Intelligent Robotics Assignment 2

GROUP 05
Sara Farris sara.farris@studenti.unipd.it,
Alberto Formaggio alberto.formaggio@studenti.unipd.it,
Michele Russo michele.russo.2@studenti.unipd.it,
Bitbucket: https://bit.ly/3SyYR4c
Additional Material: https://bit.ly/3w17lIa

January 2024

Introduction

In this report, we comprehensively explain our solution addressing the fetch and delivery behavior associated with everyday objects through the use of AprilTag and MoveIt libraries. We would like to highlight our successful completion of the Extra Point task as specified. Further details are provided below.

1 Code structure

Our project is made of several nodes, each assigned a specific task. Utilizing actions and services, we facilitate communication among these nodes. Capitalizing on the modularity provided by ROS, we have developed the following components:

- **Node_a**: It's the central node that serves as the orchestrator, coordinating all activities. It communicates with the other nodes through services or actions, directing them to execute tasks such as pick-and-place operations, object detection, motion, and more.
- **Node_b**: Functioning as a Service Server, this component takes on the responsibility of object detection. Upon invocation, it promptly responds by providing the position of the identified object. It's developed using the AprilTag library, which is popular in robotics applications.
- **Node_c**: Node_C operates as an action server, establishing communication with Node_A through an action interface. Its scope encompasses a comprehensive range of tasks related to the robot, including, pick-and-place routines, establishment of a secure position, generation of collision objects, gripper management, and both forward and inverse kinematics computations. Moreover, it communicates with *move_head_server* through a service.
- **move_head_server**: This service server, when invoked, is responsible for adjusting the orientation of the robot's head. It can move the head upwards, downwards, to the left, or the right based on the specified request.
- **object_detector**: Utilizing *Lidar* technology, this Service Server efficiently identifies the cylinders designated for object placement. When activated, it promptly furnishes a response detailing the exact positions of the recognized cylinders. **Extra points request.**
- **find_place_pose**: This Service Server, when provided with the object's identification, responds by furnishing the coordinates corresponding to the designated cylinder where the object is supposed to be placed.
- **Others**: *Robot.cpp* and *scan_clusterizer.cpp* were used and explained in Assignment 1, see previous report for details.

2 Implementation Details

In the following section, we dive into a more detailed explanation of how we achieved each single task. In Figure 1 we have highlighted the path the robot will take.

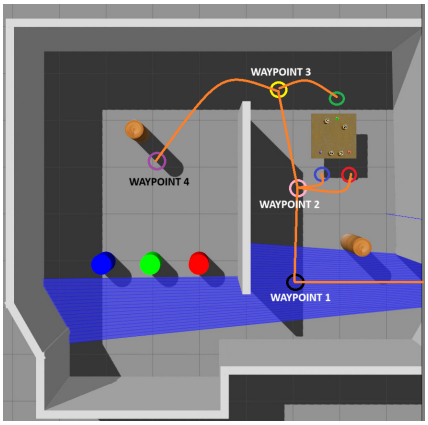


Figure 1: Robot path

2.1 Navigation

By employing the *Robot.cpp* Server, which leverages the */move_base topic* for navigation, we achieve precise robot movement within the room.

- **Towards the table**: Initially, the robot is directed through the corridor to waypoint 1, strategically avoiding the first obstacle. Subsequently, relying on the ID supplied by the human_node server, we navigate the robot towards the table in front of the correct object to be picked going through waypoint 2. If the green object is the one to be picked, the robot will proceed directly towards waypoint 3.
- **Towards the place position**: Following the completion of the picking routine, the robot is guided toward the adjacent room. Positioned in front of the second obstacle, in waypoint 4, it initiates the detection of the cylinder positions using *Lidar* technology. Once the desired position is determined, the robot proceeds to move accordingly, aligning itself in front of the designated cylinder and preparing for the subsequent placement task.

2.2 AprilTag detection

AprilTag is a type of fiducial marker system designed for visual perception applications, and it is often used in robotics and computer vision. In the context of *ROS*, *AprilTag* support is provided through the *apriltag_ros* package.

2.2.1 Head Movement

After approaching the table, we initiated the routine to move the robot's head to obtain a better view of each *Apriltag* object. To ensure a comprehensive routine, we decided to move the head also to the left and right, to capture as many objects as possible.
The service server *move_head_server* takes the request of movement of the head and sends a request to *head_action* giving the coordinates of the movement. For every head pose of the robot, we identify all visible *AprilTags* within the camera

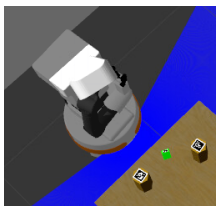


Figure 2: lateral head position

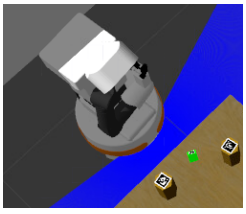


Figure 3: central head position

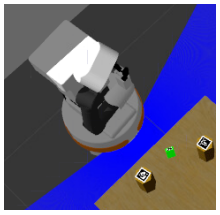


Figure 4: lateral head position

Figure 5: These images show the head's routine

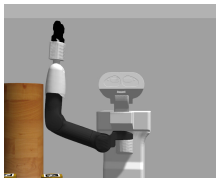
view using *Node_b*. Considering that certain *AprilTags* may be visible from various robot poses, we perform a check to ascertain whether the ID has already been detected. If the ID is not yet registered, we insert it to generate the corresponding collision object in the following stage.

2.3 Pick

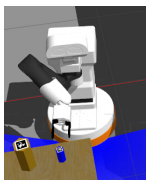
The pick routine consists of taking the three objects specified by the *Human node* at every iteration. Firstly, all objects detected during the *AprilTag* detection are converted into a collision object and added to the scene. Then the robot arm is moved to the safe position, as shown in Figure 6a.Once in the safe position, We take the yaw information of the object obtained from the orientation of the *April Tag*. Only the yaw of the object is considered as it represents the orientation along the z-axis of the object.

For each object, we set the position of the last robotic link that is different from the position of the end of the gripper. This detail is crucial to avoid collisions with the table (especially in the top pick) and to position the last link correctly. Two objects (red and green) require a top pick, while we opted for a lateral pick for the hexagon object. Additional image of the pick routine are into the folder *Additional Material* Before moving the arm through *arm_torso*, we open the fingers of the gripper using *follow_joint_trajectory* server, which is a controller for executing joint space trajectories on a group of joints. In the end, by taking the name of the fingers of the arm group of *Tiago*, we set their position to 0.9 to open them. After opening the fingers, we start moving the arm. We take the name of the group of arms we want to consider for the movement, in this case *arm_torso*, which moves not only the arm of the robot but also the torso. This first movement is made to bring the arm to an intermediate position close to the object to have a linear movement to approach the object (that is the *target position* 6b).

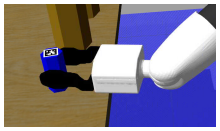
Once the arm has reached the intermediate pose, we make the linear movement of the arm to reach the target pose, and then we close the robot's finger and attach the object to the robot. Once grabbed, the object is removed from the scene as a collision object and attached to the robot by using the server to */link_attacher_node/attach* where we tell the name of the link we want to attach and the name of the object we want to attach. After these operations, we bring the robot arm into the *safe pose*, see Figure 6d. Finally, we close the robot arm to ensure safe navigation in the environment.



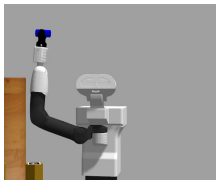
(a) Safe pose



(b) Intermediate pose



(c) Pick



(d) Safe pose with all objects

Figure 6: Complete pick routine main movements

2.4 Detection of the place position

To determine the appropriate cylinder for object placement, the robot is strategically positioned in front of three pillars (waypoint 4.1) in Figure 1. Utilizing Tiago's camera for perspective we apply a color threshold using the *InRange* method to isolate pixels matching the held object's color. For each image, a mask is generated, with white pixels corresponding exclusively to the cylinder surpassing the color threshold. The average x-coordinate for these white pixels is then calculated for each image, and the specific average is chosen based on the color of the object being identified. A sorting operation is performed on these x-coordinate averages, determining whether the target aligns with the first, second, or third position, corresponding to the assignment of the first, second, or third coordinate of the cylinder. For the part without extra points, we hard-code the positions of the cylinders but we don't explicitly tell Tiago which is Red, Green, and Blue. Tiago only knows the coordinates, but the right color associated with the object he's holding must be found by the robot itself. For the part with extra points, the procedure is the same, but the centers of the cylinders are computed as described in Section 2.6 and sorted according to the x-coordinate before running the procedure.

2.5 Place

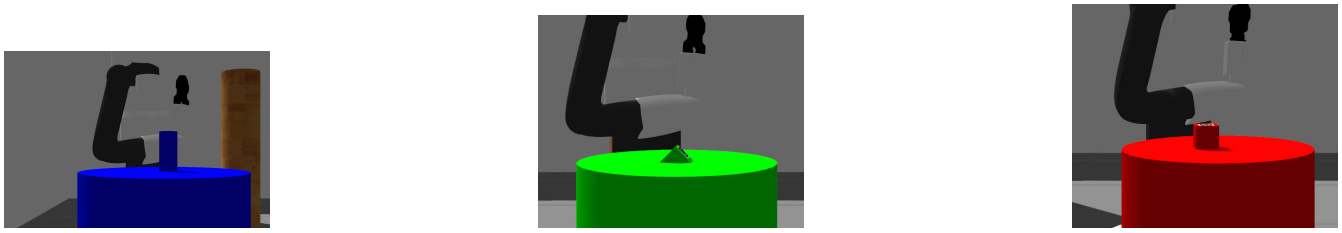
The positioning in front of the pillar varies for red and green compared to the blue pillar due to the different handling methods. For red and green, where the placement is from the top, the robot needs to be closer to the cylinder. Conversely, for the blue pillar, the object is held from the side, requiring lateral arm movement, and thus more space is needed in front of the robot to facilitate the full movement.

After positioning, collision objects associated with the tables are created, specifying the radius, height, and center position of the cylinder (only the first two are hard-coded for bonus points). The collision object is slightly enlarged for safety in Tiago's planning. Similar to the picking process, Tiago's torso is raised, followed by movement to a predefined safe position using forward kinematics.

Once in the safe position, the robot receives coordinates for the target place pillar. Utilizing backward kinematics, the arm is maneuvered to the desired position, accounting for the gripper's length and a small offset for robustness. Subsequently, the gripper is opened, controlled by the gripper_left_finger_joint and gripper_right_finger_joint joints, and

the object is detached, allowing it to drop onto the cylinders in the Gazebo environment. Finally, the robot is closed, preparing for the repetition of the procedure with other objects.

Throughout the placing process, both plannings are executed by controlling the arm_torso group and utilizing the SBLkConfigDefault planner.



(a) (b) (c)

Figure 7: Placing of the objects

2.6 Extra points part

To attain the extra points, we had to leverage the *Lidar* of the robot to determine the position of the cylinder. Employing the *scan_clusterizer.cpp* developed for the first assignment, we extracted *Lidar-detected* points and computed the centroid for each cluster and therefore, for each cylinder.

Since sometimes the circle detector may fail and detect more circles than it should (it is rare, but it happened sometimes that 4 clusters were detected instead of 3), we decided to use the knowledge that the 3 cylinders should be aligned. For this reason, we consider groups of 3 centroids and keep only the one that leads to the smallest variance.

After selecting the correct pillar by leveraging tiago's camera, we took a novel approach by realizing the potential of a geometrical solution.

In Figure 8 we depict the robot's environment. The *Lidar* detected points are illustrated in a dotted red line, while the centroid and the desired center of the cylinder are represented in blue. A line with slope m connects these two points to the robot's origin. The segment extending from the cylinder's center to the dotted line represents the cylinder's radius denoted as r with a value of $r = 0.2$. We call α the angle between this line and the x-axis whose value can be computed as $\alpha = atan(m)$.

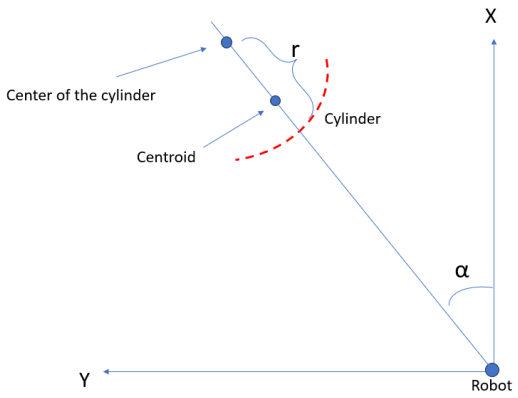


Figure 8: Robot environment

Taking a closer look at the points within the cylinder, see Figure 9, it becomes evident that α is also the angle depicted below. Utilizing trigonometry, we can calculate the displacement over x and y from the circumference to the center of the cylinder as follows:

$$dx = r * cos(\alpha)$$
$$dy = r * sin(\alpha)$$

Indeed, the determination of the cylinder's center becomes straightforward. The center coordinates can be obtained by adding the calculated displacements to the point in the circumference at the minimum distance from the centroid.

The positive aspect of this approach is that if a robot accumulates some odometry-related error, then the position of the cylinder will be wrong if converted directly to the world reference frame but correct with respect to the robot position,

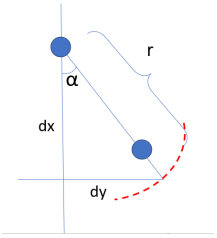


Figure 9: Close-up

thus allowing a good placement by the robot even in those circumstances. This is therefore more robust than hard-coding the positions of the cylinders with respect to the world reference frame.

3 Problems found during the implementation

In the implementation of this project, several problems were found.

Disclaimer: Sometimes the routine (planning and movement of the arm) fails randomly: this means that the same code when run multiple times may work or may not work without any explanation.

We also list some more problems we encountered below:

- **Pick**: Estimating the length of the gripper to ensure proper pick functionality was challenging.

  We also encountered issues with arm positioning depending on the machine used.

  In addition, despite providing the *April-tag* position as the goal pose to the function doing the inverse kinematics of the robotic arm, sometimes (it was random, so there is no explanation for this) the robot places its arm shifted in such a way that a movement from the target position (position at x cm away from the object) to the objects would result in a collision with the object itself.
  The object, however, is recorded correctly in *rviz* and, since we provide to the arm the position of the object untouched in x,y and modified only along the z-axis, then we assume this may be related to errors in Tiago's sensors when positioning its arm.

  We also noticed some strange behavior with planning (thus, not related to our assignment). Specifically, the robot when provided a position usually finds different paths to achieve the same position: sometimes the movement is straightforward while others it does some really strange movements. The strange thing is that, for example, when going from the safe pose to the target pose the movement is super complex, while on the way back it finds a much easier path(other groups found the same problems). It also happened a few times that for doing the movement of the arm from the target position to the object (linear movement of 10cm), the robot was computing a total new path instead of simply moving the arm forward.

- **Place**: depending on the machine placement didn't work well sometimes. Also, this was random (i.e. the same code without any modification run twice was not working on the first try but it was ok on the second try).

  The same problem with the planning described in the pick.

- **Detection using Lidar**: Although the code for the detection was already developed for the first assignment, the difference between the 2 is that the exact localization of the center of a circle strongly depends on the position, size, and distance of the table from the object. For this reason, the solution found for the 1st assignment was not viable since, from only 1 position, getting an accurate estimate of the center of the cylinder was not trivial.

4 Work Distribution

The order of the names corresponds to the amount of contribution.

- **Navigation**: Sara Farris, Alberto Formaggio
- **April Tag**: Sara Farris
- **Head Movement**: Alberto Formaggio, Michele Russo
- **Collision Objects**: Alberto Formaggio, Michele Russo
- **Pick**: Alberto Formaggio, Michele Russo

- **Attach/Detach**: Michele Russo
- **Cylinder color detection**: Sara Farris, Alberto Formaggio
- **Place**: Alberto Formaggio
- **Extra points part**: Alberto Formaggio, Sara Farris