The Final Challenge

AI and Robotics - Lab 05

Giacomo Gaiani - giacomo.gaiani@studio.unibo.it

Submission Date: 12/06/2024 Submission Number: 1

Contents

1 Executive Summary

During this project, I performed all the tasks presented in the activity proposal, with the addition of a supplementary optional tasks that extends one of the suggested ones. The main task consisted in developing a control system to handle navigation and transportation objectives, following the *Sense, Plan, Act* architecture. To achieve this, the main aspects to implement are:

- the gateway between the robot and the simulated environment to manage the sensors data gathering and actuators control,
- the hierarchical action planner to generate a sequence of atomic actions to solve a higher level task,
- the robot behaviors to control the robot movements for each of the atomic actions.

The rest of the addressed tasks involved the detection of failures and the attempt to recover from those situations, such as box misplacement or door fault, or to use the map to correct the robot self localization. An initial framework of the project was provided [1], as well as a series of ROS services to ease some of the more complex actions [2]. The testing of the system was carried out through single task assessments, changing the starting point, the end goal and, for some of the tasks, adding obstacles to the environment. The main limitation of the project is the limited capability of the system to generate higher level trajectory for the navigation, instead of using the sonar sensors reading at each step, making the robot prone to mistakes in pathfinding. Another possible problem that may arise is the use of the sonar sensor for the box and door check, that requires precise tuning in the parameters to avoid misreading.

2 Compulsory task: Plan and execute navigation and transportation tasks

2.1 Challenges

The primary challenges of this first task fall into two main categories. Firstly, creating a complete and powerful set of pyhop operators and methods, capable of generating efficient plans to be executed by the simulated robot. To achieve this, it is important to take into consideration all possible starting and target positions, while considering the hierarchical aspect of planning. Secondly, it is to define robust fuzzy behaviors able to adapt to the different possible environment situations the robot may encounter. The main obstacle of this section is to correctly implement the hybrid "GoTo" behavior, by declaring effective fuzzy rules. The remaining challenges faced to complete the task involve the implementation of functions to correctly utilize the ROS services provided for this activity and the partial redesign of the lower level of the control loop of the robot, concerning the sensor reading and velocity publishing, as the time needed to gather the sensor data act as a bottleneck for the remaining parts of the program, while the publishing of the velocities make the robot move only for a fraction of the time of the loop, making the test session extremely time consuming.

2.2 Solution

2.2.1 Pyhop methods and operators

The first step to correctly implement the task is to complete the list of pyhop methods and operators to successfully generate efficient plans. Initially, it is important to define the list of actions the robot can perform, in this case: go to a defined place; open, close or cross doors and pick up or put down boxes. After that, the main improvement is the function *doors_between*, that given two room names, returns the shortest path between them, if any. With those set of operators, was possible to define higher level methods, such as transport a box from its starting position to a target one.

2.2.2 Fuzzy behaviors

Given each pyhop operator discussed above, a fuzzy behavior with the same name has been defined. Those fuzzy behavior can be split into two categories. First, the fully implemented behaviors, namely *GoTo* and *Cross*, containing fuzzy operators, linguistic variables and fuzzy rules. Second, the behaviors that simply invoke the relative ROS service, such as *PickUp, PutDown, Open* and *Close*. The *Cross* behavior utilizes the sonar sensor information to traverse the opening of a door, trying to remain centered it is marked as complete after completely crossing the door. On the other hand, the *GoTo* behavior utilizes both the sensor information and the linear and angular distance from the target and has an hybrid structure, combining three simpler sub-behaviors, *MoveToTarget, Avoid* and *GoClose*. Under normal circumstances, the robot simply moves in the direction of the target. If an object is detected, it switches to the *Avoid* behavior, and it tries to distance form it. If the distance form the target is under a defined threshold, the *Avoid* behavior is overwritten by *GoClose*, which make the robot go slowly towards the target object. The two variables that control the behavior switch are called *DangerAhead* and *TargetNear*. The decision to implement the *GoTo* behavior in this way comes in continuity with the rest of the behaviors that, even if not presenting an hybrid nature, shares most of the structure, making it easier to understand and extend for further implementations.

2.2.3 Robot gateway

As stated, one of the challenges to overcome to complete the task effectively, is to speed up both the gathering of the sensor data and the publishing of linear and angular velocities. The solution for the sonar reading is to utilize only the five frontal sonar sensor for most of the actions, and utilize seven when crossing a door, in order to sense if the door is completely crossed. The publish of the velocities has been made asynchronous, so that the robot performs the movements more smoothly, without stopping at every cycle. This solution presents a major problem, which is the delay in the responsiveness of the robot to external factors. The problem can be partially solved by reducing the overall velocity of the robot, giving it time to sense and react, and by increasing the efficiency of the

cycle, keeping the cycle time as low as possible. The change in the sonar sensors reading contribute in this way.

2.2.4 ROS Services

The last challenge to be address is the implementation of an infrastructure to invoke the ROS services. A series of functions were created to communicate with the ROS server in order to get the doors and boxes information, and eventually modify their states.

2.3 Testing

The evaluation of the solutions presented was carried out through a series of tests of different complexity, based on two higher level actions: *navigate_to* and *transport*. The first make the robot reach a goal position, while the second make the robot reach the position of a specified box object and transport it to a target position. The tests conducted fall in three categories: single action task with a random starting position, a series of action tasks to be performed one after the other or a single action task with artificial obstacles in the environment. All the test of the first two categories were successful, while the third group of tests showed some of the limitations of the implementations. The robot were able to navigate through convex obstacles, frequently avoiding them on the right side, as it was set as a tie braking mechanism for frontal obstacles. On the other hand, the robot struggled with concave obstacles, ending up stuck in most of the situations. An example would an inner wall with the opening on the left side. The figures 1 and 2 show the trajectory of the robot, both in terms of the real and the computed position, performing a series of navigation tasks, while figures 3 and 4 show the trajectory of the robot performing a series of transportation tasks. The input goal to achieve figure 1:

```
('navigate_to', 'bed1'), ('navigate_to', 'table2'), ('navigate_to', 'stove1'),
    ('navigate_to', 'bed1')
```

The input goal to achieve figure 2:

```
('navigate_to', 'fridge1'), ('navigate_to', 'table3'), ('navigate_to', 'stove1'),
    ('navigate_to', 'wardrobe1')
```

The input goal to achieve figure 3:

```
('transport', 'box1', 'table2'), ('transport', 'box2', 'stove1'),
    ('transport', 'box3', 'bed1')
```

The input goal to achieve figure 4:

```
('transport', 'box2', 'table3'), ('transport', 'box1', 'fridge1'),
    ('transport', 'box3', 'wardrobe1')
```

2.4 Assessment

As anticipated in the **Testing** chapter, the main limitation of the presented solution are the concave obstacles or the situations that require higher level map knowledge to traverse the environment. A possible solution would be having an occupancy gridmap, or creating it in real time to avoid getting stuck. Another problem encountered during the test phase was navigating near the *table1* object, as it has a single small table leg and a larger table top, making it difficult to sense for the sonar sensors. Additionally, a problem that may arise during crossing of a door, is that the behavior utilizes the sonar senors both for moving and for assessing the completion of the action. This can cause malfunctions if other objects are present near the door, acting as noise for the sonar sensor. A possible solution would be to redesign the *Cross* behavior, modelling as a *GoTo* between the two approach point of the door. Finally, as mentioned in chapter **2.2.3**, another limitation encountered during the evaluation of the system has to do with the duration of each control cycle, which can vary from 0.3 seconds to 0.7 seconds, depending on various factors. This can cause some delays in the response of the robot to external stimuli. This could cause issues like oversteering during a turns or ending up too close to the target object during a navigation task.

3 Optional task 1: Deal with failures due to displaced boxes

3.1 Challenges

The challenges of this task are finding a reliable and robust way to check the correct placement of the boxes, using the sonar information or the *get_box_position* ROS service. After that it is important to configure the behavior of the robot in case of failure due to displaced boxes.

3.2 Solution

In order to solve the task, I have implemented two procedures with different characteristics in order to deal with the box displacement.

3.2.1 Box check

The first method uses a separate behavior called *BoxCheck*, performed after arriving in proximity of the box position and before picking up the box. A pyhop operator with the same name has been added before the every *PickUp* action. It utilizes the frontal sonar sensors to detect the presence of an object within a certain distance. If an object is detected, the action is completed and the planning proceeds. In the case where no objects are detected, the action is set as failed, but, instead of terminating the program, a replanning request is sent to the higher level of the control system. During the planning phase, the position of the box is set thanks to the *get_box_position* service. The main problem with this first method is the possible inaccuracy of the sonar sensors. If the box was originally placed near another object or a wall, or it has been replaced with a wrong object, the robot may erroneously detect the position of the box as correct.

3.2.2 Real-time box tracking

The second method uses the *get_box_position* service to check if the box has been moved from its starting position. This procedure is performed every n cycles, where n is a parameter defined by the user. If the position of the boxes changes over a certain limit, but the boxes is still in the same room, the parameter of the

GoTo behavior are updated. In case the box is in a different room with respect to its previous position, a replanning request is sent, similarly to the *BoxCheck* method. This second procedure ensures better precision and responsiveness with respect to the first one, but it requires the box to be traceable with high precision in real time.

3.3 Testing

The evaluation of the solution was conducted via two main methods: using the gazebo interface to manually move the boxes, or using the *update_position* service to randomly move the box. All of the tests conducted were satisfactory, from the point of view of the robot behavior in addressing the box misplacement.

3.4 Assessment

As mentioned in the **Solution** chapter, both methods present some limitations. The box check method is unreliable in environment containing numerous objects, as they may lead to a incorrect sensor reading. The real-time box tracking, on the other hand, requires the box to be precisely located at any time, which is a strong assumption in a lot of real world scenario.

4 Optional task 2: Deal with failures due to closed doors

4.1 Challenges

The main challenge of this task is to use the sonar data information to verify the state of a door, given that the robot can arrive a the door different angles, and decide what to do in case of failure.

4.2 Solution

The first difference made to the code was to add a pyhop operator and the relative behavior to check the state of the door before crossing it, similarly to the *CheckBox* method described in the previous chapter. The problem with this solution was make the evaluation sound to any type of error. In order to do so the action of going to a door before crossing it has been split into two actions. First the robot compute the position of the two access points of door, corresponding to the points at a certain distance in the axis perpendicular to the door, one for each side. The one in the same room as the robot is used as a target for the *GoTo* action. Second a new pyhop operator and behavior has been added call *Align*, performed before the door check, and align the robot to face directly the door. In this way, it is easier for the robot to check the door state. After a failure is detected, there are two possible options. If we assume the error was in the information given to the robot, the latter can simply update the state of the door in the environment information and open the door if needed. If we assume that the door is closed due to a malfunction and so it is either broken or locked, the robot removes the door from the internal map and request a replanning in order to find a path to the target the does not involve the specific door.

4.3 Testing

The evaluation of the two methods were conducted by testing both scenarios, unlocked and locked door, on every door present in the environment. All of the test conducted were successful. In the second case, the robot were able to find alternative paths to the target position, if they were. The figures 5 and 6 show the

situation in which, finding a locked door, the robot has to compute an alternative plan to reach the target position.

4.4 Assessment

The tests showed satisfactory results. The main limitation of the approach is to calibrate the response of the sonar sensor, as they might not work correctly for doors of different dimensions. For example, the door *door3* required an adjustment in the fuzzy predicate value, being larger than the rest of the doors of the environment.

5 Optional task 3: Use the map to correct the robot's self localization

5.1 Challenges

The main challenge of this task is to find a reliable way to correct the positional error in the robot's self localization.

5.2 Solution

The first approach I have implemented to solve this task utilizes the positional information of a door in the environment and the sonar sensor to locate the robot with respect to the door. The first attempt consisted of crossing the door and store the positional information after completely crossing it. The main problem is to identify the offset between the position of the door and the robot position, as the crossing behavior may end with slightly different distances from the door. For this reason, the solution I have implemented is to perform the crossing operation twice, one for each side of the door. Given the two point, compute the middle and compare it with the position of the door. The offsets in the two coordinates are subtracted to the position of the robot.

5.3 Testing

The evaluation of the solution presented was performed via a series of test consisting in a sequence of navigation tasks, followed by the recalibration of the robot position. In the figure 7 it is possible to see the values of the distance between the ground truth and the computed position of the robot, where the red vertical line indicates the instant of the recalibration. An example of input goal used during evaluation to achieve figure 8:

```
('navigate_to', 'bed1'),('navigate_to', 'table2'),('navigate_to', 'stove1'),
    ('navigate_to', 'table3'),('calibrate', 'D2'), ('navigate_to', 'stove1')
```

5.4 Assessment

Despite the intrinsic complexity of finding a robust method to correct the robot localization, the solution presented showed good results, by slightly decreasing

the error. The overall performances has to take into consideration that the fluctuation of the error before the recalibration procedure was small, compared to how it may increase after an extensive use of the robot navigation.

6 Optional task 4: Deal with failures due to missing box in transport

6.1 Challenges

Similarly to what mentioned in chapter **3.1**, the challenges of this task are finding a reliable and robust way to check the correct position of the box the robot is carrying configure the behavior in case of failure boxes.

6.2 Solution

In order to solve the task, the first step was to add the knowledge about the object the robot is carrying. The state of the carried object is changed at the end of the *PickUp* and *PutDown* actions, or it is detected that the robot is no longer carrying it. This last situation is identified measuring the distance between the robot position and the real-time position of the carried object, via the *get_box_position* service. This evaluation is done every n cycles, where n is a parameter defined by the user.

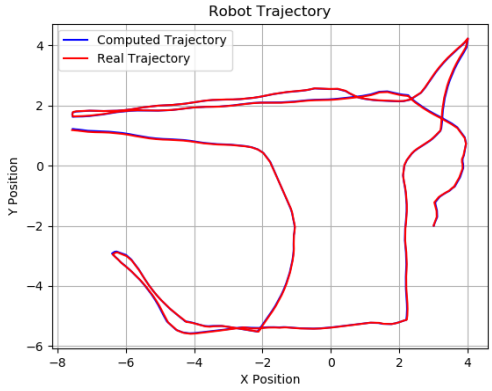
6.3 Testing

The evaluation of the solution was conducted via two main methods: using the gazebo interface to manually move the boxes, or using the *update_position* service to randomly move the box. Most of the tests conducted were satisfactory, from the point of view of the robot behavior in addressing the failure of transport action and the recovery form this situation.

6.4 Assessment

The limitations of the presented solution are mainly two. First, the method to register the failure during the transportation task could be improved by adding, in a real world scenario, specific sensors to check the correct position of the box on top of the robot, and readily act to prevent the box from falling. Secondly, in some of the test performed, the box fell off the robot while traversing a door. In this case the robot would have to cross again the door, which was unfeasible, due

to the box obstructing the passage. This was one of the limitations of the *Cross* behavior discussed in chapter **2.4**



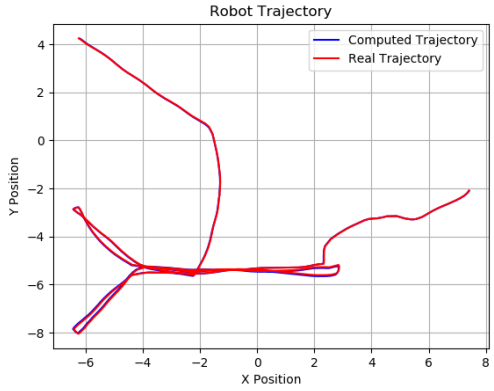
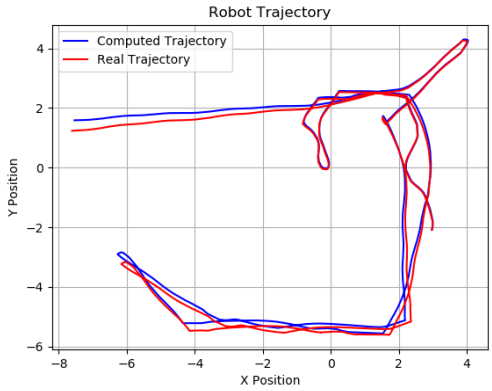


Figure 1: The robot trajectory during a series of navigation tasks

Figure 2: The robot trajectory during a series of navigation tasks



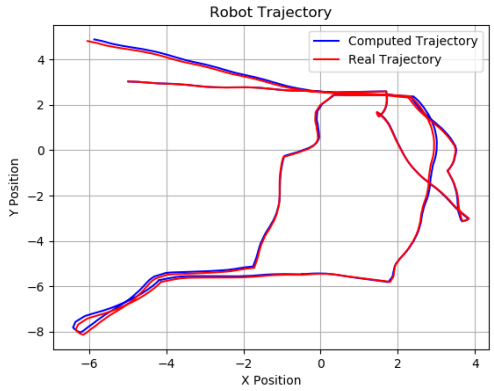


Figure 3: The robot trajectory during a series of transportation tasks

Figure 4: The robot trajectory during a series of transportation tasks

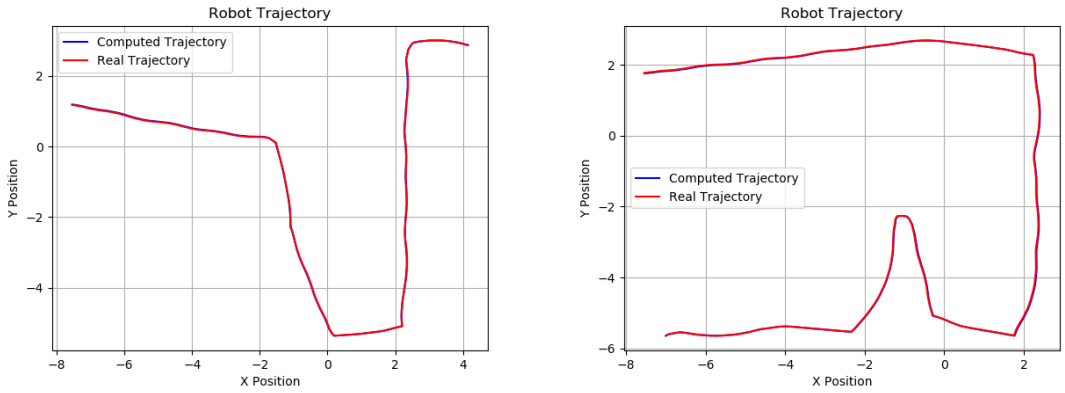


Figure 5: The robot trajectory finding the door *door2* locked

Figure 6: The robot trajectory finding the door *door3* locked

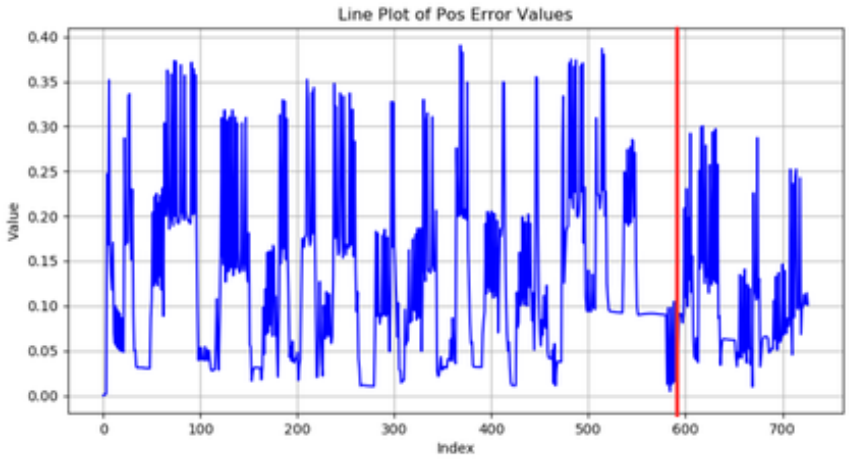


Figure 7: The distance between the ground truth and the computed position of the robot. The red line indicates the recalibration of the robot position.

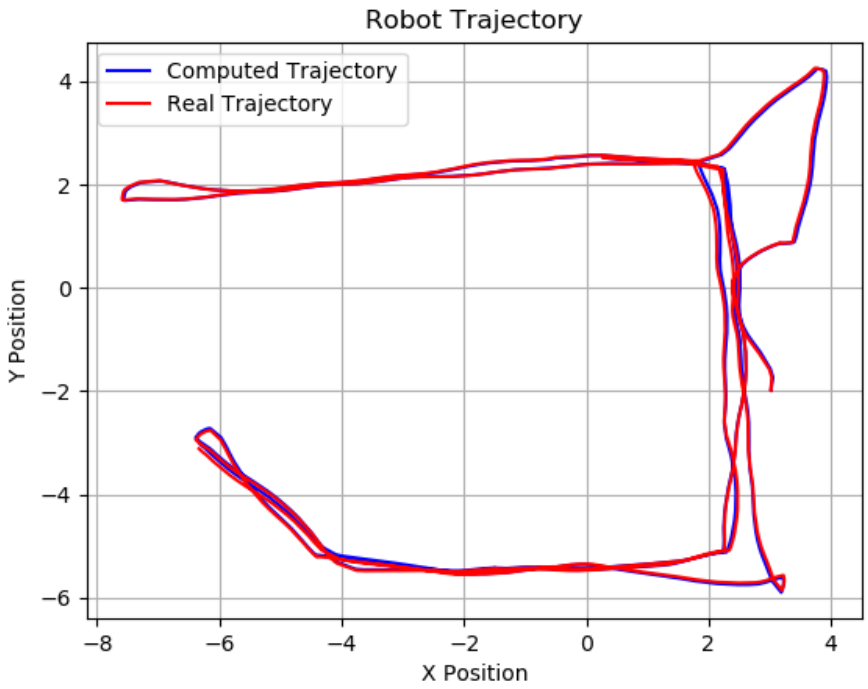


Figure 8: The robot trajectory during a series of navigation tasks, followed by a position recalibration

References

[1] Alessando Saffiotti. *BaseCode-2024*. 2024. URL: https://github.com/asaffio/BaseCode-2024.

[2] Andrea Govoni. *Ai_for_Robotics*. 2024. URL: https://github.com/GovoUnibo/Ai_for_Robotics.