Niryo One as a Vision-Based Robotic Grasping Systems

Michele Oliva

1 Grasp detection and execution pipeline

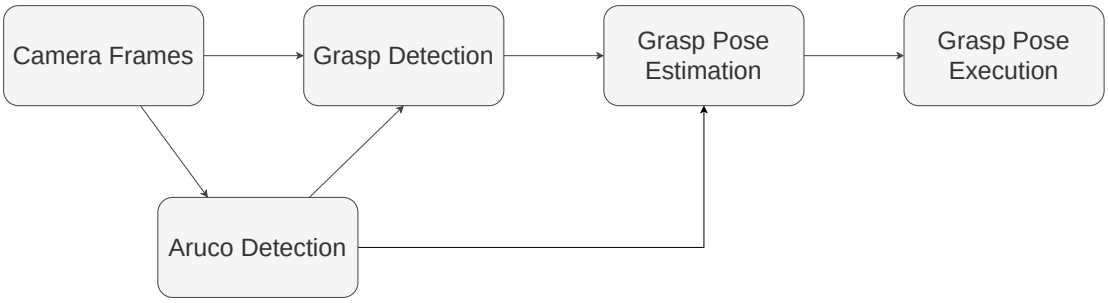

Figure 1: The designed grasp detection and execution pipeline.

Generally speaking, the designed grasp pose detection and execution pipeline consists of five stages, as shown in fig. 1.

Looking at the pipeline adopting a bottom up approach, the steps are the following:

1. the camera (an) acquires RGB and Depth images, so the depth image is aligned to the RGB image;

2. the ArUco detector, starting from the RGB image previously acquired, evaluates the workspace center coordinates in the image plane and calibrate the camera getting its extrinsic parameters;

3. the grasp detector, starting from the RGB and aligned Depth images and the knowledge of the workspace center coordinates in the image plane, crops the workspace from the images, preprocess the obtained images to match the input format of the GR-ConvNet and post-process its outputs to obtain the corresponding Grasp Rectangles;

4. the grasp pose estimator defines the transformations to convert the image coordinates to

robot's frame of reference:

$$G_r = T_{rc}(T_{ci}(G_i)) \tag{1}$$

where T_{ci} is a transformation that converts image space into camera's 3D space using the intrinsic parameters of the camera, and T_{rc} converts camera space into the robot space using the obtained camera extrinsic parameters, and so converts the obtained Grasp Rectangles to Robot's gripper poses;

5. finally, the grasp pose executor perform Inverse Kinematics to obtain the Robot's joints configuration that brings Robot's gripper in the desired pose, and impose that configuration to the manipulator.

To achieve all the steps described above, have been implemented five ROS nodes. These nodes communicate through ROS Topics, ROS services, or storing and retrieving parameters from the Parameter Server, the resulting graph is shown in fig. 2.

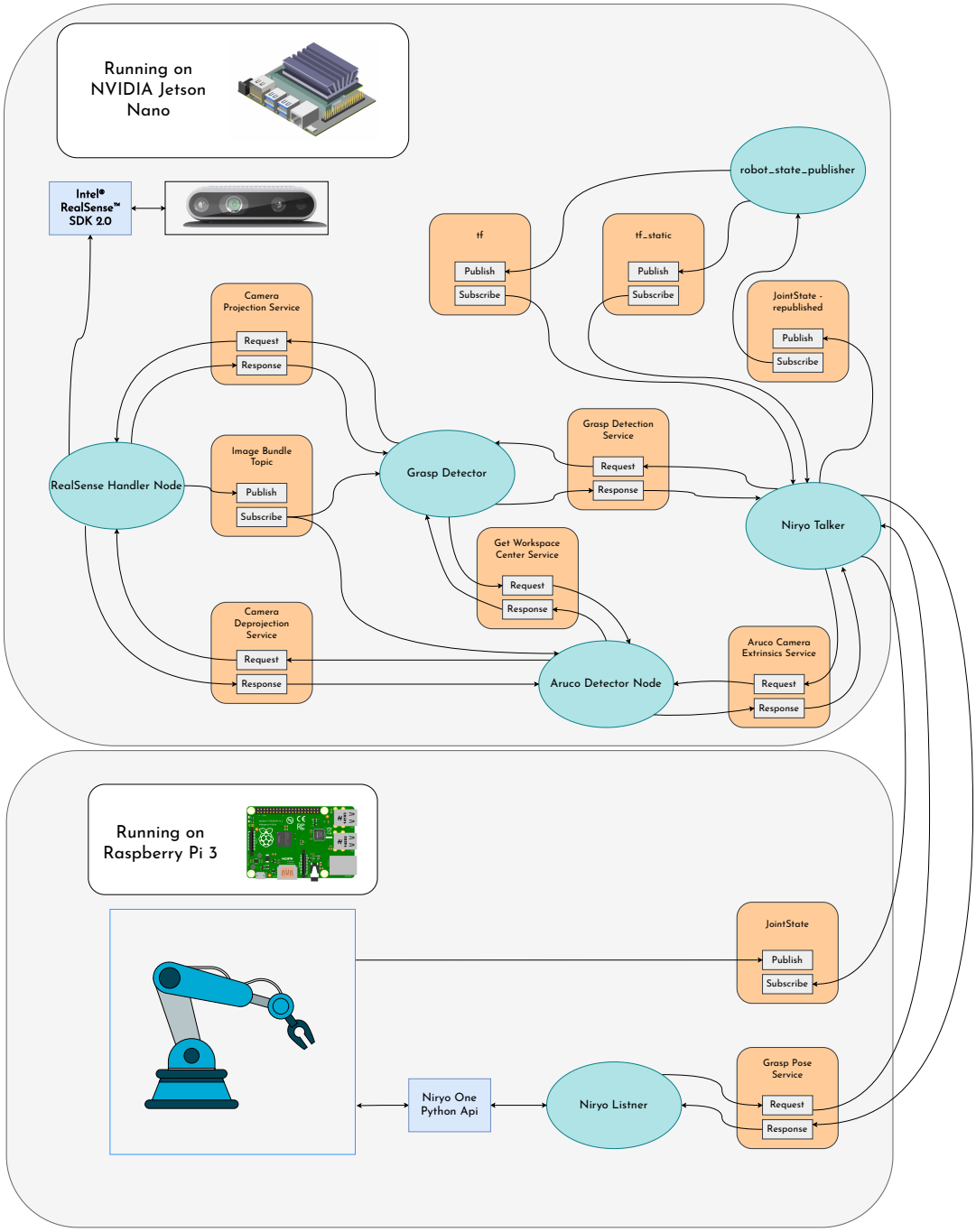

Figure 2: ROS Nodes Graph

1.1 RealSense Handler Node

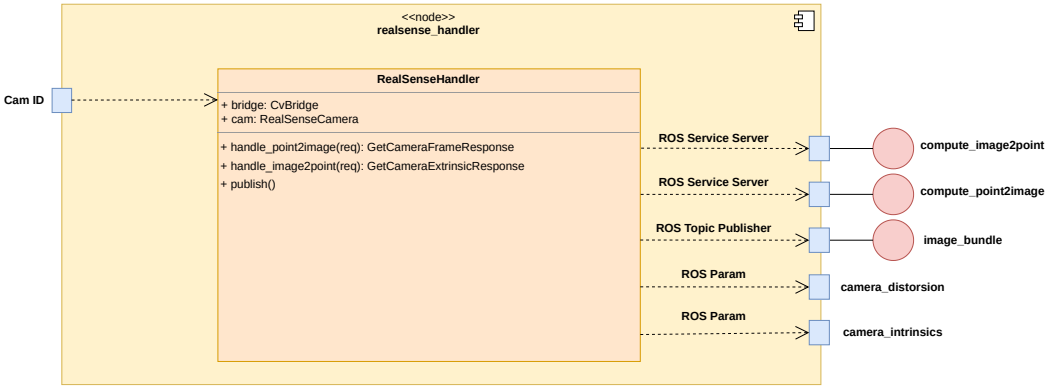

Figure 3: RealSenseHandler ROS Node class diagram

The RealSenseHandler node, whose class diagram is reported in fig. 3, is a custom RealSense ROS Wrapper, this node:

- store the `camera_distorsion` and `camera_intrinsics` on the Parameter Server;
- transmits over the topic `image_bundle`, the RGB and aligned depth images;
- exposes the `compute_image2point` service, that deprojects a 2D pixel location in the image as well as a depth, specified in meters, and maps it to a 3D point location;
- exposes the `compute_point2image` service that projects 3D point coordinates into a 2D pixel location.

Both exposed services are handled the Intel RealSense SDK 2.0 methods.

1.2 ArUco Detector Node

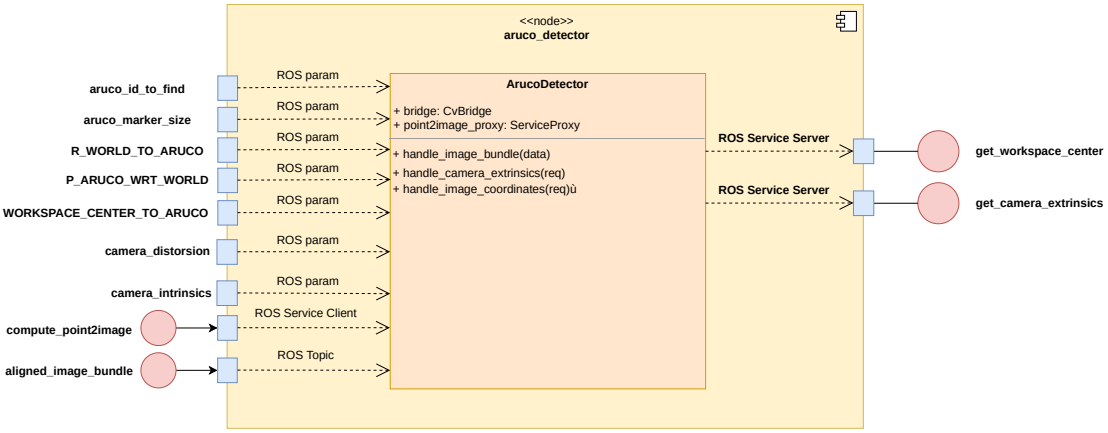

Figure 4: ArUcoDetector ROS Node class diagram

The ArUcoDetector node, whose class diagram is reported in fig. 4, exposes two services:

- the `get_workspace_center` service allows the user to get the 2D coordinates in the image of the center of the workspace;
- the `get_camera_extrinsics` service allows the user to get the pose of the camera w.r.t. the world frame.

When an ArUco [Gar+14] is recognized in the image by an OpenCV method, this method automatically returns the rotation matrix $R^{\text{camera}}_{\text{ArUco}}$, and the position of the ArUco w.r.t. the camera frame $p^{\text{camera}}_{\text{ArUco}}$.

The `get_workspace_center` service, retrieving from the Parameter Server the **known** position of the *workspace center* w.r.t. the ArUco frame $p^{\text{ArUco}}_{\text{ws}}$, simply computes the position of the workspace center w.r.t. the camera frame as

$$wc_c = p^{\text{camera}}_{\text{ArUco}} + R^{\text{camera}}_{\text{ArUco}} p^{\text{ArUco}}_{\text{ws}}$$

Once known that point, is possible to use the `compute_point2image` service to obtain its coordinates in the images.

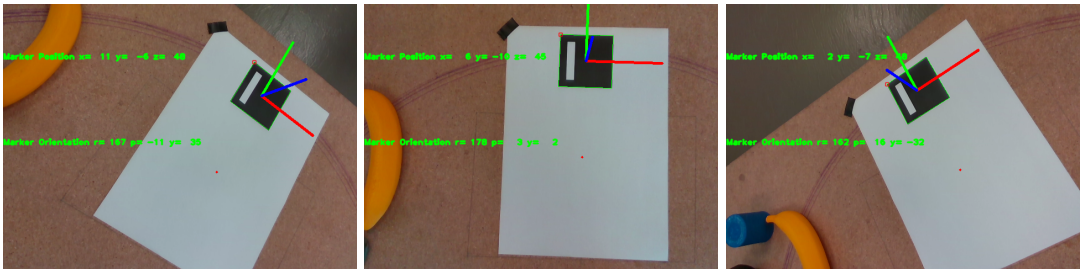

Figure 5: The workspace center (the red dot) in images captured by different point of view.

The `get_camera_extrinsics` service, retrieving from the Parameter Server the **known** $R^{\text{world}}_{\text{ArUco}}$ and $p^{\text{world}}_{\text{ArUco}}$ to obtain the does the following:

$$R^{\text{ArUco}}_{\text{camera}} = (R^{\text{camera}}_{\text{ArUco}})^T$$

$$p^{\text{ArUco}}_{\text{camera}} = -R^{\text{ArUco}}_{\text{camera}} p^{\text{camera}}_{\text{ArUco}}$$

$$p^{\text{world}}_{\text{camera}} = p^{\text{world}}_{\text{ArUco}} + R^{\text{world}}_{\text{ArUco}} p^{\text{ArUco}}_{\text{camera}}$$

$$R^{\text{world}}_{\text{camera}} = R^{\text{world}}_{\text{ArUco}} R^{\text{ArUco}}_{\text{camera}}$$

In this way has been obtained the need pose of the camera frame w.r.t. the world frame. In fig. 6 the used frames are showed.

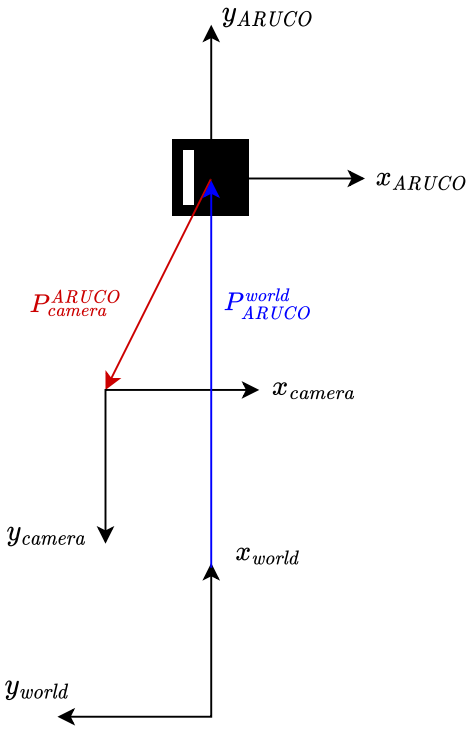

Figure 6: ArUco, camera and world frames.

1.3 Grasp Detector Node

The GraspDetector, whose class diagram is reported in fig. 7, node exposes the `compute_grasp_detection` service: the RGB-D image is cropped accordingly to the workspace center and is pre-processed to match the input format of the GR-ConvNet, the GraspMap, output of the network, is then post-processed to obtain the corresponding Grasp Rectangles.

For each Grasp Rectangles predicted, the grasp center point is deprojected to a point defined in the camera frame using the `compute_image2point`. But, to the `compute_grasp_detection` service user is not returned that point: may be that the predicted grasp center point is not on the highest point of the graspable objects, in this case, returning the deprojected point means that the manipulator will try to penetrate the workspace or collide with the object failing the grasp. To avoid that, the z value of the deprojected point is substituted with the lowest value found around the grasp center.

So the service returns that point and the additional rotation around the z-axis that the gripper must do to grasp the object.

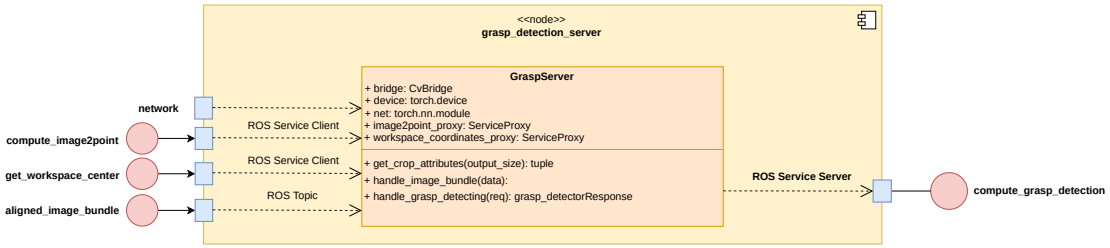

Figure 7: GraspDetectionServer ROS Node class diagram

1.4 NiryoManager Node

The NiryoManager Node, whose class diagram is reported in fig. 8, exposes the `compute_grasping_pose` service. From the knowledge of the camera pose w.r.t. the world frame, this node firstly transforms the information received through the `compute_grasp_detection` service in the grasp pose and then uses the grasp pose to define the "pre-grasp pose". The pre grasp pose is defined as the pose that the robot's gripper has to assume just before grasping the object, we have defined the grasp pose to have the same orientation as the grasp pose, but its position is five centimeters back along the z-axis of the grasp pose. Then perform Inverse Kinematics to obtain the robot's joint configurations that bring it to the two just defined poses.

Figure 8: NiryoTalker ROS Node class diagram

1.5 Niryo Listner Node

The Niryo Listner Node, whose class diagram is reported in fig. 9, is a demo script that implements all the procedures needed to perform the pick-and-place operation. It communicates

with the NiryoManager Node through the `compute_grasping_pose` service to request a new grasp pose, execute the received pose, performs a stability trajectory and places the grasped object.

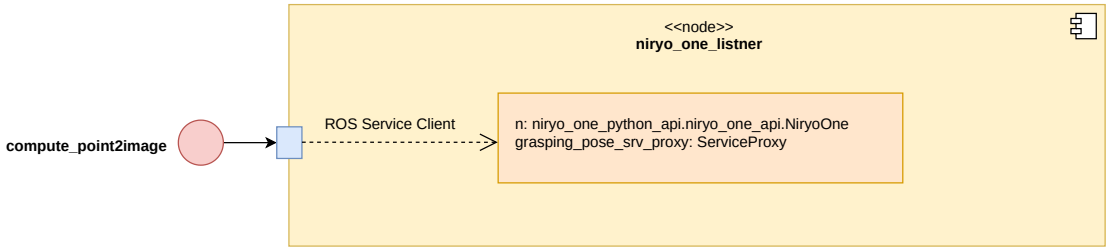

Figure 9: NiryoOneListner ROS Node class diagram

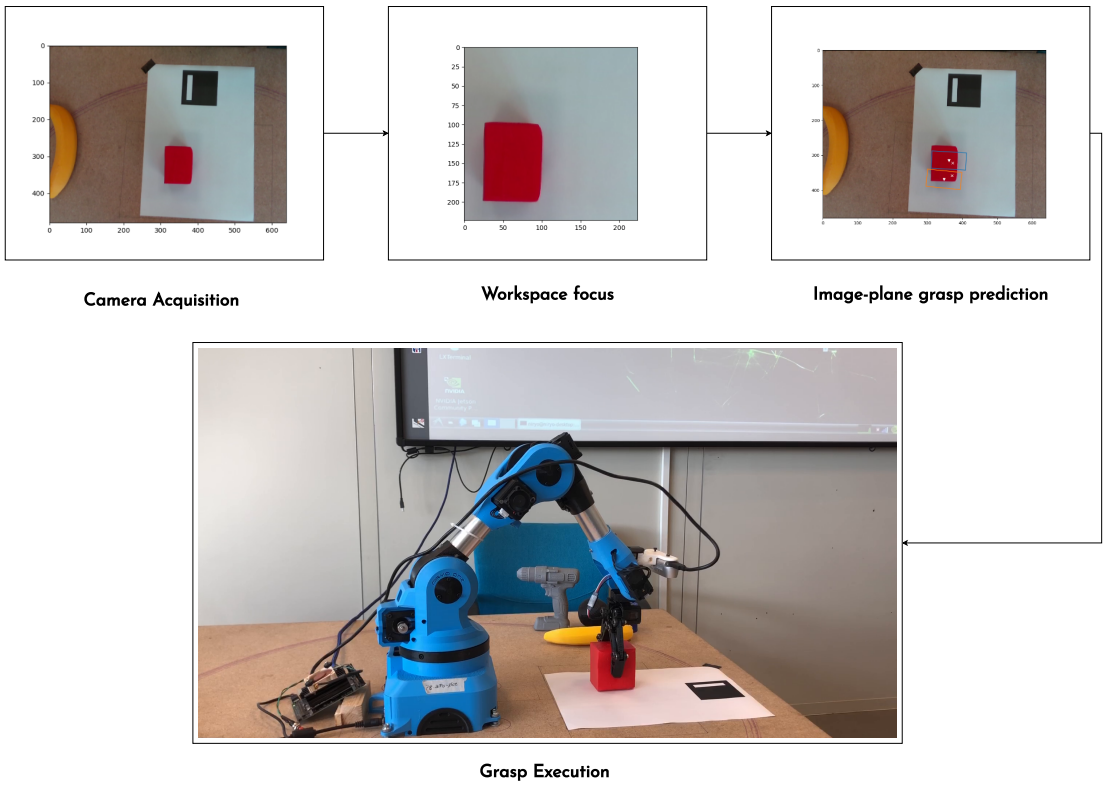

Figure 10: Grasp Detection and Execution

2 Experimental setting and procedure

We defined this Experimental Setting to perform repeatable and fair evaluation between different methods.

Definition 2.1 *The available experimental workspace is defined as a square of* 15×15 *cm, where the object under test can be placed.*

The size of the workspace has been chosen according to the GR-ConvNet input size.

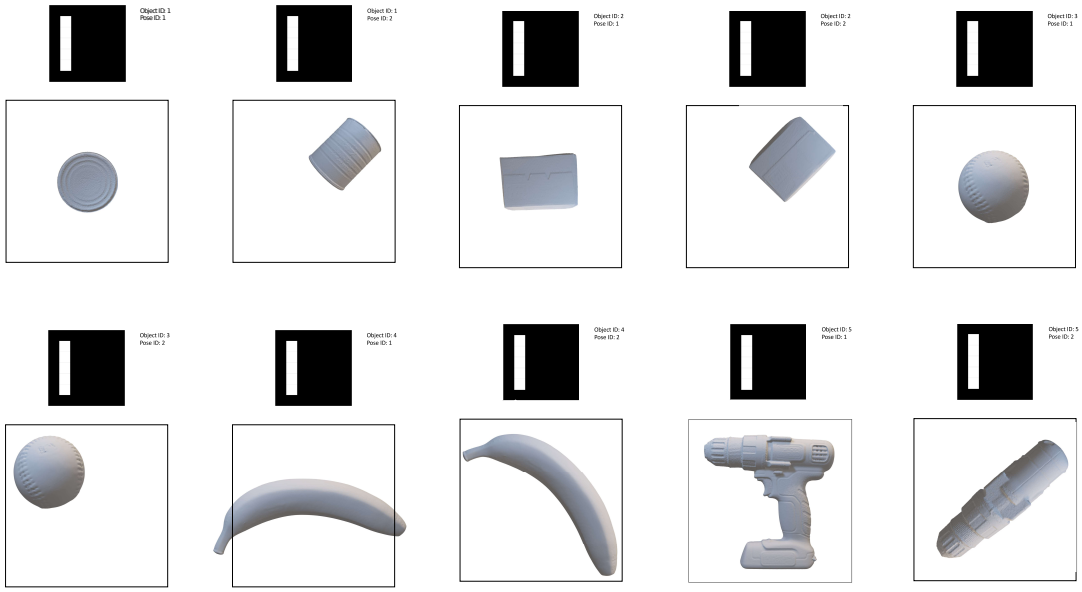

Figure 11: Object placed within the robot's workspace.

Definition 2.2 *A* ***scene*** *is defined as an object placed within the robot's workspace.*

Definition 2.3 *A* ***grasp detection pose*** *(**GDP**) is defined as a point-of-view from with the camera captures the scene.*

The defined grasp detection poses are shown in fig. 12.

Figure 13: Test Objects.

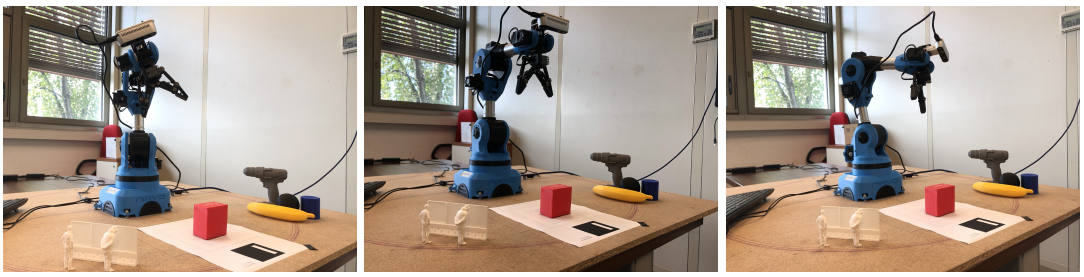

Figure 12: Grasp Detection Poses. (Left) Right GDP, (Center) Perpendicular GDP, (Right) Left GDP.

Objects chosen In the robot's workspace will be placed the five different objects, once a time. Three of five objects have been chosen in such a way to consider the basic geometry shapes: cylinder, cube, and sphere, and two daily objects: banana and drill. Each object will be placed within the workspace in **different poses**, in fig. 11 is shown the complete test set. These selected objects are a subset of YCB Benchmarks Model Set [Cal+15], as we can see in fig. 13.

All the objects has been 3D printed, the PLA (PolyLactic Acid) was used as printing material. For this reason, most of the objects are characterized by a smooth surface.

The Experimental Procedure for the i-th object and the j-th object is described in. The steps performed are the following:

- an object will be placed within the robot's workspace;
- for a given scene, and a given robot Grasping Detection Pose the robot will execute the grasp detection pipeline 7 times;
- for each GDP, the camera calibration is performed;
- for each trial, the grasp detection is performed;
- if there are detected grasps, the transformation from the image space to the corresponding Gripper Pose defined in world frame is performed;
- the first invertible gripper pose with the highest score is performed;
- the stability trajectory is performed.

The corresponding flowchart is represented in fig. 14.

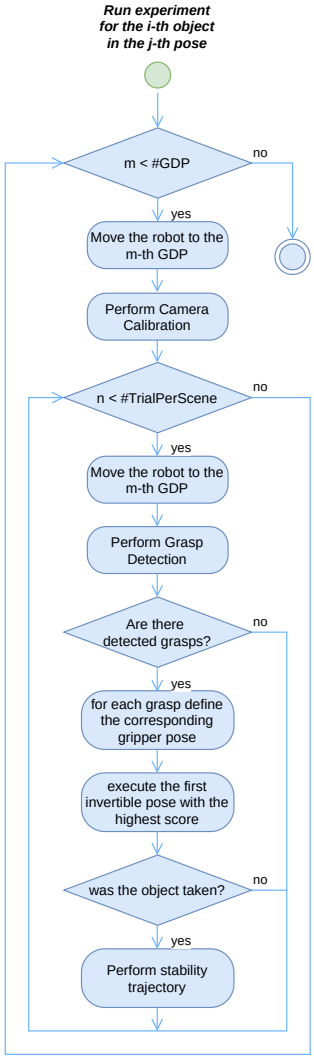

Figure 14: Experimental procedure flowchart.

Quality measures Two quality measures are proposed, the Grasp Success Rate and the Completion Rate.

Definition 2.4 *The Grasp Success Rate is defined as the ratio between the number of Correct Grasps over the number of trials executed. A grasp is defined as a successful one if the object does not slip from the gripper when the predicted grasp pose is executed, and it is a measure of the ability of the network to predict suitable grasps.*

$$\text{Grasp Success Rate} = \frac{\#\text{Successful Grasp}}{\#\text{Trials}}$$

Definition 2.5 *The Mean Grasp Success Rate is defined as the Grasp Success Rate averaged over all the objects and the poses for each object.*

$$\text{Mean Grasp Success Rate} = 100\frac{\sum_{i=1}^{\#\text{objects}}\sum_{j=1}^{\#\text{object poses}}\text{GraspSuccess}_j^i}{\#\text{object}\#\text{object poses}}$$

Definition 2.6 *The Completion Rate is defined as the ratio between the number of Completed Trajectories over the number of Successful Grasps. This metric aims to evaluate the stability of the executed grasp, that is the object does not slip from the gripper during the execution of a fixed trajectory.*

$$\text{Completion Rate} = \frac{\#\text{Completed Trajectories}}{\#\text{Successful Grasps}}$$

Definition 2.7 *The Mean Completion Rate is defined as the Completion Rate averaged over all the objects and the poses for each object.*

$$\text{Mean Completion Rate} = 100\frac{\sum_{i=1}^{\#\text{objects}}\sum_{j=1}^{\#\text{object poses}}\text{GraspCompletion}_j^i}{\#\text{object}\#\text{object poses}}$$

The stability trajectory moves the grasped object by rotating the robot's joint6 in the range $[-140°; 140°]$.

3 Experimental Results

3.1 The first GDP

For the Perpendicular GDP (definition 2.3 and fig:gdps), the network was able to predict suitable grasps for each object that had been correctly executed. However, not all the grasps have been completed, particularly the sphere and the cylinder, due to their smooth surface and the standing drill due to its size concerning the captured scene. In fig. 15 and fig. 16 are shown the partial results showing the statistics per object and pose; table 1 shows the aggregated results.

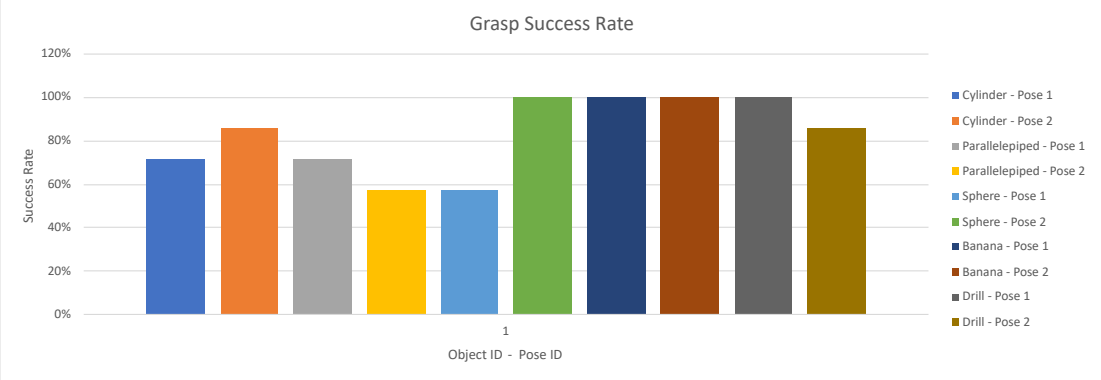

Figure 15: Grasp Success Rate per object per pose for the first GDP.

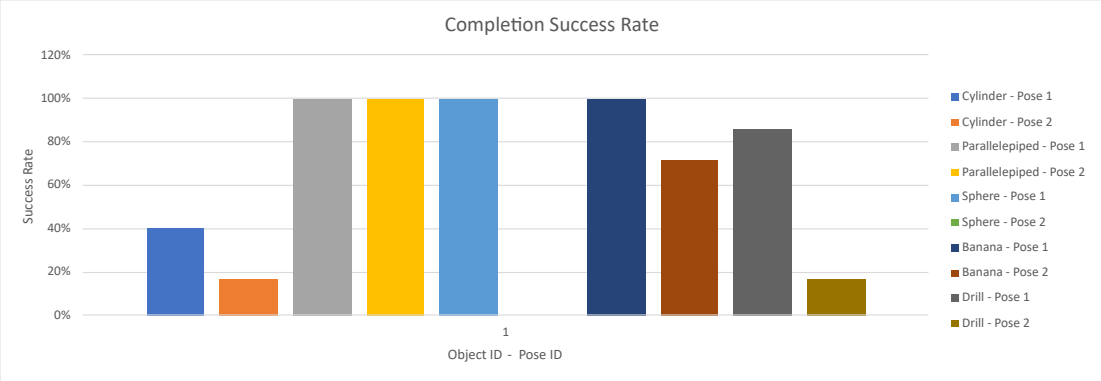

Figure 16: Grasp Completion Rate per object per pose for the first GDP.

Grasp Success Rate	83%
Grasp Completion Rate	62%

Table 1: Experimental Result, first GDP result overview.

3.2 Stress Test

The chosen network for the Grasp Rectangles prediction has been trained on the Cornell Dataset. All the sensor data in the dataset are acquired from the above of a static scene.

Test a network trained in this way with sensors data acquired from the two lateral GDPs (definition 2.3 and fig. 12) can only be seen as a stress test. In fact, we are not surprised to present the following result.

In fig. 17 and fig. 18 are shown the partial results showing the statistics per object and pose; table 2 shows the aggregated results. As shown by the figures, most of the grasps have not been completed, an exception are the banana and the drill. For these objects, due their elongated shape, some successful grasp has been predicted.

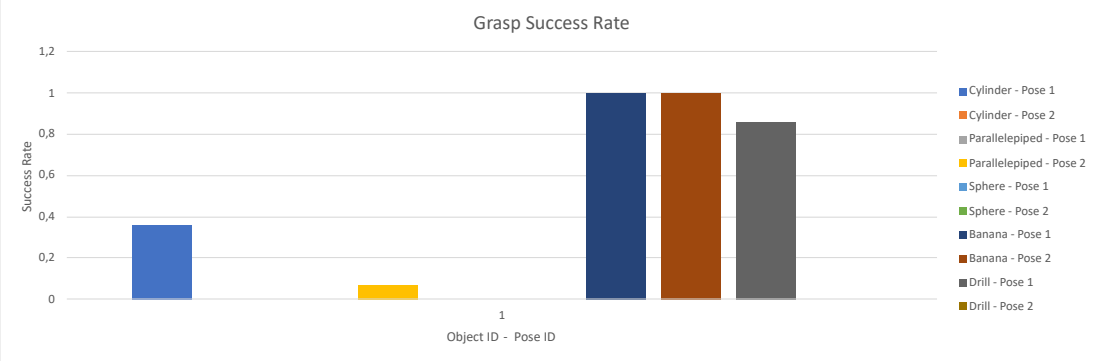

Figure 17: Grasp Success Rate per object per pose for the lateral GPDs.

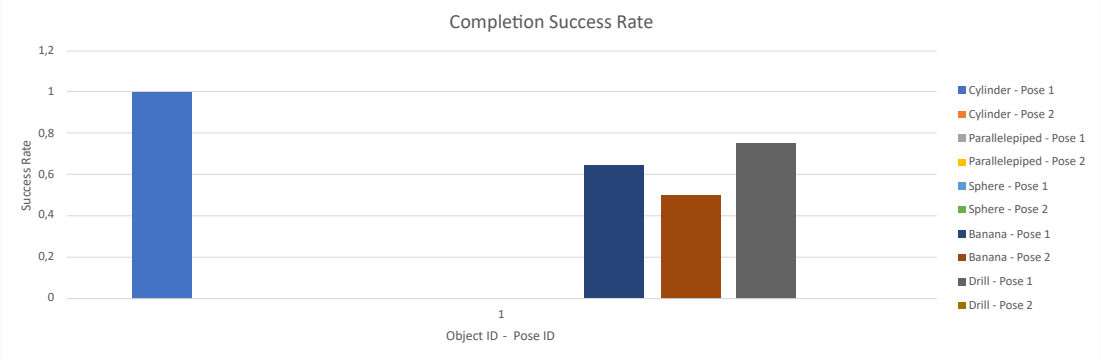

Figure 18: Grasp Completion Rate per object per pose for the lateral GPDs.

Grasp Success Rate	33%
Grasp Completion Rate	29%

Table 2: Experimental Result, GDP2 and GDP3 result overview.

4 Execution Time Considerations

Two measures have been considered regarding the amount of time the system requires to generate the set of grasp poses in response to a detection request.

The Inference Time is the elapsed time from the start of the inference until the generation of the output by the network.

The 6DOF Grasp Pose Generation Time is the elapsed time from the request of the grasp pose and the response, this time so include the execution of the entire Grasp detection and execution pipeline (section 1).

Concerning the measures defined above, table 3 reports the results obtained executing the pipeline on an NVIDIA Jetson Nano.

Inference Time [s]	0.46
6DOF Grasp Pose Generation Time [s]	0.49

Table 3: Execution times.

References

[CB93] I-Ming Chen and J.W. Burdick. "Finding antipodal point grasps on irregularly shaped objects". In: *IEEE Transactions on Robotics and Automation* 9.4 (1993), pp. 507–512. DOI: 10.1109/70.246063.

[Sic+08] Bruno Siciliano et al. *Robotics: Modelling, Planning and Control*. 1st. Springer Publishing Company, Incorporated, 2008. ISBN: 1846286417.

[JMS11] Yun Jiang, Stephen Moseson, and Ashutosh Saxena. "Efficient grasping from RGBD images: Learning using a new rectangle representation". In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 3304–3311. DOI: 10.1109/ICRA.2011.5980145.

[Gar+14] S. Garrido-Jurado et al. "Automatic generation and detection of highly reliable fiducial markers under occlusion". In: *Pattern Recognition* 47.6 (2014), pp. 2280–2292. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2014.01.005. URL: https://www.sciencedirect.com/science/article/pii/S0031320314000235.

[Cal+15] Berk Calli et al. "Benchmarking in Manipulation Research: Using the Yale-CMU-Berkeley Object and Model Set". In: *IEEE Robotics Automation Magazine* 22.3 (2015), pp. 36–52. DOI: 10.1109/MRA.2015.2448951.

[QGS15] Morgan Quigley, Brian Gerkey, and William D. Smart. *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*. 1st. O'Reilly Media, Inc., 2015. ISBN: 1449323898.

[KK17] Sulabh Kumra and Christopher Kanan. "Robotic grasp detection using deep convolutional neural networks". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 769–776. DOI: 10.1109/IROS.2017.8202237.

[LP17] Kevin M. Lynch and Frank Chongwoo Park. "Modern Robotics: Mechanics, Planning, and Control". In: 2017.

[CXV18] Fu-Jen Chu, Ruinian Xu, and Patricio A. Vela. "Real-World Multiobject, Multigrasp Detection". In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 3355–3362. DOI: 10.1109/LRA.2018.2852777.

[DDC18] Amaury Depierre, Emmanuel Dellandréa, and Liming Chen. "Jacquard: A Large Scale Dataset for Robotic Grasp Detection". In: *CoRR* abs/1803.11469 (2018). arXiv: 1803.11469. URL: http://arxiv.org/abs/1803.11469.

[DWL19] Guoguang Du, Kai Wang, and Shiguo Lian. "Vision-based Robotic Grasping from Object Localization, Pose Estimation, Grasp Detection to Motion Planning: A Review". In: *CoRR* abs/1905.06658 (2019). arXiv: 1905.06658. URL: http://arxiv.org/abs/1905.06658.

[Kle+20] Kilian Kleeberger et al. "A Survey on Learning-Based Robotic Grasping". In: *Curr Robot Rep* 1.4 (Dec. 2020), pp. 239–249. ISSN: 2662-4087. DOI: 10.1007/s43154-020-00021-6. URL: https://link.springer.com/10.1007/s43154-020-00021-6 (visited on 10/05/2021).

[KJS20] Sulabh Kumra, Shirin Joshi, and Ferat Sahin. "Antipodal Robotic Grasping using Generative Residual Convolutional Neural Network". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 9626–9633. DOI: 10.1109/IROS45743.2020.9340777.

[MCL20] Douglas Morrison, Peter Corke, and Jürgen Leitner. "Learning robust, real-time, reactive robotic grasping". In: *The International Journal of Robotics Research* 39.2-3 (2020), pp. 183–201. DOI: 10.1177/0278364919859066. URL: https://doi.org/10.1177/0278364919859066.

[Cao+21] Hu Cao et al. "Lightweight Convolutional Neural Network with Gaussian-based Grasping Representation for Robotic Grasping Detection". In: *arXiv e-prints*, arXiv:2101.10226 (Jan. 2021), arXiv:2101.10226. arXiv: 2101.10226 [cs.CV].

[Int] Intel. *Intel RealSense Depth Camera D435*. https://www.intelrealsense.com/depth-camera-d435/. Accessed: 2021-10-05.

[Nira] Niryo. *Niryo One Mechanical Specifications*. https://niryo.com/docs/niryo-one/user-manual/mechanical-specifications/. Accessed: 2021-10-05.

[Nirb] Niryo. *Niryo One Ros Stack*. https://niryo.com/docs/niryo-one/developer-tutorials/get-started-with-the-niryo-one-ros-stack/. Accessed: 2021-10-05.

[Roba] Open Robotics. *About ROS*. https://www.ros.org/about-ros/. Accessed: 2021-10-04.

[Robb] Open Robotics. *Dynamic Reconfigure*. https://wiki.ros.org/dynamic_reconfigure. Accessed: 2021-10-05.

[Robc] Open Robotics. *Parameter Server*. https://wiki.ros.org/rospy/Overview. Accessed: 2021-10-04.

[Robd] Open Robotics. *ROS Concepts*. `https://wiki.ros.org/ROS/Concepts`. Accessed: 2021-10-04.