Artificial Intelligence for Robotics II
Assignment 1 report

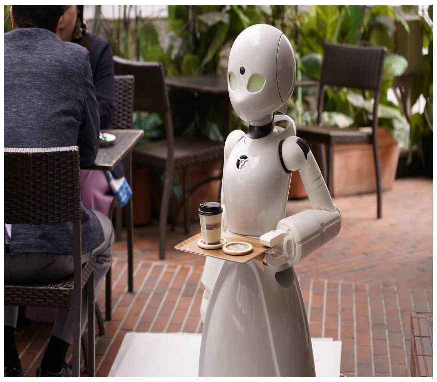
Ludovica Danovaro, Andrea Bolla, Gabriele Nicchiarelli

2 April 2023

1 Introduction

The project consists of modeling a robotic coffee shop using an Artificial Intelligence approach. The goal is to find a sequence of instructions (i.e. a plan) to synchronize the actions of two types of robots, a barista and a waiter. The former needs to prepare the drinks ordered by the customers, and the latter is in charge of serving the drinks and cleaning the table when the customers leave the table.
For our implementation of the model, the waiter is equipped with only one gripper so it can carry only one object at a time.

(a) Barman


(b) Waiter

The environment is organized into two main sections:

- The **bar:** where the barman will prepare the drinks and, when ready, put them on the bar counter so the waiter can pick them up in order to serve them to the customers.
- The **Tables:** each table is set at given distances from the bar and from the other tables. the waiter will serve the drinks (and eventually the biscuits) to the customers seated at the tables.
  An image of the coffee shop disposition can be seen in Figure 2.

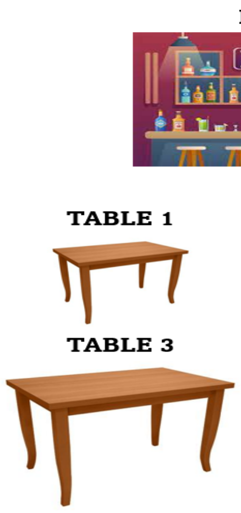


Figure 2: Illustration of the coffee shop

The cafeteria offers two types of drinks: *cold* and *warm*.
Warm drinks need 5 time units to prepare, while cold drinks need only 3 time units. Moreover, according to the optional extension, warm drinks have to be served before they get too cold.

To recap the basic features of the two robots:

- the **barista:**
  1. is always located at the bar;
  2. prepares cold and warm drinks;
  3. puts the ready drinks on the bar counter.
- the **waiter:**
  1. is initially located at the bar;
  2. has only one gripper that can grasp objects;
  3. since it has one gripper, it can only carry one drink/biscuit at a time;
  4. moves at 2 meters per time unit ($2m/tu$);
  5. it takes 2 time units per square meter ($2tu/m^2$) to clean a table.

In addition to these base features, there are some optional extensions that may lead to a better, but also more complex, solution.

In particular, 3 additional properties have been implemented:

1. **"Warm drinks cool down"**: A warm drink takes 4 time units to become cold. A customer will not accept a warm drink delivered after this time limit.
2. **"Finish your drink"**: After receiving the drink, a customer will drink it in 4 time units, and then leave the table. When all the customers of a table have left, the waiter have to clean it.
3. **"Serve food"**: The coffee shop is also serving biscuits. They need no preparation and can be picked up by the waiter at the bar counter. All the customers with cold drinks will also receive a biscuit after having received their drink. The waiter can not bring at the same time the drink and the biscuit but must deliver the drink, and then go back to the counter to take a biscuit for the customer.

2 Folder Organization

The project's folder is organized into the following subfolders:

- *ENHSP*: contains the planner along with some examples that explain the features of this planning engine;
- *results*: contains the plans generated for each problem file;
- *src*: this folder contains the source files for the implementation of the model, in particular:
  - *simple_model*: provides the base implementation of the assignment without any further extensions;
  - *extended_model*: holds the implementation of the assignment with the extensions explained in the previous section.

3 Implementation

The code is written in *PDDL+*, so the best fit is to use the *ENHSP* planning engine (version 20), which stands for Expressive Numeric Heuristic Planner.
It is a PDDL automated planning system that supports *PDDL+* along with numeric planning.
PDDL+ is an extension of the PDDL language that supports planning with autonomous processes and discrete events.
The implementation of the model is divided into:

- a *domain* file: that defines the "universal" aspects of a problem;
- the *problem* files: that express the specific situations, given an initial state, for which the planner has to find a solution (reach the goal state).

3.1 Domain

The domain contains the following specifications:

- *requirements*: the section that allows the inclusion of needed extensions; *types*: definition of the object types used in the problem;
- *predicates*: definition of predicated (which can be interpreted as boolean variables);
- *functions*: definition of fluents (which can be interpreted as numeric variables);
- *actions*: transformations of the state of the world that the planner can exploit to find a plan;
- *processes*: directly correspond to durative actions and last for as long as their preconditions are met;
- *events*: directly correspond to instantaneous actions, and happen the instant their preconditions are met.

3.1.1 Types

In the domain, 4 main types are introduced:

- **robot**: barman or waiter
- **location**: table or bar
- **drink**
- **biscuit**

3.1.2 Actions

This section describes in detail the development of each functionality of the model. Each one of them is composed of a sequence of action-process-event (with some exceptions):

- ***Move***: enables the waiter to move from one location to another one, both if it's carrying or not a drink/biscuit.

  This action *move_start* begins only if $steps \geq 0$.

  This predicate serves as a "robot's action counter" to prevent the robot from moving when not required. Every time the waiter performs an action (serve drink, serve biscuit, and clean a table), the steps are reset to value 2 and the robot can move and go to the location of the next task. The value of this predicate has been chosen due to the particular architecture used: to go from the bar to the furthest tables (tables 3 and 4) the waiter has to pass by the intermediate tables (tables 1 and 2). So it will execute a move action at most two times to reach the desired location.

In the *move_start* action, the robot is assigned the moving time with the following formula:

$$moving\_time = \frac{distance}{2}$$

Since the waiter moves at 2 meters per time unit. The *distance* function represents the distance between two locations and is defined with a fixed value for all the problems.

Afterwards, the robot begins the *move_process*, which decreases the function associated with the robot's moving time by one unit every *delta time* (which has been set to 0.5 for the assignment). This process will stop when the *moving_time* has been exhausted and triggers the *move_end* event, which allows the waiter to arrive at the destination and decreases by one the number of steps it can do, to indicate that the waiter consumes a step each time it moves.

- ***Prepare cold/warm drink***: with these actions the barman, if is *free*, can start preparing a drink. If the drink is warm, it will start the action *prep_drink_warm_start*, if the drink is cold, it will start *prep_drink_cold_start*. These actions will assign the preparation time of the drink given the different specifications for warm and cold drinks and initialize the preparation process.

  When the event *prep_drink_end* triggers, the predicate *ready_drink* activates so the waiter can now take the drink and serve it. In the case of a warm drink, it must be served to the customer within 4 time units. To make this happen, the predicate *priority_warm* is used to allow only the activation of the actions necessary to serve the drink. When the drink has been served the priority is deactivated and the waiter can perform other actions.

- ***Cooling***: this operation is used to keep track of the time that has passed since the end of the preparation of a warm drink. It will begin when the warm drink is set on the bar counter and when the *priority_warm* predicate is activated.

  Each warm drink will be served within the *cooling_time* limit thanks to the priority set previously, so it will not be rejected by the customer. If this were not the case, and the *cooling_time* reaches 0 before the drink has been served, then the *cooling_end* event will happen and the plan will be invalid. However, this can never happen because of the priority mechanism. When a warm drink is served within the time limit the *cooling_cancel* event will flush the remaining cooling time and remove the cooling state.

- ***Pick/Serve drink***: these actions will make the waiter grasp the drink and serve it to the table that has ordered it.

To keep track of how many drinks need to be served at a table, the *drinks_to_serve_at_table* function comes in hand decreasing every time a drink is served at that table.

When the drink is served, the barman becomes *free*: in this way, each drink will be prepared and served sequentially to prevent any drink to stay too long on the bar counter.

- ***Pick/Serve biscuit***: these actions will work in a similar way of the *pick/serve drink* actions. The waiter picks up the biscuit paired with the corresponding drink at the bar counter and serves it to the table where the paired drink is located.

- ***Clean table***: this action is used to make the waiter clean a table, considering the area of the table that has to be cleaned.

  The tables that have to be cleaned are specified by the predicate *to_clean*. This predicate is set to true in the initial conditions or when all the drinks at a table have been consumed.

  When the action *clean_table_start* begins, the *cleaning_time* function is assigned according to the following formula:

  $$cleaning\_time = table\_area \times 2$$

  Where 2 is the cleaning velocity of the waiter.

  After the *clean_table_process*, when the cleaning time runs out, the *clean_table_end* event occurs: the process will stop and the table will be marked as *cleaned*.

- ***Drinking***: this procedure will be triggered only for the last drink served at a table. We assume implicitly that the other customers will finish their drinks for sure before this time limit.

  When the function *drinks_to_serve_at_table* reaches zero, we will know that all the customers at that table have received their drinks and, only at this point, we can start the *drinking_process*.

  The latter will decrease the *drinking_time* until it reaches zero and the *drinking_end* event occurs. At this point, we subsequently activate the *clean_request* event that marks the table to be cleaned. However, this event will happen immediately for warm drinks, but for the cold ones, we have to wait for the biscuits to arrive at the table. So for cold drinks, this event will trigger after all biscuits have been served at that table.

4 Results

4.1 Observations

As can be seen from the generated results, the more the problem is complicated, the greater the time to find a plan will be necessary. This is actually critical with the optional extensions which make the problems much more complex.

To give a general overview of the results we decided to run the problems with two different algorithms, Greedy Best First Search (GBFS) and weighted A star (wA*).

By observing the results in the next subsection, we can see that, in general, the wA* algorithm provides shorter plans (less *elapsed time*) but takes much more time to execute since it explores way more nodes than GBFS algorithm.
This is particularly obvious in the fourth problem with extensions enabled. The wA* is approximately 5 times slower than GBFS but provides us with a better solution.

4.2 Comparison

In the following section, we provided the comparisons between the different solutions found for each problem. The metrics used for the comparison are:

- **elapsed time**: total time of the plan simulation;
- **search time**: time the planner spent to find the solution;
- **planning time**: search time plus the time to get to the solution;
- **heuristic time**: time spent by the heuristic;
- **expanded nodes**: number of nodes actually expanded during the search;
- **evaluated states**: number of nodes predicted to be expanded during the search.

4.2.1 Extensions

Problem 1

- **Greedy Best First Search**: the solution was obtained with an expansion of 262 nodes (and 598 evaluated states) within the time indicated in Table 1:

Elapsed Time	26.5 ms
Search Time	201 ms
Planning Time	1619 ms
Heuristic Time	96 ms

Table 1: Problem 1 - GBFS

- **wA***: the solution was obtained with an expansion of 924 nodes (and 2322 evaluated states) within the time indicated in Table 2:

Elapsed Time	26.5 ms
Search Time	334 ms
Planning Time	2037 ms
Heuristic Time	198 ms

Table 2: Problem 1 - wA*

Problem 2

- **Greedy Best First Search**: the solution was obtained with an expansion of 1893 nodes (and 5625 evaluated states) within the time indicated in Table 3:

Elapsed Time	43.0 ms
Search Time	591 ms
Planning Time	2329 ms
Heuristic Time	426 ms

Table 3: Problem 2 - GBFS

- **wA***: the solution was obtained with an expansion of 7689 nodes (and 19017 evaluated states) within the time indicated in Table 4:

Elapsed Time	40.0 ms
Search Time	1344 ms
Planning Time	3278 ms
Heuristic Time	1004 ms

Table 4: Problem 2 - wA*

Problem 3

- **Greedy Best First Search**: the solution was obtained with an expansion of 38425 nodes (and 121900 evaluated states) within the time indicated in Table 5:

Elapsed Time	48.5 ms
Search Time	7087 ms
Planning Time	8888 ms
Heuristic Time	5926 ms

Table 5: Problem 3 - GBFS

- **wA***: the solution was obtained with an expansion of 21057 nodes (and 68106 evaluated states) within the time indicated in Table 6:

Elapsed Time	41.5 ms
Search Time	3822 ms
Planning Time	5706 ms
Heuristic Time	3186 ms

Table 6: Problem 3 - wA*

Problem 4

- **Greedy Best First Search**: the solution was obtained with an expansion of 48437 nodes (and 238426 evaluated states) within the time indicated in Table 7:

Elapsed Time	77.5 ms
Search Time	20656 ms
Planning Time	22286 ms
Heuristic Time	18738 ms

Table 7: Problem 4 - GBFS

- **wA***: the solution was obtained with an expansion of 194325 nodes (and 992826 evaluated states) within the time indicated in Table 8:

Elapsed Time	72.5 ms
Search Time	103705 ms
Planning Time	106026 ms
Heuristic Time	94970 ms

Table 8: Problem 4 - wA*

4.2.2 No extensions

Problem 1

- **Greedy Best First Search**: the solution was obtained with an expansion of 62 nodes (and 148 evaluated states) within the time indicated in Table 9:

Elapsed Time	16.5 ms
Search Time	182 ms
Planning Time	1783 ms
Heuristic Time	60 ms

Table 9: Problem 1 - GBFS

- **wA***: the solution was obtained with an expansion of 215 nodes (and 653 evaluated states) within the time indicated in Table 10:

Elapsed Time	15.5 ms
Search Time	342 ms
Planning Time	1931 ms
Heuristic Time	148 ms

Table 10: Problem 1 - wA*

Problem 2

- **Greedy Best First Search**: the solution was obtained with an expansion of 639 nodes (and 2199 evaluated states) within the time indicated in Table 11:

Elapsed Time	27.5 ms
Search Time	409 ms
Planning Time	1892 ms
Heuristic Time	209 ms

Table 11: Problem 2 - GBFS

- **wA***: the solution was obtained with an expansion of 2615 nodes (and 9202 evaluated states) within the time indicated in Table 12:

Elapsed Time	28.0 ms
Search Time	604 ms
Planning Time	1980 ms
Heuristic Time	386 ms

Table 12: Problem 2 - wA*

Problem 3

- **Greedy Best First Search**: the solution was obtained with an expansion of 515 nodes (and 1876 evaluated states) within the time indicated in Table 13:

Elapsed Time	33.5 ms
Search Time	262 ms
Planning Time	1400 ms
Heuristic Time	152 ms

Table 13: Problem 3 - GBFS

- **wA***: the solution was obtained with an expansion of 1549 nodes (and 5445 evaluated states) within the time indicated in Table 14:

Elapsed Time	32.5 ms
Search Time	487 ms
Planning Time	1911 ms
Heuristic Time	316 ms

Table 14: Problem 3 - wA*

Problem 4

- **Greedy Best First Search**: the solution was obtained with an expansion of 1880 nodes (and 9823 evaluated states) within the time indicated in Table 15:

Elapsed Time	54.5 ms
Search Time	1022 ms
Planning Time	2900 ms
Heuristic Time	771 ms

Table 15: Problem 4 - GBFS

- **wA***: the solution was obtained with an expansion of 78943 nodes (and 391390 evaluated states) within the time indicated in Table 16:

Elapsed Time	50.5 ms
Search Time	17570 ms
Planning Time	18876 ms
Heuristic Time	15741 ms

Table 16: Problem 4 - wA*