



**Università
di Genova**

Artificial Intelligence for Robotics II

Assignment II Report

The group members :

- HAOUALA Ines
- BENKREDDA Roumaissa
- ABBAD Aicha Manar
- TRIKI Karim

The students ID

5483776
5434673
5565902
5528602

The professor :

Fulvio Mastrogiovanni

Teacher assistant :

Antony Thomas

Academic Year
2022/2023

1- Introduction :

This assignment focuses on developing an intelligent system for task and motion planning in a 2D environment. The objective is to enable a robot (R) to collect assignment reports from four regions and deliver them to a submission desk while minimizing motion costs. The environment consists of a 6m x 6m space, with each region assigned a waypoint. Randomly sampled waypoints are interconnected to create a roadmap. The robot's path is determined by the Euclidean distance between waypoints. It means that if the robot reaches the corresponding waypoint location of the region that region will be considered as being visited. The entire project can be found in the attached Zip.file or in the link to our Github repository : https://github.com/AichaAbbad/airo2_assignment_2

2-Methodology :

- **choice of Algorithm :**

****Explanation of Greedy Algorithm and its Application in Solving the Assignment****

The greedy algorithm is a simple yet powerful approach used in solving optimization problems. It follows a heuristic strategy of making locally optimal choices at each step with the aim of finding a globally optimal solution. In essence, it greedily selects the best available option at each stage, without considering the overall consequences or evaluating alternative paths. <https://www.nature.com/articles/s41598-022-17684-0>

In the context of our assignment on task and motion planning, we employed the greedy algorithm to efficiently guide the robot (R) in collecting assignment reports and delivering them to the submission desk while minimizing motion costs. The objective was to find an optimal path for the robot to traverse the environment, taking into account the lengths of the paths covered.

****Explanation of Single Source Shortest Path Algorithm and its Application in Solving the Assignment****

The Single Source Shortest Path (SSSP) algorithm is a fundamental algorithm in graph theory that aims to find the shortest path from a given source vertex to all other vertices in a weighted directed or undirected graph. It is widely used in various real-world applications, including transportation networks, routing protocols, and task planning in robotics. https://link.springer.com/chapter/10.1007/3-540-68530-8_33

In the context of our assignment, we utilized the SSSP algorithm to optimize the motion planning for a robot tasked with collecting assignment reports from different regions and delivering them to a submission desk. The objective was to minimize the robot's motion cost, which is defined as the length of the path covered by the robot.

To apply the SSSP algorithm, we constructed a roadmap of interconnected waypoints in the 2D environment. Each waypoint represented a node in the graph, and the connections between waypoints were represented as edges with associated weights, which were computed as the Euclidean distance between the waypoints. This graph served as the basis for the SSSP algorithm.

Starting from the initial location of the robot as the source vertex, we employed the SSSP algorithm to calculate the shortest path from the source to all other waypoints in the graph. This allowed us to determine an optimized path for the robot to traverse, considering the motion cost associated with each edge.

By leveraging the SSSP algorithm, we were able to find the most efficient route for the robot to collect two assignment reports and deliver them to the submission desk while minimizing its overall motion cost. This resulted in an intelligent task and motion planning solution that ensured efficient utilization of the robot's resources and reduced unnecessary movements.

- **Visit Domain in pddl :**

PDDL files are essential for solving the assignment by defining the problem domain, initial state, goal, and robot actions. In the "localization" domain, defined in the PDDL files, the environment where the robot operates is represented. The domain file (dom.pddl) specifies object types (robot and region), predicates (taken, robot_in, visited, connected), functions (take, act-cost, triggered, dummy), and durative actions (goto_region, take), which model the task and motion planning problem.

The problem file (prob.pddl) instantiates the domain by defining specific objects (regions, robot) and their initial states. It includes assignments to be collected, region connectivity, the robot's initial location, and cost-related functions. The goal describes the desired state with the robot having taken the assigned reports and visited the corresponding regions.

Additional files (landmark.txt, region_poses, and waypoint.txt) provide information about positions, orientations, and connections of regions and waypoints. These files contribute to accurate motion representation and enable efficient path planning. So according to the waypoint.txt there are four regions defined in which represent the students/assignments reports positions. They are wp1 to wp4. Whereas wp0 represents the origin point with (0,0) coordinates. Please refer to the link of our waypoint : https://1drv.ms/i/s!AopnKXyV7_OcgWg0Xo46EkX_JFAZ?e=8kN62o

Using the PDDL representation, planners and solvers can algorithmically tackle the assignment problem. They interpret the PDDL files, apply search algorithms, and generate action sequences for the robot to efficiently complete tasks while minimizing motion costs. PDDL allows flexibility in problem definition and the application of different planning algorithms to find optimized solutions.

- **Visit Module in cpp :**

The "maincpp" driver code is critical for our project, since it is in charge of arranging the visitation problem-solving process. It contains all the required header files and libraries, such as "VisitSolver.h" and "fstream."

The main function creates a VisitSolver object, and a string variable called "problem." If the right number of command-line arguments is supplied, it assigns the value of argv[1] to the "problem" variable.

The code uses the loadSolver function to load the solver and prepare it to solve the visitation issue. It then generates an "initialState" map to store the initial state fluents and produces a "region" vector to hold waypoints and regions.

3- Running the program :

The first step in running the project successfully is to install the popf-tif planner, which can be found in this repository: https://hub.docker.com/r/hypothe/ai4ro2_2. If you use the docker ubuntu image, take in mind that the planner is based on the Ubuntu-22.04 Docker Official Image, which lacks a graphical interface. To change the planning files, use the following command to mount a shared directory between your host machine and the docker image: "docker run -dit -v path/in_your/host/folder:path/in/docker_container/folder --name your_name hypothe/ai4ro2_2"

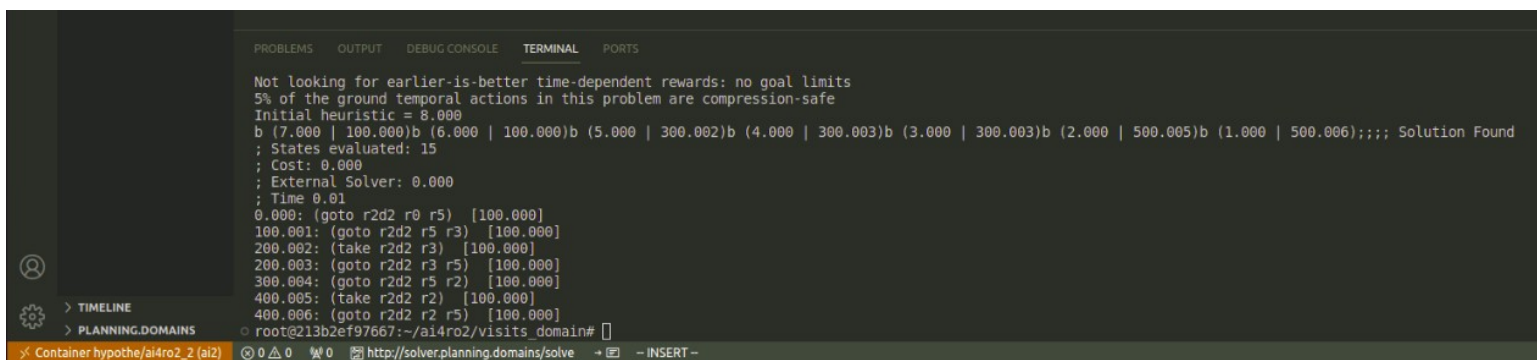
The docker image given by Dr. Antony Thomas, is the simplest method to execute the project. This image already includes all of the necessary setup to execute the project.

Simply extract the visits_domain and visits_module folders included with this repository and replace them in the '/root/ai4ro2' directory. The visits_domain folder contains a PDDL domain as well as issue files, whereas the visits_module folder contains the files required to execute the external planner.

4- Results and Conclusion :

The system's result is shown in the figure below; the total cost of the complete robot's mobility is about "NUMBER," and its breakdown is also shown in the result. Because we utilized linearization approximation, this answer is not ideal because we are utilizing a single source shortest path method algorithm.

The area, waypoint, and landmark files may be changed to add new regions and landmarks to the surroundings, allowing the robot to localize itself and discover the best route. Dr. Antony Thomas laid the groundwork for this research. The outcomes are provided below.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Not looking for earlier-is-better time-dependent rewards: no goal limits
5% of the ground temporal actions in this problem are compression-safe
Initial heuristic = 8.000
b (7.000 | 100.000)b (6.000 | 100.000)b (5.000 | 300.002)b (4.000 | 300.003)b (3.000 | 300.003)b (2.000 | 500.005)b (1.000 | 500.006);;; Solution Found
; States evaluated: 15
; Cost: 0.000
; External Solver: 0.000
; Time 0.01
0.000: (goto r2d2 r0 r5) [100.000]
100.001: (goto r2d2 r5 r3) [100.000]
200.002: (take r2d2 r3) [100.000]
200.003: (goto r2d2 r3 r5) [100.000]
300.004: (goto r2d2 r5 r2) [100.000]
400.005: (take r2d2 r2) [100.000]
400.006: (goto r2d2 r2 r5) [100.000]
root@213b2ef97667:~/ai4ro2/visits_domain#
```