Assignment 2

Group 06

Alberto Dorizza, alberto.dorizza@studenti.unipd.it
Federico Gelain, federico.gelain@studenti.unipd.it
Dario Mameli, dario.mameli@studenti.unipd.it

Repository at: `https://bitbucket.org/ir2324-group-06/ir2324_group_06/src/main/`
Additional material at: `https://drive.google.com/drive/folders/1kFaM5I_7Qww-PDPCiqpD-Wzap4wep6u_?usp=sharing`

1 Our approach

The solution we have developed consists of a subdivision of the tasks into three nodes, namely A, B, and C. Node A is the main node, tasked with managing the general movement of the whole robot in the world reference frame and communicating directly with node B and node C through actions. Node B is tasked with carrying out the detection, while node C handles the generation of the collision objects and the movement of the robot's arm via the *MoveIt* package.

1.1 Node A

Human node Node A communicates with the *human node* to obtain the array of **IDs** of the objects to grasp. The node provides the IDs in a random succession, which dictates the flow of execution of the pick and place operations that follow.

Table navigation This node seeks to provide a reliable way to navigate around the table, avoiding direct collisions with the obstacles. Since using the "move_base" topic through the action server *Assignment 1* did not accommodate this demand, due to the table not being taken into account in the path planning phase, we thought that using waypoints could represent a simple yet robust solution, assuming a static environment where objects do not change their pose.
These waypoints are subdivided into two categories. The first category is made of the **safe waypoints**, which are landmarks representing locations that are reachable without collisions, and sufficiently far away from obstacles that the following trajectories may also be collision-free. The second category is made of **table waypoints**, which are the viewpoints around the table that allow a complete visual of it and its elements. The first safe waypoint coincides with the last so that a loop is thus created.
All movements of the chassis are carried out using the action server defined in *Assignment 1*, by alternating safe waypoints and table waypoints, except when the robot is at the last table side, where some movements are realized using the *cmd_vel* topic to overcome the problem of the robot colliding with the table to keep its distance from the wall using the previously mentioned action server.

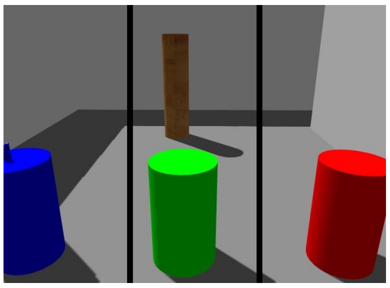
Detection and pick When a table waypoint is reached, node A moves the head of the robot to obtain a complete view of the table, then sends an action goal of type ***Detection*** to node B (Section 1.2) for each tagged element on the table. The result is used to determine whether the object with the desired ID has been found, and thus to decide whether to call node C only for the generation of collisions (in case of negative response) or to also perform the grasping (in the case of positive response), through an action of type ***Manipulation*** (Section 1.3).
In case of the object having been found and after the manipulation has finished, the navigation around the table stops, and one of two cases may arise: the robot is at any of the first two table sides, in which case the robot follows the safe waypoints to the starting one in the opposite order, or the robot is at any of the last two, in which case the navigation proceeds as defined by default. In any case, the robot ends at the end of the corridor (first safe waypoint) and the placing routine

may start.

Place The first phase consists of the robot positioning in a landmark position where a satisfactory view of the placing cylinders is achievable. By moving the head of the robot in a suitable direction, an image may be obtained by subscribing to the topic "/xtion/rgb/image_raw". After that, we convert the image into a suitable type for manipulation using functions from the *OpenCV* C++ library. In particular, the image is divided into three equally spaced **vertical sectors** (Figure 1), each containing a cylinder, and these three sectors are pushed in a vector from left to right. The average color is calculated in each sector and confronted with the color corresponding to the object with the desired ID. We then select the sector whose average color is closest to the desired one, employing Euclidean distance as the metric, and finally, we return its **index** in the vector.

- **NO EXTRA POINTS**: we define the coordinates of the cylinders by looking at the Gazebo simulation in the "map" frame (the one used to specify the pose to reach using "move_base"), and push them in a vector, sorted in decreasing order based on their x coordinate. We then select the correct cylinder **coordinates** by accessing the element at the index computed previously.
- **EXTRA POINTS**: by exploiting action server *Assignment 1*, scan data in front of the robot is available right after the pose has been reached. We have introduced a modification to the original code, where we apply further restrictions on the amount of scan data we process. Namely, only a third of the data is considered, from approximately -36.67° to 36.67°. This is due to the cylinders being positioned in front of the scan at roughly those angle intervals. When the action result is sent from the server, we filter the array of obstacle positions in the result by only considering those that satisfy the condition $x \leq 2$ (in the "base_laser_link" frame), which allows us to eliminate the outliers that may be detected far from the robot. The **coordinates** which are then chosen using the index computed in the image processing part are finally converted from the "base_laser_link" frame to the "map" frame using a transformation similar to that described in Section 1.2

The obtained coordinates are finally used to travel in front of the cylinder through the action server *Assignment 1*. Finally, after another transformation from the "map" frame to the "base_footprint" frame, we send a goal to action server *Manipulation* containing the desired position of placing. This position consists of the cylinder's coordinates and some **offsets** based on the radius of the cylinder and its height (as defined in Gazebo), which are useful for approximately placing the object at the center of the cylinder. The offsets vary depending on the objects since we place them with the same orientation of the end-effector as in the *pick* routine (Section 1.3).


(a) The vertical sectors

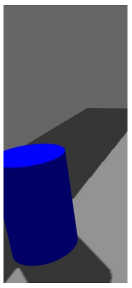

(b) A desired sector

Figure 1: A visual representation of the vertical sectors (a) and a desired sector (b), in the case of requested ID = 1.

1.2 Node B

Node B contains the implementation of action server *Detection*. After receiving from A the goal which specifies what is the ID of the object that the robot has to pick, the action server performs the following steps:

1. it subscribes to the "tag_detections" topic in order to retrieve all the information regarding each **marker** detected by the camera, which are: their IDs, their sizes (each marker is a

square, and the size represents the length of each side) and their pose, which are expressed with respect to the camera frame (in this case "xtion_rgb_optical_frame");

2. for each marker, the pose found is converted into the robot frame. We have chosen the frame "base_footprint", located at the base of the robot. To do the conversion, the **transformation matrix** from "xtion_rgb_optical_frame" (camera frame) to "base_footprint" (robot frame) is computed (let's call it T_C^B). Assuming that a certain pose P is expressed in the camera frame as P_C and in the robot frame as P_B, then the conversion is done as:

$$P_B = T_C^B \cdot P_C$$

3. the marker ID is then compared with the one received by node A. If there is a **match**, B stores separately its information.

It's important to note that, whether the object is found or not, the information of all the objects detected is still sent to A as the result of the action of the server.

1.3 Node C

Node C contains the implementation of the action server *Manipulation*. This node directly communicates with A receiving *Manipulation* goals whenever there is a need to generate collisions and optionally to perform the grasping, or if the placing operation is needed.

Generation of collisions This task is related to the creation of collision objects using the *MoveIt* library, in order to generate collision-free trajectories in the Gazebo environment.
When the instance of the *Manipulation* class is created, the **table** is built using the information about its pose and dimensions directly available in Gazebo, slightly overestimating the dimensions to plan safer paths. Collisions are generated for all of the **objects** that are visible from the robot at each table side (Section 1.1), exploiting the information about the poses of the detected objects and their IDs, coming from node B through node A (Section 1.2). Cylinder objects are created for each detected object whose ID is not either 2 or 3 (meaning green prism or red cube): box objects are instead created for these two. Their position xy is specified by the pose of the tag. All objects have dimensions that depend on the size of the tag, which is increased by a certain scale factor to approximate (by excess) the dimension of the object in Gazebo, and on the z position of the tag. By knowing the height of the table, we can directly infer the height of the object by simple subtraction.
Collisions are also similarly generated for the **placing cylinders**, with the same purpose of obtaining collision-free trajectories, each time a place operation is requested by node A.

Pick routine This routine is activated each time node A requests a *pick* operation through the *Manipulation* action. All following measurements are in the "base_footprint" frame. All movements for the **arm and torso** are carried out by first generating a plan for the move group "arm_torso", requiring either to simply specify a final pose for the end-effector or to impose a desired final configuration of the joint variables, and finally, by actuating this plan. All movements for the **gripper** are effectuated by imposing a desired final configuration of the two joint variables. The routine is synthesized as follows:

1. the robot moves its arm in the "**extended**" configuration if the object to grasp has ID equal to 2 or 3.

2. the robot moves its end-effector **on top** of the object: $n > 0$ cm above and orientated downwards if ID = 2 or 3; $n > 0$ cm above, with an offset $p < 0$ along x, and orientated towards the object otherwise.

3. the robot moves its end-effector **on** the object: $n - m, n > 0, m > 0$ cm above and orientated downwards if ID = 2 or 3; $n - m, n > 0, m > 0$ cm above, with an offset $p + q, p < 0, q > 0$ along x and orientated towards the object otherwise.

4. the gripper is closed, the collision of the object is removed, and the frame of the object is **attached** to that of the end-effector.

5. the robot **raises** its arm by m along z if ID = 2 or 3, or first moves the arm by $-q$ along x and then by m along z otherwise.

6. repeats 1., and finally moves its arm in the "**tucked**" configuration.

Place routine This routine is activated each time node A requests a *place* operation through the *Manipulation* action. It is quite straightforward, as it just moves the end-effector on the position specified by node A (Section 1.1), opens the gripper, **detaches** the frame of the object from that of the end-effector (effectively dropping the object), and finally moves back the arm in the "tucked" configuration (by first moving the arm in the "extended" configuration if the ID of the object is 1). Movements are carried out in the same ways as described in the previous paragraph, with a slight modification for the extended configuration, where the arm is slightly bent to avoid collision with the wall located on the right of the red cylinder.