Robot Task Planning

AI and Robotics - Lab 04

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

Submission Date: 07/05/2024 Submission Number: 1

Contents

1 Task 1: Implementation of an HTN planner for home navigation

1.1 Problem Description

The objective of all the tasks presented in the laboratory exercise, with the sole exception of the last one, is to modify or extend the operators and methods of the HTN planner to reach specific goals. An initial domain and operators are already defined in the python files `lab4.py` and `htn_domain.py` provided for the laboratory assignment [1]. In this first task, the goal is to generate a plan to navigate between the different rooms of the house to arrive at the target location form any possible starting point.

1.2 Objects and Data Structures

The first four tasks presented in the laboratory require to modify or define pyhop operators and methods or to extend the domain to generate the plan. All of the remaining objects and data structures are imported from the previous laboratories.

1.2.1 Pyhop State

In order to complete this first task, the HTN domain state has been extended to contain the information about the path the robot has to take to reach the target position, in form of a python list, and the set of visited room during the research of the path, to avoid infinite loops.

1.2.2 Pyhop Operators

The two operators *GoTo* and *Cross* were already defined in the laboratory assignment

1.2.3 Pyhop Methods

The *move_across_rooms* method has been modified for extend its capabilities. Firstly, the support function *doors_between* has been improved to return the list of door the robot has to cross to arrive at destination. Secondly, the method has been adjusted to work recursively, so that the robot can reach target position that requires the traversing of more that two rooms.

1.3 Control and Information Flow

The control and information flow of the first task is rather simple. After defining the HTN domain state, the operators, the methods and the task, the planner automatically generates a plan capable of solving the task, if one exists.

1.4 Parameters

The parameters used in this laboratory exercise were simply the position of the objects defined in the state domain and how the rooms are connected to one another through doors. All of those information were already present in the files provided for the activity.

1.5 Results

The objective of this first assignment is to correctly generate a plan to solve the following tree tasks of increasing difficulty:

```
('navigate_to', 'table3')
('navigate_to', 'bed1')
('navigate_to', 'stove')
```

The difficulty lies in the distance between the robot and the target object, translating into a more complex solution. The corresponding generated plans are:

```
Plan: [('GoTo', 'table3')]
Plan: [('GoTo', 'D2'), ('Cross', 'D2'), ('GoTo', 'bed1')]
Plan: [('GoTo', 'D1'), ('Cross', 'D1'), ('GoTo', 'D4'), ('Cross', 'D4'),
    ('GoTo', 'stove')]
```

1.6 Conclusions

The default version of the HTN planner provided for the laboratory activity was already capable of solving the first two of the three task presented in the *Results* chapter. Thanks to the improvement described above to the planner methods, it was able to solve the third and final task.

2 Task 2: Extension of the HTN planner with open and close door actions

2.1 Problem Description

The objective of this second task is to extend the HTN planner to be able to open and close the door, if needed, during the house navigation.

2.2 Objects and Data Structures

This second task requires to extend the domain, operators and methods of the HTN planner.

2.2.1 Pyhop State

The object of the room connections, represented by the doors, are extended to contain the information of its state, either open or close.

2.2.2 Pyhop Operators

Two new operators have been added, *Open* and *Close*, and are used to change the state of the doors if the robot is in that exact position or fail otherwise.

2.2.3 Pyhop Methods

The *move_across_rooms* method has been modified to behave differently if the door that have to be crossed is open or close. If close, the robot has to open in before crossing it, and close it afterwards.

2.3 Control and Information Flow

The operational structure of the program is identical to the first task.

2.4 Parameters

The parameters used in this laboratory exercise are the same of the first task, with the extension on the door state information, all set to close.

2.5 Results

The objective of this second assignment is to correctly generate a plan to solve the same tree tasks of the previous activity:

```
('navigate_to', 'table3')
('navigate_to', 'bed1')
('navigate_to', 'stove')
```

The corresponding generated plans are:

```
Plan: [('GoTo', 'table3')]
Plan: [('GoTo', 'D2'), ('Open', 'D2'), ('Cross', 'D2'), ('Close', 'D2'),
    ('GoTo', 'bed1')]
Plan: [('GoTo', 'D1'), ('Open', 'D1'), ('Cross', 'D1'), ('Close', 'D1'),
    ('GoTo', 'D4'), ('Open', 'D4'), ('Cross', 'D4'), ('Close', 'D4'),
    ('GoTo', 'stove')]
```

2.6 Conclusions

The plan generated for the `('navigate_to', 'table3')` task is the same as the one generate in the previous exercise as it does not require the robot to cross any door. On the other two plan, it is possible to see how the robot opens and closes the door before and after crossing it.

3 Task 3: Extension of the HTN planner with pick up and put down actions

3.1 Problem Description

The objective of this third task is to extend the HTN planner to be able to navigate to an object and pick it up or navigate to a place and put down the object it is carrying.

3.2 Objects and Data Structures

This third task requires to extend the domain, operators and methods of the HTN planner.

3.2.1 Pyhop State

The domain of the pyhop state has been extended to contain four more objects, namely *box1, box2, box3* and *box4*, where the number corresponds to the room the box is placed. Additionally, it has been added the information about the robot arm, which may be empty or contain the name of the object the robot is carrying.

3.2.2 Pyhop Operators

Two new operators have been added, *PickUp* and *PutDown*, and are used to change the state of the robot arm and the position of the object the robot is carrying.

3.2.3 Pyhop Methods

Two new methods have been added: *fetch* and *carry_to*. The first perform a *navigate_to* to reach the target object position and a *PickUp* to grab it. The second performs a *navigate_to* to reach the target position and a *PutDown* to put down the object the robot is carrying.

3.3 Control and Information Flow

The operational structure of the program is identical to the first task.

3.4 Parameters

The parameters used in this laboratory exercise are the same of the first task, with the extension on the box position information, all set to the relative room. Regarding the doors state, they are all set to open except the door between room 1 and 4, which is close.

3.5 Results

The objective of this third assignment is to correctly generate a plan to solve the following four tasks:

```
('fetch', 'box4')
('fetch', 'box3')
('fetch', 'box2')
('fetch', 'box1')
```

The corresponding generated plans are:

```
Plan: [('GoTo', 'box4'), ('PickUp', 'box4')]
Plan: [('GoTo', 'D1'), ('Cross', 'D1'), ('GoTo', 'box3'), ('PickUp', 'box3')]
Plan: [('GoTo', 'D1'), ('Cross', 'D1'), ('GoTo', 'D4'), ('Cross', 'D4'),
    ('GoTo', 'box2'), ('PickUp', 'box2')]
Plan: [('GoTo', 'D2'), ('Open', 'D2'), ('Cross', 'D2'), ('Close', 'D2'),
    ('GoTo', 'box1'), ('PickUp', 'box1')]
```

3.6 Conclusions

The program was able to generate a plan to fetch the box present in all the rooms of the simulated environment. The plan generated for the `('fetch', 'box1')` task is possible to see how the robot has to open the door to reach the *room1*.

4 Task 4: Extension of the HTN planner with a top-level transport action

4.1 Problem Description

The objective of this fourth task is to extend the HTN planner to be able to navigate to a target object and transport it to a target position.

4.2 Objects and Data Structures

This fourth task requires to combine the methods of the HTN planner to implement *transport*.

4.2.1 Pyhop State

The domain of the pyhop state is the same of the previous task.

4.2.2 Pyhop Operators

The pyhop operators are the same of the previous task.

4.2.3 Pyhop Methods

A new methods has been added, *transport* which, given an object and a target position, it moves the robot to the location of the object, picks it up and carries it to the target location where the object is put down. The method *navigate* has been modified to handle the case in which the robot carrying an object and has to open a closed door. To solve this problem, the robot, arrived at the location of the close door, put down the object, open the door, carry the object while crossing the door, put in down again, close the door, and transport the object to the target position.

4.3 Control and Information Flow

The operational structure of the program is identical to the first task.

4.4 Parameters

The parameters used in this laboratory exercise are the same of the third task.

4.5 Results

The objective of this fourth assignment is to correctly generate a plan to solve the following two tasks:

```
('transport', 'box1', 'table2')
('transport', 'box2', 'table2')
```

The corresponding generated plans are:

```
Plan: [('GoTo', 'D2'), ('Open', 'D2'), ('Cross', 'D2'), ('Close', 'D2'),
    ('GoTo', 'box1'), ('PickUp', 'box1'), ('GoTo', 'D2'), ('PutDown', 'D2'),
    ('Open', 'D2'), ('PickUp', 'box1'), ('Cross', 'D2'), ('PutDown', 'D2'),
    ('Close', 'D2'), ('PickUp', 'box1'), ('GoTo', 'table2'), ('PutDown', 'box1')]
Plan: [('GoTo', 'D1'), ('Cross', 'D1'), ('GoTo', 'D4'), ('Cross', 'D4'),
    ('GoTo', 'box2'), ('PickUp', 'box2'), ('GoTo', 'D4'), ('Cross', 'D4'),
    ('GoTo', 'D1'), ('Cross', 'D1'), ('GoTo', 'table2'), ('PutDown', 'box2')]
```

4.6 Conclusions

The program successfully generated a plan to move to the box's location and transport it back to the starting position of the robot. The first plan handle the situation of carrying an object while opening a door.

5 Task 5: Execution of the HTN planner by the simulated robot

5.1 Problem Description

The objective of this last task is to combine the HTN planner with the fuzzy behaviors developed for the previous laboratory assignment to execute in the virtual environment the generated plan.

5.2 Objects and Data Structures

Most of the object and data structures used for this laboratory exercise are the same used in the previous task, combined with the structure of the previous laboratory assignment. The only exception is the creation of a dictionary containing the positional information of the object defined in the pyhop state. Because it is only required to generate and execute a simple plan, the said dictionary only contains the following data:

```
target_pos = {
    'D1': (1.0, -5.5),
    'D4': (-3.0, -5.5),
    'stove': (-6.5, -3.0)}
```

5.3 Control and Information Flow

The control and information flow of this task can be divided in two sections. In the first part, the planner generates a plan, if one exists, to solve the defined goal. In this case the goal was: `('navigate_to', 'stove')`. Once the plan is generated as a list of pyhop operators, it is passed to the top level loop of the program where, for each action, it executes a corresponding fuzzy behavior. A link between each operator and fuzzy behavior has to be defined. For this test case, the only behaviors needed were *GoToTarget* and *CrossDoor*, corresponding respectively to the *GoTo* and *Cross* operators. A threshold on the degree of achievement is used as a stopping mechanism of the relative behavior. Once all the actions in the plan are executed correctly, the program ends.

5.4 Parameters

The parameters used in this laboratory exercise are the same of the previous tasks. The position of the objects defined in the planner domain are not taken by the simulated environment data but by roughly measuring the distance from the axis origin, introducing a small source of error.

5.5 Results

The objective of this last assignment is to generate a simple plan and execute it within simulated environment. The task of the planner is:

```
('navigate_to', 'stove')
```

The corresponding generated plans are:

```
Plan: [('GoTo', 'D1'), ('Cross', 'D1'), ('GoTo', 'D4'), ('Cross', 'D4'),
    ('GoTo', 'stove')]
```

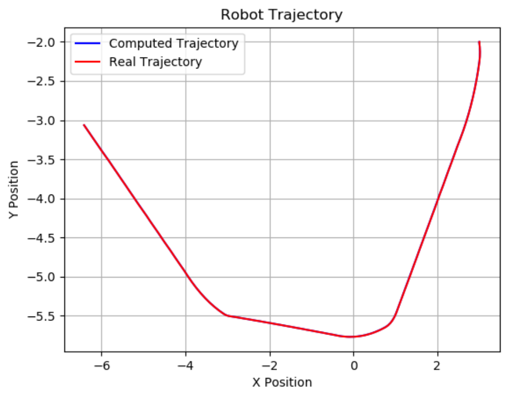

Figure 1: The robot trajectory during the execution of the plan `('navigate_to', 'stove')`

5.6 Conclusions

The program was able to correctly generate and execute the plan, as shown figure 1. Because of the simplicity of the plan and a favorable positioning of the doors with respect to the starting position of the robot and the target position, the robot was able to reach and cross both doors without modifying its trajectory to ease the traversing. Give a more complex starting setup and a more convoluted plan to execute, the robot may need to adjust its behavior to better navigate the different sections of the environment. One example would be to give the robot also the information about the orientation on the door so to end its approach movement facing directly the door.

References

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