Intelligent Robotics Presentation

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Università degli Studi di Padova

Dipartimento di ingegneria dell'informazione

Professor: Emanuele Menegatti

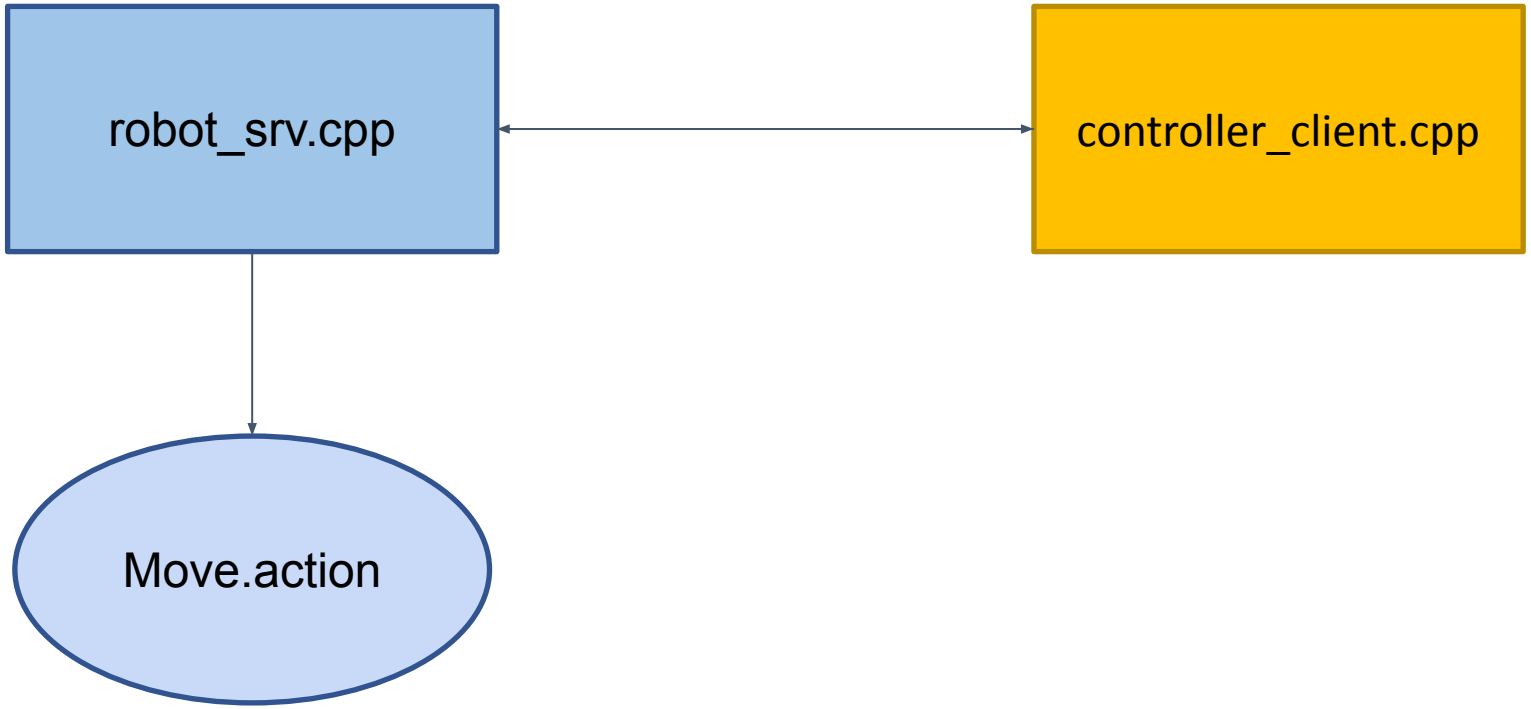
Group members:

Edoardo Bastianello

Stefano Binotto

Gionata Grotto

Assignment 1



Navigation

We used the Move_base action server to manage the navigation.

While the robot perform the navigation it updates the feedback that it’s then read by the client. We implemented three states:

- moving
- stopped
- turning

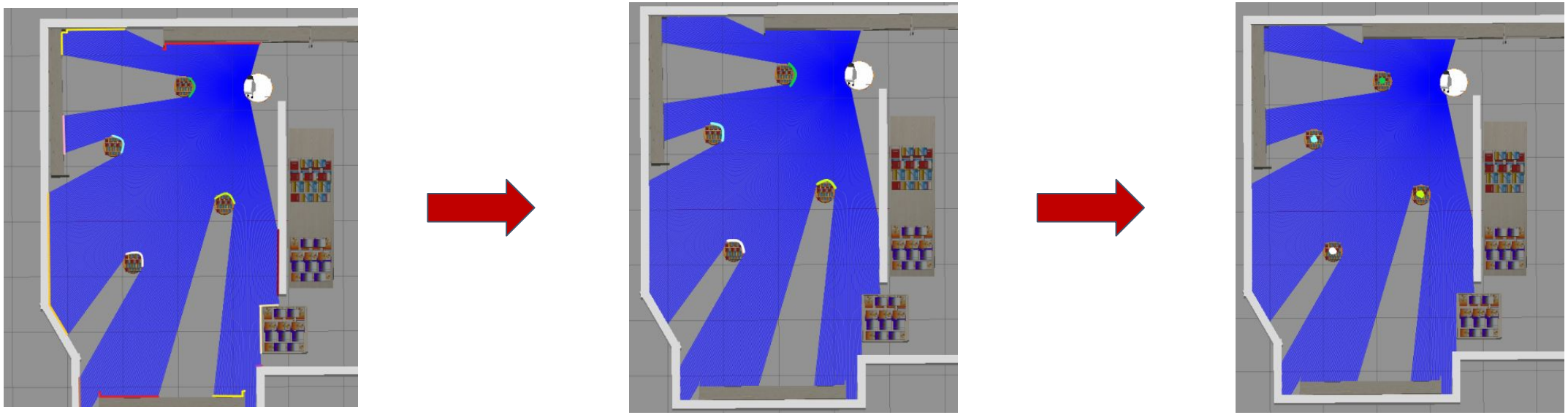
The feedback will be turning if the curve of the trajectory of the robot is greater than 8 degrees.

We also implemented a timeout to check if the robot didn’t manage to get to the final pose.

Detection of movable obstacles

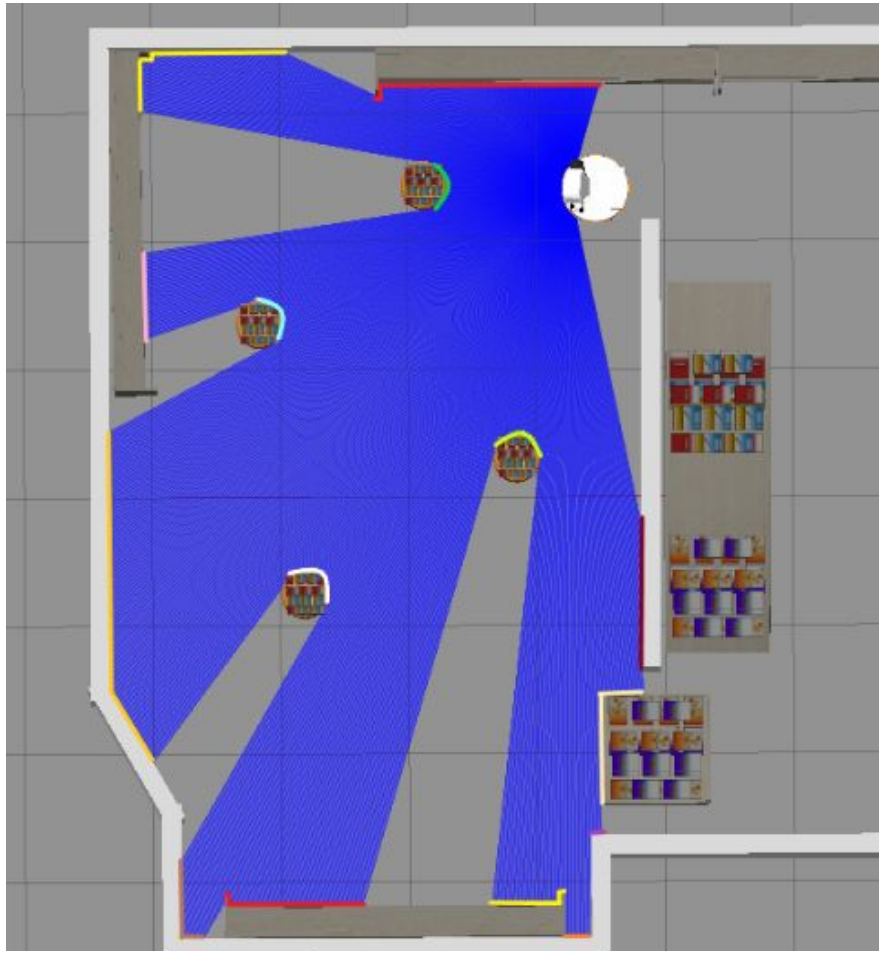
The detection of the movable obstacles is subdivided in 3 parts:

1) Subdivide the detected points in connected sections;
2) Find the most suitable candidates as movable obstacles;
3) Find the center of each detected movable obstacle.



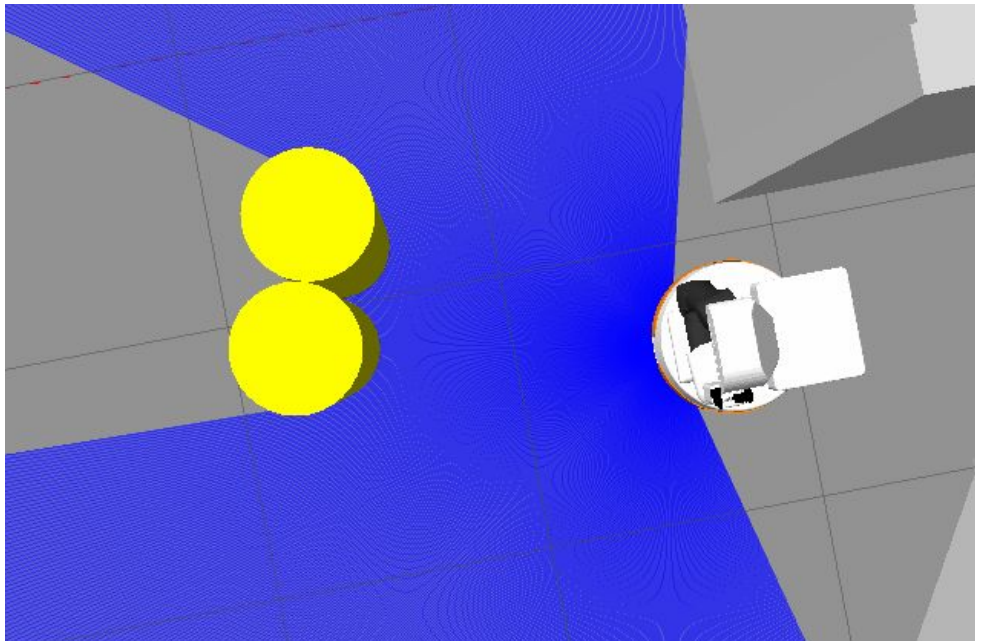
Detection – Subdivide in connected sections (1)

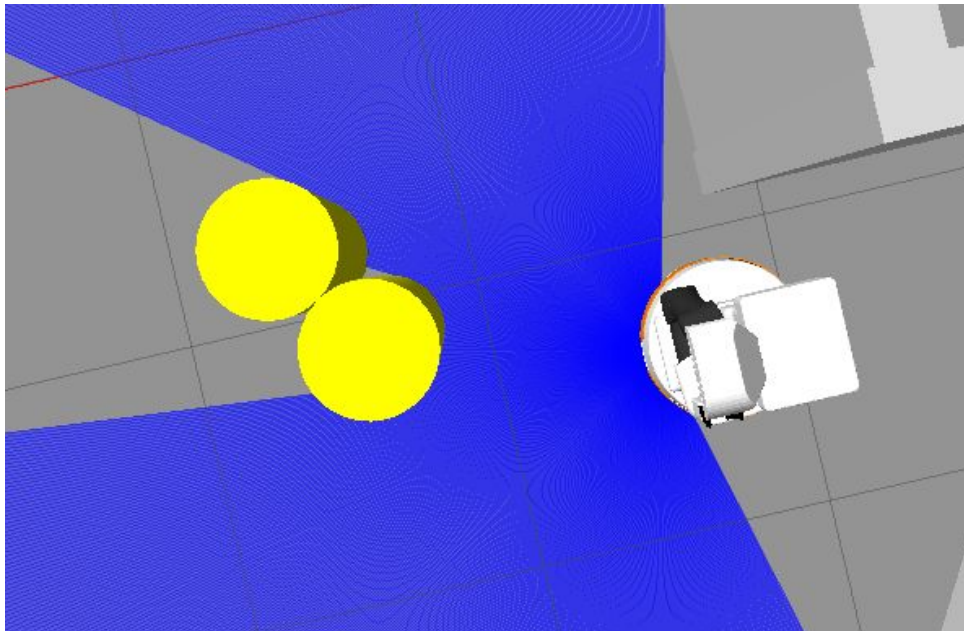
- Get the detected points;
- Discard the first 19 points and the last 18 detected points;
- Subdivide the remaining points in different connected sections
    - **Connected section:** set of consequent detections having distance between them lower than 0.25.



Detection – Subdivide in connected sections (2)

Assumption: the movable objects aren’t attached to each other.





Detection – Compute centers of circles

Compute the centers of the obstacles using three points of the specific detected obstacle.

$$(x - h)^2 + (y - k)^2 = r^2$$

We assumed that each section on which this method will be applied is a circle even if is not.

Initially we thought a different approach to compute the solution that, however, worked only if the robot saw all the movable obstacle.

Detection – Ideas

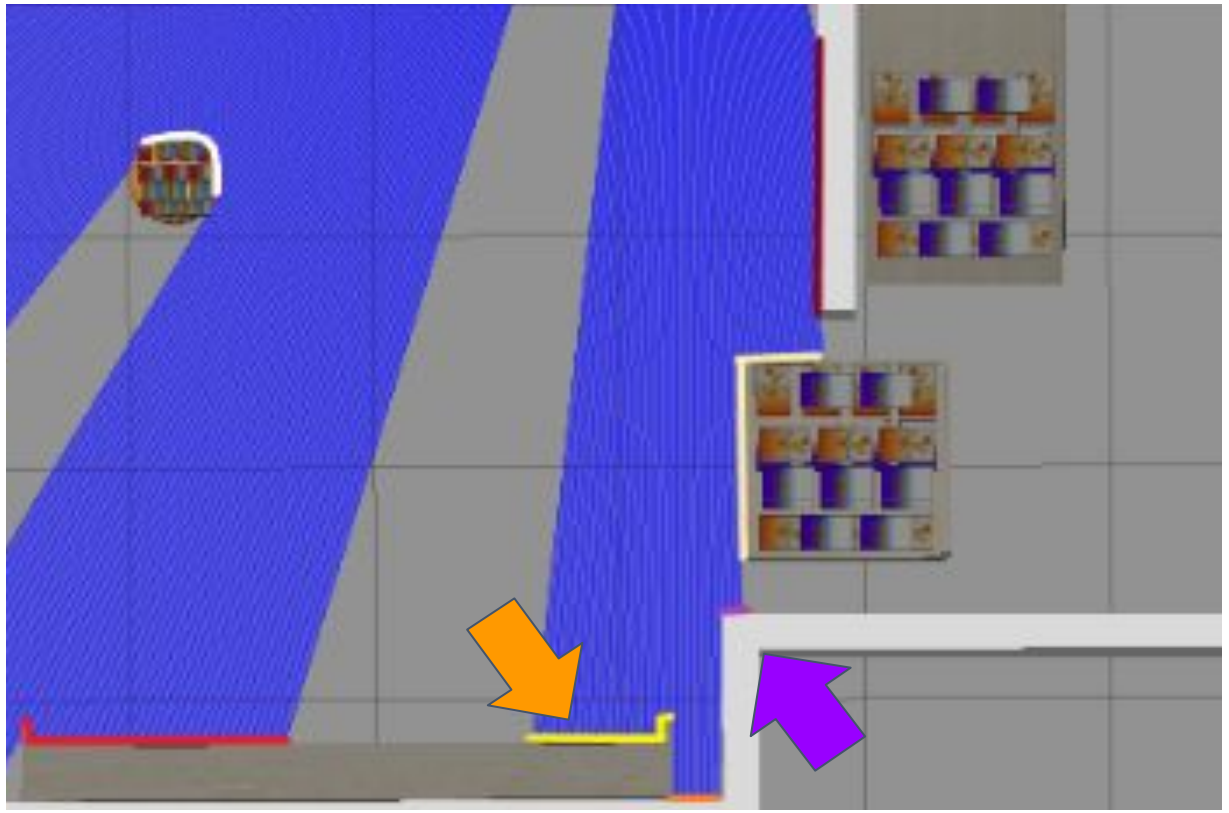
Once we found the connected sections, to understand which of them were movable obstacles we had two main ideas:

- Angular Coefficient approach
- Computer Vision approach

However we found a much simpler and more effective strategy.

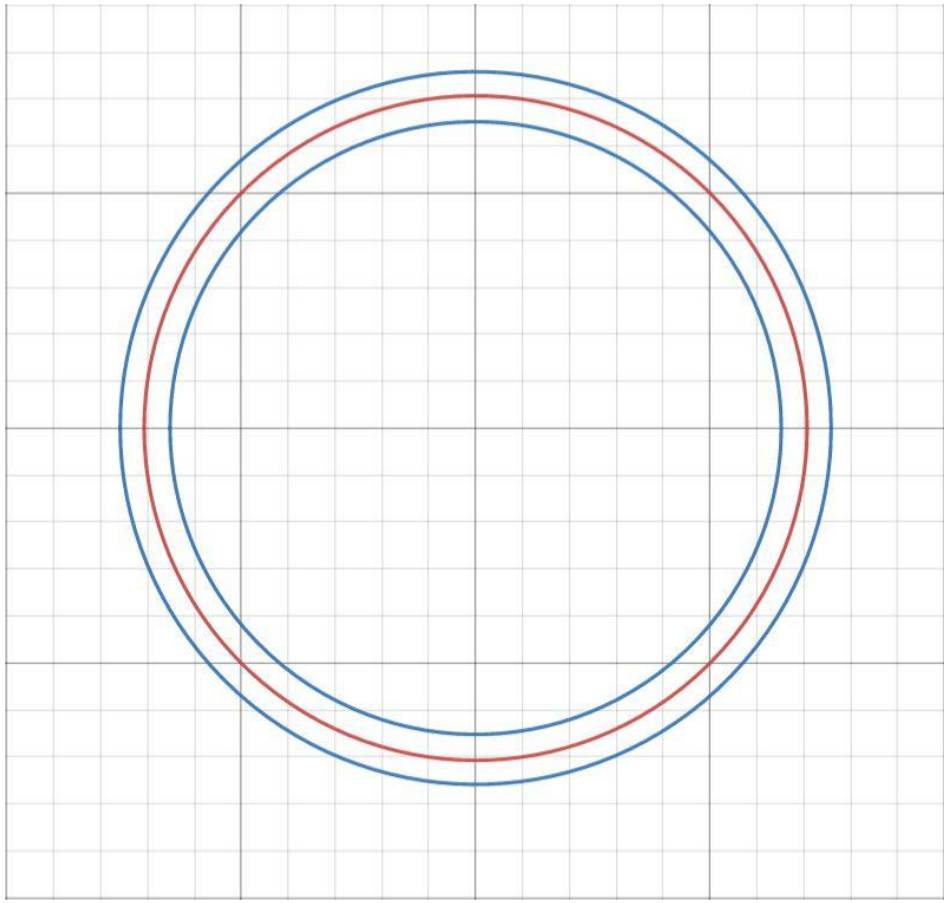
Detection – Find the obstacles (1)

- We cancel out those connected components which does not satisfy our upper and lower limits.
- **Assumption**: we know the diameter of the movable obstacle.



Detection – Find the obstacles (2)

- After computing the center of the hypothetical circle, we check whether each point of the connected component belongs to its circumference.
- **Assumption:** we know the radius of the movable obstacles and we assume that the connected component is a circle.

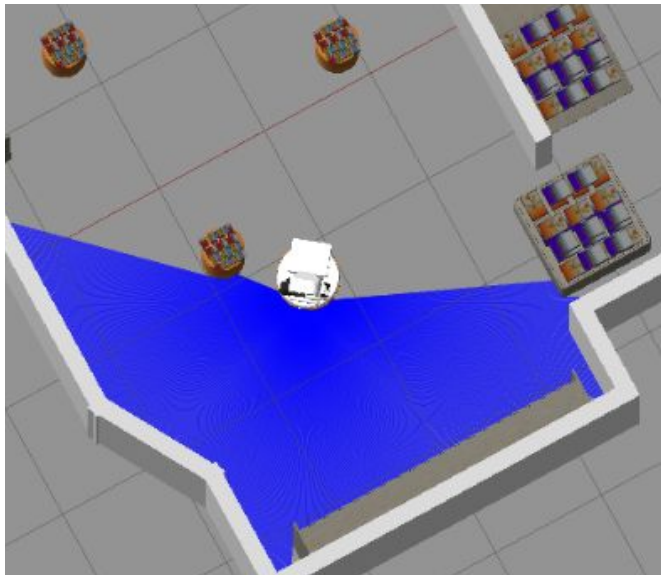


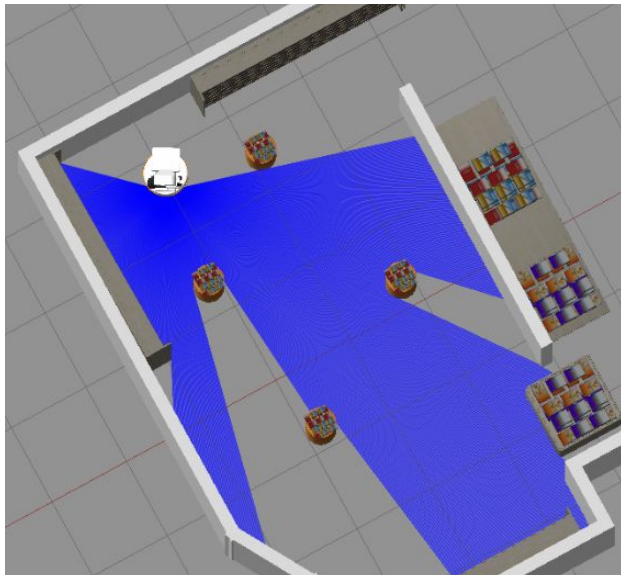
Detection – Borderline Cases

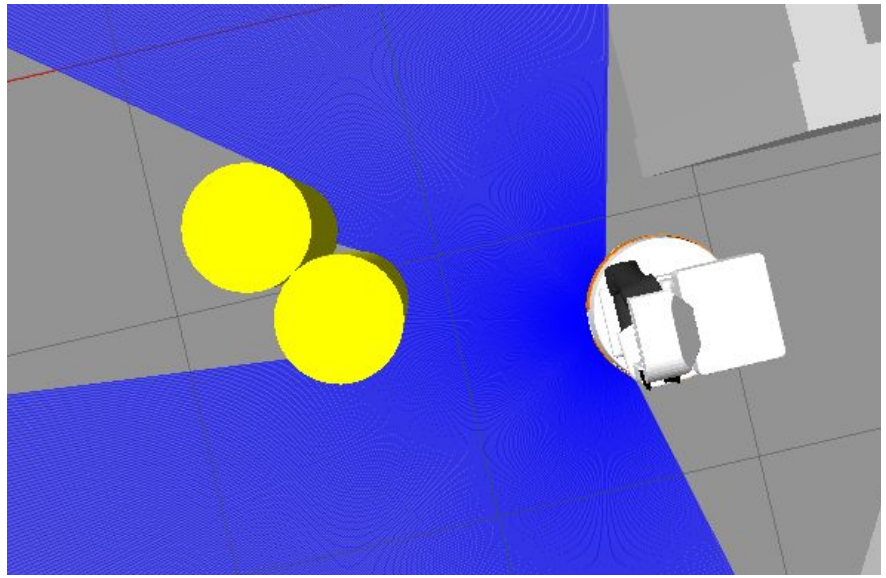
Our method performs generally well in the closed environment provided in the simulation.

In our testing we found:

- there was no false negative.
- there may be a problem if two movable obstacles are attached.

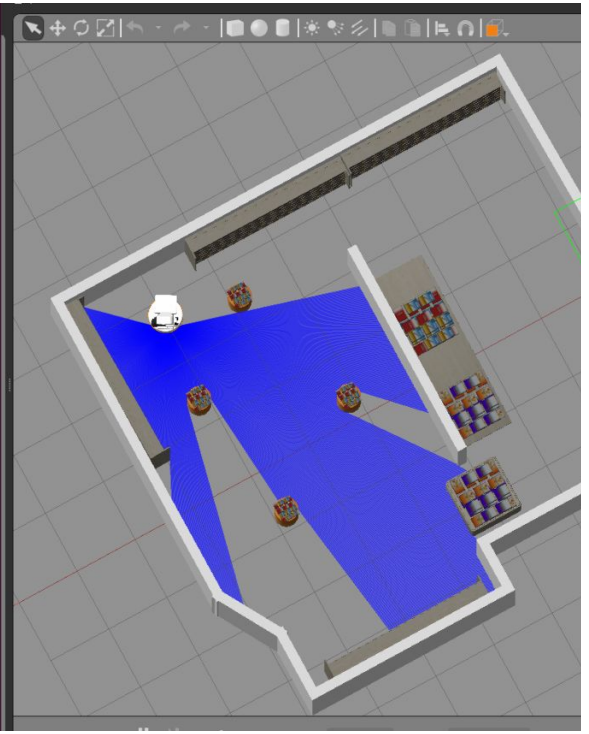




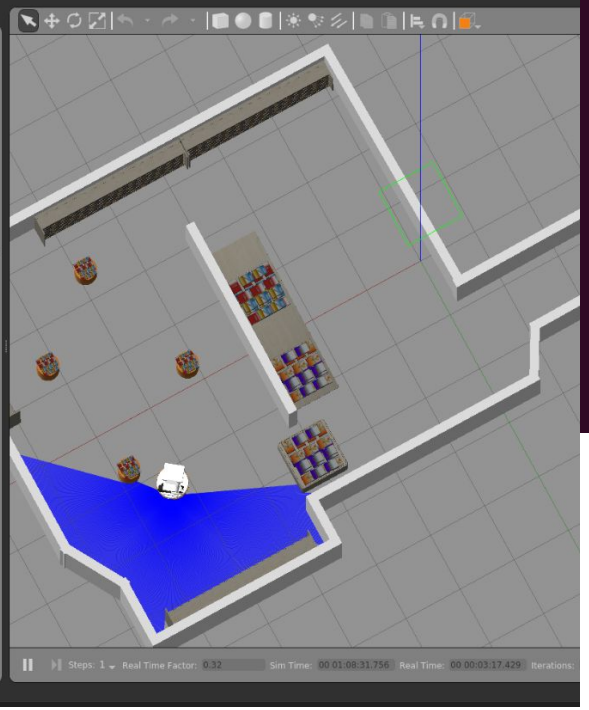


Output examples

```
APTOP-KVVC3LNK: $ rosrun tiago_iaslab_simulation controller_client 12.6 -4.2 0 0 0 0.49 0.88
[1671793948.145038100, 4150.342000000]: Waiting for action server robot_srv to start.
[1671793948.405425700, 4150.445000000]: Action server robot_srv started, sending goal.
[1671793948.405602200, 4150.445000000]: Goal has been sent.
[1671793948.707042200, 4150.565000000]: FEEDBACK STATUS: The robot is moving
[1671793948.944642600, 4150.665000000]: FEEDBACK STATUS: The robot is stopped
[1671793949.212341800, 4150.765000000]: FEEDBACK STATUS: The robot is stopped
[1671793949.212433000, 4150.765000000]: FEEDBACK STATUS: The robot reached the goal
[1671793949.630824900, 4150.913000000]: FEEDBACK STATUS: The robot started the detection of the obstacles
[1671793949.630940500, 4150.913000000]: FEEDBACK STATUS: The robot finished the detection of the obstacles
[1671793950.631273100, 4151.284000000]: MOVABLE OBSTACLE NUMBER [1]: x=1.111971 y=0.538724 z=0.000000
[1671793950.631349900, 4151.284000000]: MOVABLE OBSTACLE NUMBER [2]: x=2.763846 y=1.896144 z=0.000000
[1671793950.631473100, 4151.284000000]: MOVABLE OBSTACLE NUMBER [3]: x=1.026176 y=2.798891 z=0.000000
APTOP-KVVC3LNK: $
```



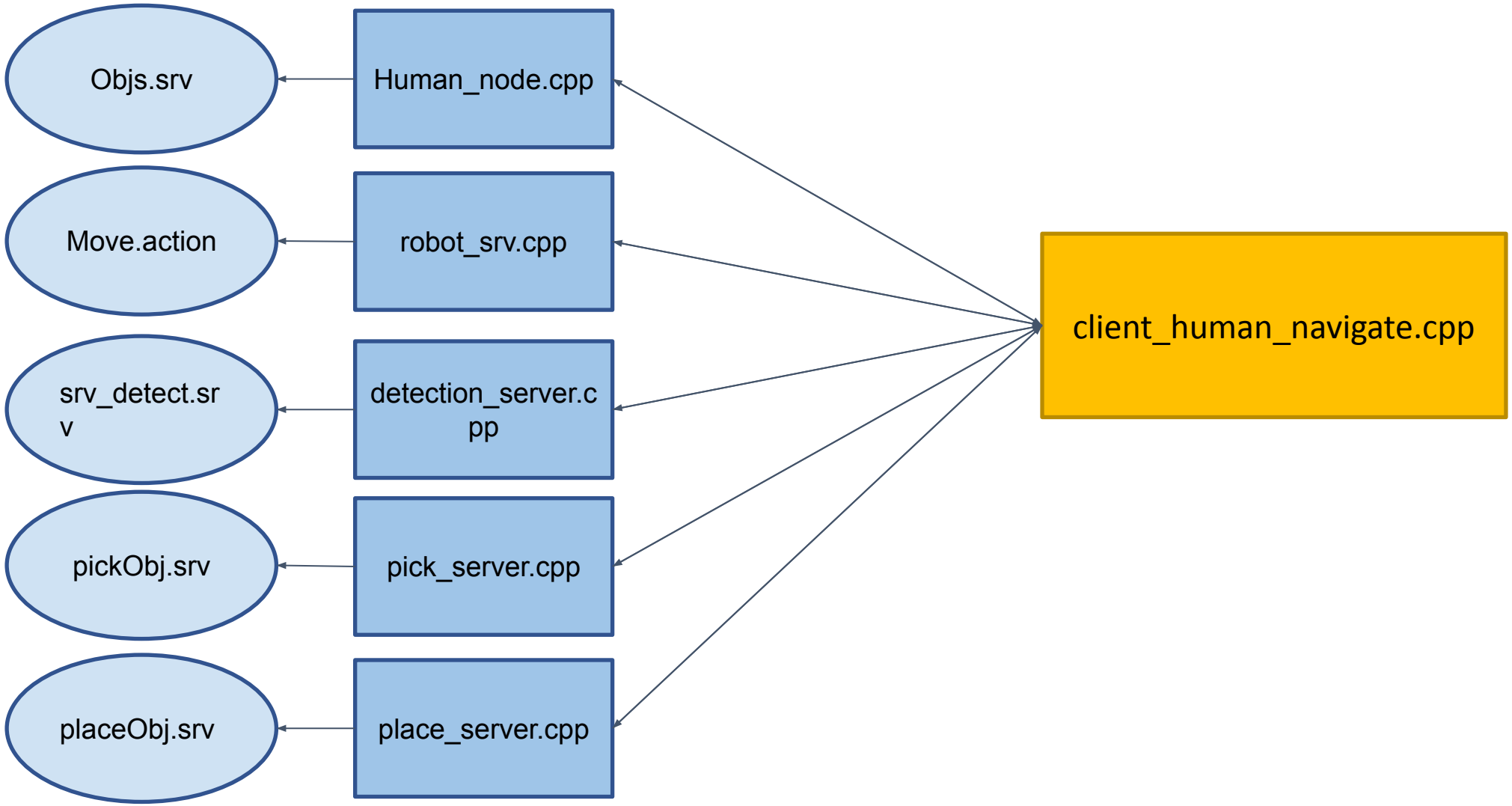
```
] [1671792338.633141600, 4088.113000000]: FEEDBACK STATUS: The robot is turning
] [1671792338.813456200, 4088.213000000]: FEEDBACK STATUS: The robot is turning
] [1671792339.038306500, 4088.313000000]: FEEDBACK STATUS: The robot is turning
] [1671792339.225284900, 4088.412000000]: FEEDBACK STATUS: The robot is turning
] [1671792339.418320100, 4088.511000000]: FEEDBACK STATUS: The robot is turning
] [1671792339.615589400, 4088.614000000]: FEEDBACK STATUS: The robot is turning
] [1671792339.818873200, 4088.713000000]: FEEDBACK STATUS: The robot is turning
] [1671792340.016212900, 4088.813000000]: FEEDBACK STATUS: The robot is turning
] [1671792340.220348300, 4088.911000000]: FEEDBACK STATUS: The robot is turning
] [1671792340.426336500, 4089.012000000]: FEEDBACK STATUS: The robot is turning
] [1671792340.620767200, 4089.111000000]: FEEDBACK STATUS: The robot is turning
] [1671792340.851522900, 4089.211000000]: FEEDBACK STATUS: The robot is turning
] [1671792341.086743200, 4089.315000000]: FEEDBACK STATUS: The robot is turning
] [1671792341.332235100, 4089.411000000]: FEEDBACK STATUS: The robot is turning
] [1671792341.567811900, 4089.512000000]: FEEDBACK STATUS: The robot is turning
] [1671792341.773977000, 4089.612000000]: FEEDBACK STATUS: The robot is turning
] [1671792341.968635500, 4089.712000000]: FEEDBACK STATUS: The robot is turning
] [1671792342.174976400, 4089.811000000]: FEEDBACK STATUS: The robot is turning
] [1671792342.381823800, 4089.913000000]: FEEDBACK STATUS: The robot is turning
] [1671792342.573207100, 4090.012000000]: FEEDBACK STATUS: The robot is turning
] [1671792342.781519700, 4090.112000000]: FEEDBACK STATUS: The robot is turning
] [1671792342.973298200, 4090.212000000]: FEEDBACK STATUS: The robot is turning
] [1671792343.168718000, 4090.310000000]: FEEDBACK STATUS: The robot is stopped
] [1671792343.365577100, 4090.410000000]: FEEDBACK STATUS: The robot is stopped
] [1671792344.365649900, 4090.919000000]: FEEDBACK STATUS: The robot reached the goal
] [1671792344.698694200, 4091.075000000]: FEEDBACK STATUS: The robot started the detection of the obstacles
] [1671792344.698825200, 4091.075000000]: MOVABLE OBSTACLE NUMBER [1]: x=-0.273587 y=-0.728643 z=0.000000
@LAPTOP-KVVC3LNK: $
```



```
                if (!((radius_obj-ERROR_RADIUS<=distance) && (distance<=rad
                    flag = false;
                    break;
                }
            }
            //if flag == true, then it's an movable obstacle
            if (flag) {
                //add current section to "valid" vector
                valid.push_back(section);
            }
        }
    }

    return valid;
}
```

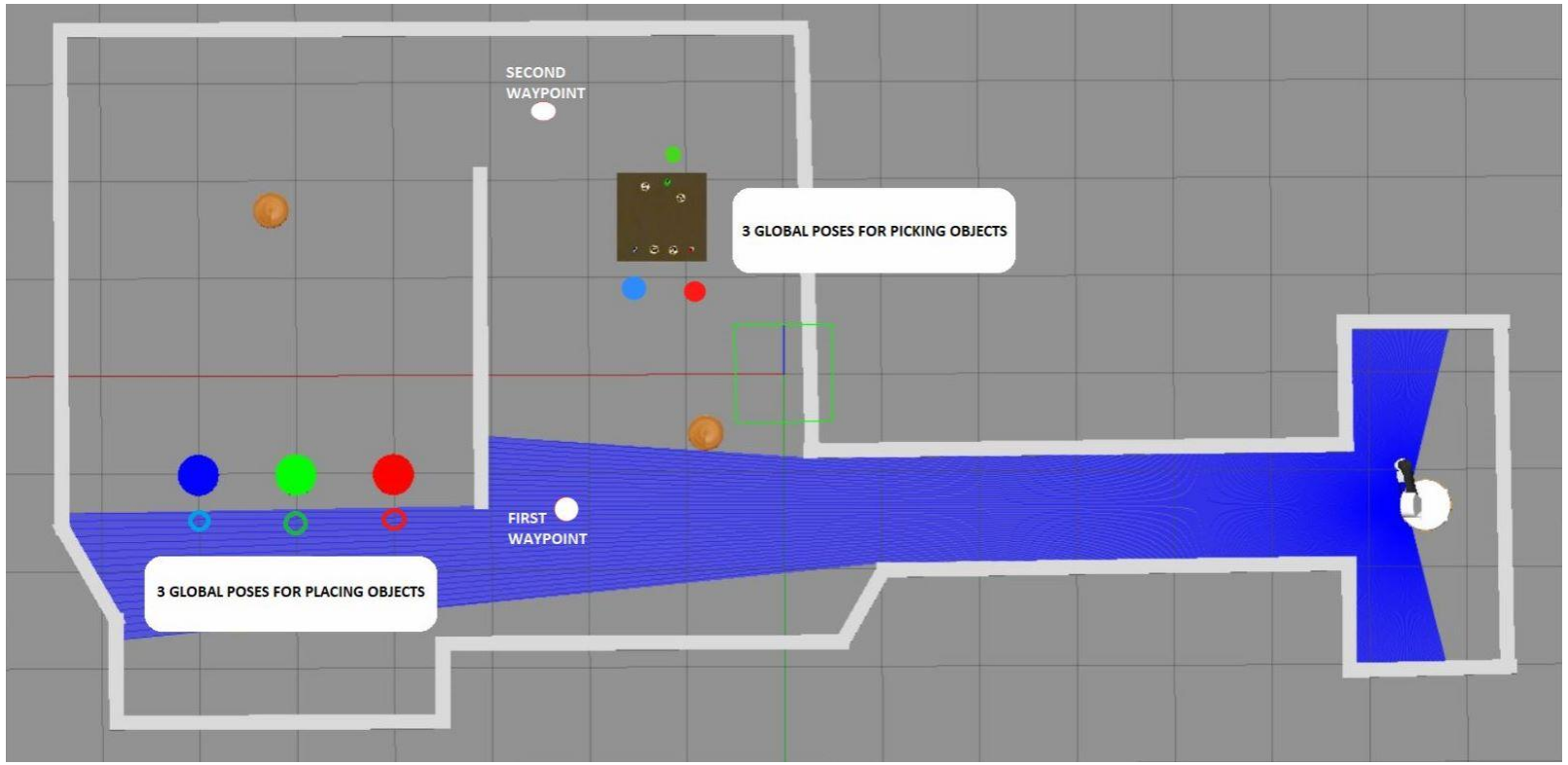
Assignment 2



Navigation - Waypoints (1)

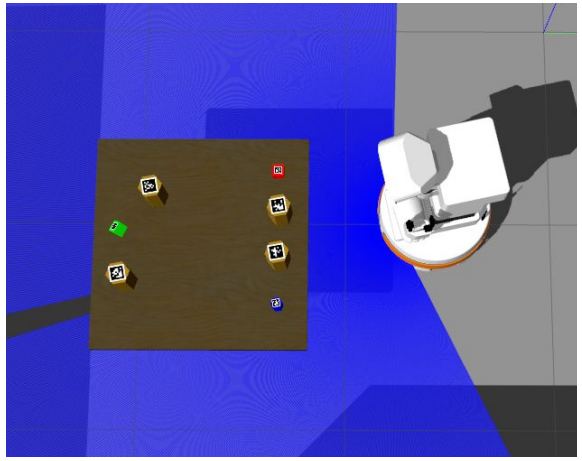
We defined 3 types of waypoints:

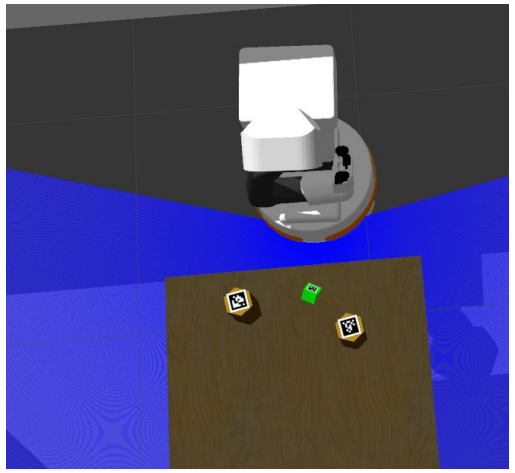
1) 3 global poses for picking objects;
2) 3 global poses for places objects;
3) 2 additional waypoints to avoid getting stuck.

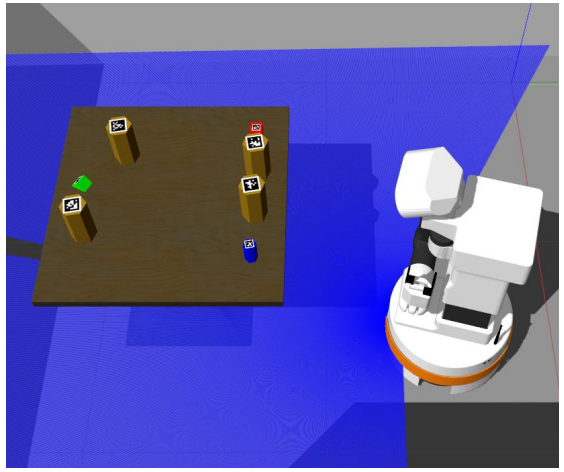


Navigation - Waypoints (2)

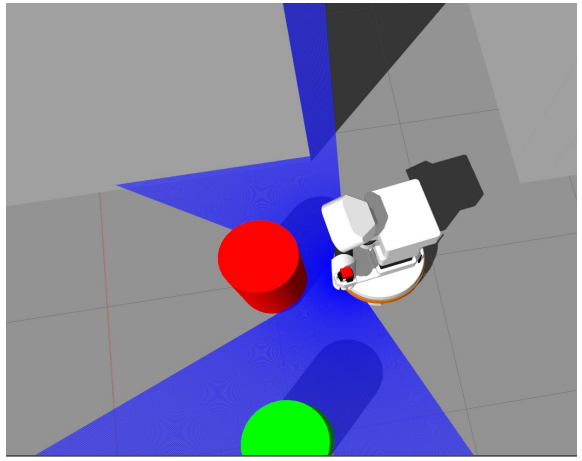
GLOBAL POSES TO PICK THE OBJECT

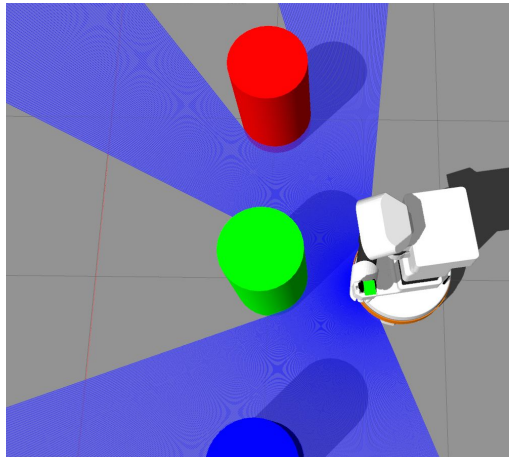


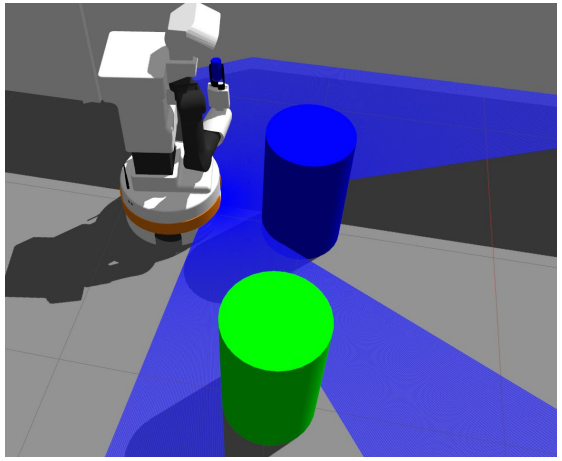




GLOBAL POSES TO PLACE THE OBJECT







Navigation - Sequence

The navigation is managed by following a sequence of waypoint that depends on the object to be picked:

Red or Blue:

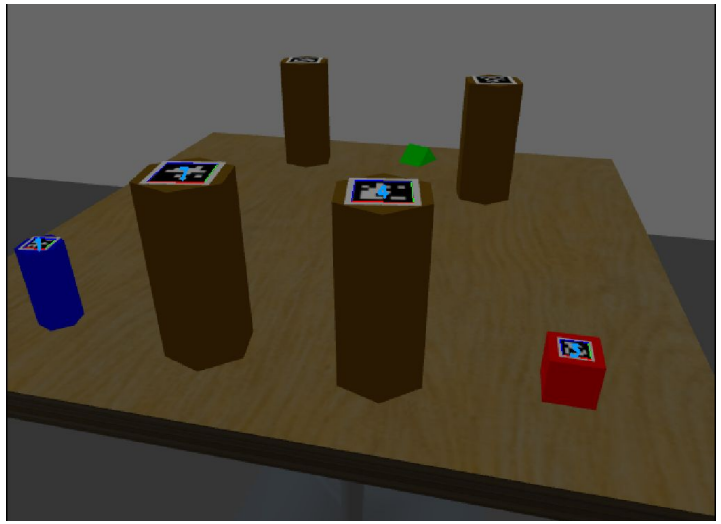
1) first waypoint
2) red/blue global position for picking
3) red/blue global position for placing

Green:

1) first waypoint
2) second waypoint
3) green global position for picking
4) second waypoint
5) first waypoint
6) green global position for placing

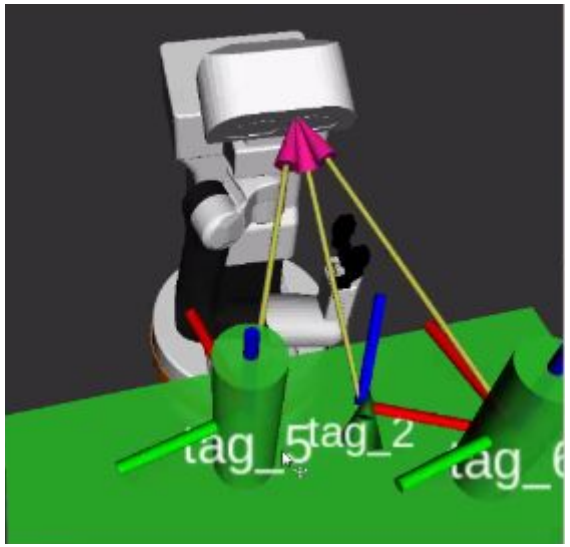
Detection – Camera

In order to maximize the performance of the detection and avoid collision we lowered the orientation of the camera.



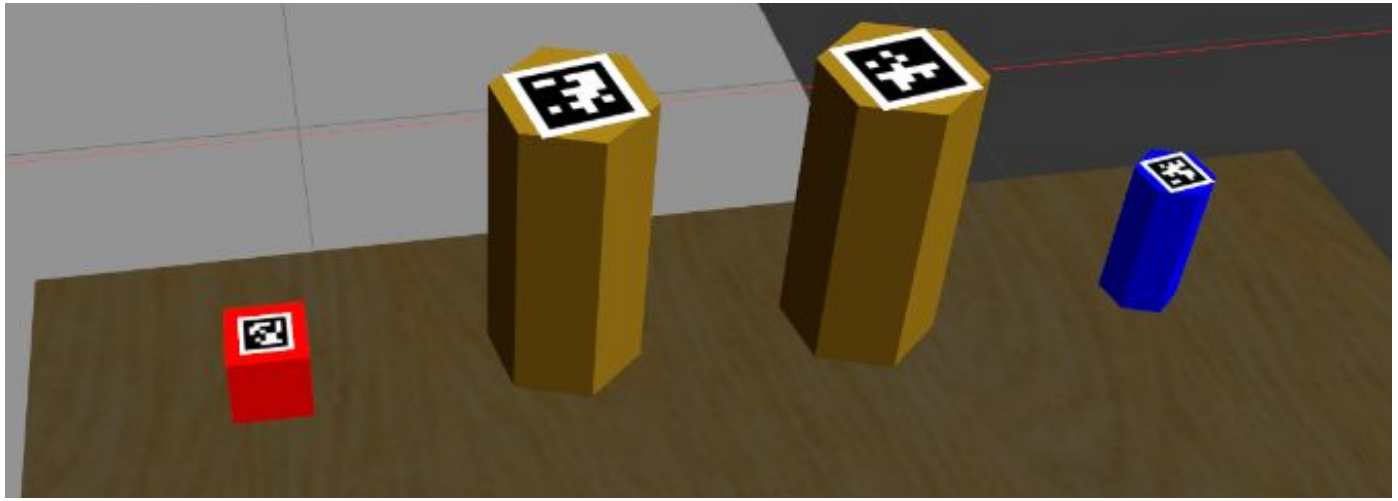
We used the Image topic on rviz to find the orientation that detect the maximum number of objects.

In order to maintain consistency we implemented that the robot have the same configuration while performing the detections.



Detection – Apriltag

We used Apriltag to detect the positions of all the tags that are in the field of vision of the robot. Using the positions of these tags we were able to build the collision objects of all the relevant objects



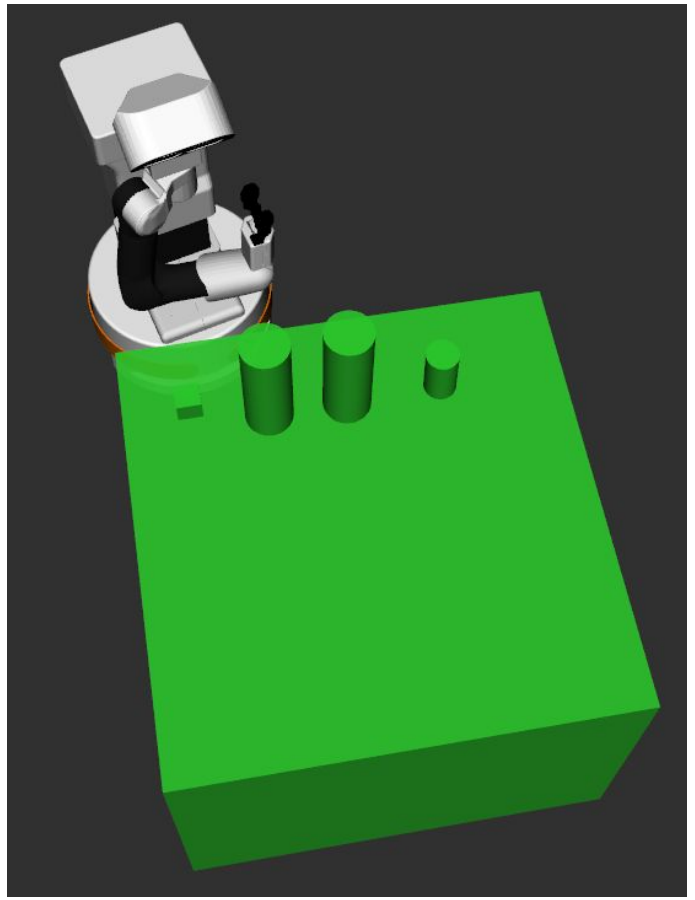
Detection – Create collision objects

In order to create the right collision objects we measured the dimensions of the objects using gazebo. The reference of the object is given by the tag applied on it.

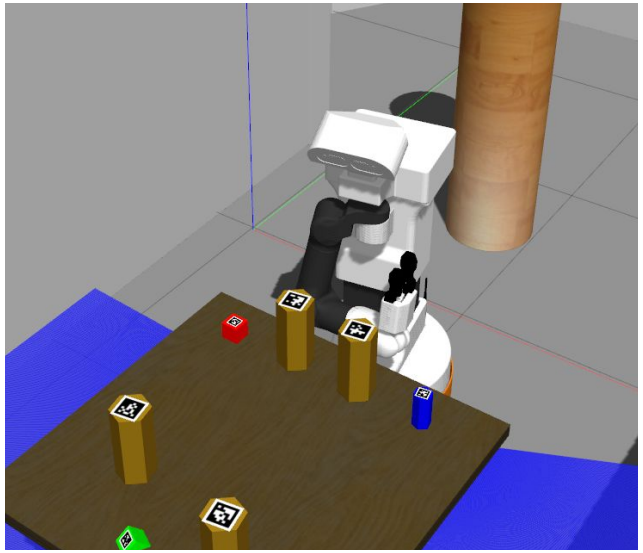
Convert the values from the camera frame to /base_footprint

We enlarged the dimensions of the objects in order to prevent collisions. For every object we did the following actions:

- **cube**: incremented every dimensions
- **triangle**: considered as a cone. Furthemore, we had to rotate its reference because the tag wasn't placed right above it like the other objects
- **hexagon** (target and obstacle): incremented its radius and its height
- **table with objects**: incremented mostly its width. We decided to build the collision objects of the table as a cube in order to avoid collisions during the movement of the arm
- **cylindrical tables**: incremented the height and the radius



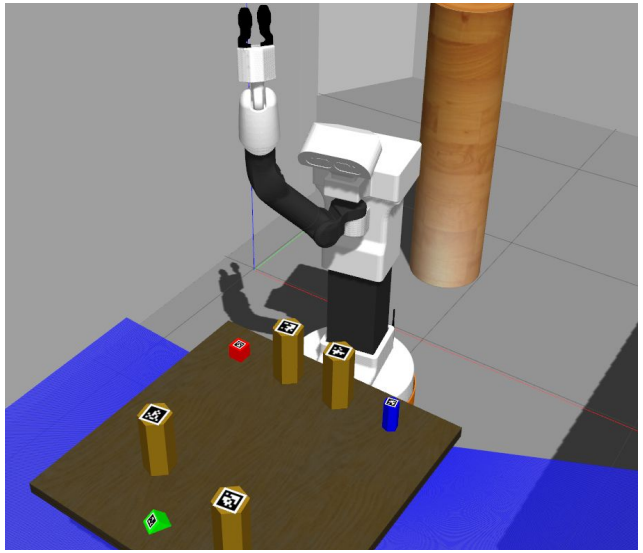
Pick – Robot configurations (1)



SAFE CONFIGURATION:
Specified in the joint space

Goals:

- achieve a safe navigation;
- provide enough room for the object that the robot is holding.



INTERMEDIATE CONFIGURATION:
Specified in the joint space

Goal: minimize the risk of collisions with the table and the objects.

➢ we raised both the torso and the arm of the robot to minimize the amount of horizontal movement in the proximity of the obstacles

Pick – Robot configurations (2)

TARGET CONFIGURATION:

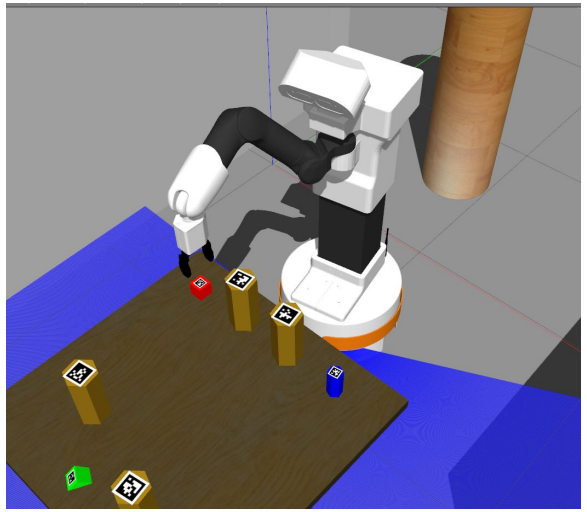
Goal: reach a position for the gripper slightly above the object to pick.

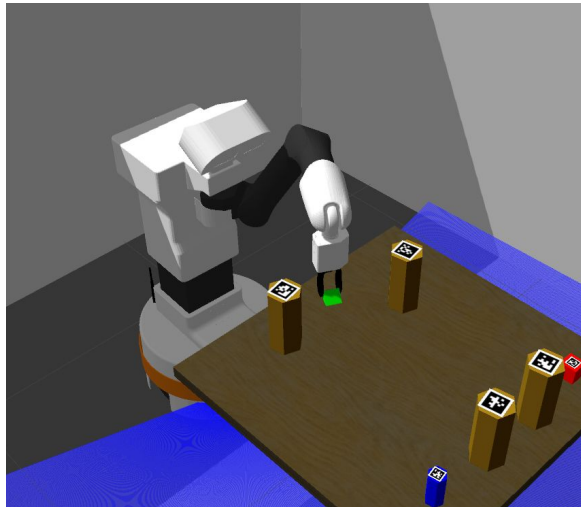
Since from the detection we're able to get the pose of the object

➢ use the cartesian space to directly set a target pose for the gripper instead of setting the value of each joint.

Multiple inverse kinematic solutions:

➢ different executions can result in different values of the joint variables.

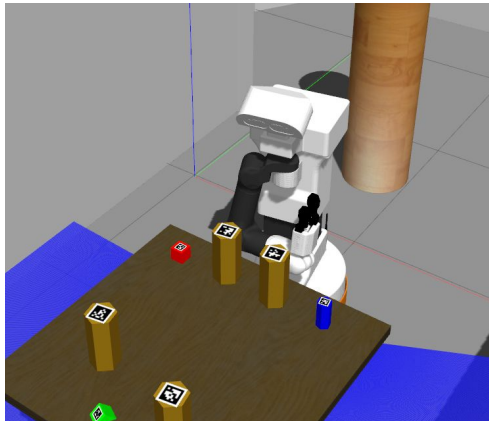




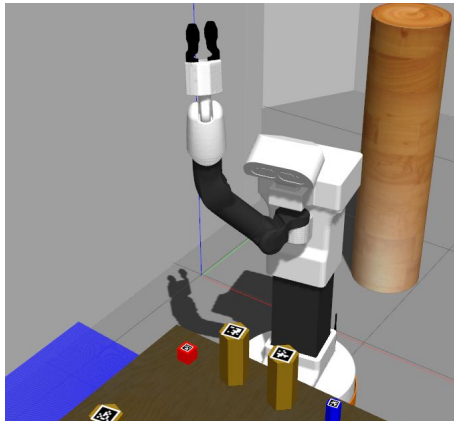
Orientation of the gripper:

- perpendicular to the plane of the table;
- keeps in consideration also the orientation of the object to pick.

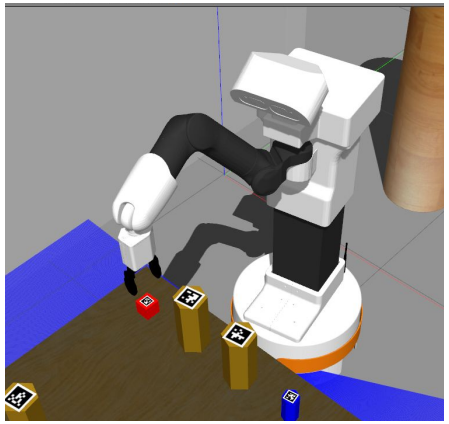
Pick – Pick sequence



SAFE CONF.

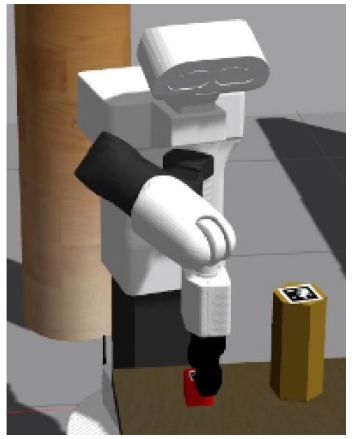


INTERMEDIATE CONF.

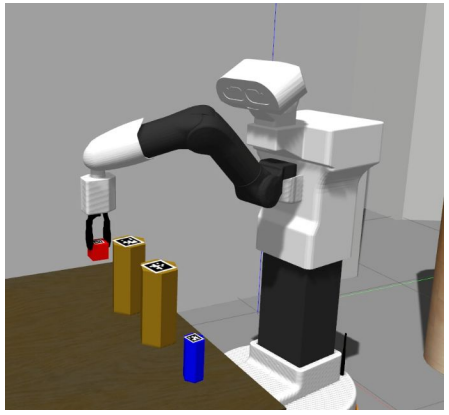


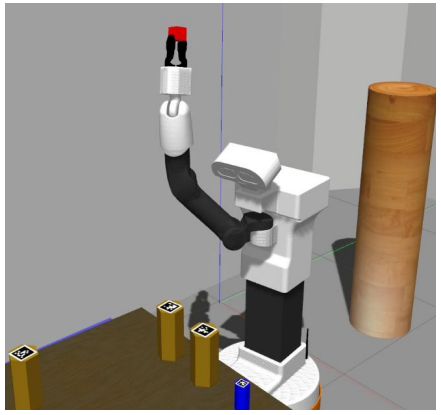
TARGET CONF.

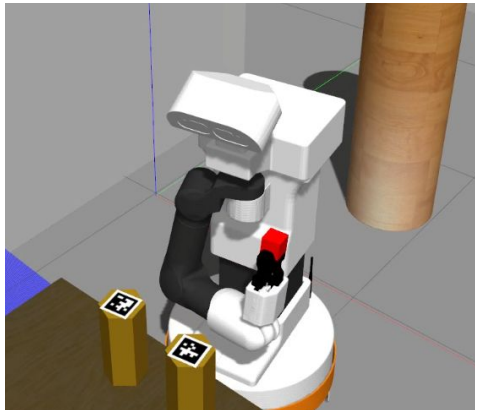
CARTESIAN PATH



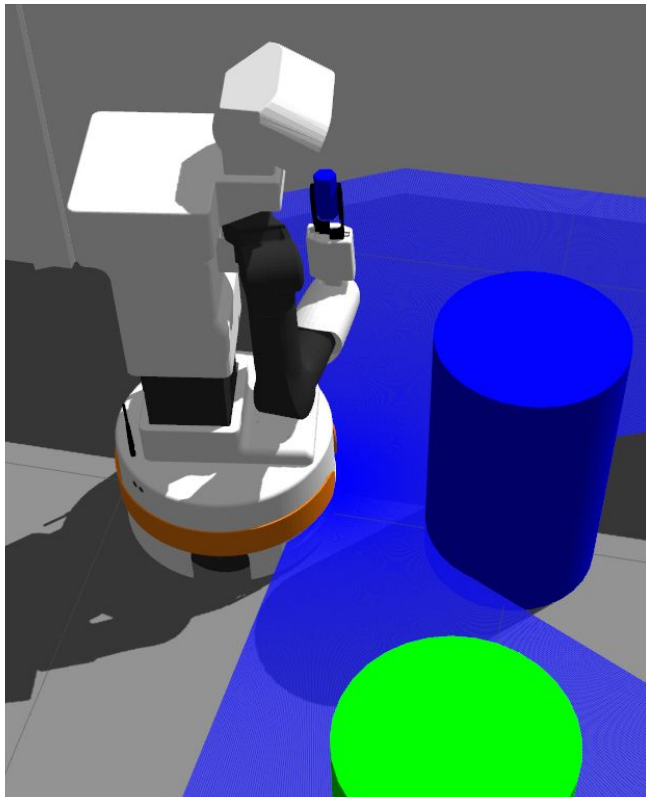
- Remove object from collision objects
- attach links
- close gripper



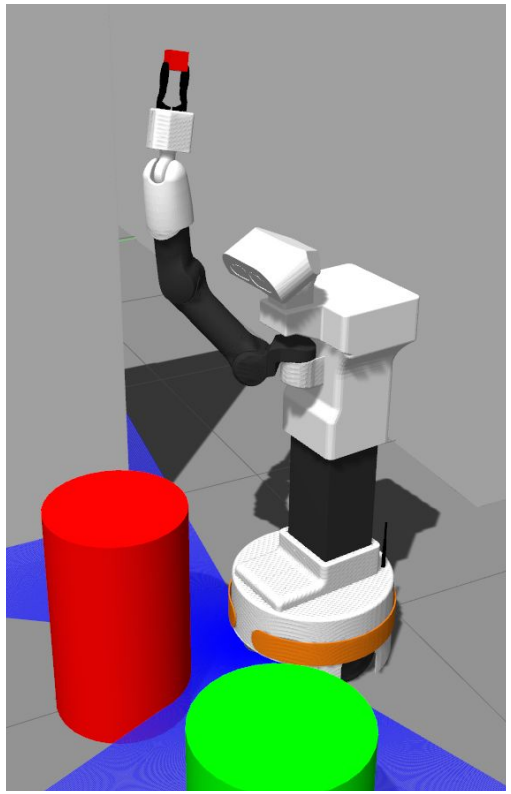




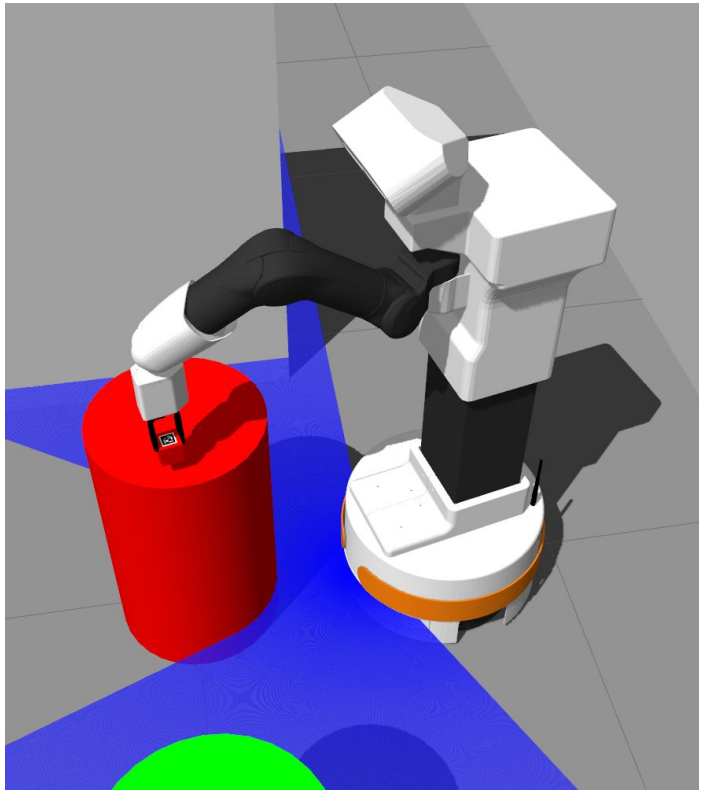
Place – Robot configurations



SAFE CONFIGURATION



INTERMEDIATE CONFIGURATION



TARGET CONFIGURATION

Place

Assumption: We computed the collision objects of the cylindrical tables measuring the dimensions using gazebo and enlarging them in order to avoid collisions.

When placing the object in the tables, we decided to release the blue hexagon from a higher position with respect to the cube and the triangle.

Assumptions & Final considerations

Given that Moveit is a probabilistic framework every execution of our program will perform the trajectories in different ways.

We assumed that the following information is known a priori:

- the pose and the dimensions of every table in the map
- the dimensions of every object (target or obstacle)

These values were obtained using gazebo.

Thanks for your attention !!