Cooperative Coverage of Rectilinear Environments

Zack J. Butler, Alfred A. Rizzi, and Ralph L. Hollis
Robotics Institute, Carnegie Mellon University
{zackb, arizzi, rhollis}@ri.cmu.edu

Abstract

A distributed cooperative coverage algorithm DC_R is presented which is derived from an earlier complete single-robot algorithm, CC_R. DC_R executes independently on each robot in a team where the individual robots do not know the initial locations of their peers and applies to systems of robots operating in a rectilinear environment that use only intrinsic contact sensing to determine the boundaries of the environment. Due to the reactive nature of CC_R, the natural extension to DC_R preserves the completeness properties of the single-robot algorithm and the outline of a completeness proof of DC_R is also presented. DC_R has been implemented in simulation, and directions for future work are presented which will make the algorithm more suited to physical robot systems.

1 Introduction

The ability to completely cover an environment, and to plan a path to do so, is a valuable capability for robot systems. Tasks such as mine detection, floor cleaning, and others are essentially coverage tasks, in which a robot must pass a sensor or effector over every point in its environment. Often the robot will not have *a priori* knowledge of its environment, and so will need to plan its path as it travels to ensure complete coverage. This task is called *sensor-based coverage*. Like coverage of known environments, its implementation depends on the capabilities of the robot performing the coverage task, and a variety of solutions have been proposed. In addition, operations such as mine detection may involve a team of robots, and it would be desirable for them to work together to efficiently produce complete coverage.

In general, for a single robot, sensor-based coverage is begun by assuming the environment to have a specific simple shape. The robot then executes a simple coverage path until it discovers evidence that contradicts the initial assumption, at which point one of several strategies is used to ensure coverage on all sides of the newly discovered obstacle. An algorithm presented by Lumelsky et al. in [1] and extended in [2] produces coverage of $\mathcal{C}^2$ environments for robots with finite nonzero sensing radius. It is a scripted algorithm that recursively builds a subroutine stack to ensure covering all areas of the environment and does not explicitly build a map. In contrast, work presented by Acar in [3] based on a planned coverage strategy outlined in [4] incrementally builds a graph based on a cellular decomposition of the environment. This introduces an element of planning into the coverage process — when a cell has been covered, the robot uses the structure of the graph to plan a path to an unexplored area, and when the graph has no unexplored edges, coverage is complete. The cell decomposition approach of [4] also inspired the algorithm presented in [5], which in turn is the basis for the current work. In [5], an algorithm CC_R is presented for coverage of rectilinear environments using only contact sensing. An outline of the behavior of this algorithm is presented in Sec. 1.1.

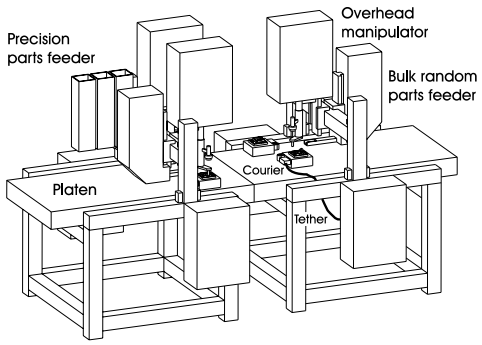

Figure 1: A small section of a minifactory, showing typical components.

In contrast to the more commonly studied coverage tasks mentioned above, the motivation for the current work comes from a manufacturing environment. The *minifactory*, an automated assembly system under development in the Microdynamic Systems Laboratory[1], is built within a framework that provides for rapid design, programming and deployment [6], and is targeted at assembly of small electromechanical products such

[1] http://www.cs.cmu.edu/~msl

as mobile telephones, disk drives and small medical devices. A minifactory, a small example of which can be seen in Fig. 1, consists mainly of overhead processors and *couriers*, small tethered robots that operate on a set of tileable *platens* which form the factory floor. The robots are computationally independent but use common network protocols and standardized algorithmic interfaces for ease of programming and to enable real-time cooperation to perform assembly tasks. In order to quickly deploy such a system and bring it to full production, the deployment process must be done without cumbersome manual calibration. The system will therefore need to be capable of self-calibration. To accomplish this, the couriers use upward-looking optical sensors to locate beacons on all overhead devices and intrinsic contact sensing to determine platen geometry. While calibration could take the form of simply verifying the location of all expected overhead devices and platens, a complete coverage of the factory area makes the calibration process more robust. In addition, having the couriers explicitly cooperate in this task would decrease the setup time, motivating the research presented here.

While the algorithm presented here is capable of minifactory calibration, it is also applicable to a somewhat greater variety of robot systems. Specifically, it applies to teams of rectangular robots with intrinsic contact sensing operating in a shared, connected rectilinear environment with finite boundary and area. In addition, the robots in the team will not know their relative initial positions or orientations, however, due to the structure of the environment, their orientation will be one of four distinct values (i.e. with axes aligned with the environment boundaries) and cannot change.

Most previous work with cooperative mobile robots, even where exploration is addressed, does not include complete coverage of an environment. Work that does has used a central controller deploying robots from a known location, which is not satisfactory for the minifactory problem. Gage's work [7] uses random walks by a large team with a common home position to generate probabilistically complete coverage. A fairly abstract algorithm presented by Rao et al. [8] uses a small team of point-sized robots with infinite range sensing to cooperatively build a visibility graph of a polygonal environment. In contrast, work by Rekleitis et al. [9] uses cooperating robots with mutual remote sensing abilities, but with explicit cooperation to reduce mapping errors rather than to increase efficiency.

1.1 Single-robot algorithm

The cooperative work presented in this paper is based on a single-robot coverage algorithm CC_R, presented in [5]. Using CC_R, a single robot with only intrinsic contact sensing can cover any finite rectilinear environment. Coverage is performed by incrementally constructing a cellular decomposition of the environment (C) and using only the structure of C to determine the coverage path. An example of the type of cell decomposition built by CC_R is shown in Fig. 2. Each cell is as wide as possible in x while having straight *floor* and *ceiling* (minimum and maximum y extents). The left and right sides are therefore at *interesting points*, which correspond to the x values of vertical boundary segments. The sides of each cell are represented by a list of *intervals*, which describe the locations of walls, cells, and placeholders adjacent to the cell. *Placeholders*, line segments that signify the entrance to an unexplored region, are also part of C.

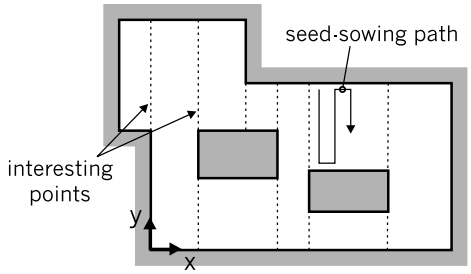

Figure 2: A cell decomposition as created by CC_R, also showing a portion of a seed-sowing path.

At each execution cycle of CC_R, the *map interpreter* (one half of the CC_R algorithm) first inspects C to determine the next appropriate straight-line motion to continue coverage. This motion is then executed without interruption from CC_R, stopping only at a collision or after an appropriately chosen distance (these occurrences are referred to as *coverage events*). After a coverage event has occurred, the *event handler* (the other half of CC_R) updates C appropriately given the type of event and the robot's position, p. C is then inspected again and a new motion is determined.

To cover each cell, the map interpreter generates a *seed-sowing* path as in Fig. 2, in which the robot travels along paths (called *strips*) which are parallel to its y axis and as far apart as the width of the robot. These continue until an interesting point is detected. The interesting point is then localized and the final edge of the cell is explored. This behavior is generated by a set of rules in the map interpreter, which, after each trajectory, are tested against C and p. The first applicable rule determines the next trajectory (both direction and distance, the distance chosen so that a change of direction would be required if that distance was traveled without collision). Rules for interesting

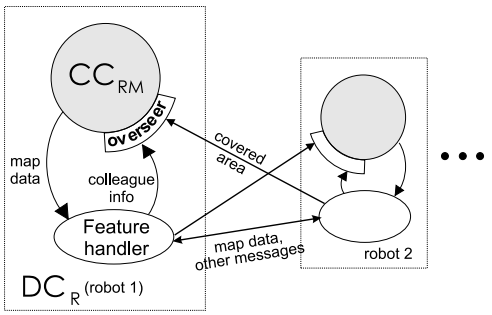

Figure 3: A schematic rendition of the algorithm DC_R, which is built around a slightly modified version of the single-robot algorithm CC_R, a copy of which is run independently by each robot performing coverage.

points are tested first, followed by the seed-sowing rule, followed by rules for when the cell containing p is complete. In this last case, the robot will return to any incomplete cell in C, otherwise a new cell will be created from an arbitrarily chosen placeholder.

Completeness of CC_R is proven in [5] by creating a finite state machine that represents all possible ways in which C can evolve under CC_R, and then showing that the FSM has no infinite loops and does not terminate until coverage is complete.

2 DC_R: Overview

To extend this coverage skill to a cooperative setting in which each robot is running independently, CC_R has been modified and enhanced to form a new algorithm DC_R (distributed coverage of rectilinear environments). DC_R, shown in schematic form in Fig. 3, includes a slightly modified version of CC_R (denoted CC_{RM}) along with additional components that provide cooperation with team members without impeding the progress of coverage. In general terms, DC_R operates by altering the cell decomposition, C, in real-time in response to messages from other robots. Since CC_{RM} (like CC_R) uses only C to plan coverage, it simply replans in the altered map without explicit awareness of the cooperation taking place.

CC_{RM} is augmented with two components that handle cooperation to form DC_R. One of these, the *feature handler*, extracts features of a pre-specified type from C and shares them with other robots in the team to try to form *colleague* relationships. Two robots are considered colleagues if they know the relative geometric transform between their coordinate systems. The other new component, the *overseer*, accepts incoming map data from all colleagues. Specifically, each robot sends to the other the geometry of each complete cell in its decomposition and the location of

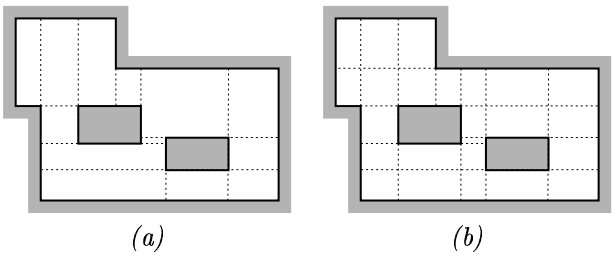

Figure 4: Examples of (a) a generic rectilinear decomposition and (b) the unique sweep-invariant decomposition of a rectilinear environment.

all beacons it has discovered. It is then the job of the overseer to add the covered area to C. This addition allows each robot to avoid covering area already covered by a colleague, but must be done in a way that leaves C in an admissible state for CC_{RM}.

2.1 Cell decompositions under DC_R

The most notable effect on CC_{RM} due to cooperation is that the class of cell decompositions encountered is more general than that created by a single robot. Since the robots in the team may have different orientations, the interesting points that define cells (and therefore which walls constitute floors and ceilings) are not necessarily the same from one robot to the next. A cell obtained from a colleague may therefore be different from any that would appear in the decomposition that the robot would create on its own.

The class of decompositions that can be constructed under DC_R are referred to here as *generic rectilinear* decompositions (GRDs), an example of which can be seen in Fig. 4a. Cells in these decompositions do not necessarily border walls on their floor and ceiling, however, each cell is still rectangular and is a superset of cells of the *sweep-invariant* decomposition (SID) of the environment. In the SID, the boundaries between cells are determined by walls and obstacle edges in both x and y. An example of this type of decomposition is shown in Fig. 4b. It should be noted that while the SID is unique for a given environment, a GRD developed under DC_R will depend on the orientation of the robot and the progress of cooperation. Also, during the performance of DC_R, the cell decomposition may at times consist of multiple disconnected components. This is because robots only share completed cells, but may meet in an area that neither has completed.

2.2 Changes to CC_R

In order for the proof of CC_R to be easily carried over to CC_{RM}, changes were implemented so that CC_{RM} would produce essentially the same behavior as CC_R over the larger class of decompositions. The

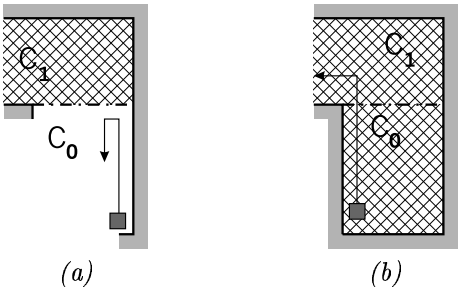

Figure 5: The effects of an exploration boundary (dash-dot line) when the robot is in (a) an incomplete cell and (b) a complete cell.

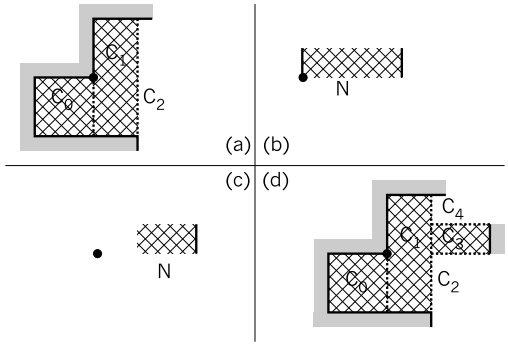

Figure 6: An example of adding new area by the overseer, in which the initial cell decomposition is depicted in Fig. 6a and the incoming cell N in Fig. 6b. The dot in each section of the figure represents a common real-world point.

most significant change required is due to the floors and ceilings of cells, and therefore the ends of seed-sowing strips, no longer always being at walls. In order for coverage to proceed correctly, virtual *exploration boundaries* are placed at the floor and ceiling of each cell where an environmental boundary does not exist. These virtual walls have the property of impeding the robot's progress only when the robot is in an incomplete cell. Therefore, when the robot is covering an incomplete cell that has a complete cell above or below it, the exploration boundary will act as a real wall to allow seed-sowing to continue as in CC_R, as shown in Fig. 5a. However, once that cell is complete, the exploration boundary effectively vanishes and so cannot hinder any planned path from one cell to another, as shown in Fig. 5b. The exploration boundaries are created by the overseer as described below and can be implemented in a variety of ways depending on whether DC_R is being run in simulation or on actual robots, and if the latter, on the robots' low-level functionality.

The changes required to the event handler are then simply to add the ability to create and maintain intervals on the floor and ceiling of each cell during coverage. Also, while the map interpreter's rules for coverage within a cell remain unchanged, the handling of C when the current cell is complete is altered in two ways. First of all, GRDs may have placeholders aligned with the x axis as well as y, and the map interpreter must be able to correctly instantiate incomplete cells from such placeholders. Also, if C becomes disconnected at any time, a path cannot necessarily be planned to any placeholder in C. However, there will always be at least one placeholder that can be reached, as shown in Sec. 3. The path planner simply searches the list of placeholders for one that a path can be planned to and selects that as its destination.

2.3 Additions to CC_{RM}

With CC_{RM} in place, the other two components of DC_R can then be added. Of the two, the feature handler has much more flexibility in its design since it does not directly affect any data structures used by CC_{RM}. The feature handler uses the data in C to generate colleague relationships between robots, however, it does not alter C. Therefore, the specific features and algorithms used by the feature handler can change from one system to the next without affecting CC_{RM}.

The overseer is responsible for interpreting all incoming data and integrating it into the structures used by CC_{RM}. Since it actively modifies these structures, its design is much more restricted. The first task for the overseer when new data arrive from a colleague is to convert these data (complete cells and beacon locations) from the colleague's coordinate system into its own. It then adds the beacons directly to its own list after checking for and removing duplicates. However, it must be more careful about adding the new cells to ensure that C remains a valid GRD. The new cells must not overlap any other cells, as required for a GRD, and their intervals must point to the correct cells (or new placeholders), so that path planning will work correctly. This addition is done in three stages, an example of which is shown in Fig. 6. In the first stage, the incoming cell N is intersected with all complete cells in C. Only the area of N that does not intersect with these cells is retained to be added to C, with N divided into multiple cells if necessary. In the example, a single area as shown in Fig. 6c is retained after intersection with cell C_1. In the second stage, incomplete cells in C are intersected with this remaining area, but in this case, the incomplete cells are reduced in area and split if necessary. This maximizes the amount of

complete area contained in C while also ensuring the creation of a valid GRD as described below. The resultant C for the example is shown in Fig. 6d. Finally, the intervals along the edges of what was N are each examined. Intervals that point to walls are retained as is. Otherwise, if the interval is adjacent to a cell in C, it is given that cell as its neighbor, and if not, a new placeholder is created for the interval to point to. In either of these cases, if the interval is along an horizontal edge, an exploration boundary is created along the line segment defined by the interval.

3 Completeness Proof Outline

To prove completeness for a coverage algorithm, it is necessary to show that the coverer will reach all points in the environment regardless of the form of the environment (within certain restrictions). For a cooperative algorithm, it is necessary to show that each robot will satisfy this requirement and also continue coverage while cooperating with its teammates, so that the team as a whole produces complete coverage. Due to space constraints, only a summary of the arguments for completeness of DC_R are presented here.

Proposition 1 *CC_{RM} inherits completeness from CC_R and produces complete coverage of any generic rectilinear decomposition C in the absence of cooperation that alters the robot's current cell.*

Since CC_{RM} was specifically designed to exhibit the same behavior as CC_R, the proof of the progress of coverage is virtually unchanged from that described in [5]. Essentially the states of the FSM that describe CC_R become more inclusive but not more numerous. Exploration boundaries ensure that any cell bounded by another cell on its floor and/or ceiling falls into the same state as it would if a wall was present instead of the other cell. The changes to the event handler and map interpreter described above ensure that all motion outcomes are handled as in CC_R. Therefore, CC_{RM} generates the same FSM for covering an individual cell as CC_R. Also, this FSM applies to any GRD containing the current cell, since the state is determined only by the current cell. The proof of CC_{RM} is completed by verifying the transitions that occur when the robot's current cell is complete. This requires additional analysis since C can now become disconnected.

To show that a correct path is always found in this instance, we define C_p as the component of C containing the robot's position p, and note that a path can always be planned from p to any cell or placeholder in C_p. If any incomplete cells exist, they must all be in C_p, because all cells in C that are not in C_p must have been contributed by other robots, and are therefore complete cells. If there are no incomplete cells in C, then there must be at least one placeholder adjacent to C_p. If this was not true, C_p would have a closed boundary and could never become attached to the remainder of C, violating the assumption of a connected environment.

It is also important to note that the decomposition created by CC_{RM} when working alone is a special case of a GRD (and the same decomposition that CC_R would generate), and therefore that DC_R is complete for the single-robot case.

Proposition 2 *The overseer always leaves (C, p) in an admissible state for CC_{RM}.*

Since the immediate behavior of CC_{RM} depends only on the structure of the robot's current cell, this proposition can be proven by showing first that the overseer produces a valid GRD (independent of the FSM of Proposition 1), and second that any alteration to the robot's current cell leaves the cell in a state contained in the FSM described by CC_{RM}.

The first statement is true if the cells added to (and changed in) C are supersets of SID cells and their boundary is correctly assigned. The area is correct if the edges of the new cell as well as any new or altered edges of incomplete cells are coincident with edges of the SID. This is shown by assuming that C is already a GRD. When an incoming cell is intersected with complete cells, the resulting intersection has edges only where the SID has edges. Also, the way the intersection is calculated by the overseer ensures that only rectangular cells result. For incomplete cells, when altered, their new known edges will always abut the new component (as in Fig. 6d) and so will also be correct. The proof of boundary assignment is based an enumeration of the types of intervals in the new cell(s). For intervals that point to walls, complete cells, or unexplored area, this assignment is simple. The remainder of the cases are defined by their adjacency (partial or complete) to complete and incomplete cells, with the enumeration limited by showing that no interval in a new cell is adjacent to more than two cells. Each case is then shown to be handled correctly by the overseer.

The second statement is shown by considering all possible robot locations at the time of cell addition. For example, if the robot is in a complete cell in C, it will stay in that cell, as the cell will remain unchanged. However, if the current (incomplete) cell is completely replaced by the new cell and the robot is just outside the new cell, it must be directed back into the new cell to continue under CC_{RM}. A complete enumeration of the various cases can be developed by describing the

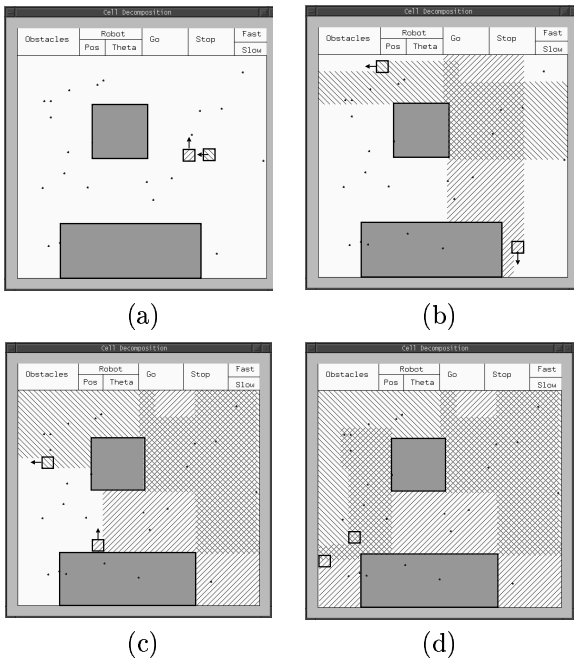

Figure 7: Screenshots of a two-robot coverage run in progress: (a) Beginning of run (b) Just after colleague relationship is formed (c) Each robot exploring a different region (d) Coverage is complete.

types of overlap between the current cell and incoming cell, and for each, whether p is in the incoming cell. Each case is then shown to be handled correctly.

Proposition 3 *Propositions 1 and 2 are sufficient to prove completeness of DC_R.*

In the context of the FSM of Proposition 1, cooperation that alters the robot's current cell effects a transition to a new state. Proposition 2 assures that this transition will lead to a state also in the FSM, so that coverage can continue regardless of the exact nature of the cooperation.

4 Conclusion

An algorithm DC_R has been presented with which a team of independent robots of a specific type can cooperatively cover their shared environment. The outline of a completeness proof of DC_R has also been presented. DC_R has also been implemented in simulation, and a series of screen shots of a single run are shown in Fig. 7. In this two-robot example, note that in Fig. 7b, once a colleague relationship has been formed, robot 0 is no longer performing seed-sowing over the full width of the environment. Finally, when coverage is complete in Fig. 7d, note that only about half of the area has been visited by both robots. While clearly not optimal with respect to time or total distance for the pair, it does show that each robot spends less time covering than it would without cooperation.

While CC_R has been successfully implemented on a single minifactory courier, some extensions to DC_R are still needed for use on a real-world robot system. Most importantly, although the simulation can effect collisions between robots, DC_R currently uses only very simple methods to cause the robots to avoid each other and make progress. These methods sometimes fail, and more intelligent methods are required. This is especially important in minifactory, where couriers' tethers introduce additional types of collision. With this in place, however, implementation on the couriers should be straightforward, as the interaction of DC_R with the physical robot is identical to that of CC_R. This application will help to verify the utility of DC_R as well as provide direction for further extensions.

Acknowledgements

The authors would like to thank Howie Choset and Ercan Acar for helpful discussions about coverage. This research was funded in part by NSF grant DMI-9527190. Zack Butler was supported in part by an NSF Graduate Research Fellowship.

References

[1] V. Lumelsky, S. Mukhopadhyay, and K. Sun, "Dynamic path planning in sensor-based terrain acquisition," *IEEE Trans. on Robotics and Automation*, vol. 6, no. 4, pp. 462–472.

[2] S. Hert, S. Tiwari, and V. Lumelsky, "A terrain covering algorithm for an AUV," *Autonomous Robots*, vol. 3, pp. 91–119, 1996.

[3] E. Acar and H. Choset, "Critical point sensing in unknown environments for mapping," in *Proc. of IEEE Int'l Conf. on Robotics and Automation*, 2000.

[4] H. Choset and P. Pignon, "Coverage path planning: The boustrophedon decomposition," in *Intl. Conf. on Field and Service Robotics*, 1997.

[5] Z. J. Butler, A. A. Rizzi, and R. L. Hollis, "Contact sensor-based coverage of rectilinear environments," in *Proc. of IEEE Int'l Symposium on Intelligent Control*, Sept. 1999.

[6] A. A. Rizzi, J. Gowdy, and R. L. Hollis, "Agile assembly architecture: An agent-based approach to modular precision assembly systems," in *Proc. of IEEE Int'l. Conf. on Robotics and Automation*, pp. 1511–1516, April 1997.

[7] D. W. Gage, "Randomized search strategies with imperfect sensors," in *Mobile Robots VIII*, pp. 270–279, 1993.

[8] N. Rao, V. Protopopescu, and N. Manickam, "Cooperative terrain model acquisition by a team of two or three point-robots," in *Proc. of IEEE Int'l. Conf. on Robotics and Automation*, pp. 1427–1433, April 1996.

[9] I. Rekleitis, G. Dudek, and E. Milios, "Multi-robot exploration of an unknown environment, efficiently reducing the odometry error," in *Int'l Joint Conf. in Artificial Intelligence*, (Nagoya, Japan), pp. 1340–1345, August 1997.