Group 01

1. Giuseppe Labate, giuseppe.labate.1@studenti.unipd.it
2. Greta Piai, greta.piai@studenti.unipd.it
3. Francesco Bordignon, francesco.bordignon.5@studenti.unipd.it

Repository link:

https://bitbucket.org/prova123456/ir2324_group_01/src/master/

To clone the repository:

git clonehttps://arcopigreta@bitbucket.org/prova123456/ir2324_group_01.git

Link to the video:

https://drive.google.com/file/d/1kKeXrSEbDEl18r2ya1eEHq4I9bm6XwHz/view?usp=sharing

To run:

At the beginning of each terminal do:

```
$> start_tiago
$> source /opt/ros/noetic/setup.bash
$> source /tiago_public_ws/devel/setup.bash
$> source /package_path/devel/setup.bash
```

- In the first terminal
  start the simulation, the navigation stack, MoveIt and the AprilTag with our custom launcher:
  ```
  $> roslaunch assignment2 project2.launch
  ```

- In the second terminal
  start the Assistant.cpp node:
  ```
  $> rosrun assignment2 Assistant
  ```

- In the third terminal
  start the HumanService node:
  ```
  $> rosrun tiago_iaslab_simulation human_node
  ```

- In the fourth terminal
  start the server_rbt.cpp node:
  ```
  $> rosrun assignment1 server_rbt
  ```

- In the fifth terminal
  start the *pick_and_place.cpp* node:
  ```
  $> rosrun assignment2 pick_and_place
  ```

- In the sixth terminal
  start the *aprilTagDetection.cpp* node:
  ```
  $> rosrun assignment2 aprilTagDetection
  ```

Code explanation:

Node details:

AprilTagDetection:

The AprilTagDetection node is built as a service server node.
It receives from the pick_and_place node a request that contains only a bool, which, if set to true makes the detection start.
The node subscribes to *"/tag_detecions"*, a topic containing the detections of all the aprilTags' IDs and corresponding positions and orientations.
These positions are taken with respect to the camera reference frame, so it needs to convert them with respect to the global frame.
To do so, the callback exploits a custom helper method named *"transformObjectCoordinates"* which transforms the detected pose to whatever reference frame the node needs.
Specifically, it transforms the detected tags' poses from *"/xtion_rgb_optical_frame"* to *"/odom"* using the *tf2* library.
After that, each position is saved in an array of *geometry_msgs PoseStamped*, in which the cell index corresponds to the ID number of the corresponding tag's pose.
Then the program returns the filled array of positions as a response to the client node.

Note that for each request some cells may be empty because the corresponding aprilTag isn't in the camera frame.

Assistant

This node is built as a service client that asks for the pick order of the objects to the Human Node service server.
After correctly receiving the order, the node plans a path to reach each docking point from which the robot can grasp or place each object.
The node can ask the *pick_and_place*.*cpp* node to pick or place a detected object, by sending to it a request containing a *"bool ready"* and an *"int32 action"*.
The first, if true, asks the robot to start one of the six actions defined in *action*:

- For action = *1, 2 or 3*, it asks the pick of respectively *blue, green and red* objects
- For action = *4, 5 or 6*, it asks the pose of respectively *blue, green and red* objects

To navigate from start to picking spots and from there to placing spots, the node exploits the *server_rbt.cpp* in *Assignment1*, by sending to it (as a simple action client) the position where the node wants the robot to go.
To avoid getting stuck and to be more accurate on the path, the node implements some waypoints, placed in strategic positions between start and arrive.

Pick_and_place:

This node functions as a service server, patiently awaiting the activation signal from the *Assistant.cpp* node indicated by the boolean flag in its request. Once this signal is received, the node proceeds to optimize the robot's configuration to enhance the visibility of the specified object and the surrounding obstacles identified by AprilTags.
In order to do so, it uses two methods:

- *"headMovement"* sets the right parameters to move the *head_1_joint* and the *head_2_joint*

- and “*liftTorso”* sets the right parameters to move the *torso_lift_joint*

By moving them, the node can properly frame the aprilTags that it needs.
The node is also able to communicate, as a service client, to the *aprilTagDetection.cpp* node to start the detection.
This is done by sending the request bool to *true.*

After the detection has finished the node creates collision objects in correspondence of the detected poses and adds to the scene plan the collision objects of the table and the placing cylinders.
In this way, when the arm is actuated, by planning the execution, it can avoid those obstacles.

Now that it can safely move the arm, it exploits an intermediate position (the same for all the picking docks) in which it moves the arm by moving each of its joints in a high-ground position.

After that, the arm is sent to the found position of the desired object, the desired object is removed from the collision objects and the gripper is subsequentially closed using “*closeGripper”*.
At the end of this sequence the object is attached to the gripper, by exploiting the “*attacherGripper”* method.
Successively the robot assumes the intermediate configuration and then the safe position(*“safePosition”*), which has been chosen to be the robot’s starting configuration.
Next, the response (containing *true* if the robot has succeeded or *false* otherwise) is sent to the *Assistant.cpp* node, which starts the robot navigation to the placing cylinder.

Once arrived at the spot, the robot starts a placing routine where the movements of the arm’s joints have been hard-coded and properly tuned.
Afterwards, the object is detached(*“detachGripper”*) and another response is sent to the *Assistant.cpp* node, which, consequently repeats the pick and place routine.

Summary of the execution

The execution starts from *Assistant.cpp*.
This node takes the pick order sent by the Human Node and sends the robot to the corresponding picking spot.
After that, the *pick_and_place.cpp* node positions the robot in the best configuration to pick the current object.
It does so by moving its head and lifting its torso.
In the next step, the *pick_and_place.cpp* asks the *aprilTagsDetection.cpp* node to start the detection.
At the end of it, *aprilTagsDetection.cpp* sends to *pick_and_place.cpp* the poses of the aprilTags detected with respect to the world reference frame.
Now these poses are taken by *pick_and_place.cpp* and used to construct the collision objects in the scene.
In this way, the node can send the arm in the intermediate pose and then can send it to pick the object, without colliding with the table and to detected objects.
After that, the node attaches the object to the closed gripper and the robot assumes the safe position.
Then the control of the execution comes back to the *Assistant.cpp* node.
The latter sends the robot to the corresponding placing cylinder and re-communicates with the *pick_and_place.cpp* this time to place the object.
The node moves the robot arm joints to place the gripper above the cylinder.
Now the gripper is opened and the object is detached.
Finally, *pick_and_place.cpp* communicates to *Assistant.cpp* that the place is done and the latter sends the robot back to the table, this time to the second object docking spot, to restart the routine.