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

Professor: Maxim Likhachev *Spring 2026 TA: Barath Satheeshkumar*

1 Undergraduate and graduate Assignment

1.1 Task:

Write a planner for the robot to catch a target in a 2D grid world while minimizing the cost incurred by the robot (NOT the time it takes). The gridworld is 8-connected (that is, the robot can only move by at most one cell along X or Y axis or diagonally). 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 positive 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. **For graduate students, the biggest grid in this assignment is around 2000x2000 cells, and for undergraduates is around 200x200 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.

1.2 Code:

Your code is within the folder `code`, where you will have c++ and python files, as well as two maps directories, `undergrad` and `grad`. The planner function must output a single robot move. The planner should reside in the `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(
    int* map,
    int collision_thresh,
    int x_size,
    int y_size,
    int robotposeX,
    int robotposeY,
    int target_steps,
    int* target_traj,
    int targetposeX,
    int targetposeY,
    int curr_time,
    int* action_ptr
```

)

1.3 Inputs:

Each cell in the map of size `(x_size, y_size)` is associated with the cost of moving through it (positive integer). Note that if the robot stays in the same cell c for T time steps, then it will incur a cost of $cost(c) \cdot T$.

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 traversable. 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 (`curr_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`.

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:

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.

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

1.4 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).

`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.

1.5 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.

1.6 Execution:

Within the `code` directory there are two subdirectories, `undergrad` and `grad`, each containing a maps to test your planner. To compile the cpp code, open a terminal in the `code` directory and run:

```
>> g++ runtest.cpp planner.cpp
```

To run the planner:

```
>> ./a.out <student_type>/map<map_number>.txt
```

where `<student_type>` is `undergrad` or `grad`. For example, to run the planner on `map1.txt` for an undergraduate student, run:

```
>> ./a.out undergrad/map1.txt
```

To visualize the robot and target's trajectory:

```
>> python visualizer.py map<map_number>.txt
```

In the example above, to visualize the robot and target's trajectory for `map1.txt`, run:

```
>> python visualizer.py map1.txt
```

Currently, the planner greedily moves towards the last position on the moving object's trajectory.

1.7 Submission:

You will submit this assignment through Gradescope. 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 you add subfolders, your code should handle relative paths.

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.

    - 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, if one exists), can it provide sub-optimality or optimality guarantees on the cost of 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 maps than the ones provided in the directory. We can only promise that these maps will be of similar size as the ones provided.

Note (bonus): For undergraduates, a bonus of up to 10 extra credit points will be given if your planner can catch the target on the grad-size maps.

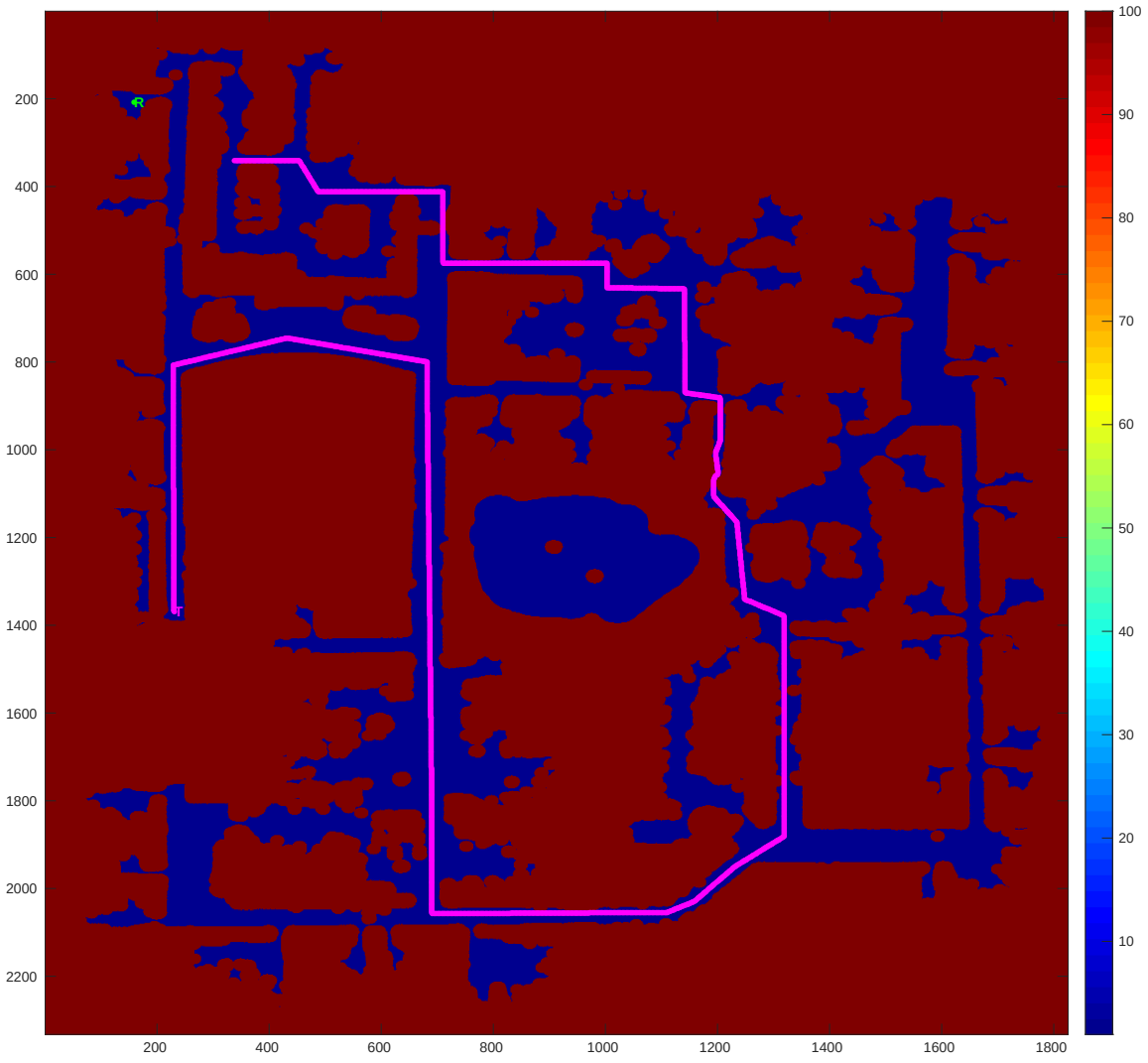

Image of information in `map1.txt`. The green R marks the starting position of the robot, the magenta T marks the starting position of the target, the magenta line is the target's trajectory. Blue cells have cost 1, red cells have cost 100, collision threshold is 100.

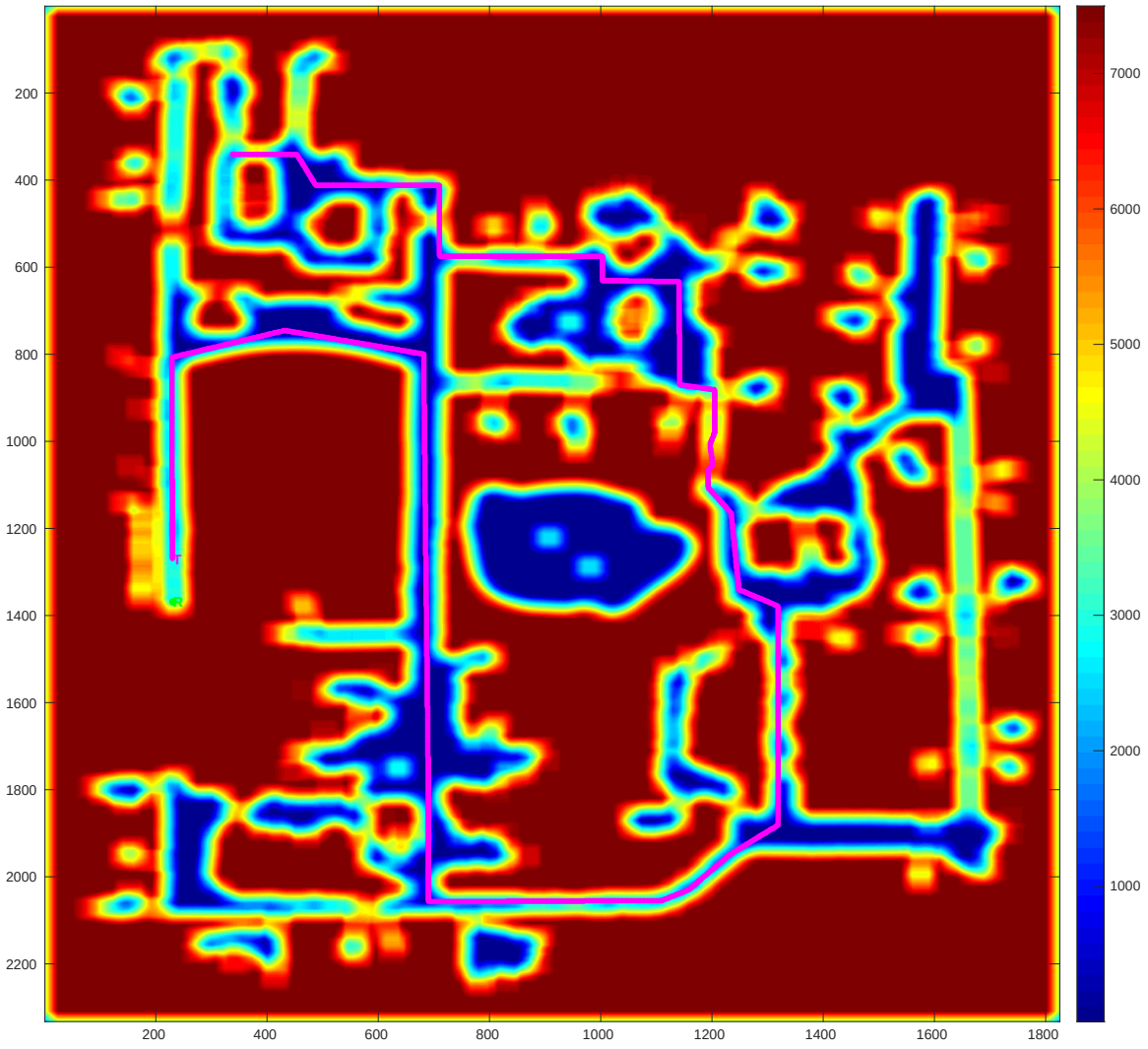

Image of information in `map2.txt`. The green R marks the starting position of the robot (directly below the starting position of the target), the magenta T marks the starting position of the target, the magenta line is the target's trajectory. Cells have cost between 1 and 7497, inclusive. Collision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

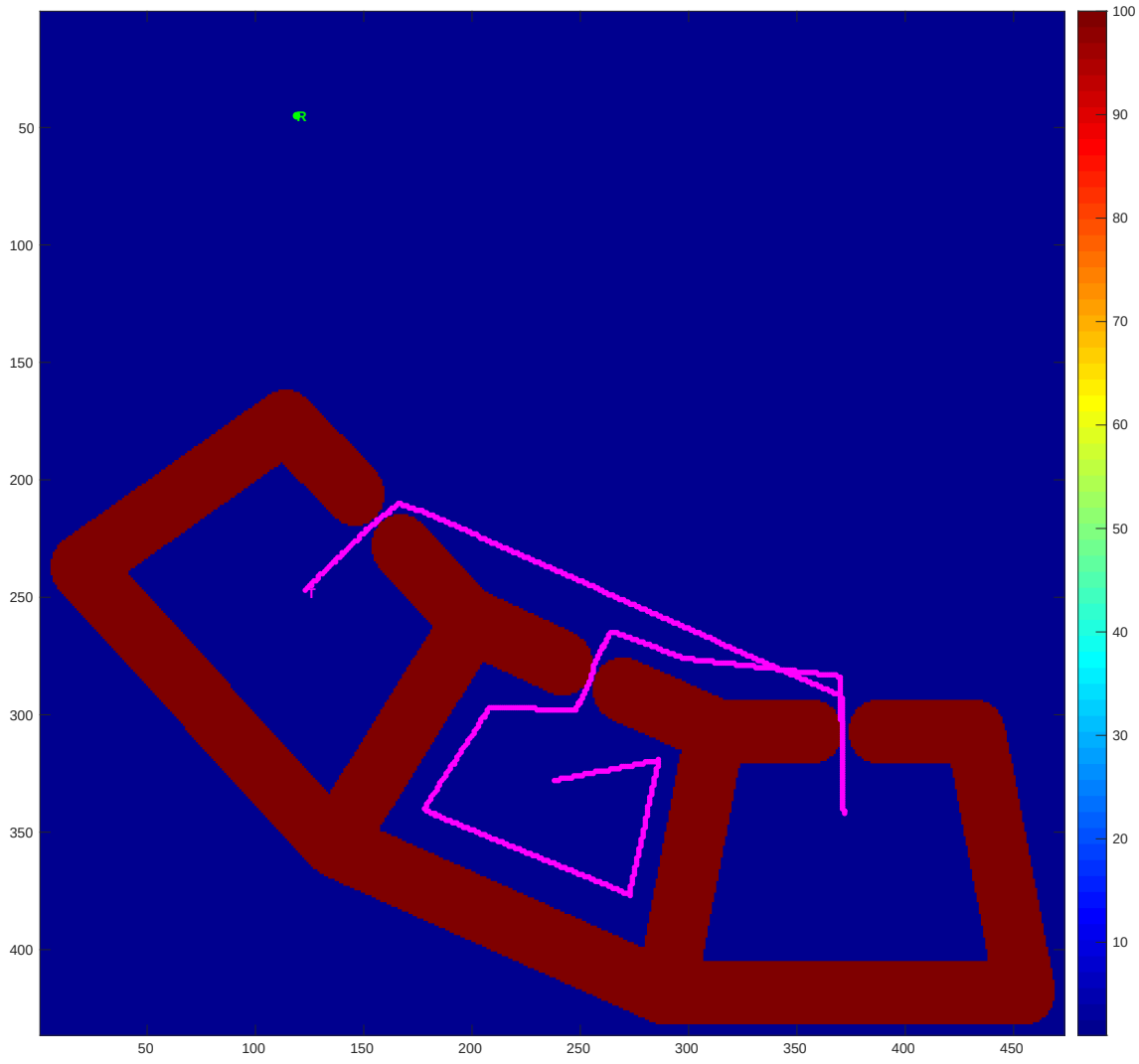

Image of information in `map3.txt`. The green R marks the starting position of the robot, the magenta T marks the starting position of the target, the magenta line is the target's trajectory. Blue cells have cost 1, red cells have cost 100, collision threshold is 100.

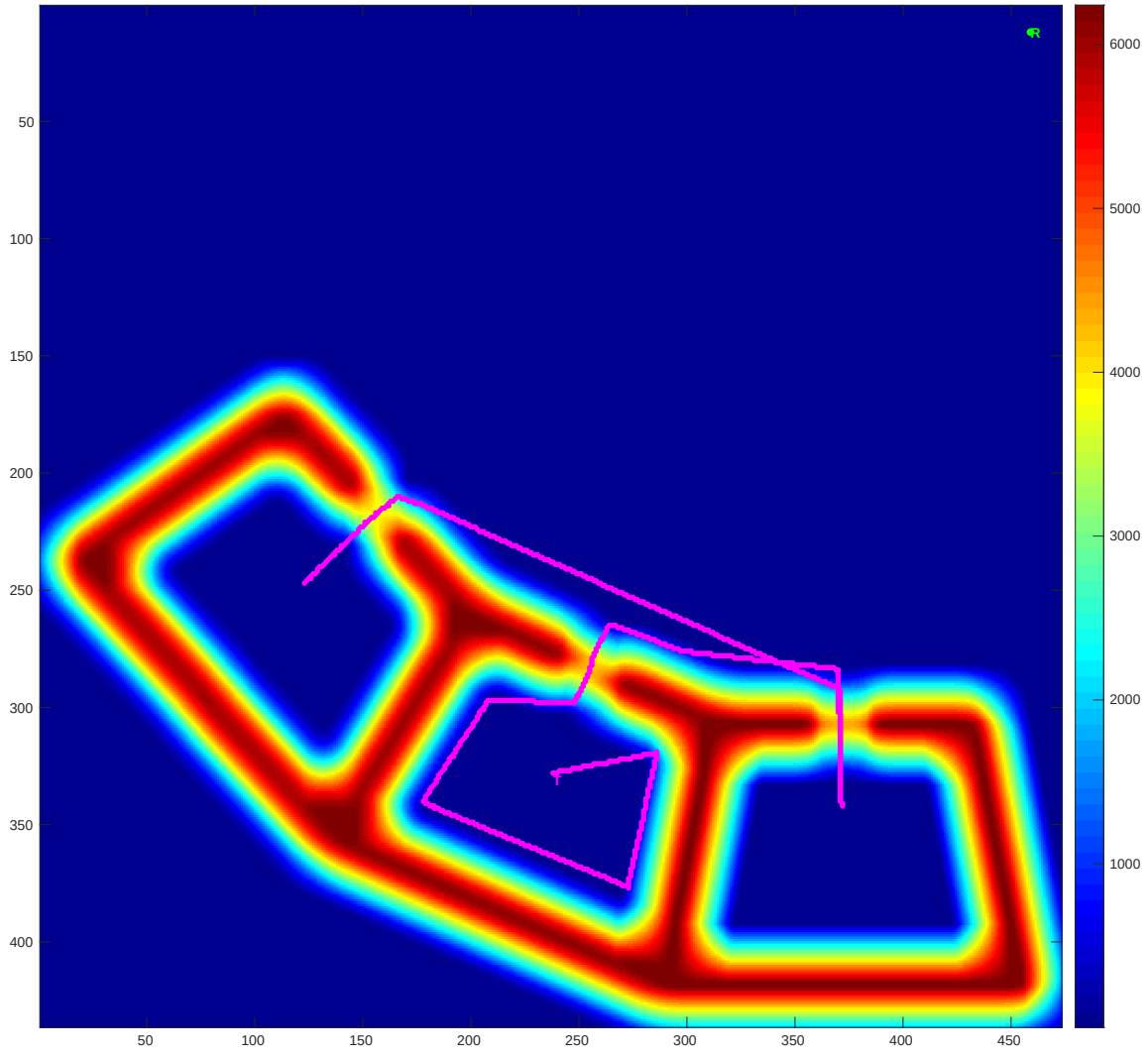

Image of information in `map4.txt`. The green R marks the starting position of the robot, the magenta T marks the starting position of the target, the magenta line is the target's trajectory. Cells have cost between 1 and 6240, inclusive. Collision threshold is 5000.

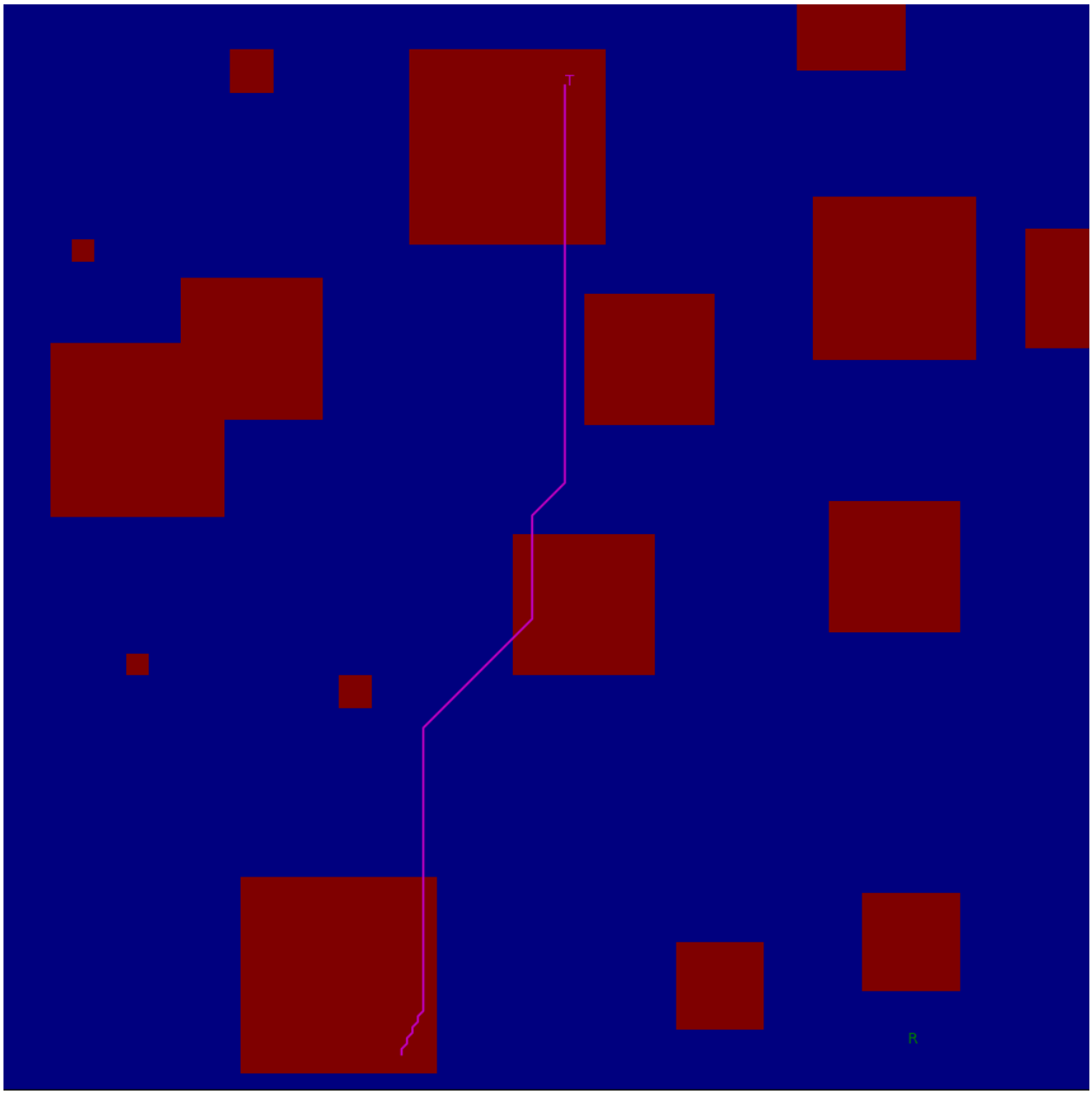

Image of information in `map16.txt`. The green R marks the starting position of the robot, the magenta T marks the starting position of the target, the magenta line is the target's trajectory. Cells have cost between 1 and 50, inclusive. Collision threshold is 50.