Robotica Autonoma Homework 3
Navigation

Elena Camuffo, Emilio Olivastri, Edoardo Trombin

October 23, 2021

1 Problem Setup

In this final homework, the overall arena configuration is given. It includes a set of cubes with apriltag landmarks attached, a robot manipulator (UR5) and a differential drive robot (marrtino). The arena is delimeted by a set of walls where a set of obstacles is also placed. The task of *homework 3* consists of moving marrtino in the arena avoiding the obstacles, and make it interact with UR5. Marrtino has to approach UR5, which in turn must pick 2 cubes and place them over marrtino, which is then lead back to the initial position.

2 Development

The proposed solution for the interaction is based on a **client-server approach**. Here the manipulator robot acts as a server, offering its own service to whichever marrtino robot reaches its table. The marrtino robot is instead thought and implemented as a client, that navigates towards the manipulator, asks for two cubes, waits for the response, and comes back to the initial pose.
The system is implemented using the `actionlib` package. An action server is used for UR5 and an action client for marrtino. The two entities communicate exchanging *actions*, where marrtino sends to UR5 its own id, its actual position and the cubes it wants to be picked, and UR5 responds with the outcome of the operation.

The management of the overall system, which governs the coordination between the movements of marrtino and UR5, is committed to a **finite state machine**, represented in figure 1. It is defined accordingly to the behaviours of the two robots and the relative positions in the arena.
In detail, the blue states represent the positions of interest that marrtino can occupy; the green ones instead represent the states of UR5, where the state *UR5 pick* includes the whole routine needed to pick a cube, and it has a loop transition, since multiple cubes can be picked; finally the *Error* state is reached only if something goes wrong during the whole process.

The main task carried out in this homework is the navigation of **marrtino**. The movement of the robot is developed thanks to the *move_base* action server which provides a way to retrieve the pose of marrtino and allows to set a goal, in order to make it move towards the desired target. When a goal is given to marrtino, it starts planning its route to the objective, according to its global planner and costmap. It then refines its trajectory according to a local planner and costmap allowing the

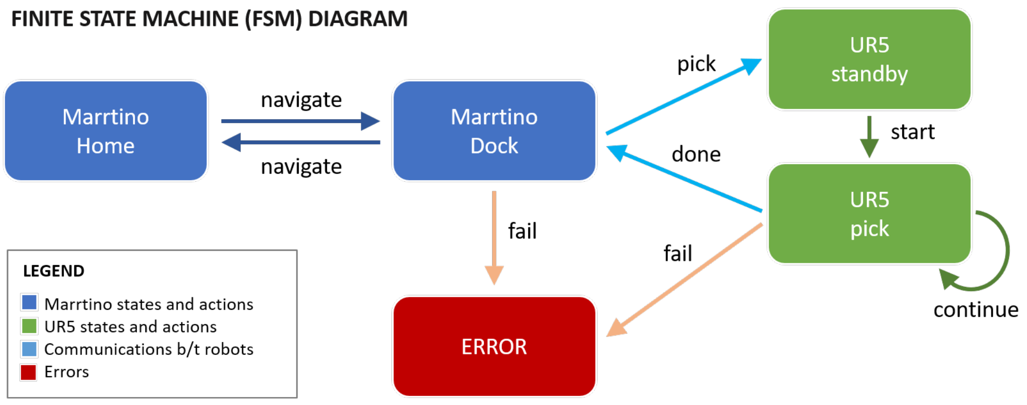

Figure 1: Finite States Machine diagram.

movement to be more consistent with the dynamicity of the environment and the sensor readings collected along the way. The correct navigation of marrtino is obtained through the tuning of the navigation parameters, found with several navigation tentatives, adjusting the values in order to obtain a negligible number of navigation abortions and a short navigation time. The exact value of all parameters is omitted for brevity but can be found in the yaml configuration files which are inside `simulation_ws\src\internal\marrtino\marrtino_navigation\config`.
The most relevant ones are listed below:

- Inflation layer parameters are needed to increase costmap's values near obstacles. The tuning of `cost_scaling_factor` and `inflation_radius` is necessary to ensure low cost paths, preferably in the middle of narrow apertures. However, low values make the robot "think" it can pass through apertures that are too small compared to its size.
  We set `inflation_radius = 0.3` and `cost_scaling_factor = 2.0` to meet our requirements.

- Global planner parameters define the shape of the global path. To ensure that the trajectory passes in the middle of apertures, the parameters are set so that `neutral_cost = 66`, `lethal_cost = 253` and `cost_scaling_factor = 0.55`. In this way, trajectories are smooth and as far from obstacles as possible.

- Local planner parameters govern the robot's step by step behavior during the navigation, especially when approaching obstacles. The most important local planner parameters are `xy_goal_tolerance` and `yaw_goal_tolerance`, which define the final pose's tolerance. Then, `occdist_scale` defines the tendency to stay far from obstacles, `path_distance_bias` how much to stay close to the global path, and `goal_distance_bias` makes the robot less attached to the global path. We set `xy_goal_tolerance = yaw_goal_tolerance = 0.1`, `occdist_scale = 0.01`, `path_distance_bias = 32.0` and `goal_distance_bias = 15.0`.

- `footprint_padding` is a parameter that enlarges the footprint of the robot, marking cells near the robot as prohibited. This conservative behavior is correct most of the times, but in case some apertures are too small, the footprint padding overlaps obstacles even when marrtino is capable of passing through. By setting `footprint_padding = 0.001`, marrtino is able to pass very close to obstacles, avoiding collisions.

Since there are two dock stations in the arena, marrtino first tries to reach one; if the planning does not succeed, e.g. if an obstacle obstructs the dock, it tries the other one; in case both are occupied the planning is aborted. In addition, a filter is built in order to avoid some noise, especially around marrtino, in correspondence to its wheels.

The **UR5 manipulator** is instead implemented exploiting the classes implemented in *homework 1* and *homework 2*, and creating a wrapper server class. In particular the problem with the singularity of the arm that was in *homework 2* is here fixed.

3 Conclusions

The correct tuning of the navigation parameters ensures that marrtino can reach almost every feasible position in the arena, also passing through tight passages. The improvement in the movement of UR5 on the other hand, has led to a cleaner solution, allowing it to pick cubes without falling in any singularity. Nevertheless the correct outcome of the whole procedure is sometimes disturbed by the malfunctioning of the simulator.

The FSM model is inspired by the generation process led by the mastermind controlling Amazon robots. The scenario implemented shows one marrtino and one UR5 robots, but the client-server model proposed is thought to fit also in case multiple robots would be involved in the arena.