Moratello Matteo, Obetti Christian, Romano Domenico
Master Degree in Computer Engineering
A.Y. 2019/20 Course of Autonomous Robotics

03/18/2020

Report Homework 2

1 Introduction

Manipulation is one of most important tasks for Industry 4.0. For this homework, a manipulation routine is implemented that allows to load the objects requested by the user on the top of a mobile robot, avoiding other objects in the scene. Due to a strong difference between real and simulated environment, several modifications have been applied to the manipulation sequence for correct functioning in both environments. In the next section we will show the main sequence used for this homework and the variants used to deal with the problems encountered during the development.

2 Manipulation Class

For a better organization and comprehensibility of the code, a class has been created. The moveit classes MoveGroupInterface and PlanningSceneInterface are passed to the constructor, the first is used to plan and control the planning group, the second instead for the collision avoidance with other objects. As class variables are also used two vectors of AprilTagDetection, which will contain the poses of objects requested by the user and the objects used for object collision. Are used also two environment variables which specify if the we are working in real or simulation environment and if we are using the gripper or the magnet. The Manipulation class contains numerous functions, each has a specific task within the manipulation sequence, the sequence used will be shown below. For each manipulator action, the corresponding function in the class is briefly described. During the description the different approaches used according to the environment (real/simulation and magnet/gripper) will also be specified.

3 Manipulation sequence

The sequence begins with the *Main Position*, the latter is placed above the table as can be seen in Figure 1 on the left. The next step is to obtain information on the poses of the objects on the table, to do this we first free the camera view from the arm by moving it to the *External Position*, Figure 1 on the right. The first two poses are set via joints.

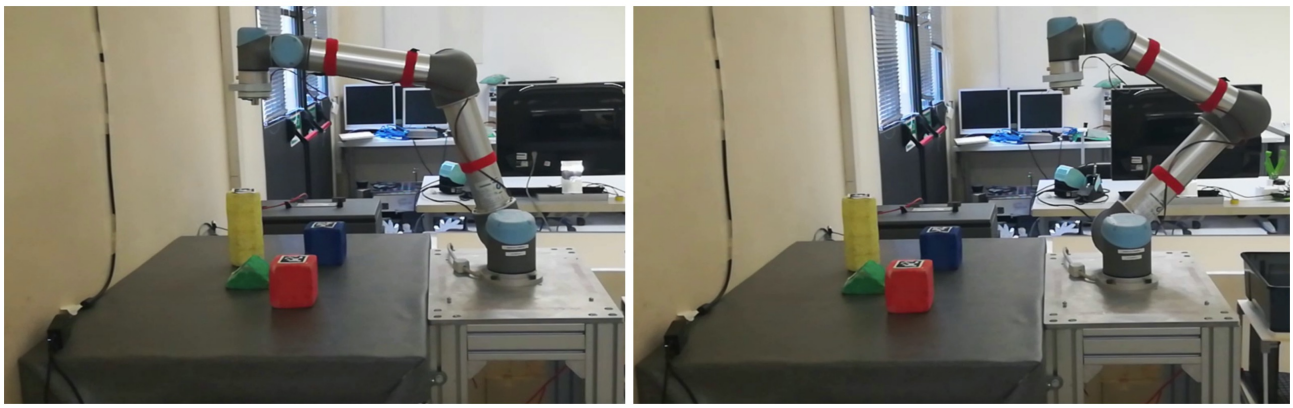

Figure 1: On the left "Main Position", on the right "External Position"

3.1 Collision Objects

In order to obtain a collision avoidance of the objects, the poses of the objects on the table are first detected through the Apriltag topic *"tag_detections"*. The poses found are then transformed from the *camera* reference system to the *world* reference system via tf2::doTransform.
Due to the error in detecting the depth of the Kinect and considering that the objects of the same type have the same size, the z positions of the poses are set with constants according to the detected object.

Real-world adjustments

Even in the real environment we have set a constant value for the z position, found experimentally for each object, to this we have added the height of the magnet (about 5.5 cm) and a small offset due to a slight inclination of the table in the x axis. This value was calculated using a linear function that returns the offset value in range [0, 0.9] cm based on the x position of the object.

The next step is to create and apply collision objects to the planning scene. With the poses just found, we have created, for every object, a collision object specifying the type, the dimension and the pose. With the cubes we used a box mesh with side of 10 cm. A cylinder mesh was used with the hexagons of height 20 cm and radius of 5 cm. The poses are set to those found previously.
Due to some errors in orientation estimation of the objects by the Kinect, roll and pitch are set to 0. Thus, they turn out to be straight and rotated only by yaw.
With the prisms we used a box mesh with side of 10 cm and with the original orientation of Apriltag detection. Since the centroid of Apriltag is not the true centroid, we have adjusted the x and y position with an offset. The offsets are computed as follows: for first we have computed the sine and cosine of yaw angle, thus we have computed the x offset as *sine*(yaw)**distance* and y offset as *cosine*(yaw)**distance*, where *distance* is the distance among Apriltag center and true center of the prism. Figure 2 explain graphically this computation.
Based on the yaw (in degree), the offsets are added or subtracted to the original x and y positions. Figure 3 shows the final result of a scene with the collision objects implemented.

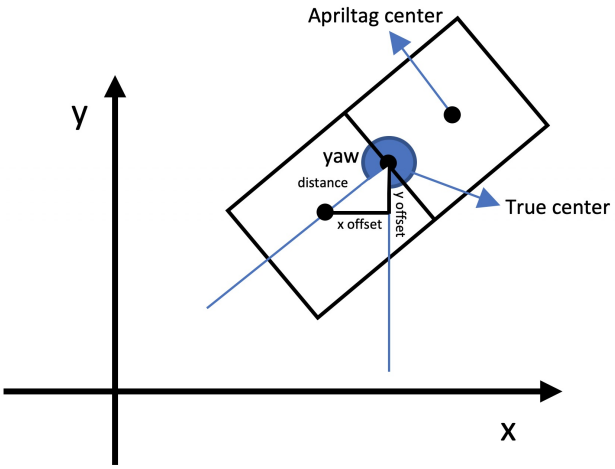

Figure 2: Centroid adjustment

Once apply the collision objects, we obtain the requested objects by the user through a subscribe to *"ur5/poses"* topic. This topic is published by the homework 1 and contains the poses of the requested objects with the same structure of Apriltag, that is AprilTagDetectionArray. We also set the z position of the objects in the same way as the collision objects.
From now on, the sequence to move the required objects from their position to a target position will be illustrated. This sequence is repeated for each object.

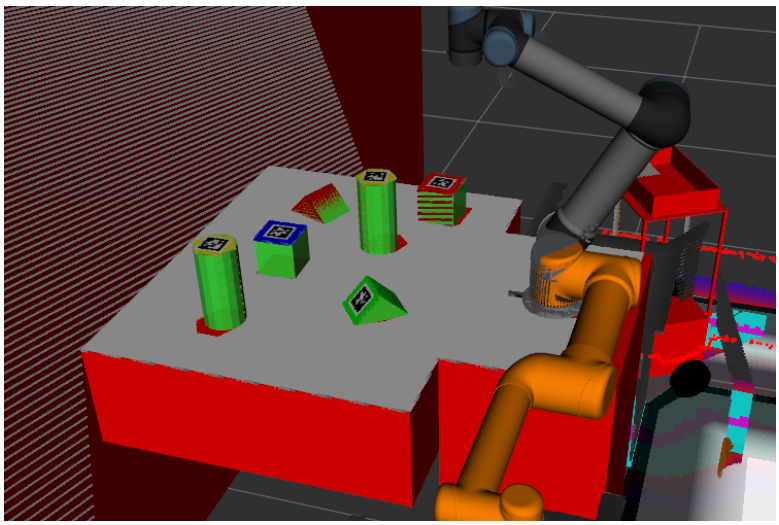

Figure 3: Collision objects

3.2 Approaching the object

The next position is again the *Main Position*, this permits to approach the objects with very few movements, facilitating the planning.
Next pose is the *Target Position*, this differ based on the environment, for this reason following we'll specify which environment we are referring to.
The *Target Position* is specified by a pose for the end effector, the poses used are those of requested objects.

3.2.1 Simulation environment

In simulation we can plan with two types of end-effector: gripper or magnet.
Planning with the Gripper
With cubes and hexagons, we have set roll = 0, pitch = $\pi/2$ and yaw unchanged for the orientation and added an offset of 21 cm for the z position from the original pose of the object.
With the prisms, we have set roll = 0, pitch = pi/2 and yaw += $\pi/2$ (for not taken the object from the inclined sides) for the orientation and added an offset of 21 cm for the z position.
The sequence continues with the opening of the gripper, the descent of the end-effector with a cartesian path of 14 cm, the closure of the gripper and finally the Gazebo link attach service (we attach the wrist_2_link of ar_ur5 to the objects link).
Planning with the Magnet
With cubes and hexagons, we have set roll = 0, pitch = $\pi/2$ and yaw unchanged for the orientation and added an offset of 20 cm for the z position from the original pose of the object.
The sequence continues with the descent of the end-effector until to meet the collision object.
With the prisms, we have set an offset of 8 cm in z position and for the orientation: roll = 2.86, pitch = 0.8 and yaw -= $\pi/2$, in this manner the end effector is folded so that it can rests on the side of the Apriltag. The sequence continues with the descent of the end-effector with a cartesian path until to meet the collision object and for all the objects with the Gazebo link attach service.

3.2.2 Real-world environment

With cubes and hexagons, we have set roll = 0, pitch = $\pi/2$ and yaw unchanged for the orientation and added an offset of 20 cm for the z position from the original pose of the object.
Using the magnet in real-world the position of the end-effector must be very precise, otherwise the object is not taken correctly as the magnet is not very powerful.
Considering this, let's get the end effector down with a cartesian path of 14 cm and then go further

down 4 mm at a time until reaching the collision object, that is when the fraction of trajectory computed by cartesian path is below 0.3.
By dropping the end-effector in this way, we are sure not to block the manipulator robot because we delimit the force applied to the object and we can avoid getting the collision object accurate to the millimeter. Figure 4 shows an example of an approach of a hexagon.

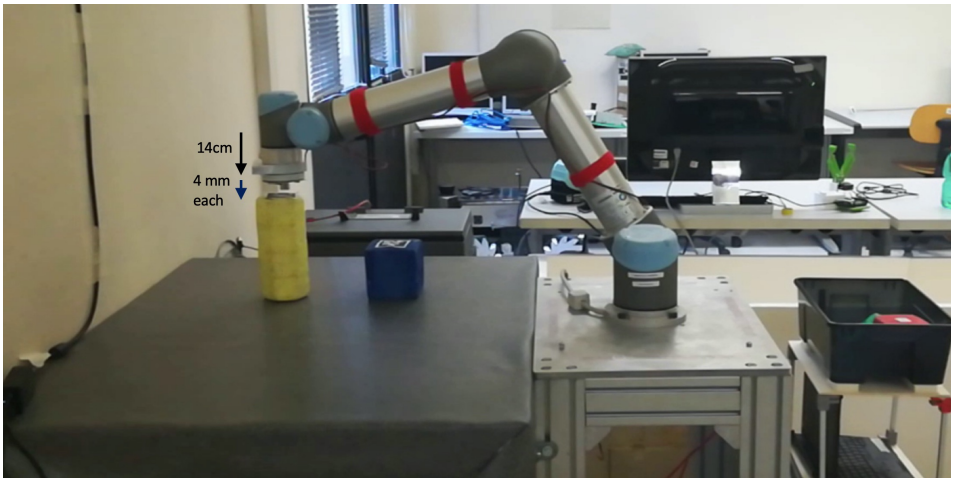

Figure 4: Approaching a hexagon with the magnet

With prisms the process is more complicated. We have first set an offset of 8 cm in z position and for the orientation: roll = 2.86, pitch = 0.8 and yaw -= $\pi/2$, in this manner the end effector is folded so that it can rests on the side of the Apriltag. In this way the end-effector constituted by the magnet is not aligned to the side of the prism, because the position x and y is referred to the eef without the magnet. We then planned an inverted V-shaped path for the eef with an angle of 90 degrees.
For planning we used the eef reference system making a transformation via tf2::doTransform. From the actual transformed pose, we have added 10 cm on z axis, then re-transformed to world reference system and planned via cartesian path.
Now we approach the object using a specific function for prisms, this function transforms as previously the actual pose in eef reference system, adds a specified value in cm in the x axis, re-transforms in world reference system and plans via cartesian path.
As done for cubes and hexagons, with this function we make the end effector descend initially by 2 cm and then by 4 mm at a time until reaching the collision object, that is when the fraction of trajectory computed by cartesian path is below 0.3. For all cartesian paths we have used default parameters.
Figure 5 shows an example of an approach of a prism.

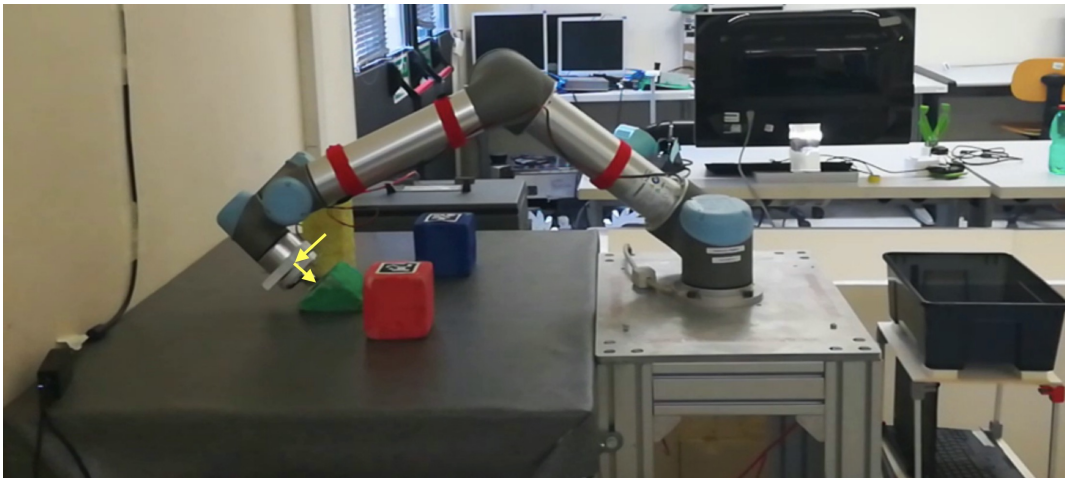

Figure 5: Approaching a prism with the magnet

The last step for attaching the object is activating the magnet through the *"/ur5/ur_driver/set_io"* service, setting command:=1. Once the object is hooked, we pull it up 15 cm planning a trajectory via cartesian path (Figure 6, on the left).

3.3 Attach object collision

To continue with the sequence, it is necessary attach the collision object to the arm.
We chose to attach it after lifting the object because in real-world the table it is slightly raised, and this can make the plan fail.
To do this, we obtain the pose of the collision object and re-add it to the scene by changing its position of 14 cm to z axis (positive), the collision object is then attached to the arm.
Figure 6 shows how the object collision is re-added to the scene and then to the arm.

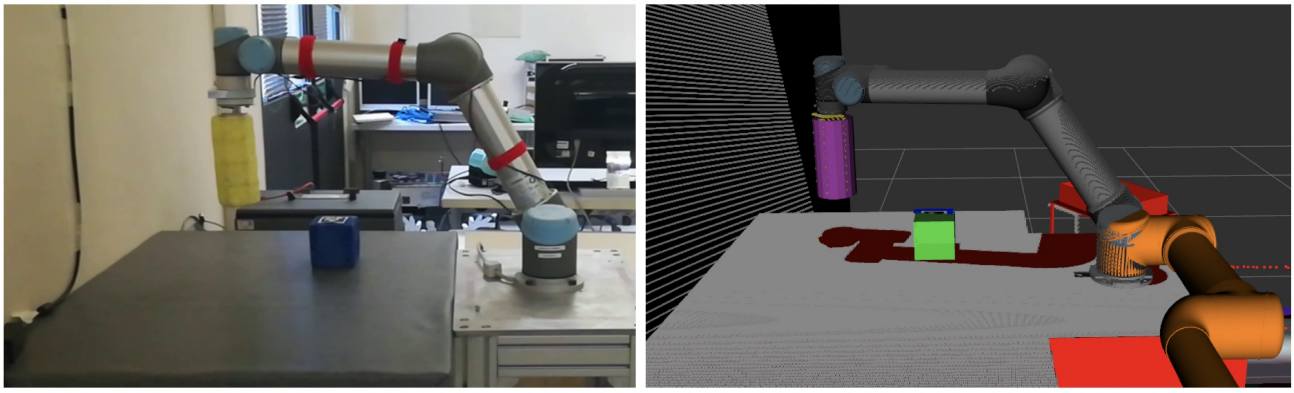

Figure 6: Object raised, on the left. Attach object collision, on the right.

The sequence continues by repositioning the arm in the *Main Position*.

3.4 Towards the final pose

To go to the loading stations, it is necessary to make a turn of about 180 degrees.
In this case, RRT-connect algorithm can plan a trajectory that is not the best one since it samples random points, and in the worst cases, the planner can fail either in plan or in its execute.
For this reason, we have decided that the robot follow a predetermined trajectory consisting of two poses taken in joints. We took the poses by moving the arm and rotating only the base joint, thus facilitating the planning. Figure 7 shows the first intermediate pose taken in joints.

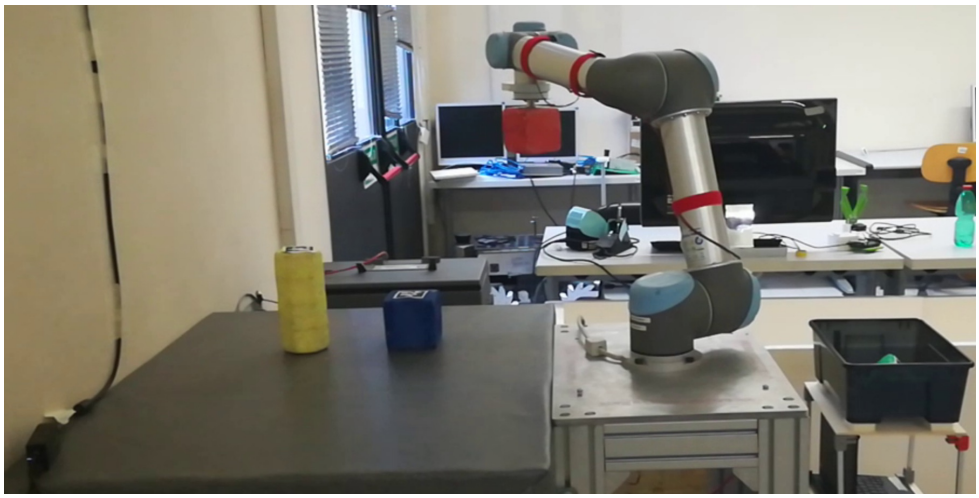

Figure 7: First Intermediate Pose

3.5 Object Recovery

Especially in a real environment, the grip of objects is not always successful, this may be due to the poor power of the magnet or because the magnets under the Apriltag are not uniformly arranged across the surface. For this reason, we have implemented a recovery function, which repeats the sequence described so far to try to hook the object again.
To check that the object has been hooked, once the arm is in the intermediate position 2, we subscribe to Apriltag and check if the id of the object is still on the table. Since the Kinect sees beyond the table and the magnet doesn't cover the Apriltag, we check that the position of the object has the position $y < 0$, making sure that it is still on the table.
If the object is still in the table, the collision object is re-added using the Apriltag information and repeats the sequence, otherwise it continues towards the final pose.
In order not to return to the same point as before, we change the x and y position of the Target pose, sampling a random point in a circle with center (x, y) of Apriltag and with a radius r of 2 cm.
Figure 8 shows the circle where the new point is sampled.

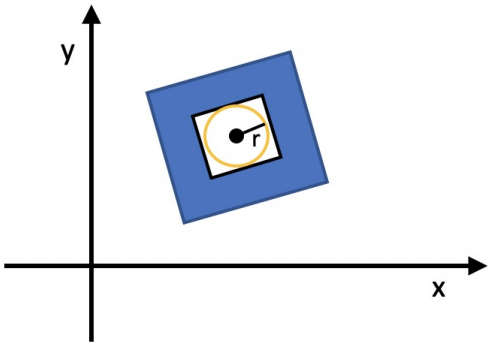

Figure 8: Sampling area

3.6 Final Pose

We conclude the sequence with the final pose. The final poses are taken based on the two loading stations. For each loading station, we took two poses at the joints, in order to unload the objects in two different areas of the box, according to their sequence number (i%2, where i is the sequence number). The loading position can be specified by the user and each box can contain about 4 objects.
Figure 9 shows the final pose in the first loading station.

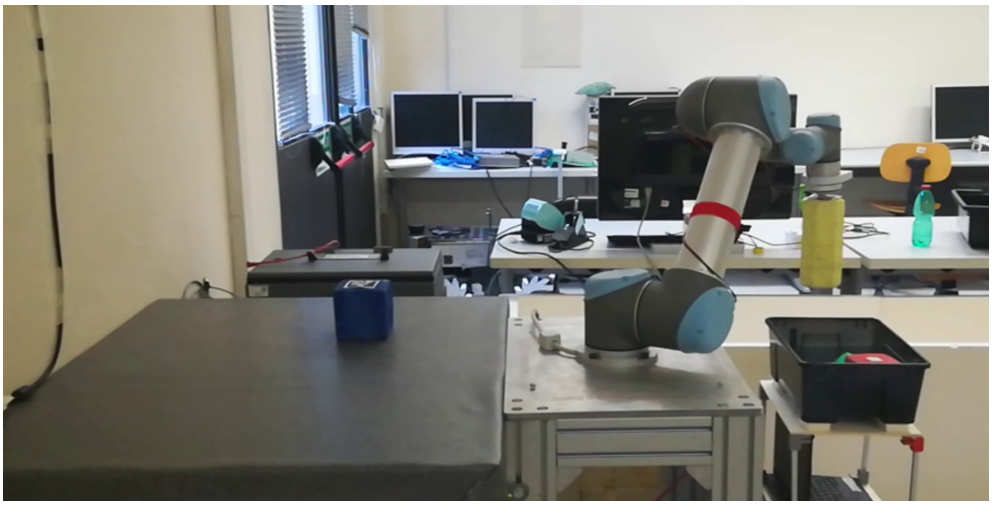

Figure 9: Final pose in the first loading station

The last step for detaching the object is deactivating the magnet through the *"/ur5/ur_driver/set_io"* service, setting command:=0.
For simulation instead, activation of the gripper opening, and activation of the Gazebo link detach service, as well as removing the collision object from and arm and the scene.
We conclude the sequence by retracing the intermediate poses in the reverse sequence. The sequence just described repeats for all objects requested by the user.

4 How to run the code

The following roslaunch command run two nodes in parallel. The first node is task 1 of hw1 which is the one that subscribe form Apriltag and publish poses of objects requested by the user, the latter are defined by **objects** keyword in roslaunch command. The second node is the hw2 node.
The parameter **simulation** permits to set the type of environment: 1 for simulation and 0 for the real one. The parameter **gripper_enable** permits to set the type of gripper: 1 for hand gripper and 0 for the magnet.

$ roslaunch hw2 hw2_moveit.launch simulation:=1 gripper_enable:=0 objects:="red_cube_2 green_prism_0 yellow_cylinder_0"

5 Conclusions

For this homework we have had quite satisfactory results, but as regards the real-world we have had the following problems:
- Power of the magnet too low, therefore you must touch the object to be able to hook it;
- Inclined and raised table;
- World positions not always consistent.