Moratello Matteo, Obetti Christian, Romano Domenico 03/18/2020
Master Degree in Computer Engineering
A.Y. 2019/20 Course of Autonomous Robotics

Report Homework 4

1 Introduction

Industry 4.0 aims to create fully automated routines, therefore without human intervention at no stage. In order to create a fully automated routine, we have used ROS services letting the two robots communicate their states.
In the first part of the routine, Marrtino proceeds to one of the two loading stations using the algorithms of homework 3. Once it reaches a loading station, it sends its status to the manipulator robot, which will load objects onto the mobile robot.
Martino with the objects loaded, continues his path until he reaches the starting position again, where he can start a new run.
The services implemented, and the changes made to the various homeworks, will be explained in detail below.

2 Services

This chapter explains the two services implemented for this homework and that permit robots to collaborate each other.

2.1 User Service

User service allow the end-user to control the unloading of the mobile robot when it is at the start position. It also manages the robot routine, deciding the number of objects to load in the current run and whether to stop the robot when all the objects have been recovered.

2.1.1 Server

The server was implemented in a separate node from those of previous homeworks. The source of the server is *user_service.cpp*.
The server waits a request by a client and expect a string containing the actual state of the robot.
The server responds to the request with two information, given in input by the end-user. For first the server asks the user if the mobile robot has to load some objects and so complete a run or to stop since all objects are been unload.
To do this the user has to write a boolean int: 1 to move the robot, 0 for stop the robot.
Then, the server asks to the end-user to specify the number of objects to load on the mobile robot in the current run.
To do this the user has to write an int from 0 to 7, since each *assembly task* has 7 objects. In this way the end user can choose how many runs to do, for example to transport all 7 objects, one can divide the routine into two runs, first carrying 4 objects and then the remaining 3 objects.
The server sends the two information to the client and wait for a new request. In the simulation, we observed that if we place many objects on top of the Marrtino, the latter becomes not very stable (until it goes crazy), for this reason in our video we made four laps with 2 + 2 + 2 + 1 objects per run.
Figure 1 shows the GUI of the Server.

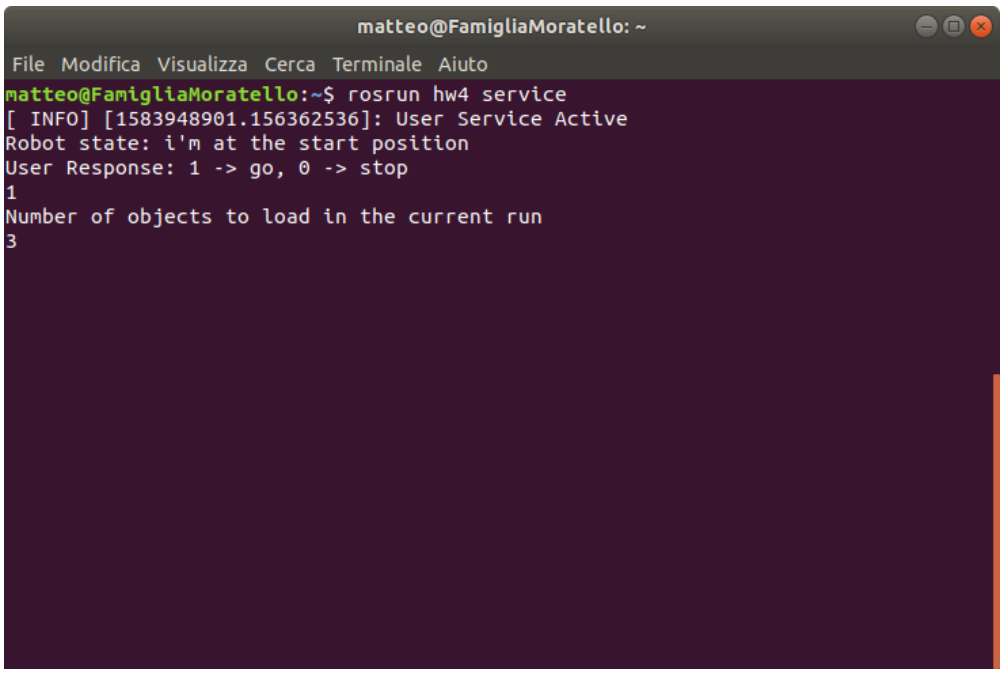

Figure 1: GUI of server-side of User Service

2.1.2 Client

The client is implemented in homework 3. The first call occurs at the beginning of the routine, which then sends a request to know how many objects to load in the run.
The number of objects requested is saved on a variable which will then be used in the second service. If, on the other hand, it receives the command to stop, the homework node 3 is closed.
The call also occurs at the end of each run. The sequence of hw3 has been modified to make the robot turn 180 degrees in its initial position and therefore to be able to face a new run. The end user can then decide how many objects to load on the robot or stop it.
Figures 2-3 show the robot turned ready for a new run.

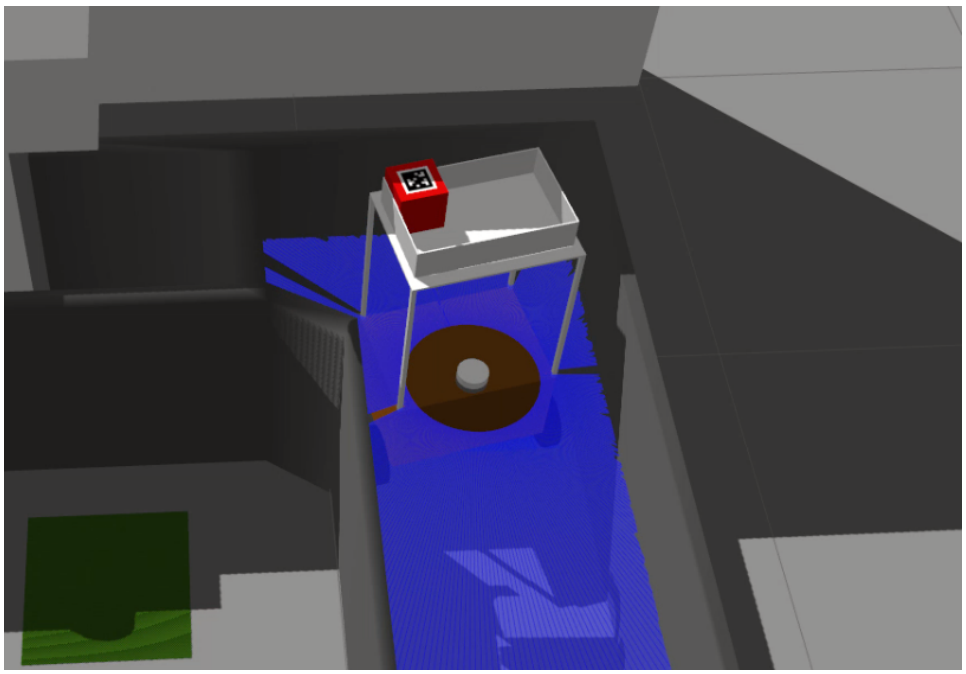

Figure 2: Mobile robot ready to unload the pieces

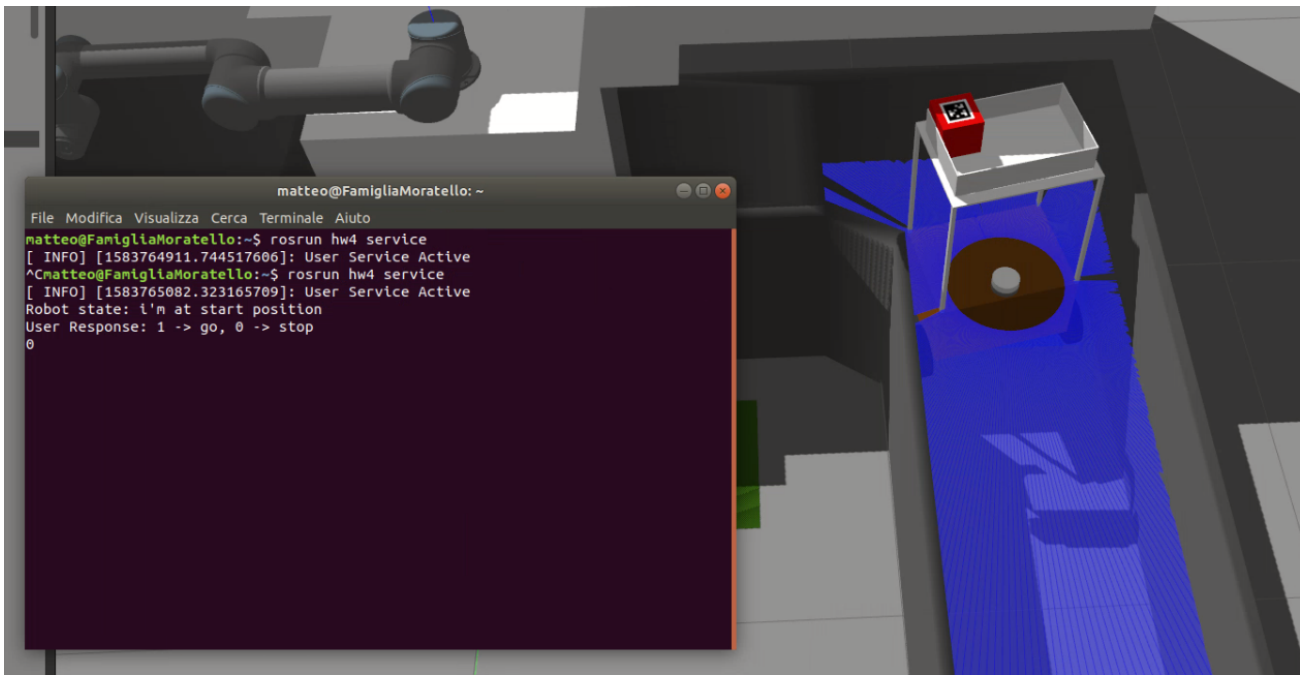

Figure 3: End-user gives a command to stop the mobile robot

2.2 Manipulation Service

Manipulation service allows the mobile robot and the manipulator to collaborate each other in a very simple way. With this service, objects on the table can be placed on the mobile robot completely automatically fashion.

2.2.1 Server

Hw2 has been adapted to host the server. The first part of the program creates, as in the original hw2, an instance of the Manipulation class, setting all the parameters for Moveit.
Subsequently, in a separate thread, the server was implemented, where in the callback is passed the instance of the class together with the services for the gripper, magnet etc.
The server, once a request from the client has arrived, starts the manipulation sequence described in detail in the report of homework 2. The server expects to receive two variables. The first variable is *station*, is a string and specifies the loading station, specifically *load1* indicates the loading station near the initial position, *load2* the one next to it.
The second variable is *objects*, it is an int and specifies the number of objects to load on the mobile robot. The manipulation sequence will then repeat *objects* number of times.
The server returns an integer that specifies the number of objects still to be picked up in order to complete the task.
Figures 4-5 shows the manipulator robot during the sequence.

2.2.2 Client

The client is implemented in homework 3. The call to this service occurs when the mobile robot arrives at a loading station. Since described in hw3, the mobile robot autonomously chooses the free loading station and saves this information. The client (mobile robot) then sends two information: the first is the loading station in which it is located (*load1* or *load2*), the second is the number of objects that must be loaded into the robot's basket, by the User Service we know to be the variable *numberObjectsToLoad*.
The mobile robot then waits for the response from the server, that is the manipulation sequence, and

then resumes the path, first in the open space and then into the various paths to return to the starting point.
Figures 6-7 shows the mobile robot leaves the loading station with objects on top.

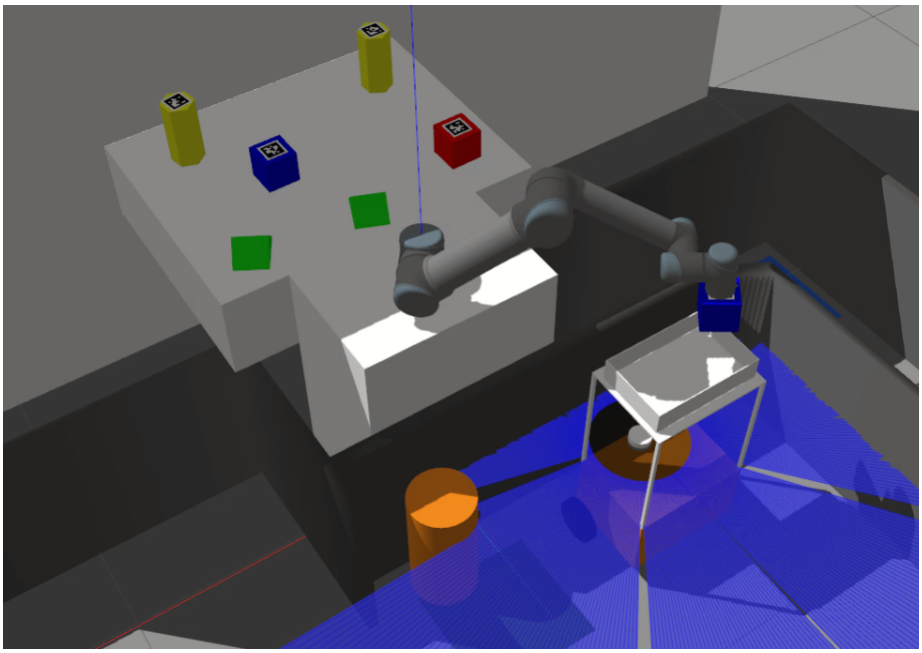

Figure 4: The manipulator robot places a cube on the basket

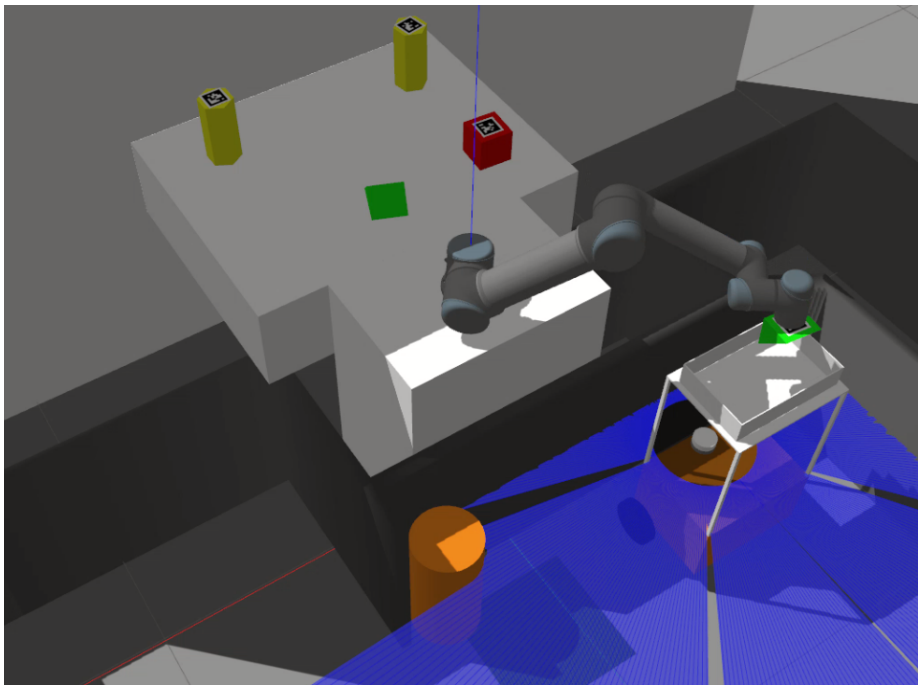

Figure 5: The manipulator robot places a prism on the basket

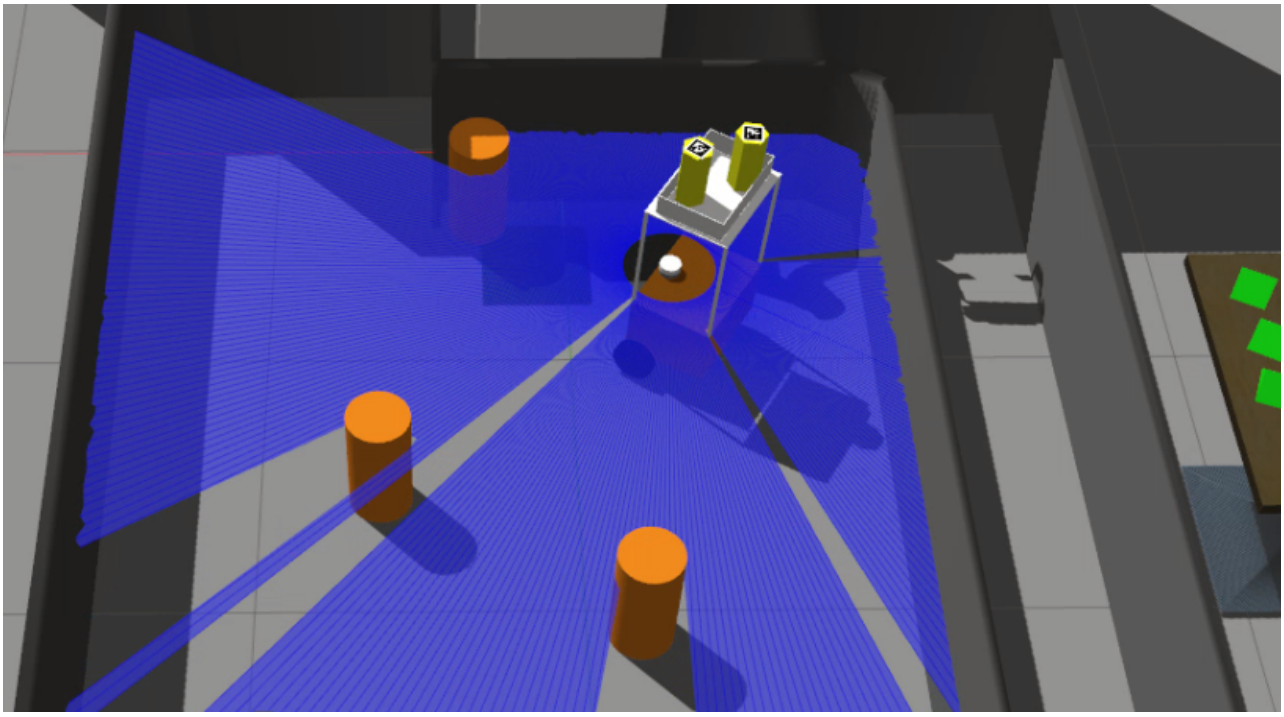

Figure 6: The mobile robot leaves the loading station

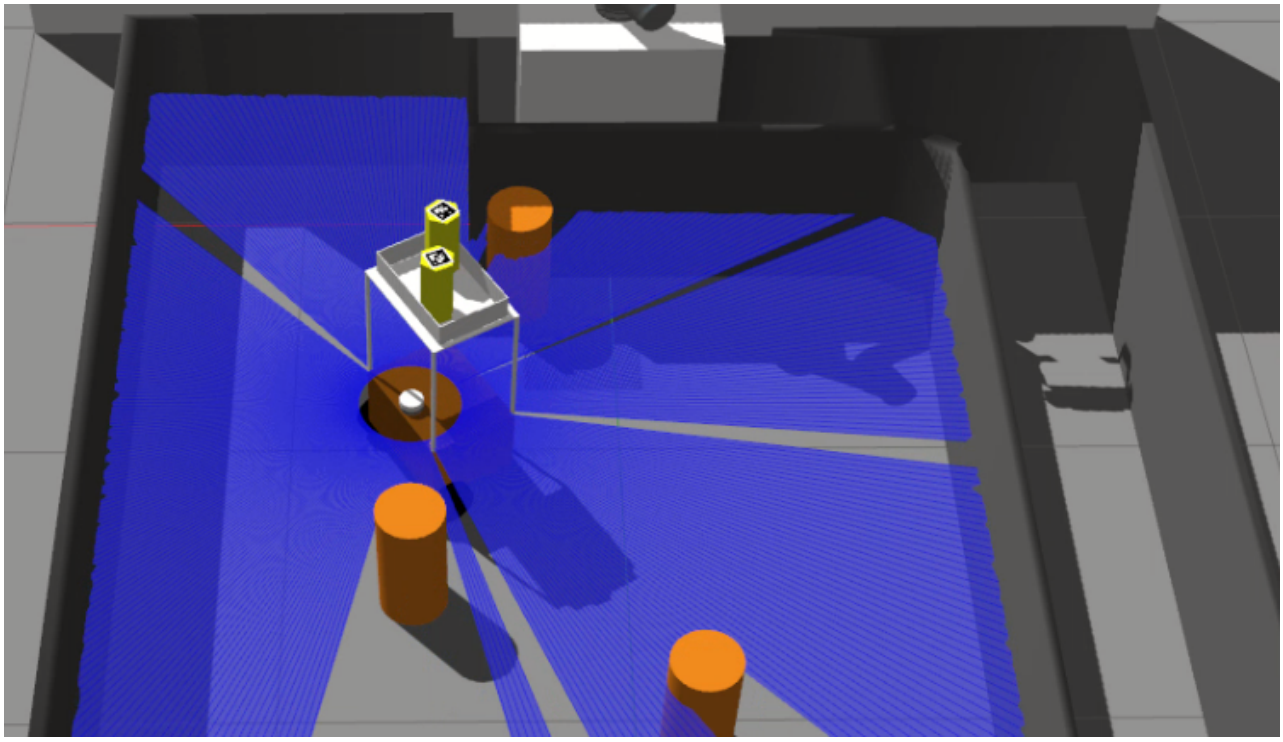

Figure 7: The mobile robot leaves the loading station

3 How to run the code

The following roslaunch command run three nodes corresponding to the first three modified homeworks. The parameter **simulation** permits to set the type of environment: 1 for simulation and 0 for the real one.
The parameter **gripper_enable** permits to set the type of gripper: 1 for hand gripper and 0 for the magnet.

$ roslaunch hw4 FSM.launch simulation:=1 gripper_enable:=0 objects:="blue_cube_0 blue_cube_1 yellow_cyl_0 yellow_cyl_1 green_prism_0 green_prism_1 red_cube_0"

In order to control Marrtino in the unloading phase run, with the following command, the User Service (described above).

$ rosrun hw4 service

4 Conclusions

With two simple services we managed to implement a finite state machine, which in a completely autonomous way is able to manipulate and transport objects within areas populated by mobile obstacles, and this is a typical situation of an industry 4.0.