GROUP #05

Intelligent Robotics

Francesco Caldivezzi : 2037893
Riccardo Rampon : 2052416
Manuel Barusco : 2053083

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DEGLI STUDI DI PADOVA

Presentation Outline (1)

Presentation Outline (2)

Assignment 1

Packages Organization

The **packages** used for solving the **first Assignment** are :

01 **detection** : Implements the cylindrical tables detection algorithm and the wall detection algorithm

02 **navigation_automatic_ros** : Implements the final execution flow for the no extra-point part of the assignment

03 **navigation_command_ros** : Implements the final execution flow for the extra-point part of the assignment

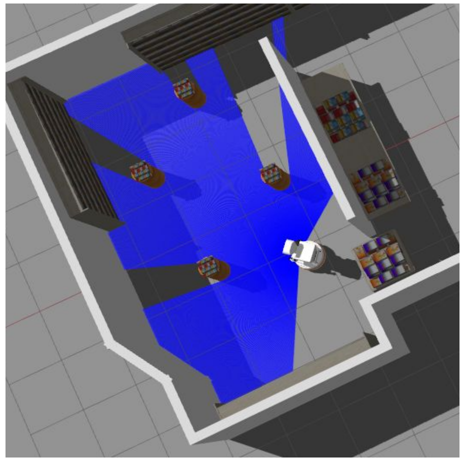
Flow Execution

The **flow execution** of the program that solves the assignment can be summarized as follows :

01 (Only for extra point solution)
Execute the control law strategy to move the robot in a **narrow space**

02 **Move** the robot to the **goal position** (position where detection starts)

03 **Detect** the **centers** of the cylindrical tables



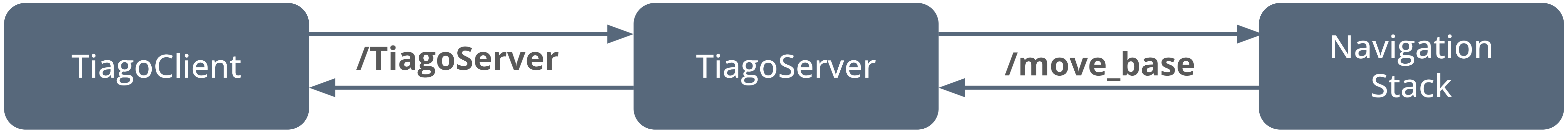
How Navigation Works (1)

How the robot moves depends on which part of the Assignment was solved. In particular :

01 For solving the Assignment **without** considering the **extra point**, we developed :

An action client a.k.a. **TiagoClient** that sends a goal through the action message of the **MoveDetect** action via **/TiagoServer** topic

An action sever a.k.a. **TiagoServer** that, given the goal sent by the client, manages the movement of the robot by forwarding the goal to the navigation stack through the **/move_base** topic, and acting as a proxy to send feedback information to the client



How Navigation Works (2)

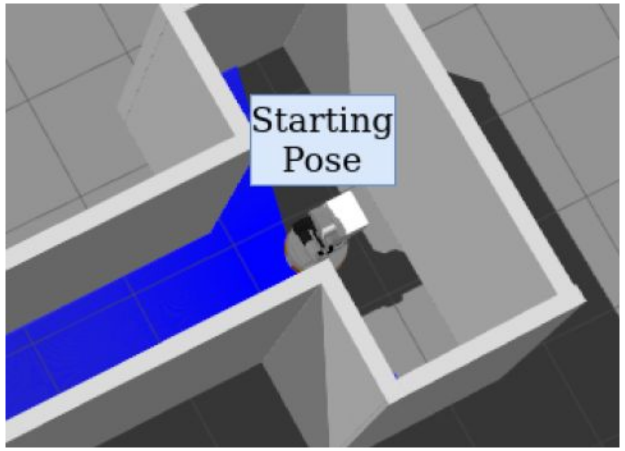
02 For solving the Assignment **considering** the **extra point**, we have :

A **Reused** action client, server and action message developed for the solution without extra point

B **Implemented** a control strategy to move the robot in **narrow spaces**

How to move the robot in narrow passages (1)

In order to move the robot in **narrow passages** we developed a procedure that consists of **detecting** if the robot is **inside a narrow space**, and eventually give to the robot a **"straight"** velocity vector to move. To do so, we **assume** that the robot is place in the following way at the beginning :

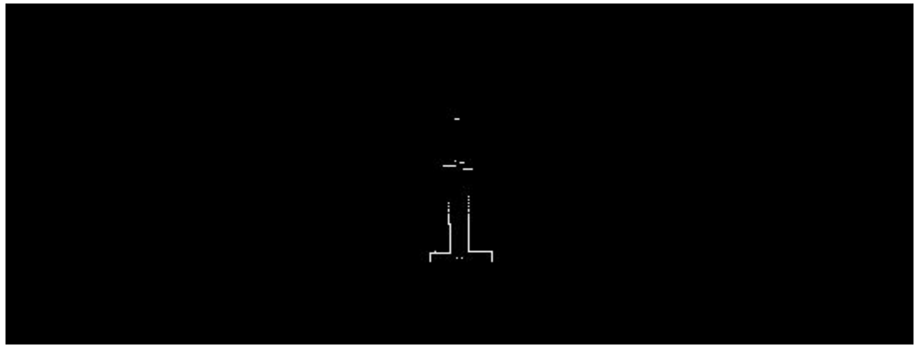


Notice that, in this way, giving a "straight" velocity vector to the robot means that the robot will move through the corridor **without hitting** it.

How to move the robot in narrow passages (2)

To detect if the robot is inside a **narrow passage** or not, we developed an algorithm that consists of the following steps :

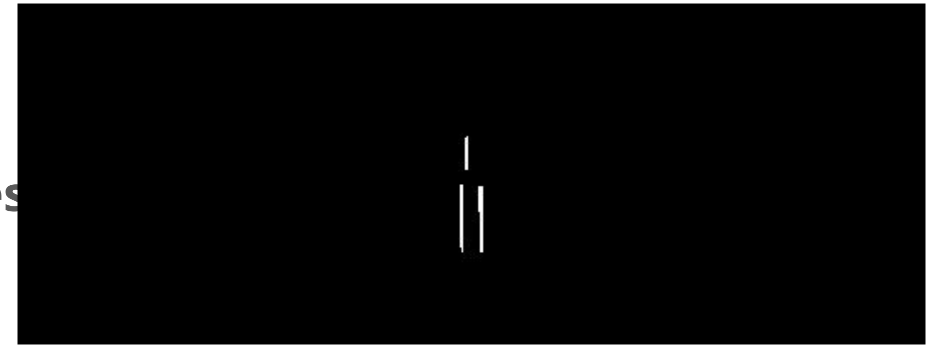
01 Construction of a **binary image** with laser range finder data



02 Given the binary image at the previous steps, to detect **"straight raw lines"** inside we apply to it :

A **Erosion** and **dilation** operators to remove occlusions

B **Hough transform** algorithm to detect the **lines** inside the pre-processed image



How to move the robot in narrow passages (3)

03 In this **final step** we :

A Cluster the lines into **two clusters**

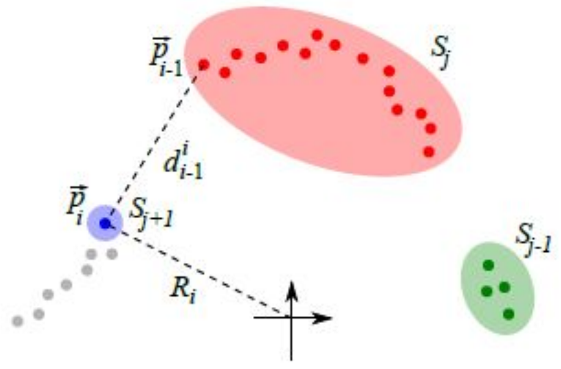
B For each cluster of lines, **merge** the lines of that cluster into a **single line** computed as the average line

Notice that, once we are **not able** to **find** two separate **clusters** of lines, than it means that the robot is **no** more **inside** a **narrow passage**

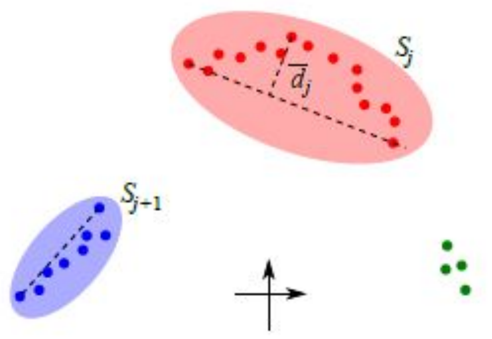
How to detect the centers of the tables (1)

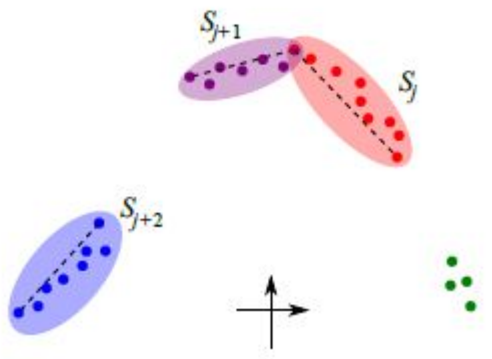
In order to find the **centers of the tables**, we decided to find the **centers of the circles** (w.r.t. **base_laser_link** reference frame), that correspond to them given the **laser range finder** data, by adapting an existing algorithm that works through these following steps :

01 **Grouping points** : provide a collection of subsets that represent possibly separate objects



02 **Splitting points :** each subset with a number of point larger than a threshold is analyzed and, eventually splitted into multiple subsets, computed by trying to fit the points into the best possible line that they represent



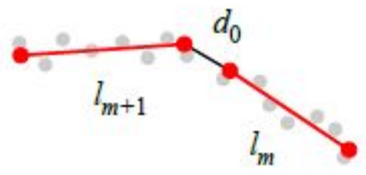


How to detect the centers of the tables (2)

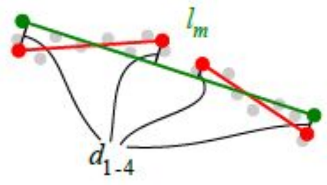
03 **Segmentation** : given a subset of points, this step finds the line that best fits such points

04 **Segments merge** : given the set of segments, in this step we try to merge two or more segments together by using two steps :

A **Connectivity test :** check if two segments are close to each other



B **Spread test :** check if two segments are collinear



How to detect the centers of the tables (3)

05

Circles extraction : in this step, given the set of segments detected previously we apply the following 3 steps :

A

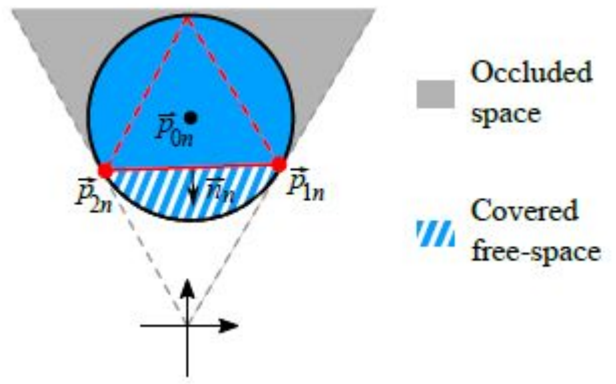
For each segment **build** an **equilateral triangle**, whose center is away from central point

B

Construct the **circumscribed** circle on the basis of the triangle

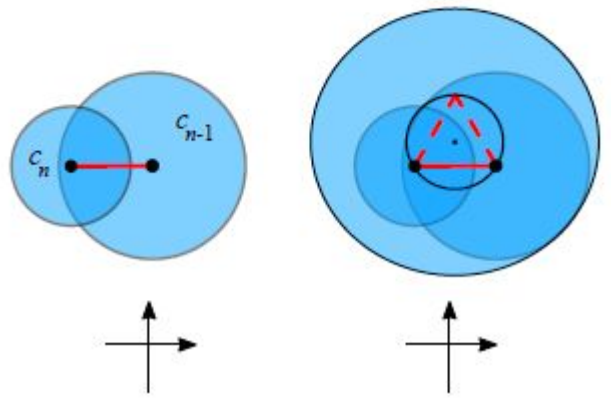
C

Enlarge the circle



06

Circles merging : in this phase two circles that intersects each other are merged.



Assignment 2

Packages Organization

The **packages** used for solving the **second Assignment** are :

01 **detection** : package that contains algorithms and procedures used for detection tasks.

02 **detection_msgs** : package that contains messages, services and actions used by the detection package.

03 **manipulation** : package that contains algorithms and procedures for manipulation tasks i.e. controlling the movement of the robot body parts.

04 **manipulation_msgs** : package that contains messages, services and actions used by the manipulation package.

05 **navigation_automatic_ros** : package used for navigation purposes i.e. controls the movement of the robot in the environment (adapted from assignment 1)

06 **solution** : package that manage the execution of all the nodes needed for solving the assignment

Navigation: Environment Obstacles Avoidance

In order to **avoid** that Tiago hits the **table bar** and the **cylindrical obstacles** placed in the environment, we set some **waypoints and key locations**

01

Pick key locations : define the locations from where Tiago picks the objects from the table bar. These locations are the **green**, **blue** and **red** circles in the image.

02

Place key locations: define the locations from where Tiago places the objects in the correct colored cylindrical tables. These locations are the **light-green**, **light-blue** and **light-orange** circles in the image.

03

Obstacle avoidance waypoints : define the positions that Tiago must reach in order to avoid the environment obstacles (big cylinders and table bar). These waypoints are the **dark-violet**, **orange**, pink and **yellow** circles in the image.

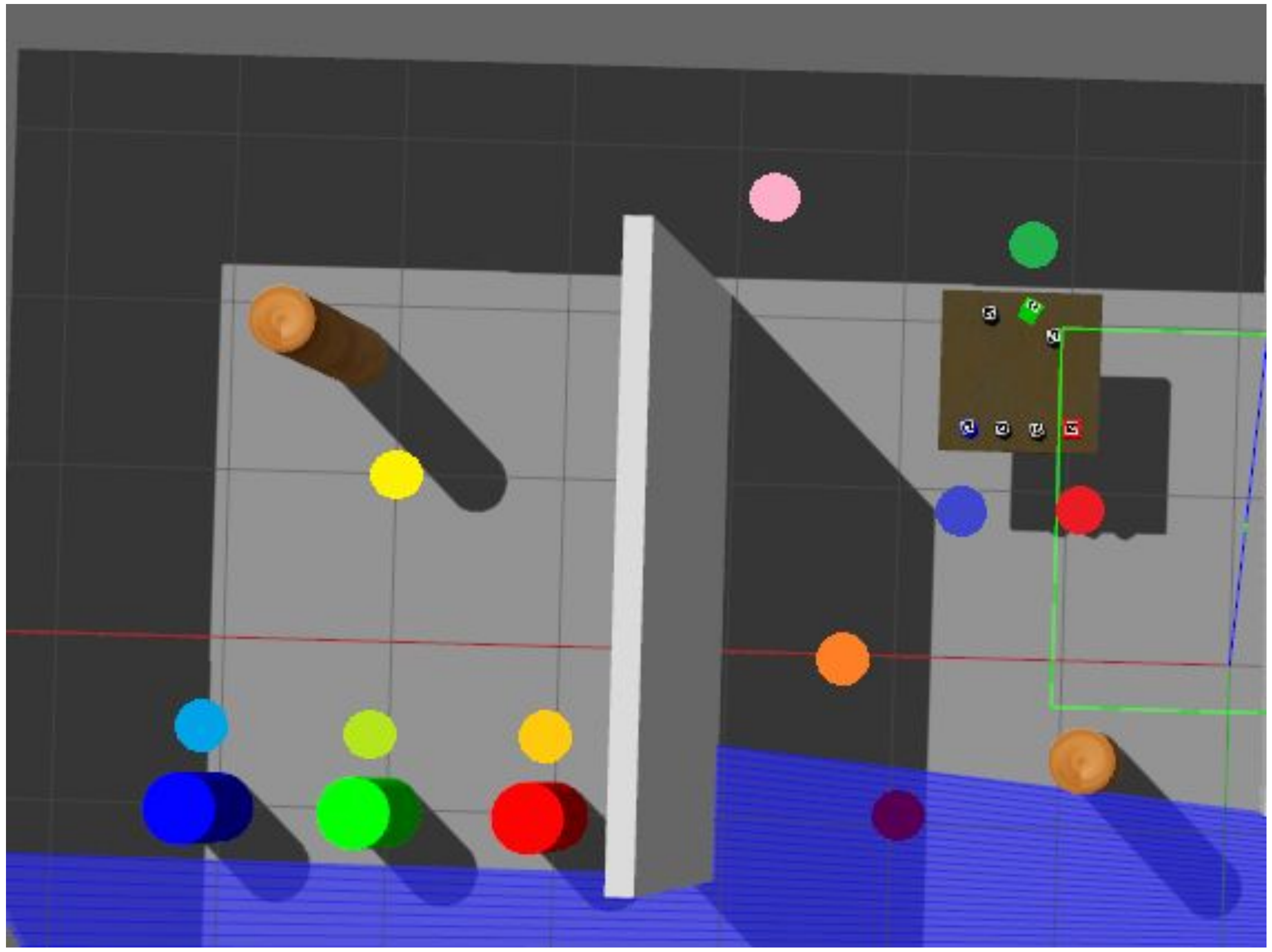


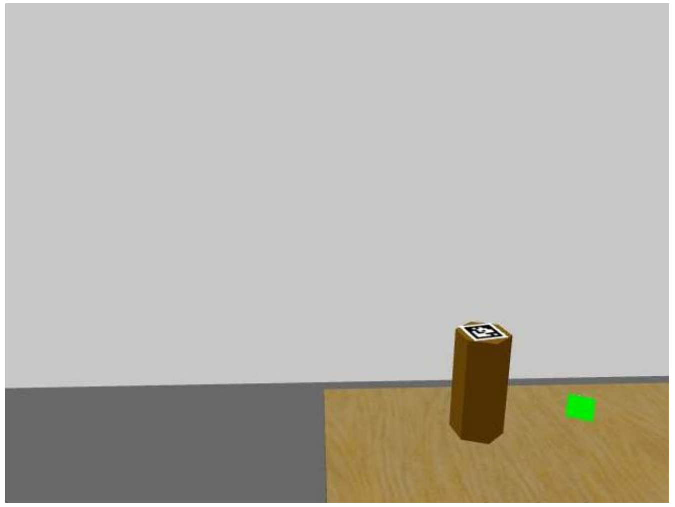
Table Detection and Pointing

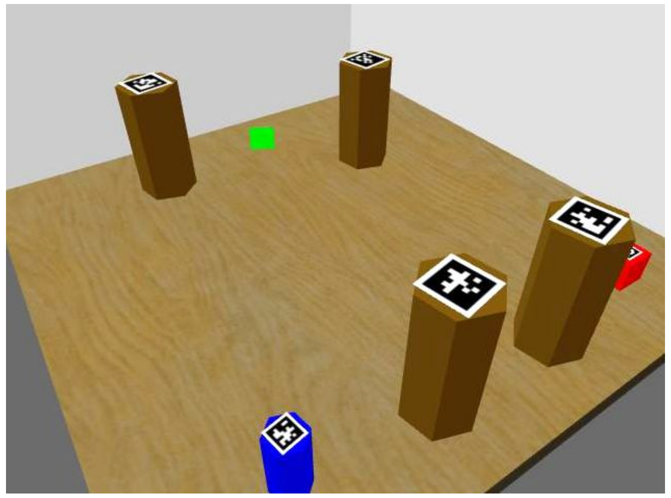
When Tiago arrives in one of the pick locations, it has to correctly point his head towards the table in order to start the detection of the objects placed on it.

The **table detection procedure steps** are the following:

01 Move the Tiago head down a little

02 Point the Tiago Head (and thus the camera) towards the table by following the color of the table. This is done by using the OpenCV library and by performing color segmentation on the Robot Camera images. The head movement is realized by using the **PointHeadAction** for moving the head by specifying a pixel in the image.





April Tags Detection: Pick and Collision Objects

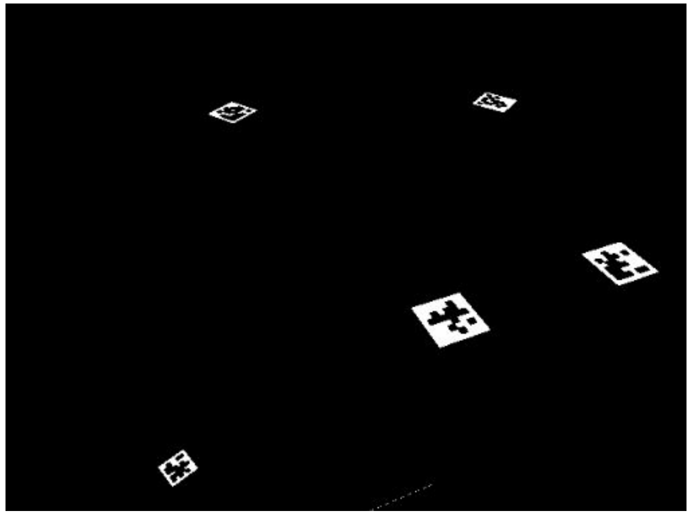
For this execution phase we assume that the table is well framed from the previous phase

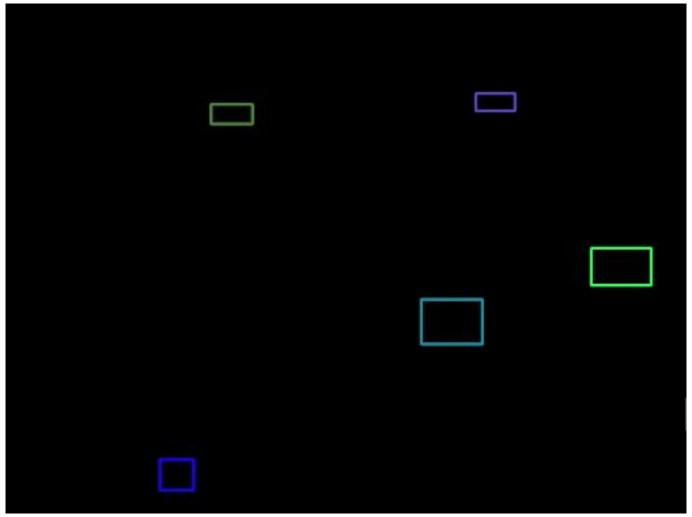
The **objects detection procedure steps** are the following:

01 Detect from the Tiago Camera images all the April Tags (done by using color segmentation)

02 Point the Tiago Head to the April Tag of the **colored object** that must be picked: this April Tag pose is then transformed from the camera ref. frame to the base_link ref. frame for the pick phase.

03 Point the Tiago Head towards all the other April Tags detected in Point 01 (except for the colored object April Tag): these April Tag poses are then transformed from the camera ref. frame to the base_link ref. frame for the pick phase. These are the poses of the **collision objects**.





Pick and Place procedure

The **Pick and Place procedure pipeline** is briefly the following:

01 Construction and setup of the correct **planning scene**

02 Move Tiago's arm to a **default pose**

03 Move Tiago's arm to the **pick pose**

04 **Attach** the picked object to the **arm_7_link** by using **gazebo_ros_link_attacher package**

05 Remove the picked collision object from the **planning scene**

06 Move Tiago's arm to the **default pose**

07 Move Tiago's arm to the **place pose**

08 **Detach** the picked object to the **arm_7_link** by using **gazebo_ros_link_attacher package**

Planning scene

To avoid a collision with the obstacles we need to tell Moveit about the objects. We achieve this by using the **planning scene**, that is representing the environment around our robot.

Given the manipulation mode we implement two different **planning scenes**, one for pick procedure and one for place procedure. The construction steps are the following:

01 Creating as many **Collision Objects** as the ones detected by the detection module:

- A Each **Collision Object** is created by using the given id, the given pose and the hardcoded dimensions

02 Apply them to the **PlanningSceneInterface**

03 **Remove** the **collision object** after pick procedure (only for pick procedure)

Move to default pose

In order to safely transport the picked object, we decided to create a **default pose** for Tiago's arm that will be performed after taking the object.

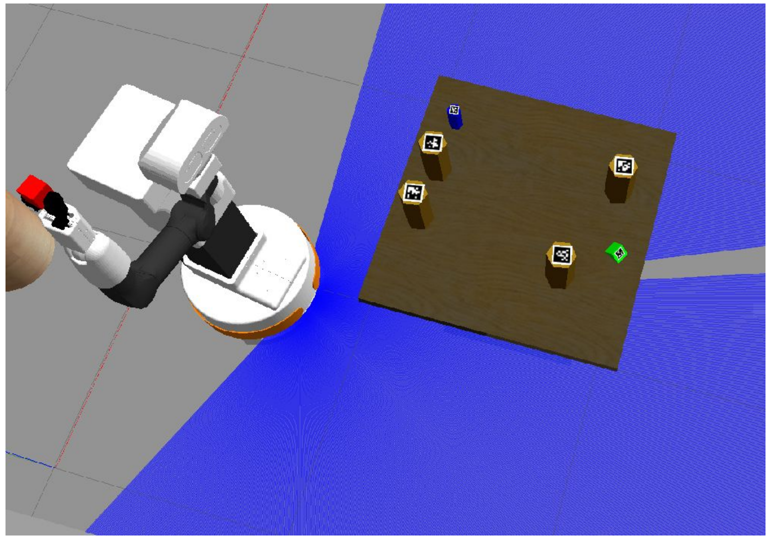
To do this, we use a **planning group** (**arm** group), and we set certain values for each joint variable of the group:

01

Such values, with respect to each joint variables, are the following:

A

arm_1_link = 0.07;
arm_2_link = 0.34;
arm_3_link = -3.13;
arm_4_link = 1.31;
arm_5_link = 1.58;
arm_6_link =0.0;
arm_7_link =0.0



02

The actual movement is executed by creating a motion **plan** and testing whether, with those target values of the joint variables, it is possible to execute the plan successfully (forward kinematics)

Set gripper to open/close position

In order to pick up the object, once the arm arrived at the pick position, it is necessary to open Tiago's gripper and, immediately after the pick procedure, to close it again.

To do this, we use a **planning group** (**gripper** group), and we set certain values for each joint variable of the group:

01 Such values, with respect to each joint variables, are the following:

A

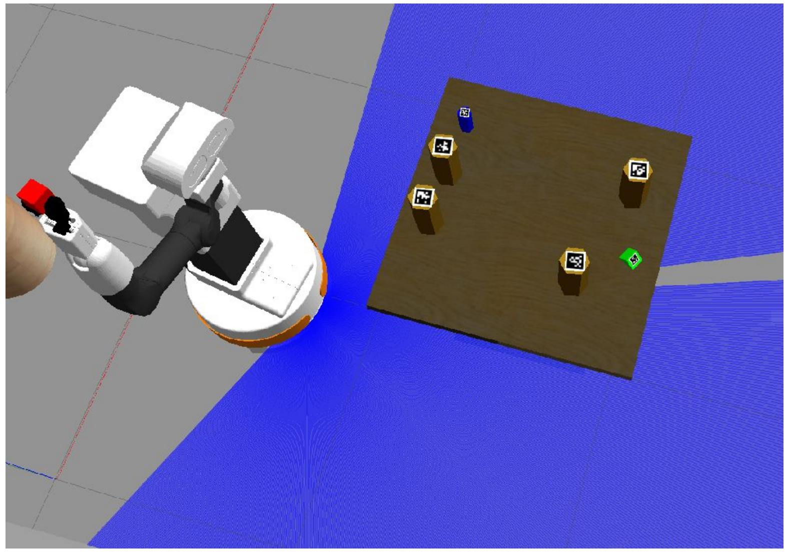
For open the gripper:	**gripper_left_finger_joint** = 0.045; **gripper_right_finger_joint**= 0.045;
For close the gripper:	**gripper_left_finger_joint** = 0.0; **gripper_right_finger_joint**= 0.0;

02 The actual movement is executed by creating a motion **plan** and testing whether, with those target values of the joint variables, it is possible to execute the plan successfully (forward kinematics)

Move to pick pose (approach/depart) (1)

In order to **pick** the object, we must move the Tiago's arm to the correctly detected and transformed pose, with respect to Tiago's reference frame. At this point we proceed in three steps:

01 **Approach**: we move the Tiago's arm to a pose given by the **pick pose + *offset***, where the *offset* is added either on the x-axis or z-axis, based on the type of pick (lateral/upwards)



02 **Pick**: we move Tiago's arm to the detected **pick pose** (with **attach** by using **gazebo_ros_link_attacher**); this is the actual pick of the object

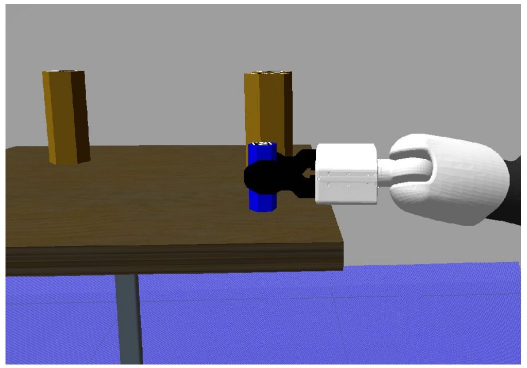
03 **Depart**: we move Tiago's arm into the old pose given by **pick pose + offset**

04 The actual movement is executed by creating a motion **plan** and checking whether there is a motion plan that takes the **arm** group from the current pose to the one we specify (**pick pose**), if so, the motion plan is successful (inverse kinematics)

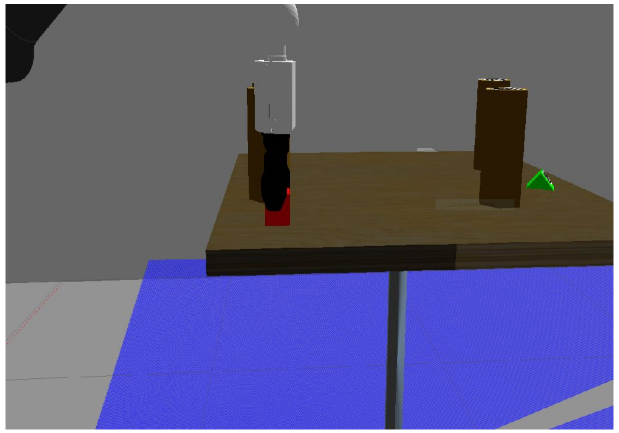
Move to pick pose (approach/depart) (2)

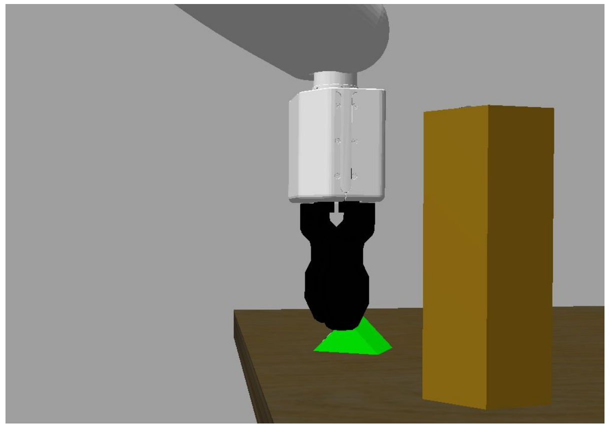
To facilitate the **pick** procedure, we used two different pick positions, one from the side and one from above:

01 **Lateral pick**: used only for the **blue hexagon**



02 **Upwards pick**: standard type pick used for **red cube** and **green triangle**





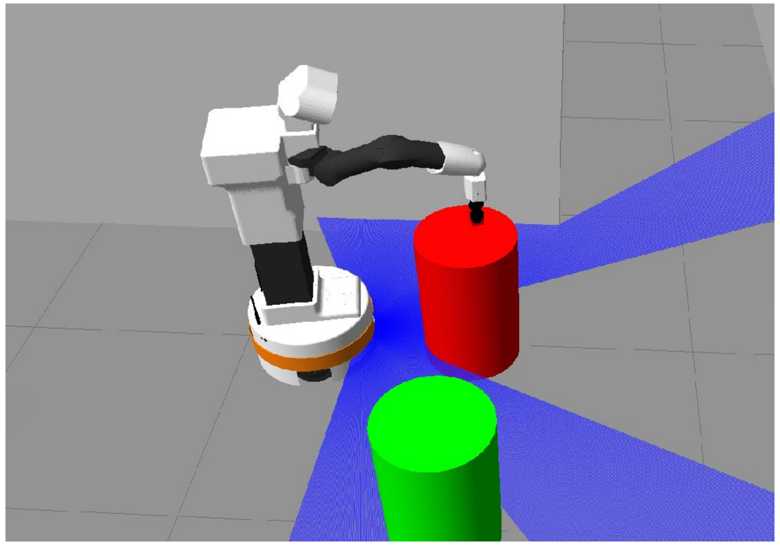
Move to place pose (approach/depart)

In order to **place** the object, we must move the Tiago's arm to the correctly detected and transformed pose, with respect to Tiago's reference frame. At this point we proceed in three steps:

01 **Approach**: we move the Tiago's arm to a pose given by the **place pose + *offset***, where the *offset* is added on z-axis

02 **Place**: we move Tiago's arm to the detected **place pose** (with **detach** by using **gazebo_ros_link_attacher**); this is the actual place of the object

03 **Depart**: we move Tiago's arm into the old pose given by **place pose + offset**



04 The actual movement is executed by creating a motion **plan** and checking whether there is a motion plan that takes the **arm** group from the current pose to the one we specify (**place pose**), if so, the motion plan is successful (inverse kinematics)

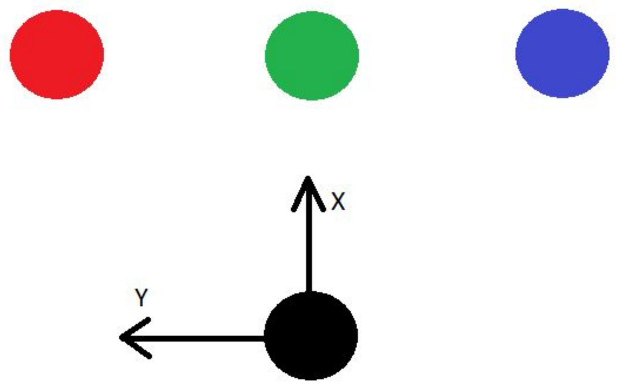
How to select the pose for placing the picked object (1)

01 For solving the Assignment **without** considering the **extra point**, we assumed that the cylindrical tables cannot move. Therefore :

A If the object to place is **red**, then, the pose in front of the table selected is the one with **largest** value of y.

B If the object to place is **green**, then, the pose in front of the table selected is the one with **neither** the **largest** nor the **smallest** value of y.

C If the object to place is **blue**, then, the pose in front of the table selected is the one with **smallest** value of y.



Notice that, the 3 poses in front of the cylindrical tables have been **hard coded**.

How to select the pose for placing the picked object (2)

02 For solving the Assignment **considering** the **extra point**, in order to select the pose for placing the objects we have developed the following algorithm :

- A **Detect** the centers of the cylindrical tables with the algorithm developed for solving Assignment 1
- B **Take a picture** with tiago camera and analyze its content in the following way :
  - a **Threshold** the image in order to remove all colors except **red**, **green** and **blue**
  - b **Compute** the **average pixel** for each color
  - c **Select** the correct center depending if the **average pixel**, corresponding to the object to pick, is on the **left**, **center** or **right** of the image, and then adding an **offset** to get the position in front of the table.

Questions?