UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Assignment 2

Report for the 2^{nd} Assignment of Intelligent Robotics

Group: 10
Author: Nicola Calzone, Leonardo Da Re, Nicolò Tesser
Professor: Emanuele Menegatti

31/01/2024

1 Authors, Emails and Links

GROUP 10
Nicola Calzone, nicola.calzone@studenti.unipd.it
Leonardo Da Re, leonardo.dare@studenti.unipd.it
Nicolò Tesser, nicolo.tesser@studenti.unipd.it

Repository, click here
Drive Link, click here

2 Extra Points

We have implemented additional features using the laser system and introduced a new routine enabling Tiago to recognize the colors of the cylinders. This includes the ability to rearrange the positions of the cylinders while ensuring they remain visible in Tiago's camera, as it is possible to see in the videos published above at this link .
For further details, please refer to the section 9 on Cylinder Recognition .

3 Our Setups

- Calzone's: VLAB, in some cases VM Ubuntu 20.04 (i5, no GPU, 4GB RAM).
- Da Re's: VLAB, in some cases bare-metal Ubuntu 2.04 (Ryzen 9, GPU, 32GB RAM).
- Tesser's: VLAB.

This time, we all mainly utilized VLAB for both code development and testing, since Gazebo simulation presented crashes and issues when executed locally. Writing code on our local machines' IDEs and testing in VLAB was time-consuming, hence the decision to rely completely on VLAB for a faster workflow.

4 How to run

To run the code, you should clone our repository into `src` directory inside your `catkin workspace` (same in which we clone the tiago_iaslab_simulation repository), which you can find at this link: on this link.

1. Navigate to the `catkin_ws` directory and execute: `catkin build assignment2`
2. Execute:
   `roslaunch assignment2 assignment2.launch`
3. Execute: `rosrun assignment2 scan_node`
   (ensure that the node did not stop after few seconds; in case, re-run this command)
4. Execute:
   `rosrun assignment2 client_node`

We decided to keep scan_node out of launch file since it may crash due to compatibility issues between VLAB and the `OpenCV` library. We also kept the `client_node` out of the launch file to have a more clear output and visualize the results.

5 Architecture

Our system architecture revolves around a central `client_node`, which interfaces with various nodes: `navigation_node`, `detection_node`, `manipulation_node`, and `scan_node`, by subscribing to their services and actions as shown in [Figure 1] and [Figure 2]. The sequence of operations carried out by the `client_node` is as follows:

1. Establishes a connection with the `human_node`, storing the three IDs in the vector `object_order`.
2. Calls the `navigation_node` to guide Tiago to the table.
3. Upon reaching the table, the `client_node` invokes the `detection_node`.
4. The `detection_node` identifies `apriltags` on the table and returns control to the `client_node`, providing a vector of `apriltag_ros::AprilTagDetection` objects.
5. Calls the `manipulation_node` in the `doPick()` function, passing the detected `apriltags` with the pickable objects and the collision obstacles.
6. Calls the `navigation_node` to guide Tiago to the "Scan Position" (shown in [Figure 3]).

7. At the Scan Position, calls the `doScan()` function, involving the `scan_node`. This node computes the central cylinders' points with the *laser* and assigns colors based on RGB thresholds from the image. The result is passed back to the `client_node`.

8. Guides Tiago to the desired cylinder position using the `navigation_node`.

9. Calls the `doPlace()` function, telling the `manipulation_node` to perform the place action.

10. Guides Tiago to the "Home Position", completing the cycle. In the final iteration, Tiago stops at the Home Position; in other cases, Tiago cycles between the Home Position, Pick Positions, Scan Position [Figure 3]

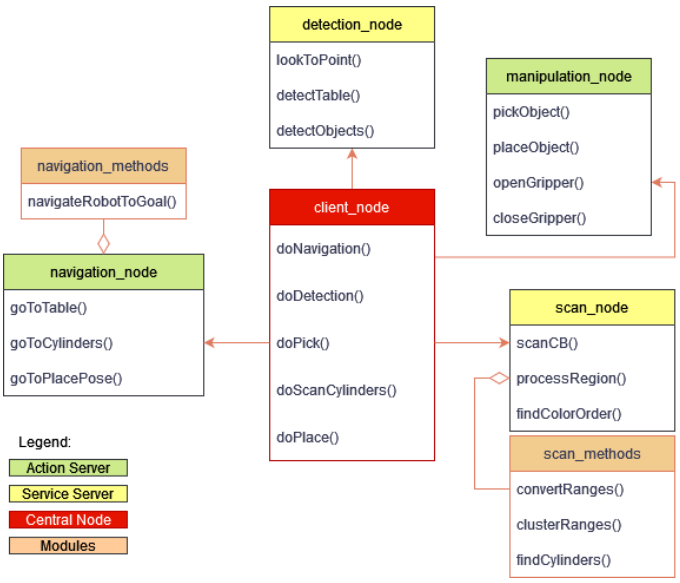


FIGURE 1: Architecture of the Nodes

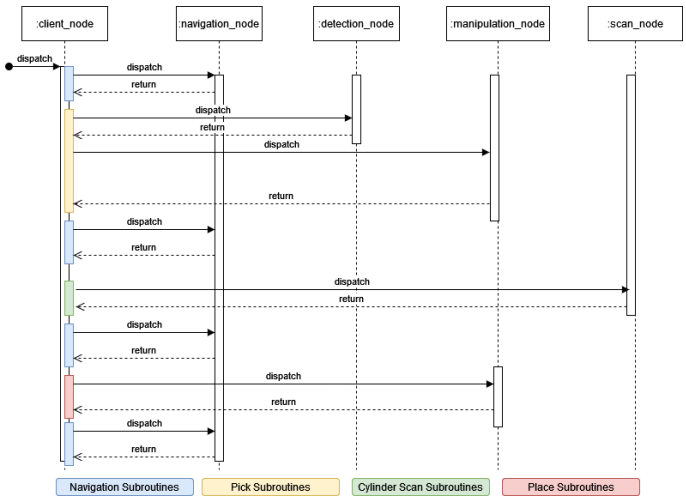


FIGURE 2: Sequence Architecture for a single loop

6 Navigation

As for Assignment 1, we have implemented the navigation_node as an action server because we wanted to receive feedbacks during the execution to have a full understanding of the process in case of errors and the possibility to trigger determined taks only when Tiago was in a certain status.

6.1 Assumptions for Navigation

- **Object Pick:** we assume knowledge of object IDs provided by the *human_node*. Tiago directly reaches three pre-defined positions in a given order to pick up objects from the table. These positions are designed to be robust and resistant to small changes in object positions on the table.

- **Waypoints to Avoid Stuck Points:** To solve the cases where Tiago gets stuck, especially near the table and the first obstacle outside the corridor, hard-coded waypoints are used.

- **Fixed Positions:** There are 5 hard coded positions, as shown in [Figure 3].

  1. Home position: Tiago performs all his navigations to the table from here. Also his last position reached.
  2. Pick positions: Three fixed positions for red-blue-green objects on the table. From here, he is able to pick the objects detecting them dynamically.
  3. Scan position: From here Tiago performs a scan of the cylinders and their colors.

- **Loop ends at Home Position:** After each placed object, Tiago returns to the Home Position to simplify navigation and avoid the need for additional waypoint computations in `navigation_node`.

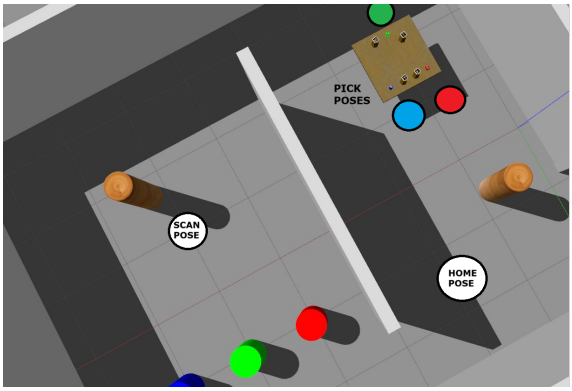


FIGURE 3: All the hard-coded positions of our code

6.2 Ideas behind Navigation

This server consists of a callback that answers in a different way depending on the `goal->operation` value passed as a request from the `client_node`.

- `goal->operation` = 1: Tiago will reach the Table passing from Home position. This operation is outside of the loop.

- `goal->operation` = 2: The loop has started, Tiago reaches the Scan Position.

- `goal->operation` = 3: The cylinders have been scanned, or if something went wrong the `client_node` has called a RecoveryNavigation function which assigns three hard-coded positions of the cylinders. Tiago will try to reach one of them based on the other parameter passed, the `order_object`.
- `goal->operation` = 4: At the end of the first two loop cycles, Tiago will reach Home Position again and then go back to the table.
- `goal->operation` = 5: At the end of the last loop cycle, Tiago reaches Home position.

To make the code more readable, robust and easy to understand and edit, we have implemented some functions for very small and specific tasks as for example: `goToTable()`, `goToHome()`, `goToScanPosition()`.

7 Detection

The detection_node, implemented as a service, focuses on obtaining object tag poses. The decision to use a service was driven by the specific needs of obtaining tag poses without requiring additional control features.

7.1 Assumptions for Detection

- The position of objects on the table may change but remains close to the original.
- The orientation of objects remains constant.

These assumptions guided Tiago's navigation to three predetermined positions near the table for obtaining object poses through the head camera. The camera publishes periodic information on specific topics, and AprilTag reads image data from these topics to return the poses of objects.

7.2 Ideas behind Detection

`AprilTag` processes the camera data and returns the poses of pickable objects and obstacles in the camera's reference frame. We initially computed the poses in the `odom` reference frame, but later on we found that we had better results computing them in the `map` reference frame for trajectory planning. Since Tiago's camera initially points straight ahead, adjustments were made to align it with the table's center. This alignment allows AprilTag to calculate tag poses accurately.
Once the object positions are known, they are passed to the manipulation_node, enabling Tiago to execute the pick action for the desired object. [Malyuta, 2017] [Wang & Olson, 2016]

8 Manipulation

The `manipulation_node` was implemented in a Client-Server architecture because the manipulation is a complex task thus we want to have some more information as feedbacks from Tiago while picking/placing the objects, like the intermediate position of the arm to better debug and adjust the trajectories.

8.1 Assumptions for Manipulation

For this task our assumptions are:

- The height of pick up table is fixed in order to calculate correct dimensions for defining collision obstacles
- Also height of place cylinders is considered fixed in order to compute correct place position for the gripper
- For the pick phase we assume that Tiago is in front of the requested object
- Sometimes objects fall to the ground because the gripping force in the simulation is not very reliable
- Obstacle objects are slightly bigger than the original to have more possibility of avoiding them

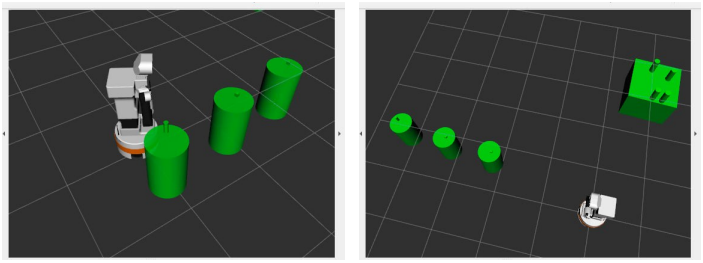


FIGURE 4: Rviz - `doPlace()` of all the items from two different perspectives

8.2 Ideas behind Manipulation

The manipulation is divided into to different actions: pick and place. After a first approach in which we tried to pick up the objects from the side, we decided that picking up the objects from above, with the gripper pointing down, is a better choice. We have three main configurations for tiago's arm during the picking trajectory:

- approach pose: above the object with the gripper pointing down
- goal pose: near to the object where the gripper is closed
- defalut pose to navigate with the arm closed

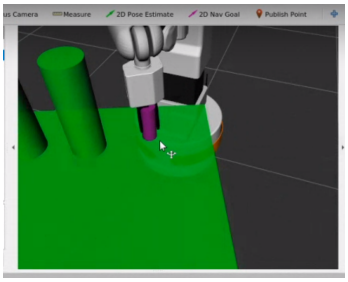
For the place phase the ideas are very similar, we use two configurations (approach and goal as before) and then the arm is closed in the default pose.

9 Cylinder Recognition

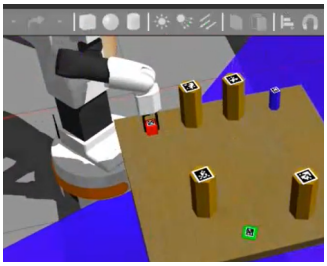
The `scan_node` is implemented as a service server. We chose this implementation because we needed the `client_node` to wait for the `scan_node` response without executing anything else in the mean time and because we did not need any feedback, but just a request to pass and a response to receive.

9.1 Assumptions for Cylinder Recognition

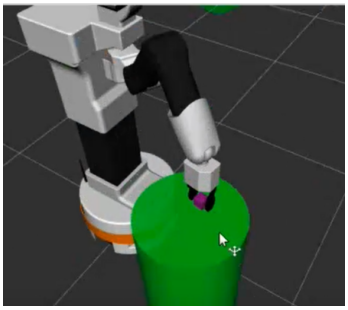
- **Optimal Scanning Position:** We assumed an optimal position exists for scanning cylinders. This position is crucial for the `scan_node`, that operates with a specified angle of Tiago's camera.



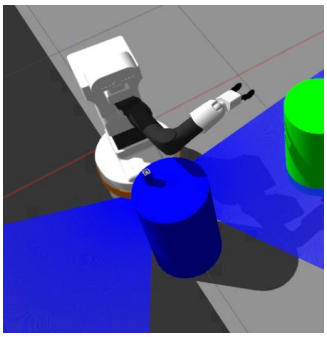
(A) `doPick()` of item 1 in Rviz



(B) `doPick()` of item 3 in Gazebo



(C) `doPlace()` of item 3 in Rviz



(D) `doPlace()` of item 1 in Gazebo

FIGURE 5: Manipulation in Rviz (left) and Gazebo (right)

- **Right-to-Left Cylinder Scanning:** Cylinders are assumed to be scanned from right to left, consistent with the clustering perspective obtained from their positions.

- **Minimal Cylinder Displacement:** While it is possible to interchange the cylinders positions completely since we are recognizing the colors, it is still recommended to avoid displacing the cylinders excessively and try to maintain the 3 original positions, ideally within centimeters, on both the x and y axes. Despite the first task ensuring correct position identification, Tiago's camera may face challenges in accurate color detection from the scanning pose.

9.2 Ideas behind Cylinder Recognition

There are two main ideas behind this module of our program:

1. LaserScan Processing: The node clusters the `ranges[]` vector, locating three cylinders based on the difference in distance on x and y coordinates from the center of the LaserScan in `'base_laser_link'`. It populates the `geometry_msgs::Pose pose` field of the `Scan.srv response`, ensuring robustness against cylinder shifts.

2. Color Recognition: The node identifies cylinder colors by dividing the image into three regions (right, central, left) and thresholding shades of Blue, Green and Red exploiting the *OpenCV* module [Bradski, 2000]. The dominant color in each region determines the cylinder's ID, populating the `int32[] ids` field of the `Scan.srv` response. **Important**: This method ensures adaptability to color changes, but displacement beyond Tiago's camera limits may result in recognition issues. Therefore, we suggest not to displace the cylinders too much.

10 Conclusions

Working on this project has been interesting and has provided us with a profound understanding of how to work with ROS, Moveit, and software tools like Gazebo and Rviz. The program effectively accomplishes its tasks, such as selecting the correct objects, navigating to the designated locations, executing the required actions, and incorporating recovery plans. Additionally, it includes auxiliary functions that come into play when something goes wrong or when changes are introduced to the environment, such as swapping cylinders or adjusting their positions by a few centimeters.
We are happy with how we managed the process, structured the architecture, and deep dived into the various tasks.

11 Group Work

To work at this project we have always worked the three of us together. We have met in person or through video calls, sharing the screen to allow easier pair-programming sessions. We worked together at all the parts of the project, but the major contribute to the single tasks was divided as follow:

- Calzone: Client - Cylinder Recognition - Navigation - Detection - Report ($\sim$ 140 hours)

- Da Re: Manipulation - Detection - Documentation of the code - Testing ($\sim$ 140 hours)

- Tesser: Manipulation - Detection - Client - Navigation - Report ($\sim$ 150 hours)

References

[Bradski, 2000] Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools.

[Malyuta, 2017] Malyuta, D. (2017). Guidance, Navigation, Control and Mission Logic for Quadrotor Full-cycle Autonomy. Master thesis, Jet Propulsion Laboratory, 4800 Oak Grove Drive, Pasadena, CA 91109, USA.

[Wang & Olson, 2016] Wang, J. & Olson, E. (2016). AprilTag 2: Efficient and robust fiducial detection. In 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 4193–4198).: IEEE.