Homework 2 intelligent robotics

Mattia Fanan, Luca Davì, Hassan Habib

January 22, 2022

We have implemented a solution to pick and place all the three element in the same simulation (extra point part).

1 Problem analysis

The problem that we had to deal with in this homework was pretty articulated. We have a blue hexagonal prism, a green triangular prism, a red cube on a table and some yellow hexagonal prisms that work as obstacle (each object has a tag that associates an ID to it). The goal is, received the target ID from the human node (a mock for the human interaction), pick the related object from the table and place it in a cylindrical table with the same color (we have one of such tables for each color).

2 Project structure

The project structure follows the structure reported in the image 1. The structure was for the most part imposed by the assignment, but we felt that the pick and place actions can be taken apart, by the simple principle that if two things are different they should reside in different components (at least this applies in standard programming). This time with some previous experience from the first homework and coding in Python that removes the exhausting C++ formalism, we were able to completely implement the planned structure.

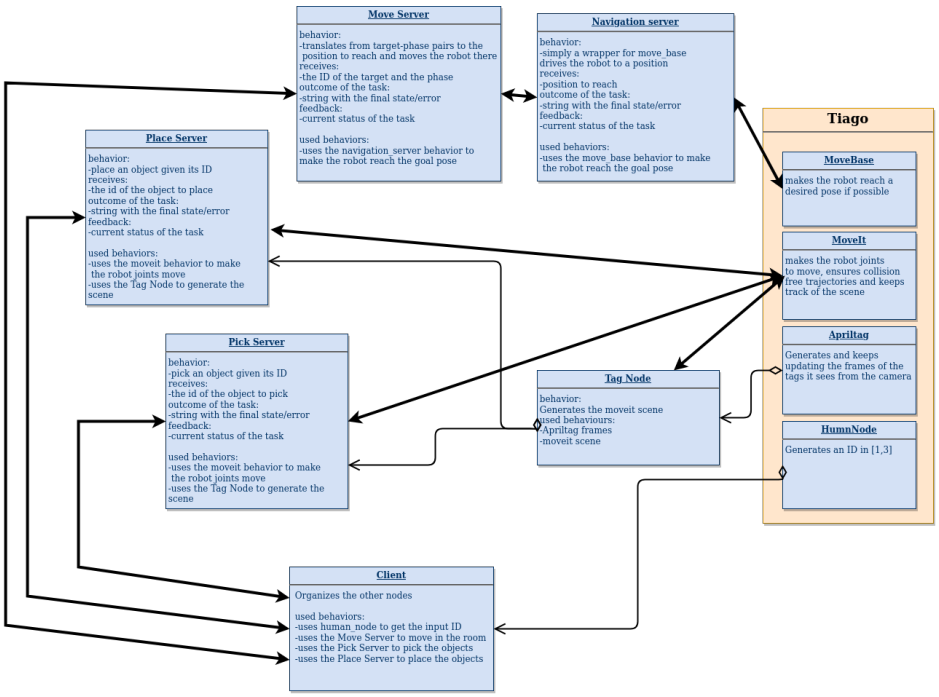

Figure 1: Structure of the system.

3 Implementation

We have implemented a solution able to pick and place all three objects (extra point part). To do so, the client will retrieve a vector of ids and will repeat the various call always in the same way.

3.1 Main Client

This node implement a simple action-client that coordinates the execution of the various steps: get human input, move to table, pick, move to destination, place.

3.2 Move Server

The Move Server encapsulates and hides the management of the positions, so that the client will only have to set a (target ID, pick\place_action) goal to ensure that the correct location will be reached.

3.3 Tags Node

For the tag identification and collision object creation we thought a lot on which between node, service or server we should use. Our final decision was to use a simple node subscribed to two topics :

- **spawn_collision_objects** that is used to enable the process of collision object spawning from the Apriltag topic, and to spawn static objects depending on the action (pick or place) given in the related message.
- **stop_spawn** that is used to disable the process of collision object spawning from the Apriltag topic.

The reasons that have driven us to develop it in this way were the fact that Apriltag continuously adjust the frame hit reads from the camera, so some of them are really unstable. For this reason we also introduced a list of already added objects so only the first time Apriltag publish an object and the spawning is on, the collision object is added to the scene. Also if we always keep the possibility of adding collision objects always enabled then is possible that when we want to remove some of them they are immediately re-added ruining the remaining part of the procedure. In the end a server or a service are not suitable since the frames from Apriltag are generated asynchronously and being the number of collision objects to generate not fixed we felt it is sensible to let it an asynchronous process.

3.3.1 Pick Server

This server is the one which is in charge of picking up the correct object from the table given it's ID. To make it more robust we felt that we can remove the suggested dependency on Apriltag frames taking the positions of the objects directly from the scene by considering its collision object, that is a mandatory dependency since we also need to manage the attached objects. This idea also solved the variability of the frames from Apriltag. Then the following steps are computed:

- The robot torso is lifted to reach an higher position from which is easier to pick the objects and see the tags.
- The Tag node is activated to build the scene and then is stopped.
- The arm is taken to a safe position from which it will start the pick procedure.
- The object is retrieved from the scene.
- The arm is taken to a pre-pick position a few cm above the object center position. Here comes into play the fact that we didn't use the frame from Apriltag (actually they come from tf but are generated and updated by Apriltag).
- The gripper is opened.
- Through a linear movement we approach the object.
- The object is removed from the scene, otherwise is not possible to touch it.

- The object is attached with the Gazebo plugin to avoid strange physical effects.
- The gripper is closed.
- Through a linear movement we depart from the target position of few cm.
- The arm is taken again in the pre-pick position through a linear movement.
- The object is put back in the scene and attached to the arm in moveit (with the gripper components allowed to touch it). Since the previous movement is extremely safe we attached to moveit after the depart motion. The objective of the attachment is to provide collision checked trajectories for the arm but if we attach it when it's near the table the two collision objects intersect and no more movement is possible. We discarded the possibility of leaving a gap between the table and the object since it means that the center of the object for the scene is no more the actual center of the real object (or the approximation of the object with the geometric entities we had at our disposal), and that is a problem since when the objects from the scene are used it is implied that the center is the actual object center.
- Then the arm is taken back to the safe position used at the start of the procedure.
- Finally the arm is taken to a safe position to allow a correct movement of the robot in the room. We chose a pose similar to the one it usually has when moving, since if it uses that position it must be safe.

3.4 Place Server

This server is the one which is in charge of placing the object. Basically it will use scene information to decide where and how to place objects. In particular the position where to place is given by the pose of the collision object related to the cylindrical table at which is summed half of its height (on z axis) because its own link is placed at the center of the cylinder. Also the procedure take into account the dimension of the object that we are placing by summing (on z axis) half the height of the attached object. At the end of the placing part, the attached object is detached so to be seen as a collision object, and avoided when returning to a safe position that allow a correct movement of the robot in the room. The entire procedure follows these steps:

- Robot torso is lifted to reach an higher position from which is easier to place the object.
- Both cylindrical table and object collision objects are retrieved from the scene, and the final pose is computed as described before.
- The arm is moved to a safe position from which it will start the place procedure.
- The arm is moved to a pre-place position a few cm above the target position.
- OPTIONAL: We wait few seconds in case the arm is swinging to avoid an erroneous placing (implemented only by code for now)
- Through a linear movement we approach the target position.
- The gripper is opened.
- The object is detached from both the gripper and the scene.
- Through a linear movement we depart from the target position of few cm.
- The arm is moved to a safe position to allow a correct movement of the robot in the room.

4 Future improvements

- Currently if there is an error on the arm movement it cannot be cached and correctly propagated since the errors like "ABORTED: reason..." suddenly appear in the screen even if it is not a server (and the documentation as usual is not so helpful).
- Add an object recognition to associate automatically a collision object shape instead of an hard-coded one.
- Make an automatic approach procedure instead of an hard-coded one. The one we tried is to iteratively try plans until one fails then execute the last planned correctly. But it's slow and warning messages appears for the failed plan.
- Automatically recognize the color of the object and the correct place table.

5 Problems

We had the problem that the mass of the objects were to heavy to lift for the arm, especially on the green target that is "guarded" by two obstacles, so when the arm had to pass over them the joint could fall under the excessive torque. As suggested in the forum we reduced the mass of the objects to 1Kg and to 0.5Kg for the tags. So there is a total weight of 1Kg object mass + 0.5Kg tag mass + end-effector.

6 Work shares

- Hassan Habib implemented the structure of the project that is then filled with the functionalities by Mattia and Luca.
- Mattia Fanan made the tag recognition and scene generation (Tag node) and refurbished the code for better encapsulation and less code repetition where possible.
- Luca Davì made the move and the place server.

Since the project hasn't an high degree of parallelism the pick server was made jointly by Mattia and Luca. Since waiting one member to complete this task, that is the more complex, would have taken away to much time for the others (the place server and scene generation are strongly dependent on how the pick is done). Also Luca was crucial to test and fix the arm poses, pick positions, ecc. Since is the only one in the group with a powerful enough machine.