Intelligent Robotics Report Assignment 2

GROUP 07

1 Group 07 Information

Sprocatti Michele, michele.sprocatti@studenti.unipd.it
Kovachev Zlatko, zlatko.kovachev@studenti.unipd.it
Serafini Lorenzo, lorenzo.serafini.1@studenti.unipd.it
Repository: *Repository* and **Drive folder**: *Addititonal material*
To run everything, we prepared a unique launch file, so the commands needed are:

```
roslaunch ir2425_group_07_assignment2 project.launch
rosrun ir2425_group_07_assignment2 ir2425_group_07_assignment2_nodeA
```

2 Nodes

The structure is shown in figure 1a. We divided the project into the following nodes:

- Node A: This node manages the entire project and implement the entire pipeline by calling the right services and manage the different operations.
- Navigation node: provide the navigation service that is requested from node A in order to navigate the map. This node based on the parameter passed will use a waypoint-based navigation or an intelligent navigation where it detects where the table are and find the set of points that must be visited (bonus point 1 implemented).
- Detection node (Node B): This node is responsible of detecting the tags using the AprilTag library and it also offer the services that are responsible for adding and for removing the different collision objects.
- Pick and place node (Node C): This node offers a service that allows the robot to reach the home configuration of the arm, to pick the collision object at a given position and to place it in the correct final position.
- Point computation node: This node implements a service that given m and q returns a point (x, y) in the line $y = m \times x + q$. The idea behind is to generate 3 points in this deterministic way (given the line): first we divide the interval between the center of table (y) and q in 2 parts so the y's of the 3 points are computed as follows q, $q + dy$, $q + 2 \times dy$ and then the corresponding x is computed.

3 Communications

To implement the communication between the different nodes, we have implemented different services:

- Collision Add Service: This service is used to add the collision objects.
- Collision Rem Service: This service is used to remove the specified collision object.
- Point Line Service: This service is used to generate point in the given line so we know where to place the objects.
- Arm service: This service is used to move the robot arm and the torso, if 0 is passed then the robot will move to the "home" configuration of the arm, if 1 is passed it will pick the object at the specified position, if 2 the place will be performed.

- Waypoint service: This service allow the waypoint-based navigation by passing the 1,2,3,4,5 that are the index of the different waypoints or it can determine the waypoints in a dynamic way if 0,1,2,3,4,5 are passed.

4 Logics

4.1 Detection and Collision Objects

The detection process allows us to add the different collision objects to the planning scene, and the idea is that the detection is active for the full program, but the collision objects corresponding to the two tables are added at the beginning, while the collision objects corresponding to the objects are added when node A requests to do it. The robot inspects the table by turning around it, and for each position, it moves the camera into 3 different positions that allow it to see all the tags on that side of the table. At each step of the camera movement, the service responsible for adding the objects is called: this service exploits the fact that the AprilTag library creates a frame for each tag that it detects, and so the service computes the transformation from this frame to the map frame in order to have the precise positioning of the different objects; then the collision objects are added.

4.2 Waypoint Navigation

Tiago's navigation operates by moving over predefined waypoints. All waypoint data and interactive functionalities are encapsulated in the class ***waypoint_generator.cpp***, while navigation is managed through a service created by ***waypoint_navigation.cpp*** using a *generator* class instance.

We opted to establish 5 distinct waypoints (1, 2, 3, 4, and 5) around the two tables: 3 for the pick-table and 2 for the place-table. After setting these initial waypoints, we realized Tiago could not move directly between them because the path planner does not detect the tables, causing Tiago to move into them. To address this, we added 4 additional waypoints (A, B, C, and D) near the table corners. With these waypoints, we developed a function to decide the sequence of waypoints Tiago should follow to reach the desired destination. For instance, to move from waypoint 1 to 4, the calculated route goes through waypoints 1 - A - B - 4 to avoid table collisions. Two more issues emerged after this update:

1. Unnecessary rotations: Tiago sometimes makes a 340° clockwise rotation instead of a 20° counter-clockwise one. We investigated this and found it is due to the navigation_stack path planner.
2. Collisions with tables: Owing to the first issue, Tiago occasionally hits a table at the start of its movement. To resolve this, we divided the movement towards a waypoint into two steps: first, rotating to face the waypoint, followed by the actual movement, thus preventing table collisions.

4.3 Intelligent Navigation[1]

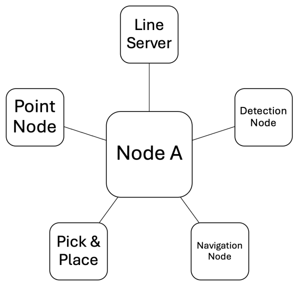
To begin with fixed waypoints, head to waypoint 1. For intelligent waypoint generation, start at waypoint 0, which selects from five preset positions to add randomness. At the chosen position, the *'scan'* message is read to form clusters. Measurements are taken sequentially, and any consecutive measurements with a distance less than **DIST_MAX** belong to the same cluster; otherwise, a new cluster is formed. Clusters with fewer than 15 elements are retained. After this threshold, only two clusters will remain with the distances between Tiago and the tables. For each cluster, theta and distance values of the first and last elements are averaged to determine table positions in **(x, y, θ, and distance)**. Finally, new waypoints are calculated based on these laser-determined table positions. Most waypoints align correctly, although occasionally, offsets are applied to match manually defined waypoints.

4.4 Pick and Place

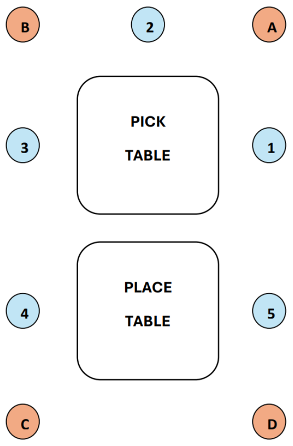
4.4.1 Home configuration for the arm

Before performing the pick and place operations, we defined a home configuration for Tiago's arm to reach. In particular we set this position for the arm gripper to be above robot's head. This position is functional to our purpose because it has the robot gripper above the table level and the whole arm is close to the robot body; in this way we avoid the collision of the arm with the walls during the navigation phase to reach a position in front of the placing table.

[1]This is the extra point 1



(a) Nodes structure



(b) Waypoints disposition

Figura 1: (A) Nodes structure (B) Waypoints disposition

4.4.2 Pose to reach calculations

For the pick operation, the arm service requires in input the pose of the apriltag of the object to pick. We set "`gripper_grasping_frame_Z`" to be our end-effector frame, this is the frame of the gripper having origin inside the gripper palm and z-axis pointing outward from the end-effector. Before sending the goal pose to `move_it`, we initially rotate the apriltag pose in order to have the z-axis pointing downward: if the original z-axis points upward, a simple rotation about the y-axis of the gripper is performed, instead if the z-axis points in an arbitrary direction, like in the case of the triangular-shaped objects, we find the axis around which we have to rotate and the angle of rotation to align the z-axis downward. This is also useful if the orientation of the apriltag is not particularly accurate. After this process, we have obtained the goal pose orientation, for the position we simply increase the z value by 0.15 m. This goal pose is not pose where we perform the grasp, instead this position is useful to approach the object from above. At this point, we pass this goal position to `move_it`.

4.4.3 Approaching and Grasping

Once we are above the object to pick, we leverage the fact that the Tiago's torso joint is prismatic along the map z-axis. By simply decreasing the value of this joint, we are able to lower the whole arm, gripper included. This allows us to then close the gripper fingers, attaching the relative collision object to the robot arm in Gazebo if we are performing the picking operation. Instead if we are placing the object: we open the gripper fingers and detach the collision object from the arm.

4.4.4 Departing

At this stage, we can depart from the table using the same reasoning as the approach: we increase the value of the torso joint, lifting the arm with the gripper. Finally we move the arm to the home configuration to navigate to the next position around the tables.

5 Issues encountered

- Even though the positions of the apriltags were well-detected, when we use them for picking up the objects, they turn out to be shifted and the alignment is not perfect. This can result in some errors during placing.
- If the object is not attached using the gazebo link attacher, Tiago doesn't grasp the objects because they slip.
- Attaching the collision objects can result in plan failing when returning back to home configuration or when performing the place operation.