16-782: Planning and Decision Making in Robotics

Problem-Set 1: Catching a Moving Target

Paramjit Singh Baweja
Andrew id: paramjib

Planner Summary

When the planner is first called, I use an Astar search implementation to explore the environment for multiple goals. Once paths are built for these multiple goals, the best goal and it's corresponding path are determined based on the possibility of the robot catching the target. In subsequent calls to the planner, simply the next state from this ideal path is returned as the robot's action.

In the first iteration, while exploring, I make plans for multiple goals. The following is my methodology:

- The first estimate I make is the least distance the robot will need to reach the target. This is the Euclidean distance between the target and the robot. Let's call this value *d.*
    - Since the robot travels one unit distance in a second, I only consider the target's trajectory starting at *d* seconds. Because, this is the least amount of time the robot will take to reach the target if it goes through all obstacles. Let's call the index of the trajectory at this point *g.*
- Then, to reduce the number of plans to be made, rather than make plans for all the remaining points in the trajectory, I perform a binary search of goals, that are sorted by the time taken for the robot to reach there. Let's call the index of final position of the trajectory *f.*
    - The first goal of the trajectory is then at the index *M=(f+g)/2.* A star search exploration is done for this goal, and the time for this trajectory is calculated.

    - If this time is greater than the time remaining in the target's trajectory, the start value of *M* is the new start value, else, *M* is the new end value. This process then repeats.
    - The goal, whose path time matches the time taken for the target to reach that same point is the one that the robot then goes to.
- The last modification is to ensure the robot catches the target: If the time taken to reach the final position of the target is the same as the time left in the target's trajectory, then the planner makes another plan, with the heuristic's bias as 100.0 so that the robot reaches the target, even with large costs.

Other implementation details:

My algorithm uses a priority queue for the open set to allow efficient node processing, an un-ordered set for the closed set to avoid re-evaluation of nodes, and a euclidean distance based heuristic. I use a custom node structure to represent every state on the map, and an unordered map to store all the nodes indexed by their grid position. Once nodes are no longer needed, they are deleted to avoid memory leaks.

Installation

My entire planner is in the file *planner.cpp*
Create a directory named *build* and run the following commands

```
cd build
cmake ..
make && ./run_test map<number>.txt
python ../scripts/visualizer.py ../maps/map<number>.txt
```

Results

Map 1

RESULT

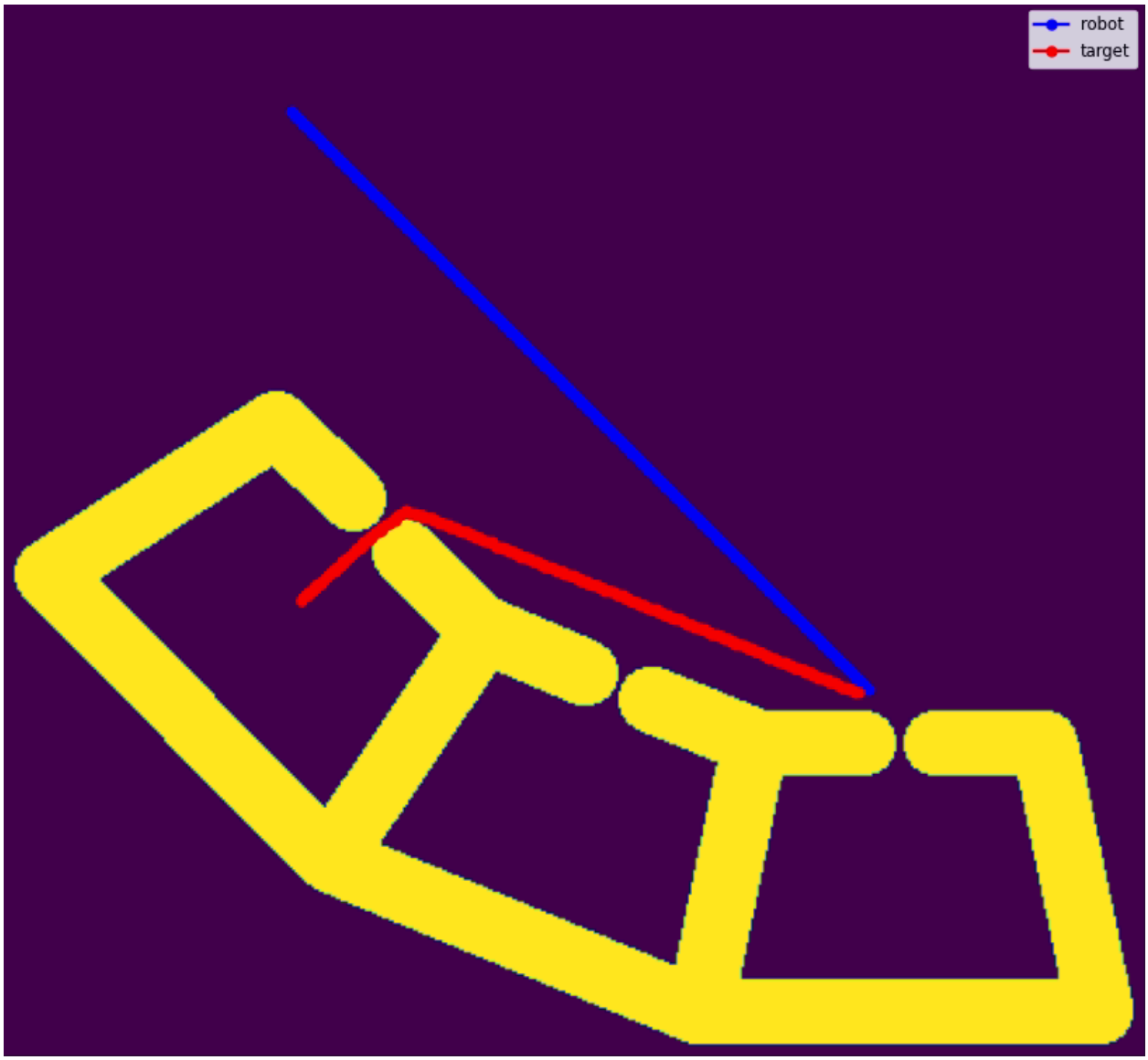
target caught = 1
time taken (s) = 2648
moves made = 2631
path cost = 2648

Map 2

RESULT

target caught = 1
time taken (s) = 3187
moves made = 2559
path cost = 6170398

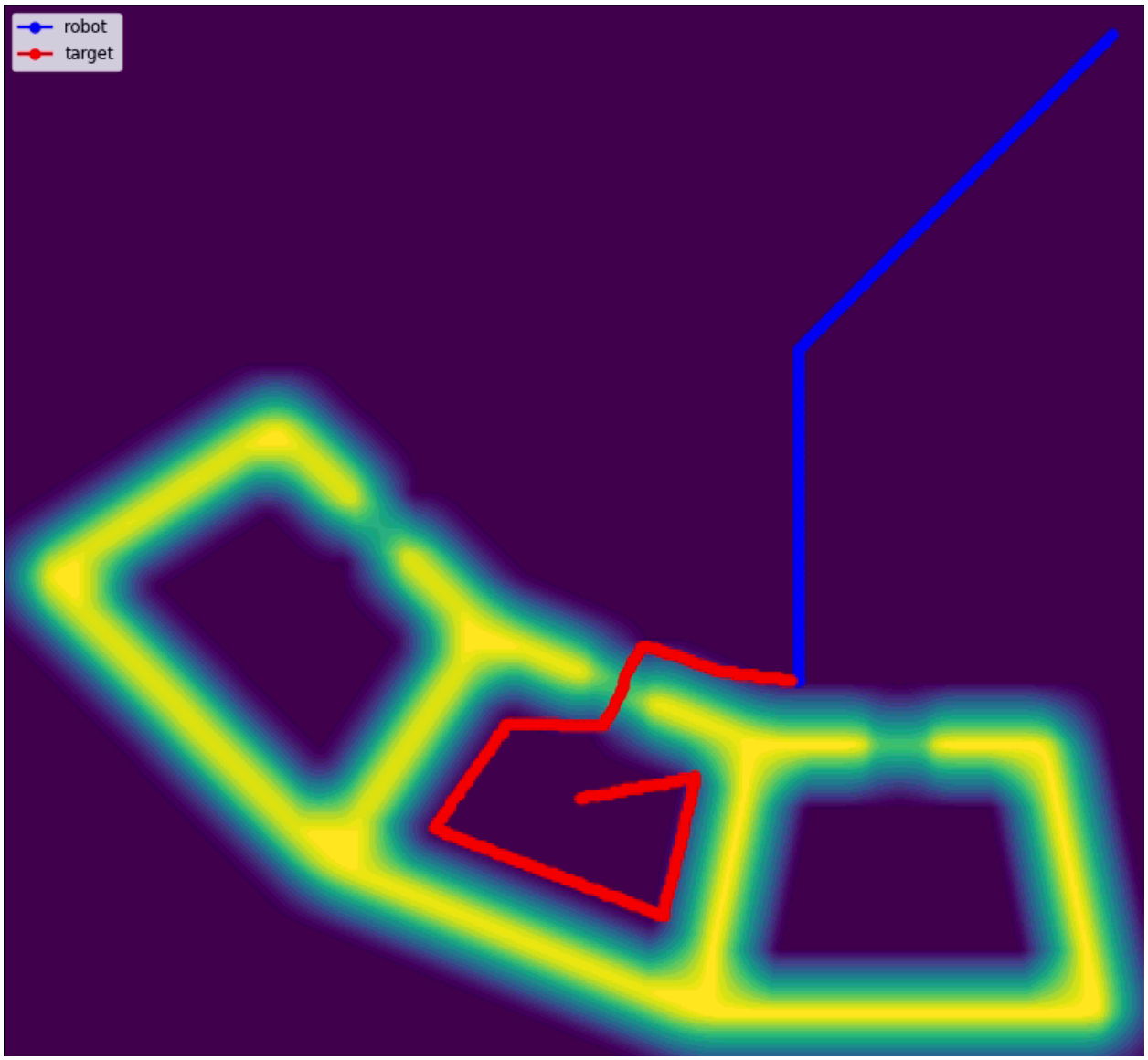
Map 3



RESULT

target caught = 1
time taken (s) = 246
moves made = 243
path cost = 246

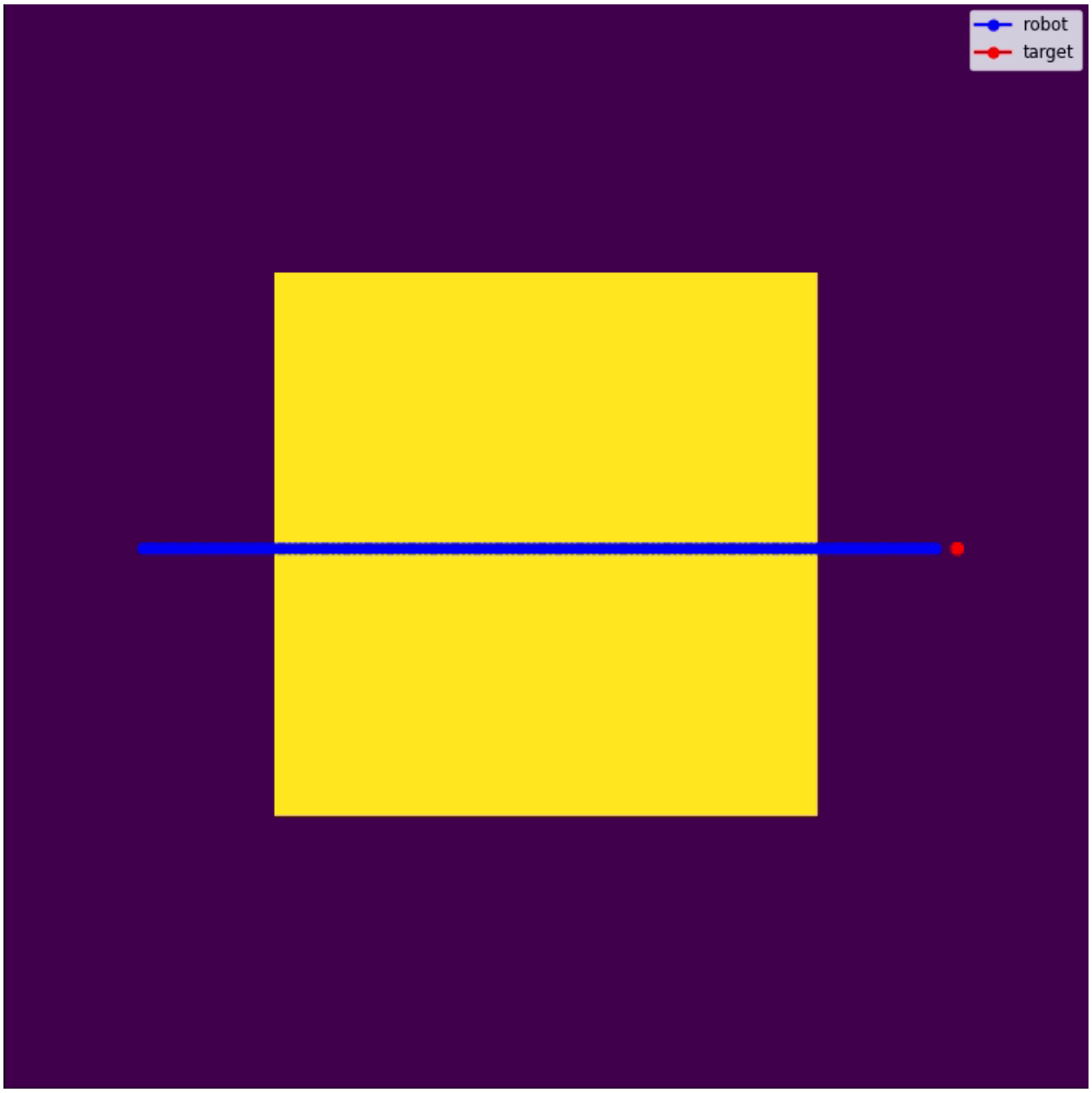
Map 4



RESULT

target caught = 1
time taken (s) = 387
moves made = 267
path cost = 387

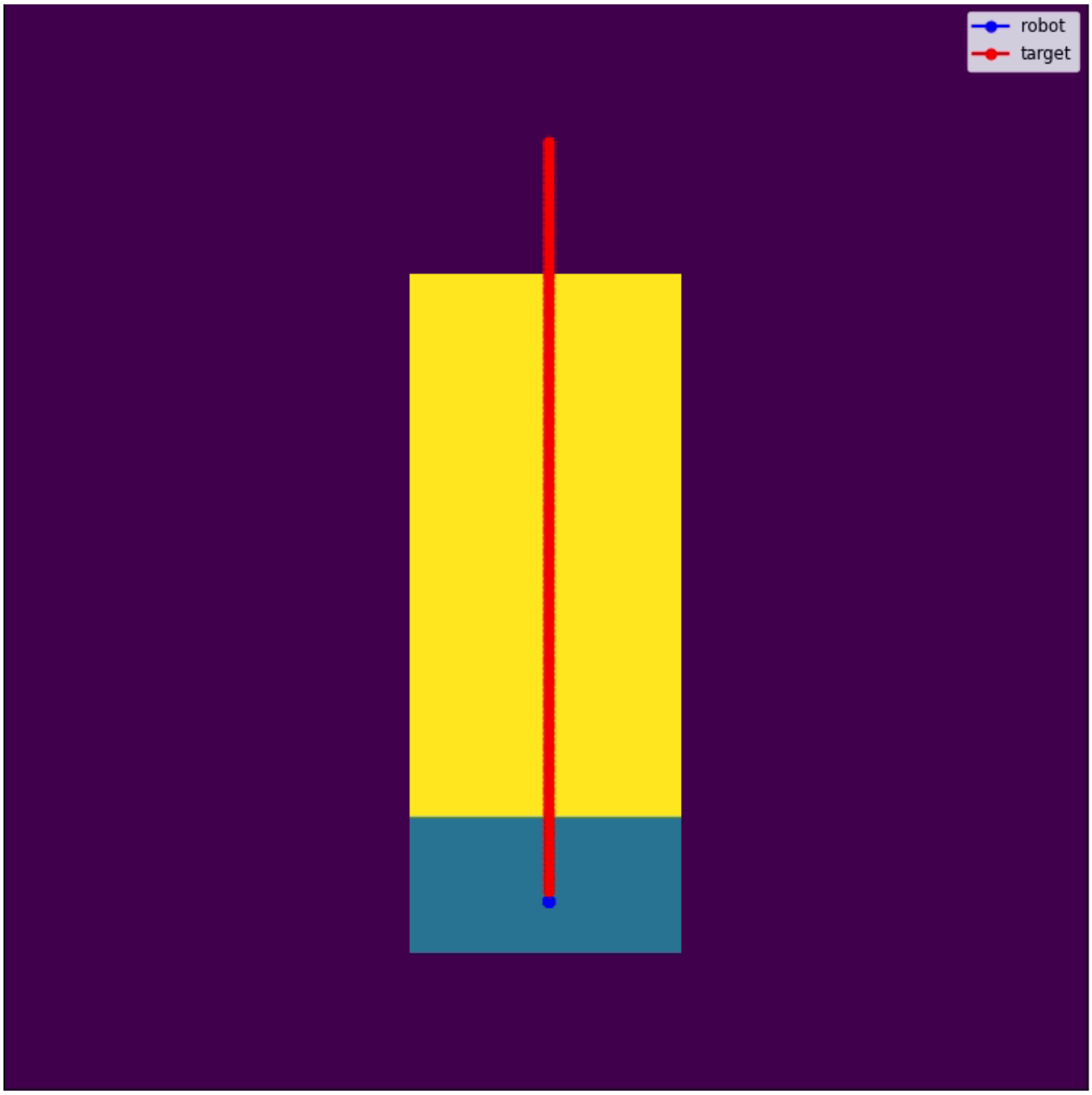
Map 5



RESULT

target caught = 1
time taken (s) = 150
moves made = 150
path cost = 5050

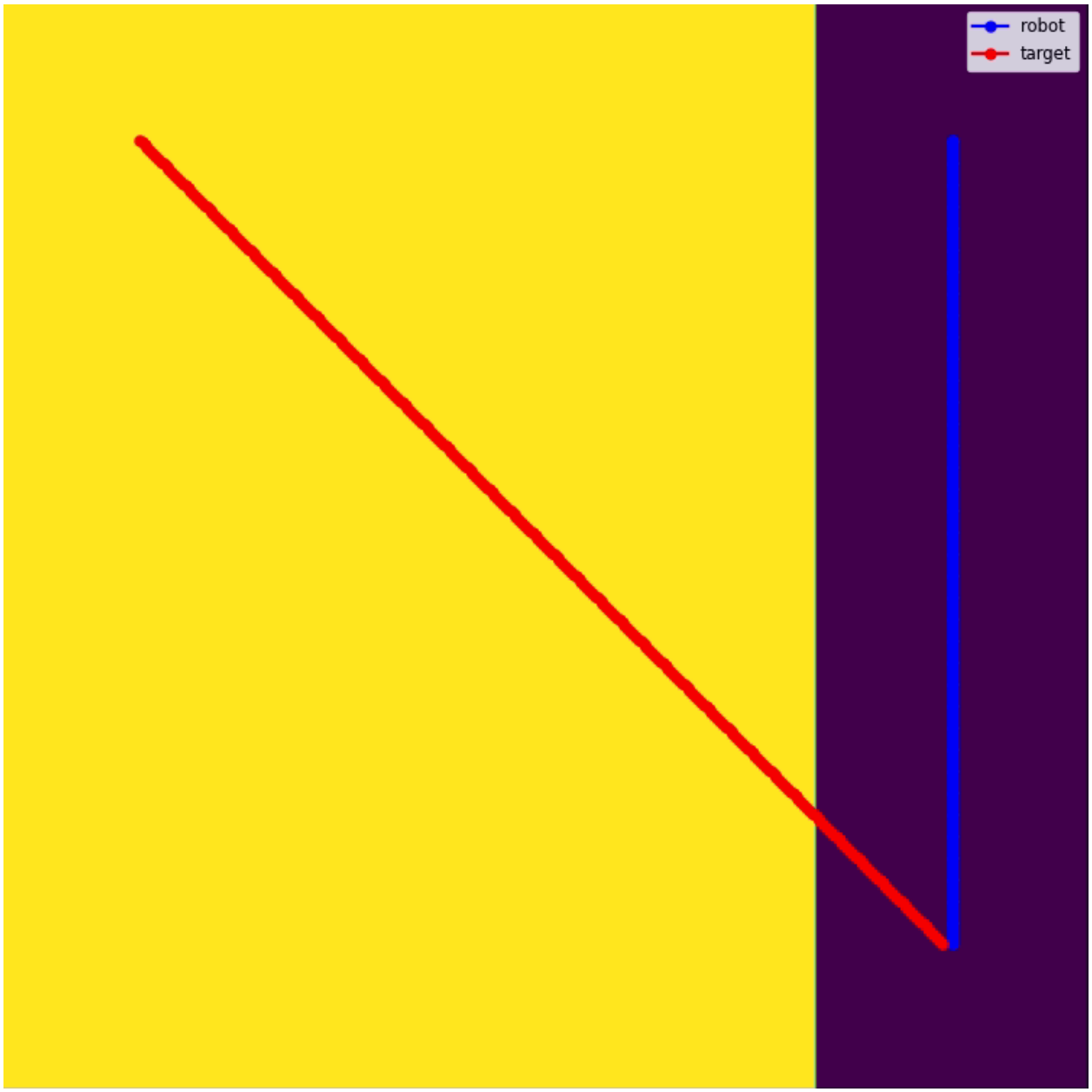
Map 6



RESULT

target caught = 1
time taken (s) = 140
moves made = 0
path cost = 2800

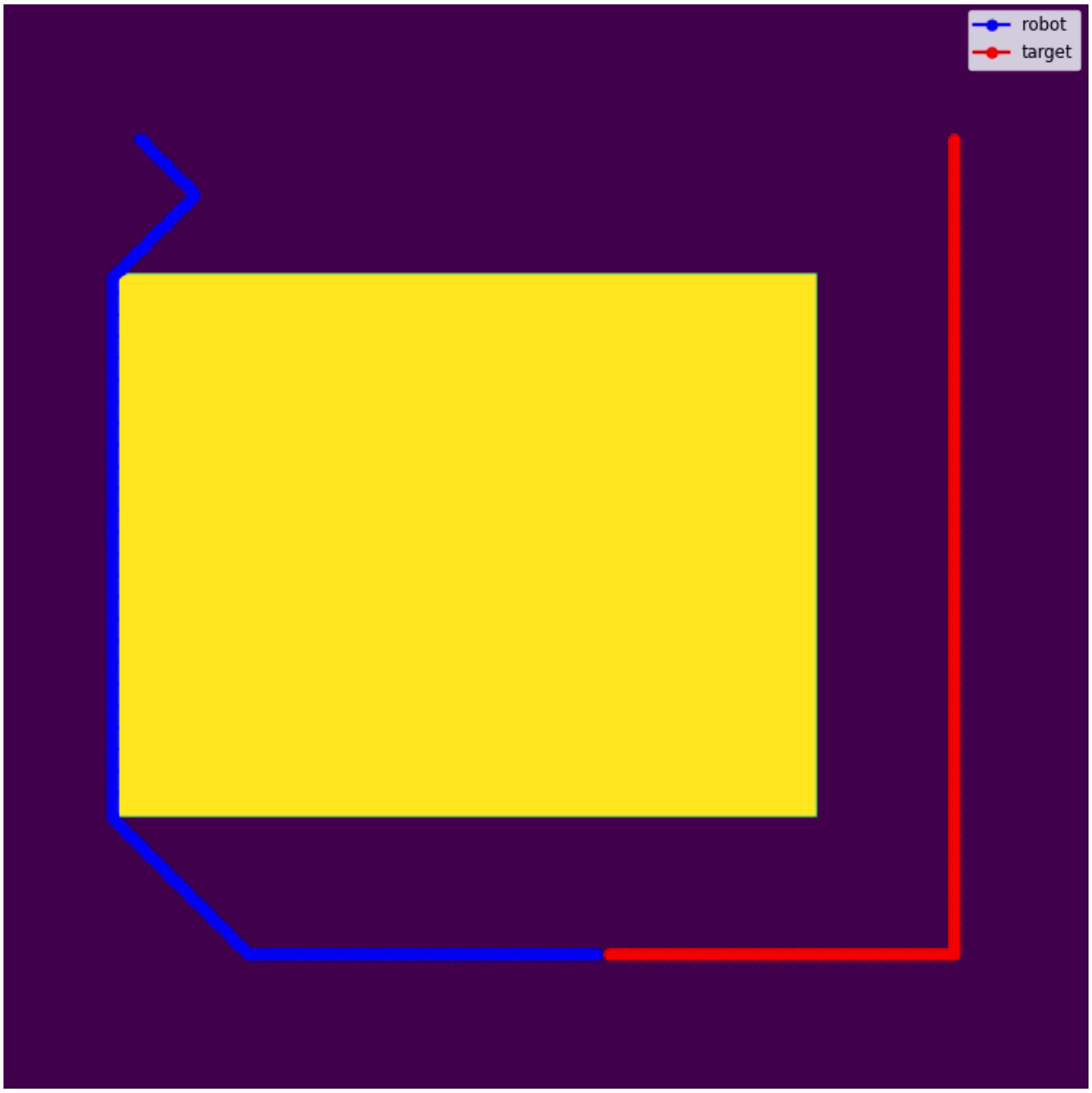
Map 7



RESULT

target caught = 1
time taken (s) = 300
moves made = 300
path cost = 300

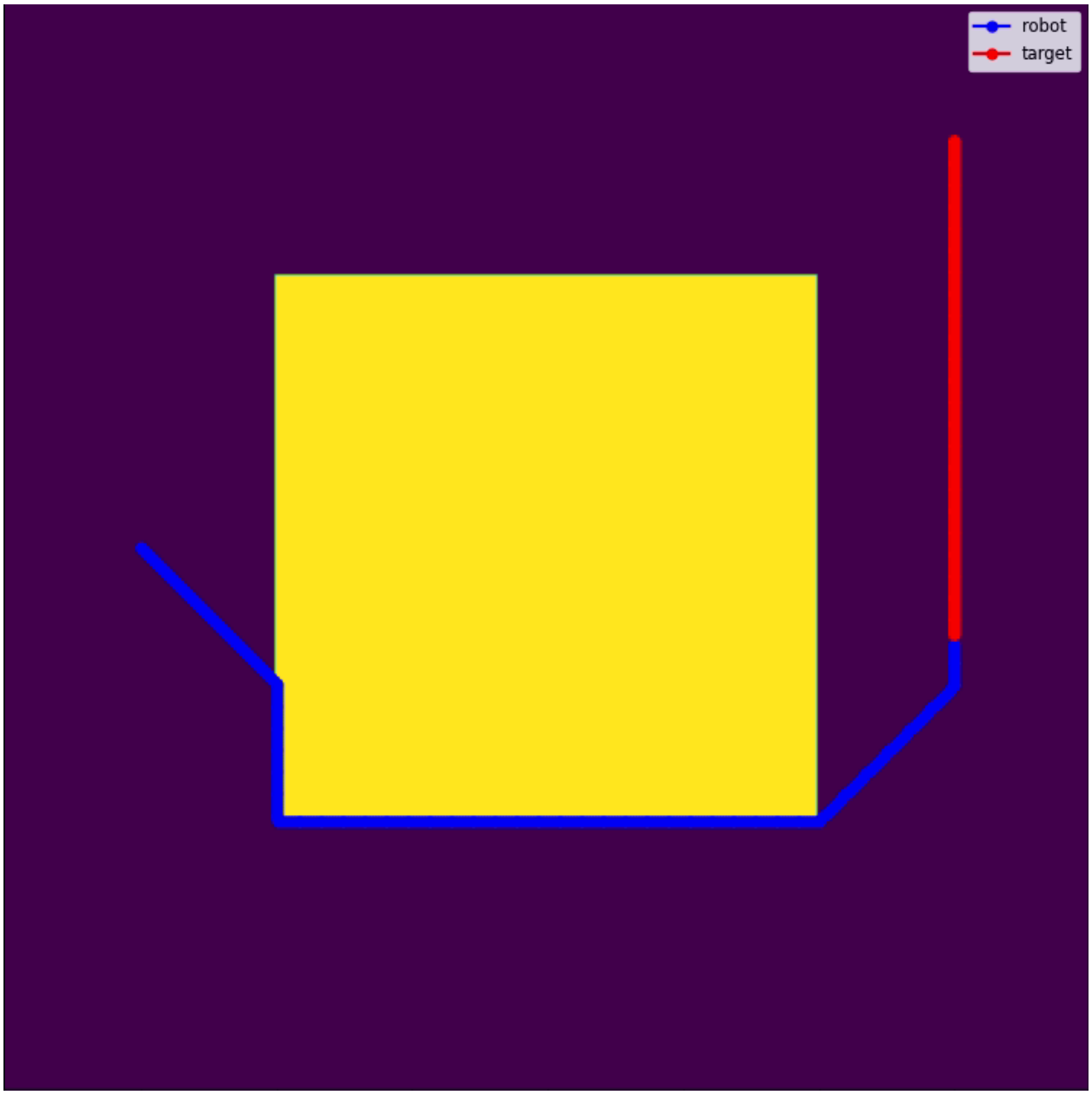
Map 8



RESULT

target caught = 1
time taken (s) = 431
moves made = 430
path cost = 431

Map 9



RESULT

target caught = 1
time taken (s) = 368
moves made = 367
path cost = 368