Intelligent Robotics Project

Assignment 2 - Group 20

Marco Mustacchi, Valentina Zaccaria,
Damiano Zanardo

We have merged everything into the master branch and use this as final delivery.

1 Parameters, Usage and Structure

1.1 Parameters and Usage

All the parameters (global poses for the pick and place phases, properties of the objects on the table as shape, dimensions and color, position and dimension of the 'pick' table, waypoints, etc.) are defined in `task_configuration.yaml`.
cmd console 1

```
>>> roslaunch tiago_iaslab_simulation start_simulation.launch world_name:=ias_lab_room_full_tables
```

cmd console 2

```
>>> roslaunch tiago_iaslab_simulation apriltag.launch
```

cmd console 3

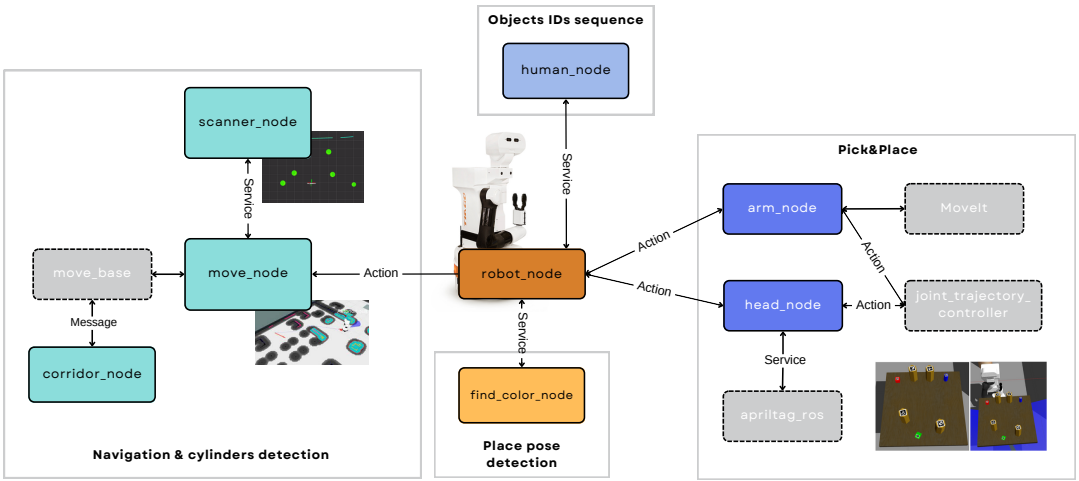
```
>>> roslaunch tiago_iaslab_simulation navigation.launch
```

cmd console 4 (first wait until Tiago has tucked its arm)

```
>>> roslaunch tiago_iaslab_simulation pick_place.launch
```

To execute the code with the optional point, the field `auto_place` is set to `true` in the configuration file. To use global poses set it to `false`.

1.2 Structure



There are four new nodes in the src/nodes folder (their implementations are inside the src folder):

- `robot_node`: main node that requests the IDs sequence, calls the action server of Ass. 1 for the navigation and cylindrical obstacles detection, calls the action server to move the head to look at the table and get the visible objects poses and ids, calls the action server to do pick or place and requests the detection of correct table in which to place the object.
- `head_node`: action server that moves the head to look at the table and requests tag detections using AprilTag packages.
- `Arm_node`: action server that performs pick and place using motion of the arm (with MoveIt package), gripper control and attach/detach of the object with Gazebo plugin.
- `ColorFind_node`: service server that given a target color, it uses the camera information to return `true` if the robot is looking at something of a similar/equal color.

Better explanation of each node is given below.

2 Objects sequence

The `robot_node` requests and gets the sequence of the IDs of the object using the provided service and service server `human_node`.

3 Navigation

As requested, the module developed in Assignment 1 has been used for navigation, through a action client-server structure. The pose for the pick is specific for each objects and it is defined in the config file.
We noticed that when attempting to reach the table, the `move_base` planner would take the shortest route and try to pass between the wall and the cylindrical obstacle. However, it got stuck and performed a recovery behaviour from which it could not continue. As a first approach we decided to exploit an intermediate waypoint before reaching the table in order to avoid passing through that area. Afterwards, we exploited an improvement of the motion control law for narrow passage of the first assignment. In particular, we also make use of the laser scans at $\pm 45^{\circ}$ relative to the frontal position in order to identify that passage as narrow passage. In addition, we exploited the fact that the robot performed a rotation on itself per recovery, so it identified the passage by already orienting itself correctly. In this case the equation for the angular velocity was adjusted as:

$$\begin{aligned} &\text{velocity.angular.z} = 0.4 \times -(\text{diff\_frontal\_laser\_scan}) \quad \text{if} \quad \text{diff\_frontal\_laser\_scan} < \text{diff\_laser\_scan} \\ &\text{velocity.angular.z} = 0.4 \times -(\text{diff\_laser\_scan}) \quad \text{if} \quad \text{diff\_frontal\_laser\_scan} \geqslant \text{diff\_laser\_scan} \end{aligned}$$

4 Pick And Place

4.1 Head control and AprilTag detections

1. Once Tiago reaches the right pick pose, it sends a goal through an action client-server infrastructure to `head_node`. The goal contains the two desired states of the joints of the head. These parameters are defined in the configuration file. The server exploits the `joint_trajectory_controller` package to move the head.
2. Then, `head_node` requests a service using `apriltag_ros` package to detect ids and poses of the visible objects.
3. Then it transforms the poses into a target frame specified in the config file and finally it sends them as result.

4.2 Arm control and Pick and Place

The routine for pick and place exploiting MoveIt has been implemented again with action server-client infrastructure. The node `arm_node` implements two actions, `pick.action` and `place.action`.

Collision Object definition

For the picking phase, the IDs and poses of the objects from the detected tags in addition with their shape, enlarged dimensions and with the pose and dimension of the "pick" table (defined in the yaml file) are used to define the collision objects using the Planning Scene interface. For the place phase, only the target cylindrical table is added. These operations are performed by the class `PlanningHelper`.
Note: when defining collision objects one has to pay attention to the height value retrieved from the tags, because the class considers the given pose as the center of the obstacle.
Note 2: to avoid the unwanted collision with the table, a much longer and wider box is defined.

Pick Action Client-Server

- The client `robot_node` sends the goal which contains the ID and the name in Gazebo of the target object (the name is needed to use the attach/detach plugin), the pose already retrieved from tags, the `pick_mode` field, to encode if it has to be grasped from the top of from the side and the field `height` to specify the height of the intermediate pose above the object.
- First, using MoveGroup Interface of MoveIt, the server plans and executes a trajectory for the group of joints `arm_torso` to raise the arm straight up (first "safe pose", if a plan can be found) and then to bring the `arm_link_7` n centimeter above the target and properly oriented (second "safe pose"). The used reference frame is `base_footprint`.
- Then, a linear movement is performed to approach the object.
- The grasp is completed by closing the gripper using the action controller interface of the package `joint_trajectory_controller`. Control parameters are in the config file.
- The object is virtually attached using the Gazebo plugin.
- Finally, the `arm_link_7` is brought back n centimeter above and then the arm is tucked.

Despite intermediate waypoints, sometimes the pick fails or the arm collides with other obstacles in the table.

Place Action Client-Server

The approach is very similar. After planning the scene, through the MoveGroup Interface, first it raises the arm straight up, then the end-effector is places above the center of cylindrical table and the gripper is oriented vertically downwards. A linear movement is performed to place the object. Then, the gripper is open with With the `joint_trajectory_controller` and with the Gazebo plugin the object is detached. The end-effector is raised and then the arm is tucked.

5 Optional Point: Docking Pose Detection

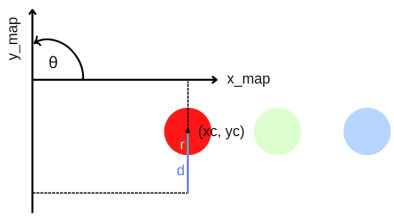
The pose in the place phase is found by exploiting both the laser data and the image seen from the camera.
The navigation/scan module of Assignment 1 has been modified in order to choose if Tiago has to detect cylindrical obstacles or not and, if so, the module returns both the estimated center and the radius.

1. First of all, Tiago must move to an intermediate pose which is properly defined in order to be able to scan the desired cylindrical table.
2. Once it arrives in the way-point, it detects the cylindrical tables.
3. For each detection, Tiago points its head toward the center of the cylinder. Then, it calls the service node `FindColor_node`. This node implement the service `find_color_srv`.
   - The client sends the target color;
   - The server gets the camera image by subscribing to `xtion/rgb/image_raw` topic. Using OpenCV library, it crops the image keeping only the central part. This works because of the agent's head pose. Then it checks if the target color (or a similar one) is present, through a basic thresholding on the three channel of the image. If this is the case, it sets the field of the response to true.
4. Once the correct table is detected, the placing pose in the map frame is defined as:

$$\theta_{target} = \pi/2$$

$$x_{target} = x_{c,map}$$

$$y_{target} = y_{c,map} - (radius + threshold)$$



Note 1: to convert the image to `cv::Mat` we use the `image_transport` package. However, this does not provide a method like `waitForMessage`. To overcome this, a while loop has been implemented. Inside it, all the available callbacks are called every 0.1s until the desired one, the image CB, stops the loop by setting the proper attribute to false.

Note 2: when we define the target color for each object/table, in one PC the camera of Tiago sees, in the BGR color space, the blue as [255, 0, 0], the green as [0, 255, 0] the red as [0, 0, 255] on the upper side of the cylinder. However, in another PC, the camera see much more darker colors, [150, 0, 0], [0 150, 0], [0, 0, 150]. We keep the latter as target colors, however there may be problems in the detection. One could solve this or by modifying the config file or by using HSV colorspace but we didn't have time to try.