16-782: Planning and Decision-making in Robotics
Homework 1: Robot Chasing Target

The provided code implements a path planning algorithm for a robot in a grid-based environment. It uses a combination of heuristics and A* search to find an optimal path from the current robot position to a target position while avoiding obstacles. The algorithm is divided into several key components:

1. **Heuristic Calculation:**
    - The code calculates heuristics for each cell in the grid using a variant of the A* algorithm, referred to as the **HeurCalculator**. This heuristic estimates the cost from each cell to the goal cell, considering obstacles and time constraints.
2. ***A* Search:***
    - The **Astar** function implements the A* search algorithm to find the optimal path from the current robot position to the target position. It uses the calculated heuristics to guide the search.
    - The code maintains priority queues for open and closed states, and it explores neighboring cells while considering the cost of reaching them.
3. **Search Strategy:**
    - The code employs a goal-directed search strategy. It uses backward heuristics to evaluate possible goals and selects the one with the lowest estimated cost.
4. **Memory Management:**
    - The code dynamically allocates memory for a 2D array named **steps** to keep track of the time steps at which each cell is visited. The memory is correctly deallocated using **delete[]** at the end of the **planner** function to prevent memory leaks.
5. **Data Structures:**
    - The code uses priority queues to manage states in the A* search. Custom comparators (**heur_minimum**, **f_minimum**, **max_g**, and **best_goal_heuristic**) are defined to prioritize states based on heuristics and cost.
6. **Input and Output:**
    - The **planner** function takes various parameters such as the map, collision threshold, grid size, robot's current position, target trajectory, and current time.
    - It computes backward heuristics, selects a goal, runs A* search, and returns the next action for the robot as **action_ptr**.

Efficiency and Quality:

- The code uses heuristics to guide the search, which can significantly improve efficiency by focusing on the most promising paths.
- It correctly handles obstacles and time constraints when evaluating cell validity.
- The goal-directed search strategy helps reduce the search space by considering likely goal locations.
- Memory management is appropriately handled to prevent memory leaks.

Let's dive deeper into the heuristic calculation, A* search, and the search strategy that evaluates possible goals in the provided code:

Heuristic Calculation (HeurCalculator):

The heuristic calculation is a critical component of the path planning algorithm as it guides the search by providing an estimate of the cost from each cell to the goal cell. Here are the key details of the heuristic calculation:

- **Input Parameters:** The **HeurCalculator** function takes several input parameters, including the current goal coordinates, current time, target steps, grid size, collision threshold, and the map of the environment.
- **Data Structures:** It uses priority queues (**Open_list**) to manage states (cells) to explore. It also maintains vectors (**Heuristic** and **Explored**) to store the calculated heuristic values and track explored cells.
- ***A* Variant:*** The heuristic calculation employs a variant of the A* algorithm. It explores neighboring cells while considering the cost of reaching them. It uses priority queues to prioritize cells with lower heuristic values, which leads to a goal-directed search.
- **Cell Validity:** Before evaluating a cell, it checks whether the cell is within the grid boundaries, whether the current time is within the target time steps, and whether the cell is not occupied by an obstacle. This ensures that only valid cells are considered.
- **Heuristic Updates:** The code updates the heuristic value for each cell if a shorter path to that cell is found during the search. This is based on the cost of the path so far and the cost of reaching the neighboring cell.

***A* Search** (**Astar** Function):*

The A* search is responsible for finding the optimal path from the robot's current position to the target position. Here are the key details of the A* search:

- **Input Parameters:** The **Astar** function takes various input parameters, including the current robot position, target coordinates, current time, target steps, grid size, collision threshold, and the map of the environment.
- **Data Structures:** It uses priority queues (**Open_list** and **Closed_list**) to manage states (cells) during the search. The **Open_list** prioritizes cells based on the estimated total cost, considering both the cost to reach the cell (**g**) and the heuristic estimate (**f**).

- **Exploration:** The code explores neighboring cells by considering potential moves in nine directions (**dX** and **dY**). It calculates the cost to reach each neighboring cell, taking into account the current time and obstacle presence.
- **Goal Reached:** If the goal is reached, it backtracks from the goal state to find the optimal path by following the states with the lowest **g** values in the **Closed_list**.
- **Efficiency:** A* search is guided by the heuristic values, which provide a good estimate of the remaining cost to reach the goal. This helps the algorithm prioritize cells that are likely to lead to an optimal path, making the search more efficient.

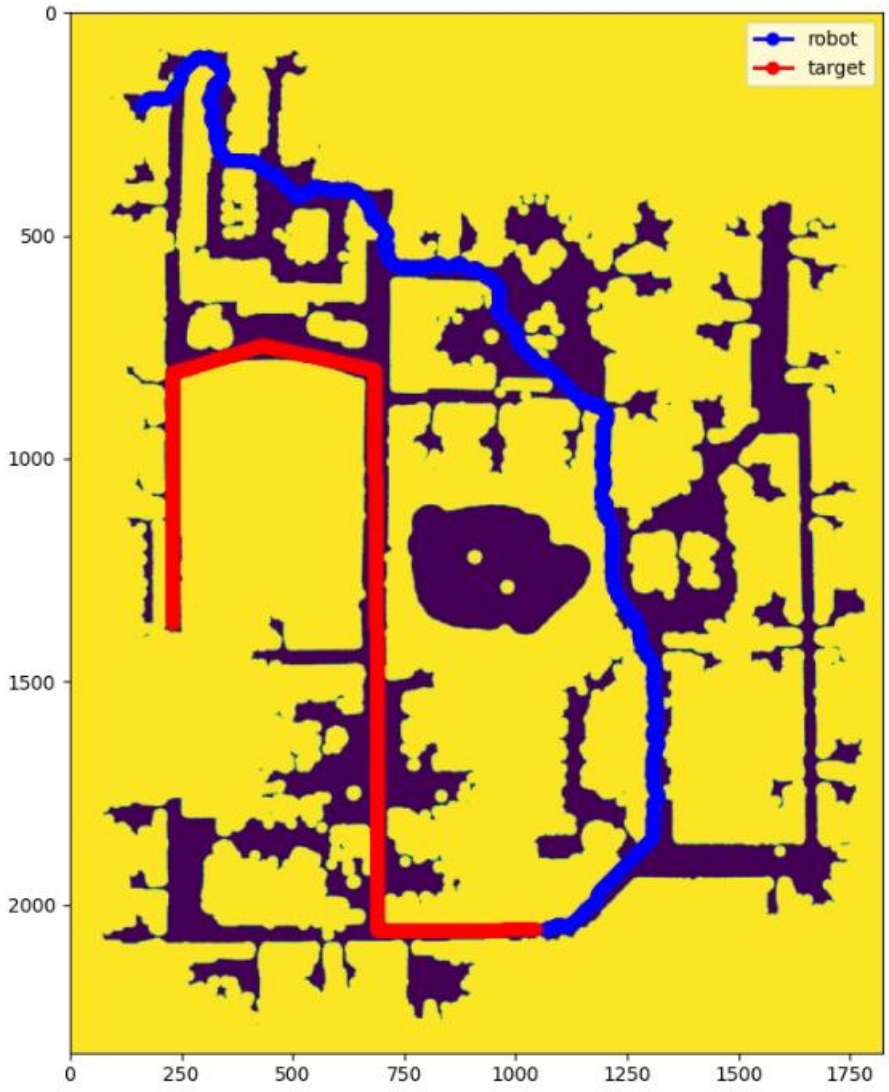
Search Strategy Evaluating Possible Goals:

The search strategy that evaluates possible goals is an essential part of the algorithm. It helps identify promising goal positions for the robot. Here's how this strategy works:

- **Initialization:** When the current time is 0 (**curr_time == 0**), the algorithm calculates backward heuristics for potential goal positions based on the robot's current position and the target trajectory.
- **Goal Evaluation:** For each potential goal, the algorithm calculates a cost estimate that combines the backward heuristic, the time difference between the current time and the last visit to that cell, and the map cost. It then pushes these potential goals into a priority queue (**least_cost_heur**), prioritizing the ones with the lowest cost estimate.
- **Goal Selection:** The algorithm selects the goal with the lowest cost estimate from the **least_cost_heur** queue as the current goal for the robot to reach.
- **Heuristic Update:** After selecting the goal, the algorithm updates the heuristic values based on this goal to guide the subsequent A* search.
- **Efficiency:** This strategy efficiently evaluates potential goals based on the available information, allowing the algorithm to make informed decisions about where the robot should move next.

In summary, the heuristic calculation provides estimates of cell costs, the A* search finds an optimal path, and the search strategy evaluates and selects promising goals for the robot to reach. These components work together to efficiently guide the robot through the grid-based environment while considering obstacles and time constraints.

Results:



Map 1:

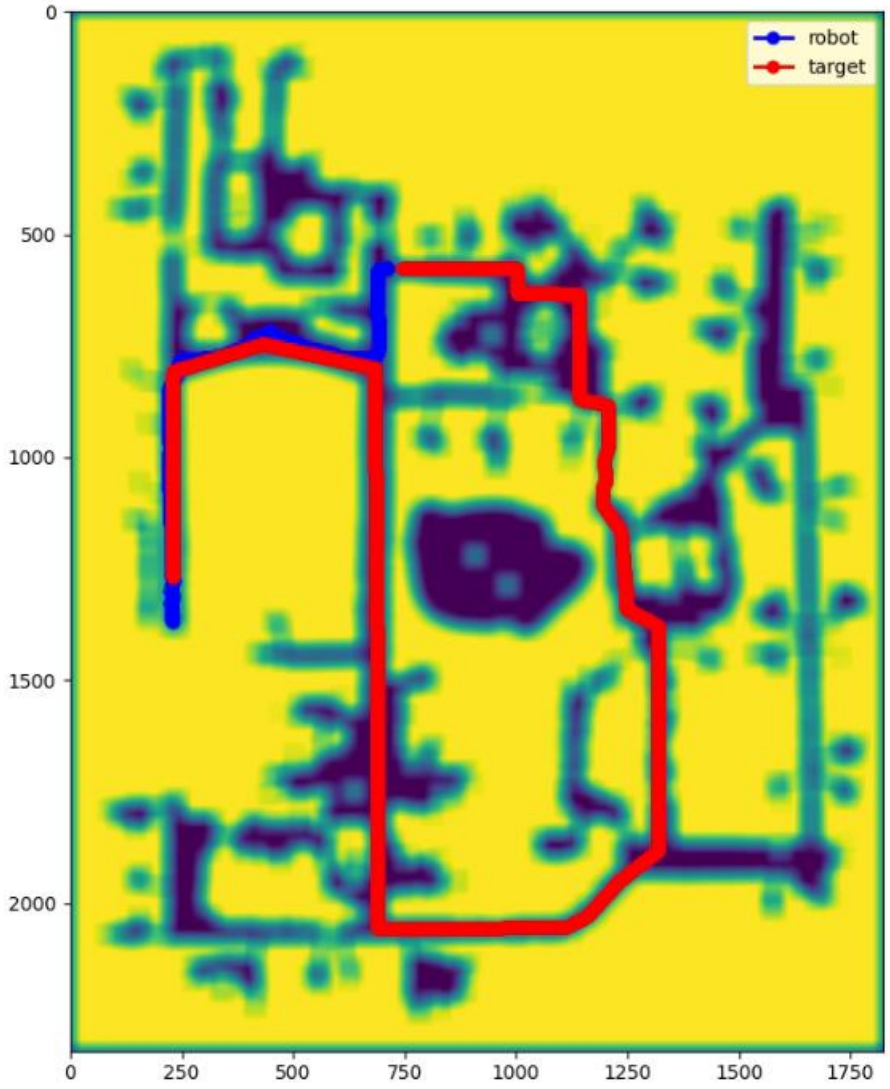
RESULT

target caught = 1

time taken (s) = 2640

moves made = 2639

path cost = 2640



Map2:

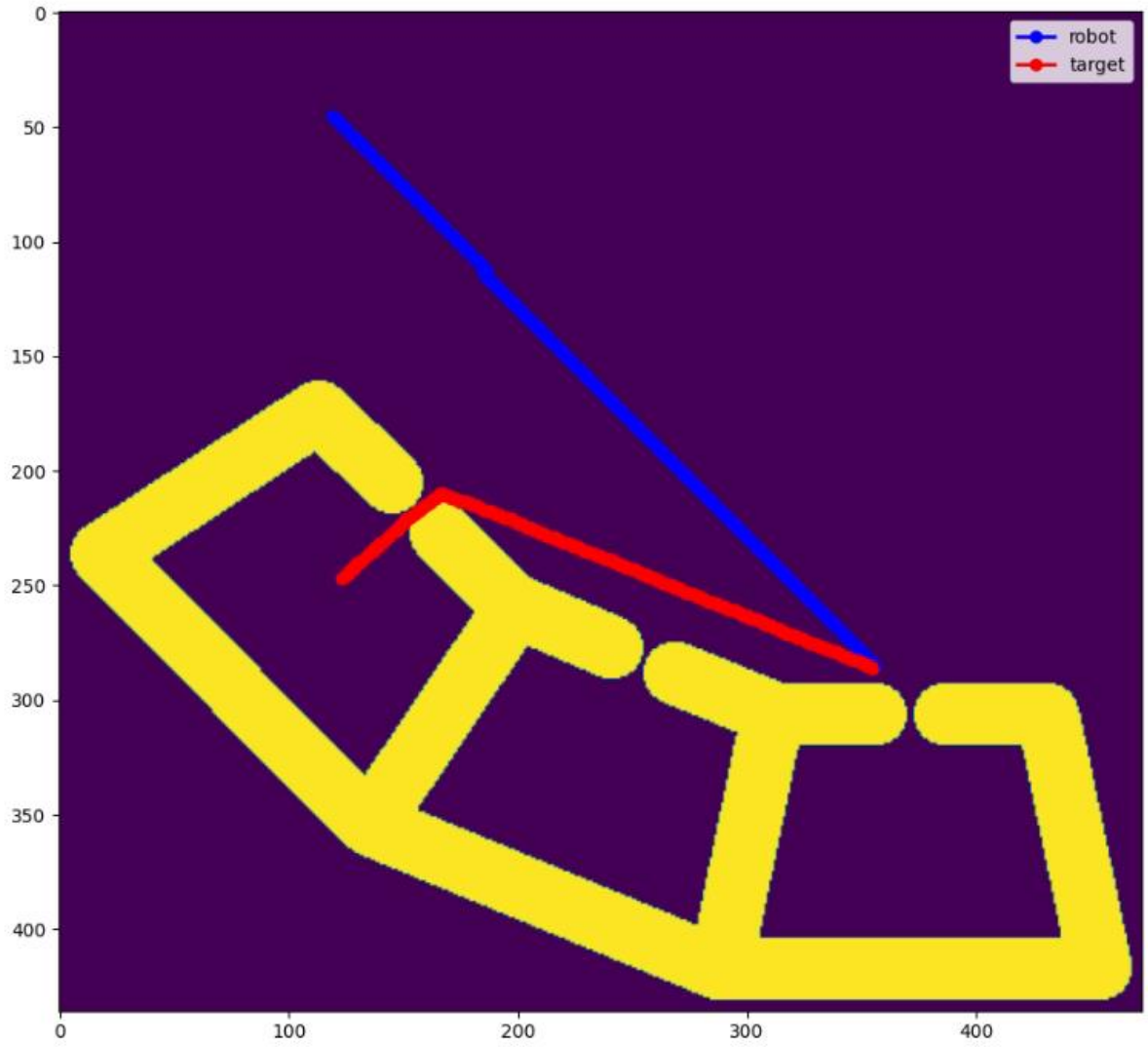
RESULT

target caught = 1

time taken (s) = 4672

moves made = 1235

path cost = 4454325



Map3:

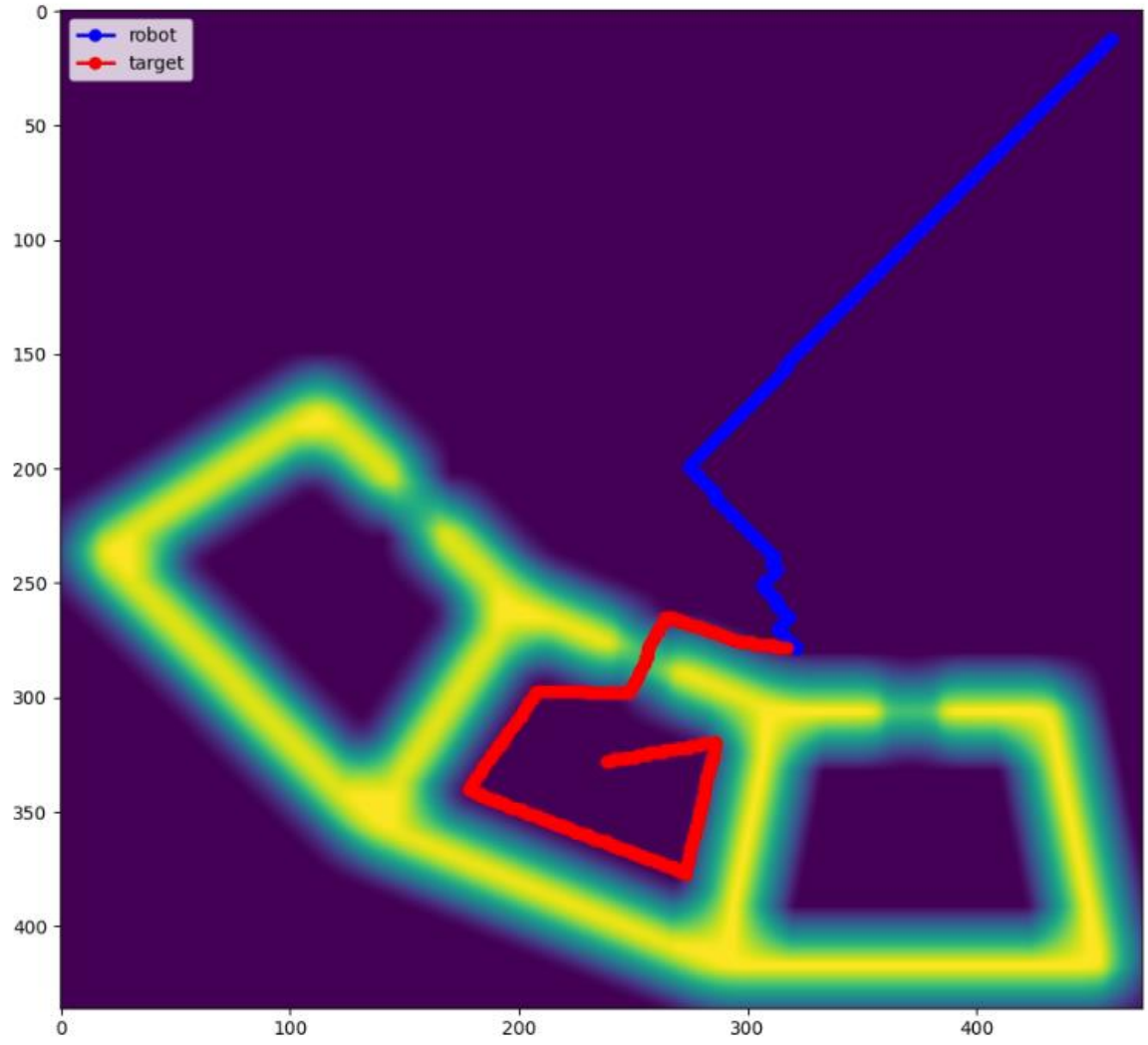
RESULT

target caught = 1

time taken (s) = 243

moves made = 242

path cost = 243



Map4:

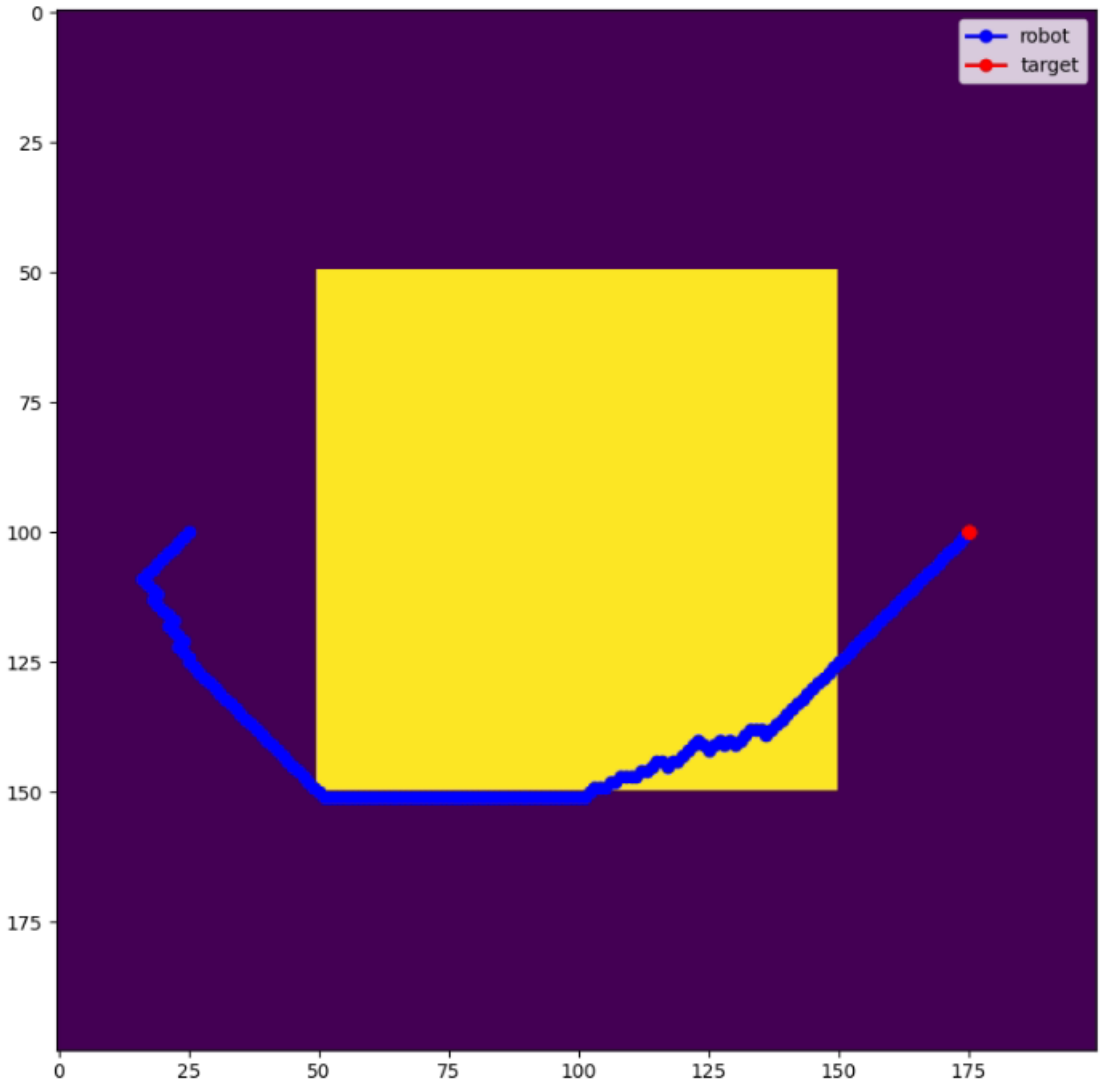
RESULT

target caught = 1

time taken (s) = 325

moves made = 253

path cost = 18332



Map5:

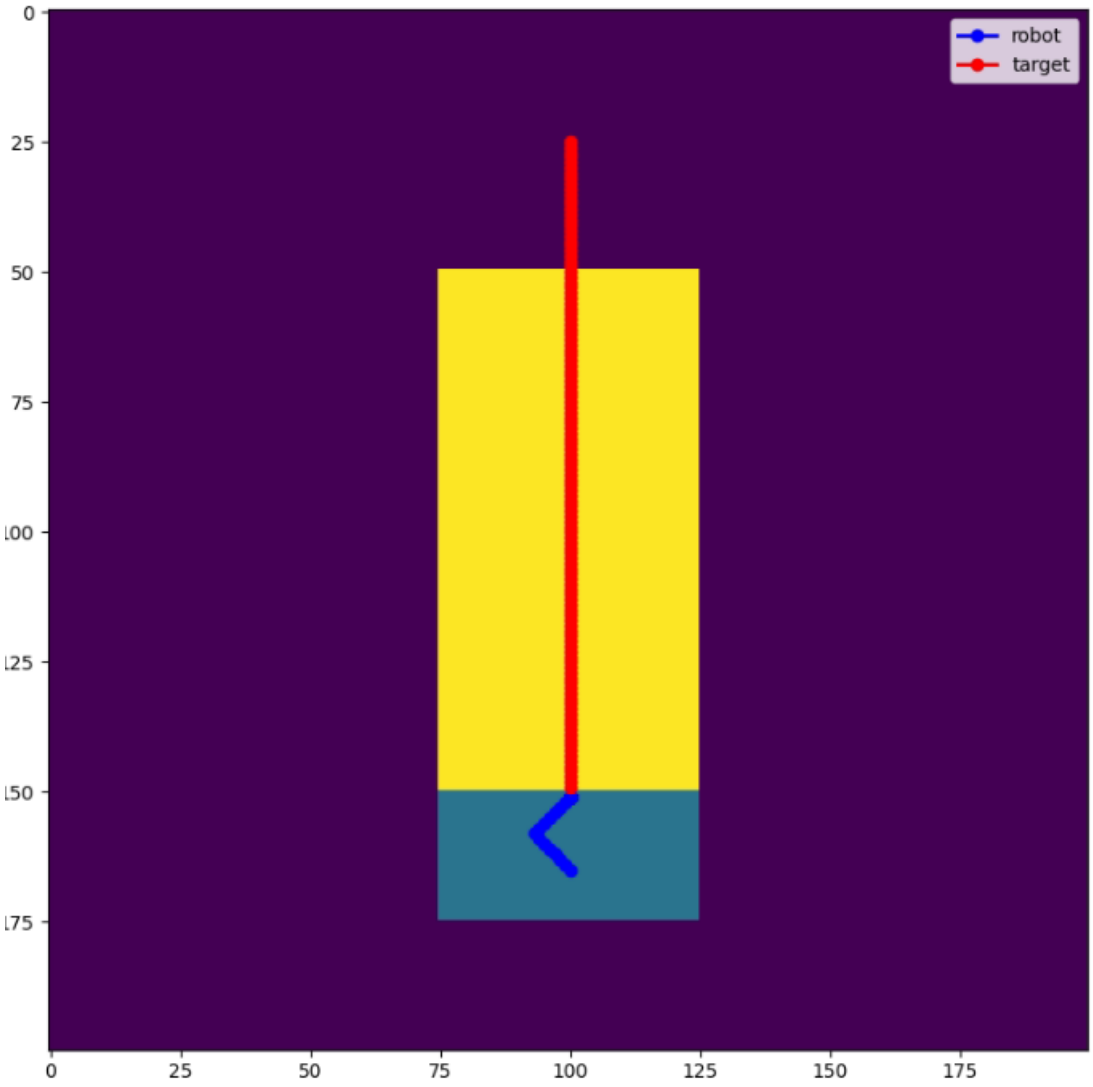
RESULT

target caught = 1

time taken (s) = 175

moves made = 175

path cost = 2576



Map6:

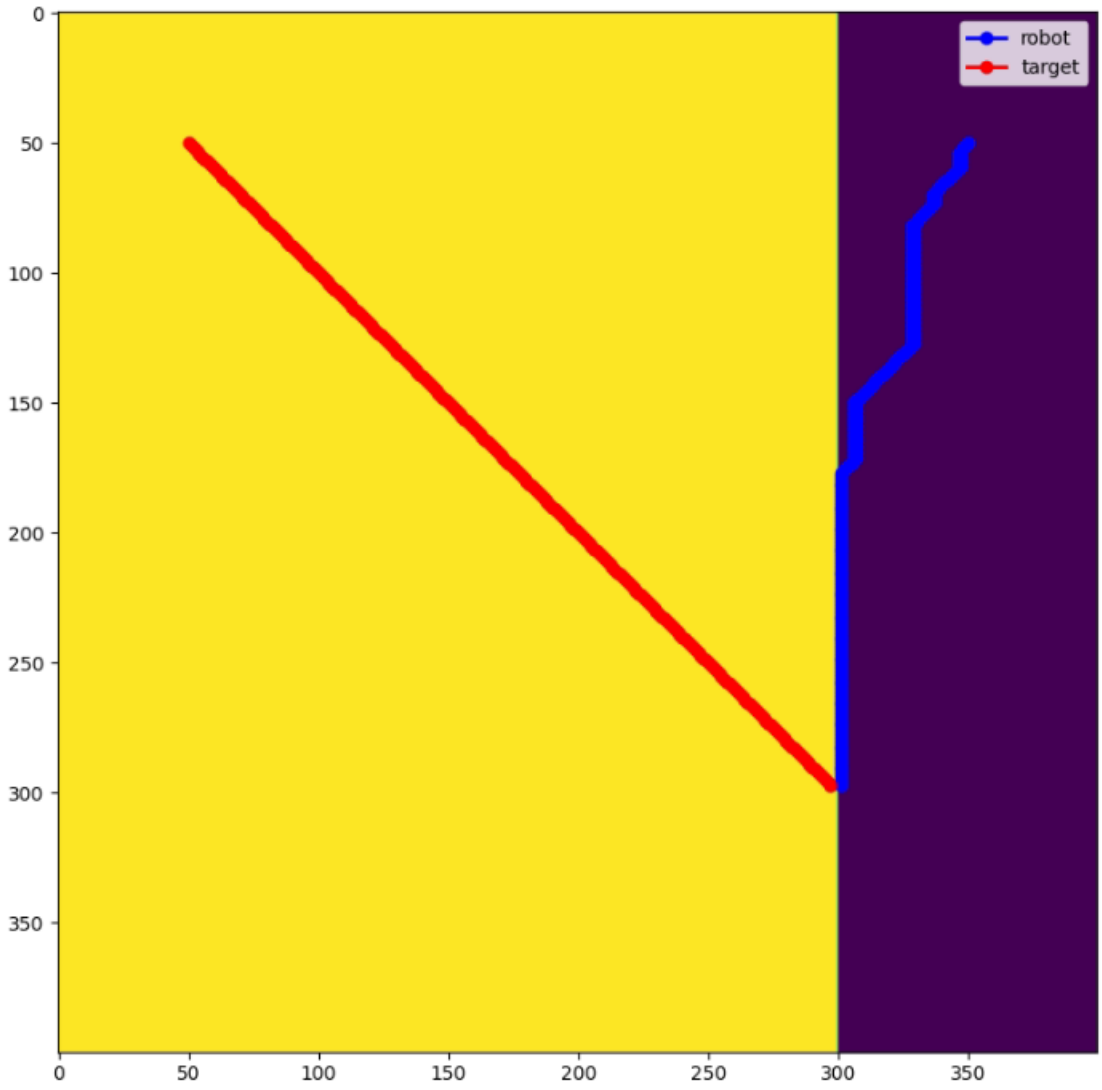
RESULT

target caught = 1

time taken (s) = 126

moves made = 14

path cost = 2520



Map7:

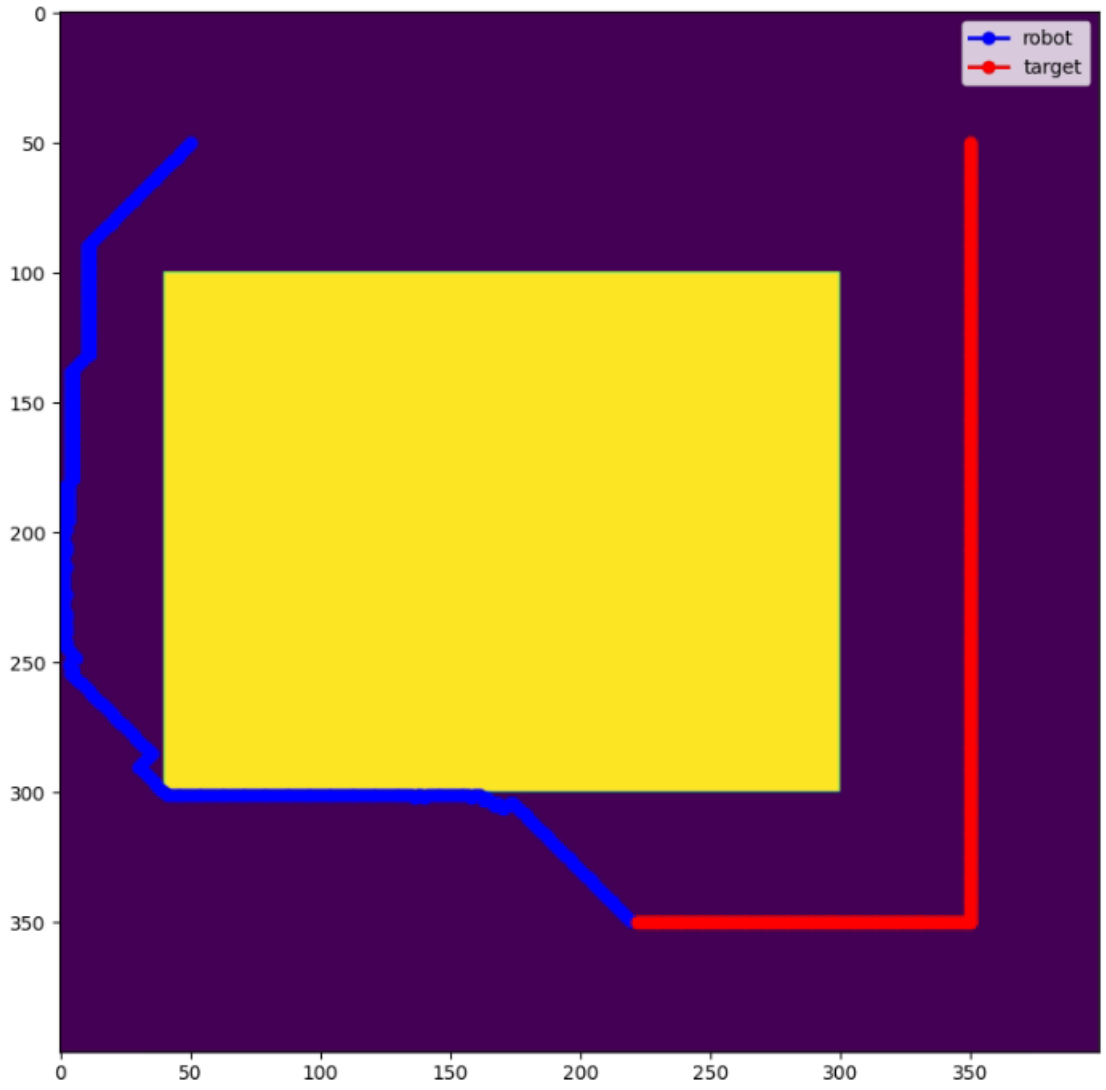
RESULT

target caught = 1

time taken (s) = 251

moves made = 251

path cost = 251



Map8:

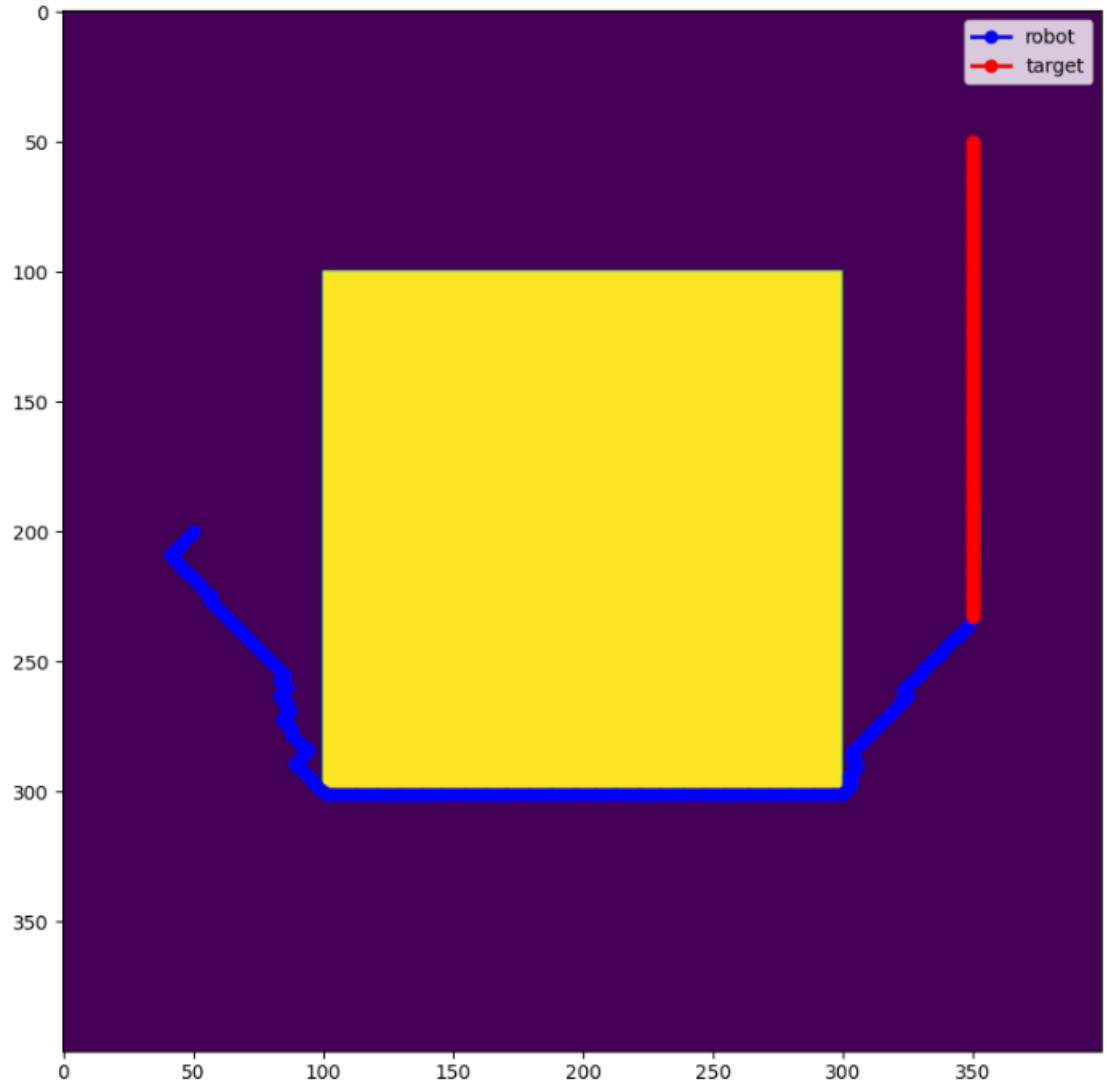
RESULT

target caught = 1

time taken (s) = 431

moves made = 430

path cost = 431



Map9:

target caught = 1

time taken (s) = 368

moves made = 367

path cost = 368

Execution:

To compile the cpp code:

>> g++ runtest.cpp planner.cpp

To run the planner:

>> ./a.out map3.txt;

To visualize the robot and target's trajectory:

>> python visualizer.py map3.txt;