UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Project report

Fetch-Delivery routine for the Robot Tiago

Developers:

Gabriel Taormina
Stefano Trenti
Matteo Villani

Link

Soruce Code

1 Description

The goal of the project is to implement a fetch and delivery behavior for the robot Tiago.
In the simulated environment there are two rooms separated by a wall, plus some obstacles into the environment. In the first room is placed a table with some objects above that the robot has to fetch (red, blue and green), plus gold hexagons that Tiago has to avoid, and thus not collide with them. Tiago has to grasp the correct objects in some defined order, not collide to not wanted objects, move to the second room where are present coloured tables, pick the color-associated table, move towards it, place the object and come back in the previous room. The goal is considered achieved when Tiago places correctly the objects on the right tables in the wanted order. In some id configurations provided by the Human node, particularly in 1, 2, 3, the pick phase exhibits abnormal behavior a few times. Although the arm has correctly inserted the collision objects into the scene, it does not take into account one of the gold obstacles causing it to fall. This occurs most of the time when the first object to be picked up is the blue one (ID 1). However, the behavior changes depending on whether the second object is the red or green one.

2 Structure

To accomplish this task we decided to exploit the modularity and scalability of ROS, implementing the structure with different service nodes and action server nodes that are all called upon by the main central node **node_a**. Because of the structure of the program we decided to implement a launch file that starts the simulation with all service and action server nodes, only **node_a** is left out and to be started manually.
In order to analyze the structure of the program we want to refer to the following image, in which are presented how the files (nodes/service/client/server) interact with each other in order to achieve the fetch-delivery behavior. In the graph are present:

- **Rounded blocks**: they represents the main **nodes** in forms of service/action client-server that manages the deliberative at higher level;
- **Square blocks**: they represents the components that perform sub-tasks demanded by the main nodes.

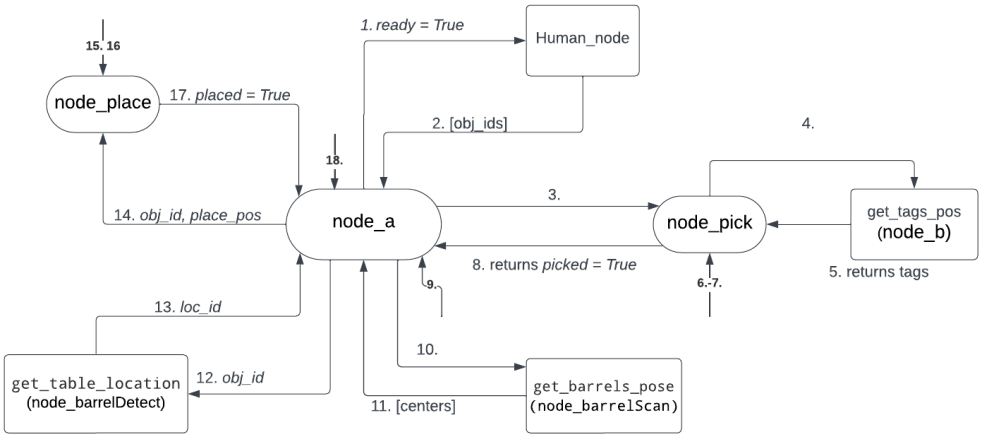


Figure 1: Block diagram

We want to briefly discuss what the most relevant parts of the structure presented in the Block diagram performs:

- **Node_a**: this node is most central node, since it coordinates the flow of the program by assigning the sub-tasks to the other components and manages the pick-place routine for every object.
- **Node_b**: it advertises the service that is in charge of obtaining the poses of the wanted objects on the pick-up table by exploiting the Tiago's camera and the **AprilTag** library, thanks to the markers placed on the top of these objects, and then converting them according to the desired frame. This was implemented as a service node so that it can be called only when required.

- **Node_Pick**: this represents an Action Server. This node is concerned of the whole **pick** routine. By exchanging information with *node_b* it obtains the required data to perform a **MoveIt** routine that accomplish the pick phase. This was implemented as an action server node.
- **Node_Place**: as the name suggests, analogously to the node pick, this Action server node is in charge of the entire **place** routine. This was also implemented as an action server node so that we can dinamically assign goals for the actions of these two nodes.
- **Node_barrelDetect**: this service node detects where the desired place table is by using some Computer Vision techniques on the BGR image obtained by Tiago's camera. This was also implemented as a service node.
- **Node_barrelScan**: this service node, together with the **Node_barrelDetect**, achieves the **Automatic docking routine** part by using Tiago's laser scanner to locate the [x,y] coordinates of the three barrels/tables where the objects are to be placed. This was also implemented as a service node.

Here below the program flow is described according to the indices in the Block diagram:

1. **Node_a** calls the service **Human node** by setting the flags, of the "Human" service, 'ready' and 'all_objs' to True.
2. The **Human_node** service response contains an array of indices (1, 2, 3) in a random order: this defines the order for which the objects have to be picked and placed.
3. Having the right order for which Tiago must take the objects, **Node_a** starts a for loop that iterates the obj_ids array returned by the service. For each ID the same pick-place actions are valid. It moves in front of the pick-table in the first room in a specific position (pre-planned) according to the obj-id and then calls the **Node_pick** Action server sending the goal as the ID of the wanted object in order to accomplish the pick phase.
4. **Node_pick** Server calls the **get_tags_pose** service with the objective to obtain the Tags coordinates in order to proceed with the grasp phase. **Node_b** advertises the **get_tags_pose** service which obtains the desired AprilTag detections from the *tag_detections* message and then it converts them according to the base reference frame (*"map"*).
5. **get_tags_pose** service returns the obtained data setting `res.tagsPose = baseFrame_tags`.
6. **Node_pick** computes the collision objects using the ID and the Tag's information of the object to be picked, and add the collision objects to the MoveIt planning scene.
7. **Node_pick** it's then ready to compute the **pick phase**. The sequence by which the pick is performed as indicated in the assignment but with one difference, the removal of the object from the collision objects is done before the pick and not after as indicated. The position of the arm before calling the pick function is slightly different for the three objects, the various configurations for the arm were obtained with the aim of placing the gripper facing down and the arm up. When the first object to be grabbed is blue the time required by the planner is strangely high while when it is grabbed second or third it turns out to be faster. The grabbing of the blue object is from above with a specific function (to avoid abnormal behavior that sometimes occurs with the function used for objects with IDs 2 and 3) and not from the side, a function for grabbing from the side was created (and is still present commented out) but while it worked it did not respect the collisions present most of the time, so it was commented out.
8. Once the pick-routine has been accomplished the action result is set to True in order to give back control to the main node.
9. At this part of the flow Tiago has the desired object grasped in the hand, with the arm in rest position. Now with the use of way points, in order to not collide with the pick-table (since the laserScan only detects the table leg) the robot moves in the second room in a specific position and orientation (*table_room*): with the Camera and Laser, Tiago has to dinamically detect the 3 coloured tables perfectly in front of it. This carries out the **Automatic docking routine** task.

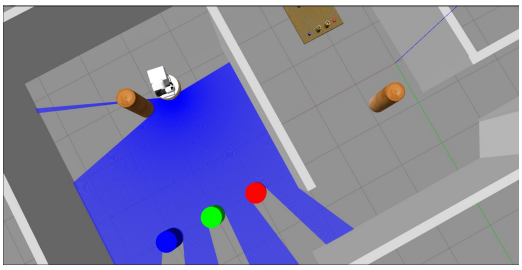


Figure 2: Table room pose

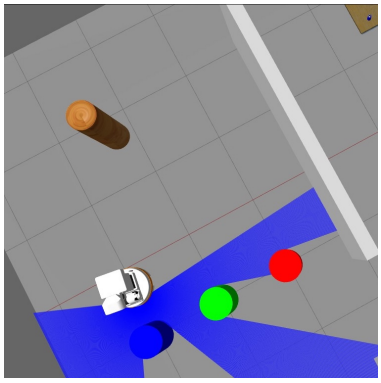
10. From the *table_room* point of view, the **Node_a** wants the robot to detect the centers [x, y] coordinates of the RGB tables, in order to pick the right one (associated with the object-in-hand color) and proceed for the place phase. It, then, calls the service **get_barrels_pose**.

11. This service complete the sub-task of locating the barrels center points using the laser scan data and returns the desired coordinates.

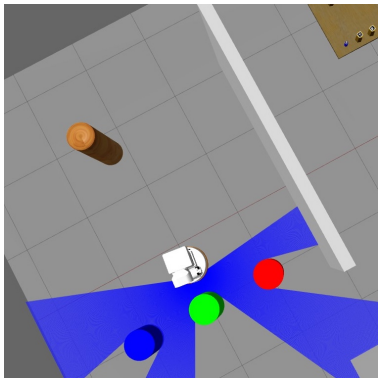
12. Having now the tables locations, we need to know which is the correct one. For this purpose the **Node_extra** is called. It takes in input the obj_id decided previously from the Human node.

13. **Node_extra** performs a segmentation according to the object color, it identifies in which part of its point of view (right, center, left) he has to direct Tiago, and thus it returns to Node a the location_idx.

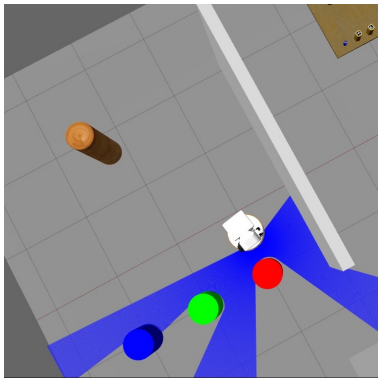
14. Having the location_idx returned by the **Node_extra**, it's possible to map this index with the actual location of the centers. The target position is then computed by an alteration to the y (so the target position is a bit far from the table so Tiago doesn't crash into it). Once this information is obtained, **Node_a** sends the goal (passing i the obj_id and the target position) to the Action server of **Node_place** to perform the place routine.



(a) Table blue pose



(b) Table green pose



(c) Table red pose

Figure 3: Coloured tables poses

15. **Node_place** computes the collision objects, then it adds to the MoveIt planning scene.

16. The **place phase** follows the order indicated by the text of the assignment. This time the arm is placed considering the reference frame base_link, the position we want it to reach then is correct only taking into account the position of the robot Tiago in front of the colored tables we defined, as it follows the logic of placing the arm at a certain distance from him (with a specific orientation).

17. Once the place action has been completed the action result *placed* is set to True and then the server returns the control to **Node_a**

18. Once the whole place routine has been completed, in summary, the robot has selected the right object to pick, moved to the room with tables, computed the centers positions of the tables, picked the right table to place the object on, moved towards it and placed it. So a whole pick-place routine has been completed and now the robot can repeat the iteration by restarting the loop from the *after_obstacle* (if some pickable objects are still on the table).

3 Automatic docking routine

This task consists in using the laser data to detect the position, for the docking routine, in front of the cylindrical tables during the object place phase. In order to achieve this task we could rely also on other sensors, rather than just laser scan. We aimed to take advantage of the Tiago's RGB Camera. The only assumption that we made in order to achieve this is that the robot has to reach the `table room` pose (Fig. 2). To start the docking routine Tiago will have to detect the position of the barrels where to place the objects using the laser scan data. This was accomplished by the service node **node_barrelDetect** with the following steps:

- Once the robot is in the table room pose, the node obtains the latest laser scan data from the topic `laser_scan`.
- Takes this data and removes data points that are too close, too far or too much on the sides
- Computes a first order discrete derivative on the clean laser data vector, this works like an edge detection algorithm for computer vision. It computes the value difference of neighbouring data bins from the laser scan.
- Cleans the newly created difference vector by setting everything below a threshold to zero, so to have only very strong edges with non zero values. This enables us to have a vector where only the right edges of objects within a certain range are present.
- It then takes the original distance value corresponding to these edges and restores the following values within a threshold thus restoring the points corresponding to the object.
- Finally having obtained a vector where only the distances corresponding to an object are non zero, and assuming the tables/barrels are round we interpolate the first, halfway and last detected points into a circle and calculate the center [x,y] coordinates of the three barrels.

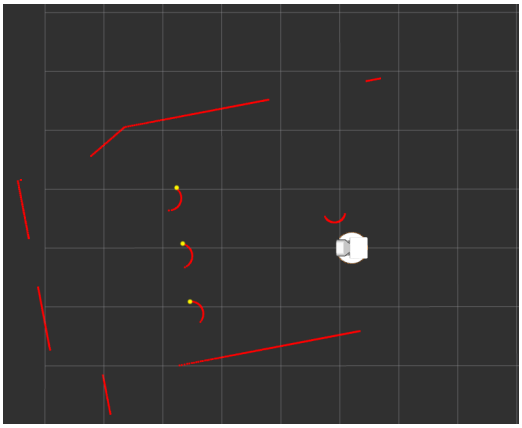
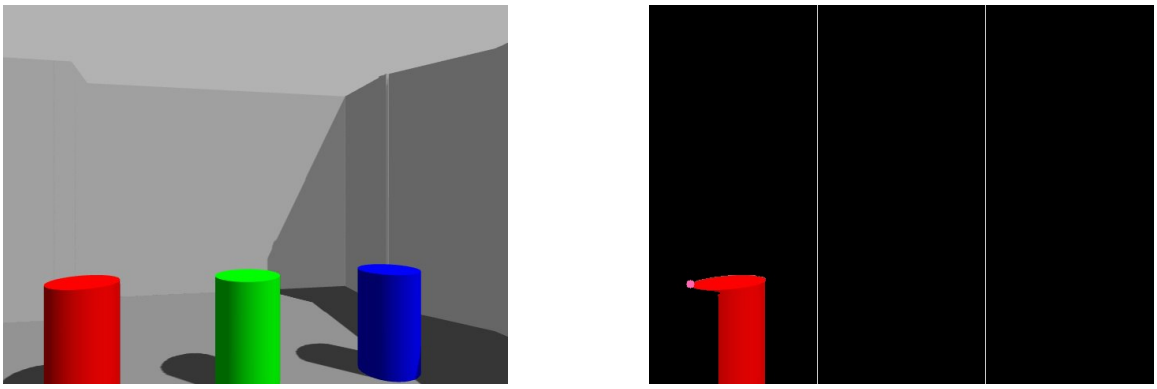


Figure 4: laser scan from table room pose, yellow dots are the detected edges

Now that the center points of the three barrels were obtained, the **node_barrelDetect** performs simple computer vision steps in order to choose on which table the picked object has to be placed:

- Once the server in charge of the movements returns a success value after the "move to table room" command, the robot initialize the RGB camera and uses its point of view to capture an image.
- Once the image is obtained, an image with 2 vertical lines is created. This image represents the three regions where the three barrels can be (left, center, right).
- Now the camera image can be segmented. This is accomplished knowing the ID of the object that needs to be placed (thus the color). For example, int the case of the red object, the Robot's point of view is scanned horizontally and a threshold based on the color is applied. This results in an black image with only one non zero region corresponding to the pixels of the desired color.
- Once the segmented image is obtained, to finally define the direction, another horizontal scan is performed: when a non-black pixel is found the algorithm compares the x-y pixel's position with the two vertical lines x coordinate. Based on this comparison we finally determine in which region the colored set is in.



(a) Robot pov from table room pose

(b) Segmented point of view

Figure 5: Robot's point of view before and after segmentation

Finally having the centers of the three tables/barrels and having identified where the desired table is we can obtain the center position of the table onto which the object has to be placed. The centers are mapped in an array in the following order: blue table center, green table center, red table center. In this way we can use the index returned by the **node_barrelScan** to obtain the center of the wanted table, and thus proceed with steps 14.