A Comparison of Robot Navigation Algorithms for an Unknown Goal

Russell Bayuk
Steven Ratering (faculty mentor)
Computer Science Department
University of Wisconsin – Eau Claire
Eau Claire, WI 54702
{bayukrj, raterisj}@uwec.edu

Abstract

Some robots have stationary bases and movable arms and grippers, while others are mobile. Some robots' movements are precisely programmed while others are more autonomous. The robots we are interested in are both mobile and autonomous. A common task for these robots is to go from one known location to another location whose coordinates are given. The hard part is avoiding obstacles on the way. Another task is finding a goal location when its coordinates are unknown. The robot senses it is at the goal only after it arrives there. This is the problem we have investigated. We have implemented several different algorithms and analyzed them in terms of how long the robot takes to get to the goal and return to its starting location.

Introduction

A primary problem of robotics is getting a robot to move from one position to another without bumping into any obstacles. This can be seen as either a path planning problem or a navigation problem. The path planning problem[1,2,3] is solved by finding an obstacle-free path through space and time from start configuration to goal configuration. The obstacles may be stationary or moving. With sufficient a priori information this problem can be solved before the robot starts to move, in fact one need not have a robot to solve this problem. On the other hand, in the robot navigation problem, the robot gathers information about the environment while it is moving. A fair amount of work has been done with the robot navigation problem when the goal is known a priori.[4,5,6,7] Not much has been done when the goal is not known a priori. The problem we address is robot navigation in an environment with unknown stationary obstacles where the goal is also unknown. After the robot reaches the goal it returns to its home cell. The robot senses obstacles when it bumps into them and it senses the goal only after it arrives there. The purpose is to determine the most efficient algorithm for navigating these unknown environments.

The unknown environment is a rectangular room. This is to simulate the normal environment of a robot, which is generally indoors. The room consists of a number of cells. Between adjacent cells, there is the possibility of a wall. When a room is created a cell is picked at random, that cell is assigned to be the goal cell. Since the room is created with random wall placements, it is possible for parts of the room to be sealed off by walls. The use of cells and walls can approximate any shape in a two-dimensional environment. A closer approximation can be achieved by making the cells sufficiently small.

The robot starts in the lower left hand corner facing to the “east” or to the right. Its beginning coordinates are (0, 0). The only information that the robot knows about the environment are the boundaries. It knows that it is contained in a rectangular area. It knows how big the rectangle is and that it cannot leave the limits of the rectangle. It knows that there is a goal inside the rectangle and that it is possible to reach the goal. Even though it is possible for parts of the room to be inaccessible, the goal will not be in these parts. The robot will know where the goal is only after it has reached the goal. It cannot see the goal no matter how close it is until it is directly on top of it. Although the robot doesn’t start out with much information about the environment, that doesn’t stop it from learning as it goes along.

Learning is a critical step during the search of the room. Every cell the robot visits, it remembers. Every wall that the robot bumps into gets recorded in a log. When the robot passes from one cell to another without bumping into a wall, this is also recorded. The robot then uses this knowledge to narrow down the places where the goal may be. The path that the robot takes to uncover information about each cell is determined by the combination of algorithms it is using.

Algorithms

There are many different types of navigation algorithms that a robot may use to traverse space. Often these algorithms have similar processes but differ only slightly in different situations. For example, when a robot hits a dead end, where does it go next? Two different algorithms could have gotten the robot there using the exact same path, but differ by how they choose the next step. Another way two algorithms could differ is in how they choose the path back to the original start state after it has found the goal.

We found there to be three specific situations where there is a major difference between navigation algorithms when choosing a path. Because of these situations, we chose to split the algorithms into three different steps. The first step deals with the generic situation, how to begin and continue the traversal of unknown space. This step chooses which one of the neighboring cells to visit. The second situation is used when the first situation cannot pick an unvisited cell to visit. This occurs when either, all the adjacent unvisited cells are blocked by walls, or if all the adjacent cells have already been visited. At this point, a decision needs to be made by the robot. It needs to choose a path to continue its search of the unknown environment. The third step is when the robot has found the goal and needs to proceed back to the initial starting point. There are several different ways that the robot may choose to return to the origin.

There are several algorithms implemented for each of the steps. For the first step there are two different algorithms. One is called “Follow the Right Wall.” This consists of the robot following a wall similarly to a person walking along a path with their hand always touching the wall on that side. A wall can be substituted with a cell that has already been visited. The second algorithm implemented for the first step is called “Right Left Sweep.” This algorithm is similar to “Follow the Right Wall” except when the robot enters a cell that is adjacent to a boundary wall, the direction switches. For example, if the robot is currently following the right wall and enters a cell with a boundary wall the robot would start following the left wall. This would continue until it enters another cell that is bordered by a boundary wall.

The implemented algorithms for the second step are as follows. The first implementation is called “Find Closest Unvisited Cell.” This algorithm will consider the cells the robot has not yet been in, and pick the one that is closest to the robot. The second algorithm is “Find Closest Unvisited Cell to Start.” This is similar to the first algorithm except that the base reference point is the start instead of the robot’s current position.

The third algorithmic step, returning to the start after finding the goal, also has multiple implementations. The first implementation is “Assume No Walls.” This is an optimistic path finding algorithm that will plot a path back to the start assuming that all unknown potential walls will not have walls. If a wall is found along the return path, a new path needs to be calculated. The second is “Assume Walls.” This is a pessimistic algorithm that will plot a path back to the start using only the information gathered. All the unknowns are assumed to be a wall when traveling back to the start with this algorithm. A new path will never need to be calculated because the return path is based upon a path the robot knows exists. The third is "Assume Walls Unless Cell is Visited." This is in between the previous two algorithms. It will only assume an unknown is not a wall if the cell it is attempting to enter has already been visited. Otherwise it will assume that all unknowns are walls.

Twelve different algorithms can be created with the different combinations of the above steps. We number the algorithms for later reference.

Algorithm Number	Left Right Sweep	Follow Right Wall	Closest to Robot	Closest to Start	Assume No Walls	Assume Walls	No Walls if Visited
1	X		X		X		
2		X	X		X		
3	X			X	X		
4		X		X	X		
5	X		X			X	
6		X	X			X	
7	X			X		X	
8		X		X		X	
9	X		X				X
10		X	X				X
11	X			X			X
12		X		X			X

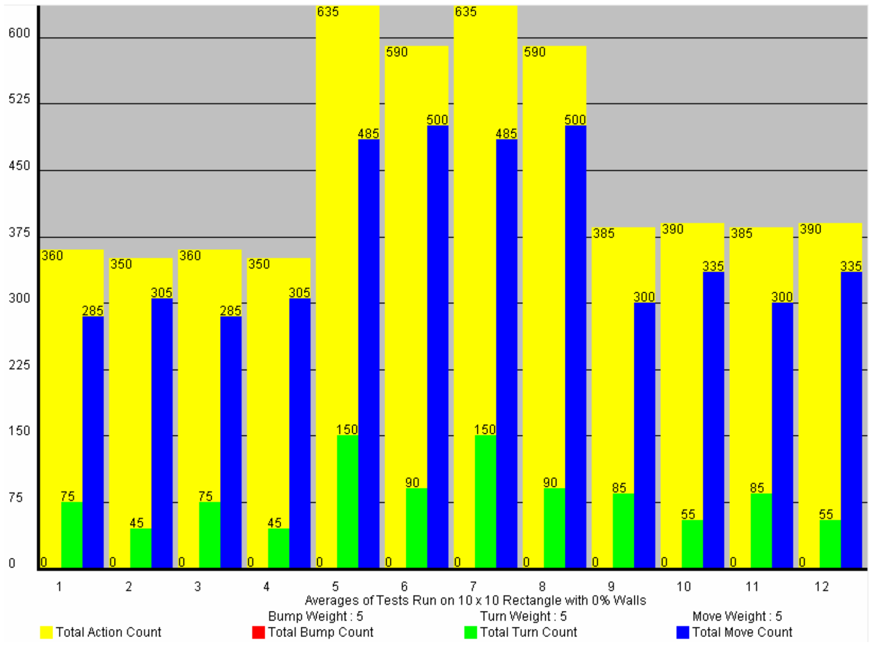
We decided to run many different test cases pitting each of the algorithms against each other to determine which algorithm, if any, is the best for a situation. The test case consisted of the average of five hundred runs, for each algorithm, in random environments with constant variables for its creation. The environment's size was 10 by 10 cells. The initial percentage of obstacles was zero and we incremented it by 10 after each test case, to maximum of 30 percent.

The results from these tests were split into three different categories. The robot logged the number of bumps, turns and moves it took during the duration of each test. With these different statistics we can apply weights to each type of movement and anticipate the amount of time it would take different robots to reach the goal in the same situation. The weights are multipliers applied to the different movement types. We used three different sets of weights against our results, each adding up to a total of 15. The first weight applied was five on each of the three movement types. This was to model the robot that takes the same amount of time to do all three movements. The second is three on one and six on the other two. This set demonstrates the robots that are quick at one of the movements and slower at the other two. The third is nine on one and three on the other two. This weight set describes the robots that are really slow at one of the movements and fast at the other two movements.

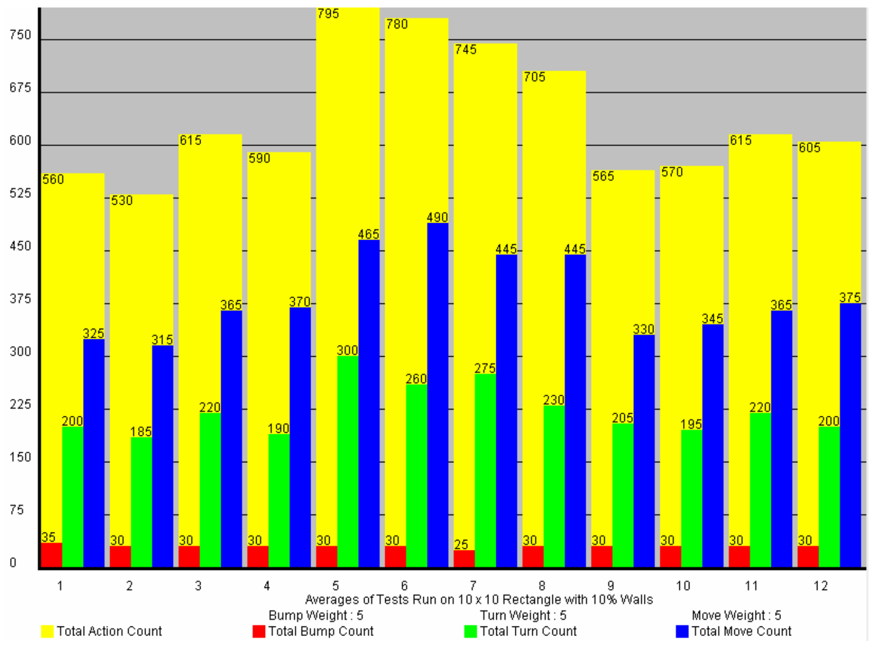
After putting all this together, we are testing 12 different algorithms in four different situations. Each of these situations yield seven different result sets, after being weighted for the assortment of robots that are being represented. The analysis of these different result sets should provide us with an accurate picture of what combinations supply better results than others. These good algorithms will be further investigated later to determine better combinations for robot navigation.

Results

The first test run was ten by ten cells with zero obstacles. The results that this test presented were very interesting. It seems that the algorithms containing the pessimistic end algorithm "Assume Walls" has higher movement counts and higher turn counts, which almost double the movement and turn counts of the algorithms that don't use "Assume Walls." Some characteristics of the algorithms are clearly showing in this result set. Algorithms with "Follow the Right Wall" have a higher movement count than the ones that have "Left Right Sweep." Algorithm numbers 2 and 4 have the lowest total count in all result sets except the two that put more weight on the move count. Algorithms 5 and 7 took the longest in every single result set. Algorithms 1-4 are the lowest and algorithms 9-12 are close behind and algorithms 5-8 take significantly longer.

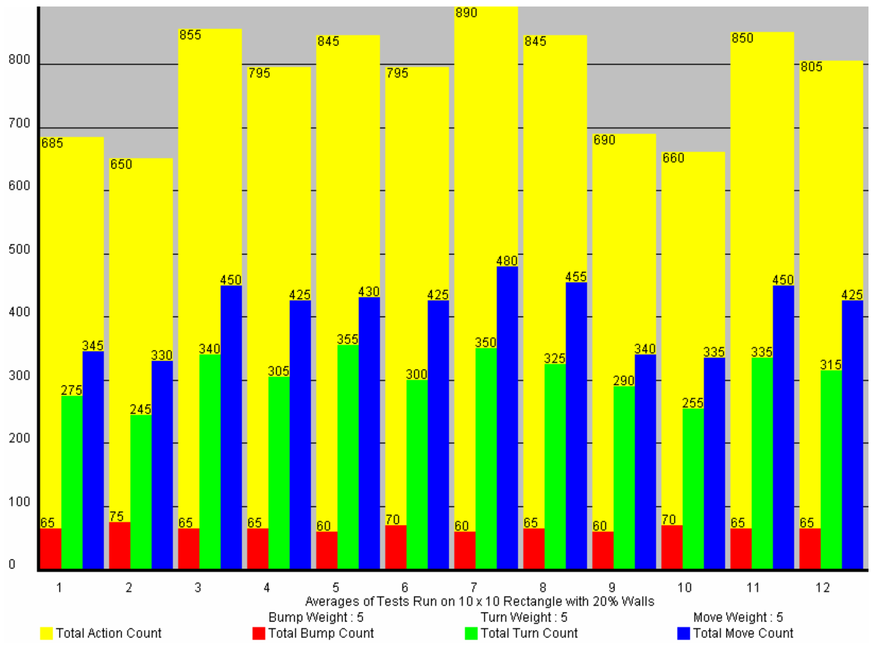


The second test run increased the percentage of obstacles to ten percent. This result set has additional information that wasn't seen in the first test run. Since there are obstacles in this test run we can find more information about the algorithms. Algorithm 1 had the most bumps while algorithm 7 had the least. This information may not be very useful because the difference between the best and worst count was 2 bumps. Once again the longest algorithms are 5-8; the "Assume Walls" algorithm appears to be responsible for the slow completion time again. An interesting variance from the first set is that the



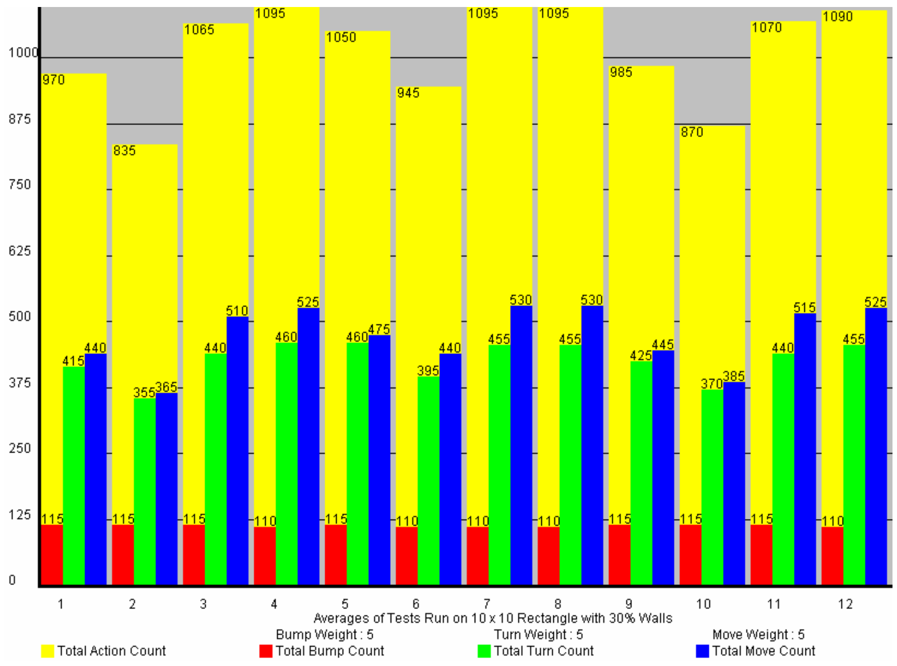
fastest four algorithms were the ones containing "Find Closest Unvisited Cell." One odd relation is the two slowest also contained "Find Closest Unvisited Cell" as part of their algorithm.

The third test run raised the percentage of obstacles up to twenty percent. With the increase in density of the obstacles some interesting patterns formed. The quickest algorithms were once again the ones containing "Find Closest Unvisited Cell." A different pattern emerged in this run. "Left Right Sweep" combined with "Find Closest



Unvisited Cell to Start" took the longest. Also the weighting didn't make much of a difference in the result set. They may need to have more drastic differences in weights to have an impact when the density of objects is higher. Algorithm 7 continued to rank among the lowest for bumping, but it again had the highest count to finish.

The final run had the density at thirty percent of the cells. Most of the algorithms took a relatively long time to complete this run. The only two algorithms that completed this run comparatively fast were 2 and 10; they both contain the algorithms "Find closest Unvisited Cell" and "Follow Right Wall". Algorithm 7 once again took the longest to complete the run, but this time it was joined by 3, 4, 5, 8, 11, and 12. Most of these algorithms contain "Find Closest Unvisited Cell to Start".



Algorithm	Total Action Count	Total Bump Count	Total Turn Count	Total Move Count
1	970	115	415	440
2	835	115	355	365
3	1065	115	440	510
4	1095	110	460	525
5	1050	115	460	475
6	945	110	395	440
7	1095	110	455	530
8	1095	110	455	530
9	985	115	425	445
10	870	115	370	385
11	1070	115	440	515
12	1090	110	455	525

Conclusion

Many interesting patterns emerged in the result sets as the percentage of the obstacles rose. Any algorithms containing "Find Closest Unvisited Cell to Start" or "Left Right Sweep" became increasingly inefficient as the obstacle number rose. The "Assume Walls" algorithm was very slow, but seemed to begin to hold its own as the number of walls rose. It may become useful if the obstacle percentage rose even higher. The winning algorithm is number 2. The pieces didn't get hindered as much as the other algorithms as the obstacle percentage rose, and it was consistently among the quickest algorithms.

References

[1] J. F.Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, Massachusetts, 1988.
[2] C. I. Connolly, J. B. Burns, and R. Weiss. Palth planning using Laplace's Equations. In *IEEE International Conference on Robotics and Automation*, pages 2102-2106. IEEE, May 1990.
[3] V. J. Lumelsky and A. A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2(4):403-430, 1987.
[4] R. A. Brooks. Solving the find-path problem by good representation of free space. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(3):190-197, March/April 1983
[5] E. Gat. *Reliable Goal-Directed Reactive Control of Autonomous Mobile Robots*. PhD thesis, Virginia Polytechnic Institute and State University, 1991.
[6] S. Ratering and M. Gini. Robot Navigation in a Known Environment with Unknown Moving Obstacles. In *IEEE International Conference on Robotics and Automation*, pages 25-30. IEEE, 1993.
[7] M. G. Slack and D. P. Miller. Path planning through time and space in dynamic domains. In *Sixth National Conference on AI*, pages 1067-1070, Seattle, WA, July 1987. AAAI.