# Active perception for localization and manipulation of fruits in bulk

by

Victor Dramé

---

MICRO-598: Master's project in robotics

| | |
|---|---|
| Proffessor: | F. Mondada |
| Supervisor: | S. Calinon |
| Project Duration: | 03, 2023 - 07, 2023 |
| Faculty: | School of Engineering, EPFL |


# Contents

# Nomenclature

## Abbreviations

| Abbreviation | Definition |
|---|---|
| ROS | Robotic Operating System |
| GUI | Graphical User Interface |
| segmap | segmentation map |
| YOLO | You Only Look Once |
| SAM | Segment Anything Model |
| ROI | Region Of interest |
| IoU | Intersection over Union |
| RGBD | Red, Green, Blue, Depth |

# 1

# Introduction

Robust manipulation of unknown object is still an unsolved task in robotics. However, the performances of proposed approaches become higher and higher and the solutions to this challenge have applications in various industries. This work is focused on applications related to fruits and vegetables. At the production phase, they needs to be picked from the tree or plant, while at retail there is a need for pick and place operations from fruit/vegetables crates to packaging for the consumer. This project will investigate the latter, where the fruit needs to be selected and arranged. Specialised automation systems already exist for this type of problem, but they often rely on multiple machines to handle each step separately. While this works well for large scale production of identical products, it becomes impractical for small scale production or when the product changes frequently. In this case, manual labour is used, which is monotonous and tedious.

This project aims to investigate the use of active perception for the manipulation of objects in bulk. It will use a robotic arm manipulator equipped with an RGBD camera. Since one of the challenges is to have a low-cost approach, we need to minimize effort when changing setup and grasping target by dividing the method into multiple modules.

# 2

# Literature review

## 2.1. Vision

Information about the external environment is necessary in order to know what to grasp. This data can be preregister or provided by multiple types of sensors, such as touch tensors, camera or lidar. For this project we chose a RGBD camera as it provides both visual feedback and depth information. Computer vision is a well-studied topic and still attracts new research, making it easy for us to leverage the current state-of-the-art. Image segmentation is generally the method of choice for grasping estimation, as it allows shape estimation and handles occlusion.

### 2.1.1. YOLO

YOLO (You Only Look Once) is a popular and widely used algorithm for real-time object detection with good accuracy. It has benefited from many improvement, allowing it to remain at the relevant throughout the years. YOLOv8 [8] is the latest version released by Ultralytics. YOLOv8-seg even allows object segmentation to leverage the YOLOACT method [1] for segmentation. It provides lightweight and accurate segmentation that can run fast even on limited computing resources.

| Model | Size (pixels) | $mAP^{box}_{50-95}$ | $mAP^{mask}_{50-95}$ | Speed CPU ONNX (ms) | params (M) | FLOPs (B) |
|---|---|---|---|---|---|---|
| YOLOv8n-seg | 640 | 36.7 | 30.5 | 96.1 | 3.4 | 12.6 |
| YOLOv8s-seg | 640 | 44.6 | 36.8 | 155.7 | 11.8 | 42.6 |
| YOLOv8m-seg | 640 | 49.9 | 40.8 | 317.0 | 27.3 | 110.2 |
| YOLOv8l-seg | 640 | 52.3 | 42.6 | 572.4 | 46.0 | 220.5 |
| YOLOv8x-seg | 640 | 53.4 | 43.4 | 712.1 | 71.8 | 344.1 |

**Table 2.1:** YOLOv8 segmentation model performances in function of model size. Data extracted from Ultralytics documentation [8]

### 2.1.2. SAM & FastSAM

During the latter part of the project, META AI released Segment-Anything-Model (SAM), which uses an image encoder to generate an image embedding, which is then queried by a variety of input prompts to produce an input mask. The input prompt can be either a pixel or a box. They present a modular segmentation method that can efficiently perform zero-shot detection. One of the main advantages of SAM is its ability to perform zero-shot segmentation i.e segmentation on objects not specifically trained for. This model is computationally intensive and could not run on the available local machine but it can be used as foundation model to trained lighter methods. Moreover, by

replacing the computationally heavy transform (ViT) part of SAM with a convolutional neural network (CNN), Zhao et al. proposed FastSAM [14], a lightweight version of SAM. Their approach is based on the YOLOv8-seg model and leads to a 50 times faster inference time with similar performance, making the model lightweight and efficient.

| Method | AP | APS | APM | APL |
|---|---|---|---|---|
| ViTDet-H [23] | 51.0 | 32.0 | 54.3 | 68.9 |
| Zero-shot transfer methods (segmentation module only): | | | | |
| SAM | 46.5 | 30.8 | 51.0 | 61.7 |
| FastSAM | 37.9 | 23.9 | 43.4 | 50.0 |

**Table 2.2:** Comparison of methods on COCO dataset, data from [14]

## 2.2. Grasping pose estimation

There are several approaches to vision-based robot grasping. They often consist of three main parts, namely object localisation, object pose estimation and grasp estimation. However, some approaches allow to remove the object pose estimation. We will discuss different techniques and their use in the following subsections. In order to fit the available material in the laboratory, we will focus on approaches using a parallel gripper. However other approaches using suction cups or a multi-finger gripper also represent a big research topic and are attracting interest.

The approaches are divided into two categories based on the estimation method: those that use a data-based approach, often using deep learning, and others that rely more on a traditional method that exploits the geometric properties of the data.

### 2.2.1. 2D planar grasps

The 2D planar grip method evaluates either the grip contact point or an oriented rectangle. Both approaches are similar, however the oriented rectangle approach considers more points for grasp selection. Mohit Vohra and Al. proposed a method using traditional method for evaluating grasp candidates [12] reported a grasp success rate of 77.03% on shared object in dense clutter and 88.16% on isolated objects. Similar performances of 85.47% are reported on the Jacquard grasping dataset for methods that use deep learning to evaluate grasp candidates, such as the one proposed by Depierre and Al. [5]. 2D methods often report real-time performance, which is highly valuable for robotics. However, these approaches don't make full use of the 6 Dof arm capacity and could pose a problem. The authors report lower performances than current methods leveraging 3D input.

### 2.2.2. 3D input data

Uren and Al. [11] assume both a known object shape and pose. From this information, they generate grasp samples and then refine them using diffusion models. Using the full object shape, Weng and Al. [13] introduce Neural Grasp Distance Field (NGDF), with which they estimate the distance between the end-effector and a continuous manifold of valid grasps generated by inference. This distance can easily be interpreted as a cost and integrated with a trajectory optimiser to allow smooth grasping and other desired

behaviour. The drawback of these methods for our task is that they require at least a known object pose and a single target object.

For this project, we are focusing on unknown object methods because we want to easily adapt to a new target. These methods usually take a partial point cloud as input and can eventually perform shape completion. Breyer and Al. [3] proposed an approach that takes as input a Truncated Sign Distance Function (TSDF) representation of the scene and outputs a volume of the same spatial resolution, where each voxel contains the predicted quality, orientation, and width of a grasp performed at the centre of the cell. It was trained on a custom synthetic dataset and reports high performance in terms of accuracy and speed, with a grasp success rate of 80%, a scene clearance rate of 90%, and an inference speed of 10ms on a GeForce GTX 1080 Ti graphics card. However, this method is task-agnostic and will try to capture everything in the scene.

A better problem statement for robust fruit bin picking would be to generate grasping pose for specific objects in a pointcloud. SunderMeyer and Al. [10] proposed a similar approach where they generate grasp on a region of interest and filter prediction not grasping the target object. With a recorded grasp success of 84.3% and a clearance rate of 90.2% over diverse objects in clutter, it is a promising method for our task.

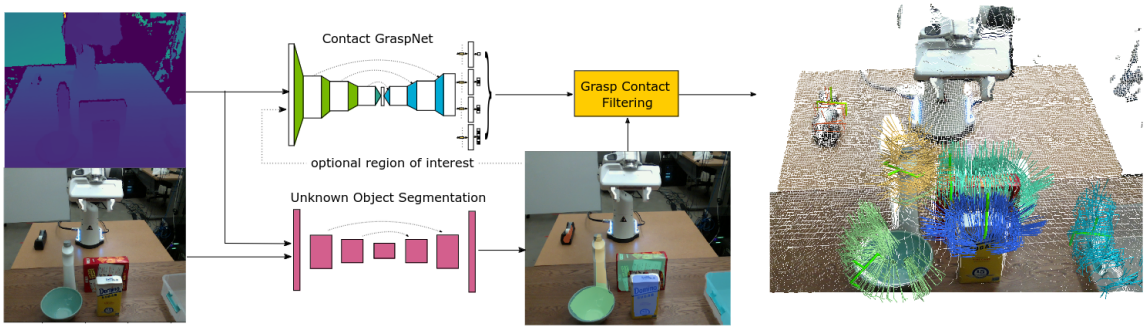



**Figure 2.1:** Contact graspnet inference pipeline from [10]. They process the entire point cloud or a region of interest around a target object. Predicted 6-DoF grips are then mapped to object segments by filtering their contact points. On the right, they show the predicted 6-DoF grasp distribution and, in bold, the most confident grasp per segment.

The approach is an end-to-end grasp generation that takes a point cloud as input and outputs a fixed number of candidate grasps with an associated score. To reduce the complexity of predicting all the parameters for a 6D pose. They simplify the problem to a prediction of four parameters

- $\mathbf{c} \in \mathbb{R}^3$, is the contact point of the grasp
- $\mathbf{a} \in \mathbb{R}^3$, $||a|| = 1$ is the approach vector
- $\mathbf{b} \in \mathbb{R}^3$, is the grasp baseline
- $\mathbf{w} \in \mathbb{R}$ is the grasp width

as shown in figure 2.3.

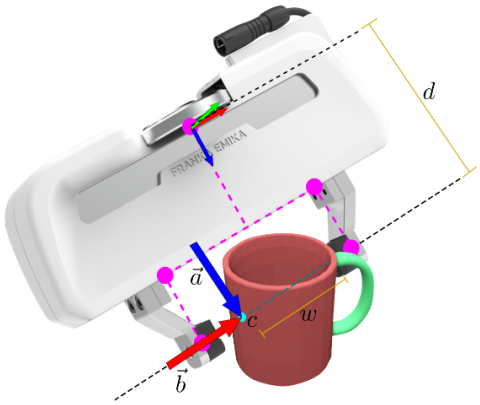



**Figure 2.2:** Contact-graspnet grasp representation from [10]: **c** is the contact point. **a** and **b** represent the 3-DoF rotation, **w** is the predicted grasp width, the constant **d** is the distance from baseline to base frame. In pink they show the five grasp points that we used to calculate the loss by comparing them with simulated grasps.

To predict these parameters, the network first uses an asymmetric U-shaped network as presented in PointNet++ [9]. Then, four heads consisting of a 1D convolution layer are used to predict the different parameters $(a, \ , b, \ , c \ , w)$. The lightweights networks allows fast inference, they reported a speed of $0.28s$ on their setup.

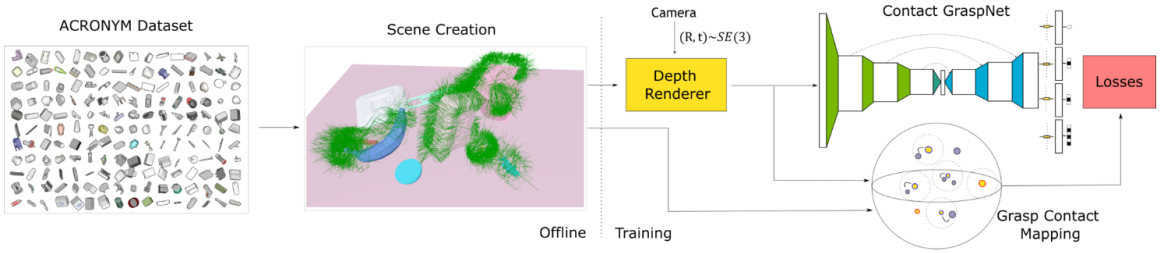



**Figure 2.3:** Contact-graspnet grasp training pipeline [10], They used randomly generated stable scenes of objects in structured clutter from the ACRONYM dataset. Grasp poses that produce gripper model collisions are removed. Resulting grasps are mapped to their contacts on the mesh surface. During training, they sample virtual cameras to render point clouds from the scenes. We consider recorded points (yellow) as positive contacts if there exists a mesh contact (blue) in a 5mm radius and associate the grasp transformation belonging to the closest mesh contact to them. These per-point annotations are used to supervise the Contact Grasp Network.

### 2.2.3. Dataset

Since the current state of the research is tilted toward data-driven method, we will have a brief overview of some recent datasets. ACRONYM [6] is a parallel grasp dataset based on the NVIDIA FLEX physics simulation engine. It contains 17.7 million grasp datasets generated on 8872 objects of 262 categories. The grasps are generated on single objects, theses objects are then disposed on a scene with light structured clutter i.e. multiple objects placed on a support surface without direct contact between them.

Zhang and al. recently built a larger scale grasping dataset named REGRAD (RElational GRAsp Dataset) which basically provided grasps generated in simulations, with a variety of labels such as 6D poses of the object, collision free and stable 6D poses, relationship label, 2D grasps and other. The aim with this dataset is to develop method generating grasps taking into account the relationship between the different object (such as spatial and support relation). The scenes they provide are in dense unstructured clut-

ters which match our problem setup.

# 3

# Method

## 3.1. Packages overview and transformation

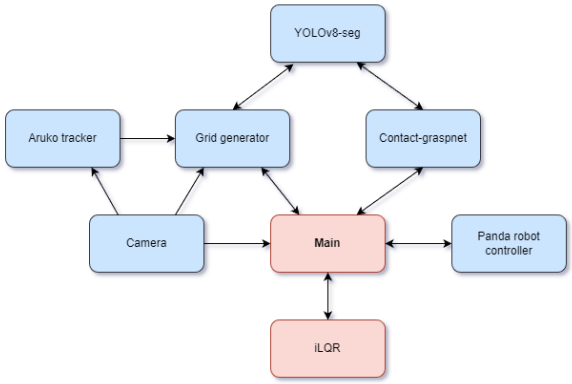



**Figure 3.1:** Overview of the different modules and their connections. Blue cells represent the ROS package, and red cells the python module.

Since our pipeline combines several modules, we implemented them in several ROS packages to have them as modular as possible. As we have different elements that are detected from and at different positions, we need to define our different transformations. The world coordinate is set as the robotic arm base coordinate. The homogeneous transformations between the different frames are listed below :

- $\mathbf{T}_{world \longrightarrow ee}$ : Transformation between the robot base and the robot end effector. It is provided by the forward kinematics.
- $\mathbf{T}_{ee \longrightarrow flange}$ : Transformation between the end-effector and the base of the panda hand. This transformation is constant and only depends on the distance between the flange and the centre of the grip area, $hand\_depth$.
- $\mathbf{T}_{flange \longrightarrow camera\_link}$ : This transformation between the flange and the camera link is done using the easy-handeye[1] ROS package, which uses the OpenCV library's Tsai-Lenz algorithm to compute the calibration matrix.
- $\mathbf{T}_{camera\_optical\_frame \longrightarrow camera\_link}$ : Transformation from the image frame to the camera frame. Provided by the camera datasheet.

[1] https://github.com/IFL-CAMP/easy_handeye

- $\mathbf{T}_{camera\_optical\_frame \longrightarrow aruco}$ : Transformation from the camera frame to the detected marker frame
- $\mathbf{T}_{grasp \longrightarrow camera\_optical\_frame}$ : Transformation from the image frame to the detected grasp frame.

In the following section we will discuss how the different packages are implemented and how they communicate with each other.

## 3.2. Vision

### 3.2.1. Object detection

For object segmentation, we use the YOLOv8-seg model due to the limited computing power of the available laptop, which also has to run the grasping pose method. The YOLO pre-training model is trained on COCO and has enough diversity in the training data set (apple, carrot, banana, orange, broccoli) for the experiments. The segmentation is implemented as a ROS service and a Python wrapper.

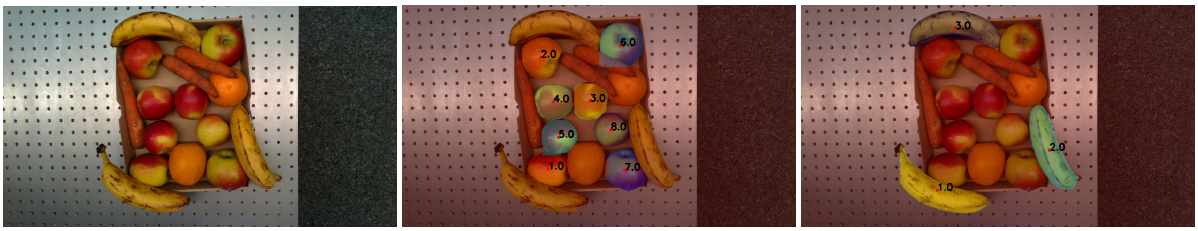



**Figure 3.2:** YOLOv8n-seg instance segmentation visualization

The method returns semantic or instance segmentation. The package takes as input a BGR image and outputs a one-channel image containing the selected mask.

### 3.2.2. Grid detection

Round fruits in retail are often packaged in a honeycomb. In order to be able to detect them easily, two aruco markers were used, as shown in figure 3.3. The aruco markers are detected directly from the camera image. First, an adaptive threshold is applied to identify marker candidates. Then, the marker candidates that match the known pattern are identified. Since we have *a priori* information about the marker size, we can estimate the transformation of the marker with respect to the camera, $\mathbf{T}_{camera \longrightarrow marker}$. This is all done using the openCV library and the standard ROS package from PAL_Robotics[2].

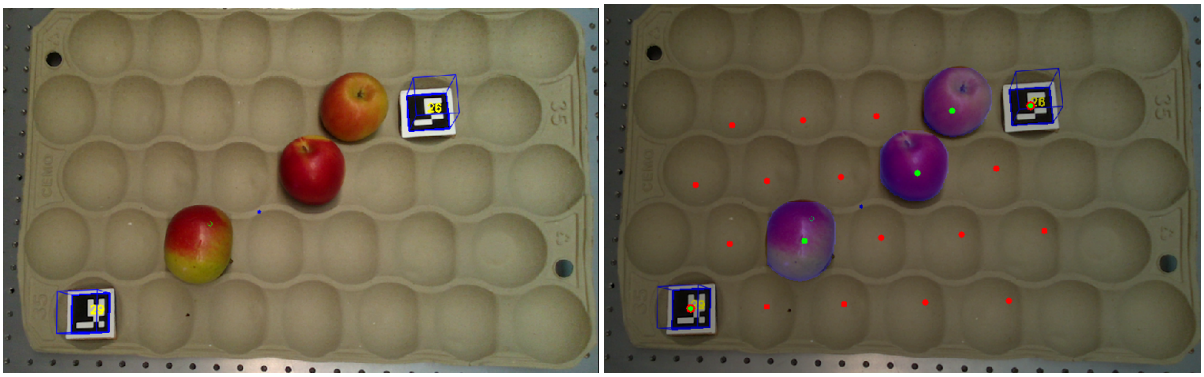


**Figure 3.3:** Grid detection input on the right and output on the left, the red dot represents the available disposal position and the green dot represents the occupied cells.

Once the aruco markers are detected, the grid is divided into a fixed number of cells, which are published as PoseArray topics with the rest of the pipeline. To detect whether some cells are occupied, the grid image is segmented and compared with the cell poses projected into the image plan. A grid with markers for each grid cell indicating whether it is occupied or not is then published as a monochrome image. In summary, this package publishes three topics:

[2]https://github.com/pal-robotics/aruco_ros

. "/grid_generator/Posearray_pub", a PoseArray containing all poses of the grids.

. "/grid_generator/disposability_grid", an image where each cell defines whether a grid is occupied or not.

. "/grid_generator/result", an image for debugging purposes, showing the superimposed image, with aruco markers, grid cells and occupied grid.

All of the poses are expressed in the camera frame.

### 3.2.3. Pointcloud registration

To infer a 6D grip pose, we need 3D information. This information is provided in the form of a pointcloud. To generate our pointcloud, we use the depth map provided by the camera package and the camera intrinsic parameters matrix, $K$, using this simple projection operation:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = z * K^{-1} * \begin{bmatrix} u \\ v \\ 1 \\ \frac{1}{z} \end{bmatrix} \tag{3.1}$$

However, the raw pointcloud generated is very dense for our application. To filter and downsample it, we use the method provided by the Open3D library.

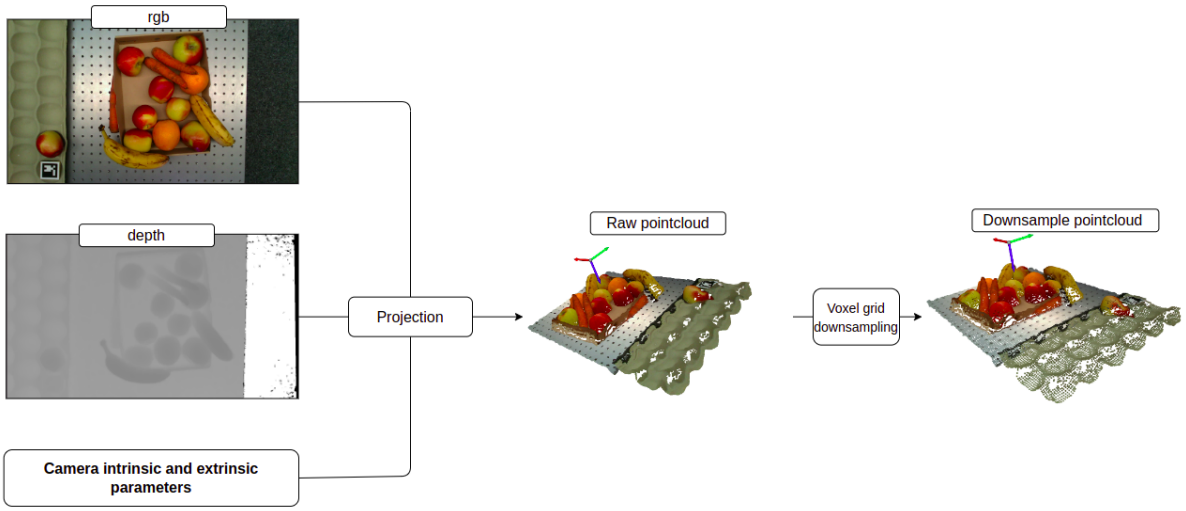
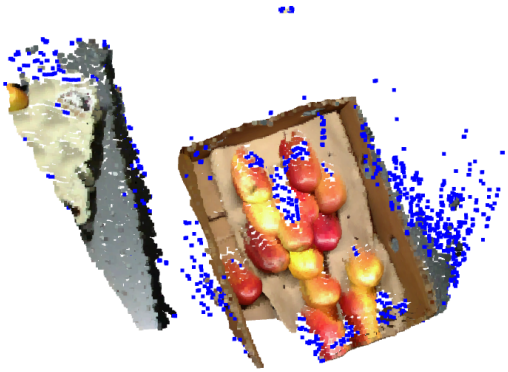



**Figure 3.4:** Pointcloud registration and downsampling

We use voxel downsampling with a resolution of 5mm and then remove outliers. Two approaches were considered to remove them. The "radius" method, which removes isolated points that have less than *neighbour_threshold* in a fixed sphere of radius $r$. The "statistical" method, where points that are further away from their neighbours by a distance *min_distance* compared to the average for the point cloud are removed. In our case, the "radius" method was the one that performed better empirically, so we used it for the rest of the project.

Pointcloud downsampled with a voxel size of 0.005

Outlier filter by radius, the outlier are represented in pink



Outlier filter by statistical method, the outlier are represented in blue

**Figure 3.5:** Visualisation of the differents pointcloud filtering methods

## 3.3. Grasping pose estimation

For the grasping strategy, we use Contact-Graspnet as a standard method and make some modifications in two points:

- Change the visualisation library to Open3D.
- Adde a collision detection method between the gripper and the detected point cloud.
- Change the lost function to select the gripper.

### Collision detection

To use the method for objects in dense unstructured clutters, we are bound to generate grasps with collisions between the environment and the gripper. Some of these grasps can be catastrophic, for example if the gripper hand collides with the table or some other fixed rigid object. While others may be necessary like the collision between finger and the environment around the target object. To detect collisions quickly, we approximate

the gripper as a sum of rectangles. The point cloud around the region of interest is then projected around the virtual gripper. The method calculates the number of collisions as the number of points from the pointcloud inside each rectangle. Three values are then returned:

- bodyCollision, The number of collisions between the hand, the camera and the environment.
- fingerCollision, The number of collisions between the finger and the environment.
- collateralGrasp, The number of points that are not in the ROI but are inside the gripper.

**Grasp selection**
Several grasp candidates are returned, sorted by the returned model score, *graspScore*. To avoid generating incorrect grips, the selection process was modified.

The first part was to remove the grasp from under the table. As the method had no knowledge of the correct orientation of the pointcloud, grasps coming from under the scene were sometime produced. All grasps with the following conditions were removed:

$$\begin{aligned} ee\_x\_angle &\notin [-\frac{pi}{2}, \frac{pi}{2}] \\ ee\_y\_angle &\notin [-\frac{pi}{2}, \frac{pi}{2}] \end{aligned} \tag{3.2}$$

.

All grasping poses with *bodyCollision* were also removed to avoid any risk of damage to the hand or the camera.

Then the score was modified as follows:

$$\begin{aligned} collisionScore &= log(1 + fingerCollision) \\ collateralScore &= log(1 + collateralCollision) \\ score &= graspScore * \frac{1}{1 + a\ collisionScore + b\ collateralScore} \end{aligned} \tag{3.3}$$

where $a = 0.25$ and $b = 5$ are the default weights for $collisionScore$ and $collarteralScore$, respectively.

**Figure 3.6:** Wrong grasp filtered.

The service associated takes as input the pointcloud of the scene (Pointcloud2) and a BGR image from the camera point of view. It return as output the position, orientation and opening of the grasp, alongside with two flag, one for successful detection and one to indicate if the detected grasp has collision between the fingers and the environment.

The returned grasp is in the camera frame and represent $\mathbf{T}_{grasp \longrightarrow camera\_optical\_frame}$.

## 3.4. Movement

Optimal control problems involve minimising a cost function $c = \sum_{t=1}^{T} c(x_t, u_t)$ considering a dynamical system $x_{t+1} = d(x_t, u_t)$ describing how the state and control command evolve over a time window $T$. Iterative LQR (iLQR) approaches a non-linear model by linearising the dynamical system using Taylor expansion around each step. The problem is solved by iteratively improving the control commands by solving an LQR at each iteration, leading us to a solution. Since it is a solved problem and the solution has already been implemented, in this project we will use the implementation of Jérémy MACEIRAS described in the document of Sylvain Calinon [4]. We used this controller to generate our trajectory and send the speed command to the robot velocity controller.

### 3.4.1. Different trajectories

Four different types of trajectory have been created to go directly to the viewpoint, grab, dispose and return. Although they have a lot in common, we'll mention the keypoints of each one.

- *viewpoints trajectory*, In this case we have only one keypoint to reach, which is the target position with good orientation precision.
- *grasping trajectory*, Two keypoints are set to grasp. *keypoints*1 is the pose set so that the orientation is aligned with the grasping orientation and the position is

offset along the z-axis of the gripper. The position, *pre_grasp*, can be calculated as

$$pre\_grasp = grasp\_rot \begin{bmatrix} 0 \\ 0 \\ z\_offset \end{bmatrix} \tag{3.4}$$

The last keypoint to reach is right at the grasping pose.

- *disposal trajectory*, The disposal is a three keypoint process, the first one is the same as *pre_grasp*, then the robot moves to a *pre_dispose* pose calculated similarly to *pre_grasp*, it is calculated as :

$$pre\_dispose = dispose\_rot \begin{bmatrix} 0 \\ 0 \\ z\_offset \end{bmatrix} \tag{3.5}$$

Finally, the last key point is at the disposal position.

- *return trajectory*, it is a trajectory with two steps. The first is at the *pre_dispose* and the last at the first viewpoint pose.

## 3.5. Next best view selection

We define $x$ a view from a set of potential views $X \in SE(3)$. If the grasp detection fails, we move the end-effector to a new position to add data points to our pointcloud. If no object of interest was detected in the camera frame, we simply move to the next viewpoint $x$ in the list.

Otherwise, a common approach is to select the next best view according to some measure of Information Gain (IG). To select our best view, we were inspired by the work of Isler and Al. [7] and their backside voxel information gain, which represents the object's "shadow", i.e. the area hidden by the visible part of the object.

We first reproject the point cloud on black images for each viewpoints and compute the gain $G_x$ as

$$G_x = \frac{\sum_{ij} I(i,j)}{w * h}. \tag{3.6}$$

where i and j are the pixel index, $w$ and $h$ are the image width and height, and $I(i,j) = 1$ if the pixel has no projected points and is in the target bounding box, and $I(i,j) = 0$ if the pixel has one projected point. $G_x$ can be interpreted as the percentage of pixels without a projected point. The result of this approach is similar to the one-way raycasting proposed by Breyer [2].

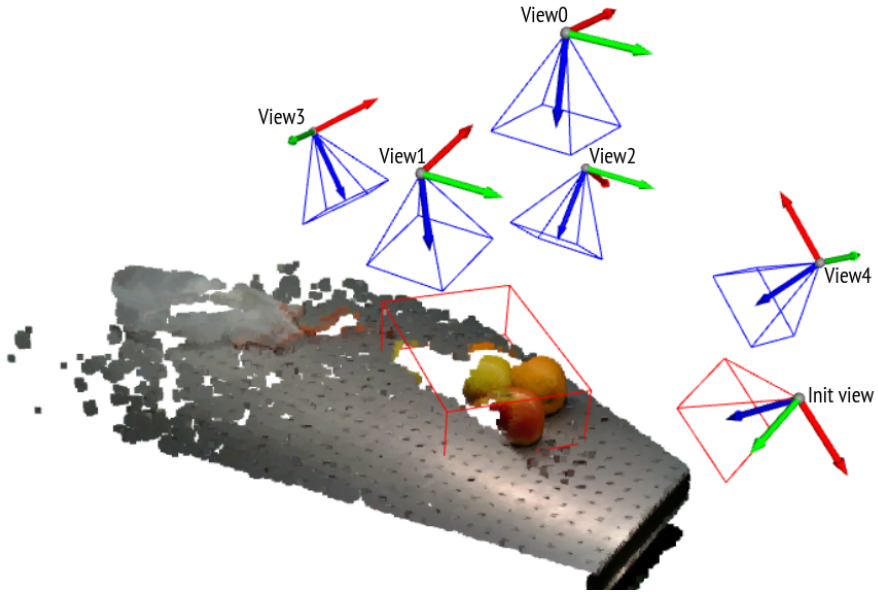
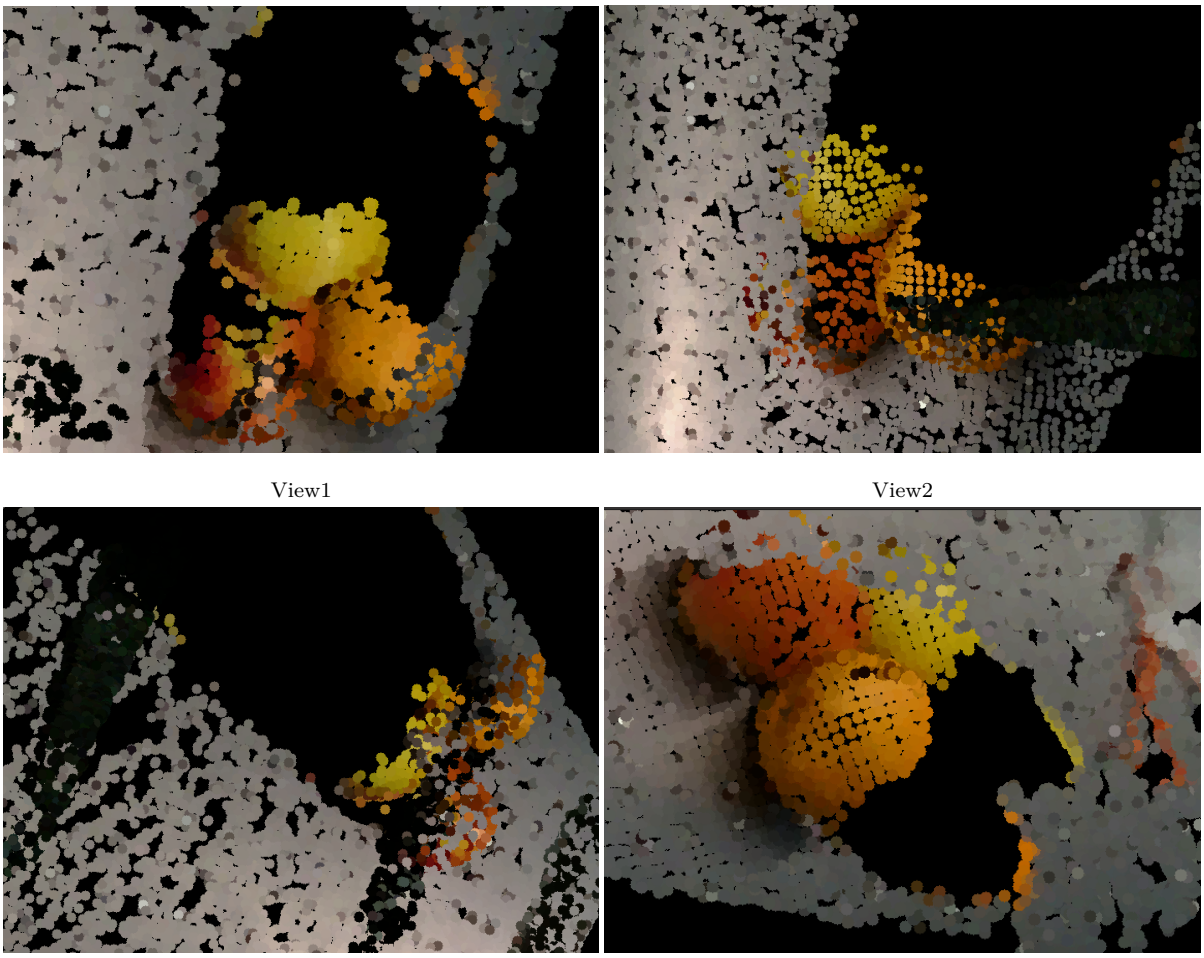



**Figure 3.7:** View and pointcloud display. The different artificial viewpoint are shown as blue triangles, and the original view used to generate the point cloud is shown as a red triangle. The region of interest is shown as a red rectangle.



View1 View2

View3 View4

**Figure 3.8:** Visual representation of the reprojected pointcloud over multiple viewpoints.

In the example of figure 3.8, the best view is the second one.

## 3.6. Simulation

Before experimenting on the real robot, the method and algorithm were tested on Robosuite [15], a simulation framework based on the MuJoCo physics engine for robot learning. Robosuite was chosen for the following reason:

- Efficient physics simulation: Mujoco is well known for its efficient physics simulation capabilities. It can handle complex multi-body systems and contact interactions with high fidelity and stability, providing accurate and realistic dynamics.
- Customisable environments: Many functions are already implemented by the framework, making it easy to create new environments.
- Realistic visual rendering: Although Mujoco's main attraction is it's physics engine, it also provides a good rendering engine that allows the use of segmentation method directly on the simulation image.

To better fit the fruit picking, we needed to set up a new environment that contained fruit in bulk. The environment was set for a fixed selection of fruit, [apple, banana, lemon, peach], any available obj file can be converted to mujoco by converting them to mesh file. However, complicated meshes require a lot of computing power, which limits the number of objects that can be rendered in a given time.

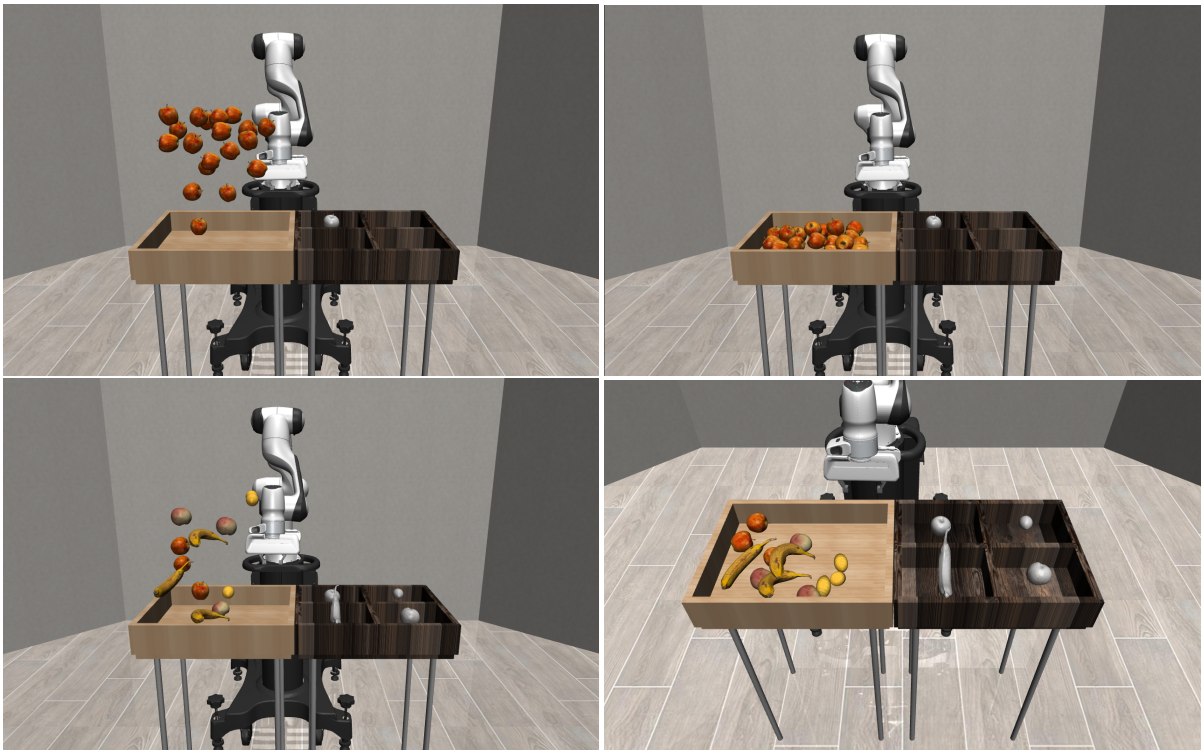


**Figure 3.9:** Robosuite adapted environment for batch of fruit.

With this environment rendered, the experiment could be performed by retrieving the image and pointcloud from the end-effector's viewpoint. This environment was used as a mock-up stage before trying the experiments on the real robot.

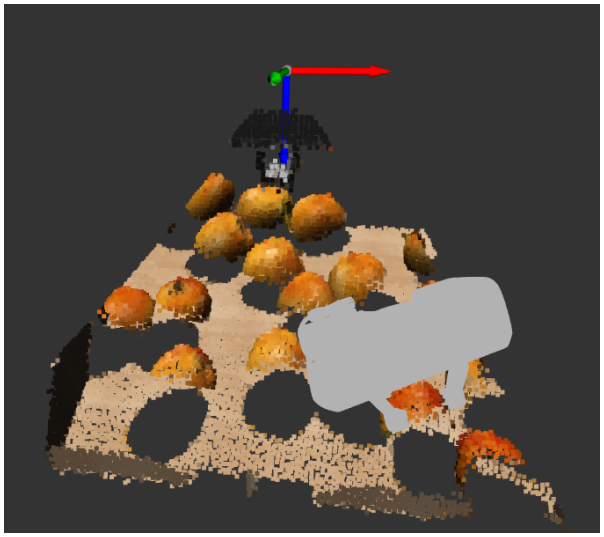


**Figure 3.10:** Grasp predicted in Robosuite environment using contact-graspnet

## 3.7. State machine and implementation

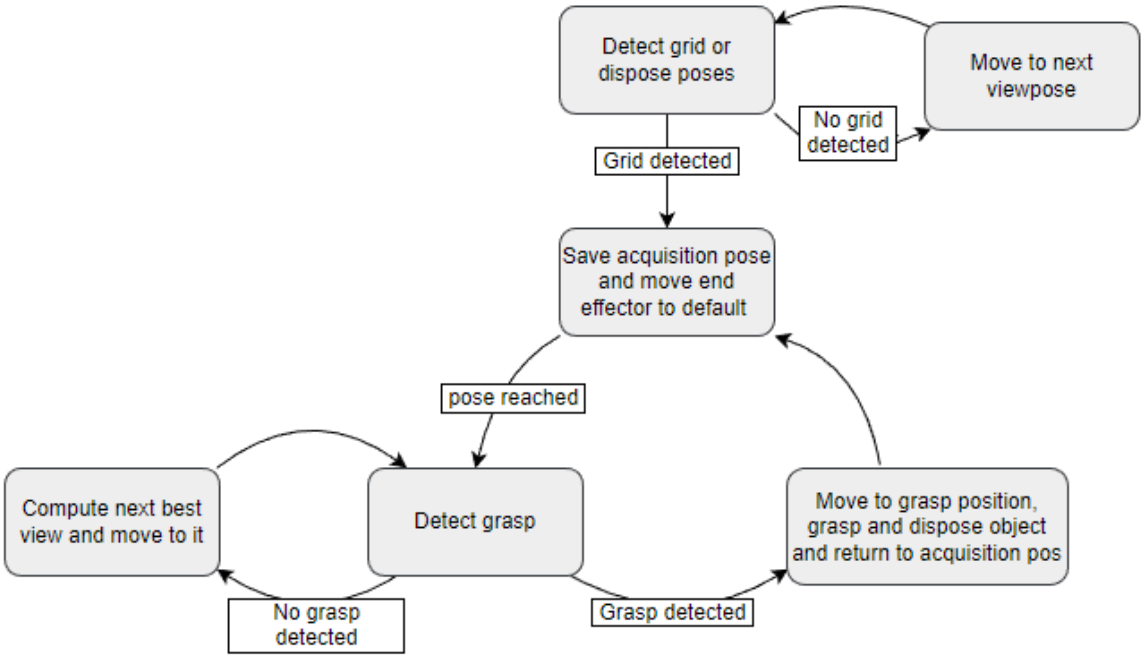



**Figure 3.11:** State machine of the grasping strategy

Architecture of the code of the main script. The loop is repeated indefinitely until the grid is full, or until a certain number of fruits have been disposed of.

# 4

# Experimental setup

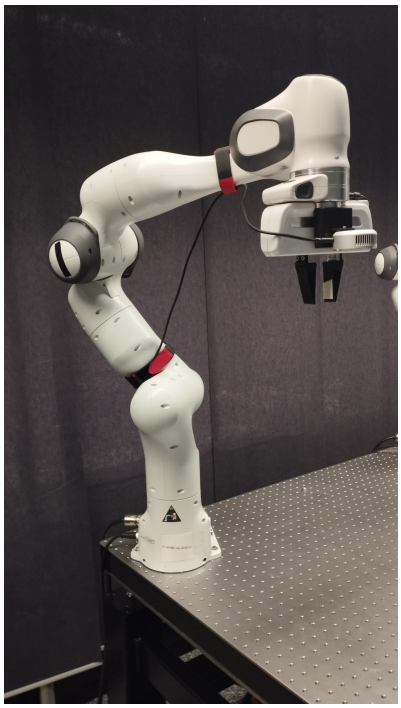


**Figure 4.1:** Robot used for the experiment

## 4.1. Computation

To control the robot, command input need to be executed within specific time constraints. To ensure that, real-time kernel are used to guarantee timely execution of tasks. On the other hand, we need CUDA to parallelised the computation of the packages relying on neural networks i.e YOLOv8-seg and contact-graspnet. When a real-time kernel and CUDA are combined on the same system, conflicts may arise. Real-time kernel typically have their own scheduling algorithm and may override or modify the scheduling decisions made by the CUDA methods. This can lead to unpredictable behavior and cancel the timing guaranty of a real-time system.

To avoid this issues, two computer were used. One laptop equipped with CUDA and a GeForce GTX 1050 Mobile with 2GB of memory and one desktop computer operating

on a real-time kernel to control the robot. The two machine communicate via ROS message on a direct ethernet connexion, with the ROS master being the desktop.

## 4.2. Robot

The robot is a 7-axis collaborative robot from Franka Emika [1]. Another small collaborative desktop robot from the company UFactory[2] is available in the laboratory, but the maximum gripper opening of 5 centimetre is too small for most fruits. A vacuum gripper was also available, but its gripping force is too weak to grip apples or other fruits.

## 4.3. Camera

Two intel depth cameras were investigated for the project, a D415[3] and a L515[4]. We chose with the L515 because the minimum optimal operating depth is 25cm and the camera could go even closer without returning major errors. On the other hand, the return depth/point cloud of the D415 tends to return large errors at distances below 40cm.

## 4.4. Gripper

The gripper used is the panda hand [5].

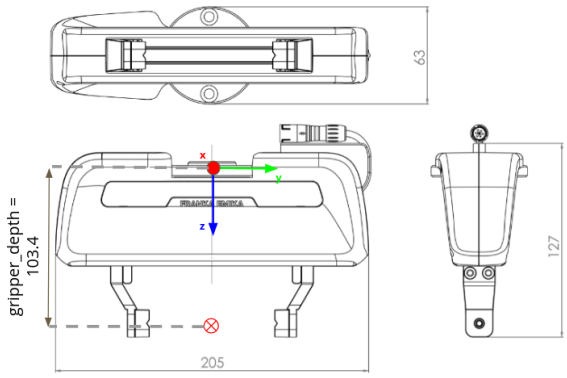



**Figure 4.2:** Panda hand outer dimension

The standard finger has a tendency to damage fruit, either by gripping too hard or by collision when reaching. To mitigate these problems, the finger has been replaced with a soft gripper,

[1] https://www.generationrobots.com/media/franka-emika-research-3-robot-datasheet.pdf
[2] https://www.ufactory.cc/lite-6-collaborative-robot/
[3] https://www.intelrealsense.com/depth-camera-d415/
[4] https://www.intelrealsense.com/lidar-camera-l515/
[5] https://download.franka.de/documents

Rigid Finger | Soft Finger | Soft Finger during grasp

**Figure 4.3:** Visualisation of the different gripper

This replacement changed the *hand_depth* and $\mathbf{T}_{see \longrightarrow flange}$. The change in weight of the end effector is considered negligible.

## 4.5. Packaging

The robot places the round fruit in an honeycomb package, which is marked with two 4x4cm aruco markers on each side. As we have fifteen element and two occupied cells by the markers, the markers were place to form a 5x4 grid.

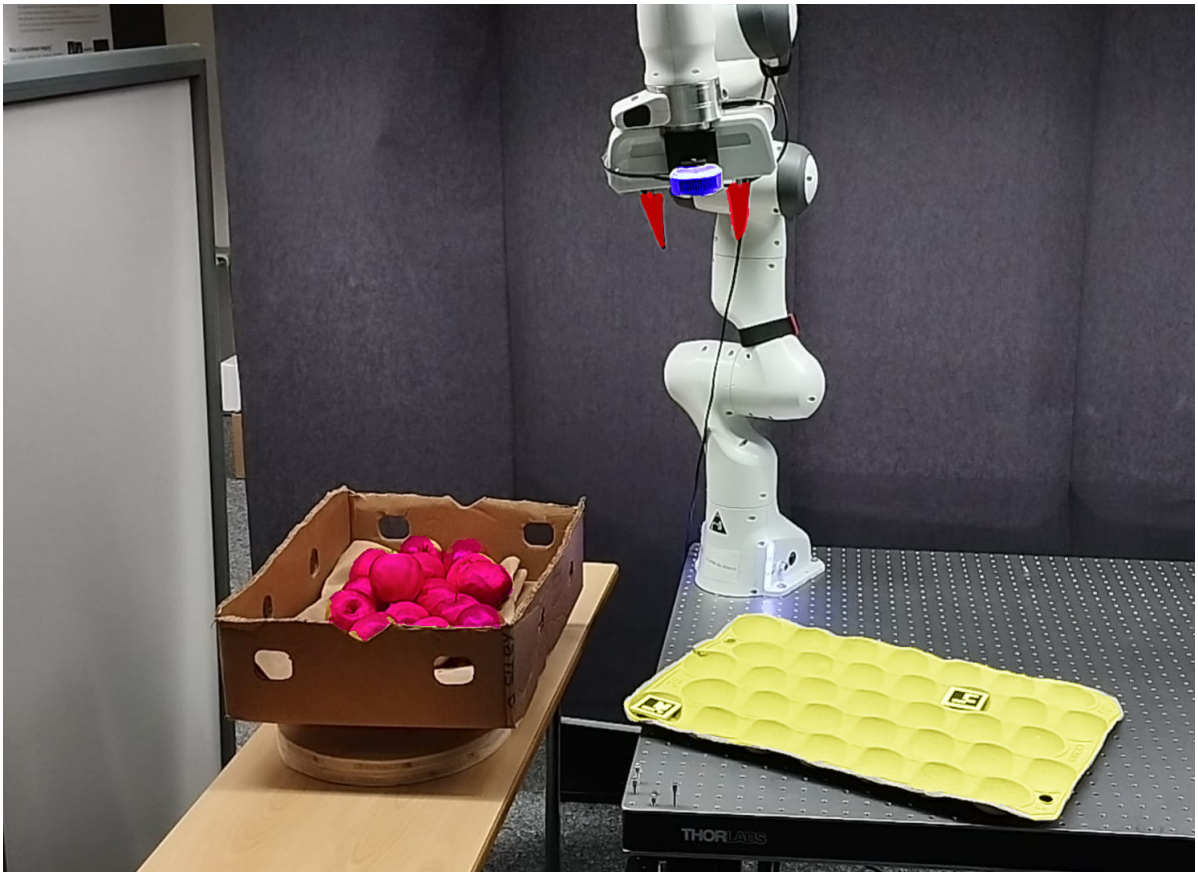



**Figure 4.4:** Overview of experiment setup. The camera is highlighted is blue, the grasping target in purple, the package in yellow and the gripper fingers in red

## 4.6. Experiment parameters

The different parameters experiment for the are

| **Parameter** | **Value** |
|---|---|
| intitial view | top view |
| minimum IG | 0.05 |
| YOLO model | X |
| IoU threshold | 0.4 |
| segmentation threshold | 0.05 |
| downsampling voxel size | 0.005 |
| Outlier regularization type | radius |
| Number of neighboors | 25 |
| radius | 0.015 |
| grasp horizon | [40, 65] |
| dispose horizon | [45, 100, 200] |

# 5

# Result

To evaluate the method, the following metrics were used:

- Number of trial.
- Clearance rate, it is the percentage of objects successfully removed from the scene and place in the package.
- Grasp Feasibility, it is the percentage of predicted grasp solvable by the iLQR planner. A grasp is considered solvable if the difference in position between the target grasp pos and the ee final pos is inferior to a threshold of $0.001meter$.
- Percentage of successful grasp, $succes\_rate$ a grasp is considered successfully if the target object is lift and stay in the gripper for the full $dispose\_traj$, only solvable grasp are considered.

| | Number of trials | Success rate | Clearance | Grasp Feasability |
|---|---|---|---|---|
| Apple | 8 scene of 15 element | 81.7% | 92% | 93% |
| Orange | 3 scene of 14 element | 77% | 85% | 92% |
| Banana | 3 scene of 4 element | 27% | 77% | 90% |
| Apple without modified loss | 3 scene of 15 element | 79% | 93% | 91% |

**Table 5.1:** Experimental result on three different fruits

## 5.1. Result discussion

We observe similar result to contact-graspnet [10] for spherical object, with lower performances for oranges due to a higher tendency to roll when grasping. However, due to the different gripper finger and setup configuration, the comparison to contact-graspnet can only be used as a side note. The method has tendency to grasp banana by the stem, this lead to a higher torque at the contact point between the gripper finger and banana. We hypothesize that this property combined with the soft finger lead to lower grasp success.

We observe in table 5.1 that the clearance rate is never 100% for any object. For spherical object, this situation arrised as the pipeline had difficulty to to predict grasps for element aggregated close to the border as illustrated in Figure 5.1

**Figure 5.1:** Apple on the crate border

For the bananas, the lower clearance grasp is also due to the repeated grasp failure.

The modified loss did not have a significant effect on performances. The only qualitative observation was that grasp selection without modified loss had tendency to predict slightly more grasp with finger colliding with the target object.

As the initial experiment viewpoint is right over the crate and target objects are convex, other viewpoints did not provide an information gain $IG$ superior to its threshold of 5%. This lead the method to switch between viewpoint in a predefined order, however new view did not improves the pointcloud completeness by a significant margin. Packaging from fruit in bulk may not be the appropriate task to use active perception.

# 6

# Conclusion and future work

## 6.1. Discussion

We can see from the results section that the pipeline is prone to errors, but they are not catastrophic to either the robot or the fruit. This shows that the method is suitable for the fruit bin-picking task. However, we observed several times that contact-graspnet had difficulties in producing a correct grasp in the edge case. We estimate that our application is not represented in the ACRONYM dataset. This was compensated for in the project by using a collision detection method to filter out false grips.

The motivation for this project is focused on small scale retail, where it is important to consider the operating costs. For this project, the main hardware costs are divided between the RGBD camera and the robotic arm. Over the year we have seen a decrease in the price of cobots, as of the submission of this thesis you can find a cobot for around 3000$[1]. If we can combine this with a more robust version of this work, it could provide a small relief for this tedious task with a short labour shortage.

## 6.2. Possible future work

Contact-graspnet is used as an off-the-shelf method in this work. However, the training dataset had some differences with our final application. Their training scene consisted of isolated objects or structured light clutter, whereas our application consisted of dense unstructured clutter in a box. We hypothesise that retraining the contact grasp network on a dataset with a similar scene to our application would improve the performance of the method and reduce the need for an explicit collision method. The REGRAD dataset proposes a dense scene of this type, so it would be a good candidate. The soft finger for the gripper could also be included in the training, but a deformable object may add complexity to the simulation and limit the reproducibility of the project.

In our pipeline, the trajectories are optimised to go to a selected end-effector pose, the trajectory keypoints are set to avoid collision with the object, box or table. A more advanced method could be used to optimise the trajectory while avoiding the obstacle. To use this robot in a collaborative environment, impedance control would be a more appropriate choice than velocity control.

[1] https://www.ufactory.cc/lite-6-collaborative-robot

Finally, grading in retail is a common thing, it consists of presenting the best side of the product to the customer, for example showing the more red and homogeneous part side of an apple in the package. A follow-up to this project could be to add grading and product filtering to the current pipeline.

## 6.3. Conclusion

The aim of this project was to explore the use of active perception for fruit packaging. This simple task for humans represents a challenge for the robot as it has to combine several subtasks such as camera calibration, point cloud registration, image segmentation, grasp synthesis, package detection, optimal trajectory and robot control. To ensure the possibility of subsequent work, this project emphasised modularity.

After reviewing the literature related to parallel finger manipulation, we saw a predominance of data-driven methods for grasp generation. This approach worked for our pipeline and allowed us to achieve our goal.

Overall, this work represents a first step towards the use of low-cost cobots for retail applications.

# References


[1] Daniel Bolya et al. "YOLACT: Real-Time Instance Segmentation". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 9156–9165. DOI: 10.1109/ICCV.2019.00925.

[2] Michel Breyer et al. *Closed-Loop Next-Best-View Planning for Target-Driven Grasping*. 2022. arXiv: 2207.10543 [cs.RO].

[3] Michel Breyer et al. "Volumetric Grasping Network: Real-time 6 DOF Grasp Detection in Clutter". In: *Conference on Robot Learning*. 2020.

[4] Sylvain Calinon. "A practical guide to learning and control problems in robotics solved with second-order optimization". In: *https://rcfs.ch/doc/rcfs.pdf*. 2023.

[5] Amaury Depierre, Emmanuel Dellandréa, and Liming Chen. "Optimizing Correlated Graspability Score and Grasp Regression for Better Grasp Prediction". In: *CoRR* abs/2002.00872 (2020). arXiv: 2002.00872. URL: https://arxiv.org/abs/2002.00872.

[6] Clemens Eppner, Arsalan Mousavian, and Dieter Fox. "ACRONYM: A Large-Scale Grasp Dataset Based on Simulation". In: *2021 IEEE Int. Conf. on Robotics and Automation, ICRA*. 2020.

[7] Stefan Isler et al. "An information gain formulation for active volumetric 3D reconstruction". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 3477–3484. DOI: 10.1109/ICRA.2016.7487527.

[8] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. *YOLO by Ultralytics*. Version 8.0.0. Jan. 2023. URL: https://github.com/ultralytics/ultralytics.

[9] Charles Ruizhongtai Qi et al. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/d8bf84be3800d12f74d8b05e9b89836f-Paper.pdf.

[10] Martin Sundermeyer et al. "Contact-GraspNet: Efficient 6-DoF Grasp Generation in Cluttered Scenes". In: (2021).

[11] Julen Urain et al. "SE (3)-DiffusionFields: Learning cost functions for joint grasp and motion optimization through diffusion". In: *arXiv preprint arXiv:2209.03855* (2022).

[12] Mohit Vohra, Ravi Prakash, and Laxmidhar Behera. "Real-time Grasp Pose Estimation for Novel Objects in Densely Cluttered Environment". In: *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. 2019, pp. 1–6. DOI: 10.1109/RO-MAN46459.2019.8956438.

[13] Thomas Weng et al. *Neural Grasp Distance Fields for Robot Manipulation*. 2023. arXiv: 2211.02647 [cs.RO].

[14] Xu Zhao et al. *Fast Segment Anything*. 2023. arXiv: 2306.12156 [cs.CV].

[15] Yuke Zhu et al. "robosuite: A Modular Simulation Framework and Benchmark for Robot Learning". In: *arXiv preprint arXiv:2009.12293*. 2020.

# A

# Appendix

## A.1. Graphical interface

To easily control the different parameters of the method, two graphical interfaces were set leveraging rqt_reconfigure.


**Figure A.1:** Control GUI

## A.2. Visualization

Since a lot of communication is done via ROS, RVIZ provides a valuable tool for visualisation of the different, image, pointcloud, array, transformation and the different poses.

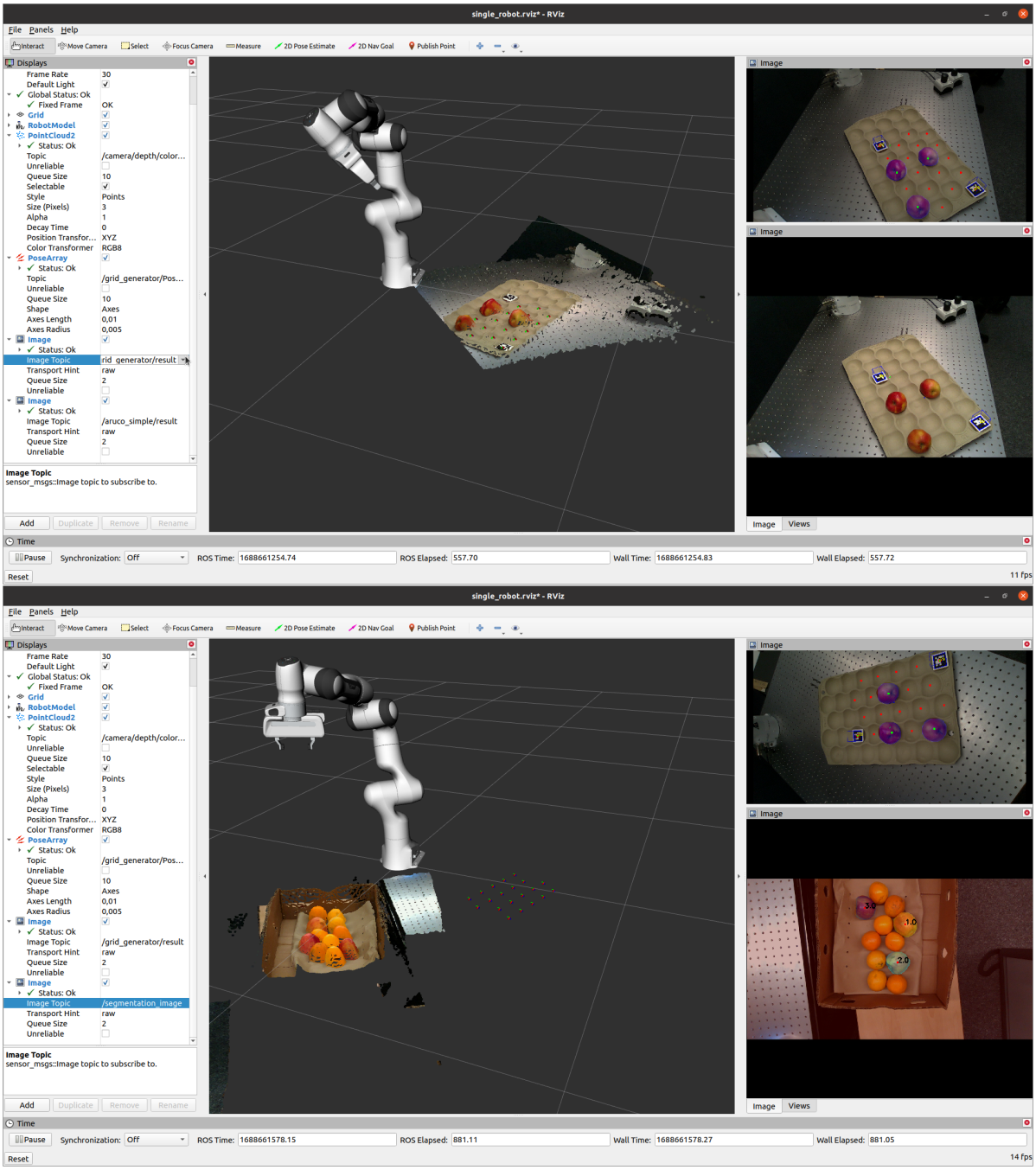



**Figure A.2:** RVIZ views, with the detected grid on the top right image, 3D view of the environment on the left with the camera pointcloud, robot simulated pos and grid pose.

Also, if the ”show_grasp” parameter is known, the grasp generation method will show a preview of the returned grasp on it’s laptop as shown in Figure A.3

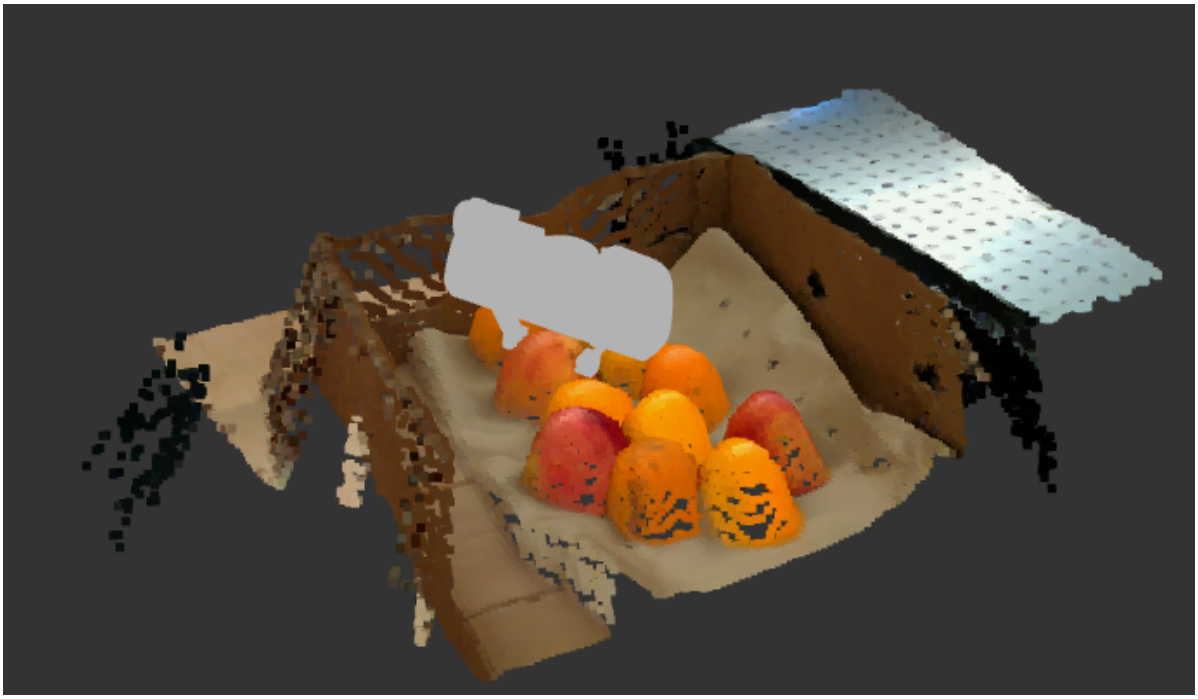


**Figure A.3:** Grasp visualization returned by the grasp generation method