16-350: Planning Techniques for Robotics
Homework 1: Robot Chasing Target
Due: Feb 19 (Wed), 11:59pm

Professor: Maxim Likhachev *Spring 2020 TA: Navyata Sanghvi*

1 Undergraduate Assignment

1.1 Task:

Write a planner for the robot to catch a target in a 2D grid world. At every time step, the target moves randomly in one of four directions. At every time step, the robot is allowed to make a move on an 8-connected grid.

1.2 Code:

Your code is within the folder `undergrad`. The planner function must output a single robot move. The planner should reside in `planner.cpp` file (or `robotplanner.m` file if you prefer to write it in MATLAB). 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 x_size,
     int y_size,
     int robotposeX,
     int robotposeY,
     int goalposeX,
     int goalposeY,
     char *p_actionX,
     char *p_actionY
     )
{}
```

1.3 Inputs:

The `map` of size `(x_size, y_size)` contains information on what are obstacles and what are not. Value of cell `(x, y)` in the `map` should be accessed as:

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

If it is equal to 0, then the cell `(x, y)` is free. Otherwise, it is an obstacle. Note that cell coordinates start with 1. In other words, `x` can range from `1` to `x_size`. The current robot pose is given by `(robotposeX, robotposeY)` and the current target pose is given by `(goalposeX, goalposeY)`.

1.4 Outputs:

The planner function should return what the robot should do next using the pointers `p_actionX` and `p_actionY`. Specifically, `*p_actionX` should be set to the desired change in robot's x coordinate (-1, 0, or 1) and `*p_actionY` should be set to the desired change in robot's y coordinate (-1, 0, or 1). 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).

1.5 Frequency of Moves:

The planner is supposed to produce the next move within 200 milliseconds. Within 200 milliseconds, the target also makes one move. The target can only move in four directions. If the planner takes longer than 200 milliseconds to plan, the target will have moved by a longer distance in the meantime. In other words, if the planner takes N*200 milliseconds (rounded up) to plan the next move of the robot, then the target will move by N steps in the meantime.

1.6 Execution:

The `undergrad` directory contains two map files (`map1.txt` and `map3.txt`). 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:

```
>> robotstart = [10 10];
>> targetstart = [200 100];
>> runtest('map3.txt', robotstart, targetstart);
```

Same map but more difficult to catch the target:

```
>> robotstart = [250 250];
>> targetstart = [400 400];
>> runtest('map3.txt', robotstart, targetstart);
```

Much larger map and more difficult to catch the target:

```
>> robotstart = [700 800];
>> targetstart = [850 1700];
>> runtest('map1.txt', robotstart, targetstart);
```

Executing `runtest` multiple times will show you that sometimes the robot catches the target, and sometimes it does not. The letters R and T indicate the current positions of the robot and target respectively.

1.7 Submission:

You will submit this assignment through Gradescope (Entry Code: 9YJ642). You must upload one ZIP file named `<andrewID>.zip`. This should contain:

1. A folder `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 using `mex` and execute `runtest.m` in the parent `code` directory.
2. Your writeup in `<andrewID>.pdf`. This should contain a summary of your approach for solving this homework, the results for both 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 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>`. We should be able to run the `mex` command to compile your code from within MATLAB under Linux!
   - We want results for the above examples of `robotstart, targetstart`. You are free to experiment with and include results from other start positions for the robot and target.
   - 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.

1.8 Grading:

The grade will depend on two things:

1. How well-founded the approach is. In other words, can it guarantee completeness (to catch a target), can it provide sub-optimality or optimality guarantees on the paths it produces, can it scale to large environments, can it plan within 200 milliseconds?
2. How fast (and how consistently) it catches 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, and a different strategy for how the target moves than the one in `targetplanner.m`. The only promise we can make is that the target will only move in four directions and the size of the map will not be larger than 5000 by 5000 cells.

2 Graduate Assignment (Bonus for Undergraduates)

2.1 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 is allowed to make a move on an 8-connected grid.

2.2 Code:

Your code is within the folder `grad`. The planner function must output the next position of the robot. The planner should reside in `planner.cpp` file (or `robotplanner.m` file 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
     )
{}
```

2.3 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 `targe_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)`$^{\texttt{th}}$ step in its trajectory `target_traj`.

You are provided with four maps. Target, robot and map cost information is 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 map's size.
2. The letter C followed by an integer on the next line. This is the map's collision threshold.
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.

The file `readproblem.m` parses the text files and returns all of the data. In `runtest.m`, this parsing is performed once at the beginning, after which your planner function is called (with these inputs) once per simulation step.

2.4 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, and three integers specifying the time 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.

2.5 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 find (and execute) a full solution.

2.6 Execution:

The `grad` directory contains four map files. 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`.

2.7 Submission:

You will submit this assignment through Gradescope (Entry Code: 9YJ642). You must upload one ZIP file named `<andrewID>.zip`. This should contain:

1. A folder `code` that contains all code files, including but not limited to, the ones in the homework packet (**name the folder `bonus` if you are an undergraduate who has solved this problem for a bonus**). If there are subfolders, your code should handle relative paths. We will only compile using `mex` and execute `runtest.m` in the parent `code` directory.
2. Your writeup in `<andrewID>.pdf`. This should contain a summary of your approach for solving this homework, 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), and 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>`. We should be able to run the `mex` command to compile your code from within MATLAB under Linux!
   - 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.

2.8 Grading:

The grade will depend on two things:

1. How well-founded the approach is. In other words, can it guarantee completeness (to catch a target), 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.

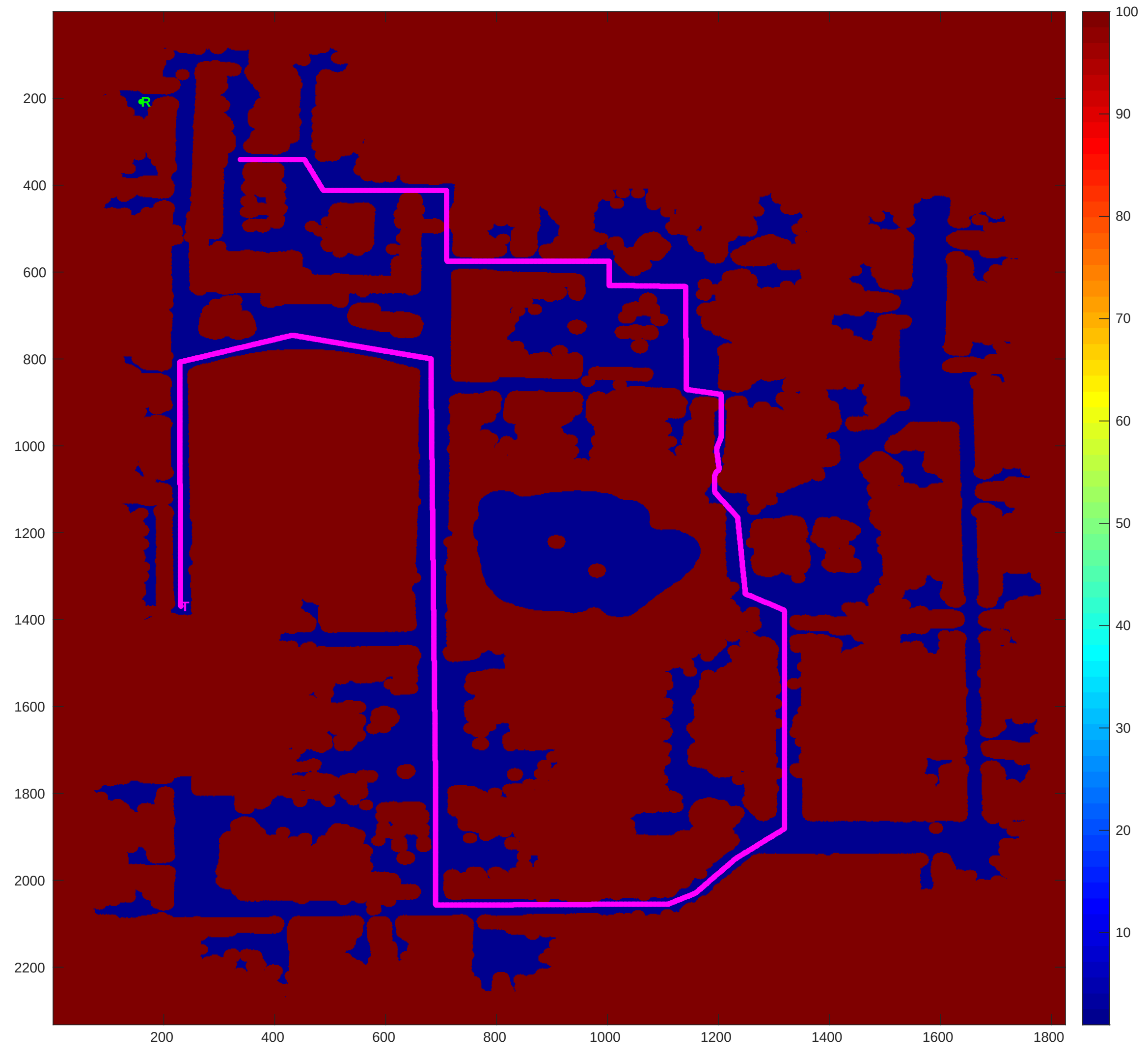

Image of information in `map1.txt`. The green R marks the starting position of the robot, the magenta T marks the starting position of the target, the magenta line is the target's trajectory. Blue cells have cost 1, red cells have cost 100, collision threshold is 100.

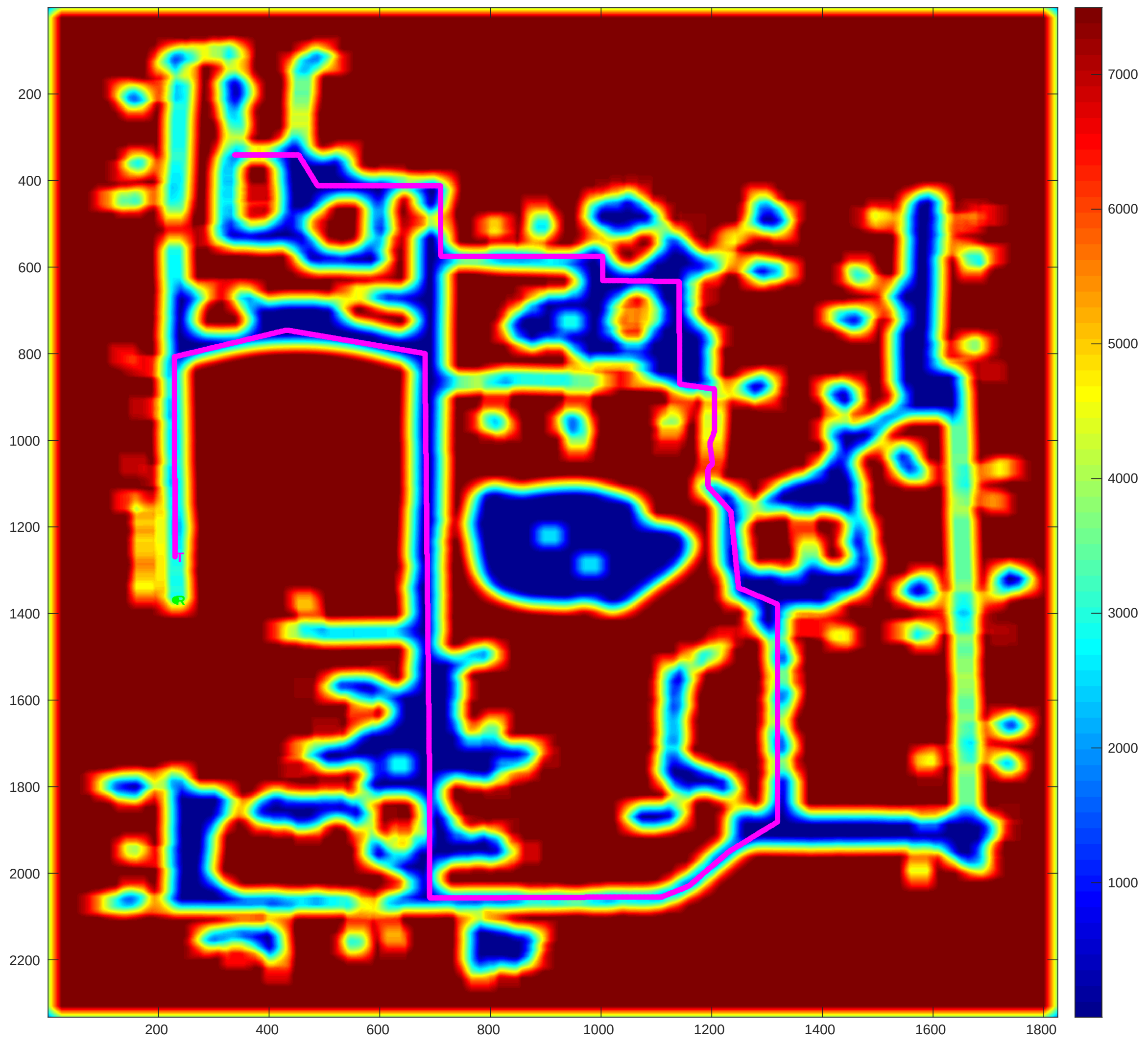

Image of information in `map2.txt`. The green R marks the starting position of the robot (directly below the starting position of the target), the magenta T marks the starting position of the target, the magenta line is the target's trajectory. Cells have cost between 1 and 7497, inclusive. Collision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

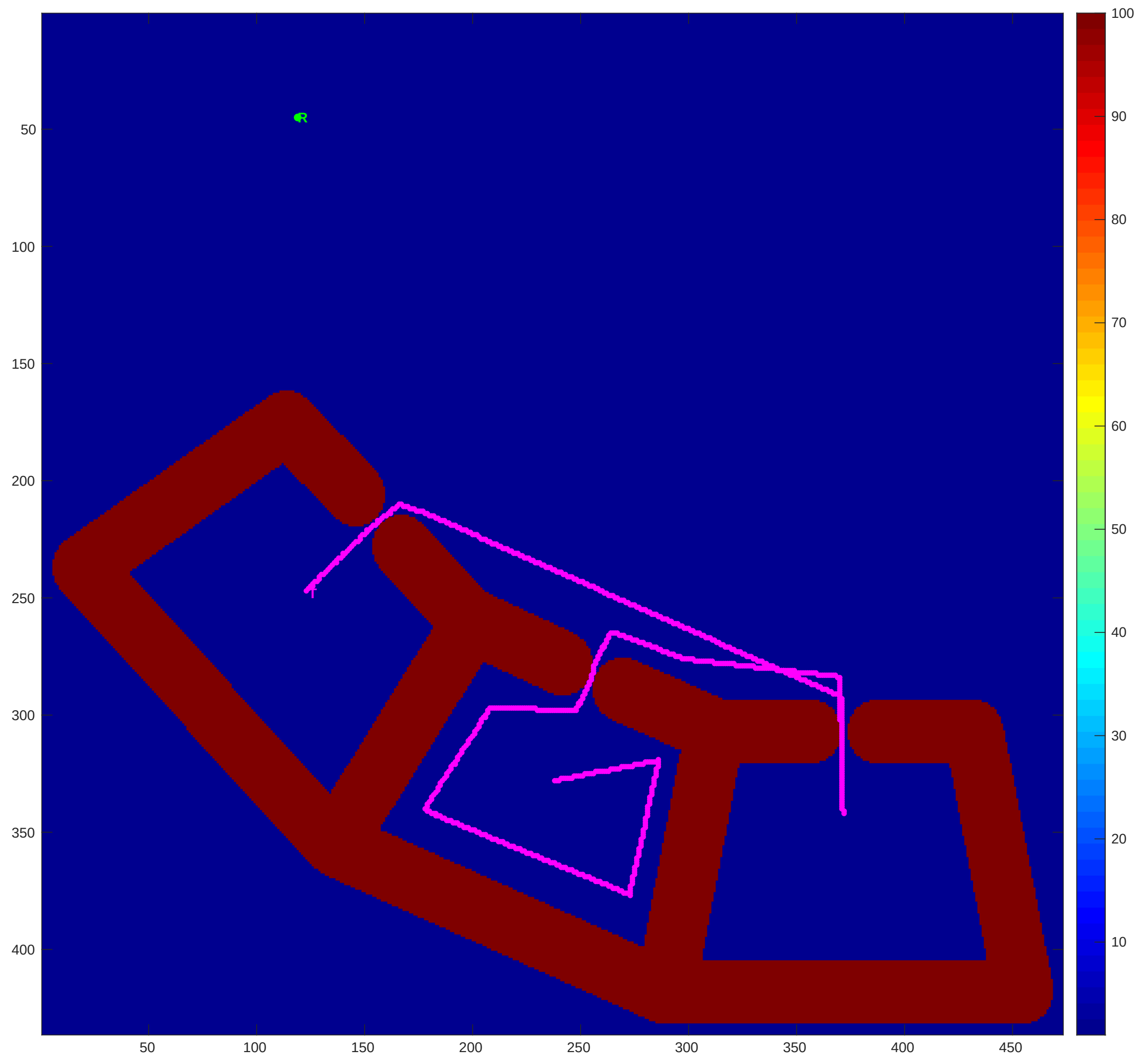

Image of information in `map3.txt`. The green R marks the starting position of the robot, the magenta T marks the starting position of the target, the magenta line is the target's trajectory. Blue cells have cost 1, red cells have cost 100, collision threshold is 100.

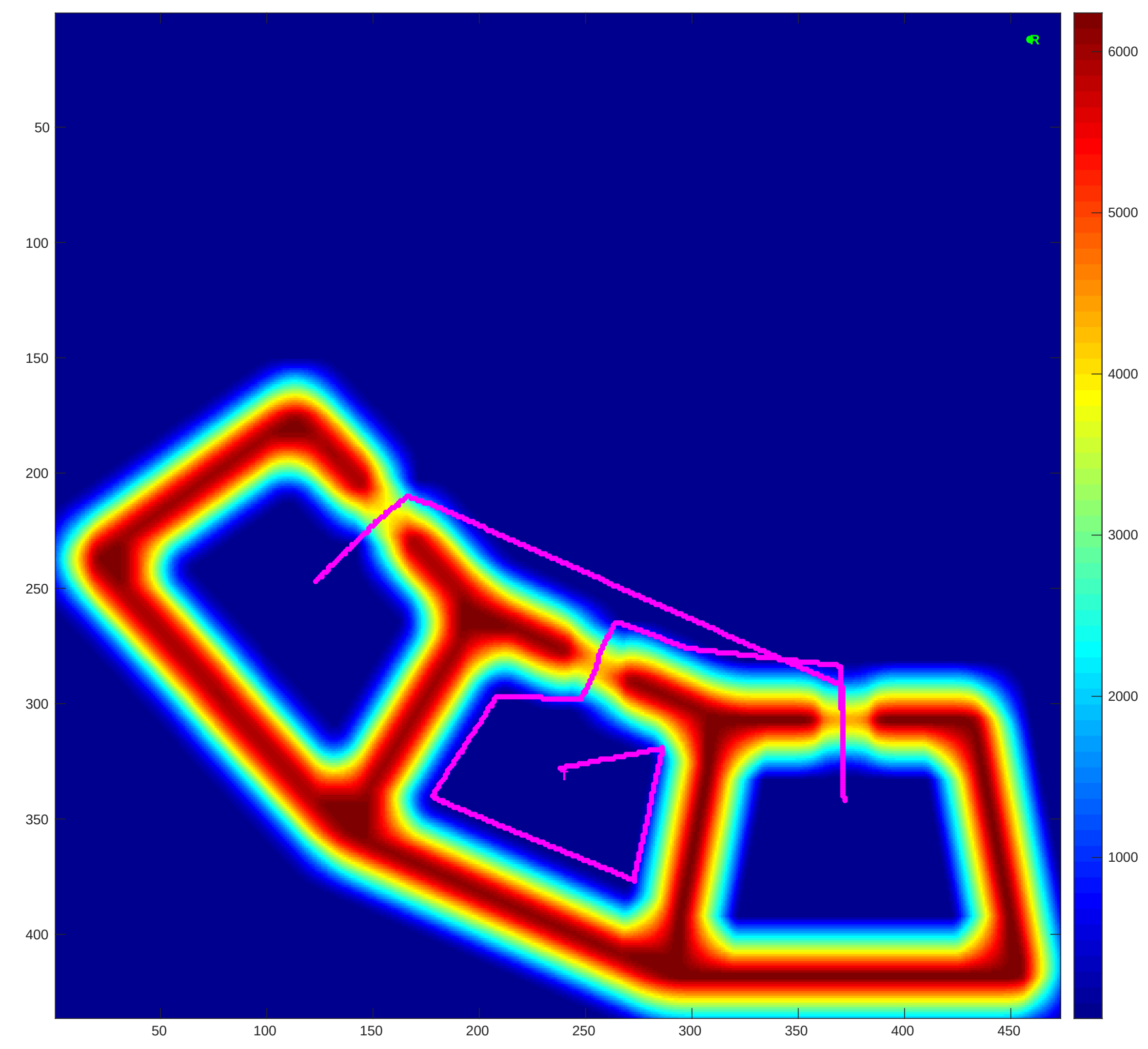

Image of information in `map4.txt`. The green R marks the starting position of the robot, the magenta T marks the starting position of the target, the magenta line is the target's trajectory. Cells have cost between 1 and 6240, inclusive. Collision threshold is 5000.