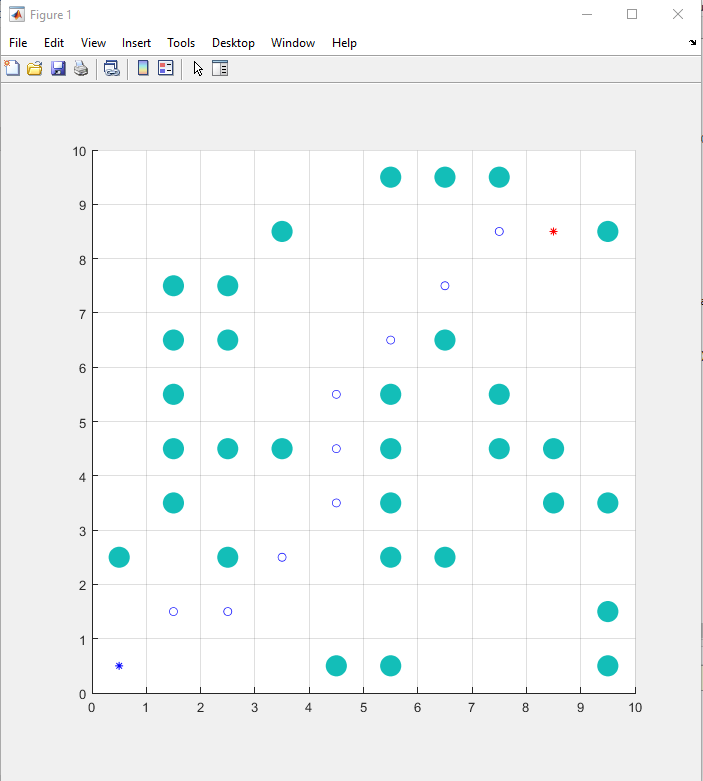
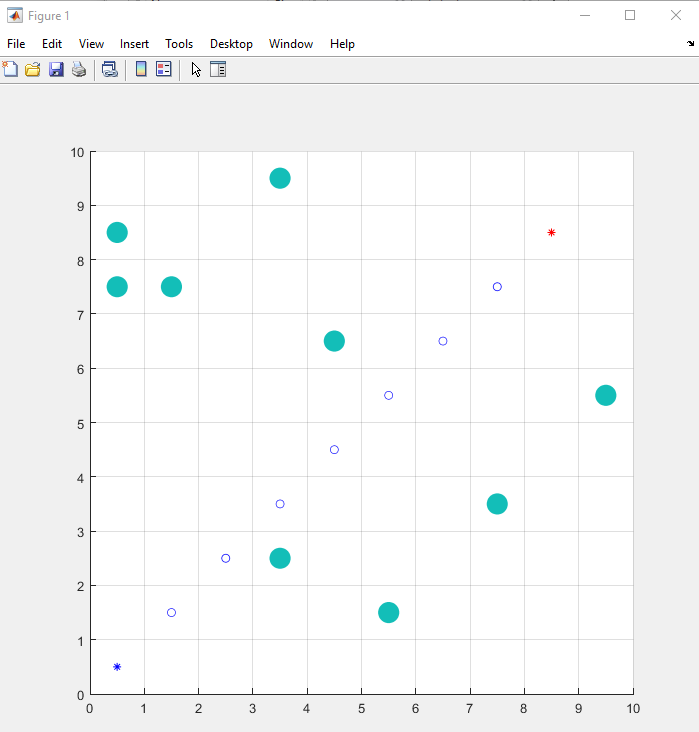
Chapter 2.1 Report for A\* planning algorithms by MATLAB

User ID: rabbit5024

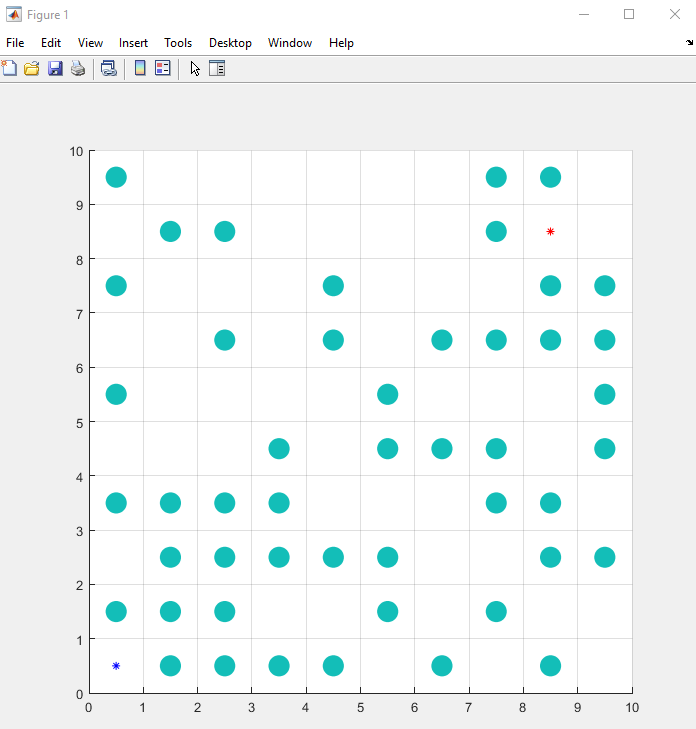
1. The screenshot of planning results for 3 random generated maps.
   1. MAP1 (obstacle\_ratio = 0.25)



* 1. MAP2(obstacle\_ratio = 0.1)



* 1. MAP3(obstacle\_ratio = 0.5)



1. Results analysis

At normal obstacle situation, the A\* algorithm could find the optimal path with minimal distance cost. If there are too many obstacles and no path exist, the algorithm will return an empty path.

1. Others that are interests

The ‘min\_fn’ function requires the **xTarget** and **yTarget**, but I think it is unnecessary for the sort function to know the target point. I commented these variables in the subfunction, and the program still works well.

A screenshot of a social media post

Description automatically generated

**A\* method steps:**

1. Add the starting square (or node) to the open list.

2. Repeat the following:

A) Look for the lowest F cost square on the open list. We refer to this as the current square.

B) Switch it to the closed list.

C) For each of the 8 squares adjacent to this current square …

* If it is not walkable or if it is on the closed list, ignore it. Otherwise do the following.
* If it isn’t on the open list, add it to the open list. Make the current square the parent of this square. Record the F, G, and H costs of the square.
* If it is on the open list already, check to see if this path to that square is better, using G cost as the measure. A lower G cost means that this is a better path. If so, change the parent of the square to the current square, and recalculate the G and F scores of the square. If you are keeping your open list sorted by F score, you may need to resort the list to account for the change.

D) Stop when you:

* Add the target square to the closed list, in which case the path has been found, or
* Fail to find the target square, and the open list is empty. In this case, there is no path.

1. Save the path. Working backwards from the target square, go from each square to its parent square until you reach the starting square. That is your path.

Version 2:

// A\* Search Algorithm

1. Initialize the open list

2. Initialize the closed list

put the starting node on the open

list (you can leave its **f** at zero)

3. while the open list is not empty

a) find the node with the least **f** on

the open list, call it "q"

b) pop q off the open list

c) generate q's 8 successors and set their parents to q

d) for each successor

i) if successor is the goal, stop search

successor.**g** = q.**g** + distance between

successor and q

successor.**h** = distance from goal to

successor (This can be done using many

ways, we will discuss three heuristics-

Manhattan, Diagonal and Euclidean

Heuristics)

successor.**f** = successor.**g** + successor.**h**

ii) if a node with the same position as

successor is in the OPEN list which has a

lower **f** than successor, skip this successor

iii) if a node with the same position as

successor is in the CLOSED list which has

a lower **f** than successor, skip this successor

otherwise, add the node to the open list

end (for loop)

e) push q on the closed list

end (while loop)