UNIVERSITÀ DEGLI STUDI DI SALERNO – DIEM

LM-32 INGEGNERIA INFORMATICA

Course of

Cognitive Robotics

A.Y. 2018/2019

Final Project

Boccuzzi Stefano
Bottiglieri Ilaria
Palatucci Ugo
Pasquale Donatella

Summary

1. Introduction

Purpose of the document

This document aims to describe the design and development work to allow a Niryo One Robot to be able to insert in the boxes the highest number of balls having the same color, in a given time interval: this represents the final project of the Cognitive Robotics course.

This document is attached to the final project as official documentation, in order to offer a complete description of the functional specifications of the system, as well as the hardware components used and the software architecture adopted.

Purpose of the project

The purpose of the project consists in three principle steps:

1. Find a ball
2. Put the ball in the box having the same color
3. Repeat STEP 1 until one of the following conditions is verified
    I. There are not more balls
    II. The time is expired

2. Functional System Specifications

Design choices

FIrst thing to do was to analyze all the possibilities for the camera's position and understand all the problems concerned. Indeed, the camera's position inevitably influences other important choices like the strategy to search the objects and the vision algorithm to understand the workspace.

In the specific, the team deepened three main possibilities: zenithal position, on robot's arm, or use two cameras on both positions.

Using two cameras the team expected to be the more accurate and fast set. One camera had to be connected to the Jetson and one had to be connected to the robot. With this set, the first camera could capture a snapshot of the objects and the second camera could precisely control the robot movements to grab the balls. In the robot's configuration the team encounter a lot of problems installing the required Intel Realsense libraries and build the robot's ROS workspace so it had to leave this path.

As discussed before, the zenithal camera give the possibility to have a scene's snapshots and this means to be able to handle a workspace that can change fast. Additionally, control algorithm can be simpler than a camera on the arm because there's not need to implement a algorithm to map the workspace. On the other hand have a camera on the arm intrinsically give the possibility to do a correction during the ball's grab.

The team chose to use a zenithal camera after some experimentations. The reason was that the precision using this set was enough to grab the balls without corrections. And also, if needed, a correction could be added by making a gripper recognition or adding a second camera.

Hough Circle Transform

Hough Circle Transform is a basic technique used in digital image processing, for the detection of circular objects in a digital image; with the following technique it is possible to extract the features for the detection of the circles which is composed of two main phases: first of all, the detection of

the edges and the search for the possible centers of the circle is made. Then, the best radius is found for each candidate center.

Hough Circle Transform was used together with a HSV color-based recognition.

Viola Jones

Another technique used was Viola-Jones for the recognition of the boxes and the gripper. This technique was not used for ball recognition because the pixel size of them was too small (less than 20x20 pixels) in order to recognize them with Viola-Jones, therefore, the Hough Circle technique presented above was chosen.

Moreover, after some experiments the Viola Jones algorithm with a zenithal and fixed camera turned out to be the best solution with respect of a Neural Network. With Viola Jones the computation time is significantly shorter than the Neural Network and the training is also simpler.

Viola Jones on Box

Regarding the recognition of the boxes:

Positives	2.423
Negatives	16.524

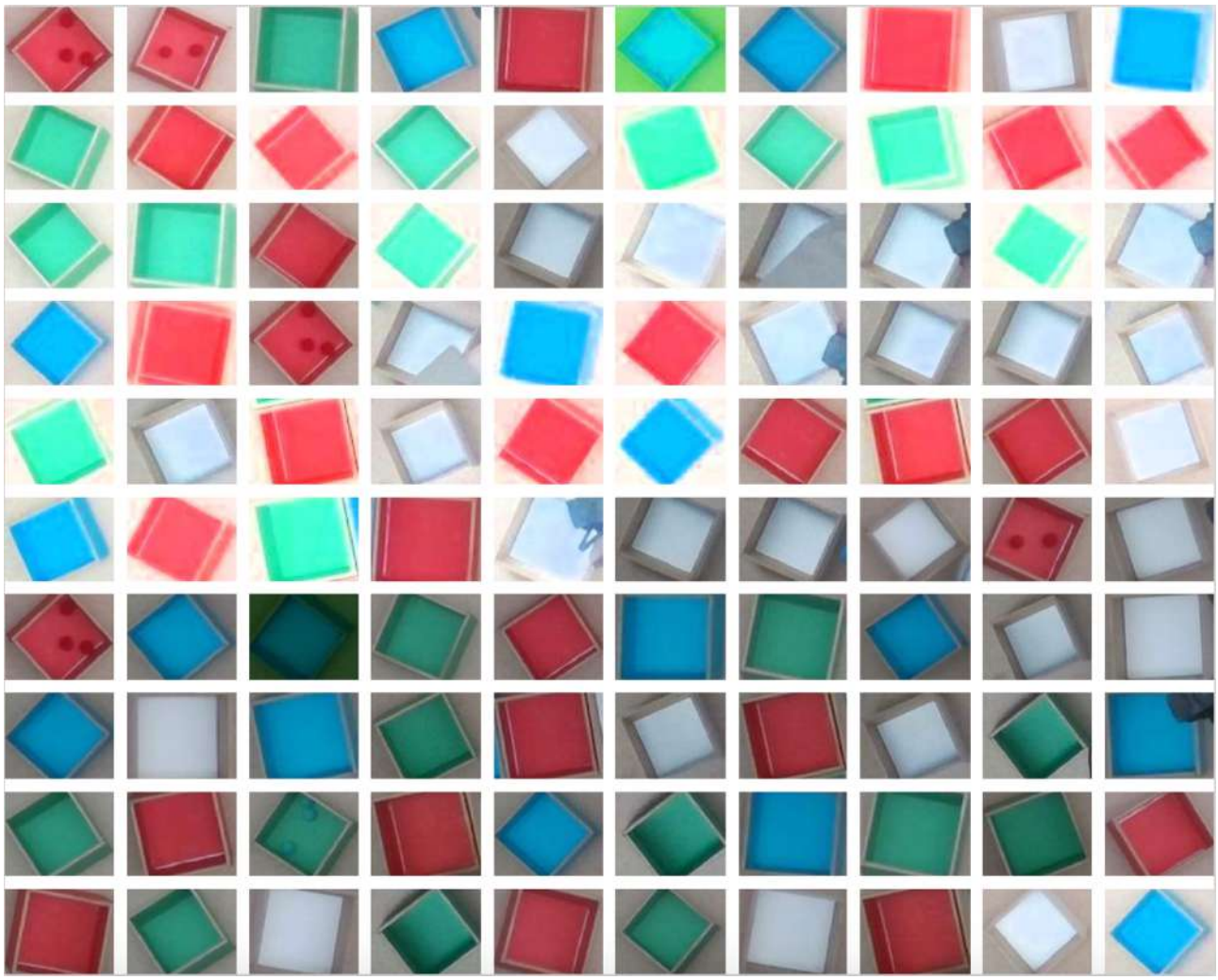


Figure 2-1 - Positives examples used for the Box Classifier

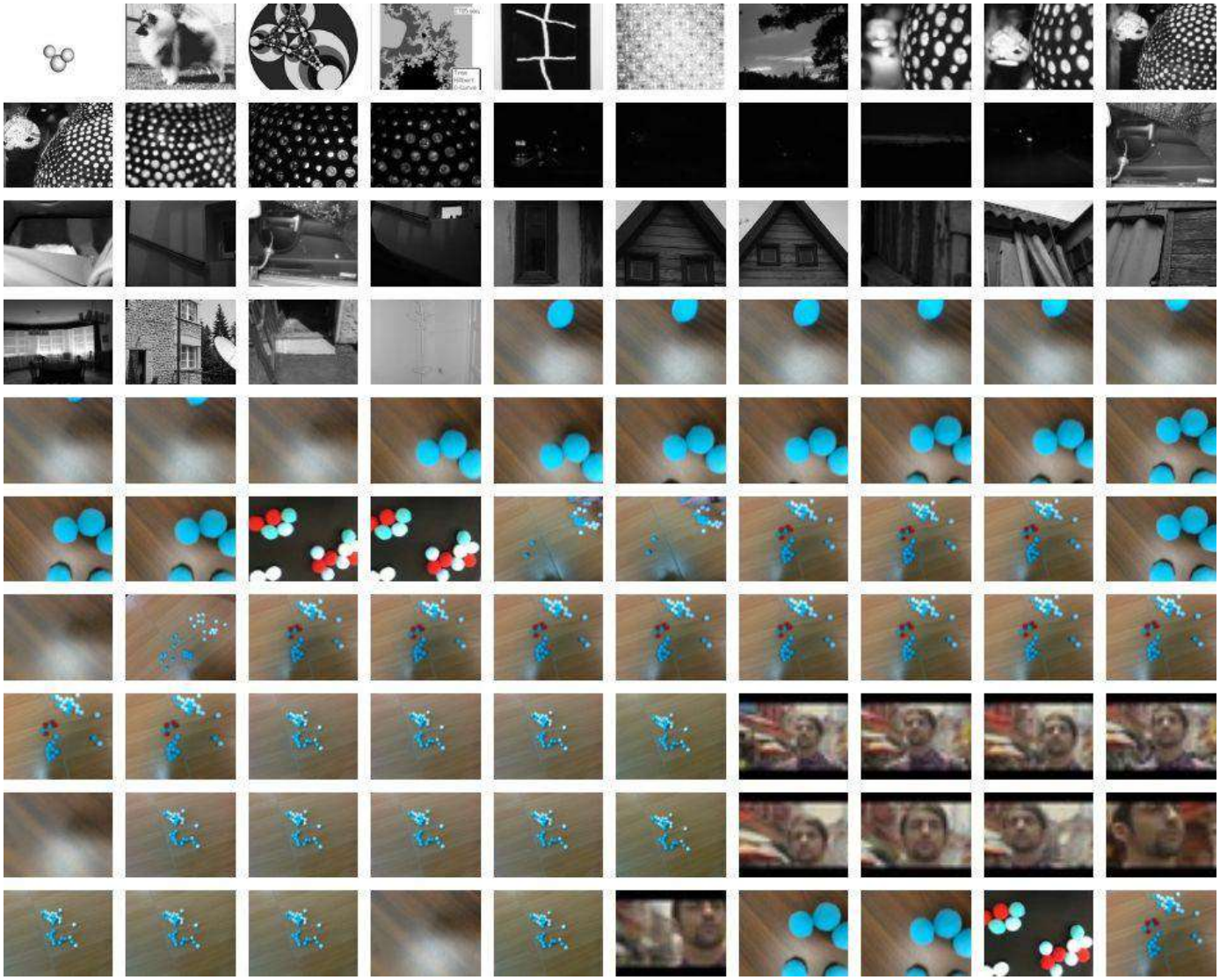


Figure 2-2 - Negatives examples used for the Box Classifier

Examples of Boxes detection:

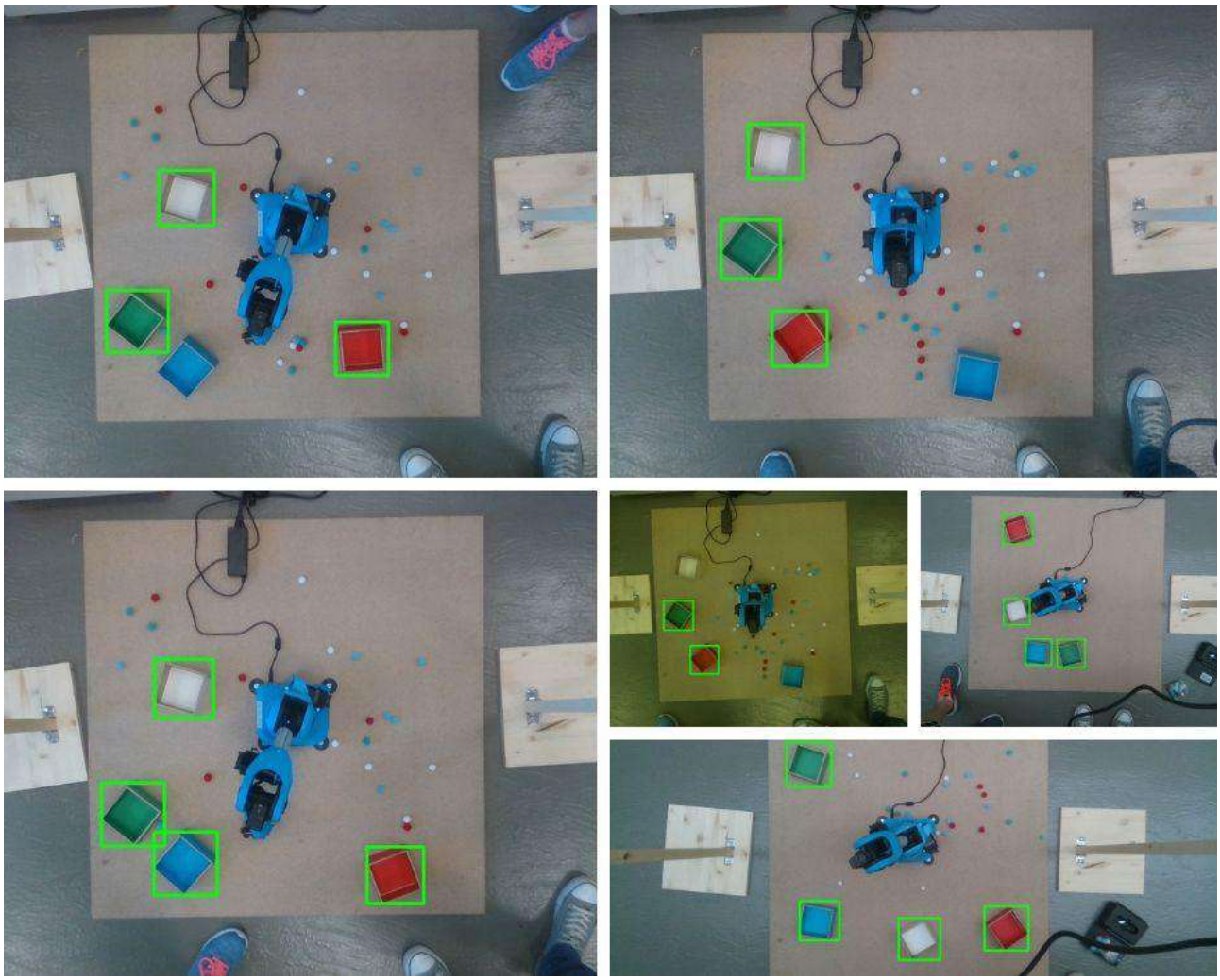


Figure 2-3 - Example boxes detection with Viola Jones

Viola Jones on Gripper

The gripper classifier was made with the initial idea of adding feedback to the system.

In the final project, the system presented is not feedbacked, because it was decided to concentrate on solving some problems about vision, trying to optimize it, and to solve problems on several trajectories, decided by the algorithm, that led to errors in the program.

Therefore, in order to solve these structural problems, it was decided to stop the development of feedback part.

However, to detect the gripper the team tried several methods like print some patterns on the gripper and the Viola Jones. OpenCV have some methods to detect patterns like QR-code and chessboard but they took around 7-6 seconds to found the pattern, also not in every configurations the camera could see all the pattern so the team decided to move on and change strategy.

The following are images of the realized classifier:

Positives	4.899
Negatives	27.371

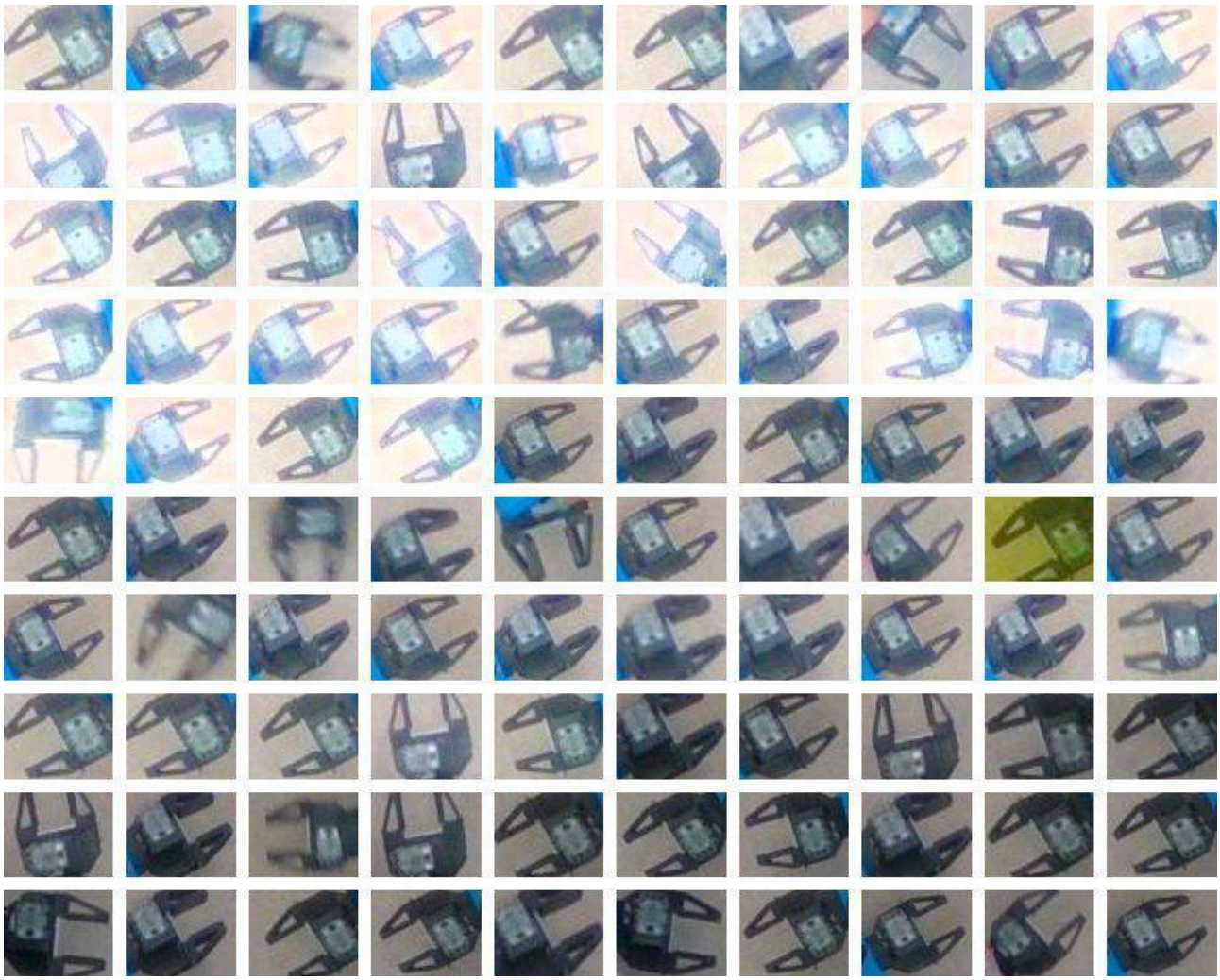


Figure 2-4 - Positives examples used for the gripper classifier

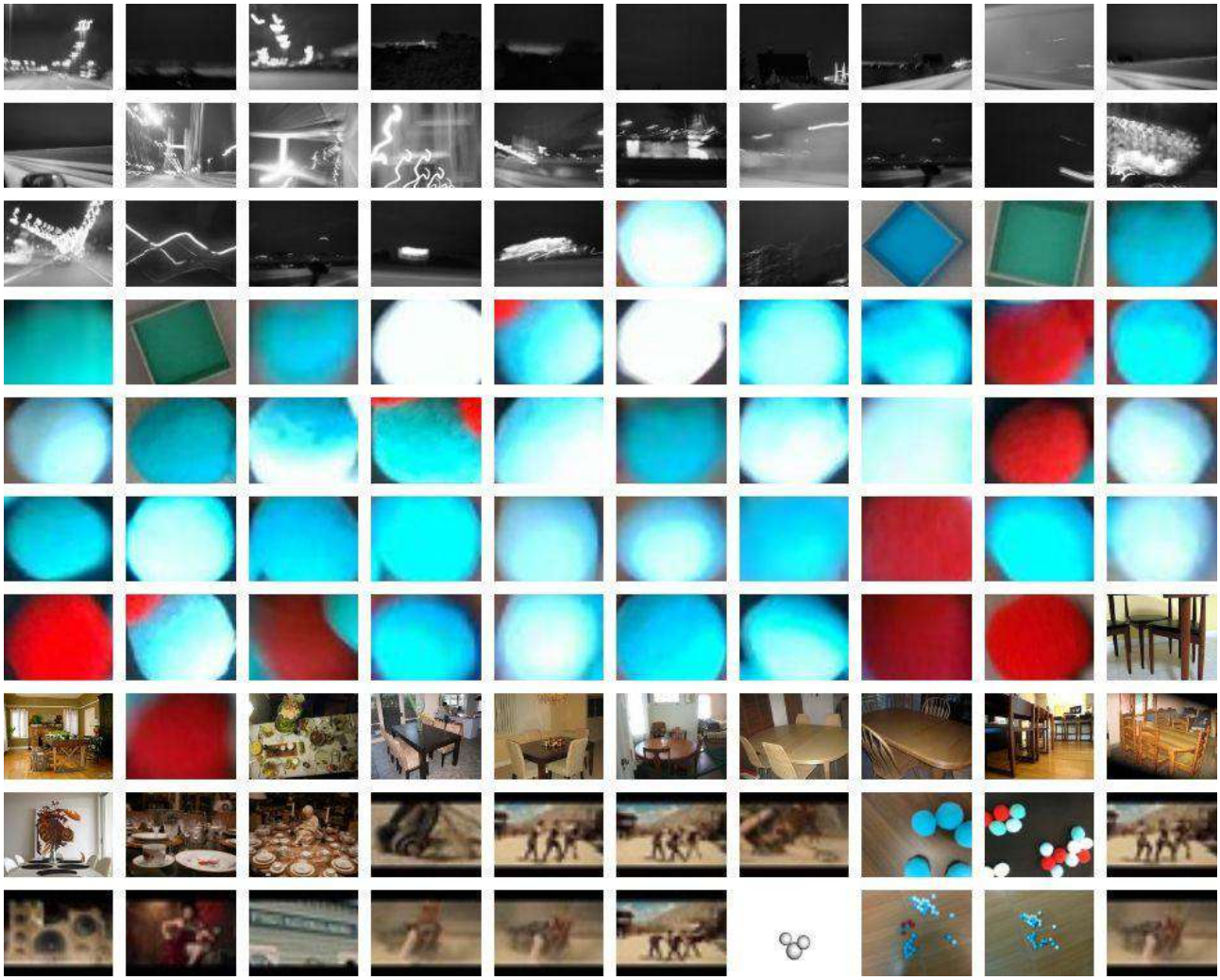


Figure 2-5 - Negatives examples used for the Gripper classifier

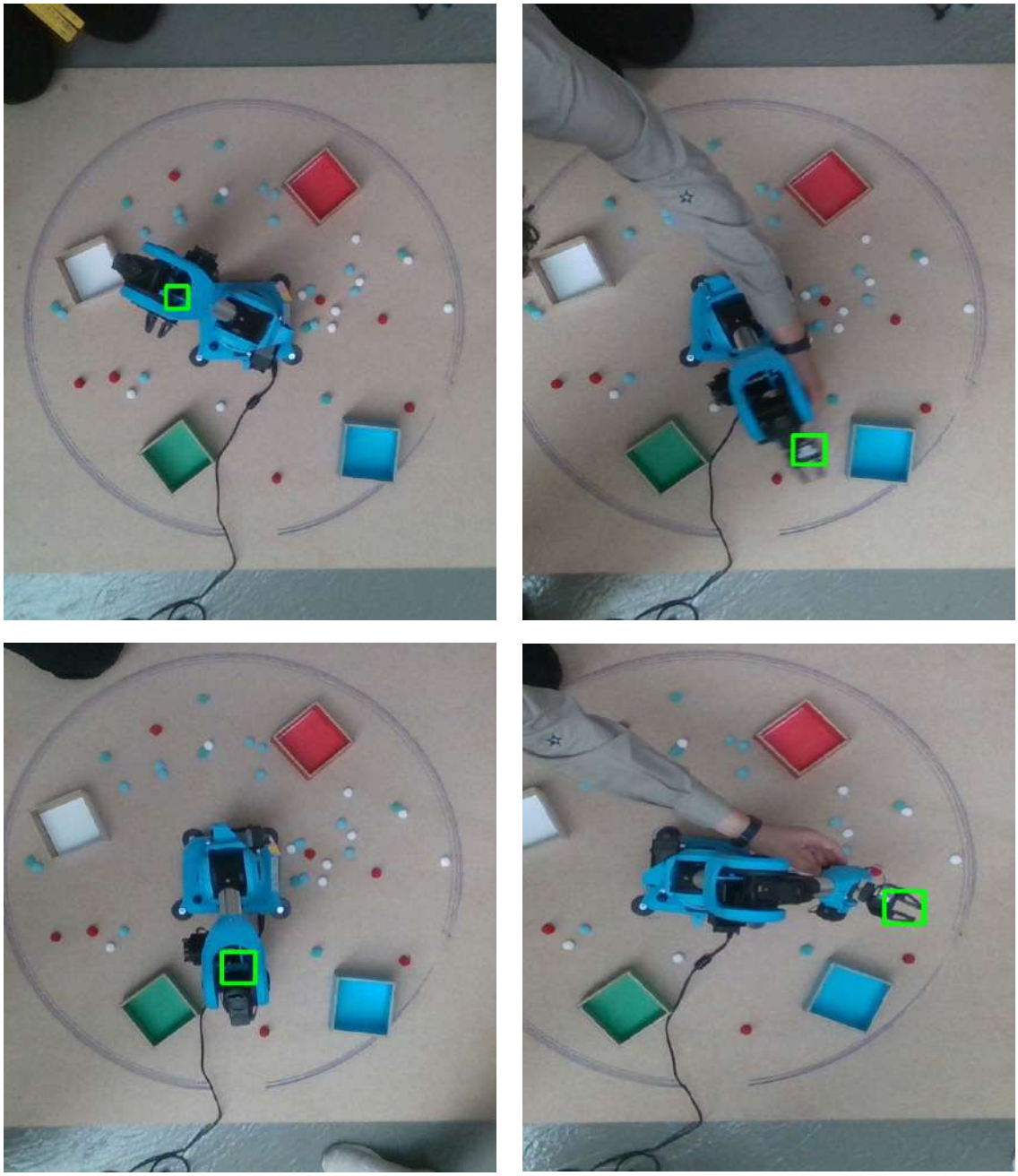


Figure 2-6 - Example Gripper detection with Viola Jones

3. Hardware architecture

Intel Realsense Depth Camera D435

Figure 3-1 - Intel RealSense Depth Camera D435

The Intel RealSense Depth Camera D435 captures stereo depth in a variety of applications that help perceive the world in 3D. The camera includes the Intel RealSense Vision Processor D4 featuring high depth resolution - up to 1280x720 at 30 frames per second (fps), long-range capabilities, global shutter technology and a wide field of view. With the global image shutter and wide field of view (91.2° x 65.5° x 100.6°), the Intel RealSense Depth Camera D435 offers accurate depth perception when the object is moving or the device is in motion, and it covers more field of view, minimising blind spots. It is designed for easy setup with USB 3.0 and is in a portable form factor and has integrated its versatile Solomon AccuPick 3D SV software, Solomno 3D bin picking Using Intel RealSense D435, the best-in-class 3D camera using active stereo vision.

Technical Specifications

	D435
Use Environment	Indoor/Outdoor
Depth Technology	Active IR stereo
Image Sensor Technology	Global Shutter, 3 um x 3 um pixel size
Components Included	Realsense Vision Processor D4

	Realsense Module D430
Depth Field of View	85.2deg x 58deg (+/-3deg)
Depth Stream Output Resolution	Up to 1280x720
Depth Stream Output Frame Rate	Up to 90fps
Minimum Depth Distance (MinZ)	0.11m
Max Range3	~10 meters Varies depending in calibration, scene, lighting condition
RGB Sensor Resolution & Frame Rate	1920x1080 at 30fps
RGB Sensor FOV	69.4deg x 42.5deg (+/-3deg)
Camera Dimension	90 mm x 25 mm x 25 mm
Connectors	USB 3.0 Type-C
Mounting Mechanism	One ¼ -20 UNC thread mounting point Two M3 thread mounting points

Table 3-1 - Technical Specifications Intel RealSense

Highlights

This depth camera includes:

- A powerful vision processor that uses 28 nanometer (nm) process technology and supports up to five MIPI Camera Serial Interface 2 lanes to compute real-time depth images and accelerate output
- A new and advanced stereo depth algorithm for more accurate and longer-range depth perception
- A set of image sensors that enable capturing of disparity between images up to a 1280 x 720 resolution
- Support for the new cross-platform and open source Intel® RealSense™ SDK 2.0
- A dedicated color image signal processor for image adjustments and scaling color data
- Active infrared projector to illuminate objects to enhance the depth data

Advantages

- With the rolling image shutter and narrow field of view, the Intel® RealSense™ Depth Camera D415 offers high-depth resolution when the object size is small and more precise measurements are required.

- With the global image shutter and wide field of view (91.2° x 65.5° x 100.6°), the Intel® RealSense™ Depth Camera D435 offers accurate depth perception when the object is moving or the device is in motion, and it covers more field of view, minimizing blind spots.

Niryo One

Niryo One is an accessible 6 axis robotic arm, made for makers, education, and small companies. The robot is 3D printed and powered by Arduino, Raspberry Pi, and Robot Operating System. STL files and code on the robot will be open source after the first shipments.
More than a product, we are building a complete set of cloud services around Niryo One, and a community around open source projects. Having 6 axis, Niryo One can grab anything in its area, with any given orientation. Endless number of applications are possible by using Niryo One: pick and place objects with a suction pump, a gripper, or an electromagnet tool; automate your 3D printer; drill and so on.

Figure 3-2 - Niryo-One

Dimensions

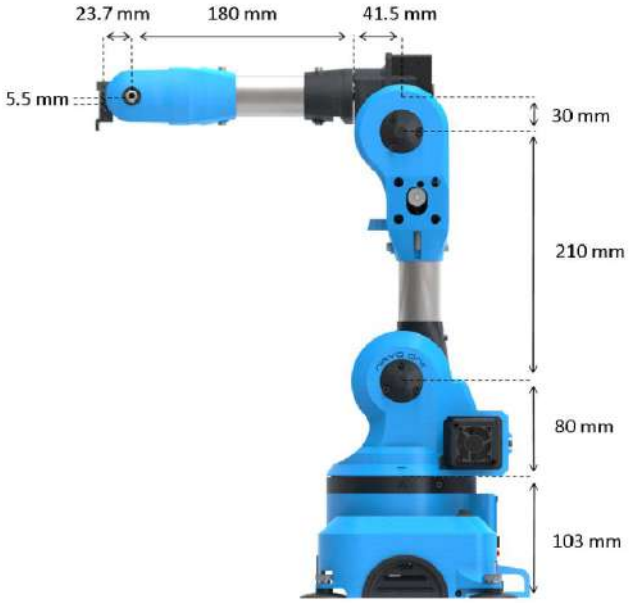
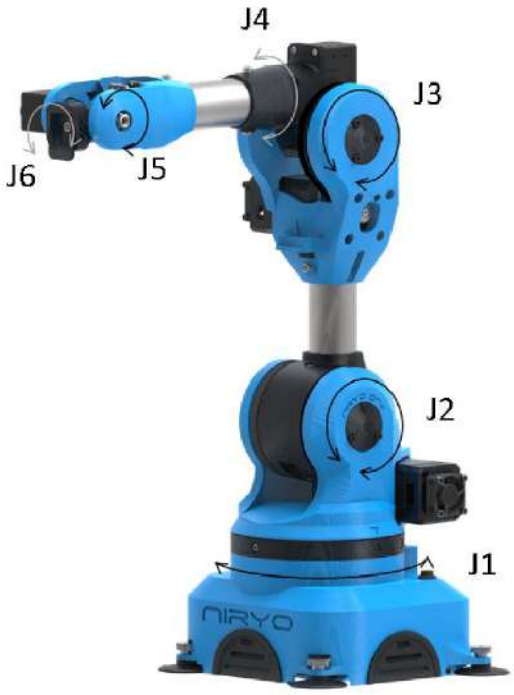


Figure 3-3 - Niryo-One Dimensions

Degrees of Freedom	6
Weight	3,3 kg
Reach	440 mm

Max Rotation



	Min	Max
J1	-175°	175°
J2	-90°	36.7°
J3	-80°	90°
J4	-175°	175°
J5	-100°	110°
J6	-147.5°	147.5°

Figure 3-4 - Niryo-One Max Rotation

Technical specifications

Specification	Value
Number of axis	6
Weight	3.2 kg
Max Payload	0.5 kg
Max Reach	440 mm
Base joint range	360 °
Repeatability	+/- 0.5 mm
Communication	Ethernet | Wi-Fi | Bluetooth | USB
User interface	Web app | Android app | iOS app | Gamepad
Programming interface	ROS | APIs | source code
Power Supply	12 V – 7 A
Power Consumption	~ 60 W
Materials	Aluminum, PLA (3D printing)
Motors	5 steppers + 2 servos
Ports	Ethernet + 4 USB
Hardware	Arduino Mega + Raspberry Pi 3
Joint sensor	Magnetic sensor

Table 3-2 - Niryo-One Technical Specifications

Nvidia Jetson

The NVIDIA® Jetson Nano™ Developer Kit is a small AI computer for makers, learners, and developers. After following along with this brief guide, you'll be ready to start building practical AI applications, cool AI robots, and more.

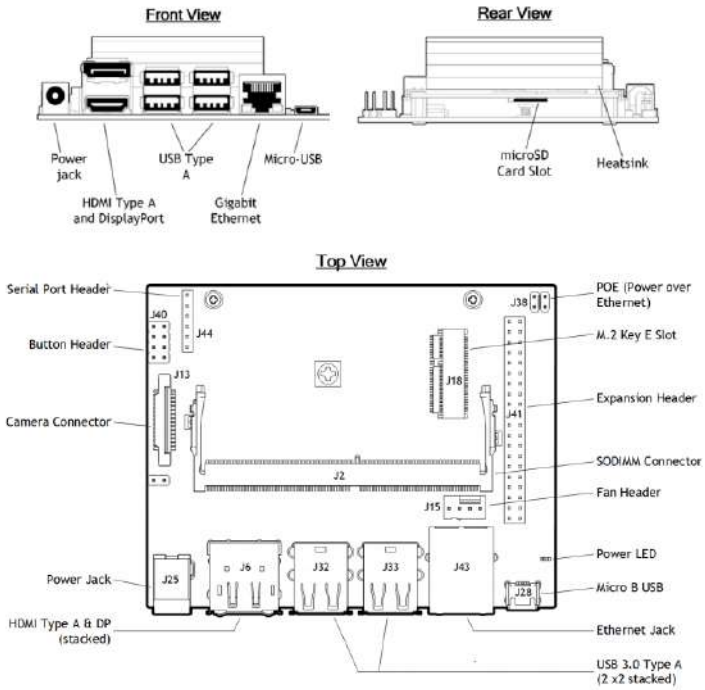


Figure 3-5 - NVIDIA Jetson

The Jetson Nano Developer Kit box includes:

- Jetson Nano Developer Kit
- Small paper card with quick start and support information
- Folded paper stand.

But we also need:

- microSD card (16GB UHS-1 minimum)
- USB keyboard and mouse
- Computer display (either HDMI or DP)
- Micro-USB power supply (5V⎓2A)

Technical Specifications

GPU	NVIDIA Maxwell Architecture with 128 NVIDIA CUDA® cores
CPU	Processore Quad-core ARM® Cortex®-A57 MPCore
Memory	LPDDR4 4 GB 64-bit
Storage space	16 GB eMMC 5.1 flash drive

Video encoding	4K to 30 (H.264 / H.265)
Video decoding	4K to 60 (H.264/H.265)
Camera	12-way (3x4 or 4x2) MIPI CSI-2 DPHY 1.1 (18 Gbps)
Connectivity	Gigabit Ethernet
Display	HDMI 2.0 or DP1.2 | eDP 1.4 | DSI (1 x2) 2 simultaneous
UPHY	1 x1/2/4 PCIE, 1x USB 3.0, 3x USB 2.0
I/O	1x SDIO / 2x SPI / 4x I2C / 2x I2S / GPIOs
Dimensions	69,6 mm x 45 mm
Mechanics	260 pin side connector

Table 3-3 - Technical Specifications NVIDIA Jetson

System architecture

Since there were some problems about power supply, a specific power supply was built using a power supply with a jack of 6 mm with a step down that carries the voltage of 12 Volts and 3 Amps at 5 Volts and 5 Amps (so it is regulates both in voltage and amperage).

A jumper has been inserted on the Jetson in order to start it from the jack and not from the usb.

A USB adapter was used for the wi-fi network: since the Jetson is at a height of 1.60 meters, it has been connected to the robot using wi-fi.

With the computer, it was not possible to connect to the Jetson using Robot's wi-fi. This can, probably, be explained by the impossibility of keeping too many connections open at the same time. Then, the connection with the Jetson is made via Ethernet cable and by establishing on the computer a DHCP Server, through "Tftpd32" software, capable of giving a static IP to the Jetson.

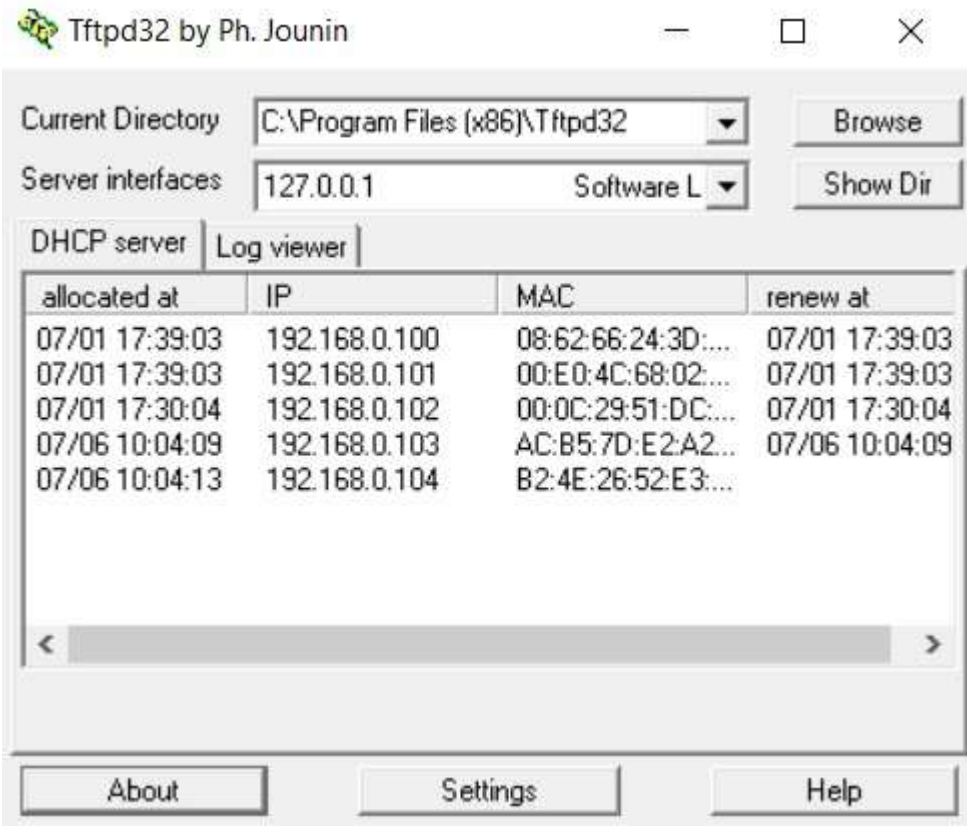


Figure 3-6 - "Tftpd32" software

By connecting to both the Jetson and the Robot using ssh, it is possible to start the demo by launching the scripts described in the next chapter.

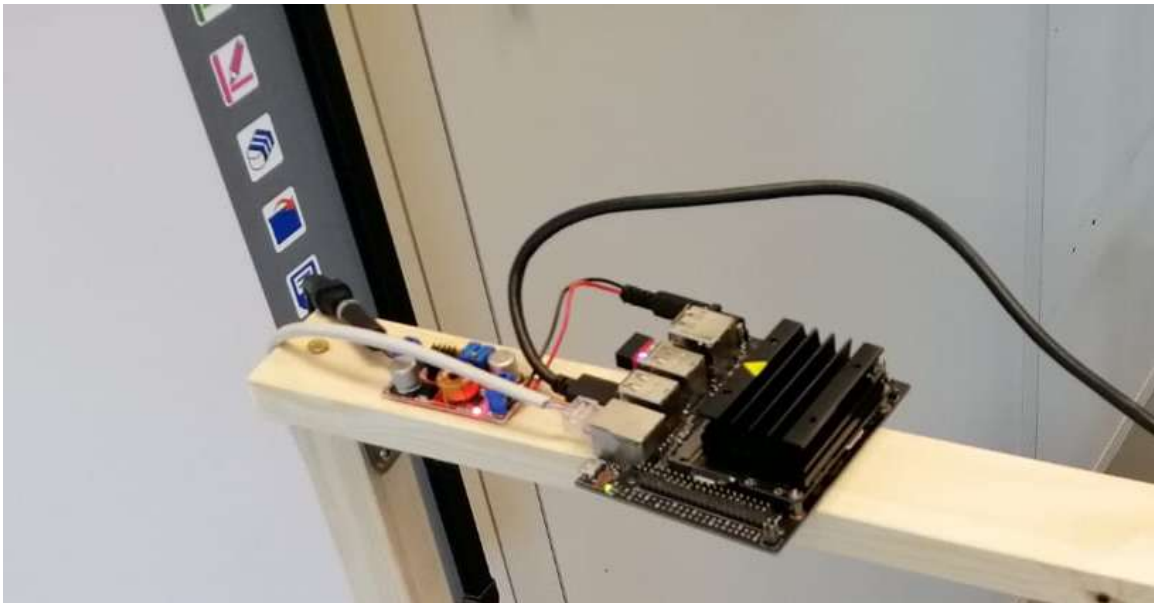


Figure 3-7 - connecting Jetson

4. Software architecture

For the realization of the project, two .py files have been created:

- *object_detection.py*: presents on NVIDIA® Jetson and inside it is built a ROS (detector) node that performs both the role of Subscriber and of Publisher. Subscriber to be able to register on topic to the real sense Camera, in order to get the real sense camera frames to be able to perform the detection operations of boxes and balls. Publisher is to be able to transmit to the Robot the topic '/objects' a dictionary containing the information about the balls and boxes detected by the single frame.
- *controller.py*: presents on the raspberry of the Robot and inside it is built a ROS node (controller) that performs the role of Subscriber, in order to obtain the dictionary of balls and boxes from on topic '/objects' in which the robot is registered and to carry out the handling and gripping operations of the balls.

  Finally, publishes on the topic 'niryo_one/commander/robot_action' the actions that the robot can undertake to send them to the ROS (RobotMoveAction) node. The latter will carry out the handling operations.

The ROS scheme of the nodes and topics is shown below.

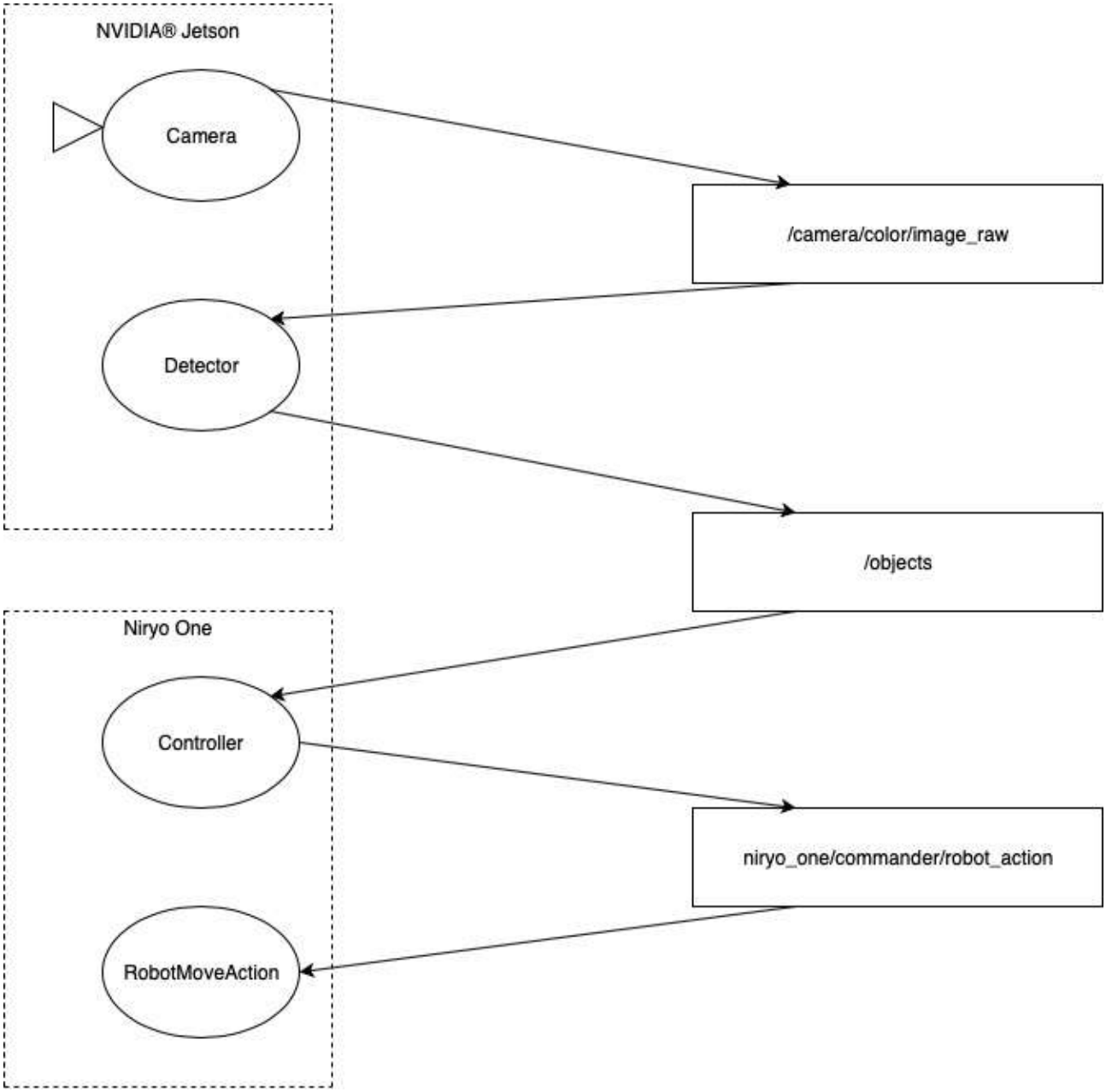


Figure 4-1 - ROS scheme

object_detection.py

The “ObjectDetection” class, as previously mentioned, initializes the ROS 'detector' node, after which from the .json configuration file, we can take the configuration information related to the Classifiers, the RGB colors both for the balls and for the boxes as well as further configuration parameters for example for conversions from pixels to meters.

From the topic '/ camera / color / image_raw' of which the following node is a Subscriber the detect_ball_boxes function is called.

The following function allows to take the frame from the real sense camera and reduce the frame size taking into consideration only the work space we need. For this reason it is important that the chamber is centered to perfection, and the robot is perfectly configured in a central position, to then be precise in the grip and in knowing the precise position of balls and boxes.

In this phase we decided to use as a data structure to transfer messages to the controller node a python dictionary divided into balls and boxes, with further subdivisions of color; and as value you will have an array of the coordinates identified for the individual class and the single color.

1. The first operation of the detect was the detection of the boxes, as mentioned in the previous chapters we used Cascade Classifier for the detection of the boxes, but we were also interested in the detection of the color of the box. Having the bounding box of the boxes, To determine the box's color, it is built a region's histogram and take the most prevalent color present inside of it. This most prevalent color is compared with four RGB colors that represent the ground truth colors of the boxes. If the most prevalent color is near one of those colors, the algorithm decide the color of the box. Otherwise the box is considered as a false positive. For each box detected we can add to the python dictionary the coordinates of the center of the box with the relative conversion of pixels to meters.
2. The second operation of detect concerned the detection of the balls. It turned out that the RGB comparison It didn't work well with different illumination environment for the balls. In fact above all the green and blue balls have similar RGB codification. For this reason we decided to set a mask for all different balls colors using a HSV image codification. It is applied the inRange function of opencv to create a mask in which is set to 255 the pixel that have a representation in the range and 0 the pixel that haven't. We defined a different range in HSV for all the colors. After, It is applied the hough transform to this mask and find the circles for the specific color. With this algorithm the different ball's colors are detected correctly, but with a group of ball near to each other the hough transform have some difficult to find all the circles. It is not necessary to find all the circles in a group of balls because if at least one ball is found, and the robot take the ball correctly, another ball can be found after. For each ball detected we can add to the python dictionary the coordinates of the center of the ball and the radius with the relative conversion of pixels to meters.

Finally, the topic '/ objects' is published by the Publisher, the conversion from json into String of the dictionary containing all the coordinates.

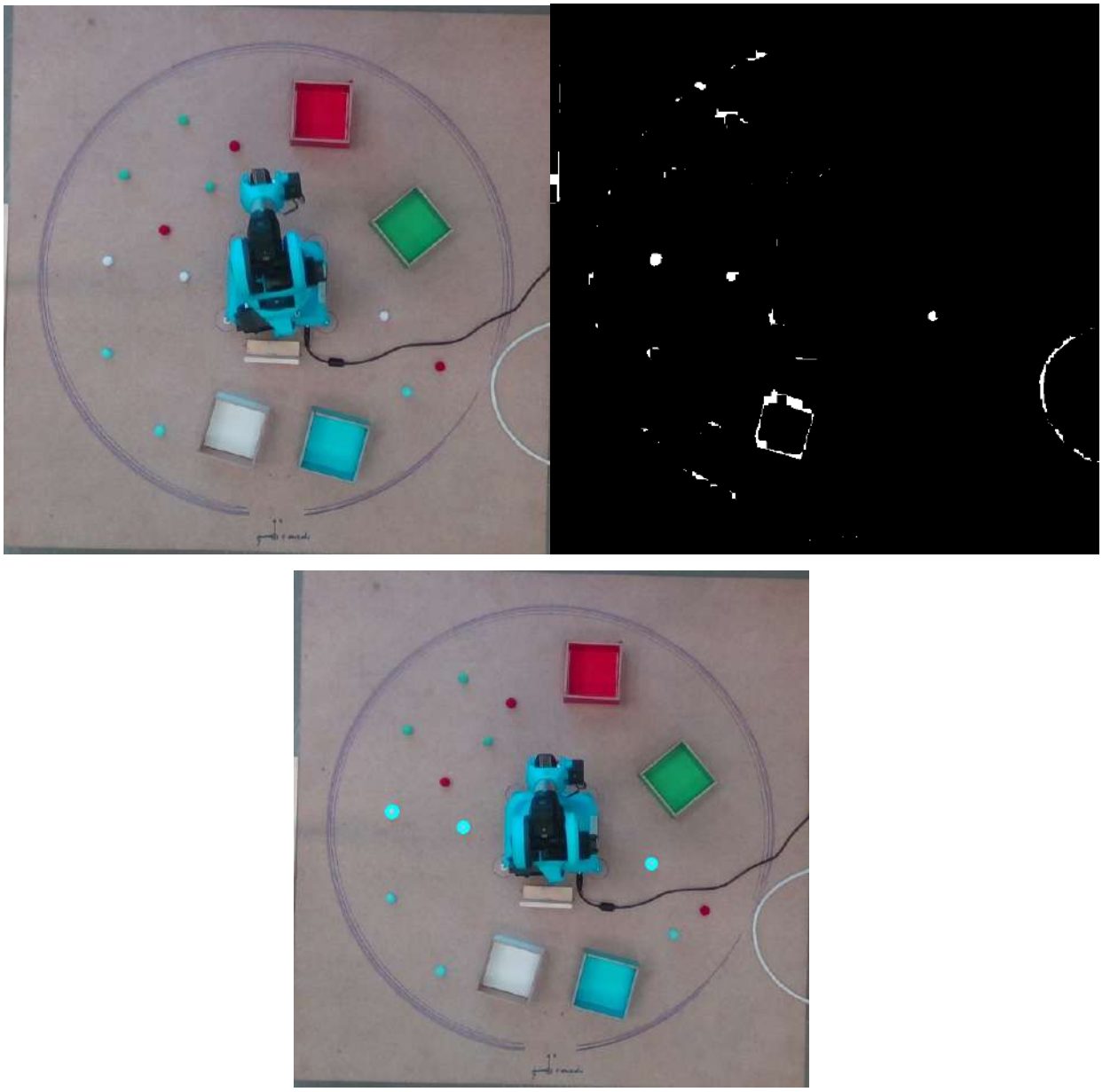


Figure 4-2 - Example of white balls detection

controller.py

The Controller class as previously mentioned initializes the ROS node "controller".

The action function is called from the topic '/ objects' of which the following node is a Subscriber.

The following function allows you to retrieve the python dictionary of the coordinates, and so it will be possible to continue to the next step in which you have to decide the ball to be taken by the gripper.

Scrolling through the various ball coordinates, we are going to exclude the balls that are:

- inside a box
- close to a box since in this case the gripper could fail to take them, or hit the box and then a new configuration of the Robot would then be necessary.
- ball which the box of the same color was not detected.

On detecting a ball we have its coordinates, and it will be possible to proceed with the operations of:

- movement of the gripper up to the position of the ball, remaining at a certain height (and ensuring that the gripper is open)
- lowering of the gripper to be able to hook the ball
- gripper closure and return to the default height.

At the end of this operation, if the box has undergone a new change of coordinate, the robot updates the new position and can proceed:

- to the gripper movement until the center of the box is laid
- opening the gripper to insert the ball in the box
- return to the starting configuration position.

The return to the starting configuration position is mainly due to the fact that having the real sense Camera in Zenithal position, if the robot was in a different configuration from the starting one (state of rest) it could cover box and ball and distort the detection of the latter.

As a last addition, the FAILURE HANDLING problem has also been solved, since the Robot could generate errors such as the wrong calculation of the trajectory, and in this we can guarantee the continuous functioning of the system.

The robot movement scheme is shown below.

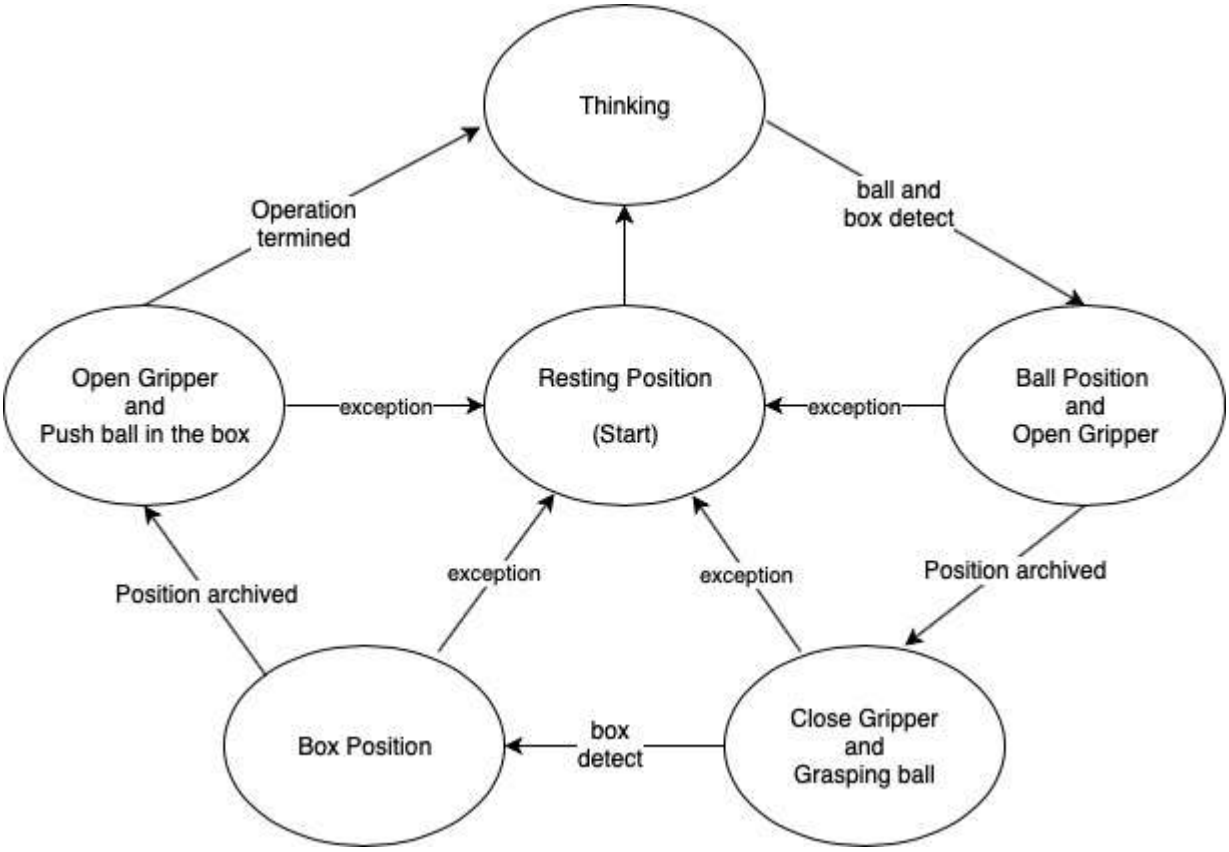


Figure 4-3 - Robot movement scheme

5. Results and Conclusions

In conclusion, even if the realized system is not feedbacked and the numerous problems encountered, the movement of the robot, with the purpose of taking a ball and inserting it in the right box, is quite accurate. Therefore, the lack of feedback has not been a great loss considering the required task. Anyway, for future developments, it might be useful to add it, trying not only to see the initial solution for the gripper recognition with Viola Jones. Indeed, the proposed solution involves a change to the gripper position that could cause some problems in the grasping of objects that are in the workspace.

For what concern the choice to have the camera in a zenithal position, although it had its downsides, allowed to the overall system to be reactive to changes in the workspace (for example the change in the positions of the boxes).
Another solution could be to add a second camera on the robot arm in order to avoid problems that can be encountered with the gripper recognition using Viola-Jones technique. In this way, there will also be a greater speed given by the zenithal camera as well as having greater precision for the addition of this latter camera.