16-350, Spring’21, Planning Techniques for Robotics

Homework I:
Planning for a robot trying to catch a target
DUE: March 1st (Monday) at 11:59PM

Task

Write a planner for the robot to catch a target in a 2D grid world, while minimizing the cost incurred by the robot. At every time step, the target moves by one step along a given trajectory. At every time step, the robot can make a move on an 8-connected grid.

Code

The planner function must output a single robot move. The planner should reside in ***planner.cpp*** (or ***robotplanner.m***) if you prefer to write it in MATLAB. We strongly suggest you use C/C++ as it is highly unlikely something written in MATLAB will be fast enough to solve the problem). Currently, the planner greedily moves towards the last position on the target's trajectory. 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 current time,
        double *action ptr
        )
{}
```

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 (*current_time* + 1)th step in its trajectory *target_traj*.

The gridworld will be of size M by N, with the bigger of two gridworlds in this homework around 2,000 x 2,000 units. You are provided with four maps. Target, robot, and map cost information are specified in text files named ***map*.txt***. Specifically, the format of the text file is:

1. The letter ***N*** followed by two comma separated integers on the next line (say N1 and N2 written as *N1,N2*). This is the size of the map.
2. The letter ***C*** followed by one integer on the next line. This is the collision threshold for the map.
3. The letter ***R*** followed by two comma separated integers on the next line. This is the starting position of the robot in the map.
4. The letter ***T*** followed by a sequence of two comma separated integers on each line. This is the trajectory of the moving object.
5. The letter ***M*** followed by N1 lines of N2 comma separated floating point values per line. This is the map.

Images of these maps are at the end of this document.

The file ***readproblem.m*** included in the homework packet parses the text files and returns all of the data. It is called inside the ***runtest.m*** script once at the beginning.

Outputs

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).

The ***runtest.m*** script returns four values - a boolean specifying whether the object was caught, the number of time steps taken to run the test, number of moves made by the robot, and the cost of the path traversed by the robot. It also prints this information out before returning.

Frequency of Moves

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 = 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 plan (and execute) a full solution.

Execution

Following are a few examples of running the tests from within MATLAB (in the same directory where all source files are placed).

To compile the cpp code:
```
>> mex planner.cpp
```
To run the planner:
```
>> runtest(`map3.txt');
```
Much larger map and more difficult to catch the target:
```
>> runtest(`map1.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.

Submission

You will submit this assignment through Gradescope. You need to upload one ZIP file named *<Andrew ID>.zip*. This should contain:

1. A folder named *code* that contains all code files, including but not limited to, the ones in the homework packet. If there are subfolders, your code should handle relative paths. We will only compile your MEX code (according to instructions in the writeup) and execute the ***runtest.m*** script.
2. A PDF file named *<Andrew ID>.pdf*. This should contain a summary of your approach for solving this homework, the results for all four 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), and most importantly instructions for how to compile your code. This should be one line for us to execute in MATLAB of the form "*mex <file 1> <file 2> ... <file N>*".

- For your planner summary, we want details about the algorithm you implemented, data structures used, heuristics used, any efficiency tricks, memory management details etc. Basically any information you think would help us understand what you have done and gauge the quality of your homework submission.
- Include plots of the maps overlaid with the object and solved robot trajectories.

Please **do not** include the map text files in your submission.

Grading

The grade will depend on two things:

1. How well-founded is the approach? In other words, can it guarantee completeness (finds the solution if one exists), can it provide sub-optimality or optimality guarantees on the paths it produces, can it scale to large environments?
2. How much cost the robot incurs while catching the target?

Note: To grade your homework and to evaluate the performance of your planner, we may use different/larger maps than the ones provided in the directory. The only promise we can make is that size of the map will not be larger than 5000 by 5000 cells.

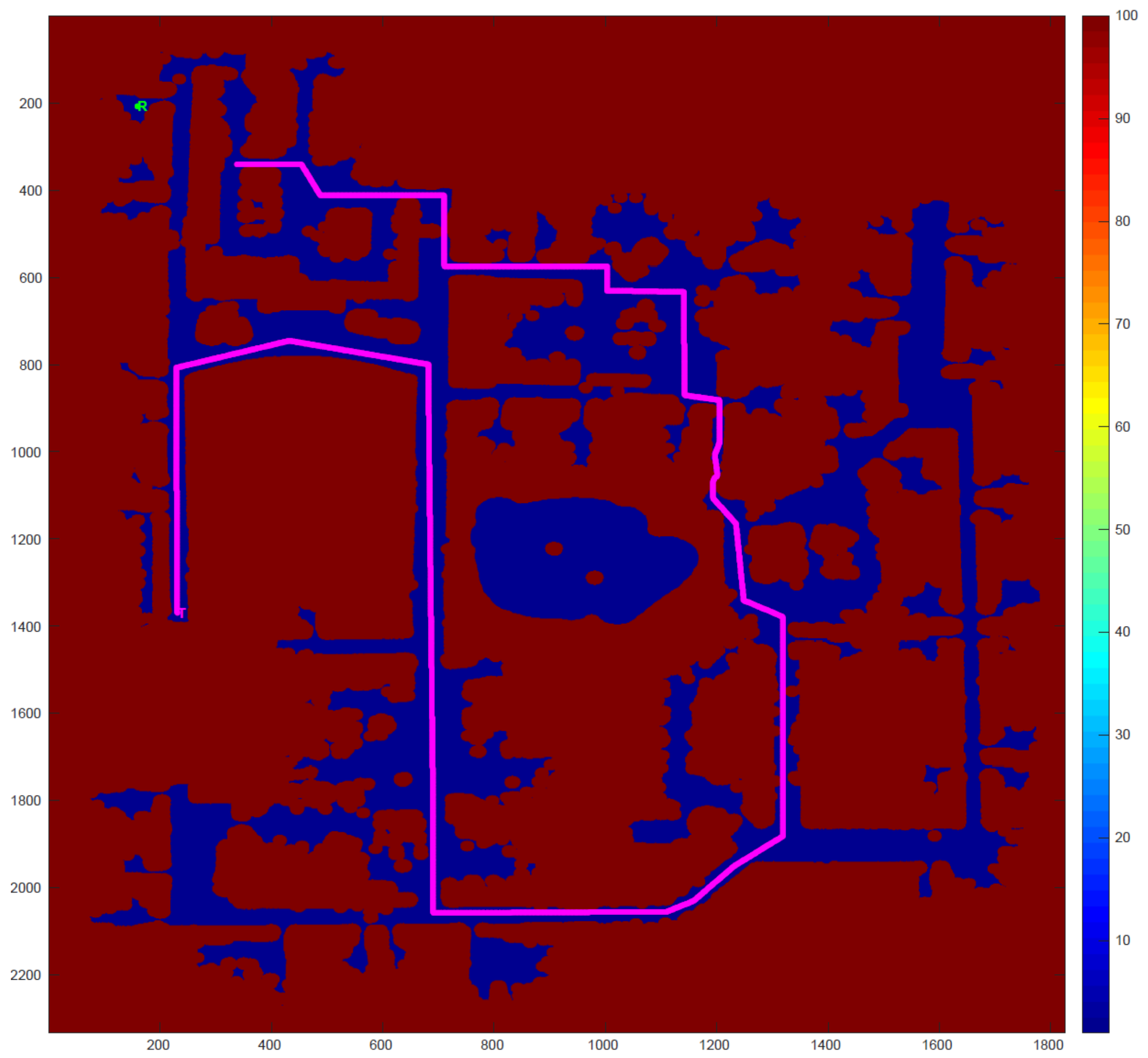

Image of information in ***map1.txt***. The green dot is the starting position of the robot, magenta dot is the starting position of the moving object, dashed magenta line is the object's trajectory. Blue cells have cost 1, red cells have cost 100, collision threshold is 100.

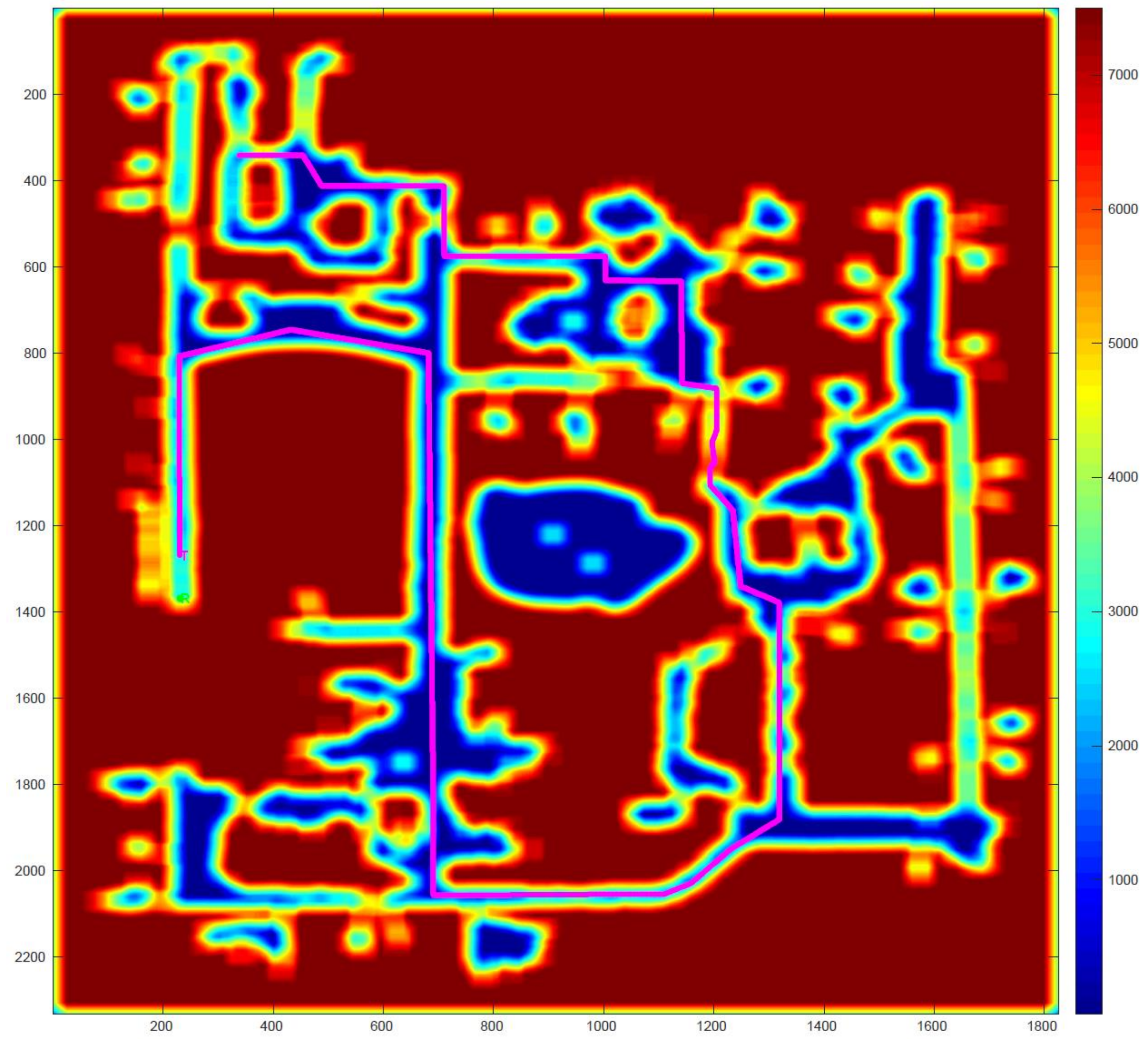

Image of information in ***map2.txt***. The green dot is the starting position of the robot (directly below the object dot), magenta dot is the starting position of the moving object, dashed magenta line is the object's trajectory. Cells have cost between 1 and 7497, inclusive. Collision threshold is 6500.

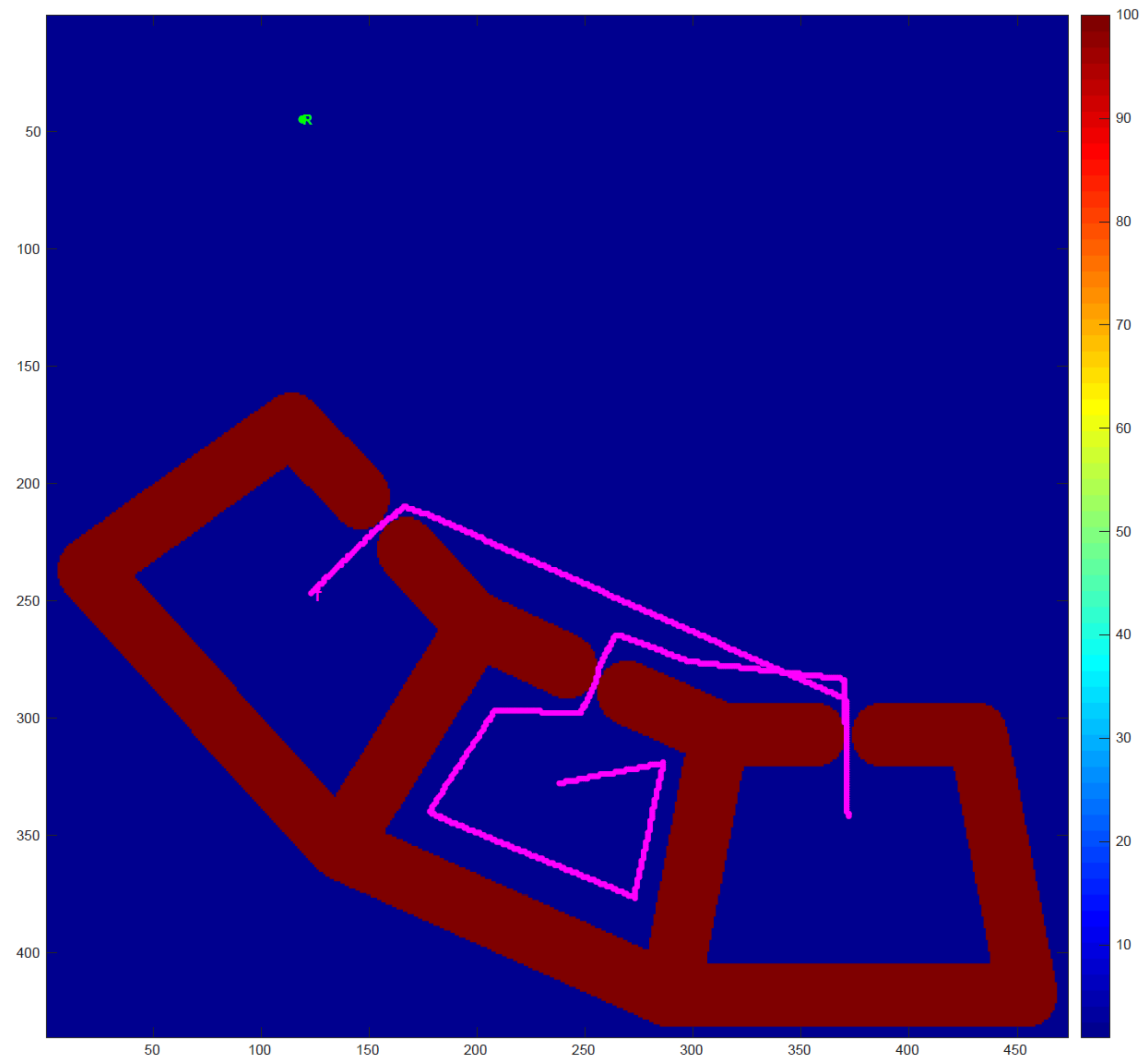

Image of information in ***map3.txt***. The green dot is the starting position of the robot, magenta dot is the starting position of the moving object, dashed magenta line is the object's trajectory. Blue cells have cost 1, red cells have cost 100, collision threshold is 100.

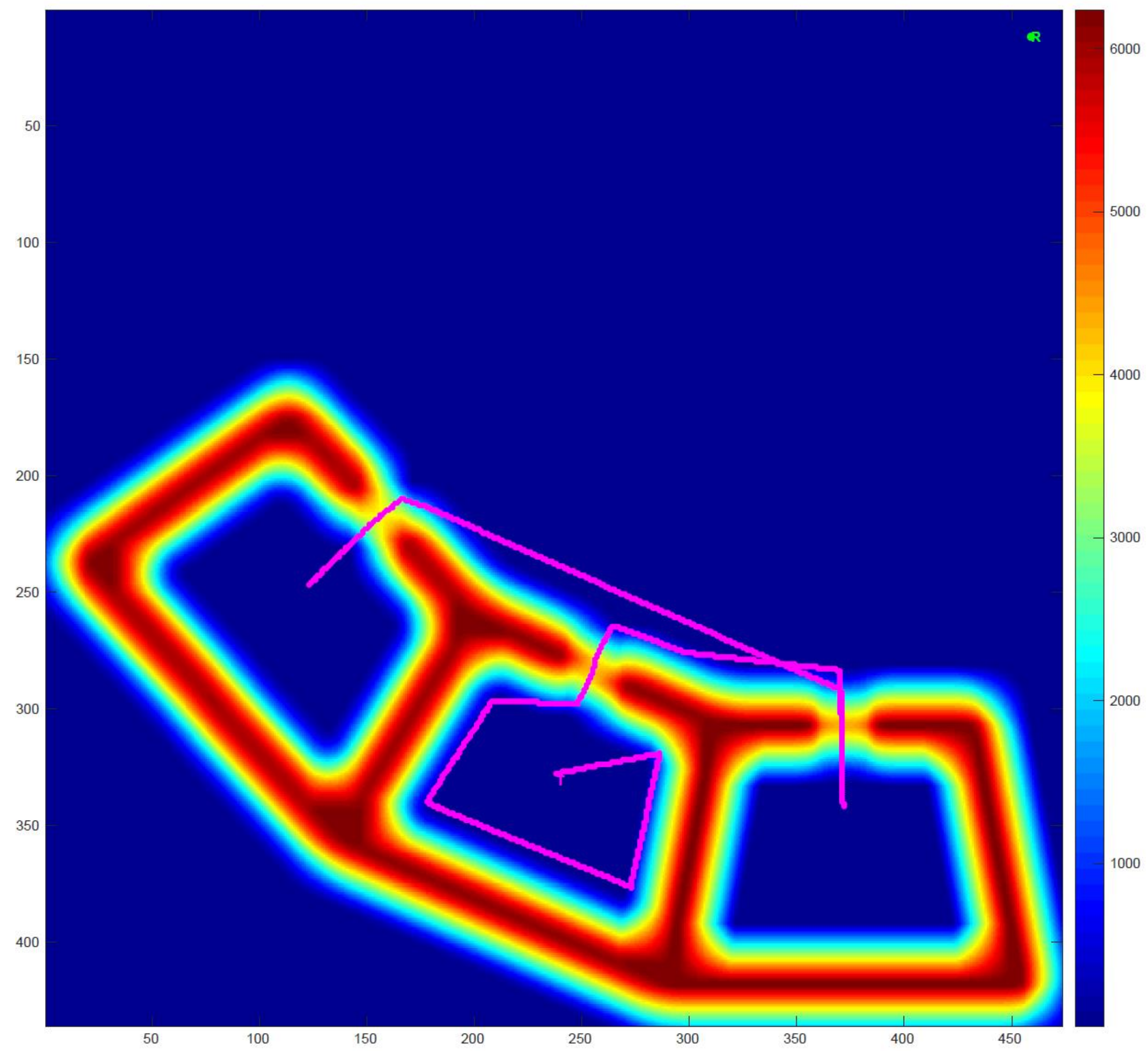

Image of information in ***map4.txt***. The green dot is the starting position of the robot, magenta dot is the starting position of the moving object, dashed magenta line is the object's trajectory. Cells have cost between 1 and 6240, inclusive. Collision threshold is 5000.