Final project presentation

Group_01

Francesco Bordignon, Giuseppe Labate, Greta Piai

Assignment 1

Tasks we had to accomplish

1) Create a **callback based client node** that takes the user's input and, using an action file, communicates with the server to guide the robot to the correct location and receives the results

2) Create a **server node** that would receive the client's request to move towards a certain point, detect the number of obstacles and return the result to the client

The callback based client

This node aims to retrieve the input from the user (i.e. the coordinates of the position and orientation on the map that the user wants the robot to reach in order to start the detection). It is based on SimpleActionClient.

It waits for the action server to start, after that it sends the objective coordinates to the server through the goal field of the associated action file.

While waiting for the final result, the client eventually shows on terminal feedbacks that the server sent while approaching to the goal (doneCb, activeCb, feedbackCb).

In the ends, it prints the number of obstacles and their position and orientation on the map.

Action server: reaching the goal

To reach the goal, the node takes the desired goal in the action file and saves it in a geometry_msgs PoseStamped message.

Then the node publishes it in the ***/move_base_simple/goal*** topic, which sends the robot to the desired pose.

Giuseppe

Action server: reaching the goal

During the travel, the node ***subscribes*** to ***/move_base/status***, a topic that sends at each time frame a ***feedback*** value describing the robot status.

We exploited and remapped it to transmit to the action file and so to the client the feedback of the process.

1. goal received
2. goal cancelled because another goal was received
3. goal reached
4. invalid coordinates, goal cancelled

10. the detection started
11. the detection ended

Action server: the detection

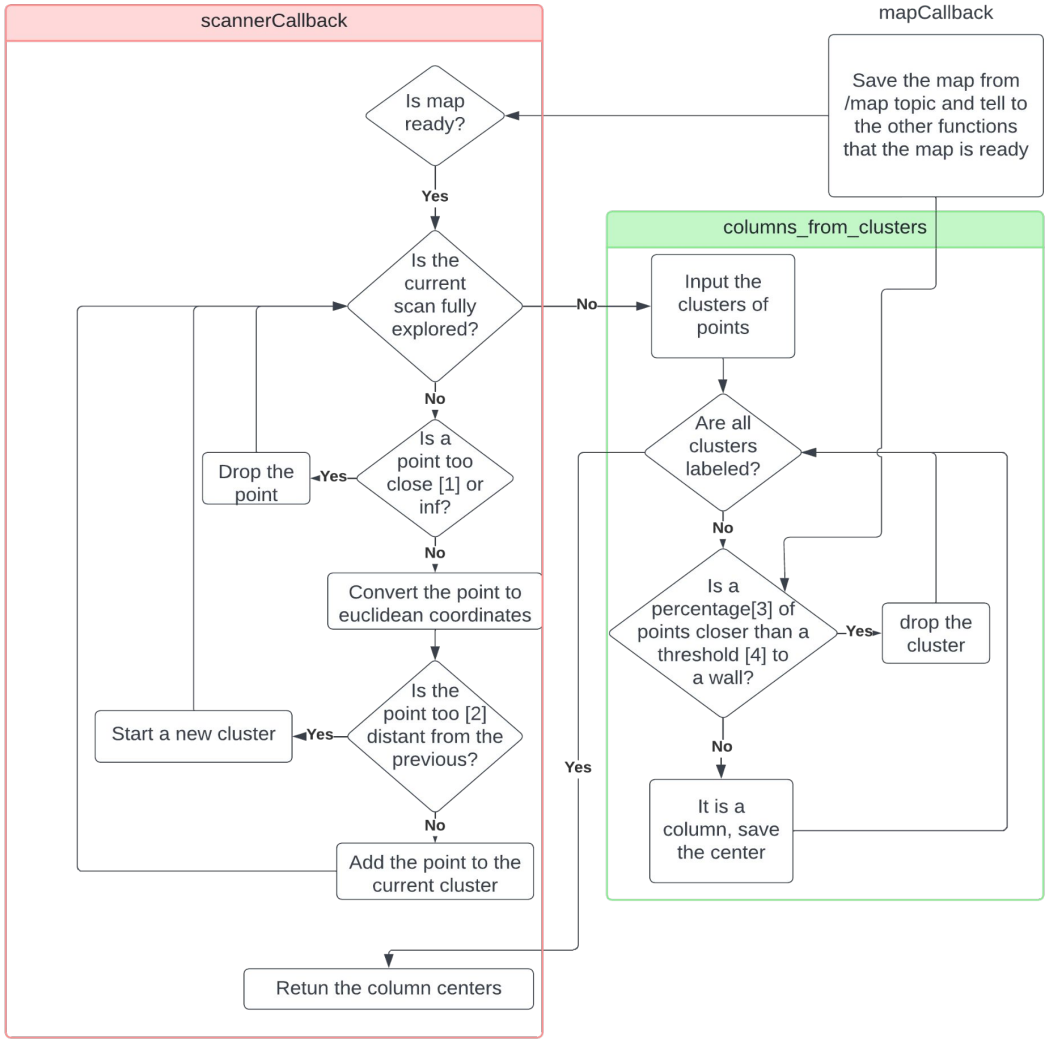
Parameters:

[1] 20cm

[2] 20 cm

[3] 10%

[4] 16 map cells



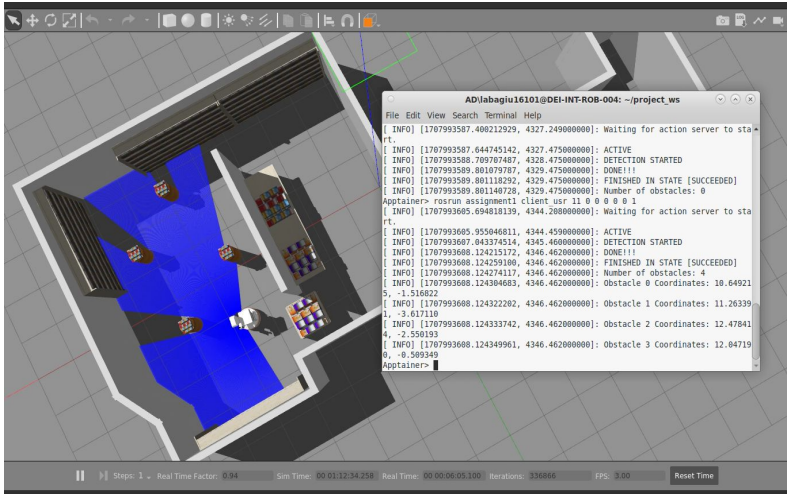
Action server: the detection

Advantages:

- Fast execution
- Walls always ignored
- Column detection also in case of low visibility

Disadvantages

- Doesn’t recognize a column if too close to a wall
- Doesn’t recognize attached columns as different columns from certain perspectives

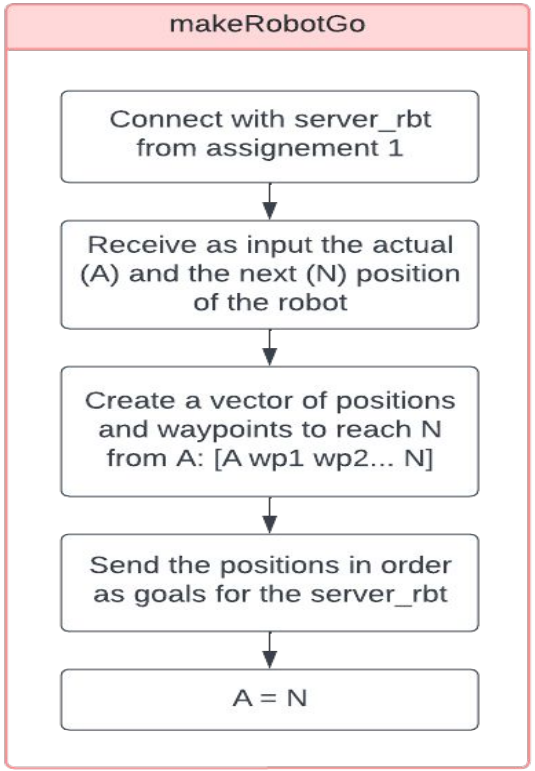


Assignment 2

Tasks we had to accomplish

1) Retrieve order of objects to pick from human node
2) Bring robot to a hard coded pose on the map where it could easily grab the object on the table (paying attention to not hit either the table or other obstacles)
3) Elevate the robot, lower the head and do the AprilTag detection
4) Define the collision objects with the data obtained from the detection
5) Perform an arm motion that allows to approach the object to pick
6) Pick the object
7) Navigate to the destination cylinder (again, considering the obstacles on the path)
8) Place the object
9) Repeat from 2)

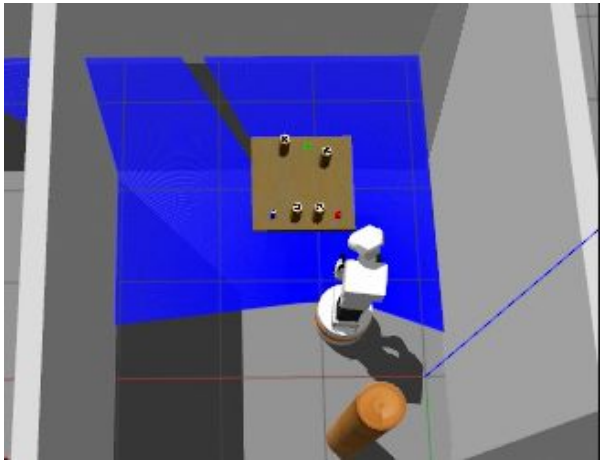
Bringing the robot close to the table/cylinders



Node Assistant (A) communicates with the Human node and makes tiago go in the positions given by the human passing through waypoints in order to avoid collisions with obstacles.

Positions:

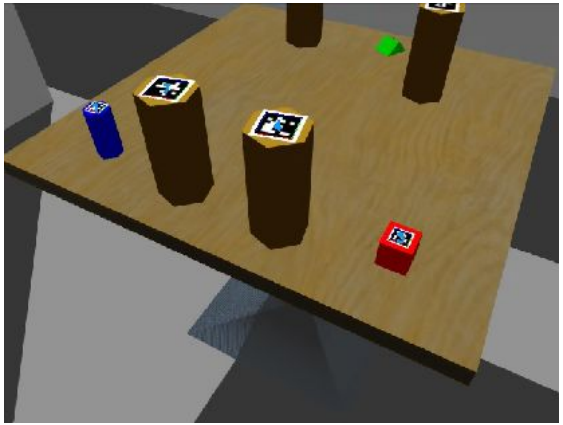
- origin
- pick_red
- pick_blue
- pick_green
- place_red
- place_blue
- place_green



AprilTag detection

Upon reaching the designed pick and place position, the robot ***configures*** its ***head*** and ***torso*** elevation for optimal detection.
To do so, we send the goal position of the desired joints to the "joint_name"/follow_joint_trajectory SimpleActionServer

After that, the pick and place node acts as a service-client and sends a ***request*** to the aprilTag detection node to start the detection.



AprilTag detection

The node subscribes to the ***/tag_detecions*** topic, which contains the ID and pose information for the detected aprilTags.

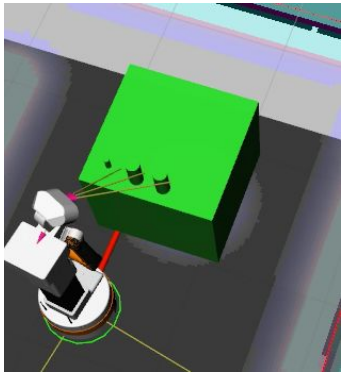
Utilizing the tf2 library, this information is ***transformed*** from camera reference frame to world reference frame.

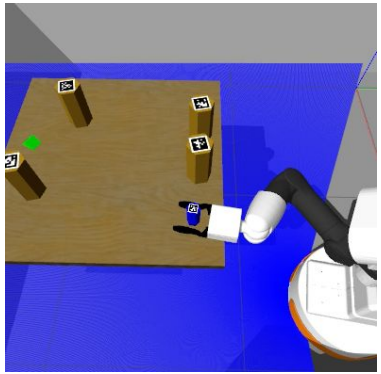
Finally, the transformed data is sent as a ***response*** in the form of a geometry_msgs::PoseStamped ***array*** in which each cell corresponds to a specific tag ID, holding its corresponding pose within the world reference frame.

Greta

Arm motion, collision objects and gripper

After the detection is complete, all the perceived collision objects (except for the table which position and dimension are hard-coded) are ***added to the map*** with the positions' values obtained from the AprilTag detection.

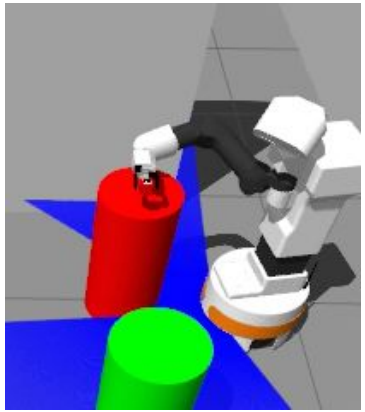




It then proceeds with the ***motion planning and execution*** of the arm in order to approach the object to pick (that is a couple centimeters above the real AprilTag position of the object). Finally, the arm lowers down and the ***gripper closes***.

Arm motion, collision objects and gripper

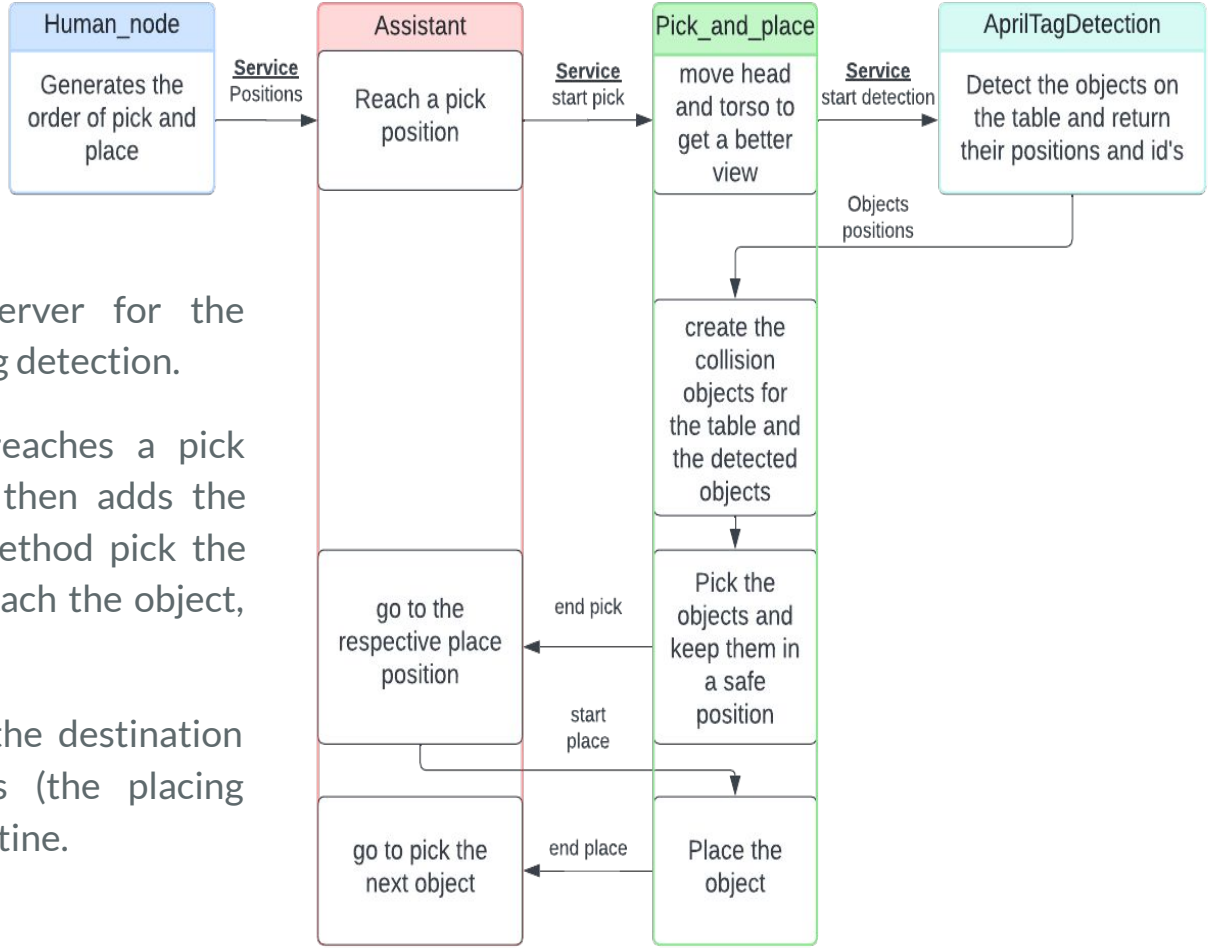
We hard-coded a ***safe position*** for the robot's arm (that is the starting position) to reach every time it has picked an object and has to move to the destination cylinder.



Tiago robot in safe position

When it reaches the cylinder, it performs an arm movement that brings the gripper on top of the cylinder's center and it ***drops the object by simply opening the gripper***.

Communication



Node pick_and_place acts like a server for the Assistant and as a client for the AprilTag detection.

It is queried everytime the robot reaches a pick position and it starts the detection, then adds the collision objects, performs through method pick the arm's movements that allows to approach the object, returns the result to the Assistant.

Finally, the robot moves towards to the destination cylinder, adds new collision objects (the placing cylinders) and performs the placing routine.

Bug correction

In the final version of our code we had a ***bug*** in which the detection of the third object couldn't start and so the robot couldn't pick the final object.

We corrected the bug the next day, by simply adding a ***iteration = 0;*** line in the aprilTagDetection.cpp file .

By resetting the iteration count, we can enter in the ***while loop*** and the execution of the final detection can start and all the pick and place routines are perfectly done!

```
bool coordinates(assignment2::tags::Request &req, assignment2::tags::Response &res)
{
    bool start_detection = req.start;
    if(start_detection)
    {
        iteration = 0;
        ros::NodeHandle n;
        ros::Subscriber sub = n.subscribe("/tag_detections", 200, callbackFunction);
        while(iteration < 15)
            ros::spinOnce();
        for(int i = 0; i < 8; i++)
        {
            ROS_INFO("-----------------array [%d] =  ", i);
            ROS_INFO("Orientation: (%f, %f, %f, %f)", output[i].pose.orientation.x,
                     output[i].pose.orientation.y, output[i].pose.orientation.z,
                     output[i].pose.orientation.w);
            ROS_INFO("Position: (%f, %f, %f)", output[i].pose.position.x,
                     output[i].pose.position.y, output[i].pose.position.z);
        }
        for(int i = 0; i < 8; i++)
            res.poses.push_back(output[i]);
    }
    return true;
}
```

Thank you for your attention!