Trail-Laying Robots for Robust Terrain Coverage

Jonas Svennebring
College of Computing
Georgia Institute of Technology
Atlanta, GA 30312-0280
jonas@cc.gatech.edu

Sven Koenig
College of Computing
Georgia Institute of Technology
Atlanta, GA 30312-0280
skoenig@cc.gatech.edu

***Abstract*— Robotics researchers have studied robots that can follow the trails laid by other robots. We, on the other hand, study robots that leave trails in the terrain to cover closed terrain once or repeatedly. How to design such ant robots has so far been studied only theoretically for gross robot simplifications. In this paper, we describe for the first time how to build physical ant robots that cover terrain. We show that a modified version of node counting can model the behavior of the ant robots and report on first experiments that we performed to understand their behavior better. These experiments confirm that our ant robots indeed cover terrain robustly even if the trails are of uneven quality, the ant robots are moved without realizing this, or some trails are destroyed. Finally, we report the results of a large-scale experiment where ten simulated ant robots covered a factory floor of 25 by 25 meters repeatedly over 85 hours without any ant robots getting stuck.**

I. Introduction

How to cover terrain once or repeatedly is an important problem in mobile robotics, for example, in the context of mine sweeping, surveillance, surface inspection, and guarding terrain. Consequently, researchers have developed many coverage methods. Most of these methods assume that the robots know their location. The currently popular POMDP-based robot architectures [5] attempt to overcome this problem by providing robots with the best possible location estimates [13]. However, this approach is complicated and can be brittle for robots that are small and cheap and thus have extremely noisy actuators and sensors. In this paper, we therefore explore trail-laying robots (ant robots) as an alternative to this approach. Our inspiration came from those researchers who have studied ant robots that can follow the trails laid by other ant robots, similar to ants that lay and follow pheromone trails [1]. Ant robots that follow trails arrive at their destination without having to know their exact location, which eliminates solving difficult and time-consuming localization tasks. They need only simple sensors, namely sensors that are able to sense the trails, which are artificial landmarks that can be carefully designed to simplify sensing. We utilize a similar idea to build ant robots that cover closed terrain once or repeatedly without knowing where they are in the terrain. As before, they only have to leave trails in the terrain and sense the trails in their neighborhood. Different from before, however, they need to move away from the trails rather then follow them.

In previous work, we and other researchers have studied theoretically how to build ant robots for gross robot simplifications. Unfortunately, the resulting approaches are not very practical for implementations on physical ant robots. In this paper, we describe for the first time how to build physical ant robots that cover closed terrain once or repeatedly. Our ant robots robustly cover terrain even if they do not have any memory, do not know the terrain, cannot maintain maps of the terrain, nor plan complete paths. In particular, they cover terrain even if the trails are of uneven quality, some ant robots are moved without realizing this (say, by people running into them and pushing them accidentally to a different location) or some trails are destroyed.

We first discuss related work, the robot that we use and how we augmented it with trail-laying and trail-sensing hardware and ant-coverage software, and first experiments that we performed to understand its behavior better. We then show how a modified version of node counting can model its behavior. Finally, we report the results of a large-scale simulation experiment where ten ant robots covered a factory floor of 25 by 25 meters repeatedly over 85 hours without any ant robots getting stuck.

II. Related Work

Empirical researchers have studied physical ant robots that follow trails. Some researchers have imitated nature closely [8] while others only got inspiration from it. The ant robots have typically left short-lasting trails in the terrain, such as heat trails, alcohol trails, and odor trails [11], [12]. Some ant robots have also used virtual trails only [3], [9], [14]. We, on the other hand, study ant robots that leave actual trails in the terrain to cover it once or repeatedly. The different task demands both longer-lasting trails (to be able to mark terrain that has already been covered) and different ant-coverage software. There was an earlier effort by Gabrieli, Katan and Rogel at the Technion to build terrain covering ant robots that lay trails using an evaporating liquid but no results have been reported on the success of this project.

III. Theoretical Foundation

Theoretical researchers have studied ant robots that cover terrain for gross robot simplifications [6], [16]. For

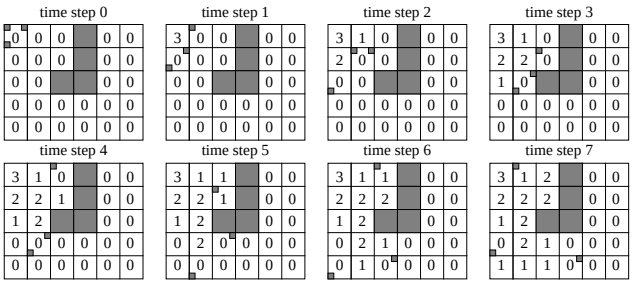

Fig. 1. Node Counting.

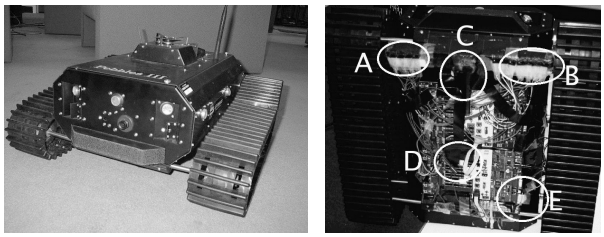

Fig. 2. Pebbles (left: birds-eye view; right: bottom view).

example, we have studied real-time search methods [7] for this purpose in earlier work. Real-time search methods are able to cover graphs repeatedly with a small cover time [15], [6], which suggests that they can be used to build ant robots that cover terrain once or repeatedly [6], [16]. The simplest real-time search method is probably node counting [10]. Ant robots that use node counting work as follows: Imagine that they operate on a four-connected grid and can move to each of the four neighboring cells of their current cell provided that the destination cell is traversable. They associate a number with each cell that corresponds to how often the cell has been visited by them. These numbers can be interpreted as markings that they leave at the cells. When an ant robot enters a cell, it increases the number of the cell by one. It then moves to the neighboring cell with the smallest number, breaking ties randomly. Figure 1 demonstrates the behavior of three ant robots that all use node counting. White cells are empty and gray cells are blocked. If a cell contains an ant robot, one of its corners is marked. Different corners represent different ant robots. For simplicity, we make the (unrealistic) assumption in the figure that the ant robots move in a given sequential order and that several ant robots can be in the same cell at the same time.

Single ant robots or teams of ant robots that use node counting cover grids and, more generally, all strongly connected graphs once or repeatedly. The ant robots do not even need to communicate with each other except via the markings. They only need to have very limited sensing and computational capabilities and do not need to know or learn a map of the terrain. They only have to leave markings in the terrain, sense markings at their neighboring

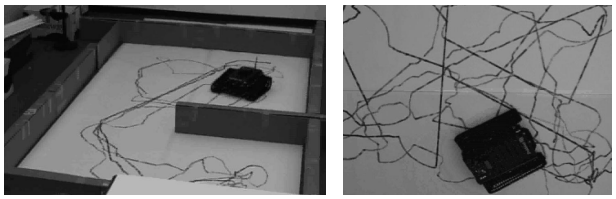

Fig. 3. Test Terrain (left: birds-eye view; right: top view).

cells, and change the marking of their current cell. Furthermore, ant robots that use node counting have advantages over ant robots that use other methods to cover terrain. For example, they navigate far more systematically than ant robots that perform random walks and, different from ant robots that use chronological backtracking (depth-first search), can be suspended and restarted somewhere else, without even knowing that they were moved or where they got restarted. This is important because sometimes ant robots might get pushed accidentally. Most of the time they will not even realize this. Ant robots that use node counting handle these situations automatically. It is known that the cover time of ant robots that use node counting can be exponential in (the square root of) the number of vertices on strongly connected undirected graphs [6], where the cover time is the time it takes to visit each vertex at least once. It is currently unknown whether the cover time is smaller on grids but experimental results indicate that it increases only linearly with the number of cells on grids and, for grids of a given size, remains small over time as the terrain is covered repeatedly. Unfortunately, despite these advantages, it is very difficult to build physical ant robots based on node counting and other real-time search methods. Their unrealistic assumptions include that the ant robots only move in discrete steps, that the ant robots have markings of a large number of different intensities available, that they can mark cells uniformly, and so on. Methods that cover continuous terrain [17] assume no actuator or sensor noise or largely depend on random motion. In this paper, we describe for the first time how to build physical ant robots that cover closed terrain once or repeatedly and demonstrate that they indeed cover terrain robustly even if the trails are of uneven quality, the ant robots are moved without realizing this, or some trails are destroyed. Figure 2 (left) shows our prototype ant robot, Pebbles, and Figure 3 (left) shows one of our test environments.

IV. The Ant-Coverage Hardware

For our research, we use a Pebbles III robot from IS Robotics, shown in Figure 2 (left) from birds-eye view and Figure 2 (right) from below. Its size with the tracks is 45 (width) by 44 (length) centimeters, and the size of its main body is 26 by 41 centimeters. Its on-board computer is based on a Motorola 68000 micro-controller

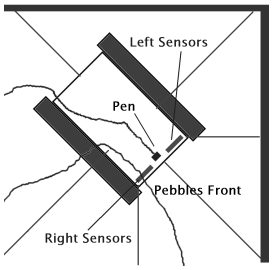

Fig. 4. Navigation Example.

that executes a special version of Common Lisp. It uses several peripheral micro-controllers to control its sensors and actuators, including its sonar sensors, its six infrared proximeters (sensing obstacles in the front, front-left, front-right, left, right, and rear of Pebbles), and its bump sensors for obstacle avoidance as well as its two motors for actuation. It navigates on two tracks and uses two rechargeable 7.2V NiCd batteries that enable it to operate for about 30 minutes. We added both a trail-laying and a trail-sensing mechanism to Pebbles. The new components are marked in Figure 2 (right). Eventually, Pebbles will lay trails by dripping a fluorescence or phosphorescence substance. For the time being, however, it lays trails using a black pen with a thick tip that is mounted below the front center of its body (C). To detect trails, it uses two one-dimensional arrays of proximeters (trail sensors) that are mounted to the right (A) and left (B) of the pen, which allows it to sense its trails right away and avoids it getting trapped in local minima. Each trail-sensor array consists of four Sharp 2L01 proximeters and covers an area of about 4 by 1 centimeters. Each proximeter detects the amount of light that is reflected from an LED via the floor to it. This allows it to detect trails since there is less surface reflection in darker areas, that is, on trails. The signal from the proximeter is then sent through a multiplexer to an analog-to-digital converter integrated in an Atmel AVR Mega163 micro-controller (D) and sampled approximately 2000 times a second. The value that corresponds to the darkest area is stored until a read command is sent from the external computer via an RS232 interface (E) approximately every five times a second. The value is then thresholded, reported, and subsequently reset to zero. This allows Pebbles to move fast without missing trails.

V. The Ant-Coverage Software

The ant-coverage software on Pebbles implements a schema-based navigation strategy [2] with two behaviors that are active at the same time, namely an obstacle-avoidance behavior and a trail-avoidance behavior. Pebbles switches the schema-based navigation strategy off only in the rare occasion when one of its bump sensors triggers, that is, if it ran into an obstacle. In this case, it moves away from the obstacle for a short time before it resumes its normal operation. We use the example situation from Figure 4 to explain the obstacle-avoidance behavior, the trail-avoidance behavior, and their combination.

A. Obstacle-Avoidance Behavior

The obstacle-avoidance behavior moves Pebbles away from walls and is fairly standard. The obstacle-avoidance vector is the weighted sum of a number of vectors. There is one vector for each obstacle sensor. It starts at the obstacle sensor and points towards the center of Pebbles. The length of the vector is inversely proportional to the distance of the sensed obstacle from Pebbles. Its weight is proportional to the importance of the obstacle sensor. The weight of the front obstacle sensor is 1/10, the weight of the rear obstacle sensor is 1/20, the weight of the front left and front right obstacle sensors is 1/40, and the weight of the left and right obstacle sensors is 1/55. The weights reflect, for example, that obstacles in front of Pebbles are more important than obstacles in its rear. The lines in Figure 4 show the placement of the obstacle sensors and the distance from Pebbles to the closest wall. The front, front-left, left and rear obstacle sensors sense walls. The resulting obstacle-avoidance vector has length 0.1367 and points about 137 degrees to the right of Pebbles, suggesting to turn away from the walls to the left of it.

B. Trail-Avoidance Behavior

The trail-avoidance behavior moves Pebbles away from trails. Its vector points away from trails with a fixed length of 0.1. There are four proximeters in the left trail-sensor array and four proximeters in the right trail-sensor array. Each proximeter of the left (right) trail-sensor array that senses a trail changes the angle of the vector by 17 degrees to the right (left). Thus, if all eight proximeters sense trails, then the vector points straight ahead. If all four proximeters of the left trail-sensor array sense trails but no proximeter of the right trail-sensor array senses a trail, then the vector points 68 degrees to the right. A problem with this approach is that the trail-sensor array observes only a small area of the terrain. This makes it difficult for Pebbles to determine a good trail-avoidance vector based on the current information from the sensor array alone. Pebbles therefore calculates the direction of the new trail-avoidance vector as above but then adds the direction of the old trail-avoidance vector to it, weighted with a decay factor smaller than one. (If the resulting angle is larger than 90 degrees to the left or right, it gets reduced to 90 degrees.) This way, the new trail-avoidance vector is influenced not only by the current information from the trail-sensor array but also the information from the trail-sensor array in the recent past. If Pebbles continues to detect trails on the same side, it turns more and more

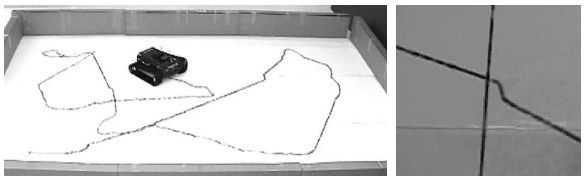

Fig. 5. Trail-Avoidance Behavior (left: bad; right: good).

sharply away from that side. If it stops detecting trails, it turns less and less sharply until it moves straight again. The decay factor ensures that the influence of trails decays over time, since they are further away from Pebbles and their actual location is no longer known with certainty. It cannot be set too low because otherwise Pebbles will make many small turns when it senses a trail. These turns need to be sharp to turn Pebbles sufficiently. Thus, the motion of Pebbles will be jerky when it senses a trail. On the other hand, the decay factor cannot be set too high either because otherwise Pebbles will continue to slowly turn even long after it has stopped sensing a trail. We determined experimentally that a decay factor of 0.5 resulted in smooth trajectories that turn Pebbles for only a short time after it senses a trail. So, if Pebbles continuously senses a trail on its left (that is, exactly one proximeter of its left trail-sensor array always senses a trail), then Pebbles will eventually turn 34 degrees to the right. Usually, the trail is no longer below its body before it achieves this turn angle, resulting in turns of only approximately 30 degrees before it moves straight again. The following table gives a fictitious example (that is different from the example from Figure 4) of exactly how the direction of the trail-avoidance vector is computed over time, assuming for simplicity that each trail is sensed by only one proximeter at a time. All angles are relative to Pebbles and were rounded to integers.

Time	Event	Old Angle (in degrees)	New Angle (in degrees)	
				0.0000
1	no trail	0.0000	$0.0000 + 0.5 \times 0.0000 =$	0.0000
2	trail on the right side	0.0000	$-17.0000 + 0.5 \times 0.0000 =$	-17.0000
3	no trail	-17.0000	$0.0000 - 0.5 \times 17.0000 =$	-8.5000
4	trail on the right side	-8.5000	$-17.0000 - 0.5 \times 8.5000 =$	-21.2500
5	trail on the left side	-21.2500	$17.0000 - 0.5 \times 21.2500 =$	6.3750
6	no trail	6.3750	$0.0000 + 0.5 \times 6.3750 =$	3.1875

For our main example, Figure 4 shows that the trail has been under the right trail-sensor array for the last 10 to 15 centimeters. The resulting vector of the trail-avoidance behavior has length 0.1 and points about 34 degrees to the left of Pebbles, thus suggesting to turn away from the sensed trail.

C. Combining the Behaviors

The obstacle-avoidance behavior and the trail-avoidance behavior both produce their own recommendation for how Pebbles should move. Pebbles always calculates the weighted average of the obstacle-avoidance and trail-avoidance vectors and moves in the direction of the resulting vector with a speed that is proportional to the length of this vector. The behavior of Pebbles is sensitive to the choice of these weights, and we thus optimized them by hand for the physical characteristics of Pebbles. The weight of the obstacle-avoidance behavior is larger than the weight of the trail-avoidance behavior since obstacle avoidance is more important than trail avoidance. The obstacle-avoidance behavior suggests to move right in the example from Figure 4, whereas the trail-avoidance behavior suggests to move left. This disagreement results in a short overall vector and thus a slow speed of Pebbles, which is desirable. The higher weight of the obstacle-avoidance vector results in an overall vector of length 0.0410 that points about 114 degrees to the right of Pebbles. Once Pebbles has turned away from the walls, its speed increases again and its navigation behavior is again mostly influenced by the trail-avoidance behavior.

The weight of the obstacle-avoidance behavior cannot be set too low because otherwise Pebbles can run into walls. On the other hand, it cannot be set too high either because otherwise Pebbles does not cover terrain close to walls and corners. Similarly, the weight of the trail-avoidance behavior cannot be set too low because otherwise Pebbles does not avoid previously covered terrain well enough and the cover time thus increases. On the other hand, it cannot be set too high either because otherwise trails can become barriers for Pebbles that are time consuming to cross and the cover time thus increases as well. To understand this phenomenon, assume that a trail separates two parts of a room that does not contain other trails. This trail is hard to cross for Pebbles because it gets repelled from it, as the last turn of Pebbles in Figure 5 (left) shows. Furthermore, the trail gets reinforced every time Pebbles approaches it but does not cross it. The emerging barrier can only be crossed easily once the trail density in the part of the room that Pebbles is in has become sufficiently high. Thus, the weight of the trail-avoidance behavior needs to get tuned carefully to allow Pebbles to cross orthogonal trails. Then, Pebbles first turns, say right, to turn away from the trail but continues to move forward and thus crosses the trail before turning. It then turns left, again to turn away from the trail, and eventually continues on its old trajectory, as shown in Figure 5 (right). If the weight of the trail-avoidance behavior is tuned carefully, this behavior clearly dominates, as shown in Figure 6.

VI. EXPERIMENTS

We conducted several experiments to evaluate the performance of Pebbles. We used areas of sizes from 2 by 2.5 meters to 3 by 4.5 meters. Their floor was covered with white paper, and they were surrounded with brown cardboard walls.

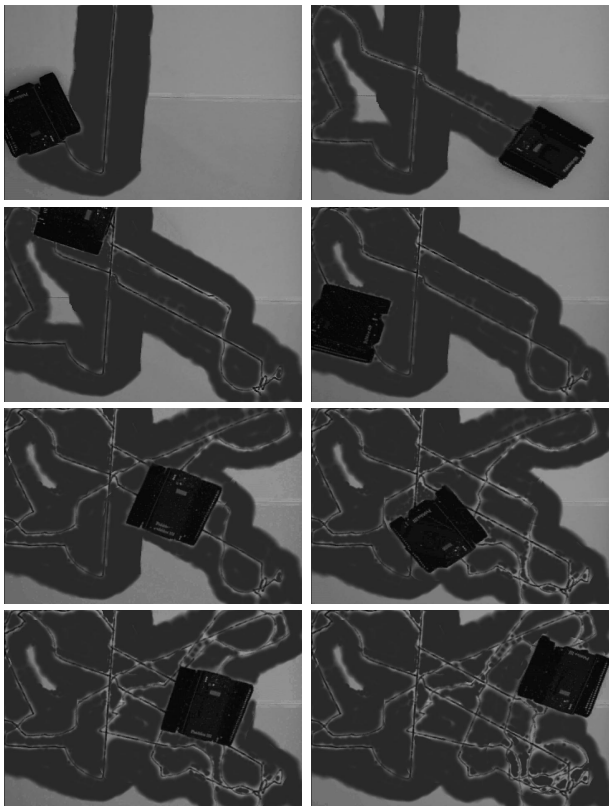

Fig. 6. Start of First Terrain Coverage (equal time steps).

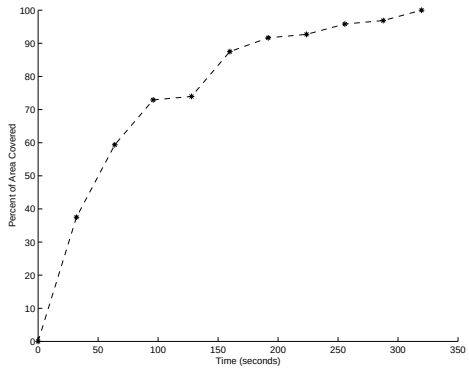

Fig. 7. Area Covered.

A. Regular Coverage

We first verified that Pebbles indeed covers terrain of different shapes multiple times without getting stuck, including the one shown in Figure 3 (left). Figure 6 shows the beginning of an example run in a simple rectangular terrain, where we shaded the covered area by hand in dark gray. We then measured the cover times for repeated terrain coverage. We say that the first coverage of the terrain is completed when each patch has been swept at least once by the body of Pebbles (except for patches close to walls). We say that an additional coverage of the terrain is completed when each patch has been swept at least once by the body Pebbles after the previous coverage was completed. For example, Pebbles covered an obstacle-free terrain of size 2 by 2.5 meters in 320 seconds. Figure 7 shows the covered area as a function of time, until the first coverage is completed. Pebbles covered terrain very quickly at the beginning (when it is easy to find uncovered areas since there are so many of them) but needed more time to find uncovered areas towards the end of the first coverage. This is true for the subsequent coverages as well. Pebbles needed 329 seconds for the second coverage and 489 seconds for the third coverage. The cover times increase because the terrain gets saturated with trails. This causes two problems. It makes it harder to place new trails and decreases the influence of each newly placed trail on the navigation behavior of Pebbles. Although more complex ant-coverage hardware and software will certainly be able to shorten the cover time, the current software of Pebbles does well given the small sensor field. For example, it faired well compared to a navigation behavior where Pebbles moves forward (without laying trails) while avoiding obstacles. In this case, Pebbles moved along the walls and covered about 50-60 percent of the terrain in 451 seconds but never covered the terrain completely in a reasonable amount of time.

B. Error Conditions during Coverage

One of the attractive properties of our ant-coverage software is that it covers closed terrain robustly even in situations where the trails are of uneven quality, where Pebbles is moved without realizing this, and where some trails are destroyed. We demonstrated the latter two properties earlier for rather unrealistic robot simulations [6] and demonstrate in the following that they continue to hold on Pebbles.

1) Trails of Uneven Quality: First, we measured the cover time when the pen of Pebbles was nearly exhausted. This is important because its pen is not constantly refilled with ink and its trails thus get lighter over time and harder to detect. This makes it more likely that Pebbles misses trails. Since Pebbles moves at different speeds, the pressure on the pen changes and the trails are not only faint but also of uneven quality. Since the intensity of the trails adds up over time, Pebbles continued to cover terrain robustly although the resulting cover time of 573 seconds is larger than the regular cover time of 320 seconds. Its coverage became also more uneven since some trails were stronger than others and became barriers for Pebbles.

2) Moving Pebbles: Second, we measured the cover time when Pebbles was moved without realizing this. This is important because people or other ant robots can easily run into Pebbles and accidentally push it to a different location. In the middle of a run, we moved Pebbles twice

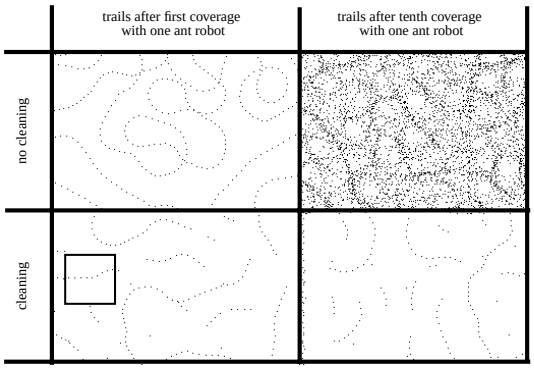

Fig. 8. Trails (with and without cleaning).

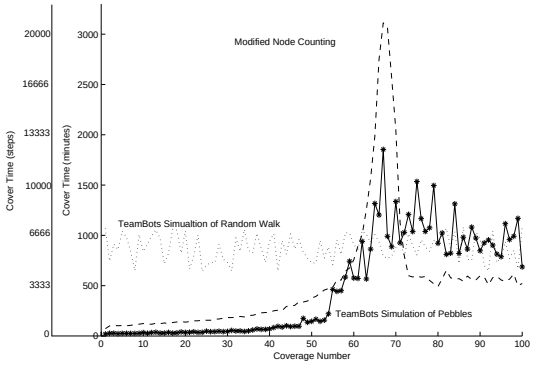

Fig. 9. Cover Time.

by a couple of meters in a random direction. Pebbles coped well with this problem since the ant-coverage software does not need to know its current location. The resulting cover time of 385 seconds is larger than the regular cover time of 320 seconds but the difference is not significant. This is not surprising since one of the random displacements moved Pebbles to the start corner (which increases the cover time) but the other one moved Pebbles to a location that it had not covered before (which decreases the cover time).

3) Removing Patches of Trails: Third, we measured the cover time when some trails were destroyed. This is important because trails can get destroyed accidentally due to wind, dust, rain, humans, or other ant robots. After Pebbles had covered about 90 percent of the terrain, we randomly placed three sheets of paper of size 15 by 20 centimeters on areas that it had already covered and that thus contained trails. Pebbles covered the three areas again only 103 seconds later since it was drawn to each area once it had sensed part of it and noticed that it did not contain trails.

VII. Computational Models of Ant Robots

So far, we have shown that our ant-coverage software makes Pebbles cover closed terrain robustly. We confirmed these results in TeamBots [4], a realistic robot simulator, with only slight modifications to the ant-coverage software. TeamBots has no battery limit and thus allowed us to extend Figure 7 to a large number of coverages. Note that the cover times of Pebbles cannot be compared to the cover times in simulation since the simulation uses its own clock, which does *not* correspond to actual time. The top row of Figure 8 shows that the terrain gets saturated with trails over time. (The square in the lower left corner corresponds to the simulated Pebbles.) The graph "TeamBots Simulation of Pebbles" in Figure 9 shows how this causes the cover time to increase for a terrain of size 10 by 10 meters. Eventually, the terrain is completely saturated with trails and the behavior of the simulated Pebbles degrades to a random walk, resulting in a cover time that is 40 times larger than the initial cover time. The cover time of node counting (not shown in the figure), on the other hand, remains small over time since node counting does not model that the terrain gets saturated with trails over time. We therefore modify node counting to provide a better computational model of Pebbles. Remember that ant robots that use node counting increase the number of their current cell by one and then move to the neighboring cell with the smallest number, breaking ties randomly. Ant robots that use the modified version of node counting, on the other hand, increase the number of their current cell by one only with probability $(k-x)/k$, where x is the current number of the cell and k is a constant (we use $k = 170$). Otherwise they leave the number of their current cell unchanged. Then they move to the neighboring cell with the smallest number, breaking ties randomly. To understand the idea behind the modified version of node counting, assume that each cell is divided into k small areas that are initially unmarked. Let x be the number of marked areas. If the ant robots randomly select one area of their current cell and mark it, then x increases by one with probability $(k-x)/k$, the probability that the chosen area was unmarked. Otherwise, x remains unchanged. The number of marked areas of each cell corresponds to its number. Thus, the probability with which ant robots that use the modified version of node counting increase the number of a given cell gets smaller and smaller over time, until the number of the cell is k and then does not change any longer. The modified version of node counting is a somewhat simplistic computational model of Pebbles but does model that it becomes harder and harder to add trails to areas that already contain a large number of trails, and that the terrain eventually gets saturated with trails and the behavior of Pebbles then degrades to a random walk. The modified version of node counting also models an interesting effect that we did not anticipate. Figure 9 shows that the cover times of both ant robots that use the modified version of node counting (in an obstacle-free grid of size 15 by 15 cells) and the

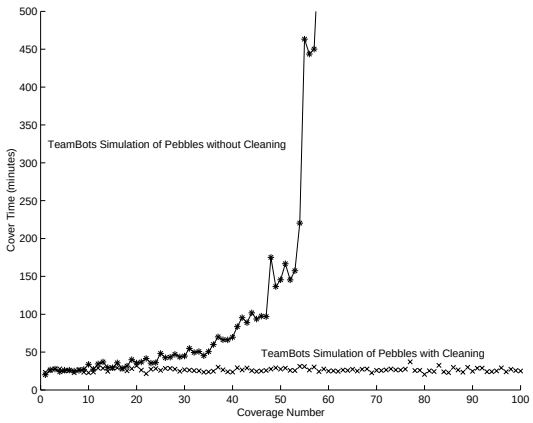

Fig. 10. Cover Time.

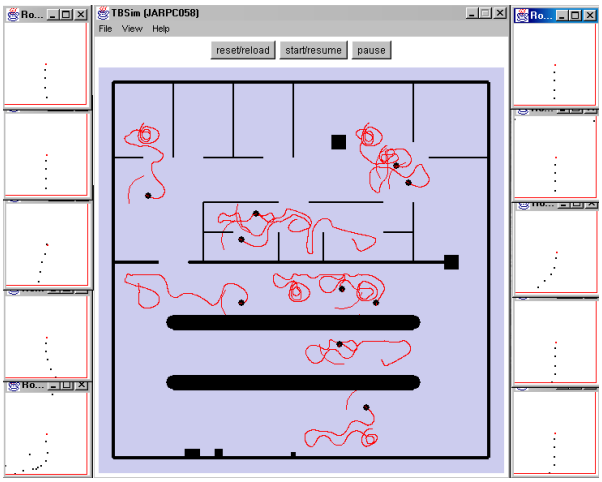

Fig. 11. Large-Scale Simulation Study.

simulated Pebbles (in an obstacle-free terrain of size 10 by 10 meters) peak and only then reduce to the cover time of a random walk, at which point the cover time does not change any longer. The modified version of node counting predicted the peak first and only then did we run experiments with our ant robots to verify it. We verified the cover time of a random walk by saturating the terrain with trails. It turns out that the peak is due to local minima in the trail density. For example, consider an area that is not completely saturated with trails but enclosed by areas that are completely saturated. It then takes both ant robots that use the modified version of node counting and the simulated Pebbles a long time to leave this area, longer than a random walk, since they first need to increase the trail density in their area to that of the surrounding ones, which takes time. This explains the peak. We assumed that each cell was divided into k small areas. This number is a parameter that determines how quickly the terrain gets saturated. We determined it empirically for our example to make the peaks of the cover times coincide.

VIII. Scaling Up

It is undesirable that the terrain gets saturated with trails over time since this increases the cover time. Thus, the trails need to either evaporate or get removed to keep the cover time small in the long run. Evaporating trails are problematic because the evaporation rate needs to get optimized for each application, for example, the size of the terrain. Thus, we propose to use long-lasting trails that Pebbles removes itself. However, the pen that Pebbles currently uses is not suited for this purpose. Instead, we simulated trails that consist of drops of a fluorescence or phosphorescence substance and used a cleaning method that removes all trails in two cleaning areas. Depending on the trail material, the trails could be removed with brushes, vacuum cleaners, heat (for alcohol trails), and light (for some photo chemicals) but it is future work for us to build such hardware. The bottom row of Figure 8 shows that this prevents the terrain from getting saturated with trails and Figure 10 shows that the cover time remains a small constant even after a large number of coverages in a terrain of size 10 by 10 meters. We also confirmed that the simulated Pebbles with the cleaning method continues to cover closed terrain robustly even in situations where it is moved without realizing this and where some trails are destroyed.

We then performed a large-scale simulation study to demonstrate that a large team of our ant robots with the cleaning method covers a large terrain repeatedly over long periods of time without ant robots getting stuck. We placed ten ant robots into an area of 25 by 25 meters that resembled a factory floor with two production lines and a number of office rooms, shown in Figure 11. This complex but very realistic environment is very difficult to cover due to its many narrow passages. We stopped the experiment after the ant robots had covered the factory floor for 85 hours without getting stuck. This result is important because teams of ant robots cover closed terrain faster and are more fault tolerant than single ant robots. The trails also coordinate the ant robots implicitly and allow them to cover terrain faster than without any communication.

IX. Conclusions

In this paper, we have described how to build physical ant robots that leave trails in the terrain to cover closed terrain once or repeatedly. The ant robots do not need to be localized, which completely eliminates solving difficult and time-consuming localization problems. We showed that a modified version of node counting can model the behavior of the ant robots and showed experimentally that physical ant robots robustly cover terrain even if the trails are of uneven quality, the ant robots are moved without realizing this (say, by people running into them), and some trails are destroyed. We also showed how ant

robots can keep the cover time small when repeatedly covering terrain, namely by removing old trails. A large team of simulated ant robots covered a large terrain repeatedly over long periods of time without any ant robots getting stuck. We are now working on demonstrating the advantages of teams of ant robots on physical ant robots. The purpose of this article was to demonstrate the robustness of our minimalistic ant robots despite their limited ant-coverage hardware and simplistic ant-coverage software. We are now working on ant-coverage software that decreases the cover time of our ant robots even more while continuing to let them cover closed terrain robustly without knowing where they are. We are also working on comparing our ant robots to other coverage algorithms, including more traditional ones.

ACKNOWLEDGMENTS

We thank Ashwin Ram for making his hardware available to us. The Intelligent Decision-Making Group is partly supported by NSF awards under contracts IIS-9984827, IIS-0098807, and ITR/AP-0113881 as well as an IBM faculty partnership award. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations and agencies or the U.S. government.

X. REFERENCES

[1] F. Adler and D. Gordon. Information collection and spread by networks of patrolling ants. *The American Naturalist*, 140(3):373–400, 1992.

[2] R. Arkin. *Behavior-Based Robotics*. MIT Press, 1998.

[3] T. Balch and R. Arkin. Avoiding the past: A simple, but effective strategy for reactive navigation. In *International Conference on Robotics and Automation*, pages 678–685, 1993.

[4] T. Balch and A. Ram. Integrating robotics research with JavaBots. In *Proceedings of the AAAI Spring Symposium*, 1998.

[5] S. Koenig and R.G. Simmons. Xavier: A robot navigation architecture based on partially observable Markov decision process models. In D. Kortenkamp, R. Bonasso, and R. Murphy, editors, *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, pages 91–122. MIT Press, 1998.

[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] R. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2-3):189–211, 1990.

[8] D. Lambrinos, R. Möller, R. Labhart, R. Pfeifer, and R. Wehner. A mobile robot employing insect strategies for navigation. *Robotics and Autonomous Systems*, 30:39–64, 2000.

[9] D. Payton, M. Daily, B. Hoff, M. Howard, and C. Lee. Autonomy-oriented computation in pheromone robotics. In *Proceedings of the Autonomous Agents Workshop on Autonomy Oriented Computation*, 2001.

[10] A. Pirzadeh and W. Snyder. A unified solution to coverage and search in explored and unexplored terrains using indirect control. In *Proceedings of the International Conference on Robotics and Automation*, pages 2113–2119, 1990.

[11] R. Russell. *Odour Sensing for Mobile Robots*. World Scientific, 1999.

[12] R. Sharpe and B. Webb. Simulated and situated models of chemical trail following in ants. In *Proceedings of the International Conference on Simulation of Adaptive Behavior*, pages 195–204, 1998.

[13] S. Thrun. Probabilistic algorithms in robotics. *Artificial Intelligence Magazine*, 21(4):93–109, 2000.

[14] R. Vaughan, K. Stoey, G. Sukhatme, and M. Mataric. Whistling in the dark: Cooperative trail following in uncertainty localization space. In *Proceedings of the International Conference on Autonomous Agents*, pages 187–194, 2000.

[15] I. Wagner, M. Lindenbaum, and A. Bruckstein. Efficiently searching a dynamic graph by a smell-oriented vertex process. *Annals of Mathematics and Artificial Intelligence*, 24:211–223, 1998.

[16] I. Wagner, M. Lindenbaum, and A. Bruckstein. Distributed covering by ant-robots using evaporating traces. *IEEE Transactions on Robotics and Automation*, 15(5):918–933, 1999.

[17] I. Wagner, M. Lindenbaum, and A. Bruckstein. MAC vs. PC: Determinism and randomness as complementary approaches to robotic exploration of continuous unknown domains. *International Journal of Robotics Research*, 19(1):12–31, 2000.