A Comparative Analysis of Three Algorithms for Uniform Coverage

Jeramey Tyler and Suranga Hettiarachchi
Computer Science Department
Indiana University Southeast
New Albany, IN 47150
812-901-2698
jeatyler@ius.edu, suhettia@ius.edu

Abstract

Our research objective is to design algorithms for autonomous mobile robots to achieve uniform coverage in obstacle-laden environments. This paper presents a comparative analysis of three algorithms to accomplish this objective: random wanderer, primary movement, and A* path finding. We use randomly generated simulated environments with maps to test our algorithms. We present our results based on performance metrics where we vary the number of robots and obstacle densities.

Introduction

There are environments that are dangerous for humans to navigate. Natural and manmade disasters can leave areas strewn with hazardous debris, gas leakages, fires, and radiation. Some of the environments may have heavier obstacle densities than others. It is possible to use autonomous mobile robots to navigate these environments and perform tasks, such as search and rescue, while reducing the risk to human life. We are motivated to design algorithms that are capable of accomplishing uniform coverage for search and rescue missions.

In this work, our objective is to investigate and provide a comparative analysis of three algorithms for the uniform coverage problem. The three algorithms are random wonderer (RW), primary movement (PM), and A* path finding (A*). In each algorithm, robots share a series of communal maps. These randomly generated maps are updated by sensor data received through the range sensors of the mobile robots. To update the map, the robots should determine which portions of the map have not been viewed yet and navigate a path around obstacles to those portions. The robots continue to navigate until 100% coverage is achieved or there are no acceptable movements available.

We test our three algorithms using randomly generated simulated environments with maps. We have designed and implemented a user-friendly simulation for our experiments. The design of the simulation allows the user to specify many of the experiment parameters. We use percentage of the environment viewed (map coverage) and number of physical movements taken by the robots as our performance metrics. The second measure of the performance metrics, movements taken by robots, is analogous to the time spent by robots in the environment.

Background

Prior research in this area is numerous. Some of our work relies on prior research. One of our algorithms, A*, relies heavily on the A-Star search algorithm [1]. Given a starting point and an ending point, the A-Star algorithm will determine a least cost path to travel from start to goal. The A-Star algorithm was first published in 1968 in [1], and it is modeled off of the work by Edsger Dijkstra described in [2]. In the specific context of uniform coverage, some of the relevant papers are [3], [4], and [5]. In [3], Soucy et al., described a system for automatically digitizing a completely unknown object to a prescribed sampling density to achieve complete

surface coverage by analyzing sensor trajectories. Pimenta et al., in [4] addressed the problem of simultaneously covering an environment and tracking intruders. They translated the uniform coverage problem with time-varying density functions under the locational optimization framework. In [5], Maxim et al., presented a physics-based algorithm for uniform coverage in tunnels using a swarm of robots in linear formations. Majority of prior work addresses uniform coverage problem with 10%-15% obstacle density or no obstacles at all in the environments. We intend to achieve similar results with less computational complexity but in environments with greater obstacle density of up to 30%. Our work does not utilize swarm robotic approaches.

Methodology

Three Algorithms

In our experiments we examined three algorithms, each of which we will describe below. Each of the three algorithms that we examined implements the same technique for determining the direction that a robot is facing. A robot is only capable of facing either a cardinal or an ordinal direction of the compass. Each direction is represented as an integer value ranging from zero to seven. This integer value is called the robot's bearing. Bearing values begin with zero lying on the positive x-axis and is increased by one each time the angle is increased by 45 degrees in the counter-clockwise direction. Multiplying a robot's bearing by 45 will produce the angle from zero that the robot is currently facing.

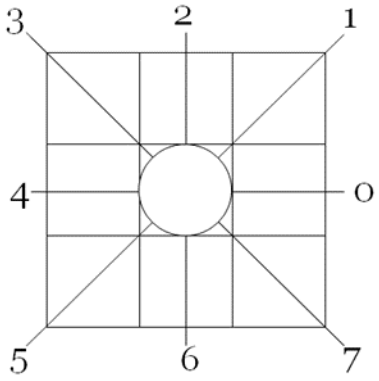

Figure 1: The bearing system is represented as a series of integer values ranging from 0 to 7. Each integer represents a 45° division of a circle.

Random Wanderer

The first algorithm that we examined was the Random Wanderer (RW) algorithm. RW is the baseline against which we measure the efficiency of the other algorithms.

A robot running RW has three movements that it is capable of performing: a 45 degree rotation left, a 45 degree rotation right, and a one cell forward movement. Before a robot performs a movement a random number between zero and two inclusive is generated. If the number generated is a zero the robot will rotate left; if the number generated is a two the robot will rotate right; if the number generated is a one the robot will move forward one cell if that cell is currently unoccupied.

Primary Movement

The second algorithm that we examined was the Primary Movement (PM) algorithm. PM is an algorithm of our own design. In PM a robot has one goal, to explore areas of the map that

have not previously been explored. To achieve its goal, a robot has two movements that it can perform: moving forward one cell and rotating 45 degrees to its left.

Before a robot performs a movement it determines if moving forward one cell will allow it to explore an unexplored portion of the map. If a forward movement will allow the robot to explore an unexplored portion of the map and the cell in front of the robot is unoccupied the robot will move forward one cell; otherwise, if a forward movement would cause the robot to explore a previously explored portion of the map or if the cell in front of the robot is occupied the robot will rotate 45 degrees to its left. If a robot ever rotates a full 360 degrees and is unable to perform any forward movement the robot will cease operation.

A* Path Finding

The final algorithm that we examined was the A* path finding (A*) algorithm. A* utilizes a modified version of A-Star which is described below.

In A* we find the unexplored cell in the map that is closest to a particular robot, this cell is set as the goal of A-Star. In a standard A-Star implementation cells are evaluated until the goal cell has a parent assigned to it [6]. In our implementation of A-Star cells are evaluated until a cell has a parent and that cell is close enough to the goal cell that the goal cell would lie in a sensor's field of vision.

In order to determine the unexplored cell in the map that is closest to a particular robot, we devised a technique to iterate outward through a map from a given cell; we call this technique spiral iteration. Using the standard technique of iteration to search a map would return the unexplored cell closest to the starting cell, the top left corner, of the map. Using spiral iteration we are able to specify the starting point of iteration, the robot's current location, and iterate through every cell of the map in an increasing spiral until an unexplored cell is found. The first cell that is found using spiral iteration will be an approximation of the closest unexplored cell to the robot. An example of spiral iteration can be seen in Figure 2. The green cell (light grey) represents the current position of the robot, the red cells (dark grey) represent valid goals, and the white cells represent invalid goals. In the example, standard iteration would return a valid goal in 16 iterations but the returned goal would not be the closest goal to the robot; using spiral iteration would return the closest valid goal in 7 iterations.

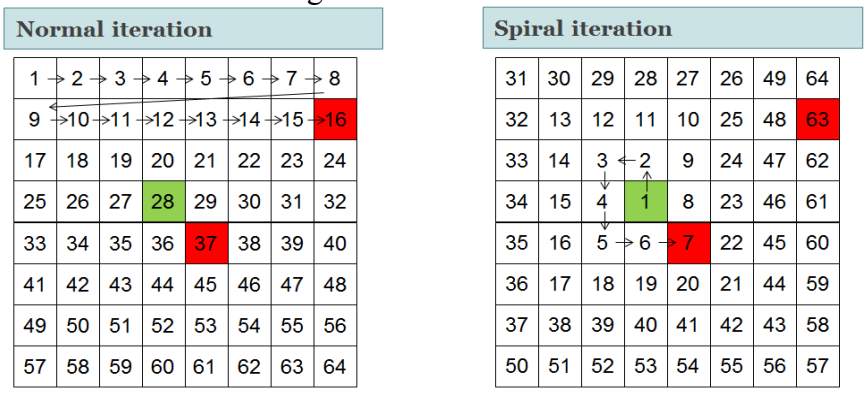

Figure 2: Standard iteration (left) vs. spiral iteration (right). The green cell represents a robot, the red cells represent unexplored cells, and white cells represent cells that are unoccupied and explored.

Environment

In an effort to maintain consistency across experiments, we designed and implemented an application that allows us to customize experiments. The application is divided into two views, the parameter selection window and the simulation window.

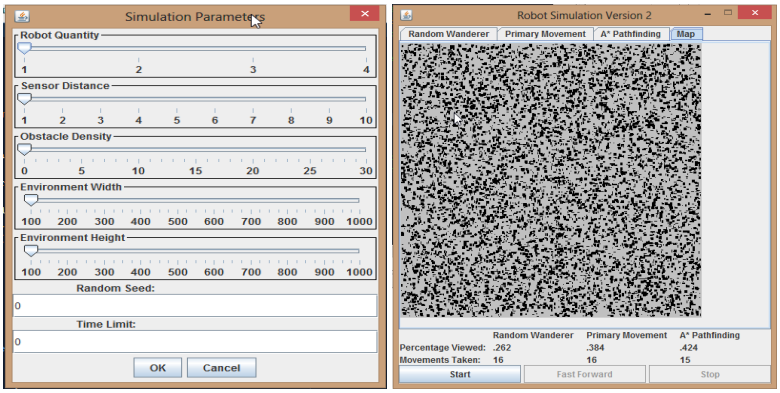

Figure 3: The panel for setting environment parameters (left) and a randomly generated map with panels for three algorithms (right).

Parameter Selection Window

When the application is started the user is presented with the parameter selection window. In the parameter selection window the user is able to select the parameters that will be used during the current experiment. There are 7 parameters that the user can specify: robot quantity, sensor distance, obstacle density, environment width, environment height, random seed, and time limit. Robot quantity is the number of robots in the experiment. Sensor distance represents the distance that the robots' sensors can reach in relation to the robot's size. Obstacle density is the percentage of the environment that is occupied by obstacles. Environment width and height represent the quantity of cells that make up the environment along the corresponding axes. The random seed is an optional parameter that represents the seed value for the randomly generated map. Time limit is an optional parameter that represents the number of robotic movements an algorithm is allowed to perform before it ceases operation.

Simulation Window

After the user has selected the experiment parameters they will be presented with the simulation window. In the simulation window, the user is presented with a tabbed panel, the experiment output panel, and the control panel. The output panel displays the percentage of the environment that has been viewed and the number of robotic movements performed by a particular algorithm. The control panel provides the user with buttons that allow them to control the current status of the application.

The tabbed panel presents the user with four tabs: Random Wanderer, Primary Movement, A* path finding, and Map. The Map tab provides a visual representation of the environment that each algorithm will be navigating. Black cells represent portions of the map that are occupied by obstacles while grey cells represent unoccupied cells in the environment. The Random Wanderer, Primary Movement, and A* path finding tabs each provide a visual representation of the environment as it has been explored by the corresponding algorithm. In an

algorithm's tab dark grey cells represent areas of the map that have not been explored, red cells represent a robot, and green cells represent a robot's sensor.

Experiment Setup

We conducted four different control studies using the three algorithms where obstacle density of the randomly generated environment is the focus. The obstacle density of 0%, 10%, 20% and 30% were tested with varying number of robots of 1 to 4 for all three algorithms. The results of all control studies presented in this paper are averaged over 10 independent runs.

Results

Table 1 through Table 4 shows the percentage of the environment covered and the movements taken by robots in each algorithm to achieve uniform coverage where the number of robots performing the task varies from 1 to 4.

# of Robots	RW		PM		A*	
	Coverage	Movements	Coverage	Movements	Coverage	Movements
1 Robot	15.30	2760.7	34.96	468.9	100.0	2757.2
2 Robots	14.55	1405.5	43.40	394.6	100.0	1395.2
3 Robots	26.65	1027.2	48.24	300.8	100.0	1016.4
4 Robots	24.49	758.3	56.92	268.8	100.0	701.4

Table 1: Coverage achieved and Movements taken with 30% obstacle density by 1-4 robots.

# of Robots	RW		PM		A*	
	Coverage	Movements	Coverage	Movements	Coverage	Movements
1 Robot	14.48	2235.1	45.67	583.3	100.0	2227.0
2 Robots	20.04	1187.1	52.87	442.2	100.0	1178.4
3 Robots	28.24	895.9	60.20	346.6	100.0	888.4
4 Robots	26.59	726.0	71.12	303.0	100.0	716.1

Table 2: Coverage achieved and Movements taken with 20% obstacle density by 1-4 robots.

# of Robots	RW		PM		A*	
	Coverage	Movements	Coverage	Movements	Coverage	Movements
1 Robot	17.47	1888.9	52.21	640.7	100.0	1883.2
2 Robots	23.51	1086.3	54.64	456.9	100.0	1081.0
3 Robots	28.83	757.2	70.76	398.3	100.0	753.0
4 Robots	36.82	684.9	82.82	370.8	100.0	674.3

Table 3: Coverage achieved and Movements taken with 10% obstacle density by 1-4 robots.

# of Robots	RW		PM		A*	
	Coverage	Movements	Coverage	Movements	Coverage	Movements
1 Robot	15.24	1622.6	100.0	1603.0	100.0	1621.0
2 Robots	31.13	958.1	97.53	959.2	100.0	957.0
3 Robots	37.38	753.5	100.0	566.0	100.0	749.0
4 Robots	38.76	562.9	100.0	421.0	100.0	553.0

Table 4: Coverage achieved and Movements taken with 0% obstacle density by 1-4 robots.

It is obvious that A* produces superior coverage in all four control studies. The performance of the PM algorithm degrades with increasing obstacle density though the

movements taken to achieve coverage remains relatively smaller and consistent. It is evident that PM should be able to achieve similar coverage as A* if more robots are deployed in the environment. It is also possible for PM to achieve this uniform coverage with fewer movements than A*. As expected, the worst performance is by RW, the baseline. PM's suboptimal performance can be attributed to its inability to acquire a goal outside of its sensor range while RW's poor performance is caused by the random determination of movements. A* is able to achieve superior performance because of its ability to acquire a goal anywhere in the map and determine a path to its goal.

Conclusion

In this paper, we explored the behavior of three algorithms, random wanderer, primary movement, and A* path finding to accomplish uniform coverage. We presented our results with a comparative analysis of the three algorithms using four control studies where we varied obstacle density of randomly generated environments and the number of robots. Our results confirm the superiority of the A* path finding algorithm. The performance of primary movement algorithm is promising but it requires further studies. Future work of this research will focus on physics based hybrid algorithms where robots adapt to switch from one control algorithm to another based on the environment conditions and swarm approaches as in [7].

Works Cited

1 Hart PE. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on System Science and Cybernetics. 1968 100-107.

2 Dijkstra E. A Note on Two Problems in Connexion with Graphs. In Numerische Mathematik 1. 1959. 269-271.

3 Soucy G, Callari FG, Ferrie FP. Uniform and Complete Surface Coverage with a Robot-Mounted Laser Rangefinder. IEEE/RSJ IROS. 1998. 1682-1688.

4 Pimenta LCA, Schwager M, Lindsey Q, Kumar V, Rus D, Mesquita RC, Pereira GAS. Simultaneous Coverage and Tracking (SCAT) of Moving Targets with Robot Networks. In: Chirikjian GS, Chosest H, Morales M, Murphey T, editors. Algorithmic Foundation of Robotics VIII Selected Contributions of the Eight International Workshop on the Algorithmic Foundations of Robotics; 1998. p. 85-99.

5 Maxim PM, Spears WS. Robotic Uniform Coverage of Arbitrary-Shaped Connected Regions. The Carpathian Journal of Electronic and Computer Engineering. 2010.

6 Lester P. A* Pathfinding for Beginners. [Internet]. 2005 [cited 2019 April 24]. Available from: http://www.policyalmanac.org/games/aStarTutorial.htm.

7 Spears D, Kerr W, Spears W. Physicomimetics Physics-Based Swarm Intelligence. Berlin Heidelberg: Springer; 2012. 3-124.