCA. This might be because K-Means occasionally outputs highly non-elliptical clusters (due to noise, wrong classification of some pixels, etc.).
- Concerning the input data:
  - Computations based on colorized depth data yield much better results for all methods. This is due to the fact that the color contrast on colorized depth data is higher than on raw RGB-images, which leads to better image segmentation. Moreover, the depth data is less sensitive to the noise induced by the potatoes roots (most of the time the roots are to thin to be detected by the IR-stereo sensor).
  - Computations based on RGB-data are much more sensitive to luminosity than depth data. Indeed, the depth is sensed by active sensors (sending their own IR light) and does not rely on ambient light.

Knowing that the object-detection and selection/tracking (see section 5.1.1) would benefit from real-time computations, K-Means image segmentation was not retained. The selected solution for this this project is to use **colorized depth images as input data, Canny Edge Detection for image segmentation and PCA to estimate the angle of the potatoes**. This design allows real-time orientation computations with acceptable accuracy (mean absolute error = 8.3 [deg]) and precision (std = 19.9 [deg]) performances (note that PCA could be replaced by ellipse fitting without changing significantly the performances).

Since potatoes are rather easy to seize, the achieved accuracy and precision are satisfying for the specific needs of this project. In fact, given the prior knowledge that potatoes are growing in aeroponics under gravity, it is even questionable whether computing the potatoes' orientation is really required. In such environments potatoes almost always grow with their length axis oriented vertically. Consequently, the gripper could probably always successfully seize the potatoes simply by considering that they all have a 90 degree (trigonometric) orientation. Even if the orientation has a small offset with the vertical, the gripper would simply push the potato in a vertical angle while seizing them and still harvest them successfully.
Nevertheless, it is important to note that if the algorithm is used on another setup, for other vegetables or fruits for which the seizing is more sensitive to orientation, angle computation might be crucial. For other crops, if

more accuracy and precision is required, K-Means image segmentation might be considered (however, in such case real-time computations are impossible and the control would need to be adapted in this sense). All in all, the choice of the orientation algorithm is mostly a trade-off between accuracy/precision and computational time and needs to be chosen based on the application.

5.2 Improvement and further work possibilities

Concerning the 3D-localization of the potatoes, the obtained accuracy and precision are satisfying for the specific task of harvesting potatoes, but if the algorithm wants to be used for picking other sorts of crops, a finer estimation might be required. One way to improve the 3D-localization is to calibrate the camera using *Intel RealSense D400 Series Dynamic Calibration Tool*, which was not done in this project.
Beside this, an other improvement would be to manually offset the computed 3D coordinates in the code in order to correct the mean error. As a matter of fact, Table 4.2 shows that the potatoes seem to be perceived around 1 cm further than what they really are (the mean error on z is equal to 0.95 cm). Subtracting 0.95 cm to the z coordinate's computation should improve the 3D-localization's accuracy. However, before performing such manual corrections, more data is required in order to obtain a better estimation of the true offsets.

Another improvement possibility would be to explore other object detection architectures that are not available as pre-trained models in the Tensorflow detection zoo. As a matter of fact, Table 4.1 suggests that R-FCN with Inception v2 or Mobilenet v2 as features extractor could yield better speed performances and might be able to perform real-time object detection and compete with SSDs.

In this project, object detection only makes use of the RGB data provided by the camera, even though depth data is available. Recent research has developed RGB-D object-detection frameworks and suggests that incorporating depth data to the models might improve detection's accuracy. [24], [25], [26]. Such models were not explored in the scope of this work since most of the online-available training frameworks and pre-trained models are strictly based on RGB-data.
Furthermore, deep convolutional networks for pose estimation have been developed both based on RGB [28] and RGB-D data [27]. Such approaches could improve the rather poorly performing pose estimation methods used in this project, without compromising the real-time implementation, as some methods such as YOLO6D [29] support real-time pose estimation. Again, a more precise and accurate pose estimation is not required in the scope of this work, but might be interesting if the algorithm is translated on another setup for harvesting other fruits or vegetables.
Additionally, another simpler approach to improve the accuracy of the pose estimation would be to implement mask-RCNNs [30]. Mask-RCNNs are deep convolutional networks that perform instance segmentation, meaning that, instead of a bounding box (as in object detection), the output of the network is a pixel-wise segmentation of the objects. This approach would allow to perform image segmentation with higher accuracy than the technique presented in this work (K-Means and Canny Edge Detection). However, the drawbacks are that mask-RCNN are usually slower than object-detection networks (at least slower than SSDs) and that the image labeling is highly time demanding as the exact shapes of the objects need to be specified.

Finally, the solution developed in this project does not take into account potential obstacles. Indeed, in a setup where potatoes grow in aeroponics, there might be some roots in the way to the targeted potato that could complicate harvesting. *Hemming et al.* [31] proposed an obstacle detection technique based on stereo cameras for automated harvesting of sweet pepper. Such a method could be implemented for this project as well (it would require to add some extra cameras) or, alternatively, the obstacle detection could be implemented based on the depth data gathered by the RealSense Camera.

6. Conclusion

To conclude, this work proposes a solution for detection and pose estimation of potatoes growing in aeroponics. After exploring and testing different state-of-the-art approaches, the selected design is to use a SSD300 deep-convolutional network with Mobilenet v2 as feature extractor to perform potato detection. Selection of a target potato to be harvested is carried out base on potatoes' ripeness and distances to the camera, while tracking of the target is implemented using a deep learning-based tracker. The 3D-coordinates, sizes and orientations of the root vegetables are computed based on depth data. For orientation estimations, image segmentation using Canny Edge Detection is performed followed by PCA to find the principal axis of the potatoes. The developed solution allows real-time potatoes' perception and results in enough positional accuracy and precision to perform successful harvesting.
Importantly, the proposed design can be modulated for perception of other crops on different environments. To do so, the object-detection networks needs to be retrained or fine-tuned on new data. In case, the algorithm wants to be translated to an application that requires significantly different pose estimation precision and accuracy or speed performances, the design can adapted by replacing parts of the selected solution by some of the many approaches that were tested in this work.

List of Figures

2.1 The Root System 3
2.2 Potatoes growing in aeroponics 3
2.3 Adaptation of the Root System 4

3.1 The RealSense camera mounted on the gripper 5
3.2 Algorithm Architecture 6
3.3 Display output of the perception algorithm 7
3.4 Faster-RCNN architecture 9
3.5 R-FCN architecture 10
3.6 SSD architecture [23] 10
3.7 Structure of the tracking algorithm 12
3.8 Example of the selection/tracking algorithm 13
3.9 Canny Edge Detection pipeline 14
3.10 K-Means clustering pipeline 14

C.1 3D-localization measurements 30

D.1 Object detection with different models 33

List of Tables

3.1 Principal Training Parameters 11
3.2 ROS-message content 15

4.1 Performances of the object detection models 16
4.2 3D localization performances 16
4.3 Pose estimation performances 18

Bibliography

[1] J. Blasco, N. Aleixos, and E. Moltó, *Machine vision system for automatic quality grading of fruit*, Biosystems Engineering, vol. 85, no. 4, pp. 415–423, 2003.

[2] S. Cubero, M. P. Diago, J. Blasco, J. Tard Águila, B. Millán and N. Alexios, *A new method for pedicle/peduncle detection and size assessment of grapevine berries and other fruits by image analysis*, Biosystems Engineering, vol. 117, pp. 62–72, 2014, image Analysis in Agriculture, http://www.sciencedirect.com/science/article/pii/S1537511013000950

[3] D. Dey, L. Mummert, and R. Sukthankar, *Classification of plant structures from uncalibrated image sequences*, Applications of ComputerVision (WACV), 2012 IEEE Workshop on. IEEE, 2012, pp. 329–336.

[4] C. McCool, I. Sa, F. Dayoub, C. Lehnert, T. Perez, and B. Upcroft, *Visual Detection of Occluded Crop: for automated harvesting*, 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, 2016, pp. 2506-2512, doi: 10.1109/ICRA.2016.7487405.

[5] K. Yamamoto, W. Guo, Y. Yoshioka, and S. Ninomiya, *On Plant Detection of Intact Tomato Fruits Using Image Analysis and Machine Learning Methods*, Sensors, 2014.

[6] S. T. Nuske, S. Achar, T. Bates, S. G. Narasimhan, and S. Singh, *Yield estimation in vineyards by visual grape detection*, Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots andSystems (IROS '11), September 2011

[7] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, *Object Detection with Deep Learning: A Review*, arXiv:1807.05511, 2019,

[8] J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers, and A.W.M. Smeulders, *Selective Search for Object Recognition*, International Journal of Computer Vision, 2012

[9] W. Liu1, D. Anguelov, D. Erhan, C. Szegedy, S. Reed4, C.-Y. Fu1, A. C. Berg1, *SSD: Single Shot MultiBox Detector*, arXiv:1512.02325v5, 2016

[10] J. Redmon and A. Farhadi, *Yolo9000: better, faster, stronger*, arXiv:1612.08242, 2016.

[11] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, *You Only Look Once: Unified, Real-Time Object Detection*, arXiv:1506.02640v5, 2016

[12] R. Agarwal, *Demystifying Object Detection and Instance Segmentation for Data Scientists*, Towards Data Science, 2019, https://towardsdatascience.com/a-hitchhikers-guide-to-object-detection-and-instance-segmentation-ac0146fe8e11

[13] S. Ren, K. He, R. Girshick, J. Sun, *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*, arXiv:1506.01497, 2016

[14] L. Ma, F. Zhang, and L. Xu, *Fruit Detection Using Faster R-CNN Based on Deep Network*, Proceeding of the Second International Conference on Smart Vehicular Technology, Transportation, Communication and Applications, 2018

[15] I. Sa, Z. Ge, F. Dayoub, B. Upcroft, T. Perez, and C. Mccool, *DeepFruits: A Fruit Detection System Using Deep Neural Networks*, Sensors, 2016

[16] Y. Onishi, T. Yoshida2, H. Kurita, T. Fukao, H. Arihara, and A. Iwai, *An automated fruit harvesting robot by using deep learning*, Robomech Journal, 2019

[17] N. Lamb, M. C. Chuah, *A Strawberry Detection System Using Convolutional Neural Networks*, IEEE International Conference on Big Data, 2018

[18] S.M. Beitzel, E.C.Jensen, and O.Frieder, *MAP*, in: L. Liu, M.T. Özsu, *Encyclopedia of Database Systems*, Springer, Boston, MA, 2009.

[19] J. Dai, Y. Li, K. He, J. Sun, *R-FCN: Object Detection via Region-based Fully Convolutional Networks*, arXiv:1605.06409v2, 2016

[20] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, *MobileNetV2: Inverted Residuals and Linear Bottlenecks*, arXiv:1801.04381v4, 2019

[21] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate*, arXiv:1502.03167, 2015.

[22] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, arXiv:1512.03385, 2015

[23] J. Huang et al., *Speed/accuracy trade-offs for modern convolutional object detectors* , arXiv:1611.10012v3, 2017

[24] S. Gupta1, R. Girshick, P. Arbelaez, and J. Malik, *Learning Rich Features from RGB-D Images for Object Detection and Segmentation* , arXiv:1407.5736v1, 2014

[25] S. Zia, B. Yüksel, D. Yüret, and Y. Yemez, *RGB-D Object Recognition Using Deep Convolutional Neural Networks*, IEEE International Conference on Computer Vision Workshops, 2017

[26] I. R. Ward, H. Laga, and M. Bennamoun, *RGB-D image-based Object Detection: from Traditional Methods to Deep Learning Techniques*, arXiv:1907.09236, 2019

[27] Y. He, W. Sun, H. Huang, J. Liu, H. Fan, and J. Sun, *PVN3D: A Deep Point-wise 3D Keypoints Voting Network for 6DoF Pose Estimation*, arXiv:1911.04231v2, 2020

[28] S. Zakharov, I. Shugurov, and S. Ilic, *DPOD: 6D Pose Object Detector and Refiner*, arXiv:1902.11020v3, 2019

[29] B. Tekin, S. N. Sinha, and P. Fua, *Real-Time Seamless Single Shot 6D Object Pose Prediction*, arXiv:1711.08848v5, 2018

[30] K. He, G. Gkioxari, P. Dollár, and R. Girshick, *Mask R-CNN* , arXiv:1703.06870, 2018

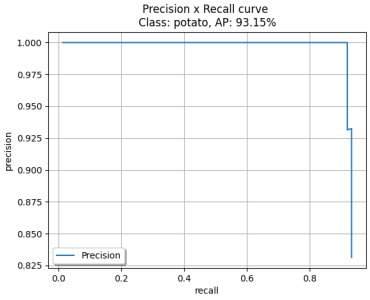
[31] J. Hemming, C. Wouter Bac, B. A.J. van Tuijl, R. Barth,J. Bontsema, and E. Pekkeriet, *A Robot for Harvesting Sweet-Pepper in Greenhouses* , International Conference of Agricultural Engineering, Zurich, 2014

A. Intel RealSense D415 datasheet

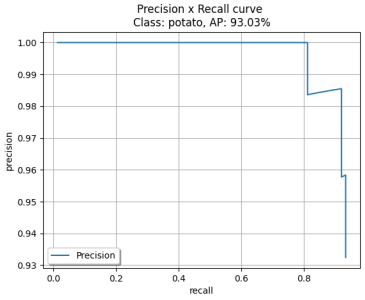
The datasheet can be found at the following link:

https://www.intelrealsense.com/wp-content/uploads /2020/06/Intel-RealSense-D400-Series-Datasheet-June-2020.pdf

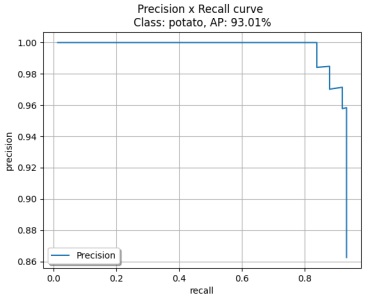
B. Precision-Recall Curves



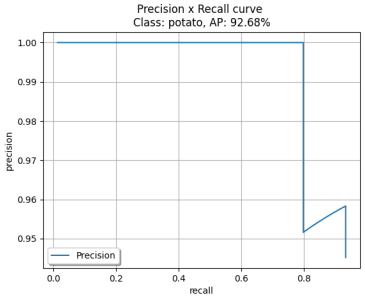
(a) F-RCN ResNet101



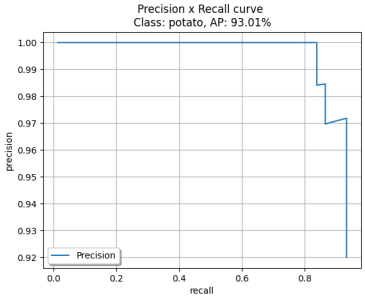
(b) Faster-RCNN ResNet101



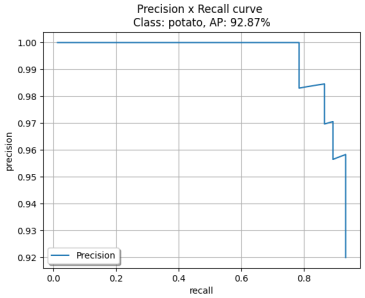
(c) Faster-RCNN Inception



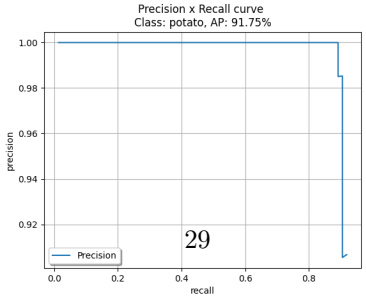
(d) SSD300 Mobilenet



(e) SSD300 Inception



(f) SSD512 Mobilenet



(g) SSD512 Inception

C. 3D-Localization Data

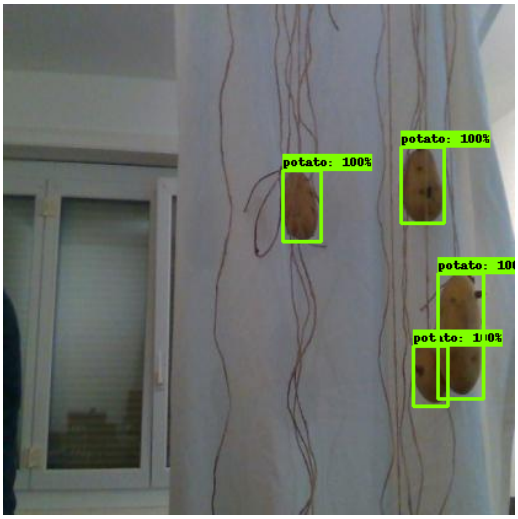
Potato #	Measured values (ground truth)			Computed values		
	x [cm]	y [cm]	z [cm]	x [cm]	y [cm]	z [cm]
1	0	10	33	0.12	9.31	33.4
1	0	-10	33	0.06	-9.72	34
1	10	0	33	8.98	0.01	34.5
1	-10	0	33	-9.6	0.57	33.2
1	10	10	33	9.08	9.48	34
1	-10	-10	33	-9.97	-9.18	33.6
1	10	-10	33	9.61	-9.5	33.3
1	-10	10	33	-9.32	9.67	33
2	0	10	31.5	-0.59	9.14	32.4
2	0	-10	31.5	-0.1	-8.74	33.4
2	10	0	31.5	9.17	0.18	33.5
2	-10	0	31.5	-8.95	0.01	32.2
2	10	10	31.5	8.87	9.48	33.1
2	-10	-10	31.5	-8.64	-9.52	32.8
2	10	-10	31.5	9	-9.04	33.9
2	-10	10	31.5	-8.7	9.03	32.1
1	0	10	46.5	-0.22	9.1	46.3
1	0	-10	46.5	-0.4	-9.29	48.4
1	10	0	46.5	9.94	-0.31	47.6
1	-10	0	46.5	-9.93	0.48	47
1	10	10	46.5	9.12	9.28	46.7
1	-10	-10	46.5	-9.81	-9.15	49.6
1	10	-10	46.5	9.28	-9.51	48.9
1	-10	10	46.5	-10.21	9	47.1
2	0	10	45.5	-0.5	9.6	44.9
2	0	-10	45.5	-0.07	-9.95	47
2	10	0	45.5	8.99	0.01	46.2
2	-10	0	45.5	-10.39	0.32	45.9
2	10	10	45.5	10.19	8.58	45.5
2	-10	-10	45.5	-9.99	-9.15	47.1
2	10	-10	45.5	9.17	-9.43	46.1
2	-10	10	45.5	-9.63	9.74	45.9

Figure C.1: 3D-localization measurements

D. Object detection with different models

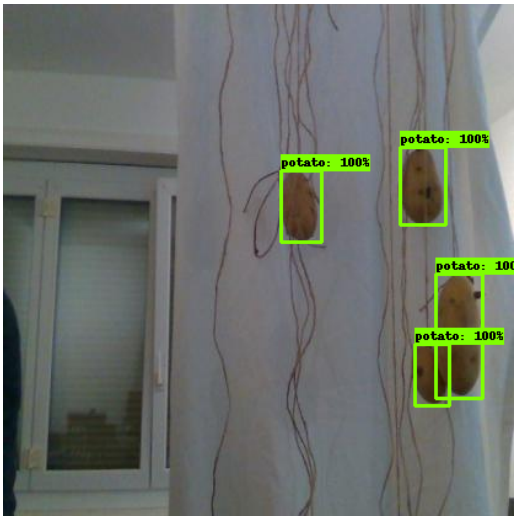
(a) Raw image 1

(b) Raw image 2



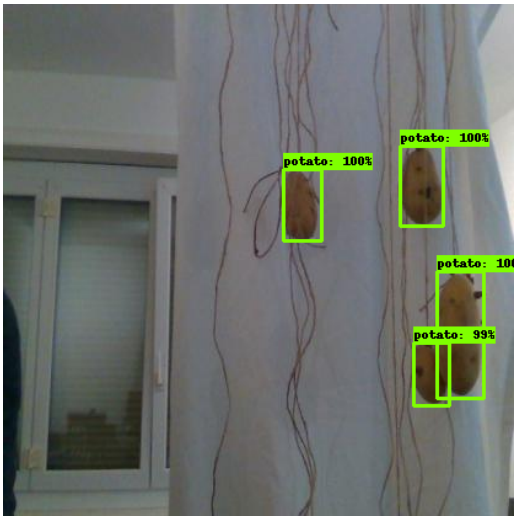
(c) R-FCN ResNet101

(d) R-FCN ResNet101



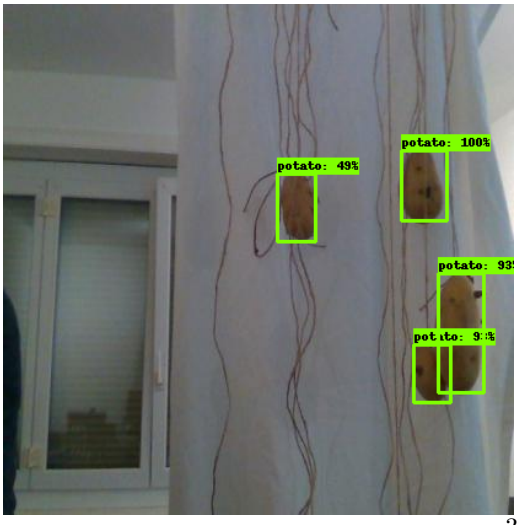
(e) Faster-RCNN ResNet101

(f) Faster-RCNN ResNet101

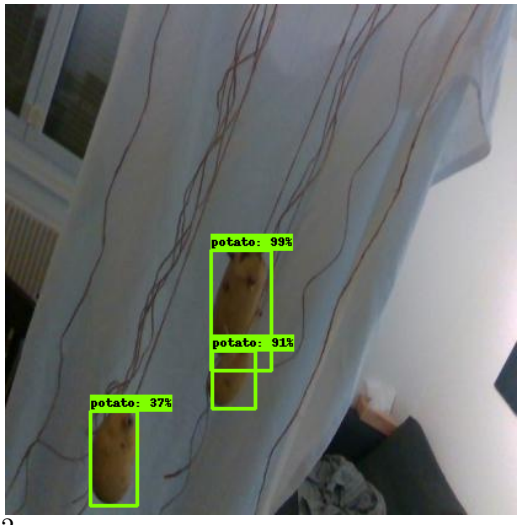


(g) Faster-RCNN Inception v2

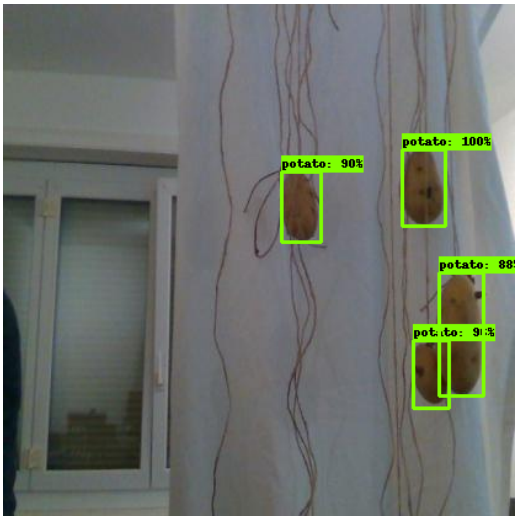
(h) Faster-RCNN Inception v2



(i) SSD300 Mobilenet v2

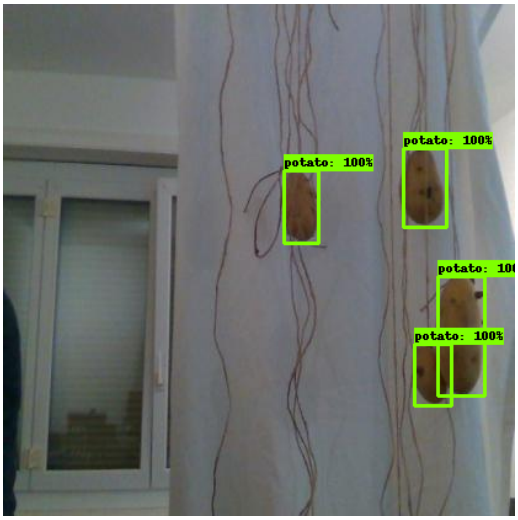


(j) SSD300 Mobilenet v2

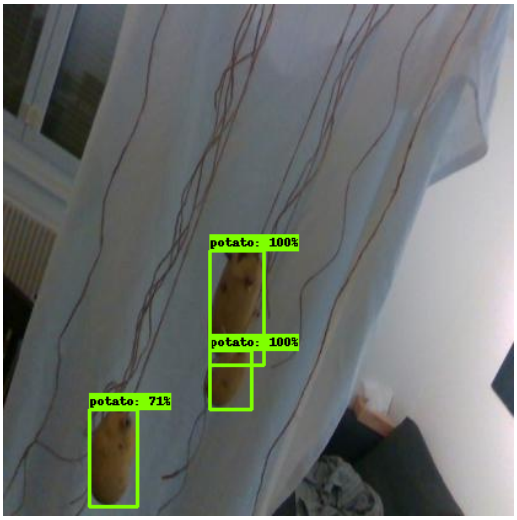


(k) SSD300 Inception v2

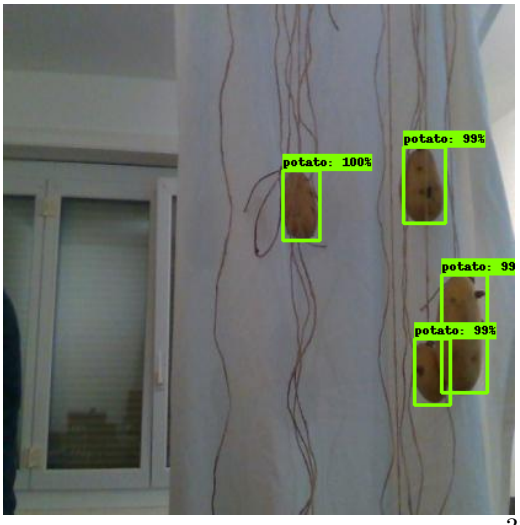
(l) SSD300 Inception v2



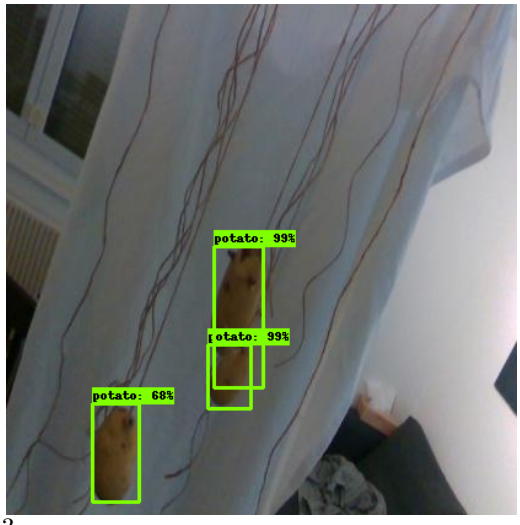
(m) SSD512 Mobilenet v2



(n) SSD512 Mobilenet v2



(o) SSD512 Inception v2



(p) SSD512 Inception v2

Figure D.1: Object detection with different models