Robotics Project

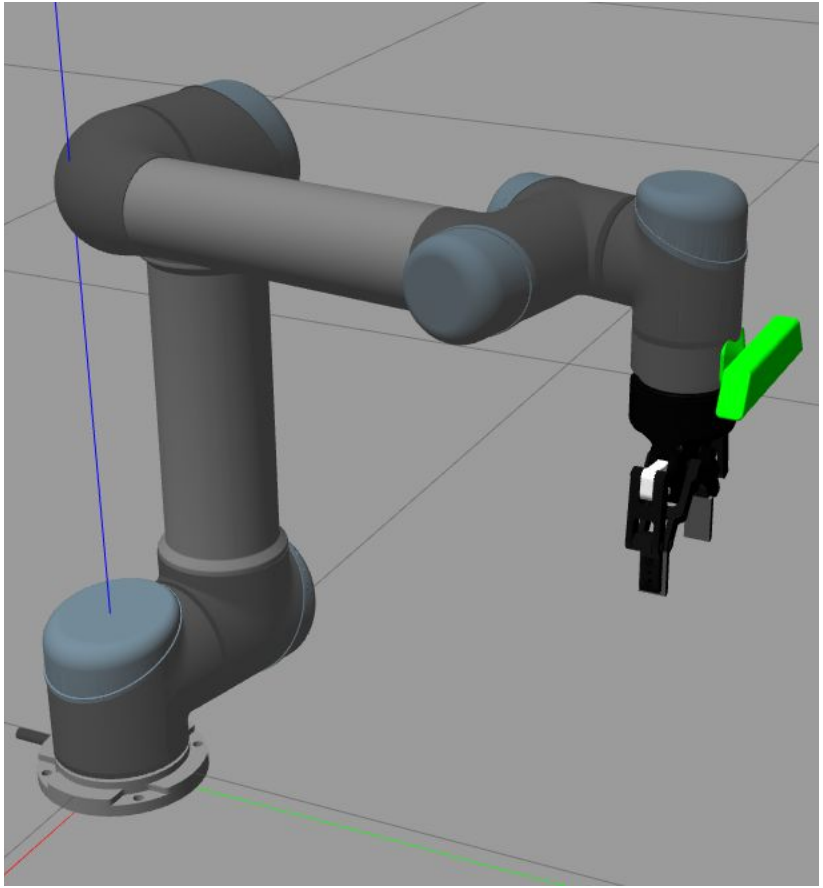
Andreolli Fabio Moiola Christian Trainotti Samuele

Introduction

The project consists in creating a C ++ program using ROS and Gazebo, that is able to move the robot in order to perform specific pick-and-place operations.

These operations consist in moving pieces of lego to specific positions.

The project is divided into 4 tasks with increasing implementation difficulties.

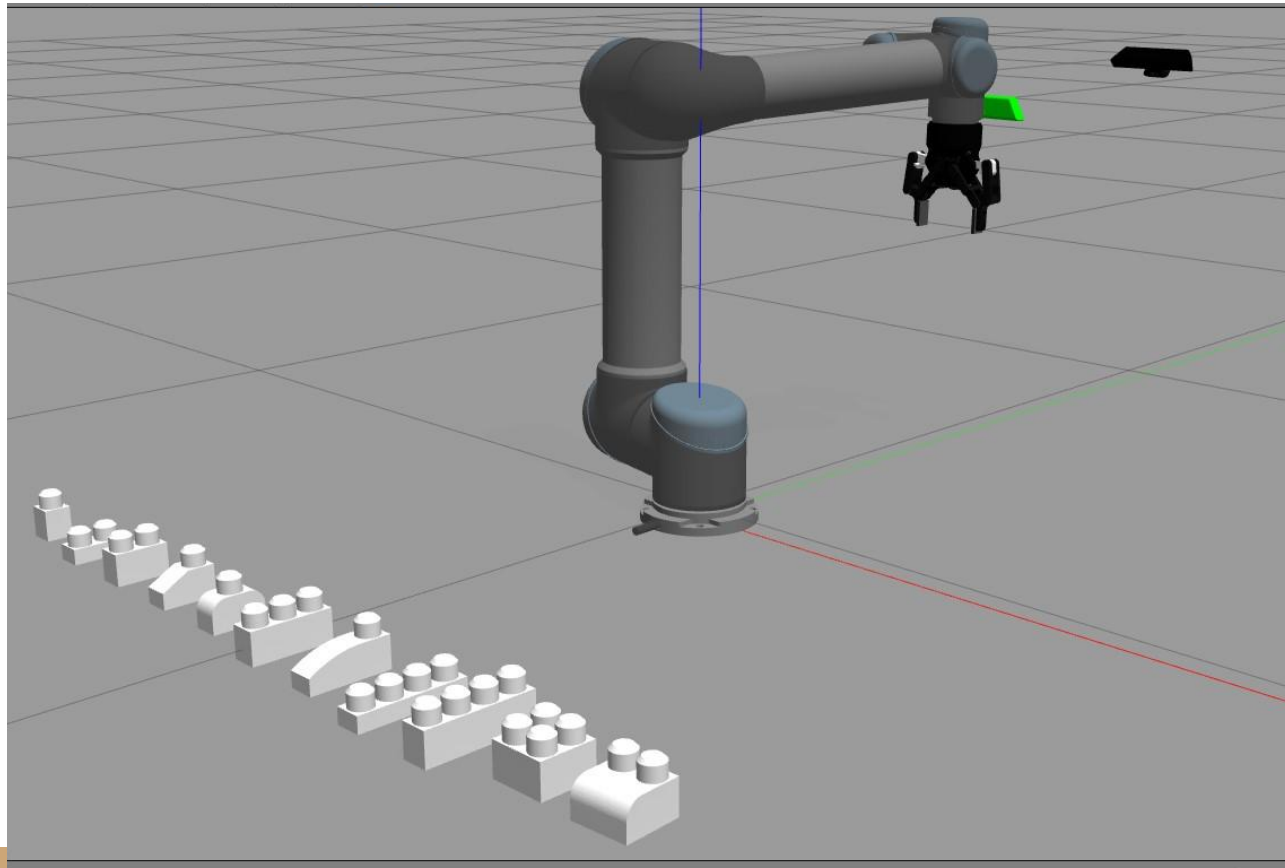


Position of the pieces

The positions of the various classes of pieces are those shown in the image.

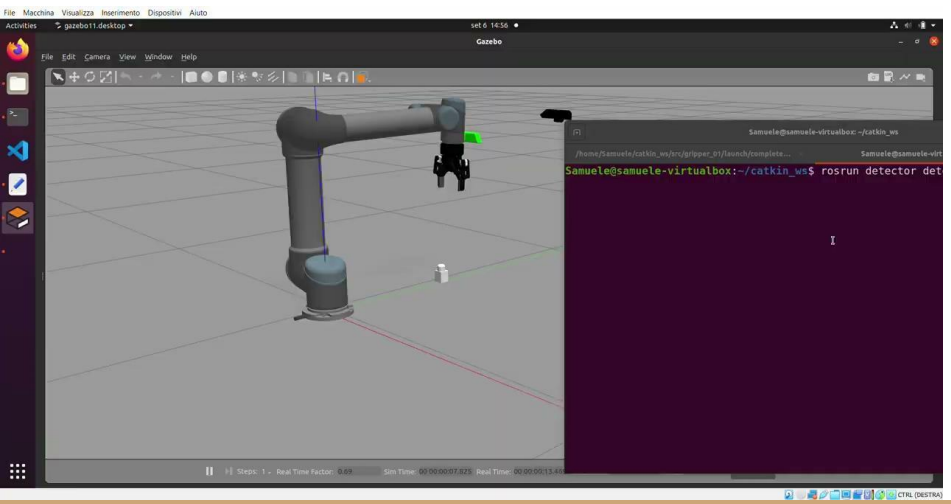
The lego placeholders have been removed during the simulations to make the program lighter.

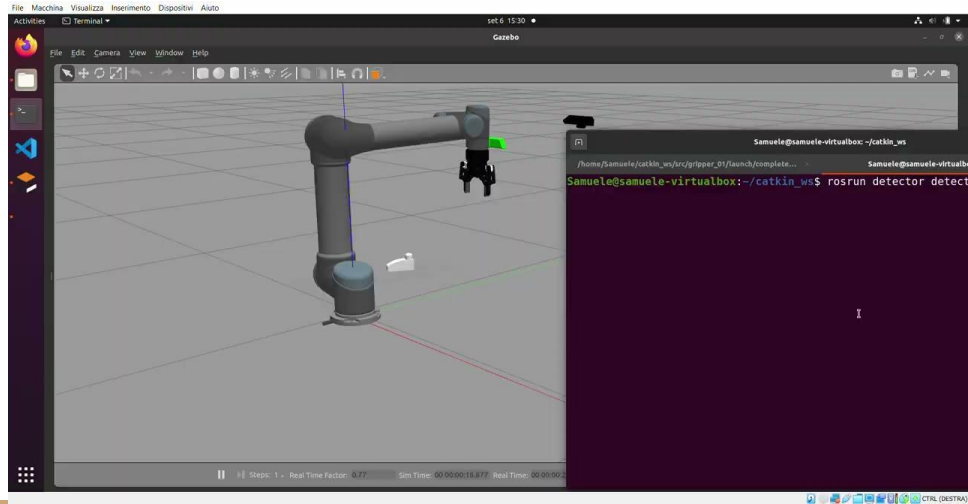
However the pieces are placed in the correct position and the detected class of the piece is well visible in the bash screen.



Assignment 1

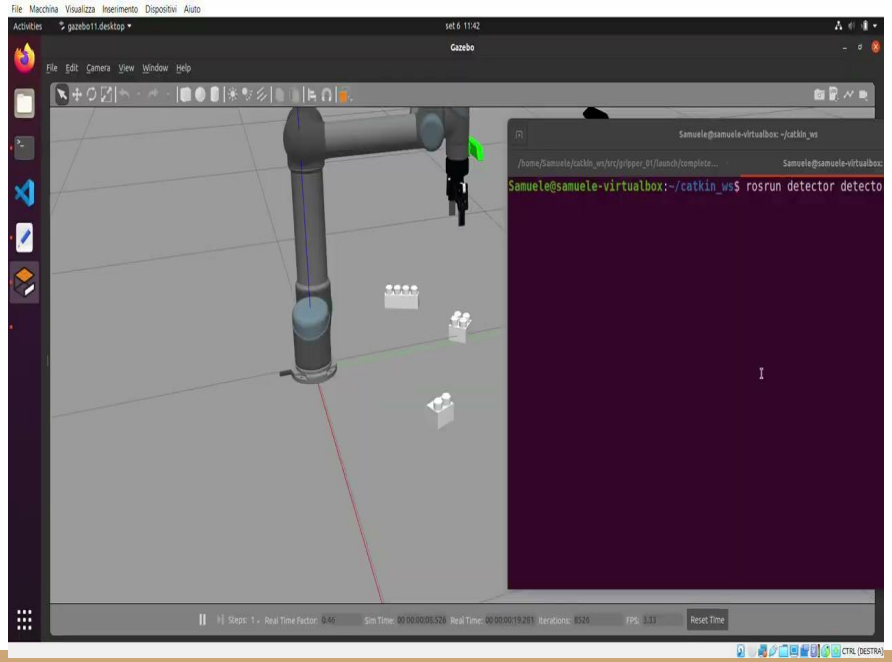
In the first assignment there is only one piece that is positioned naturally on the ground and the scope is to recognize this piece and put it in the right place according to its type.





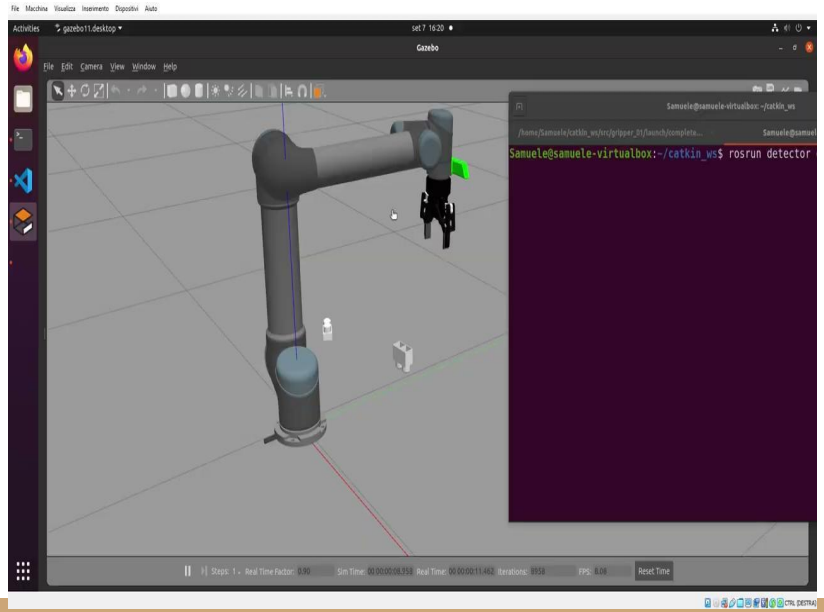
Assignment 2

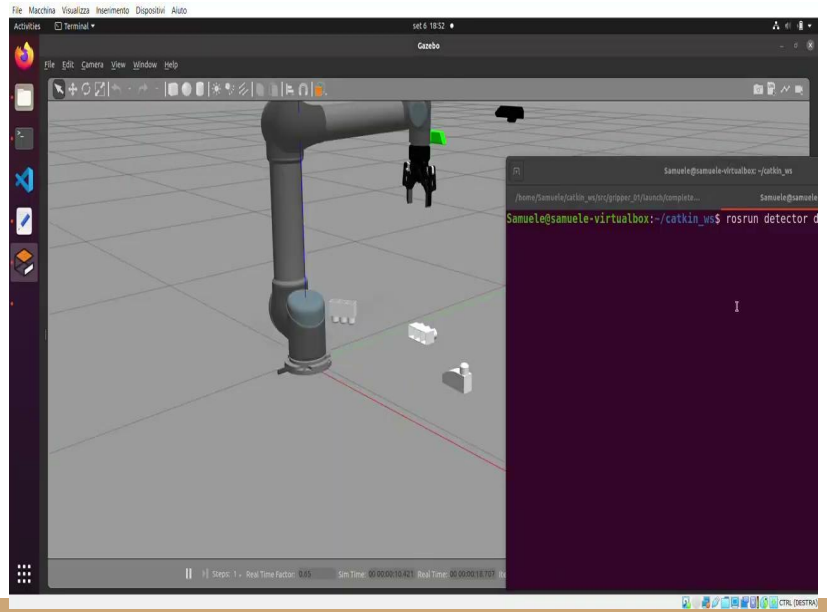
The second task is to recognize the different types of objects (only one for each class) and to position them in the right place according to their class.



Assignment 3

This task consists in recognizing different types of objects that could be on its top or on its sides. After that the robot has to take these objects and put them in the right place according to the class. If there are several objects of the same class, the legos must be stacked.

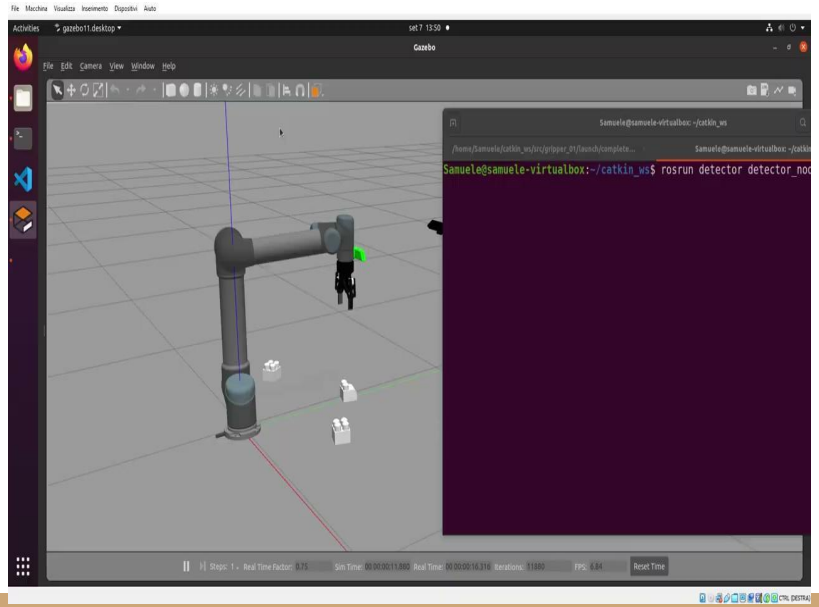




Assignment 4

This task has the same requirements as the previous one, except for the final position.

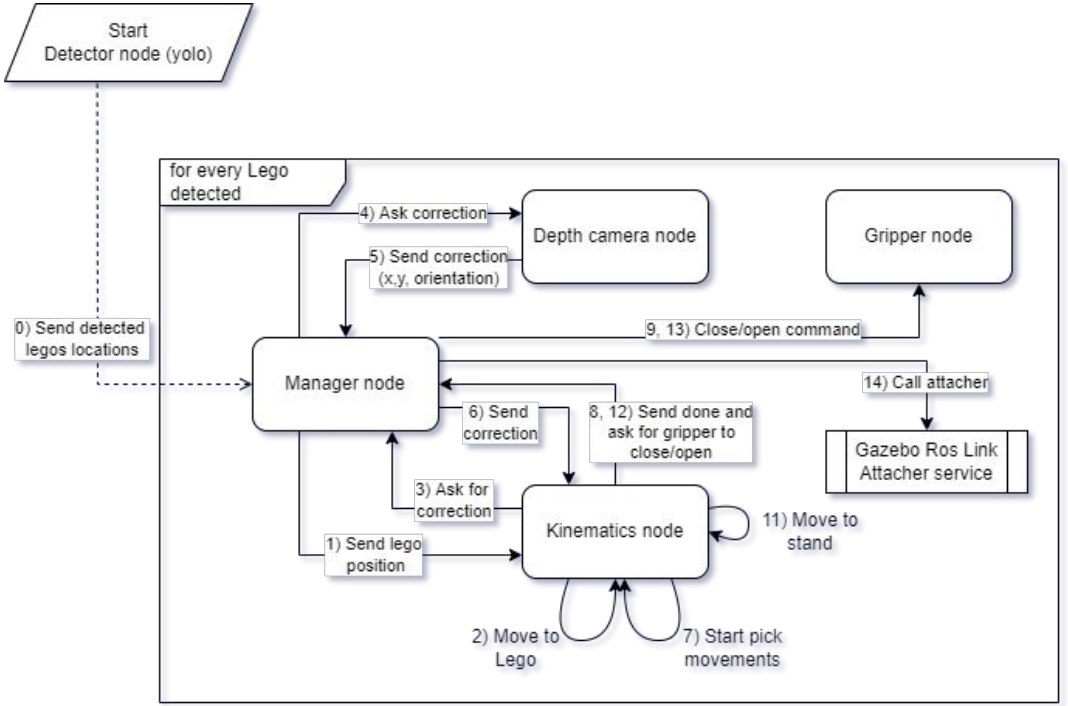
In this case objects have to be stacked in a proper way in order to make some predefined building.



How did we organize the project?

The program is divided mainly in 5 parts:

- Manager
- Detect Camera
- 3d sensor
- robot
- gripper



Manager

The manager is in charge of directing and coordinating all the parts of the project.

Its first job is to wait for the Detect node to return the class of each piece detected and its approximate positioning.

After that, the arm is moved to a predefined height above each piece and the 3D sensor node is called, which returns precise coordinates of the piece and its orientation in space.

Finally, the robot and gripper nodes are used to take the piece, turn it if necessary, and move it to the desired position.

Manager

Entering more specifically, the manager communicates through the other nodes of the program, receiving and sending the data necessary for each node to proceed

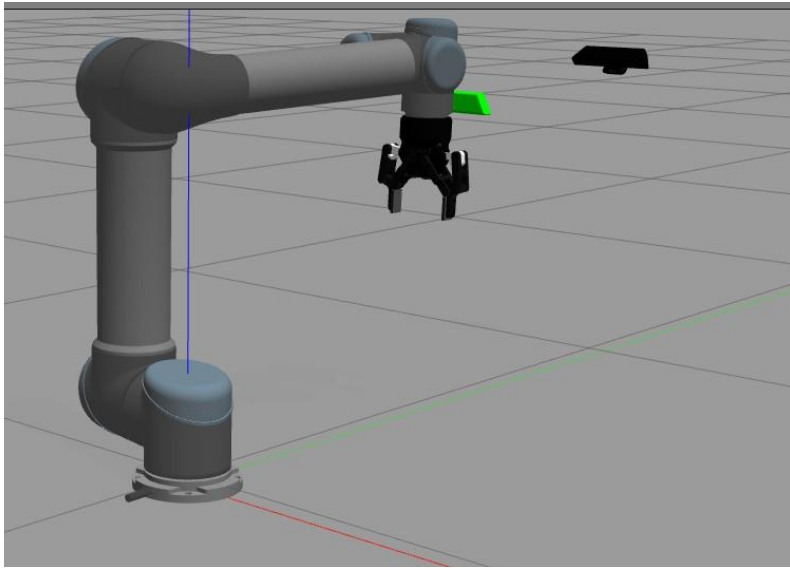
This is done through ROS messages.

Detect Camera

The Detect module takes care of the preliminary detection of objects. This results in detecting their class and approximate position in the work plane.

In order to do this, it was decided to use a normal 2D camera placed at a predetermined height, distance and angle with respect to the work surface.

The positions of the objects are fixed until the robotic arm intervenes, so it was decided to use a single photo coming from the camera and not the constant video in order to make the program lighter and faster.



Detect Camera

The node then proceeds to take a picture of the worktop by the camera and then save it.

After that, the following sub-blocks are executed, they analyze the photo and extract from it the information necessary for the manager to continue his work.

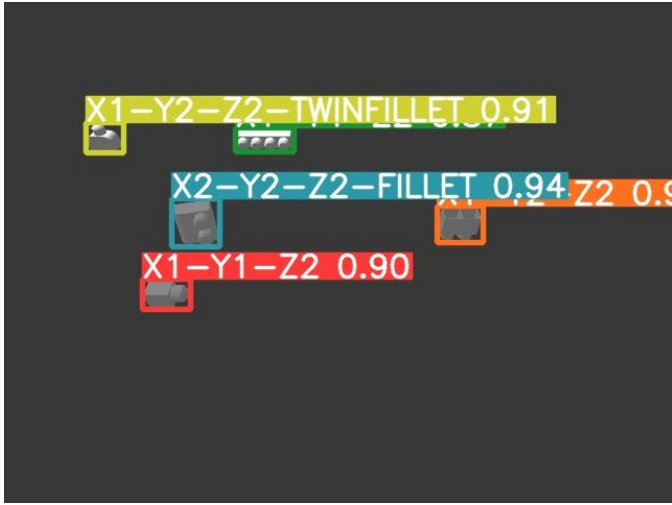
Detect Camera - YOLO

It was decided to use the latest version of Yolo available, YoloV5.

This software uses python for its execution unlike the rest of the program which is in C++.

Yolo allows to analyze the image and detect the objects. It has been trained to recognize legos inside the photo.

The software returns a text file containing the classes of the identified objects, their position inside the image and the dimensions expressed as the width and length of the square that contains the identified piece.

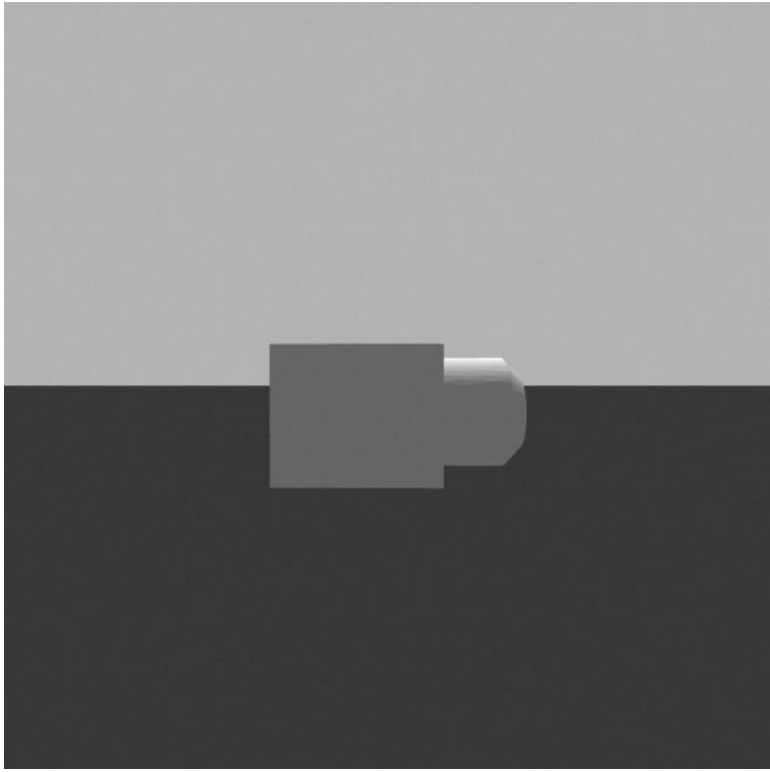



```
1 8 0.389062 0.273958 0.0875 0.0479167
2 0 0.242188 0.582292 0.071875 0.0604167
3 4 0.15 0.269792 0.05625 0.0604167
4 2 0.68125 0.442708 0.06875 0.0729167
5 10 0.285937 0.439583 0.071875 0.0958333
```

Detect Camera - YOLO training

In order to allow Yolo to work with the legos needed for the project, it is necessary to carry out a customized training of the software.

The program is designed for this and provides a training script to which it is necessary to provide a set of images already labeled and through these creates the .pt file which will subsequently be used to detect the pieces.



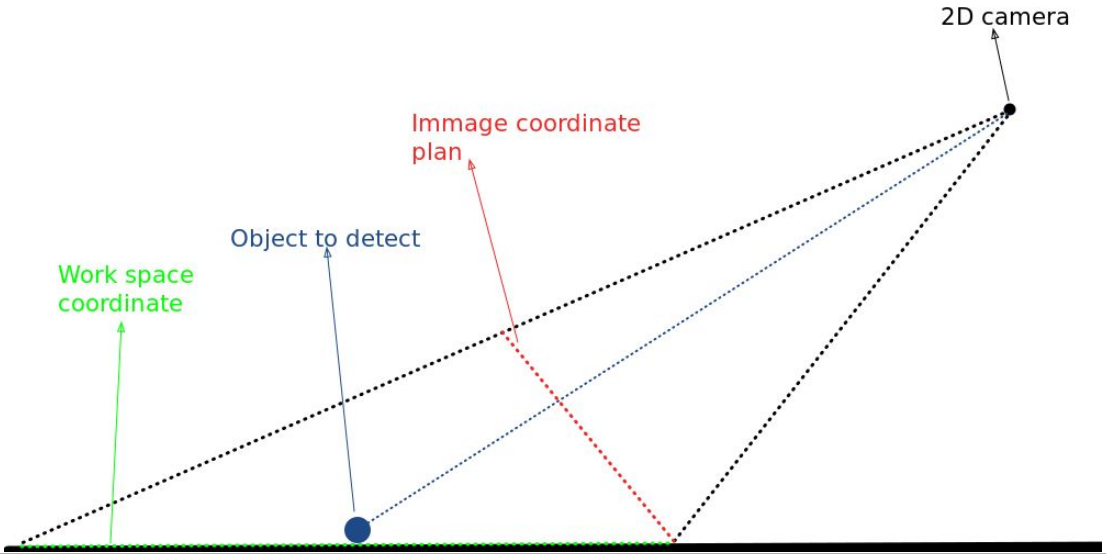
```
1 0 0.5140625 0.53984375 0.34140625 0.1984375
```

Detect Camera - coordinate extraction

Once Yolo has finished predicting the various positions and classes they are saved in a text file in a specific format.

A dedicated program reads such data and analyzes them in order to transform the coordinates of the image into Gazebo coordinates.

Finally, these data are sent to the Manager who will use them.

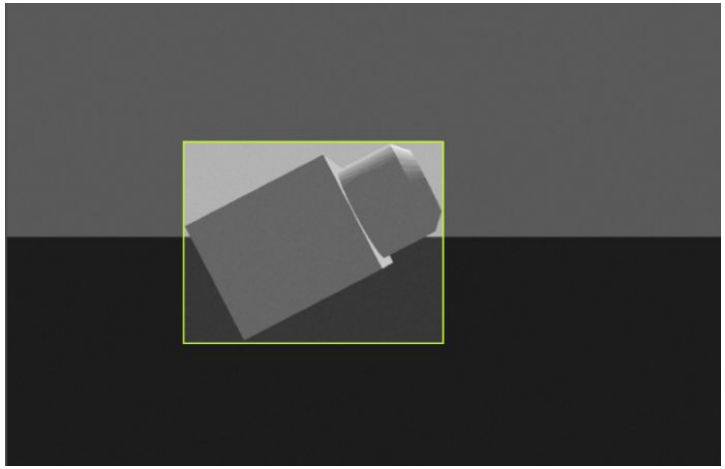


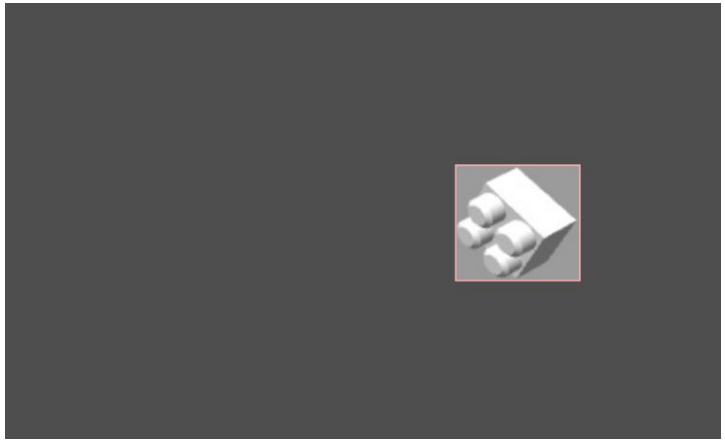
Detect Camera - Problems

- A problem encountered in using Yolo is the way it detects objects, i.e. by drawing a square around them.

  This causes the coordinates provided by yolo to correspond to the center of the detected rectangle and not to the actual center of the object.

  This does not allow to have coordinates with high precision as the output of this system.





Detect Camera - Problems

- Another problem is the accuracy with which yolo detects object classes.

  This problem is due to the training of the software and in particular to the dataset of images already labeled which is not particularly large and to the low training precision forced by the low power of the computers available to carry out the training.

  This problem has been partially solved by increasing the training dataset up to almost 8000 images which, however, are not sufficient to arrive at an accuracy that is always error-free.

  To further improve the training, the training accuracy and therefore the architecture of the developed model could be increased. However, this requires a lot of computing power.

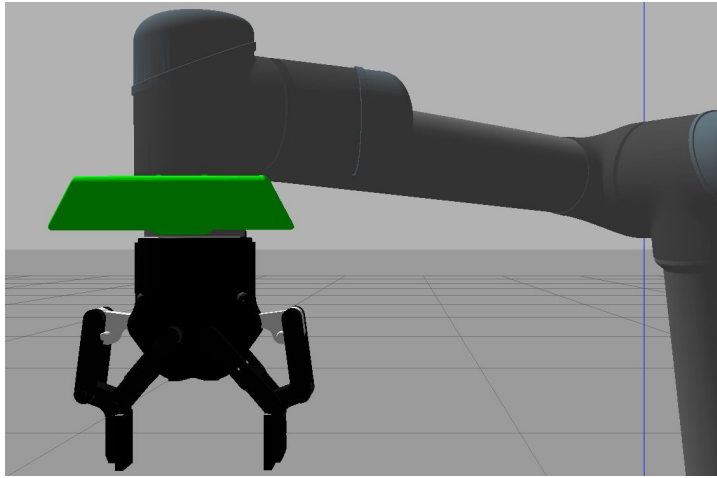
3D Sensor

This module takes care of detecting precise coordinates for a single object and its orientation in space.

In particular it detect if the blocks are straight, on the side and upside down.

The manager, in fact, moves the arm over a piece through the coordinates provided by Yolo at a predefined height and perpendicular to the ground.

We decided to use the grayscale image and not the point cloud provided by the 3D sensor.



3D Sensor - Coordinate calculation

Considering that the position of the piece provided by yolo is not precise it is necessary to calculate it more precisely.

To do this, when the arm is positioned in the approximate position, the 3d sensor calculates the distance in pixels between the center of the image and the center of the detected piece and sends it to the manager.

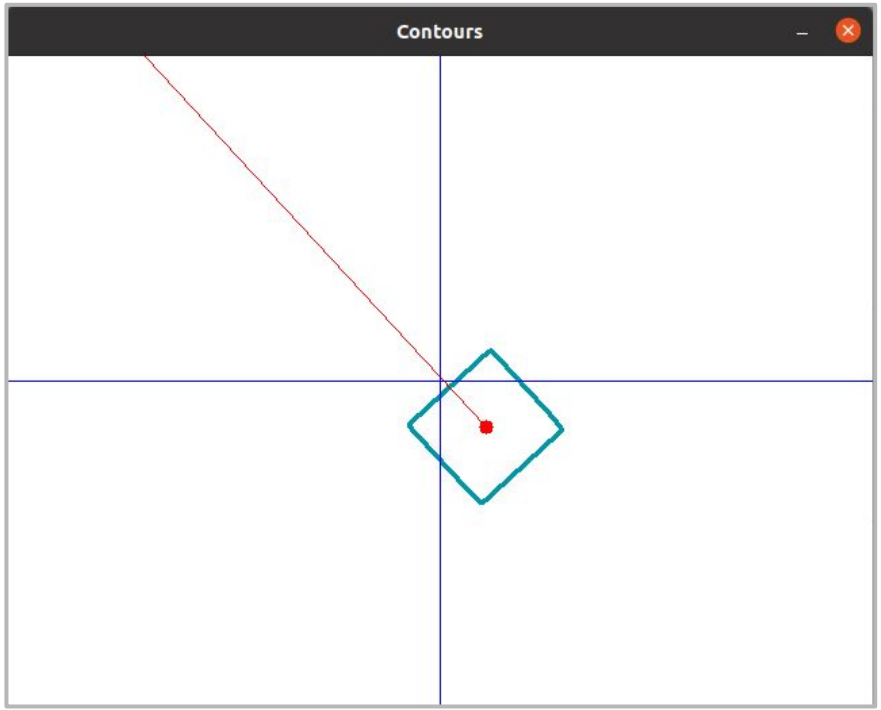
3D Sensor - Angle

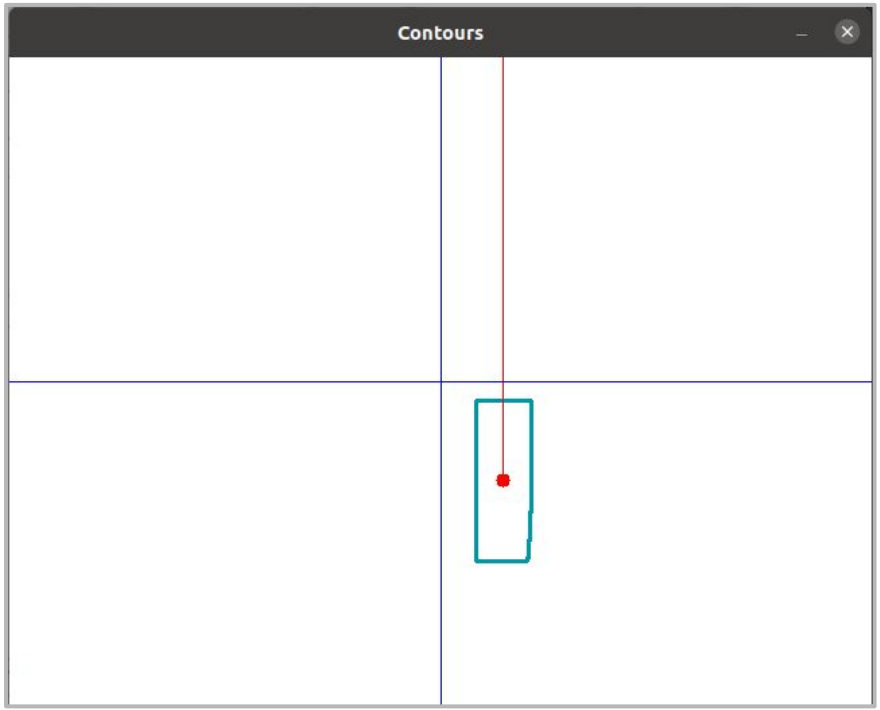
To calculate the angle of rotation of the lego we start by finding the rectangle with the smallest area within which the object can be contained.

The center, the vertices and the sides of this rectangle are then found and thanks to these, knowing the class of the object and the relationships between the sides of the various classes, it is possible to calculate the angle of rotation of the piece.

This angle is given considering the Y axis as the origin.

3D Sensor - Angle



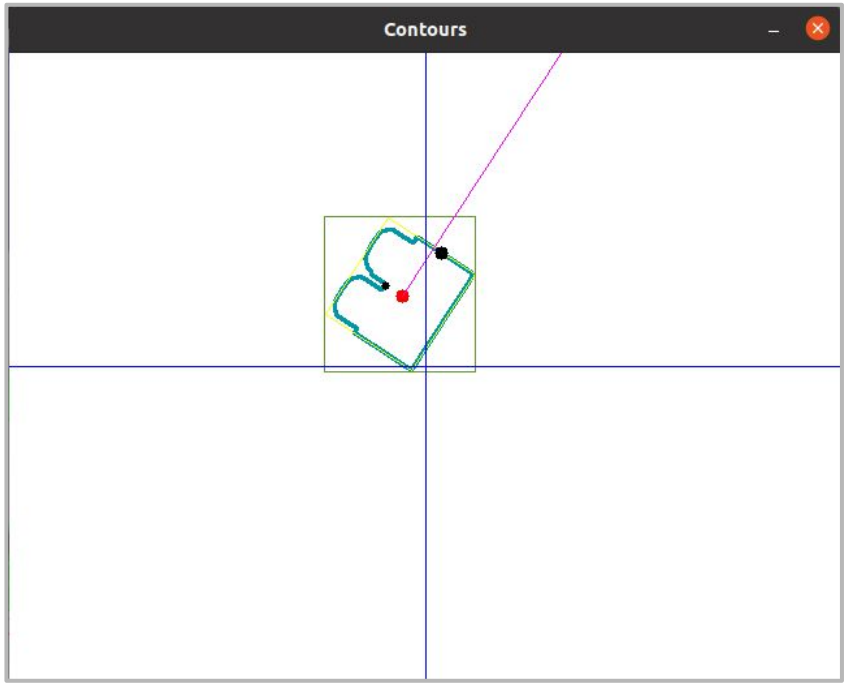


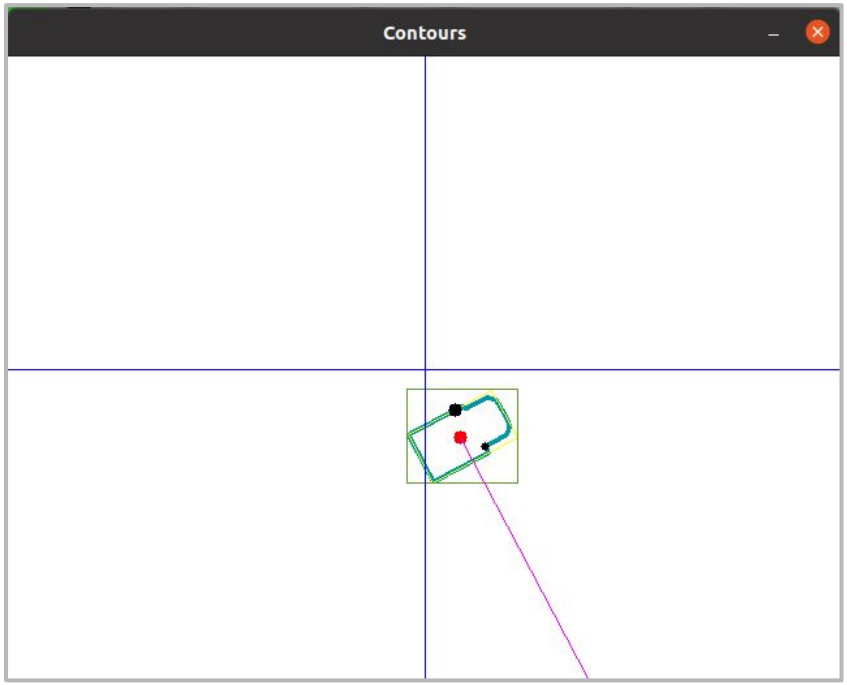
3D Sensor - How to know if a lego is placed on the side?

To calculate if the piece is on the side we calculated if there is at least one convexity point inside the perimeter of the piece. We also compare these points with a threshold to avoid camera distortion issues.

To check the side on which the lego is lying down (left or right) we find the plane to which the the convexity point belongs, i.e. is over or under the rotation axis.

3D Sensor - How to know if a lego is placed on the side?





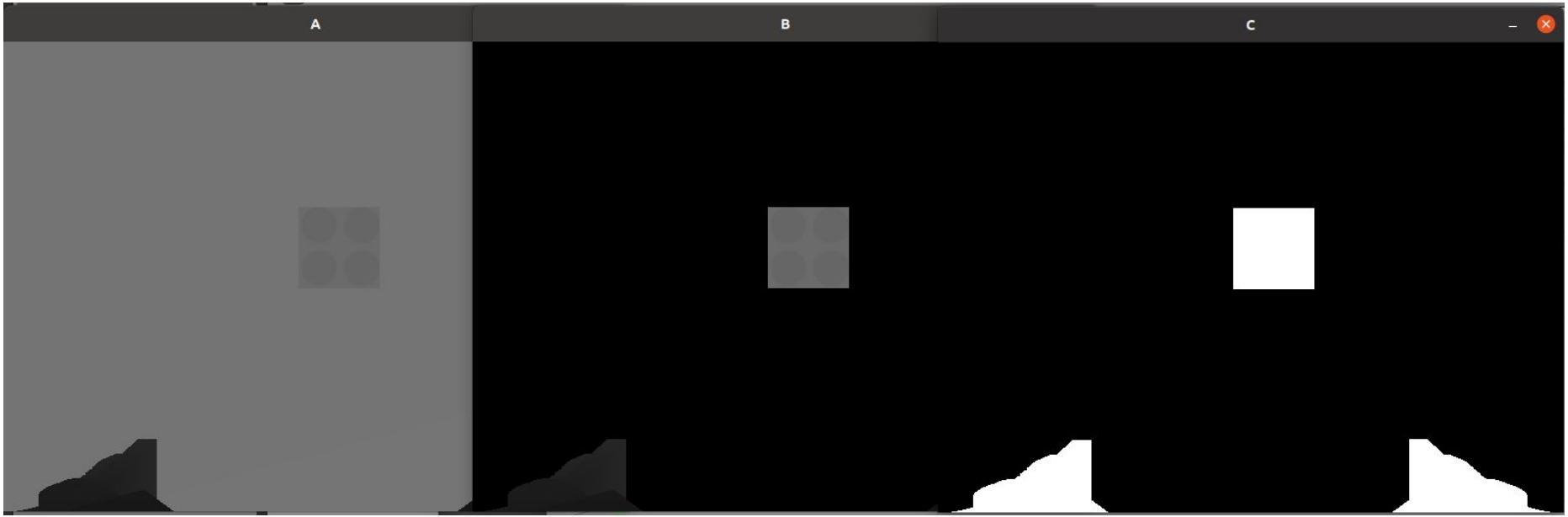
3D Sensor - Lego detection straight and upside down

This procedure detects whether the piece is straight or upside down.

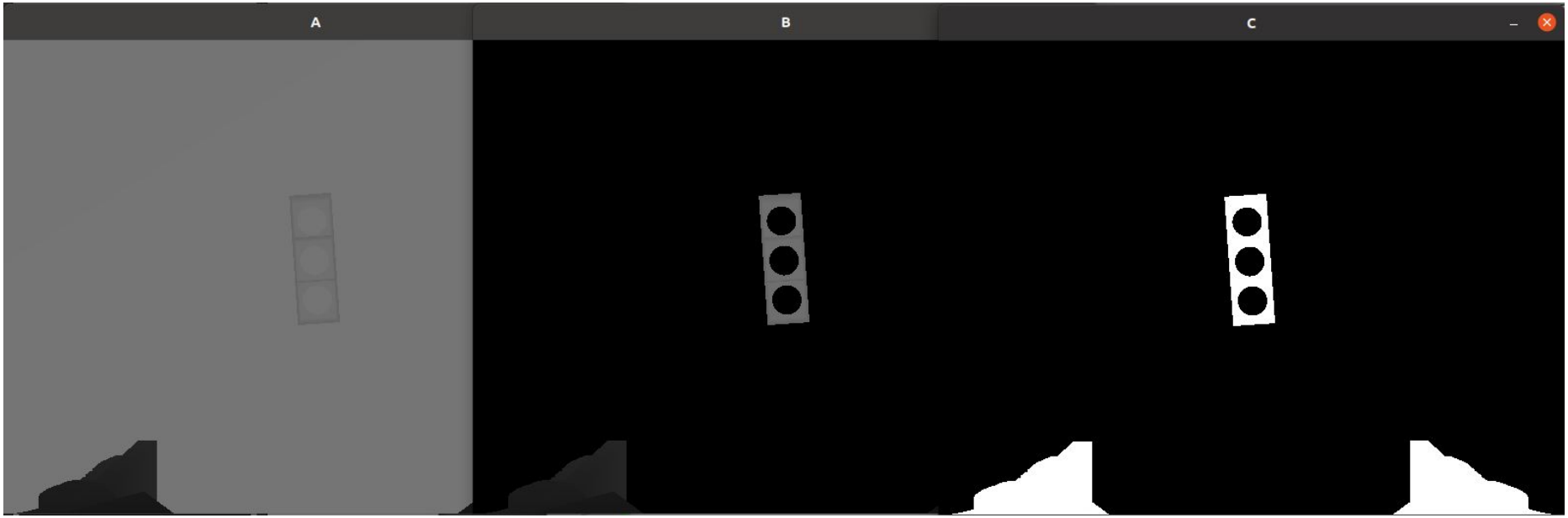
The opencv function is then used to detect the contours and if at least one child contour is detected inside the main one (parent), then the object is upside down.

If this condition is not true that means that the object is straight.

3D Sensor - Lego detection straight and upside down



3D Sensor - Lego detection straight and upside down



3D Sensor - Problems

One of the problems with the 3D camera concerns its positioning over the gripper which causes it to be included within the image.

To solve this problem it was decided to mask it with a "black rectangle". Black in the various procedures is used as the background color and therefore represents the working ground and this eliminates the problem.

3D Sensor - Problems

- Another problem detected is the case in which the robotic arm does not position itself perfectly parallel to the ground.

  This generates incorrect depth readings and not all the threshold is detected as such and this creates problems with the procedures described above and with the detection of the center and the contour of the piece itself.

  To solve this, at least if the error of the arm is limited, a threshold value has been entered that leaves a certain margin with respect to the measured distance of the work plane and this increases the points accepted as background.

Robot module

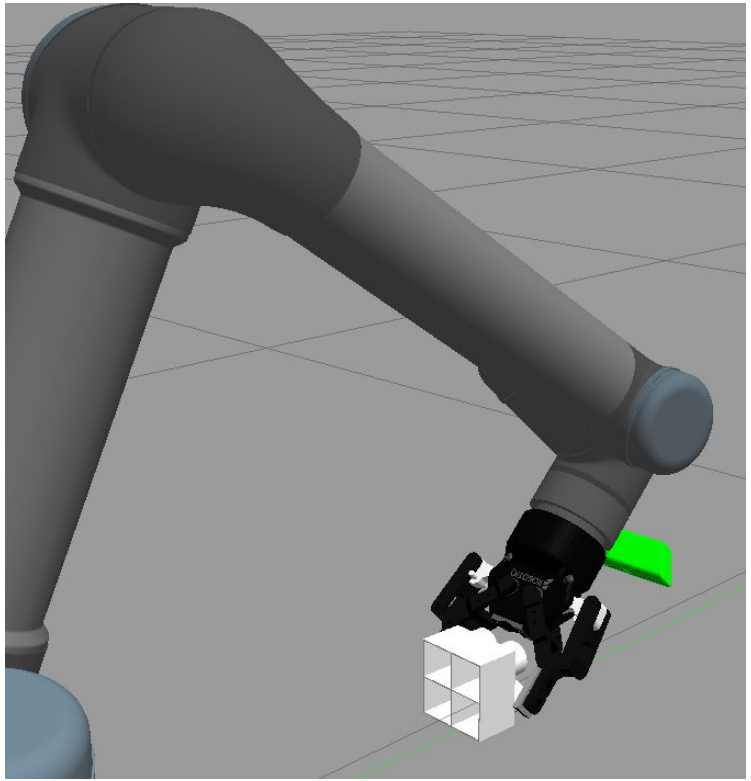
This project module deals with all robot movements for pick and place operations.

Once the position, rotation, class of the object and it’s orientation have been obtained from the manager, the program calculates the various positions of the robot joints in order to take the piece and turn it if necessary and then position it in the right place.

These positions are then sent to the robot controller in order to make it move.

Main movements

- Straight object
- Object upside down
- Object on the side
  - long side
  - short side



Robot - Motion of the robot point to point

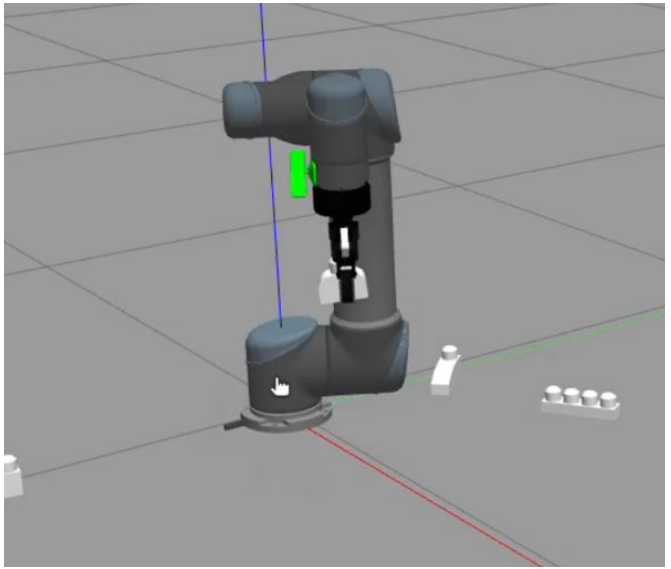
This function takes care of the actual movement of the robot. Starting from the initial and final point of the robot, in fact, through inverse kinematics it calculates the initial and final positions of the joints and subsequently calculates the intermediate points for each joint.

These intermediate points are calculated through a 3 degree polynomial (used to minimize energy) by imposing positions and speed of the initial and final joints.

Having the path it is possible to send to the robot one by one the position of the joints to realize the movement.

Robot - Angle calculation

To have a constant speed in the point-to-point movement we calculated the 2d vectors corresponding to the starting and ending positions. We then calculated the angle between these two vectors and used it to calculate the duration of the motion.



Robot - Direct kinematics

This function allows to obtain the position of the robot knowing the position of its joints and is used to calculate the starting point of each movement.

In addition the direct kinematics is used to calculate the error between the real position of the robot and the desired position.

This error is printed on the screen so that the operator can always monitor the accuracy of the arm and it was useful as a performance index to improve the program.

Robot - Inverse kinematics

Through the inverse kinematics it is possible to calculate the position that each of the 6 robot joints must assume in order to allow the robot to reach the desired position.

This function provides up to 8 possible combinations of joints for each robot position. Usually the first is chosen, that is the one with the arm more extended unless this is not available.

in fact, it is possible that one or more of the 8 positions are NAN, i.e. not reachable by the robot due to the degree of freedom of one or more joints.

If all 8 positions are NAN an unreachable position message is returned.

Robot - Problems: float variables

Initially, float variables were used to perform forward and inverse kinematics calculations.

This resulted in positioning errors of more than one centimeter.

This is due to the fact that the calculations used are numerous and complex and therefore the sum of numerous approximations caused this error.

To solve this problem, it was decided to use double variables which, being more precise, allow a smaller approximation for each result. This made it possible to reduce the total error and bring it to a few millimeters.

Robot - Problems: DH parameters

Another problem encountered were the DH parameters of the UR5 robotic arm.

Initially we use parameters of the official tutorials of the same arm model but they turned out to be incorrect.

This problem was solved by changing these parameters with the correct ones and verified through measurements.

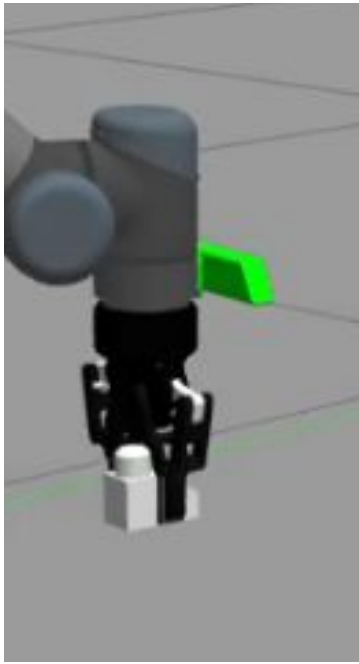
Gripper

This component is an additional module attached to the arm that allows you to pick up and move objects.

Receiving a specific command from the manager, in fact, the end effector closes following a predefined trajectory. During this movement the error between the real position and the desired one is calculated and when this exceeds a certain threshold this means that the piece has been taken and returns the command to the manager who can then move the object.

Through another command from the manager, the gripper detach the object previously taken.

The only commands accepted by the gripper are the opening and closing commands.



Gripper - Problems

Another problem encountered with the gazebo is the closing of the clamp. This operation in fact causes various problems as it may not actually collect the piece or create other types of problems.

To solve this problem, the operation was entrusted to a specific plugin that creates dynamic links between the lego collected and the gripper, always guaranteeing its correct functioning.

Thanks for your attention