Towards Robust On-line Multi-Robot Coverage

Noam Hazon, Fabrizio Mieli and Gal A. Kaminka*
The MAVERICK Group, Computer Science Department
Bar Ilan University, Israel
{haoznn,mielif,galk}@cs.biu.ac.il

***Abstract*— Area coverage is an important task for mobile robots, with many real-world applications. In many cases, the coverage has to be completed without the use of a map or any apriori knowledge about the area, a process referred-to as on-line coverage. Previous investigations of multi-robot on-line coverage focused on the improved efficiency gained from the use of multiple robots, but did not formally addressed the potential for greater robustness. We present a novel multi-robot on-line coverage algorithm, based on approximate cell decomposition. We analytically show that the algorithm is complete and robust, in that as long as a single robot is able to move, the coverage will be completed. We analyze the assumptions underlying the algorithm requirements and present a number of techniques for executing it in real robots. We show empirical coverage-time results of running the algorithm in two different environments and several group sizes.**

I. Introduction

Area coverage is an important task for mobile robots, with many real-world applications. In this task, a single or multiple robots are given a bounded work-area, possibly containing obstacles. Each robot is assumed to have an associated *tool* of a given shape —often corresponding to the robot's relevant sensors' and/or actuator range— that must visit every point within the work-area [6]. Since the tool size is typically smaller than the work-area, the robot must find a path that will take the tool over the entire work-area.

Many coverage applications must utilize *on-line* coverage algorithms [4], [9], [10]. Here, the robots cannot rely on apriori knowledge of the work-area, and must construct their movement trajectories step-by-step, addressing discovered obstacles (and/or collisions, in the case of multiple robots) as they move. This is in contrast to *off-line* coverage, where the robots are given a map of the work-area, and can therefore plan their paths ahead of deployment [3].

We focus in this paper on on-line coverage by multiple robots. Previous work has often pointed out that one advantage of using multiple robots for coverage is the potential for more efficient coverage [3]. However, another potential advantage of using multiple robots is that they may offer greater *robustness*: Even if one robot fails catastrophically, others may take over its coverage subtask. Unfortunately, this important capability has been neglected in existing work on on-line algorithms.

We present a *guaranteed robust* multi-robot on-line coverage algorithm. The algorithm is based on the use of spanning tree coverage paths [6]. It runs in a distributed fashion, using communications to alert robots to the positions of their peers. Each robot works within a dynamically-growing portion of the work-area, constructing a local spanning-tree covering this portion, as it moves. It maintains knowledge of where this spanning-tree can connect with those of others, and selects connections that will allow it to take over the local spanning trees of others, should they fail.

We also address the challenge of using the robust on-line multi-robot coverage algorithm with physical vacuum cleaning robots. We present techniques useful in approximating the assumptions required by STC algorithms (e.g., known positions, within an agreed-upon coordinate system). We show the effectiveness of our implemented algorithm in extensive experiments.

II. Background

Recent years are seeing much interest in multi-robot coverage algorithms, thanks to two key features made possible by using multiple robots: (i) robustness in face of single-robot catastrophic failures, and (ii) enhanced productivity, thanks to the parallelization of sub-tasks.

Choset [3] provides a survey of coverage algorithms, which distinguishes between *off-line* algorithms, in which a map of the work-area is given to the robots in advance, and *on-line* algorithms, in which no map is given. The survey further distinguishes between *Approximate cellular decomposition*, where the free space is approximately covered by a grid of equally-shaped cells, and *exact decomposition*, where the free space is decomposed to a set of regions, whose union fills the entire area exactly.

We focus in this paper on an on-line multi-robot coverage algorithm based on an approximate cellular decomposition. Previous work by Gabriely and Rimon [6] developed online and offline STC (Spanning-Tree Coverage) algorithms, but only for a single robot. Our own previous work introduced multi-robot coverage algorithms (MSTC—Multi-robot Spanning Tree Coverage), but only for offline usage [8]. Acar and Choset [1] presented a robust on-line single robot coverage algorithm while their robustness quality is the ability to filter bad sensors readings.

There have been additional investigations of on-line multi-robot coverage, but these do not guarantee a complete coverage if one of the robots failed. Wagner et al. [14] proposes a series of multi-robots ant-based algorithms which use approximate cellular decomposition. The algorithms involve simulated pheromones for communications. Some of these algorithms solve only the discrete coverage problem and the others can not guarantee robustness due to their heuristic nature.

*This work is partially supported by ISF Grant #1211/04.

Rekleitis et al. [11] uses two robots in on-line settings, using a visibility graph-like decomposition (sort of exact cellular decomposition). The algorithm uses the robots as beacons to eliminate odometry errors, but does not address catastrophic failures (i.e., when a robot dies). In a more recent article, Rekleitis et al. [12] extends the Boustrophedon approach [3] to a multi-robot version. Their algorithm also operates under the restriction that communication between two robots is available only when they are within line of sight of each other, but has many points of failure, i.e., it could stop functioning if one of the key robots fails.

Butler et al. [2] proposed a sensor-based multi-robot coverage, in a rectilinear environment, which based on the exact cellular decomposition. They do not prove their robustness, and the robots could cover the same area many times.

III. An On-line MSTC Algorithm

We focus in this paper on the on-line coverage case, in which the robots do not have apriori knowledge of the work-area, i.e., the exact work-area boundaries and all the obstacles locations (which are assumed to be static). The robots only know their absolute initial positions. Each robot has an associated square-shaped tool of size D. The objective is to cover the work-area using this tool. In real-world applications, the tool may correspond to sensors that must be swept through the work-area to detect a feature of interest, and the size D may be determined by the effective range of the sensors. Or, in vacuum cleaning application, the tool may correspond to the opening of the vacuum itself, typically underneath the robot. As with previous work [6], [8], we assume robots can move (with the tool) in the four basic directions (up/down, left/right), and can locate themselves within the work-area to within a D-size cell.

We divide the area into square cells of size $4D$, each one consists of four (4) sub-cells of size D. Denote the number of cells in the grid by N, and denote the number of sub-cells by n,i.e., $N = 4n$. The area occupancy in the beginning is unknown so every cell is initially considered to be empty.

The starting point is the work-area and k robots with their absolute initial positions: $A_0, \ldots, A_{k-1}$. The initial position of every robot is assumed to be in an obstacle-free cell, and the robot should know its position. Indeed, one assumption the algorithm makes—as previous work [6], [8] does—is that robots can locate themselves within an agreed-upon grid decomposition of the work area. In practice, of course, this assumption is not necessarily satisfied. Section IV below discusses methods for approximating this assumption in practice, which we utilize in our work with physical robots.

We seek algorithms that are *complete*, *non-redundant*, and *robust*. An algorithm is complete if, for k robots, it produces paths for each robot, such that the union of all k paths complete covers the work area. An algorithm is non-redundant if it does not cover the same place more than one time. The robustness criteria ensures that as long as one robot is still alive, the coverage will be completed.

The algorithms below are run in a distributed fashion, and generate on-line coverage that is complete, non-redundant, and robust. Each robot runs the initialization algorithm first (Algorithm 1), and then executes (in parallel to its peers) an instance of the ORMSTC (On-line Robust Multi-robot STC, Algorithm 2). Each ORMSTC instance generates a path for its controlled robot on-line, one step at a time. It is the union of these paths that is guaranteed to be complete, non-redundant, and robust.

We begin by describing Algorithm 1. The initialization procedure constructs the agreed-upon coordinate system underlying the grid work area. It then allows each robot to locate itself within the grid, and update its peers on the initial position of each robot.

Algorithm 1 On-line MSTC initialization()

```
1: Decompose the working area into $2D \times 2D$ cells (grid),
   agreed among all the robots.
2: Decompose each $2D \times 2D$ cell into 4 sub-cells (size $D$)
3: $i \leftarrow$ my robot id ID
4: if $A_i \neq$ the middle of a sub-cell then
5:     $s_i \leftarrow$ the closest sub-cell
6:     Move to $s_i$
7: else
8:     $s_i \leftarrow A_i$
9: $S_i \leftarrow$ the cell that contains $s_i$
10: Announce $S_i$ as your starting location to the other robots
11: Receive $S_j$, where $j \neq i$, starting cells of other robots
12: Update map with $S_0, \ldots, S_{k-1}$
13: Initialize $connection[0 \ldots k-1][0,1] \leftarrow null$
```

Once the grid is constructed and robots know their initial positions, Algorithm ORMSTC (Algorithm 2) (executed in a distributed fashion by all robots) carries out the coverage. This recursive algorithm receives two parameters: X, the new cell that the robot just entered, and W, the old cell from which the robot has arrived. We denote a cell with an obstacle in one (or more) of its 4 sub-cells, or one that contains the robot's own spanning tree edge, as *a blocking* cell. In the first recursive call to the algorithm, the argument X is the robot's starting cell S_i. W is chosen such that it is consistent with future calls to the algorithm—closest to s_i (Figure 1).

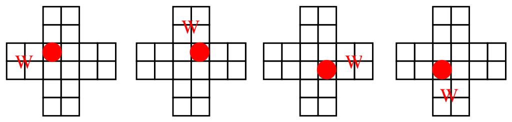

Fig. 1. The 4 possible initial positions (marked with a dot), and their respective recommended W.

The idea behind algorithm 2 is that each robot gradually builds a local spanning tree of uncovered cells that it discovers, while tracking the state of any of its peers whose path it has met. The spanning tree is built by a depth-first-like procedure: Scan for a non-occupied neighboring cell (Lines 1–2), build a tree edge to it (Line 15), enter it (Line 16) and continue recursively with this cell (Line 17). If there is no free cell, the

Algorithm 2 ORMSTC(W, X)

```
1: $N_{1..4} \leftarrow X$'s neighboring cells in clockwise order, ending with $W = N_4$
2: for $i \leftarrow 1$ to 3 do
3:   if $N_i$ = blocking cell then
4:     continue to the next $i$
5:   if $N_i$ has a tree edge of robot $j \neq i$ then
6:     check whether robot $j$ is alive
7:     if robot $j$ is alive then
8:       if $connection[j][0] = null$ then
9:         $connection[j][0] \leftarrow$ the edge from $X$ to $N_i$
10:      $connection[j][1] \leftarrow$ the edge from $X$ to $N_i$
11:      continue to the next $i$
12:    else {robot $j$ is not alive}
13:      remove robot $j$ from connections array and broadcast it
14:      mark $j$'s cells as empty on the map and broadcast it
15:  construct a tree edge from $X$ to $N_i$ and broadcast it
16:  move to a sub-cell of $N_i$ by following the right side of the tree edge
17:  execute ORMSTC($X$, $N_i$)
18: if $X \neq S_i$ then
19:   move back from $X$ to $W$ along the right side of the tree edge
20:   return from recursive call
21: if $W \neq$ blocking cell then
22:   execute ORMSTC($X$, $W$)
23: move to sub-cell $s_i$ along the right side of the tree edge
24: broadcast completion of work
25: while not all the robots announced completion do
26:   if $\exists j$, s.t. $connection[j][0] \neq null$ and robot $j$ is not alive then
27:     mark $j$'s cells as empty on the map and broadcast it
28:     broadcast withdrawal of completion
29:     decide which connection: $connection[j][0]$ or $connection[j][1]$ is closer to $s_i$ when moving in clockwise or counter-clockwise direction along your tree edges
30:     move to this connection in the appropriate direction
31:     $X \leftarrow$ your connection cell
32:     $Y \leftarrow$ robot $j$'s connection cell
33:     remove robot $j$ from connections array and broadcast it
34:     construct a tree edge from $X$ to $Y$ and broadcast it
35:     move to a sub-cell of $Y$ by following the right-side of the tree edges
36:     execute ORMSTC($X$, $Y$)
```

robot goes back along its local spanning tree to the previous covered cell, exiting the recursion (Lines 18–20).

During this gradually-expanding coverage process, the first time a robot i meets a cell with robot j's tree-edge ($i \neq j$), it examines its peer's state (Lines 5–6). If robot j is still alive, robot i saves the edge which connects its tree to robot j's tree as $connection[j][0]$ (Lines 7–9). From this point on, robot i will update $connection[j][1]$ to save the last edge which connects its tree to robot j's tree, i.e., whenever robot i meets a cell with robot j's tree edge (Lines 10–11).

If, during this phase, robot i discovers that robot j is not alive anymore, it announces to the other robots that robot j is dead. Then all robots delete the entries for robot j from their *connection* arrays, and the cells which robot j was responsible for are marked empty (Lines 12–14). Robot i and the other robots can now build their spanning tree edges to these cells and cover them (see below for a discussion of the case where two robots want to enter the same cell).

When a robot has no neighboring cells to cover, and it is back in its initial position, it makes sure that W (this is the initial W given as input) is covered (Lines 21–22). Then the robot finishes covering its starting cell and announces to the other robots that it has completed its work (Lines 23–24).

However, the coverage process is not completed until all the robots announce completion of their work. Until then, a robot who finishes its work monitors the state of all the robots for whom it has a non-empty *connection* entry (Lines 25–26). If such a robot j is not alive, the robot sets all cells assigned to j in the map to empty, and updates the other robots (Line 27). It then turns to cover robot j's cells, thus withdrawing its previously-declared completion of its work (Line 28). The robot has two possibilities to reach robot j's cells: Along the left side of its spanning tree edges, till it reaches $connection[j][0]$, the first connection edge between it and robot j's path; or in an opposite direction along the right side of the spanning tree edges, till it reaches $connection[j][1]$, the last connection edge between it and robot j. The robot chooses the best option and moves to the chosen connection edge (Lines 29–30). Now it can delete robot j from the connection array (line 31), and continue to construct the spanning tree edges for the new cells by recursively calling the algorithm (Lines 32–36).

Algorithm 2 makes several assumptions about the robots' capabilities. First, in lines 1–2, each robot explores its three neighboring cells. To do this, each robot must have the ability to sense and determine if its three neighboring cells are free from obstacles. If the cell is partially occupied with an obstacle it will not be covered. Second, the algorithm requires reliable communication. Each informative message that a robot receives (a cell that is now occupied with a tree edge, a dead robot, etc.) updates the map and overall world state (in the memory of its peers). Obviously, there is also an assumption here that robots are cooperative, in that when a robot is asked if it is alive, it broadcasts truthfully if it can.

In lines 15 and 34 the robot constructs a spanning-tree edge. A synchronization problem could occur if more than one robot wants to construct a tree edge in the same cell. This can be solved by any synchronization protocol. For instance, we can require robots to notify the others whenever they wish to construct an edge to a cell Q. If a conflict over Q is detected, robots can use their distinct IDs to select who will construct the edge (e.g., highest ID), or they may allow the robot with the smallest number of covered cells to go first (intuitively, this is the most underutilized robot). The other robots treat Q as a cell with another robot's spanning tree edge and continue with the algorithm.

We prove the completeness of the ORMSTC algorithm (Theorem 3.1). Each robot constructs its own spanning tree and circumnavigates it to produce a closed curve which visits all the sub-cells of the tree cells. Completeness is achieved by ensuring that every cell (within the area boundaries) will have a tree edge connection from one of the trees.

Theorem 3.1 (Completeness): Given a grid work-area W, and k robots, Algorithm 2 generates k paths k_i, such that $\bigcup_i k_i = W$, i.e., the paths cover every cell within the the work-area.

Proof: By induction on the number of robots k.

Induction Base ($k = 1$). with only one robot, ORMSTC operates exactly like the On-line STC Algorithm which was proven to be complete (Lemma 3.3 in [6]).

Induction Step. Suppose it is known that $k - 1$ robots completely cover the area. We will prove it for k robots. Without loss of generality, let us consider robot i. Executing ORMSTC, i will build its local spanning tree edges, and generate a path to cover some cells. The other robots treat these cells as occupied, exactly as if they were filled with obstacles. Therefore all the other cells will be part of $k - 1$ paths and covered by the $k-1$ robots, according the induction relaxation. Robot i treat all the cells of the other $k-1$ robots as occupied cells, so it will completely cover its cells according to the induction base case. □

We now turn to examining ORMSTC with respect to coverage optimality. Previous work has discussed several optimization criteria [8], one of which is *redundancy*, the number of times a sub-cell is visited.

ORMSTC can be shown to be non-redundant. Theorem 3.2 below guarantees that the robots visit all the cells only once (if no failure has occurred—see below for a discussion of robustness). This guarantee is in fact a feature of many spanning-tree coverage algorithms, as circumnavigating a tree produce a closed curve which visits all the sub-cells exactly one time [6].

Theorem 3.2 (Non-Redundancy): If all robots use Algorithm 2, and no robot fails, no cell is visited more than once.

Proof: If no robot fails, then each robot only covers the cells for which it builds a tree edge. If there is already a tree edge to a cell, the robot will not enter it (Line 5). Thus every cell is covered only by a single robot. Since robots never backtrack, every point is only covered once. □

As key motivation for using multiple robots comes from robustness concerns, we prove that Algorithm 2 above is robust to catastrophic failures, where robots fail and can no longer move. Lines 12–14 and 25–36 guarantee the robustness. If one robot fails, there is always at least one robot that will detect it and will take the responsibility to cover its section (see below for formal proof). Conflicts over empty cells are handled as described above.

Theorem 3.3 (Robustness): Algorithm 2 guarantees that the coverage will be completed in finite time even with up to $k-1$ robots failing.

Proof: Based on the completeness theorem (Theorem 3.1), any number of robots can cover the work area. Thus if one or more robots fail, all the cells that were not occupied by tree edges of the failing robots and are accessible to other live robots will be covered. So all we have to prove is that cells with tree edges of a dead robot, or cells which are accessible only to a robot that has died will be covered by another robot. It can be possible to have such cells because of the work area structure, or because the robot can create a line of covered cells which blocks the access of other robots to a group of free cells.

Cells with existing tree edges of a robot are treated by the other robots as cells with obstacles. According the completeness theorem, there is at least one robot that will cover a neighboring cell of one of these cells, thus will have a connection to this cell. There are two possible cases:

1) A robot failed before a robot that has a connection with it reached the connection. In this case, lines 13–14 ensures that the dead robot's covered cells will be declared free so they will be covered by other robots.
2) A robot fails after all the robots that have a connection with it reached the connection. In this case, lines 27 and 33 apply, to ensure that the robot's covered cells will be declared free so they will be covered by other robots.

In both possible cases, the freeing of cells previously-covered by the dead robot also makes any cell which was only accessible to the dead robot accessible to others. Based on the completeness theorem, at least one other robot is guaranteed to reach all these cells. Thus the algorithm is proved robust. □

IV. From Theory to Practice

In real-world settings, some of the assumptions underlying ORMSTC can not be satisfied with certainty, and can only be approximated. This section examines methods useful for such approximations, and their instantiations with physical robots.

In particular, we have implemented the ORMSTC algorithms for controlling multiple vacuum cleaning robots, the RV-400 manufactured by Friendly Robotics [5]. Each commercial robot was modified to be controlled by an small Linux-running computer, sitting on top of it. A generic interface driver for the RV-400 robot was built in Player [7], and a client program was built to control it. Each robot has several forward-looking sonar distance sensors, as well as sideways sonars. One robot is shown Figure 2.

The ORMSTC algorithm (indeed, many of the STC algorithms) make several assumptions. First, there are assumptions as to the work area being provided as input. ORMSTC

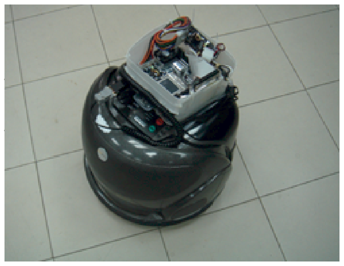
Fig. 2. RV-400 robot used in initial experiments.

assumes, for instance, that the work-area has known bounds, and that it is divided into a grid that is known by all robots (i.e., all robots have the same division). ORMSTC assumes robots can communicate reliably, and locate themselves within a global coordinate system. Finally, ORMSTC makes assumptions about the sensory information available to the robots. In particular, ORMSTC makes the assumption that each robot can sense obstacles within the front, left, and right $4D$ cells.

One challenging assumption is that of a global coordinate system that all robots can locate themselves within. In outdoor environments, a GPS signal may in principle be used for such purposes (note that the position only has to be known within the resolution of a sub-cell). However, in circumstances where a global location sensor (such as the GPS) is unavailable, a different approach is needed. In particular, this is true in the indoor environments in which the vacuum cleaning RV-400 is to operate.

For the purposes of the experiments, we have settled on letting the robots know their *initial* location on an arbitrary global coordinate system. Once robots began to move, however, they relied solely on their odometry measurements to position themselves. In the future, we hope to experiments with alternative approaches.

One advantage of ORMSTC in this regard is that its movements are limited to turns of $90°$ left or right, and to moving forward a fixed distance. This offers an opportunity for both reducing errors by calibration for odometry errors specific to this limited range of movements, and by resetting after each step, thus avoiding accumulative errors. Indeed, this was the approach taken in the experiments (see next section).

Given a global coordinate system, ORMSTC also requires robots to agree on how to divide up the work-area into a grid. This agreement is critical: Differences in the division may cause grids created by different robots to be mis-aligned, or overlap. To do this, the bounds of the grid have to be known, in principle. Once the bounds are known, the robots only have to decide on the origin point for the approximate cell decomposition.

Here again a number of approximating solutions were found to be useful. First, one can have the robots use a dynamic work-area. During the initialization phase, the robots determines the maximal distances, X_{max} and Y_{max} (along the X- and Y- axes, respectively), over all pairs of robots. They then build a temporary rectangular work-area around them, with sides greater or equal to X_{max}, Y_{max}. As the robots move about, they will push the boundaries of the work-area into newly discovered empty cells that lie beyond the bounds, or they will encounter the real bounds of the work area, which will be regarded as obstacles. A related approximation is to provide the robots with an initial work-area that is known to be too big, and allow the robots to discover the actual bounds. This was the technique we utilized.

Robustness against collisions is an additional concern in real-world situations. Normally, as each robot only covers the path along its own tree, Theorem 3.2 guarantees that no collisions take place. This separation between the paths of different robots decreases the chance of collisions. In practice, localization, movement errors, and the way the grid is constructed may cause the robot to move away from its assigned path, and thus risk collision. We utilized our bumps sensors to cope with this problem as they are often used as a key signal in vacuum-cleaning robots. Our heuristic is to simply respond to a bump by moving back a little, waiting for a random (short) period of time trying again. If bumps occur three times in a row in the same location, the location is marked as a bound or obstacle. A more complicated solution which requires more communication is to coordinate between the robots that have adjacent tree edges when a collision is likely to occur.

A final challenge was offered by the robots' limited sensor range. The robot is equipped with ten sonar sensors which are not capable of sensing all three neighboring cells of the robot cell at the same time as described in the algorithm requirements before. We solved this problem by dividing the original sensing and movement phases to three steps. The robot first senses its first cell by turning its sensors towards it. If it is empty, it continues with the regular algorithm flow. If not, in moves forward to be as close as possible to the border of the next require-sensing cell and only then it turns to sense it and continues with the algorithm. The same procedure is performed to the third neighboring cell. Although it slowed down the algorithm performance this fix enabled us to run the algorithm in the simulation with the robots constraints, so it can be applied also to run the algorithm on different real robots with limited sensors.

V. Experimental results

We conducted systematic experiments with our implementation of the ORMSTC algorithm, to measure its effectiveness in practice with the RV400 robot. The experiments were conducted using the Player/Stage software package [7], a popular and practical development tool for real robots. Initial experiments were carried out with physical RV400 robots, to test the accuracy of the simulation environment used. However, to measure the coverage results accurately, the experiments below were run in the simulation environment. Figure 3 shows a screen shot of running example with six robots in the one of the simulated environments used in the experiments.

In the experiment, we focused on demonstrating that the ORMSTC algorithm—and our implementation of it for real robots—indeed manages to effectively use multiple robots in coverage. We ran our algorithm with 2,4,6,8 and 10 robots.

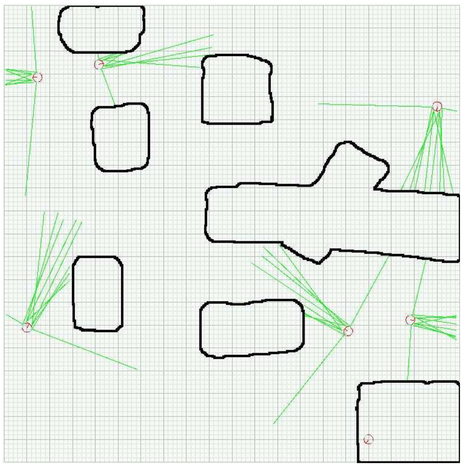

Fig. 3. Simulation screen shot of six robots covering the *Cave* environment

Each team was tested on two different environments. The *Cave* environment had irregularly-shaped obstacles, but was relative open. The *Room* environment had many rectangular obstacles, and represents a typical indoor office room. For each team size and environment type, 10 trials were run. The initial positions were randomly selected.

The results are shown in Figure 4. The X-axis measures the number of robots in the group. The Y-axis measures the coverage time. The two curves represent the two different environments. Every data point represents the average ten trials, and the horizontal line at each point shows the standard deviation in each direction.

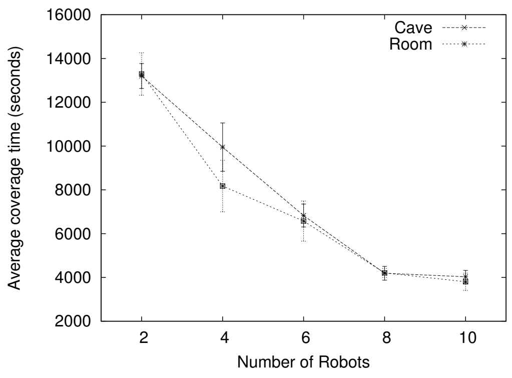

Fig. 4. Overall coverage time

The results show that in both environments, coverage time decreases in general when increasing the group size. However, we can also see that the marginal coverage decreases with the addition of new members. This is a well-known phenomenon (in economics, but also in robotics [13]). It is due to the overhead imposed on a bigger group of robots, in collisions avoidance and communication load. The overhead cost can be also seen when comparing the two overage times of the two environments. Although the indoor environment is smaller, the coverage time is almost the same because there are more obstacles and doors to pass and there is a greater chance of collision with walls or other robots.

VI. Conclusion and future work

Many real-world coverage applications require multiple robots to completely cover a given work-area, with no apriori knowledge of the area. We presented the ORMSTC, a multi-robot coverage algorithm which is able to cover an unknown environment. We analytically showed that ORMSTC algorithm is complete and robust in face of catastrophic robot failures. As there is always a gap between theory and practice, we analyzed the assumptions underlying the algorithmic requirements. We discuss various approximation techniques for these requirements, to allow the algorithm to work in real world situations. Based on early trials with real-robots, we conducted systematic experiments with our implementation, to measure the ORMSTC's effectiveness in practice. The results show that the algorithm works well in different environments and group sizes. For future work, We intend to improve the algorithm to generate paths with less turns and to cover also cells which are partially covered by obstacles.

Acknowledgments. We thank Noa Agmon, Yair Hershkovitz, Gilad Asharov, Asa Yohai, and Ido Ikar for their assistance. Special thanks to K. Ushi and Shira Hazon.

References

[1] E. U. Acar and H. Choset. Robust sensor-based coverage of unstructured environments. In *International Conference on Intelligent Robots and Systems*, pages 61–68, Maui, Hawaii, USA, 2001.

[2] Z. Butler, A. Rizzi, and R. L. Hollis. Complete distributed coverage of rectilinear environments. In *Workshop on the Algorithmic Foundations of Robotics*, March 2000.

[3] H. Choset. Coverage for robotics—a survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31:113–126, 2001.

[4] J. Colegrave and A. Branch. A case study of autonomous household vacuum cleaner. In *AIAA/NASA CIRFFSS*, 1994.

[5] Friendly Robotics®, Ltd. Friendly robotics vacuum cleaner. http://www.friendlyrobotics.com/friendly_vac/.

[6] Y. Gabriely and E. Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. *Annals of Mathematics and Artificial Intelligence*, 31:77–98, 2001.

[7] B. P. Gerkey, R. T. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the International Conference on Advanced Robotics*, pages 317–323, Coimbra, Portugal, Jul 2003.

[8] N. Hazon and G. A. Kaminka. Redundancy, efficiency, and robustness in multi-robot coverage. In *ICRA-05*, 2005.

[9] S. Hedberg. Robots cleaning up hazardous waste. *AI Expert*, pages 20–24, May 1995.

[10] Y. Huang, Z. Cao, and E. Hall. Region filling operations for mobile robot using computer graphics. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 1607–1614, 1986.

[11] I. Rekleitis, G. Dudek, and E. Milios. Multi-robot exploration of an unknown environment, efficiently reducing the odometry error. In *International Joint Conference in Artificial Intelligence (IJCAI)*, volume 2, pages 1340–1345, Nagoya, Japan, August 1997. Morgan Kaufmann Publishers, Inc.

[12] I. Rekleitis, V. Lee-Shue, A. P. New, and H. Choset. Limited communication, multi-robot team based coverage. In *IEEE International Conference on Robotics and Automation*, pages 3462–3468, New Orleasn, LA, April 2004.

[13] A. Rosenfeld, G. A. Kaminka, and S. Kraus. Adaptive robot coordination using interference metrics. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 910–916, 2004.

[14] 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.