Dispersing robots in an unknown environment

Ryan Morlok and Maria Gini

Department of Computer Science and Engineering, University of Minnesota, 200 Union St. S.E., Minneapolis, MN 55455-0159 {morlok,gini}@cs.umn.edu

Summary. We examine how the choice of the movement algorithm can affect the success of a swarm of simple mobile robots attempting to disperse themselves in an unknown environment. We assume there is no central control, and the robots have limited processing power, simple sensors, and no active communication. We evaluate different movement algorithms based on the percentage of the environment that the group of robots succeeds in observing.

1 Introduction

The problem we address is that of dispersing a group of mobile robots in an unknown environment. We assume the robots do not know how many other robots are operating in the same environment, where those robots are located, and where those robots have been.

The primary motivation for this work comes from the Scout project [9]. The scouts are small, two wheeled robots with extremely limited processing capability. In general, the scouts are deployed by being hauled or launched into the environment by a larger robot. Their job is then to disperse throughout the environment so that it can be effectively monitored. Currently the scouts are teleoperated, but they can also perform some autonomous operations, such as hiding and watching for motion [9], by proxy processing over a radio link.

One of the major issues with these type of robots is communication. Since the robots are small, and therefore do not have a great deal of available electrical power, it can be difficult (and in some cases impossible) to generate a signal strong enough to communicate with all other robots. This problem is worsened by the fact that the scouts are designed to explore hostile environments, which may have physical characteristics that further hamper any sort of radio based communication. Because of this, we will constrain our algorithms not to require any explicit communication, and to use the sensors to communicate implicitly by observing cues from the environment. This type of communication, which is called *stigmergy* in the biology literature, is common in swarm approaches to robotics [1]. We will further assume that the robots have enough local processing power, so all computation is done locally.

2 Related Work

Coverage of terrain during motion is important in many application domains, such as floor cleaning, lawn mowing, harvesting, etc. A recent survey [2] classifies the existing algorithms for terrain coverage.

Wagner et al. [11] formalize the terrain covering problem and propose two algorithms. one called mark and cover (MAC), the second called probabilistic coverage (PC), both for single and multiple robots. They show how several cooperating robots can obtain faster coverage. Algorithms inspired by insect behaviors, such as ants, are becoming popular both for terrain coverage [6], where robots leave trails and cover the terrain repeatedly, and for optimization of paths [8].

The study by Hsiang et al. [4, 5] is the closest to our work. In their work, they examined methods for dispersing robots from fixed locations to cover the entire environment. They assume a continuous stream of robots would be entering the environment through specific, predetermined locations. The goal of the robots would then be to position themselves such that the entire area of the accessible space is covered. While this work has great properties/guarantees, it is not immediately applicable to the problem we are investigating in this paper. The reason is that while each robot only has extremely local knowledge of the environment, through the use of the deterministic movement and infinite supply of robots the information available at the point at which any given robot is located is sufficient to guarantee that the robot will make the correct choice. In our investigation, it would be impractical to assume that there are enough robots to cover the entire map and to guarantee that every robot can remain within sensor range of the other robots. In our experiments, we assume there are at most 50 robots present in the environment. In environments such as the Hospital World (shown later in Figure 2), this would allow possibly two robots per room explored. Clearly there are not enough robots to make Hsiang's algorithm feasible.

3 Motion algorithms

The purpose of this study is to examine how the selection of the movement algorithm for a multi-robot system affects the coverage of robot observation in a variety of environments.

We considered four distinct movement algorithms, all of them reactive in nature. Each movement algorithm controls two types of movements: forward/backward and turning. Turning can occur in place or while the robot is moving. The sensors available to the movement algorithms are 16 sonar range

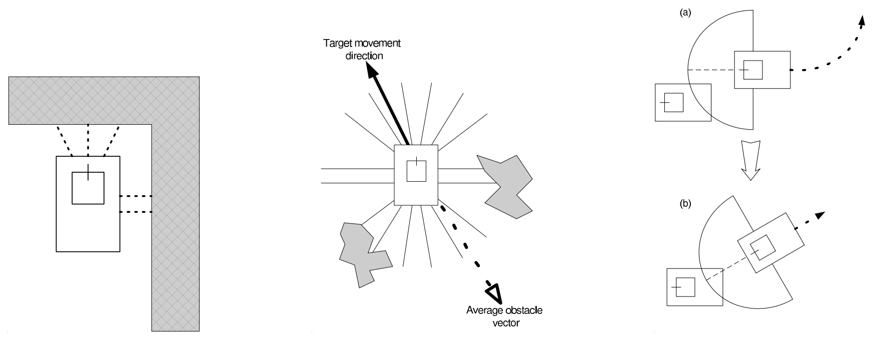

Fig. 1. *Left:* A robot using the FollowWall algorithm navigates a corner. *Center:* A robot using the SeekOpen algorithm calculates the average obstacle vector and moves in the direction opposite from this vector. *Right:* The Fiducial movement algorithm. (a) A robot detects another robot behind it within its sensor range, and begins to adjust its forward motion to turn so the detected robot will be immediately behind. (b) The detector robot has successfully positioned the detected robot behind it; the detector robot will now continue with straight forward movement.

finders, each of which returns the distance of the nearest object detected in the direction in which the range finder is pointing. A fiducial range finder is also used to determine the location of other robots, which is needed for the Fiducial algorithm. Robots have only local knowledge, they are not under global control (no central source knows the state of all of the robots), and do not have any knowledge of the environment other than what they can detect with their sensors.

3.1 Random Walk

The RandomWalk movement algorithm is the most basic of the algorithms we examined. A robot using this algorithm can be in one of two states: random forward movement or obstacle avoidance. In random forward movement, the robot moves forward with a small random turn factor between $-10\,°$ and $10\,°$ (the robot's path is curved) which is changed at random intervals, ranging between 10 s and 15 s. The amount of the turn is constrained to ensure the robot does not end up going in small circles. Once the robot detects that it has encountered an obstacle, it enters the obstacle avoidance state. In this state, the robot will stop, turn a random amount (in the range $120\,°$ to $240\,°$), and transition back to the standard forward movement state.

3.2 Follow Wall

The idea behind the FollowWall algorithm comes from the fact that in many indoor environments, if a robot could find an outer wall of the building and follow it, the robot would be led through much of the structure. A robot using the FollowWall algorithm will search for an obstacle (presumably a wall or large object in the environment) and then proceed to follow that wall indefinitely. In this algorithm, a robot has four states: find wall, align to wall, follow wall, and navigate corner. If the robot believes that it has lost the wall in any of the three non-find-wall states, it will reset back to the initial find wall state and search for a new wall to follow.

The major problem of this movement algorithm is that it assumes every obstacle encountered is a wall, rather than trying to determine if the observed entity is something smaller, such as a robot at close range. Because of this, when many robots using this algorithm are together, they will tend to perceive each other as walls and try to align themselves to each other. This is wasteful, since the alignment procedure will not effectively spread the robots out in the environment.

3.3 Seek Open

The SeekOpen movement algorithm causes a robot to move toward open areas in the map. The motivation for the SeekOpen algorithm is similar to the fiducial robot avoidance algorithm (discussed next). According to the assumptions of the experiment, all robots start out in the same general area, grouped fairly close together. Because of this, all the robots tend to have objects (generally other robots) close to themselves at the beginning of a run. The goal of the seek open algorithm is to motivate the robots to disperse as quickly as possible.

SeekOpen is executed by first calculating the average obstacle vector for all obstacles in sensor range. The average obstacle vector is computed by summing the vectors pointing to all of the objects within sensor range and dividing by the number of vectors summed. The magnitude of the vector must be large for objects close to the robot and small for objects far away. This is accomplished by setting the magnitude of a perceived obstacle vector equal to the maximum range of the sensors minus the perceived distance a given obstacle is from the robot, or by using some other function which decreases with distance, as done when using artificial potential fields for navigation [7]. After the average obstacle vector is computed, the goal of the robot becomes to move in the opposite direction of the average obstacle vector. The robot turns toward the direction of the negative obstacle vector. The rate of turn is determined by the magnitude of the average obstacle vector. This allows the SeekOpen algorithm to not run into walls as well as disperse from other robots. This is illustrated in Figure 1.

3.4 Fiducial

The Fiducial movement algorithm was inspired by the idea that the robots would be able to recognize other robots, and therefore move away from them. The original concept involved a simple signal (possibly a weak radio signal) that each robot would emit, so that another robot could detect the signal, and determine an approximate distance to the originating robot based solely on signal intensity. The problem of moving away from other robots would then become a goal of finding areas in which signal intensities are low, which can be done by any hill-climbing algorithm. Unfortunately, there was no straightforward way to implement such as system within Stage, and therefore an alternative method was sought.

The solution to the simulation problem was to use a fiducial device (generally used to find beacons in the Stage simulator) and attach a beacon to every robot. This allows a given robot to know the polar coordinates of other robots within sensor range (sensor range is a semi-circle of fixed radius) with respect to its own position and orientation. The information can be used to steer away from other robots.

With the fiducial information, implementation of an avoidance algorithm is straightforward. Whenever a robot detects another robot within sensor range, the robot adjusts its movement so that it is moving away from the detected robot. When no robots are in sensor range, a robot simply moves according to the random walk algorithm. If at any time a robot encounters a physical obstacle such as a wall, the obstacle avoidance technique takes precedence over whatever movement algorithm the robot is currently executing.

4 Simulation Environment and Data Analysis

To compare the algorithms, we performed a large number of experiments within the Player/Stage [10, 3] simulator. The virtual robot used for experiments is a rectangular, four-wheeled, Pioneer-like robot. Although the motivation for this work comes from the scout project, the scouts are not currently modeled in Stage, and this motivated the change in platform. The robots used in the simulations have 16 sonar range finder sensors, a pan-tilt-zoom camera with blob finding software capabilities (not used for any experiments), a laser fiducial finding device, and a truth device (a device used in the simulator to extract information about the robot's position status to record experimental data). The dimensions of the simulated robots are 33x44 cm.

The experiments were carried out by executing each of the four movement algorithms in five different simulated environments with different numbers of robots (10, 20, 30, 40, and 50) and two different durations (5 min and 10 min). The environments are described in Table 1 and shown in Figure 2. In each experiment the robots started out clustered together near the center of the map, each robot facing a random direction (except for the house world where the robots began in the left most room).

Experiments were carried out within the Player/Stage [10, 3] architecture. The Player Java client library was used to control the robots for all experiments. Each robot's control ran on its own thread, and all robot control code was executed on the same machine as the Stage simulation. This was done

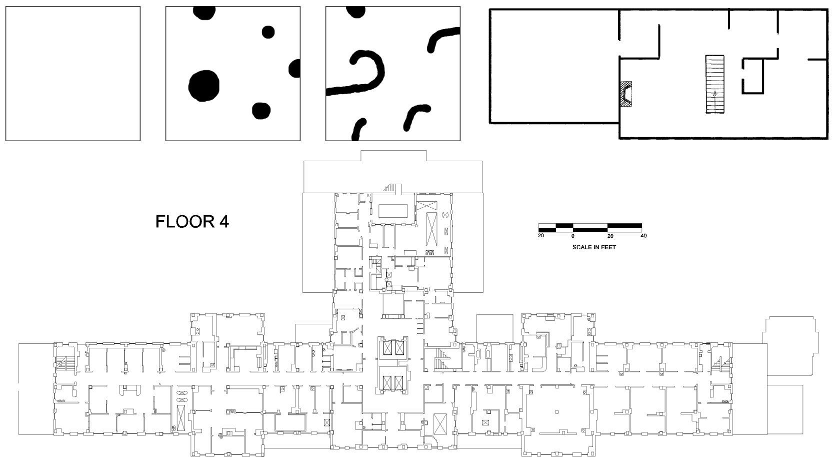

Fig. 2. The simulated worlds used in the experiments. *Top:* Square, Convex, Concave, and House worlds. *Bottom:* Hospital World.

World	**Size**	**Key Features**
Square	30x30 m	Simplest world, large open area designed to provide a baseline for other worlds
Convex	30x30 m	Simple world with basic, convex obstacles; no locations where robots can get trapped
Concave	30x30 m	More extended, concave barriers in which robots can get trapped if they are unwilling to backtrack
House	41x16 m	World modeled after a simple house blueprint; robots interact with a simplified map of a real world environment
Hospital	140x54 m	Complex world (packaged with Stage) designed to test robots in complicated world with many corridors and rooms

Table 1. Simulated test environments; see Figure 2 for visual reference.

primarily because of the difficulty of starting all robots simultaneously on multiple machines. Each of the experiments was run four times.

Because the Stage simulator does not provide built in utilities for analyzing the performance of the robots as they observe the environment, snapshots of relevant data were taken from the simulation for later off-line analysis. This was done by attaching a Truth device (a simulated device in Stage) to each robot. The Truth device is a device that can either get or set the world coordinates/orientation of any object to which it is attached. For each robot, an additional thread running on the same JVM queried the robot via the Truth device and recorded its position/orientation once per second. Data for all robots for a given experiment was written out to the same file. The data could then be used later (in combination with the world map) to determine the observation coverage of the environment.

For each experiment, a single, large file containing the position coordinates of all the robots was created. In order to determine observation coverage of the robots, the data was analyzed in combination with the world map used in the experiment. The procedure was implemented in Matlab.

First, all data points are loaded from the file. A binary bitmap of the same size as the world bitmap is created. Each pixel in the binary bitmap represents a discrete location in the world, which can be in one of two states: observed (1) or not observed (0). Initially, all pixels are not observed. For each location that is taken from the robot position file, all of the pixels within a set radius are set to observed. This is repeated for all the locations in the position file.

After all locations in the position file are processed, the observed region is oversized. Some areas are marked as observed when they are in fact unobserved. This is because the observed region includes pixels that are in fact obstacles, as well as those that are outside the accessible region of the world (a closed region from which the robot cannot escape). To account for this, a logical AND is performed on the observed binary bitmap and the interior region of the world. The interior region of the world is found by performing a flood fill, beginning at the start location of one of the robots. This leaves the observed bitmap as the locations that have fallen within the robots' observed radii at some point, that are valid, and accessible points in the world.

Note that this procedure is not a perfect model of what the robots could actually observe, especially in blueprint-like environments. Consider the situation shown in Figure 3. Suppose the elongated rectangle represents a wall separating two rooms (both of which are accessible to the robot). Here the algorithm will mark the area beyond the wall as observed, where clearly it is not. We developed a method to deal with this problem, but we did not use it in the data reported, since it significantly increases the time for data analysis and only leads to a minor improvement in accuracy for a relatively small view radius of the robots. Because of this, it should be noted that data reported for the House and Hospital worlds are slight overestimates of the actual values.

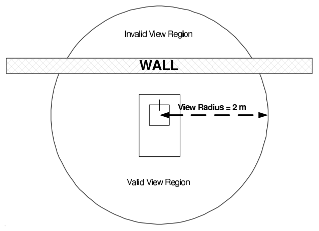

Fig. 3. A problem with the model used to calculate the region observed by a robot.

5 Experimental Results

Table 2 summarizes the results from both the five and ten minute experiment runs for the four algorithms in all five environments with different numbers of robots. The percents values in the table indicate the percentage of the accessible area of the environment that was observed by the robots.

	5 Minutes					*10 Minutes*				
	10	20	30	40	50	10	20	30	40	50
	Square Environment									
Fiducial	61.0%	77.4%	81.6%	85.5%	83.4%	81.2%	95.8%	96.9%	96.9%	97.1%
Follow wall	28.8%	57.4%	52.8%	53.8%	57.3%	56.6%	76.1%	77.0%	68.4%	78.6%
Random walk	50.2%	65.7%	76.4%	84.8%	79.8%	73.3%	90.6%	96.6%	95.8%	97.7%
Seek open	18.7%	36.7%	43.9%	47.3%	44.4%	41.8%	72.7%	70.7%	87.4%	77.0%
	Convex Environment									
Fiducial	52.2%	66.1%	71.6%	75.1%	75.6%	74.7%	82.9%	93.7%	94.4%	92.0%
Follow wall	22.7%	41.8%	36.2%	36.0%	37.5%	28.8%	64.6%	61.1%	59.8%	55.4%
Random walk	44.2%	58.5%	62.4%	64.9%	65.9%	69.8%	82.4%	84.0%	88.2%	93.1%
Seek open	18.9%	30.2%	40.9%	38.4%	33.6%	36.6%	52.8%	59.9%	65.9%	56.9%
	Concave Environment									
Fiducial	46.2%	58.7%	64.5%	67.3%	69.0%	67.8%	85.9%	85.8%	90.7%	88.5%
Follow wall	14.9%	34.6%	37.1%	35.7%	35.5%	35.5%	53.1%	58.3%	52.4%	56.3%
Random walk	33.8%	48.2%	56.2%	64.8%	59.8%	51.6%	73.4%	78.6%	79.0%	86.4%
Seek open	16.2%	29.4%	35.0%	33.5%	40.8%	36.1%	49.1%	53.4%	54.6%	60.9%
	Home Environment									
Fiducial	37.0%	40.0%	43.0%	40.9%	40.7%	39.7%	46.3%	47.2%	44.5%	43.9%
Follow wall	23.9%	22.3%	27.3%	30.9%	35.2%	31.4%	32.6%	39.1%	37.0%	40.7%
Random walk	33.3%	37.1%	40.0%	38.6%	40.4%	39.2%	41.6%	42.9%	44.4%	44.1%
Seek open	23.8%	31.6%	33.6%	33.4%	35.2%	35.5%	37.6%	36.6%	37.1%	36.9%
	Hospital Environment									
Fiducial	5.6%	6.0%	7.0%	7.7%	8.7%	8.4%	11.0%	10.6%	10.3%	13.3%
Follow wall	3.3%	3.6%	3.1%	3.7%	4.3%	4.1%	6.5%	5.5%	7.0%	5.0%
Random walk	4.7%	4.4%	4.1%	6.1%	5.9%	5.0%	6.5%	4.9%	7.5%	8.0%
Seek open	3.4%	3.5%	3.7%	3.9%	4.8%	3.4%	3.5%	3.8%	4.3%	5.1%

Table 2. Results for all experiments

The data shows that the FIDUCIAL algorithm performs the best in every situation. This is not surprising in that this algorithm has access to more data than the other algorithms. It does, however, indicate that knowledge of the locations of the other robots can help to speed up the exploration process.

The results for the FIDUCIAL algorithm can be seen graphically in Figure 4. This illustration shows that the robots had difficulty observing the more enclosed areas of the map, which is to be expected since the obstacle avoidance portion of the movement algorithm tends to favor regions in which it does not run into things. To get into the enclosed-hook region, the robot would have to intersect the obstacle, and then by random chance be redirected towards the hook region. None of the movement algorithms have the ability to be naturally attracted toward this type of region.

Figure 4 also shows the performance of the FIDUCIAL algorithm in the House world. All robots started out in the garage (the large, left most rectangle that is all green (light colored)) clustered together, facing different, random directions. Here we can see that the robots managed to make it into the house,

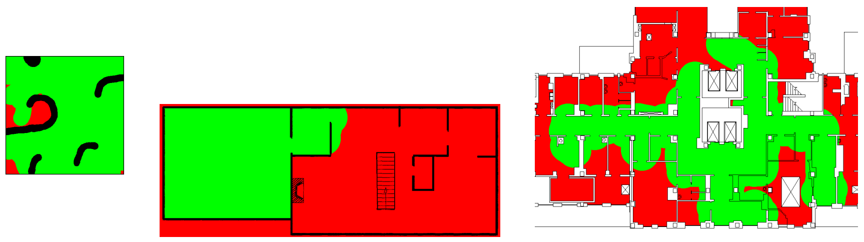

Fig. 4. Results of the Fiducial algorithm in the concave, house, and hospital worlds. Red (dark) indicates an unexplored area, green (light) indicates an observed region. The concave and house results were obtained by running 50 robots for 10 minutes, and the hospital results were obtained by running 50 robots for 1 hour.

but not past the first room. Doors are a natural obstacle for all movement algorithms in this experiment, in that they present a situation where there is obstacle on both sides, with a small opening. This can cause the avoidance logic to move away from the door area when it is encountered, unless the robot hits the door dead on.

What was somewhat surprising was that the RandomWalk movement algorithm performed second best among those algorithms tested, and generally close to the performance of the Fiducial movement algorithm. The similarity in performance between them should be expected, since for the majority of the time, both are acting in the same manner. The fiducial algorithm will only act differently than the random walk algorithm early in the simulation, when all of the robots are still clustered together. After the robots have spread out, the movement patterns become identical. The Fiducial algorithm simply speeds this spreading process.

Both FollowWall and SeekOpen have some innate flaws. The flaw in FollowWall is that it assumes two things. First, it assumes the robot to be in a closed region with no internal, isolated obstacles. Robots using FollowWall are likely to find an obstacle and orbit it indefinitely. The robot would require some form of self-position estimation to detect when it traverses the same positions over and over again. The second problem with FollowWall has been mentioned previously. The algorithm has no ability to distinguish between robots and obstacles (namely walls) because of this, in the initial robot cluster, many robots ended up trying to follow each other as walls, and ended up going in circles.

The fundamental problem with the SeekOpen algorithm is that it is subject to orbiting around local maxima for "openness." Some robots using this algorithm were observed to be traveling in small circles, remaining in one area of the map. SeekOpen also tends to prevent robots from going through narrow passageways such as doors. The hook in the Concave world, as shown, could act as one such narrow passageway.

6 Conclusions and Future Work

We have examined the performance of several movement algorithms at dispersing a group of robots in an unknown environment when starting from a initial cluster. The algorithms have been tested in various virtual worlds varying from a simple open area to a complex real-world building. The results show that even approximate knowledge of the locations of close-by robots helps the robots to spread out. The next step is to move into the real world. The combination of these algorithms and a group of small robots would make for an effective system for automatically exploring unknown worlds.

Acknowledgments

Work supported in part by NSF under grants EIA-0224363 and EIA-0324864.

References

1. E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press, 1999.
2. H. Choset. Coverage for robotics - a survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31:113–126, 2001.
3. B. P. Gerkey, R. T. Vaughan, K. Stöy, A. Howard, G. S. Sukhatme, and M. J. Matarić. Most valuable player: A robot device server for distributed control. In *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, pages 1226–1231, Oct. 2001.
4. T.-R. Hsiang, E. Arkin, M. A. Bender, S. Fekete, and J. Mitchell. Algorithms for rapidly dispersing robot swarms in unknown environments. In *Proc. 5th Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2002.
5. T.-R. Hsiang, E. Arkin, M. A. Bender, S. Fekete, and J. Mitchell. Online dispersion algorithms for swarms of robots. In *Proc. of the 19th Annual ACM Symposium on Computational Geometry (SoCG)*, pages 382–383, 2003.
6. S. Koenig, B. Szymanski, and Y. Liu. Efficient and inefficient ant coverage methods. *Annals of Mathematics and Artificial Intelligence*, 31:41–76, 2001.
7. J. C. Latombe. *Robot Motion Planning.* Kluwer Academic Publ., Norwell, MA, 1991.
8. D. Payton, M. Daily, R. Estkowski, M. Howard, and C. Lee. Pheromone robotics. *Autonomous Robots*, 11(3):319–324, Nov 2001.
9. P. E. Rybski, S. A. Stoeter, M. Gini, D. F. Hougen, and N. Papanikolopoulos. Performance of a distributed robotic system using shared communications channels. *IEEE Trans. on Robotics and Automation*, 22(5):713–727, Oct. 2002.
10. R. T. Vaughan. Stage: A multiple robot simulator. Technical Report IRIS-00-394, Institute for Robotics and Intelligent Systems, University of Southern California, 2000.
11. I. A. Wagner, M. Lindenbaum, and A. M. Bruckstein. MAC vs PC – determinism and randomness as complementary approaches to robotic exploration of continuous unknown domains. *Int'l Journal of Robotics Research*, 19(1):12–31, 2000.