Software Architecture Project

Group: HRC-02
Students: Matteo Azzini, Enrico Fiasché, Suraj Deshini
Tutors: Simone Macciò, Mohamad Alameh, Alessandro Carfi
Year: 2020 – 2021

Human-Robot collaborative manipulation

Table of Contents

The following report is made up by a brief abstract with the general overview and six sections which present the whole structure of the project.

Abstract

The project is a simulation of Human-Robot collaborative manipulation of objects. In the scene there are a human and the robot, Baxter, one in front of the other, in the middle there is a table with six red blocks, five blue blocks and two boxes on top, the first box is red and it is on the Baxter's right, the second one is blue and it is on the opposite side of the table.
Baxter has to pick the blue blocks and place them into the blue box, while the human is doing the same with the red blocks, in order to put them into the red box, the robot should also avoid the collision with the human.

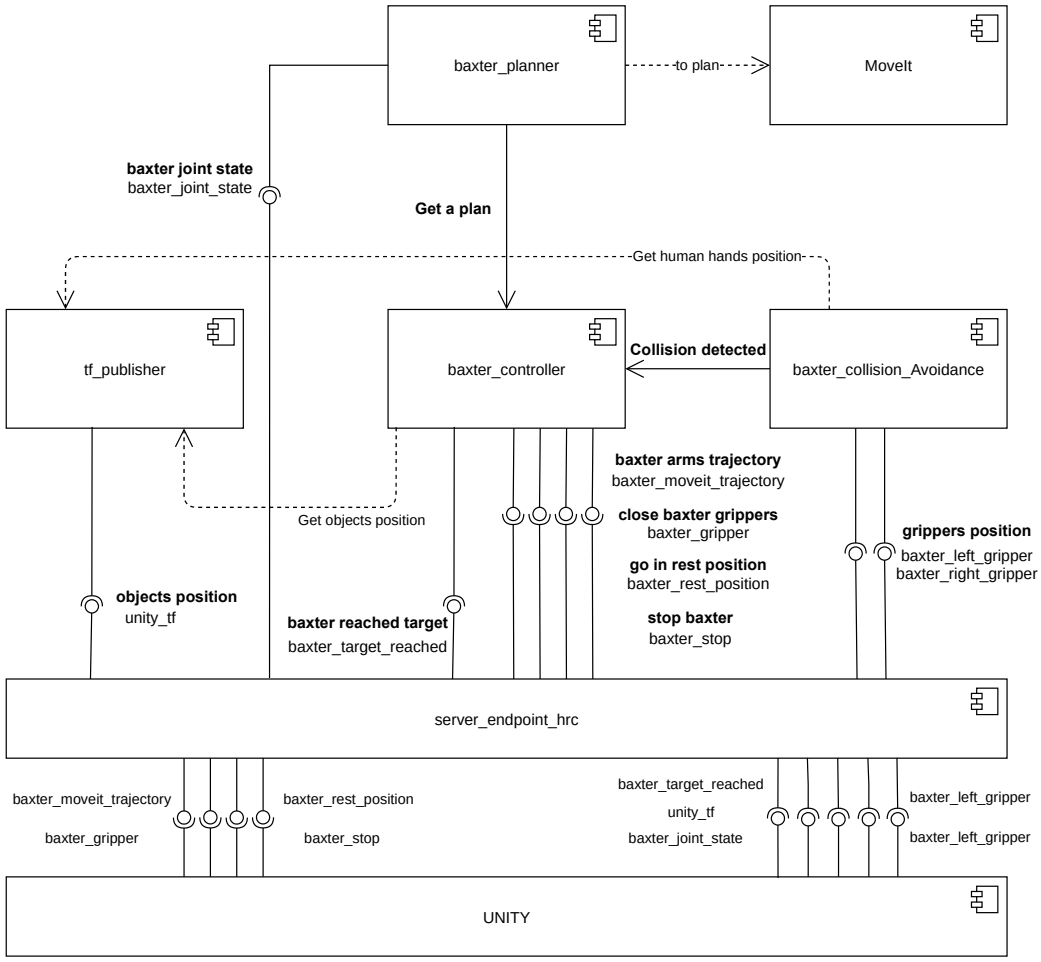
The project is available at the following repository: https://github.com/EnricoFiasche/HRC-02

1. Introduction

The whole simulation is developed on Unity and it should be installed on Windows, we had to develop ROS nodes on a Linux system in order to interact with it and control the robot, the two machines communicate via socket connection.
The aim of the project was to define a software architecture in ROS to control the robot in the Unity simulation, in particular we should plan robot movements using the package **MoveIt** and let Baxter to pick and place the right blocks in a pre-established order, avoiding the collision with the human.

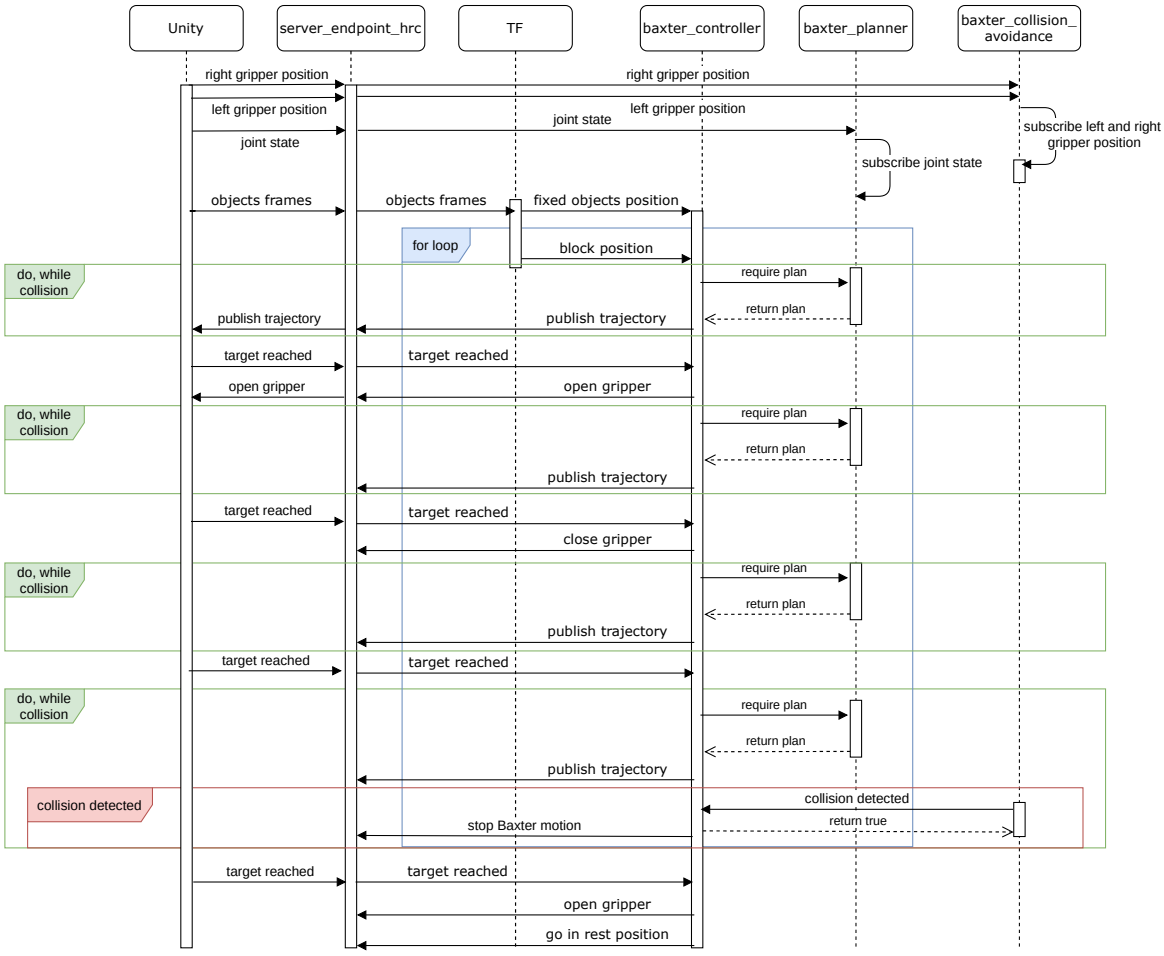
2. Architecture of the System

In the following UML diagram is shown the whole ROS architecture.



The main node is **baxter_controller** which actually controls the robot publishing the trajectory to be performed by Baxter in the simulation in **Unity**, it is also responsible for the opening/closing of the grippers and for the robot motion, such as reaching an object calling **baxter_planner** (it use MoveIt package to plan) to get a plan, coming back into the rest position or stopping it.
In the meanwhile, the node **baxter_collision_avoidance** checks periodically the distance between Baxter's grippers and human's hands in order to detect a possible collision and trigger the controller.
The **tf_publisher** node reads from **Unity** the objects position and makes them available for the other nodes.

Below is available a sequence diagram of the execution of the simulation that show how the modules interact among them, with also an example of collision detected.



3. Description of the System's Architecture

All the following modules are developed on ROS Noetic, available at the link https://github.com/EnricoFiasche/HRC-02/tree/main/ROS_packages.
All the ROS modules required the ROS package **moveit_robots,** to plan and move the robot, and **baxter_common**, **baxter**, which contains the Baxter's description.

3.1. baxter_controller

It is the main module in charge to accomplish the project tasks, it has a priori known order of couples that define the blocks to be picked and the position within should be placed.
The module uses:

- four publishers:
  - On topic *baxter_moveit_trajectory,* used to publish the BaxterTrajectory (arm,tarjectories) to be executed on Unity
  - On topic *baxter_gripper,* used to open and close the grippers
  - On topic *baxter_rest_position,* used to go back in the rest position for an arm
  - On topic *baxter_stop,* used to stop Baxter motion
- a subscriber:
  - On topic *baxter_target_reached*, used to know when Baxter has completed a trajectory
- two services:
  - a client for the service *baxter_planner_service,* used to make a request to the server **baxter_planner** passing the arm and the goal position to be reached, the server compute a trajectory and sends it back as response
  - a server *baxter_collision_avoidance,* used to receive a request when the module **baxter_collision_avoidance** detect a possible collision

First of all, it takes the position of fixed key places, such as the blue box and the middle placement, which is the position where the robot places an object with right hand in order to take it with the left one. Then, for each block the module checks the target position for the block and decides which is the arm to be used, therefore it acquires the transform of the block position listening to the adapter **tf_publisher** and starts the motion routine.

The motion routine is composed by four different trajectories:

- go in pre-grasp position, 15 cm over the block
- go down to reach the block
- lift up back to the pre-grasp position with the block in the gripper

- go to the target position to drop the block

The pre-grasp position is obtained adding 15 cm on z coordinates to the block position, to obtain the trajectory the module makes a request for *baxter_planner* service passing the arm to be used and the goal position, as response it receives the BaxterTrajectory, it publishes this data on topic *baxter_moveit_trajectory* in order to let Baxter reach the position in the Unity simulation.
The topic has a RobotTrajetory[] as type of message, but the module sends just one element at time in the array to have the possibility to update the start joint state before compute any new trajectory, in this way the Unity simulation and the ROS architecture are perfectly synchronized and also the collision avoidance is as fast as possible in trigger the controller.
Before computing the next trajectory, the module checks two variables, one that is updated when a message is received on topic *baxter_target_reached* to know when the robot in the simulation has reached the target, and the other one modified if and only if a collision is detected, in this case the robot is stopped in the Unity simulation publishing on topic *baxter_stop* and the module waits few seconds, then recomputes a new plan if it is possible.
At this moment the module sends a message on topic *baxter_gripper* to open the gripper, then it follows the same mechanism like the one applied for the previous target but with a new goal position, the block. When it is reached the module close the gripper publishing again on the topic *baxter_gripper* in order to grasp the object.
Again the trajectory for the lift up position and target position are computed in the same way, the first one coincide with the pre-grasp position, the second one is the position of the target adding 10 cm on the z coordinate, when this final position is reached, the module publishes on *baxter_gripper* topic on order to place the block inside the blue box.
Finally, having completed the task, the arm goes back in the rest position when the module publishes on topic *baxter_rest_position.*

3.2. baxter_planner

This module is required to use MoveIt package to find a trajectory to reach a target position. It is implemented as a server which receives as request the arm to be used and the final position to be reached and gives back the BaxterTrajectory to reach it. The module has a subscriber on topic *baxter_joint_states* use to know at any time the joint state of the arm and to set the start position for the plan.
First of all the server creates the MoveGroup for each arm and sets the table as a constraint, then when a request is received, it sets the start joint state for the arm that will be used, therefore it sets the target position for the move group of the arm and calls the plan method to find a trajectory.

If the planner has been able to find a trajectory, it creates a BaxterTrajectory and sends it back as service response.

3.3. baxter_collision_avoidance

This module is in charge of detecting possible collision with human hands and arms while Baxter is moving. It has a service client for *baxter_collision_avoidance* service to make a request to **baxter_controller** in case of possible collision.
To check a collision the module periodically subscribes the position of human hands and lower arms, if the distance between one of these elements and one of the Baxter gripper is less than 10 cm, the module makes a request for the previously mentioned service.

3.4. tf_publisher

This is an adapter module that subscribe on topic *unity_tf* to get object positions, then it publishes them on a TransformBroadcaster to make them available for all other modules.

3.5. server_endpoint_hrc

The module starts the ROS-Unity Endpoint communication with a Ros communication objects dictionary for routing messages.
In order to work the module requires the ROS package ros_tcp_endpoint, which defines the tcp endpoint communication features.
The topics defined are:

- unity_tf
- baxter_joint_states
- baxter_target_reached
- baxter_left_gripper
- baxter_right_gripper
- baxter_moveit_trajectory
- baxter_gripper
- baxter_rest_position
- baxter_stop

3.6. Unity

This is not a ROS module, available as Unity Project, but it is part of the architecture.
It communicates with the ROS side, as previously mentioned, through topics, in particular we have done some changes starting from the original code in **BaxterController.cs** and **ROSInterfaces.cs** files:

- Added a publisher on topic *baxter_target_reached* not periodic
- Added a publisher on topic *baxter_left_gripper* with a frequency of 2Hz
- Added a publisher on topic *baxter_right_gripper* with a frequency of 2Hz
- Added a subscriber on topic *baxter_gripper* and relative response function
- Added a subscriber on topic *baxter_rest_position* and relative response function
- Added a subscriber on topic *baxter_stop* and relative response function

4. Installation

To test the project two machines (or virtual machines) are needed, one with a Windows OS and the other with a Linux OS.
On Windows side should be installed Unity and relative frameworks to show the simulation, on Linux side should be installed ROS Noetic to run the modules.

UNITY INSTALLATION (ON WINDOWS)

1. Visit https://unity3d.com/get-unity/download and download Unity Hub
2. Through the Hub, install Unity 2020.2.2 (the version for which the projects have been developed) or later. Beware that later versions (e.g. Unity 2021.1.x) may be incompatible with the projects.
3. In order to install the framework for human simulation, visit https://github.com/Daimler/mosim_core/wiki/InstallPrecompiled and follow the steps to install the precompiled framework.
    a. In the downloaded folder, go into the MMUs sub-folder, there many other sub-folders that contains several files in *.dll* extension, each one should be unlocked in its proprieties.
4. Visit https://github.com/EnricoFiasche/HRC-02/tree/main/Unity_project/Human-Robot-Collaboration to download the Unity project folder, extract it, then open Unity Hub and ADD the project to your projects list using the associated button.
5. Open the project.
6. In the bar on top of the screen, open the *Robotics/ROS Settings* tab and replace the ROS_IP with the IP of the machine running ROS.

7. If the previous steps have been successful, you should be able to enter Editor mode (via the Play button) and play the simulation.
8. Il the communication is running correctly, on UBUNTU you can echo the topics that are being exchanged between ROS and Unity.

ROS NOETIC INSTALLATION (ON LINUX UBUNTU 20.04)

1. Install ROS Noetic following the tutorial on http://wiki.ros.org/noetic/Installation/Ubuntu.
2. Create your ROS workspace.
3. Visit https://github.com/EnricoFiasche/HRC-02 and download the ROS packages needed for the project
4. Extract the packages to your workspace.
5. Open the 'human_baxter_collaboration' package, then navigate to config folder and open the params.yaml file. Under ROS_IP, insert the IP address of your machine.
6. Compile packages with catkin_make.
7. Please, don't forget you need MoveIt framework for this project, thus visit https://moveit.ros.org/install/ and install the binaries.
8. Once all dependencies are resolved, do: 'roslaunch human_baxter_collaboration human_baxter_collaboration.launch' to start MoveIt and communication with Unity (ensure the server communication is up and running, you should see the following line in the terminal 'Starting server on YOUR IP:10000').

TIPS

1. If you experience a strange window when opening the Unity Project for the first time, simply do Quit, then reopen the project and it should not bother you further.
2. Should you experience issues in the communication between ROS and Unity, especially when publishing FROM ROS TO Unity, remember to disable the firewall of your PC, as it could interfere with the sending of messages over TCP.
3. The connection on the machines has to be private and the type has to be BRIDGE.
4. When running your project, be sure to launch the ROS files BEFORE playing the Unity simulation, in order to have the server endpoint node up and running before initializing communication.
5. Remember to check that all python files in human_baxter_collaboration/scripts are executable

RUN THE CODE

1. Start the simulation on ROS side executing in the workspace:

   roslaunch human_baxter_collaboration human_baxter_collaboration.launch
2. Start frameworks on Windows executing the file MMILauncher as administrator, placed in the folder Environment\ Sprint\ 13/ Environment/Launcher.
3. Open the project from the Unity Hub and press the Play Button on top of the screen, wait a message of initialization completed on the console, then press the button Start Simulation in the scene.

4. On ROS side, to start moving Baxter, execute in the workspace:
   roslaunch human_baxter_collaboration baxter_CCA.launch
5. On Unity scene press Pick and Place button to start the human motion.

5. System Testing and Results

Testing the whole architecture, the tasks are completed successfully, there was no collision human-Baxter and all blocks were placed in the right box.
The a priori known list of blocks to be picked, that defines the order, is thought taking into account human movements, so the two agents are never to close each other, but in case of possible collision the module **baxter_collision_avoidance** is always active in order to detect it and trigger the **baxter_controller.**
In fact, we also have tested with a collision and the module has recognized it, working as predicted.

6. Recommendations

After the testing phase we noticed that there was just one apparently probabilistic behavior, sometimes the frame “MiddlePlacementN” or the block placed in this position are not reachable, so Baxter is unable to find a plan. This is strange because theoretically that position is always fixed and impossible to be modified. We solved this problem refreshing the cache of previous target position each time the robot has to plan a new trajectory, to do this we use the method “clear_pose_targets” of MoveGroup class.