UNIVERSITA DEGLI STUDI DI GENOVA
ARTIFICIAL INTELLIGENCE FOR ROBOTICS II

Second Assignment Report

DIBRIS

DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY, BIOENGINEERING, ROBOTICS AND SYSTEM ENGINEERING

Authors

Mura Alessio, Pisano Davide, Ruggero Miriam Anna, Terrile Ivan

May 2023

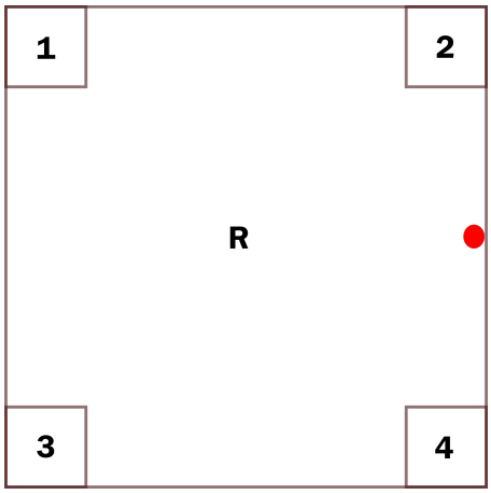
Contents

1 Description of the model

In this second assignment of AI2, we have a 2D environment (shown in the figure below) where there are:

- **four student groups**, each assigned to regions 1-4, working on their assignments;
- a **robot R**, responsible for collecting the assignment reports and delivering them to the submission desk;
- a **submission desk** (red in the figure).

The dimension of the environment is 6m $\times$ 6m. R is initially at (0, 0), and the submission desk is at (3, 0). For the sake of simplicity, assume each region is of size 1m $\times$ 1m and associate a single way-point (x, y) with each region.



2 Objective of the assignment

The objective for robot R is to collect any 2 assignment reports and bring them to the submission desk while minimizing its motion cost. The motion cost is the length of the path covered by the robot. To calculate the length of the path, we need to randomly sample 24 way-points (x, y) from the environment, but outside the four regions and without considering the robot's position and the submission desk's position. After that, we need to connect the way-points to form edges. We can choose a value of k such that each way-point is connected to a maximum of k other way-points. This results in a road-map with way-points as nodes. Finally, the 4 way-points associated with each region, the initial location of R, and the submission desk are to be connected to the nearest nodes in the road-map. The path length is computed by adding the Euclidean distance between the way-points (edge costs) traversed by the robot.

3 Implementation

For this assignment, we made modifications to the following files: *VisitSolver.cpp* and *main.cpp* inside the *../visit_module/src* directory, as well as the *dom*1.*pddl* and *prob*1.*pddl* files inside the *../visit_domain* directory.
In the *VisitSolver.cpp* file, we added this function:

- **void generateRandomWaypoints()** to generate the list of all the required way-points and write them to the *waypoint.txt* file inside the *../visit_domain* directory. The first way-point is the starting point of the robot R at (0, 0). The next four way-points are (-2.5, 2.5), (2.5, 2.5), (-2.5, -2.5), and (2.5, -2.5), corresponding to the four regions. The fifth way-point is the submission desk at (3, 0). The remaining 24 way-points are randomly generated and change with each program execution. The 24 way-points are generated with two decimal places and checked to ensure they are not the starting point of the robot, the submission desk, the points within the four regions and that they are not located within the regions of size 1x1;

- **void connectWaypoints(const vector<tuple<double, double, double>>& randomWaypoints)** to loops through each pair of way-points and computes the distance between them using the **computeDistance** function. It then adds the edge to the edges vector. After all edges have been added to the edges vector, the function calls the **Kruskal** function, passing in the edges vector, the number of way-points, and the maximum number of connections per node. The **Kruskal** function is responsible for connecting the way-points based on the edges in the edges vector.

- **double computeDistance(tuple<double, double, double> wp1, tuple<double, double, double> wp2)** to calculate the Euclidean distance between two waypoints in a 2D space.

- **void kruskal(vector<Edge>& edges, int numNodes, int numMaxConnection)** to implements the Kruskal's algorithm to find the minimum spanning tree of a graph.The Kruskal algorithm is a greedy algorithm used to find the minimum spanning tree (MST) of a connected, weighted graph. The minimum spanning tree is a subgraph that connects all the vertices of the original graph with the minimum total weight. It sorts the edges in increasing order of weight, merges sets of nodes using the findSet and unionSet functions, and updates the adjacency matrix with the weights of the edges in the minimum spanning tree.The generated graph is shown in Figure 2.

- **double minimalPath(string from, string to)** to implement Dijkstra's algorithm to find the shortest path between two nodes in a graph. It initializes the distances and visited arrays, finds the unvisited node with the smallest distance, marks the node as visited, and updates the distances to neighboring nodes if the new distance is smaller than the current distance. The **minimalPath** function is called within the **externalSolver** function to calculate the minimum distance between two regions. The return value of the **minimalPath** function is assigned to the variable *act_cost* and will be used to calculate the total cost of movement. In our case, the regions are passed from the planner.

In the *dom*1.*pddl* and *prob*1.*pddl* files, we modified it in this way:

- in the dom1.pddl file, we define two durative actions: **goto_region_and_take_assignment** and **goto_region_and_submit_assignment**. These actions are used to move the robot within the environment, take assignments from student groups, and submit the assignments to the submission desk. To support these actions, we introduce a new type called "assignment" and two new predicates: *(taken ?a - assignment ?r - region)* and *(assignment_at ?a - assignment ?r - region).*

- in the prob1.pddl file, we have defined the goal,it is to visit regions r1 and r2, and to take assignments a1 and a2 to region r5. The and operator is used to specify that all of these conditions must be true for the goal to be achieved. The :metric section of the code defines the metric used to evaluate plans. In this case, the minimize keyword is used to indicate that the goal is to minimize the value of the act-cost fluent.

4 Results

In this section, we present an example of the results.
(The plot is generated by a Python script inside the *visit_domain* folder. To run the script, navigate to the folder using the terminal and execute the command *python3 script.py*)

```
Distance from r0 to r2: 4.937
b (9.000 | 100.000)b (8.000 | 100.000)
Distance from r2 to r5: 7.455
b (7.000 | 200.001)b (6.000 | 200.001)
Distance from r5 to r1: 9.973
b (3.000 | 300.002)b (2.000 | 300.002)
Distance from r1 to r5: 9.973
b (1.000 | 400.003);;;; Solution Found
; States evaluated: 9
; Cost: 32.337
; External Solver: 0.000
; Time 1.01
0.000: (goto_region_and_take_assignment r2d2 r0 r2 a2)  [100.000]
100.001: (goto_region_and_submit_assignment r2d2 r2 r5 a2)  [100.000]
200.002: (goto_region_and_take_assignment r2d2 r5 r1 a1)  [100.000]
300.003: (goto_region_and_submit_assignment r2d2 r1 r5 a1)  [100.000]
```

Figure 1: Terminal output

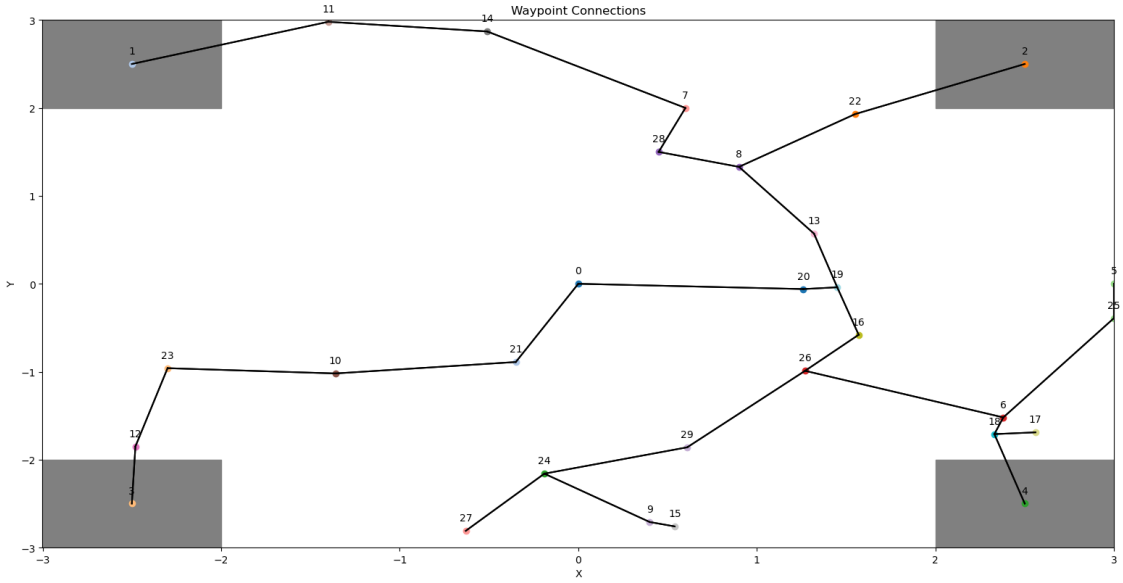


Figure 2: Graph plot