Robotica Autonoma Homework 2
Manipulation

Elena Camuffo, Emilio Olivastri, Edoardo Trombin

October 23, 2021

1 Problem Setup

The given setup includes a robotic arm placed on a table, and a set of 8 cubes, 4 blue and 4 red, each with an apriltag attached. A kinect is placed above the scene to detect the cubes poses, by means of the apriltags recognition. The task of homework 2 consists of placing 2 cubes in the area detectable by the kinect, and then to move the robotic arm in order to reach one of them (selected by the user via `frame_id`), pick it and place it in another arbitrarily chosen position.

2 Development

The routine followed by the manipulator robot to pick a user selected cube is implemented in `cube_picker` and requires an *argument parameter* defining the `frame_id` of the target cube. This routine exploits the `pose_converter` node, which was part of the *Homework 1* and provides the poses relative to each cube present on the manipulator's table, directly in world coordinates[1].
The main points of the `cube_picker`'s routine are described in the following, together with the implementation choices at every step:

1. The manipulator goes to a **default state pose**. The pose is specifically chosen to avoid covering the perception of the cubes from the kinect.
2. Each cube present on the table is linked to a **collision object**, that defines its presence in the planning scene. The collision object of each cube is a little smaller w.r.t. the original cube and has a positive offset on the z coordinate, in such a way that it doesn't directly touch the table. This choice is due to the fact that the manipulator has to grasp the cube without creating a collision between the cube and the table; in this way the problem is fixed.
3. Once the detection is completed, the manipulator starts approaching the selected cube, if the selected `frame_id` is correct and that cube is on the table. The pose reached at this step is such that the end effector is over the cube and oriented downwards, ready to grasp it.

[1]The errors committed in Homework 1 were here fixed. The node gives now a different message for each cube detected. (also the problem of directories' positioning was fixed).

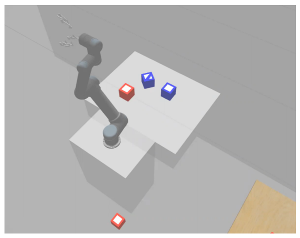
4. At this point, a combination of **Cartesian path** planning and rotation around the Z axis (yaw) is employed to find the optimal way to grasp the cube, making use of a set of *waypoints* to guide the manipulator in order to get to the goal pose. It is worth noting that the whole movement from initial to final pose is **adaptive**, i.e. it is faster at the beginning and becomes slower as soon as the end effector gets nearer to the cube, in order to achieve an higher level of precision, as it should be when considering the execution of this task in a real environment.

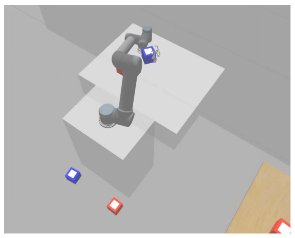
5. If the planning succeeds (we set a slight margin for errors in order to avoid complete abortion in case of negligible errors), the end effector opens the gripper and the manipulator approaches the cube along the z direction, downwards. Then, the end effector closes the gripper and catches the cube.

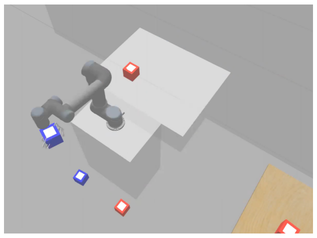
6. The system composed by the manipulator and the selected cube attached to the end effector moves upwards following the inverse Cartesian path, and then proceeds to reach the final pose. This final pose depends on the *color* of the object: the red cubes are released in the back-right side of the manipulator's table, while the blue cubes, in the back-left side.

7. As the **final pose** is reached, the end effector opens the gripper, and the cube falls. At this point, the collision object related to that cube is removed from the planning scene.

8. The manipulator comes back to the default pose and it is ready to receive another instance of the `cube_picker`, and to pick another cube.


Default pose


Pick obj


Release obj

3 Conclusions

The final result is sufficiently satisfactory, even though some issues related to the running environment (Gazebo) arise sometimes in the execution. A smooth motion is ensured by the trajectory constraints, that help to avoid wasting time because of useless twirls.
The choice of the final position, on the other hand, is constrained by the fact that once the objects are placed there, they won't be detectable by the kinect anymore. Consequently the floor in the back of the table has been chosen, because if the sides of the table had been chosen instead, the object would have been detectable but not reachable by the manipulator.
The system works pretty well in the simulated environment. However, the system of collisions wouldn't work with real objects, because it exploits the bounding boxes system, designed for simulated environments only. Nevertheless, even with a careful design, it might be difficult to apply this implementation to a real robot, as they are almost surely more error prone and the managing of the trajectories would be even harder.