Assignment 3: Robot Chasing Target using grid based planner

course code:-SC-627

MARCH 2024

1 TASK

Write a planner for a point robot to catch a moving target in a 2D grid world. The gridworld is 8-connected (that is, the robot can only move by at most one unit along the X and/or Y axis). During execution, the planner will be given a costmap and the collision threshold. The costmap contains the associated cost for moving through each cell in the grid. This cost will be a non-negative integer. Any cell with a cost greater than or equal to the collision threshold is to be considered an obstacle that the robot cannot traverse. The biggest grid in this assignmentover is around 2000x2000 cells.

The planner will be given the start position of the robot, along with the trajectory of the moving target as a sequence of positions (e.g., [(5,6), (5,7), (4,5)]). The target will also be moving on the 8-connected grid, and has a speed of one step per second.

2 code

Your code is within the folder `code`. The `planner` function must output a single robot move. The planner should reside in `planner.cpp` file. Currently, the file contains a greedy planner that always moves the robot in the direction that decreases the distance in between the robot and the target. The `planner` function (inside `planner.cpp`) is as follows:

```
static void planner(
    double *map,
    int *collision_thresh,
    int x_size,
    int y_size,
    int robotposeX,
    int robotposeY,
    int target_steps,
    double* target_traj,
    int targetposeX,
    int targetposeY,
    int curr_time,
    double* action_ptr
)
```

3 Inputs:

Each cell in the map of size (x_size, y_size) is associated with the cost of moving through it. This cost is a positive integer. The cost of moving through cell (x, y) in the map should be accessed as:

(`int`)`map`[`GETMAPINDEX`(x, y, x_size, y_size)].

If it is less than *collision thresh*, then the cell (x, y) is free. Otherwise, it is an obstacle that the robot cannot traverse. Note that cell coordinates start with 1. In other words, x can range from 1 to x_size. The target's trajectory *target traj* of size *target steps* is a sequence of target positions (for example: $(2, 3), (2, 4), (3, 4)$). At the current time step *current time*, the current robot pose is given by $(robotposeX, robotposeY)$ and the current target pose is given by $(targetposeX, targetposeY)$. The target will also be moving on the 8-connected grid, at the speed of one step per second along its trajectory. Therefore, at the next second, the target will be at $(currenttime + 1)$th step in its trajectory *target traj*.

You are provided with a few test maps. Target, robot, and map cost information is specified in text files named *map*.txt*. Specifically, the format of the text file is:

- The letter N followed by two comma separated integers on the next line (say N1 and N2 written as N1,N2). This is the map's size.
- The letter C followed by an integer on the next line. This is the map's collision threshold.
- The letter R followed by two comma separated integers on the next line. This is the starting position of the robot in the map.
- The letter T followed by a sequence of two comma separated integers on each line. This is the trajectory of the moving object.
- The letter M followed by N1 lines of N2 comma separated floating point values per line. This is the map.

`runtest.cpp` parses the text files, and calls your planner function (with these inputs) once per simulation step.

4 Output

At every simulation step, the planner function should output the robot's next pose in the 2D vector action ptr. The robot is allowed to move on an 8-connected grid. All the moves must be valid with respect to obstacles and map boundaries (see the current planner inside planner.cpp for how it tests the validity of the next robot pose).

`runtest.cpp` evaluates and prints four values - a boolean specifying whether the object was caught, and three integers specifying the time taken to run the test, the number of moves made by the robot, and the cost of the path traversed by the robot.

5 Frequency of move

The planner is supposed to produce the next move within 1 second. Within 1 second, the target also makes one move. If the planner takes longer than 1 second to plan, the target will have moved by a longer distance in the meantime. In other words, if the planner takes K seconds (rounded up to the nearest integer) to plan the next move of the robot, then the target will move by K steps in the meantime.

Note: After the last cell on its trajectory, the object disappears. So, if the given object's trajectory is of length 40, then at time step $t = 41$ the object disappears and the robot can no longer catch it. This means for a moving object trajectory that is T steps long, your planner has at most T seconds to find (and execute) a full solution.

6 Execution

The code directory contains a few map files to test your planner. To compile the C++ 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
```

Currently, the planner greedily moves towards the last position on the moving object's trajectory. If you run it as is, the planner only succeeds on `map4`. `map1` and `map2` are larger, and it is more difficult to catch the target.

7 submission

same as previous two assignment provide report and all files in a single zip file naming <YOUR NAME_ROLL NO>.zip

1. Your writeup in `<YOUR NAME_ROLL NO>.pdf`. This should contain:

- A summary of your approach for solving this assignment.
- The results for all maps (whether the object was caught, the time taken to run the test, number of moves made by the robot, and the cost of the path traversed by the robot).
- Instructions for how to compile your code.

For your planner summary, we want details about:

- The algorithm you implemented.
- Heuristics used.
- Any efficiency tricks.

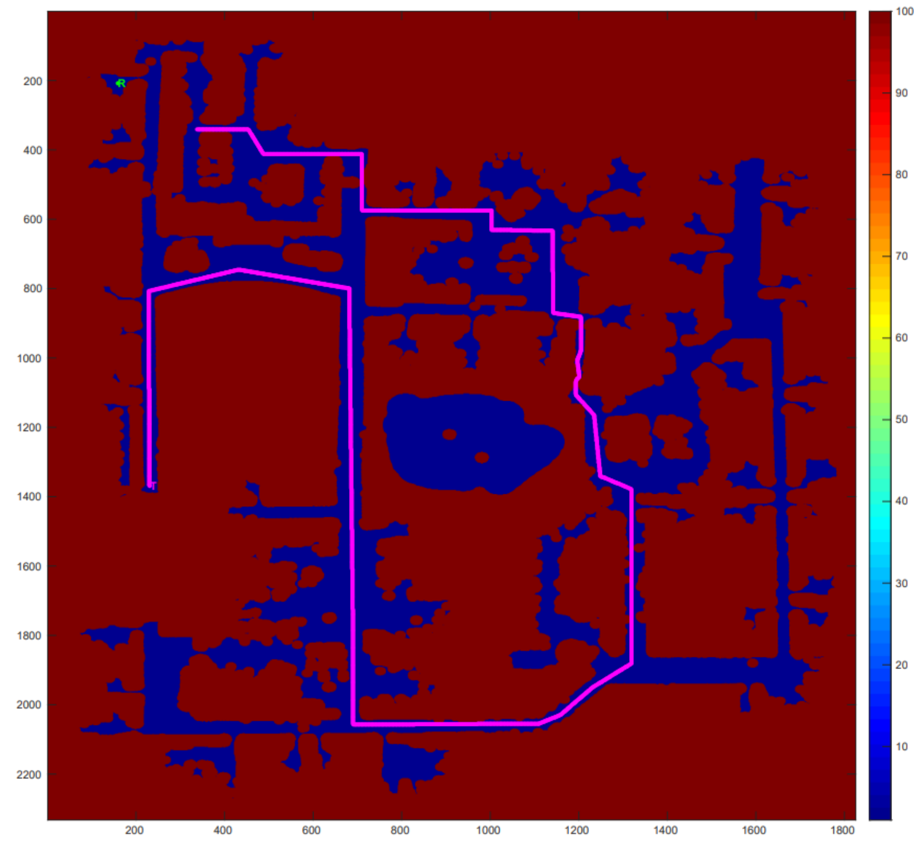

Figure 1: Image of information in `map1.txt`. The green R marks the starting position of the robot, the magenta T marks the starting position of the target, the magenta line is the target's trajectory. Blue cells have cost 1, red cells have cost 100, collision threshold is 100

- Include plots of the maps overlaid with the object and solved robot trajectories. Please do not include the map text files in your submission.

Basically, any information you think would help us understand what you have done and gauge the quality of your assignment submission.

8 grading

The grade will depend on two things:

- How well-founded the approach is. In other words, can it guarantee completeness (to catch a target). whether it is optimal or not you have to justify.
- How much cost the robot incurs while catching the target.

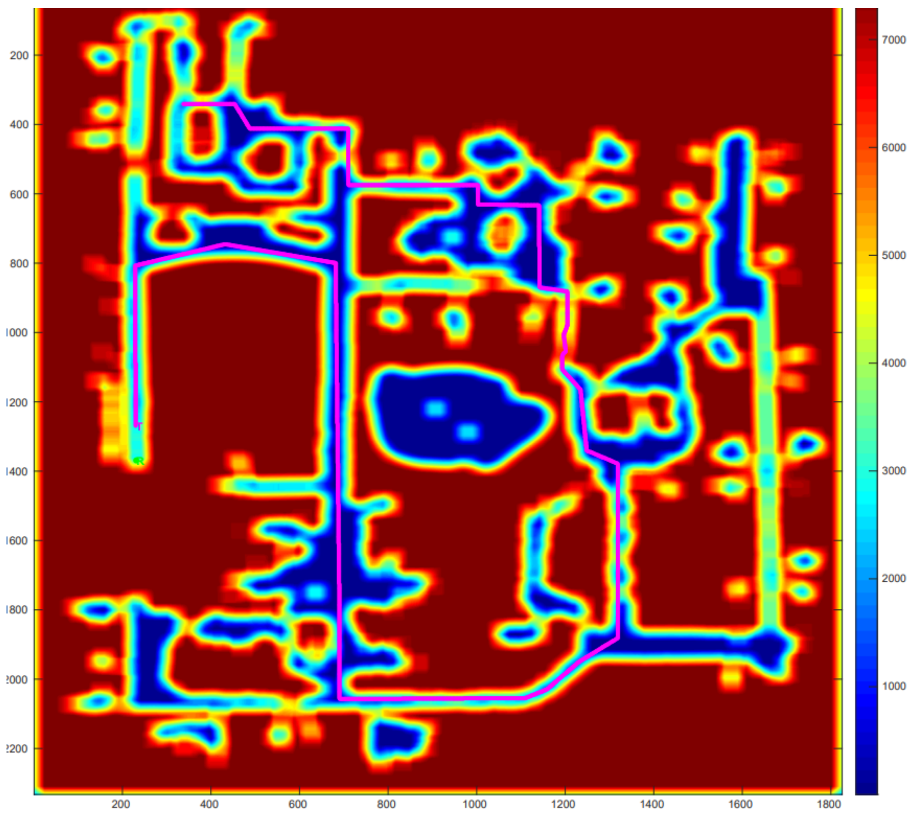

Figure 2: Image of information in `map2.txt`. The green R marks the starting position of the robot (directly below the starting position of the target), the magenta T marks the starting position of the target, the magenta line is the target's trajectory. Cells have cost between 1 and 7497, inclusive. Collision threshold is 6500

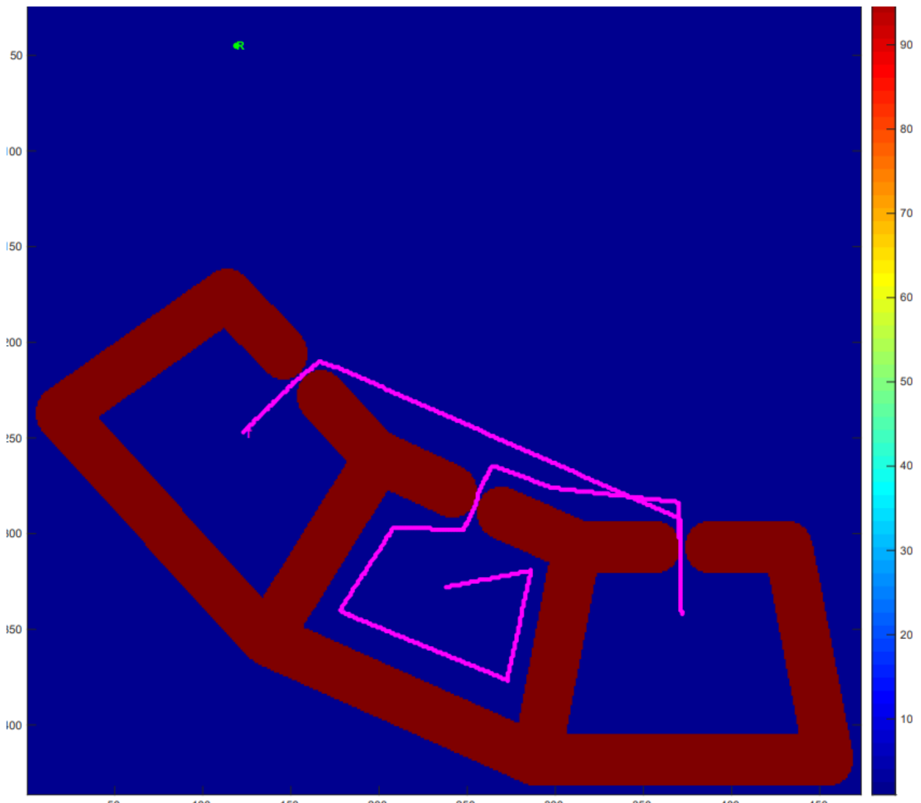

Figure 3: Image of information in `map3.txt`. The green R marks the starting position of the robot, the magenta T marks the starting position of the target, the magenta line is the target's trajectory. Blue cells have cost 1, red cells have cost 100, collision threshold is 100.

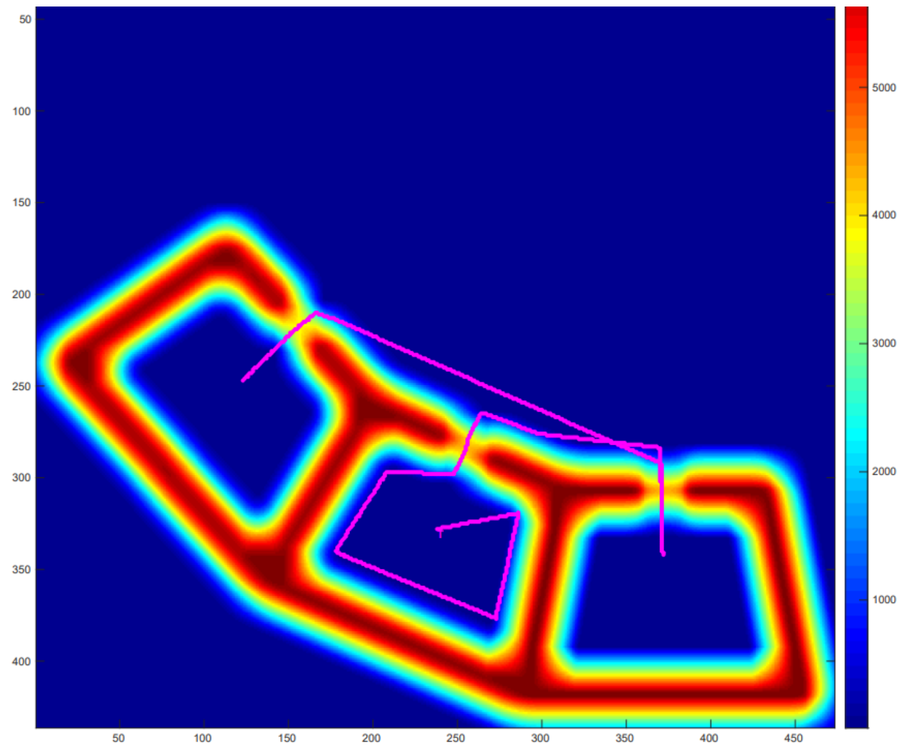

Figure 4: Image of information in `map4.txt`. The green R marks the starting position of the robot, the magenta T marks the starting position of the target, the magenta line is the target's trajectory. Cells have cost between 1 and 6240, inclusive. Collision threshold is 5000