Group 04: Pietro Picardi, Giacomo Gonella, Paolo Bresolin

Assignment 2

Link to drive:
https://drive.google.com/drive/folders/1QERER438IPhYsfBkhrXuQ9-1QWm1ohXS?usp=share_link

Structure

We decided to implement our solution with three nodes:

- Node_A: dictates the action to perform (move, detect, pick/place) calling the right action server at the right time;
- Node_B: handles the detection of the apriltags for picking the objects and creates the planning scene;
- Node_C: implements the pick and place routines.

The choice of building the three nodes as action client-server is due to the fact that in this way we can have feedback on what actions are being performed during the execution.

Node_A

We partitioned this node in four phases:

1. initialization of variables/connections to all servers and services needed for the execution;
2. retrieving the random picking order from the human service;
3. handling of the extra point;
4. pick/place cycle.

Clearly the most interesting phases are the last two, so we will describe just them.
As long as phase 3 is concerned, we exploited the simple_navigation_goals server returning the (x, y) coordinates of the circular obstacles detected once the target pose has been reached: using the functions transformPoseToAbsoluteFrame and findPlacePoses we were able to transform those points from relative to absolute reference frame and estimate the place poses for both Tiago and the coloured objects.
Moving on, phase 4 is rather simple: we alternate pick and place routines through a boolean flag. However, to safely navigate in the environment we had to define two waypoints, otherwise sometimes Tiago would have stuck: we believe this is due to the proximity to some obstacles (i.e. the two brown barrels and the table), probably Tiago doesn't manage to retrieve any viable trajectory because of them.
It goes without saying that in case of obstacles occupying the waypoints our solution will not work: a possible workaround could be detecting the brown barrels and create the waypoints online, exploiting the coordinates of the barrels centres and some offsets; however we didn’t manage to implement this procedure (we believe in the testers’ fairness).

Node_B

This node defines the dimensions of the objects on the table and the table itself to create the planning scene then used by node_C to avoid collisions during the pick routine. Since each object is identified by a unique tag_id (*Apriltag*), we exploited the Tiago camera to read it, producing an appropriate collision object (measurement gathered looking up on Gazebo and Rviz). Even if we know that the table position could have been detected through the laser scan, we decided to define a fixed position to make our code more robust and not subject to errors given by uncertainties due to probabilistic calculations.
Last thing that Node_B does is to call Node_C to initiate the picking phase, giving it the position of the target object.

Node_C

In order to distinguish between pick and place routines we decided to add a boolean flag to the goal of the C.action file (true = place, false = pick); once the routine is established we exploit the obj_pose field of the goal to reach the desired pose with Tiago arm.
Once again, to ensure robustness, we defined some waypoints to guarantee the desired result. Nevertheless it may happen that some strange behaviours still occur: without changing the code, consecutive runs of the simulation may produce different results, especially during the picking phase. We believe this depends on the probabilistic and randomised solver used by *MoveIt* to solve the motion and trajectory planning problems: the solver doesn't always solve the same problem in the same way; therefore even if it often finds a solution, it may happen that none is found and, as a result of the creation of the link between object to be picked and robot arm via the *gazebo_link_attacher* service, the obstacles fall over the table.
As already said we established some waypoints and some offsets to make our code as robust as possible, however we decided to limit their amount to have a general solution: the tradeoff process has definitely been the most critical part of this assignment.
That being said, although we managed to secure the pick with the gripper, we followed the suggestion to use the *gazebo_link_attacher* service to be sure that the object wouldn't fall off.
Moreover, we had to define two functions that have the same effect but need different approaches: moveArmTowardsPose and planCartesianSpace. They both make Tiago reach a pose in the cartesian space with its arm, however the former is used in the pick routine, while the latter in the place one. This is due to the presence of obstacles on the table and the need to be careful during the pick phase, while the placing process instead can be tackled in a more general way.

Extra point

For the extra point, we exploited the simple_navigation_goals server returning the coordinates of the circular obstacles (a function already implemented in the code for our first assignment) when reaching waypoint_2 (in front of the three barrels) for the first time, since we need to reach this position for each placement routine. From this information we defined the centres of the placing barrels and the pose the robot has to take to place the object; for the latter one we simply offset the centre of the barrel.

Problems

During our tests, we have encountered some random issues:

- detecting blue object: even if reaching the same robot pose and being the tag clearly visible (use the command *rosrun look_to_point look_to_point*), the object isn't detected and the program fails;
- picking green first: it may happen, once having reached waypoint_2, that a warning regarding the transformation of tag_2 from *base_footprint* (reference frame that we don't even use) arises, interrupting the program and leaving Tiago stuck.

Contribution of each member

Like in the previous assignment, we did not subdivide the work, because we preferred to meet and work together (also since we have only one machine powerful enough to run the simulation in a rational amount of time, we were both forced to do so and slowed down). Indeed, we can say that each member of the group contributed equally to the entire project both in terms of code and ideas, because almost all the code has been written together and all the ideas have been discussed together as well.