Report Assignment 2 - Group 27

Bejaj Xhaku
Fonsato Andrea
Sambin Luca

February 4, 2023

Introduction

To realize the assignment we implemented an action infrastructure. It is designed into 3 nodes:

- **Node_a** : used to send useful information to the other nodes and to implement the navigation control. We implemented the bonus part in this node.
- **Node_b** : performs the detection of apriltags but also the head and torso motion control.
- **Node_c** : implements the pick and place routine using Moveit! library.

Node_a

Node a receives the sequence of IDs to pick given by the human node and sends it to node b as an action goal. Moreover, in node a we have all the information needed to node b and node c to perform their tasks. Using the *updateVariable* function, we define: a fixed pose for all the objects we want to pick a detection point for the extra point, and a waypoint in order to avoid the cylinder at the end of the narrow corridor. We use the *pick_place_object* and *marker_detection* functions to send action goals to node c and node b respectively, in this way we control when the 2 tasks will start. The best solution is to start when Tiago has reached the table with the objects we want to pick. Of course we need the *checkRobotPosition* function to know if Tiago is at the start of the narrow corridor. If that's the case, then we will call the *move_tiago* function and use the action server, developed in the first homework, to safely navigate to the table. Once in front of the table, we start the detection routine by sending a goal to node b via the *marker_detection* function and store the result. Now we need to send to node c the ID, pose and size of the object we want to pick and of the tag we need to detect. After this, we can call node c via the pick and place function and start the task. Once the pick task is over we can go to the detection point perform a scan and go to the respective cylinder, using the *move_tiago_detection* function, keeping in mind that we need and offset of 0.40m to successfully place the object.

Node_b

Let's explain now the implementation of Node b. First of all the detection task. In order to implement it we store the array of detections provided by the apriltag node, using the *updateArrayTags* function. To make the detected tag poses useful for the pick and place task we need to change the coordinate frame from the camera frame, which is the frame where the apriltag node publishes the detections in the */tag detections* topic, to the robot frame, i.e. the *basefootprint* frame; here the designed transformPose function comes in handy. In order to detect the tags, we need to move the Tiago's head with *move_head* and *build_head_goal* functions. The former implements a simple action client, taking as input two variables to move the head right and left or up and down, respectively, and send those as goals to the *head_controller*. The function *build_head_goal* is used to define the goal trajectory position and velocity for the head to send to the *head_controller* by the previously introduced *move_head* function. The same approach is used for the torso control. The detection proper is done by the *aprilCallback* function. Here, first of all, we move the head so that the apriltag node is able to detect the tags of interest. Then we start the detection and we'll send to node a all the information needed by node c, specified in the *DetectionAction* file.

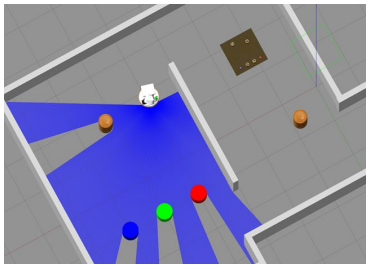


Figure 1: Approaching the detection point

Node_c

This section explain how Node_c is implemented and its principal functions. First of all to allow the gripper's motion we define *move_gripper* to perform the end effector movement and *build_gripper_goal* which generates a trajectory given a waypoint. Then there are another two functions to open and close the gripper, respectively *openGripper* and *closeGripper*. Since the objects are over a table and mixed between several obstacles, we need to construct different collision objects that allow the robot to avoid collisions. Therefore we implemented two functions: *addPlaceCollisionObjects* and *addPickCollisionObjects*. The former is used to define a collision object for every different colored cylindric table where Tiago has to place the object. While the latter defines the following collision objects: one for the table where the objects are initially positioned, one for every obstacles and one for every object to pick. In particular, for the obstacles and the blue object which have a hexagonal shape, we define a cylindrical collision object. For the green triangle, we define a cone collision object instead for the red cube we define one with a cubic shape.
To compute approach and target poses with respect to the gripper, we define a support function: *computeObjectApproachPosition_Y_angle* to define poses of all the three objects while z coordinate is used only for the orientation. Angle's values for x and y are predefined while for z we have specified default values, subsequently is applied a correction given by the detection. The computation of these angles is done following the reasoning in the last page of this document. There are other three important functions: *perform_linear_motion* that allow to execute a linear motion from an initial pose to a target pose, *perform_motion* instead executes a movement given a target pose and then *perform_joint_motion* which sets joint's values.
Now we explain the two main functions that allow to execute the pick and place routine: *pick_object* and *place_object*. Let start with *pick_object*. This function begins by saving objects and obstacles dimensions, positions and IDs specified in the *PickPlaceAction* file. Calling the *addPickCollisionObjects* function, it defines a collision object for every object detected and for the table. At this point an initial configuration is assigned to the Tiago's arm. Then the arm moves to the target pose performing the following routine: it moves to an approach pose staying at 5 cm for red and green object or at 10 cm for the blue one. Then, through a linear motion it reaches the target pose and again with a linear motion it goes in the grasp approach pose. At this point, it completes the grasping and the picked object is removed from the collision objects vector. The object is attached to the gripper with *attachGripper* function, checking the ID to determine which object we have to attach, and then the gripper is closed. Finally, the arm passes through an intermediate pose and then it moves to a secure pose, doing a joint motion, to avoid collisions before navigate to the destination. All of these motions avoid any type of collision with obstacles.

Now let describe the *place_object* function. It starts calling the *addPlaceCollisionObjects* method to define a collision object for the cylindrical place table. Then, after the robot has arrived in front of the destination, it checks the object's ID and performs two different kinds of placing. If the object is the blue hexagon, the function performs a front approach. Instead, if the object is the green triangle or the red cube, it performs a top approach. When the movement is completed, *openGripper* is called to open the gripper and *detachGripper* function detaches the object from the end effector. Finally, the arm returns to a safe pose so that it can pick a next object, and the cylindrical table is removed from the collision object vector. The pick and place proper is done by the *PickPlace* function. Here there are some checking conditions, given by two boolean variables in *PickPlaceAction* file, to decide if the program has to perform the pick or the place routine. Both of these two routine are implemented here in this node. But it could also happen there is an error in the goal and so the function is aborted telling the type of error.

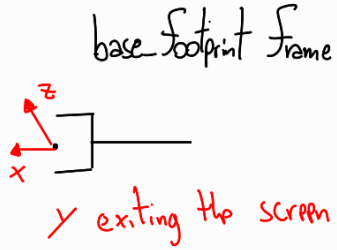
Bonus part

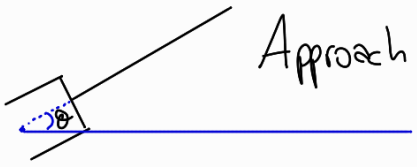
We implemented the extra point in node_a. For this purpose we modified the *server_node* and the TiagoAction file designed in homework 1. In particular we modified the *publishObstacle* function, so that it transforms from *base_laser_link* frame to *map* frame and we added a *detection* field in the action. It will be set to true since we use the laser data for the docking routine, as requested.
Let move back to node_a. Once we've picked the object, we go to the detection waypoint position, previously defined in the *updateVariable* function. Now, by means of the *move_tiago_detection* function, we look for the target cylinder using the modified *server_node*. The information published by this node, with the addition of the height of the cylinder obtained from gazebo, will be used by node_c to create a collision object for the place task. To obtain the place position, we apply an offset of 0.4m plus the radius to the result published by *server_node* in the function *find_cylinder_position*. For the place task we have to distinguish 2 cases: if we want to place the blue object we'll approach the cylinder from the front, otherwise for the other two objects we will approach from the top of the cylinder.

Approach procedure

Blue object:

The gripper approaches from the side of the object:





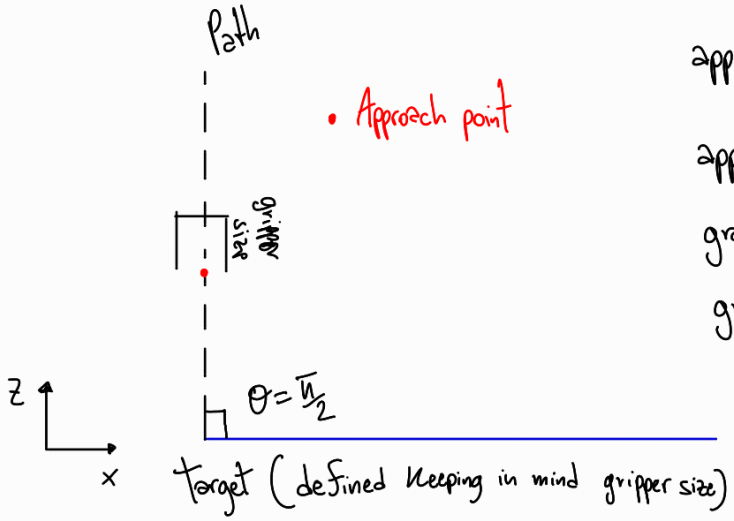
Approach pose: Perform a positive rotation of angle θ in Y

Rotation to perform:

R	P	Y
$\frac{\pi}{2}$	$\frac{\pi}{8}$	0

Green object and red object

For the green object we apply the same reasoning as above but we need to add a rotation in Z. We have default values for roll, pitch, yaw angles, but for yaw we need the values given by the detection in order to success in the task.




```
approachPose.x -= cos(aAngle_Y) * (gripper_size + distance)
approachPose.z += sin(aAngle_Y) * (gripper_size + distance)
graspPose.x -= cos(aAngle_Y) * (gripper_size)
graspPose.z += sin(aAngle_Y) * (gripper_size)
```

	R	P	Y
Green object	$\frac{\pi}{2}$	computeObjectApproach_Y_angle	sin(aAngle_Y)
red object	$\frac{\pi}{2}$	computeObjectApproach_Y_angle	sin(aAngle_Y)