CS365 AI: NAVIGATION ROBOT

Ashish Gupta, Y8140

Abstract:

The project aims to build a navigational robot to identify static as well as dynamic objects moving with a small probability. The project has been successfully able to handle the following cases:

1) **Static Objects:**

* Objects touching each other
* Objects along the boundary as well as corners
* Objects having overlapping rectangles (having minimum separation of 1 unit)

2) **Dynamic Objects**:

* Dynamic Objects have been considered with almost 100% accuracy in the cases where objects do not touch each other in the dynamic environment. (Details in the Dynamic Object Section)

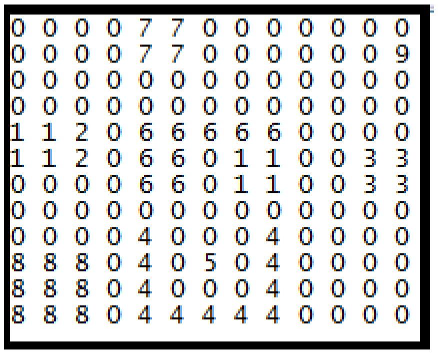
Technique Used: Random Restart with Reset

Model of the Robot:

The robot has been modeled with its head pointing a particular direction i.e. North, West, South, and East. The robot always tries to keep the objects either to its left or to its right, thereby making the traversal of the adjacent objects possible.

2) Algorithm for Static Objects:

The robot follows a snake like traversal to follow the complete grid covering one row at a time. Initially, the boundary is traversed covering any objects in between to find the length and width of the boundary. The boundary objects needs to handle uniquely. So, the robot makes a second trip across the boundary to map the boundary objects in its memory map. The following grid is successfully navigated:



```
0 0 0 0 7 7 0 0 0 0 0 0 0
0 0 0 0 7 7 0 0 0 0 0 0 9
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 2 0 6 6 6 6 6 0 0 0 0
1 1 2 0 6 6 0 1 1 0 0 3 3
0 0 0 0 6 6 0 1 1 0 0 3 3
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 4 0 0 0 4 0 0 0 0
8 8 8 0 4 0 5 0 4 0 0 0 0
8 8 8 0 4 0 0 0 4 0 0 0 0
8 8 8 0 4 4 4 4 4 0 0 0 0
```

Fig1: Boundary Objects as well as overlapping objects
have been handled.

During the snake like traversal, the robot continues to traverse the row till it encounters an object. Three cases may arise depending on the type of the object:

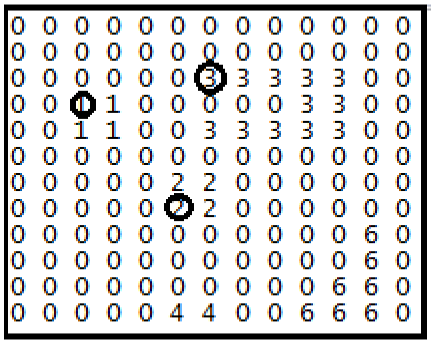
a) **Boundary Objects:** Boundary Objects have already been mapped in the memory. So there is no need to do a getID or sense() for traversing and reaching to other part of the same row. The map is used to traverse such an object. Two types of boundary object are possible: one which touches a single boundary and other which touches two or more boundaries. All cases have been handled.

b) **New Object:** Let say the robot is at position P when it encounters a new object. The robot here tries to cover the object in clockwise direction i.e. always orienting itself such that the object is to the right of the robot. When the robot reached P, it has completely traversed the objects. It then maps this objects in its memory using temporary lists it maintained during traversing the object. The objects boundary is also stored separately.

c) **Old Objects:** Since the object is old, we already have a mapping in the memory map of the robot. Whenever we arrive at a cell, we check to see that currently if we are in front of an old object. If so, we traverse the object from below so it do not encounter any new object during the process using its stored boundary to reach the next part of the same row.

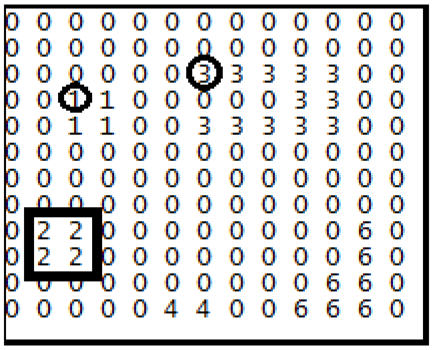
3) Algorithm for dynamic Objects:

The grid has been traversed only using move and getID. Sense() has not been used to avoid simulating the object unnecessarily. The Dynamic objects have been traversed using the initially generated grid as the reference i.e. the final answer is always with respect to initial grid which is also the final grid at the reporting time. Random restart technique along with reset has been used to maintain the initial grid as reference.



0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	3	3	3	3	3	0	0
0	0	1	1	0	0	0	0	0	3	3	0	0
0	0	1	1	0	0	3	3	3	3	3	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	2	2	0	0	0	0	0	0
0	0	0	0	0	2	2	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	6	0
0	0	0	0	0	0	0	0	0	0	0	6	0
0	0	0	0	0	0	0	0	0	0	6	6	0
0	0	0	0	0	4	4	0	0	6	6	6	0

Fig 2: Grid with the start block marked.



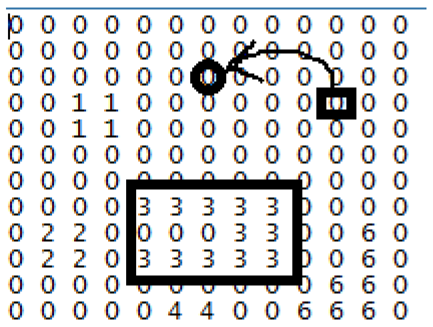
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	3	3	3	3	3	0	0
0	0	1	1	0	0	0	0	0	3	3	0	0
0	0	1	1	0	0	3	3	3	3	3	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	2	2	0	0	0	0	0	0	0	0	6	0
0	2	2	0	0	0	0	0	0	0	0	6	0
0	0	0	0	0	0	0	0	0	0	6	6	0
0	0	0	0	0	4	4	0	0	6	6	6	0

Fig3: Grid with the moved object marked.

Start Blocks of each object: During the first traversal of the object, whenever after interaction with the robot, some objects may move or may not. Irrespective of that, the mapping of the first block (encircled in the figure) which the robot encounters for a particular object is mapped with the objID. This block is then used to find out if the object is in its original position or it has been moved.

Assumption: Boundary Objects are supposed to be stationary. No partitions are considered. And the object if it moves after interaction is only allowed to transport and not move one step forward in a way that robot is not able to detect the movement.

Traversing a new Object: Let the say we encounter the new object at position P. Now similar to static case, we try to traverse the object in the clockwise direction getting id's in between to take care of touching objects. The object may move in between. Using assumption, the robot will not find that particular id which it expects since it has not reached its start position P. So, the robot will include this objID in the lost objects list and retrace itself to P and continue with the snake like traversal.



```
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 3 3 3 3 3 0 0 0 0
0 2 2 0 0 0 0 3 3 0 0 6 0
0 2 2 0 3 3 3 3 3 0 0 6 0
0 0 0 0 0 0 0 0 0 0 6 6 0
0 0 0 0 0 4 4 0 0 6 6 6 0
```

Fig 4: Bot traverses to its start position marked by arrow
on the movement of object with id = 3

This particular object if encountered anywhere again is simply ignored and traversed over. It would be handled during the next restart () of the grid. By this, we ensure that the objects that are mapped in the memory have correct location with the initial grid.

Random Restart (): After reaching the top, if any objects remains in the lost objects, the restart () is called which the places the bot at some random location and transforms the grid as it was during the start of the navigation.

Initial Align: The robot is initially aligned after random restart to reach the (1,1) block. During this process, some of the objects may move but we will be able to detect them using our mapping of unique start blocks recorded during the first traversal.

Traversing Objects after a restart: We would look into our memory to check if we are facing any already mapped object. If so, we will traverse the object using the boundary recorded, not simulating the object so it may not move. Any new object must be a lost object. If successfully traversed this time, it is removed from the lost objects list.

Observation: After every complete traversal of the complete grid, the number of objects recorded with their position similar to initial grid (also the grid after restart) keeps on decreasing. So after fixed amount of time, we will be able to terminate the algorithm.