Turn-minimizing multirobot coverage

Isaac Vandermeulen[1], Roderich Groß[1], and Andreas Kolling[2]

Abstract— **Multirobot coverage is the problem of planning paths for several identical robots such that the combined regions traced out by the robots completely cover their environment. We consider the problem of multirobot coverage with the objective of minimizing the mission time, which depends on the number of turns taken by the robots. To solve this problem, we first partition the environment into ranks which are long thin rectangles the width of the robot's coverage tool. Our novel partitioning heuristic produces a set of ranks which minimizes the number of turns. Next, we solve a variant of the multiple travelling salesperson problem (m-TSP) on the set of ranks to minimize the robots' mission time. The resulting coverage plan is guaranteed to cover the entire environment. We present coverage plans for a robotic vacuum using real maps of 25 indoor environments and compare the solutions to paths planned without the objective of minimizing turns. Turn minimization reduced the number of turns by 6.7% and coverage time by 3.8% on average for teams of 1–5 robots.**

I. INTRODUCTION

Minimizing path length is a common, but flawed, objective in robotic path planning. It implicitly assumes the robot moves at a constant speed and turns instantaneously which is never true because real robots need to decelerate to turn. A better objective is to minimize the travel time along the path which depends on turns, acceleration, distance, and speed. In this paper, we consider the minimizing of mission time during multirobot coverage by minimizing the number of turns made by the robots, as well as the distance. Fewer turns also means that real robots get stuck less often and have improved localization.

Coverage is the problem of planning a path such that the robot's tool passes over every point of its environment at least once. Lawn mowing, painting, milling, vacuuming, plowing, and surveillance are all coverage problems. In each application, the robot's tool, such as a rotating blade, paint brush, or camera, traces out a two dimensional region as it moves. The goal of coverage is to find a path such that the tool covers the required area while minimizing some the time needed to follow that path. In multirobot coverage, the combined areas covered by several robots must equal the required area.

A. Related work

Two basic coverage strategies are the contour-parallel and direction-parallel paths [1]. In these strategies, the path either follows the environment's perimeter or moves back and forth in straight lines called *ranks*. For non-convex polygons, these strategies are applied by first decomposing the environment into convex regions using a method such as the boustrophedon decomposition [2]. The order that the cells are covered by contour- or direction-parallel motion is determined by solving the travelling salesperson problem (TSP). Like the TSP, the problems of finding the shortest and time-minimal coverage paths are NP-hard [3].

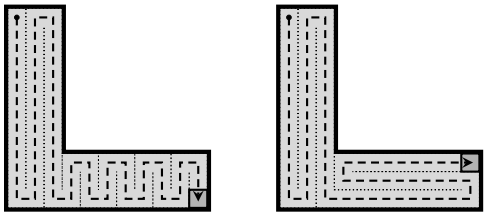

Fig. 1. A coverage path with a single orientation (left) requires more turns than one with two orientations (right). Both paths have the optimal length.

Geometric decompositions form the basis of other coverage approaches [4], [5]. Variants of the boustrophedon decomposition [6], [7] are exact decompositions with large cells. Approximate decompositions, such as Agmon *et al.*'s minimum spanning tree (MST) approach [8], use a fine grid of squares to guarantee a minimal path length with no repeat coverage. Both boustrophedon- and MST-based approaches create paths with many turns. Turns can be reduced by using long straight ranks. In many environments, fewer ranks are needed if multiple orientations are used (Figure 1).

Turn-minimizing coverage involves covering each cell in a direction which minimizes the altitude of that cell [9]. Decompositions obtained by merging polygons in a finer decomposition require exponential time to compute [9], [10]. A faster decomposition technique for turn minimization [11], [12] is cutting a non-convex polygon at each of its concave vertices. Turn-minimizing coverage has been applied successfully to UAV applications [13], [14], [15]. We are not aware of any existing multirobot coverage strategies for non-convex polygons that use turn minimization.

Coverage time can be decreased by using more robots. If the environment is first divided up into regions with equal area, each robot can plan its coverage independently [16], [17]. This approach can be made more robust by replanning during the coverage mission to account for variable speeds [18] or changes in the environment [19], [20]. Alternatively, the robots can plan cooperatively using a modified boustrophedon decomposition [21] or MST-based strategy [22].

[1] Isaac Vandermeulen and Roderich Groß are with the Department of Automatic Control and Systems Engineering, University of Sheffield, Sheffield, UK {iavandermeulen1,r.gross}@sheffield.ac.uk

[2] Andreas Kolling is with iRobot, Pasadena, California, USA akolling@irobot.com

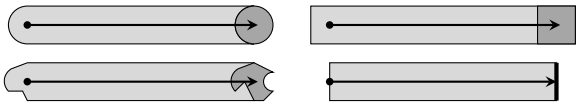

Fig. 2. Regions covered by robots with circular, square, irregular, and straight line tools when moving along a straight path.

B. Contribution

In this paper, we present a new multirobot coverage planner that explicitly considers turn minimization and works for any polygonal environment. It minimizes turns using a novel asymptotically optimal partitioning heuristic which divides the environment into a minimal number of ranks that completely cover the environment. This rank partition is converted into a coverage path for each robot by solving a version of the minmax m-TSP using an existing solver. Our strategy has successfully been used to create coverage plans for teams of 1–5 robots in real environments that were mapped experimentally.

II. PARTITIONING THE ENVIRONMENT

The total time a robot takes to follow a path, including the time needed to slow down for turns, can be approximated by

$$t_{\text{total}} = \frac{\ell_{\text{path}}}{v_{\text{robot}}} + n_{\text{turn}} t_{\text{turn}},$$

where ℓ_{path} is the path length, v_{robot} is the robot's linear velocity, n_{turn} is the number of turns on the path, and t_{turn} is the time needed to make one turn including the time wasted decelerating and accelerating before and after it. A turn is considered any motion between two long straight segments of a robot's path, which usually are by an angle of π but may be other angles. We assume a fixed turning time, although in reality it varies slightly with the angle of the turn.

When a robot moves along a path, the region covered depends on the shape and size of its tool's footprint (Figure 2). On a long straight path, the covered region consists of a long rectangle with some additional caps at either end whose shape depends on the shape of the tool. For the remainder of this paper, we assume that the tool's footprint is a straight line so that the covered region consists solely of the rectangle.

Since the covered area is equal to the tool width times the path length, a complete coverage path's length is bounded by the environment's area divided by the robot's tool's width. This path length is achieved by any path which covers each point of the robot's environment, $\mathcal{E}$, exactly once. Any paths with no redundant coverage (Figure 1) have the same path lengths but vary in their number of turns. Since t_{turn} is nonzero for any real robot, it is important to also minimize the number of turns.

In a coverage path, the number of turns equals the number of straight line segments which each result in the coverage of a long thin rectangle called a *rank*. Our goal is to partition the environment into a minimum number of ranks which cover the entire space.

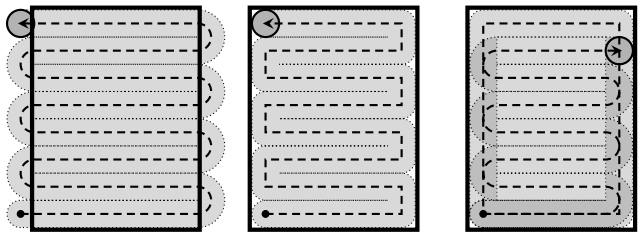

Fig. 3. Complete coverage when turning outside the perimeter (left), incomplete coverage when turning inside the perimeter with zero turning radius (middle), nearly complete coverage when turning inside the perimeter after following the entire perimeter once (right).

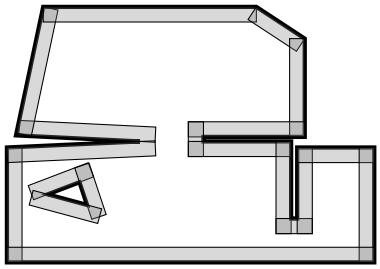

Fig. 4. Perimeter ranks for a polygonal environment. Ranks adjacent to a corner with angle less than $\pi/2$ have been shortened; ranks adjacent to corners with angles greater than π have been lengthened. There is some overlap between perimeter ranks to ensure complete coverage.

Problem 1. *For a polygon, $\mathcal{E}$, find a set of unit width rectangles $\{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$ such that $\cup_{i=1}^{n} \mathcal{R}_i = \mathcal{E}$ while minimizing n.*

In Problem 1, the robot's environment is represented by a polygon with holes, $\mathcal{E} \subset \mathbb{R}^2$. The robot is assumed to have a unit width tool and the coverage ranks are represented by unit width rectangles, $\mathcal{R}_i$, which may be rotated.

A. Perimeter following

The environment's perimeter is difficult to cover because the robot needs to turn around when it reaches the perimeter. If the robot can travel outside the perimeter, it can achieve complete coverage by turning around outside the environment (Figure 3). If it is constrained to the environment, it must follow ranks along the perimeter to achieve near perfect coverage. Due to the shape and size of the robot, some small regions in the corners cannot be covered by any path. We therefore assume that the polygon, $\mathcal{E}$, in Problem 1 has had these small unreachable areas removed.

For problems where the robot is constrained to the environment, we always include one *perimeter rank* per edge of the perimeter (Figure 4). If the angle the edge makes with the next edge is between $\pi/2$ and π, the adjacent ranks end exactly at the corner. If the angle is less than $\pi/2$, the rank is shortened to be contained within the environment. If the angle is greater than π, the rank is extended by the width of the robot to prevent missed coverage near the corner.

B. A rectilinear contraction

Regions of $\mathcal{E}$ not covered by perimeter ranks need to be covered by *interior ranks*. If $\{\mathcal{P}_1, \ldots, \mathcal{P}_{n_{\text{pr}}}\}$ are the perimeter ranks, then the region that still needs to be covered

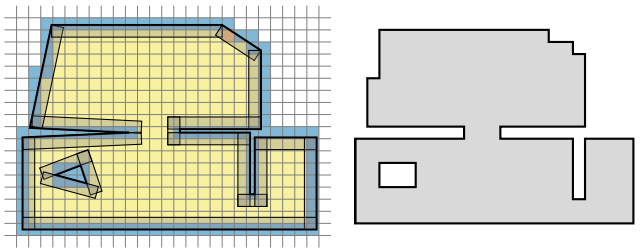

Fig. 5. Overlayed grid (left) used to define the rectilinear contraction (right). Yellow cells are part of the polygon under both definitions; orange cells are part of the polygon only under one of the definitions; blue cells are never part of the contraction.

is $\mathcal{E}_{\mathrm{i}} = \mathcal{E} \setminus \cup_{i=1}^{n_{\mathrm{pr}}} \mathcal{P}_i$. Coverage can be achieved by covering any region $\mathcal{E}_{\mathrm{c}}$ with $\mathcal{E}_{\mathrm{i}} \subseteq \mathcal{E}_{\mathrm{c}} \subseteq \mathcal{E}$. We will choose $\mathcal{E}_{\mathrm{c}}$ to be a rectilinear polygon with integer side lengths. For an integer rectilinear polygon, Problem 1 always has a disjoint solution consisting of some vertical ranks and some horizontal ranks. Since most indoor environments are roughly rectilinear, they can be efficiently covered by these two directions. Although some environments, such as the agricultural fields in [23] are highly non-rectilinear or even curved, if a robot is not able to precisely follow curved paths or make irregular turns, a rectilinear coverage approach may still be more appropriate for these problems.

The rectilinear contraction, $\mathcal{E}_{\mathrm{c}}$, can be obtained by overlaying a unit width grid on top of $\mathcal{E}$ and $\mathcal{E}_{\mathrm{i}}$. This grid should be rotated to maximize the length of perimeter that aligns with the grid axes. Once a grid has been chosen, the contracted rectilinear polygon can be computed in one of two ways (Figure 5).

1) The largest $\mathcal{E}_{\mathrm{c}} \subseteq \mathcal{E}$ is the union of all grid cells fully contained in $\mathcal{E}$.
2) The smallest $\mathcal{E}_{\mathrm{c}}$ with $\mathcal{E}_{\mathrm{i}} \subseteq \mathcal{E}_{\mathrm{c}}$, is the union of all grid cells fully or partially contained in $\mathcal{E}_{\mathrm{i}}$.

If a cell is partially contained in $\mathcal{E}_{\mathrm{i}}$ but not fully contained in $\mathcal{E}$, these two definitions will be different. We use the first definition, and will later extend interior ranks to ensure that all of $\mathcal{E}$ gets covered completely.

C. A coarse checkerboard partition

Partitioning a rectilinear polygon into a minimum number of disjoint horizontal and vertical ranks is non-trivial. On the other hand, partitioning a rectangle is trivial as it should always be covered by a single direction of ranks. Based on this observation, we first partition the rectilinear polygon into disjoint rectangles. When covering rectangles in this partition, it is not always best to cover each rectangle with ranks along its longest axis (Figure 6 left). Ranks of neighboring rectangles can be merged to reduce the total number of ranks, so a partition with different directions of ranks in a single rectangle may be better (Figure 6 middle). However, if a different rectangular partition had been chosen, each rectangle could have been covered by a single direction of ranks (Figure 6 right).

To reduce computational complexity, the rectangular partition should be chosen so that the optimal rank decomposition

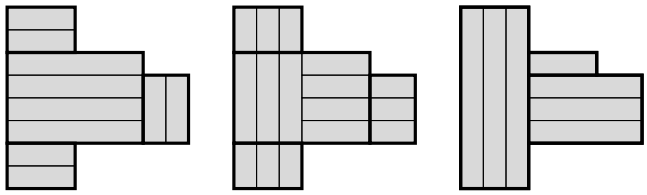

Fig. 6. Coverage of rectilinear environment using locally optimal ranks results in 10 ranks (left). Locally suboptimal ranks result in 16 ranks before merging and 7 after merging (middle). On a better rectangular partition, the locally optimal orientations results in the same 7 ranks (right).

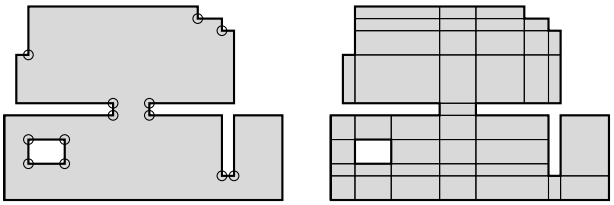

Fig. 7. Concave vertices (left) define the coarsest checkerboard partition (right). The partition is obtained by extending each edge incident to a concave vertex until it intersects with another edge of the polygon's boundary.

of each rectangle contains a single direction. The mixed orientations of the central rectangle of Figure 6 middle was necessary because its vertical neighbors were narrower than it. In general, if all of a rectangle's neighbors share an entire edge with it, the optimal rank decomposition has a single orientation on that rectangle. This observation motivates us to use a *checkerboard partition* where every rectangle has the same width as its vertical neighbors and same height as its horizontal neighbors.

Checkerboard partitions are closely related to the polygon's concave vertices. In any checkerboard partition, each edge of a rectangle extends until it intersects with an orthogonal edge of the rectilinear polygon's boundary. As the edges of the rectilinear polygon are guaranteed to be edges of some rectangle in the partition, edges incident to concave vertices must be extended in any checkerboard partition. The coarsest checkerboard partition can be obtained by using only these edges (Figure 7). We will use this partition, which contains $\mathcal{O}(n^2)$ rectangles where n is the number of vertices of the environment, when computing the rank decomposition.

D. Orienting the rectangles

An assignment of orientations—either horizontal or vertical—to each rectangle of the checkerboard partition defines a rank partition. The objective is to assign orientations to minimize the number of ranks and solve Problem 1. For a checkerboard partition with N rectangles, there are 2^N possible assignments so it is not feasible to check them all. Instead, we use a heuristic which creates a locally optimal assignment.

Local optimality means that the number of ranks from the assignment cannot be improved by changing the orientation of a single rectangle. The locally optimal orientation of a rectangle depends on its dimensions and its neighbors' orientations. As a neighbor's orientation only matters if it

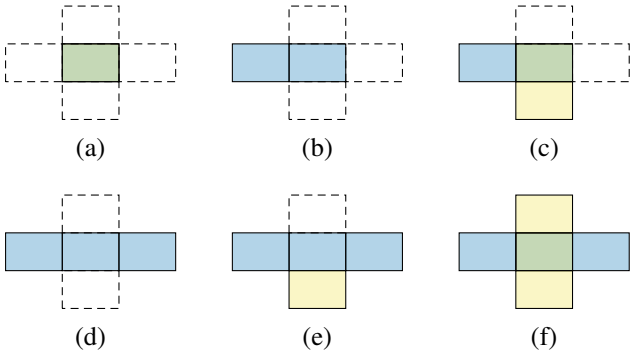

Fig. 8. Possible cases for a rectangle's four neighbors and their orientations. Blue represents horizontal ranks; yellow represents vertical ranks; white with a dotted outline represents no neighbor or a neighbor oriented the wrong way. If the central rectangle is green (cases (a), (c), and (f)), it may be oriented horizontally or vertically depending on its dimensions. If the central rectangle is blue (cases (b), (d), and (e)), it must be oriented horizontally to locally minimize number of turns.

is compatible for merging, there are six cases up to symmetry to consider (Figure 8). The locally optimal orientation maximizes the number of ranks merged minus the number of new ranks added.

(a) No compatible neighbors: Optimal orientation is aligned with the longest edge to minimize new ranks added.
(b) One compatible neighbor: Optimal orientation is aligned with that neighbor so no new ranks are added.
(c) Two compatible neighbors in different directions: Both orientations are optimal and do not add new ranks.
(d) Two compatible neighbors in the same direction: Optimal orientation is aligned with both neighbors to reduce the total number of ranks.
(e) Three compatible neighbors: Optimal orientation is aligned with the direction in which it has two neighbors to reduce the total number of ranks.
(f) Four compatible neighbors: Optimal orientation is aligned with the shorter edge to maximize the number of ranks merged.

A locally optimal assignment is any assignment where every rectangle's orientation is locally optimal.

The criteria for local optimality can also be used to convert any assignment into a locally optimal one by flipping the orientation of any rectangle whose orientation is not locally optimal. Flipping the orientation causes a strict decrease in the cost which is equal to the number of ranks. As the cost is bounded below by the cost of the globally optimal assignment, this procedure is guaranteed to terminate.

This result motivates a heuristic (Algorithm 1) for generating locally optimal solutions to Problem 1. First, it chooses a random orientation for each rectangle (line 1). In each round of the algorithm (lines 3–20), the orientations of rectangles are repeatedly flipped if not locally optimal or set to the *bias* if there are two locally optimal orientations. The bias (line 2) is fixed in each round and is used for case (c) and for cases (a) and (f) if the rectangle is square as both orientations are optimal (line 9). By using a bias we change rectangles' orientations without changing the cost which may enable a different cell to flip later to decrease the cost. In each round, we keep track of which rectangles have already been checked (line 16) and uncheck rectangles if their neighbor flips (lines 10 and 13). Once all rectangles have been checked, the bias is flipped (line 18) and a new round begins if any improvements were made in the previous round. Improvements are defined as flips which decrease the cost of the assignment (line 15). The algorithm terminates after a round where no improvements were made (line 20).

Algorithm 1: Orient rectangles

Input: Checkerboard partition, $P = \{r_1, \ldots, r_N\}$
Output: Locally optimal assignment of orientations $o : P \to \{\mathrm{H}, \mathrm{V}\}$

```
1  o ← Random assignment of orientations
2  bias ← Random orientation (H or V)
3  while True do
4      Set r_1, ..., r_N to unchecked
5      Improvement ← False
6      while there are unchecked rectangles do
7          r ← Random unchecked rectangle
8          L ← {locally optimal orientations for r}
9          if |L| = 2 and o(r) ≠ bias then
10             Flip o(r)
11             Set r's neighbors to unchecked
12         else if |L| = 1 and o(r) ∉ L then
13             Flip o(r)
14             Set r's neighbors to unchecked
15             Improvement ← True
16         Set r to checked
17     if Improvement then
18         Flip bias
19     else
20         Return o
```

If a different bias is used in the last round of Algorithm 1, different locally optimal assignments with the same cost may be returned (Figure 9). Algorithm 1 is guaranteed to reach a local optimum, but not the global optimum. As each iteration of the innermost loop (lines 7 to 17) can be performed in constant time, the algorithm runs very fast and can be repeated multiple times to increase the probability of finding the global optimum.

E. The final rank partition

The locally optimal assignment of orientations for the checkerboard partition can be converted into a rank partition which solves Problem 1 for $\mathcal{E}_\mathrm{c}$. First, adjacent compatible neighbors are merged into larger rectangles. These rectangles are sliced along their long axes into unit width rectangles which are the ranks of the partition which solves Problem 1 for $\mathcal{E}_\mathrm{c}$ (Figure 10 left). These ranks are extended to the perimeter of $\mathcal{E}$ to get the interior ranks, which together with the perimeter ranks from Section II-A solve Problem 1 on $\mathcal{E}$ (Figure 10 right). Extending the interior ranks guarantees that

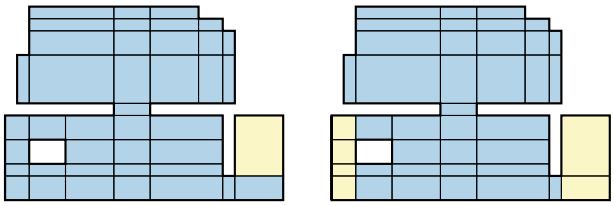

Fig. 9. Optimal orientations for the rectangles (blue is horizontal; yellow is vertical) in a checkerboard partition which were obtained using Algorithm 1. Both solutions result in the same number of ranks. The left solution was optimized with a horizontal bias in the final round of Algorithm 1; the right solution finished with a vertical bias.

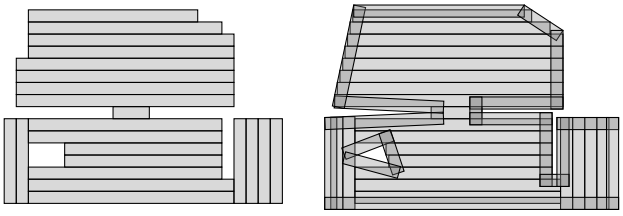

Fig. 10. Rank partitions for $\mathcal{E}_{\mathrm{c}}$ (left) and $\mathcal{E}$ (right).

the combination of perimeter and interior ranks covers the entirety of $\mathcal{E}$ which is reachable given the robot's shape and size. The overall algorithm (Algorithm 2) therefore produces a locally optimal feasible solution to Problem 1. By using different initial orientations in the inner loop (lines 5–10), after many iterations the algorithm finds a global optimum almost surely and it is therefore asymptotically optimal.

Algorithm 2: Rank Partition

Input: Polygonal region, $\mathcal{E} \subset \mathbb{R}^2$; Number of iterations, $n_{\mathrm{iterations}}$

Output: Set of ranks, $\{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$ which solve Problem 1

1 $\{\mathcal{P}_1, \ldots \mathcal{P}_{n_{\mathrm{pr}}}\} \leftarrow$ Perimeter ranks of $\mathcal{E}$
2 $\mathcal{E}_{\mathrm{c}} \leftarrow$ Rectilinear contraction of $\mathcal{E}$
3 $P = \{r_1, \ldots, r_N\} \leftarrow$ Checkboard partition of $\mathcal{E}_{\mathrm{c}}$
4 $n^*_{\mathrm{ir}} \leftarrow \infty$
5 **for** $i \in \{1, \ldots, n_{\mathrm{iterations}}\}$ **do**
6 $o \leftarrow$ Orientations for P `/* Algorithm 1 */`
7 $\{\mathcal{I}_1, \ldots \mathcal{I}_{n_{\mathrm{ir}}}\} \leftarrow$ Interior ranks determined by o
8 **if** $n_{\mathrm{ir}} < n^*_{\mathrm{ir}}$ **then**
9 $n^*_{\mathrm{ir}} \leftarrow n_{\mathrm{ir}}$
10 $I^* \leftarrow \{\mathcal{I}_1, \ldots, \mathcal{I}_{n_{\mathrm{ir}}}\}$
11 **return** $\{\mathcal{R}_1, \ldots, \mathcal{R}_n\} \leftarrow \{\mathcal{P}_1, \ldots, \mathcal{P}_{n_{\mathrm{pr}}}\} \cup I^*$

III. COMBINING RANKS INTO PATHS

Coverage of the entire environment can be achieved by consecutively covering all of the ranks returned by Algorithm 2. The optimal order to cover the ranks can be determined by solving a variant of the travelling salesperson problem (TSP). The TSP is NP-hard [24], however several heuristics [25], [26] and computer packages [27], [28] exist for it. As these methods are well-established, we only present an approach for converting our problem into a TSP instance and do not discuss how to solve the TSP.

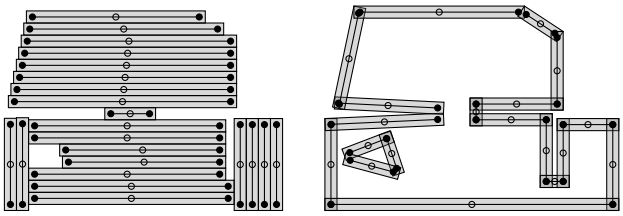

Fig. 11. Endpoints and midpoints of interior ranks (left) and perimeter ranks (right) which are vertices in the graph used by the TSP solver to generate the coverage path. Filled circles represent endpoints; empty circles represent midpoints which are used to enforce that endpoints of ranks are visited consecutively. Lines connecting endpoints and midpoints represent the only edges incident to midpoints with finite weights.

The TSP is usually formulated as a graph, where vertices represent cities and weights on edges represent the travel times between cities. In coverage, we would ideally have the graph's vertices represent coverage ranks and the graph's edge weights represent the travel times between the ranks. However, the travel times between two ranks depend on which endpoints of the ranks the robot is ending and starting at. Since the travel times between these endpoints is usually large, we instead use one vertex for each of a rank's endpoints. Solving the TSP on all rank endpoints does not guarantee a solution where both endpoints of a rank appear consecutively resulting in coverage of the rank. To force the robot to follow the ranks, we constrain the TSP so that endpoints of the same rank are always adjacent. This constraint is enforced by adding an additional midpoint vertex for each rank with infinite cost to any vertex other than its rank's endpoints forcing the rank's endpoints to be visited consecutively (Figure 11). The edge weights for edges between endpoints of different ranks represent the travel time from the end of one rank to the start of another. They can be computed in cubic time by constructing a visibility graph using Welzl's algorithm and solving for the shortest paths using the Floyd-Warshall algorithm [29].

Solving the TSP on the complete weighted graph consisting of all rank endpoints and midpoints and the travel times between them gives a time-minimizing path on the graph (Figure 12 left). For multirobot coverage, we can find paths for each robot by solving the minmax m-TSP [30], [31], [32] on the same graph to minimize the time taken by the slowest robot (Figure 12 right). There exist variants of both the TSP and m-TSP which either specify or do not specify the robots' start and end locations. These variants can be used to solve coverage problems with specified or unspecified start and end locations for each robot.

IV. RESULTS

During the development of the iRobot Roomba i7+™ robotic vacuum cleaner, we experimentally mapped 25 indoor test environments using the robot. These test environments are furnished home and office environments with areas ranging from $10\,\mathrm{m}^2$ to $107\,\mathrm{m}^2$. The combined area of the 25 environments is $1285\,\mathrm{m}^2$. The maps are built from sensor

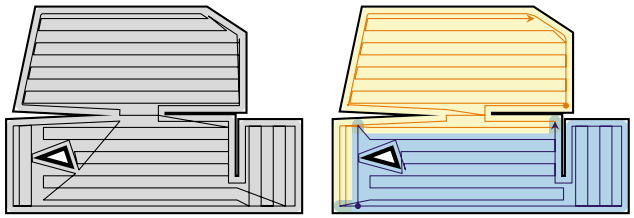

Fig. 12. Turn-minimizing coverage strategies for one robot with no depot (left) and two robots with one depot each (right).

Table I. Cumulative path lengths, numbers of turns, and expected mission times when 25 test environments are covered by teams of 1–5 robots using two different strategies. The 25 environments have a combined coverable area of $1285\,\mathrm{m}^2$ and the robots have a tool width of $10\,\mathrm{cm}$. The expected mission times are for robots which travel at $30\,\mathrm{cm/s}$ and take $5\,\mathrm{s}$ per turn.

n_{robots}	Strategy	ℓ (km)	n_{turns}	t (hh:mm:ss)
1	1 orientation	15.224	12185	30:59:07
	2 orientations	15.195	11377	29:50:08
	Improvement	0.19%	6.63%	3.71%
2	1 orientation	15.326	12260	15:35:14
	2 orientations	15.303	11380	14:58:10
	Improvement	0.15%	7.18%	3.96%
3	1 orientation	15.479	12335	10:28:36
	2 orientations	15.461	11533	10:05:51
	Improvement	0.12%	6.50%	3.62%
4	1 orientation	15.637	12410	7:55:05
	2 orientations	15.564	11586	7:35:49
	Improvement	0.46%	6.64%	4.05%
5	1 orientation	15.757	12485	6:22:53
	2 orientations	15.715	11663	6:08:37
	Improvement	0.27%	6.58%	3.72%

data from the robot's camera, bumper, and odometry and are then smoothed to obtain a polygonal boundary.

For these maps, we computed coverage plans using two strategies: the turn-minimization strategy with two rank orientations presented in this paper and a similar strategy with only one rank orientation. Paths were computed from the rank decomposition using the m-TSP approach described in [32]. The two strategies were compared on the basis of total path length, total number of turns, and expected mission time when all 25 environments are covered by teams of 1–5 robots (Table I). Sample paths for a team of two robots in the largest of the 25 environments using both strategies are shown in Figure 13. The two approaches have nearly identical path lengths; however, our turn minimization approach reduced turns by 6.7% resulting in a 3.8% reduction in total mission time. When more robots are used, the total path length and number of turns remain similar but the expected mission time, decreases by a factor of approximately $\frac{1}{n_{\text{robots}}}$ because the robots are covering the environment simultaneously.

When computing optimal rank partitions, Algorithm 1 ran 50 times with different random initial conditions and we recorded the number of iterations of the inner loop (lines 6–16) and computation time needed to reach the local minimum. The number of iterations scaled linearly with the number of rectangles in the checkerboard partition and the computational runtime scaled proportional to $n_{\text{v}}^{1.59}$ where n_{v} is the number of vertices (Figure 14) and only required $15\,\mathrm{ms}$ of computing time for the largest real environment.

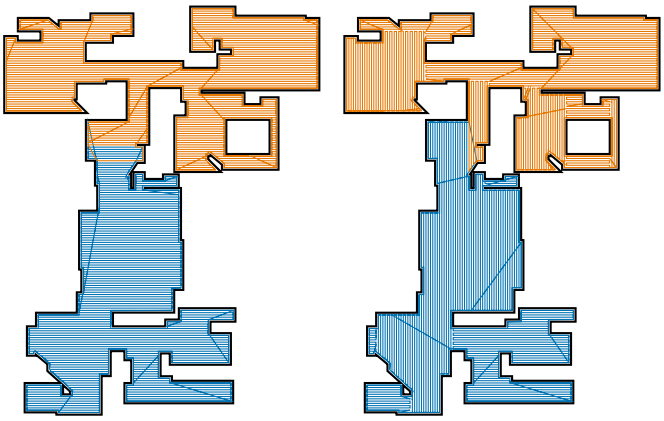

Fig. 13. Comparison of robot coverage plans for a team of two robots in a $107\,\mathrm{m}^2$ test environment using one orientation (left) and two orientations (right). For the 1 orientation strategy, the robots have expected coverage times of 1:17:30 (blue) and 1:17:27 (orange). The 2 orientation strategy's mission time is 3.7% faster with expected coverage times of 1:14:38 (blue) and 1:14:28 (orange).

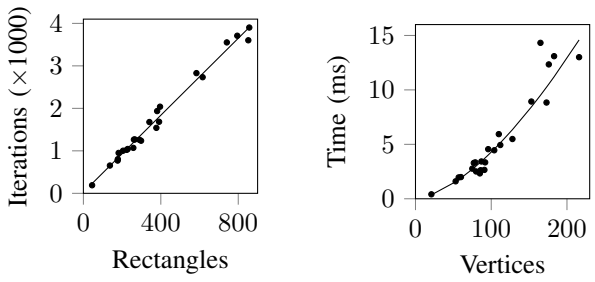

Fig. 14. Regression results showing linear relationship ($\widehat{y} = 4.53x + 26.24$) between number of iterations of the inner loop of Algorithm 1 and the number of rectangles in a checkerboard partition (left); and relationship of $\widehat{y} = 0.002826x^{1.59}$ between the computational runtime of Algorithm 1 and the number of vertices in a polygon. Computations were performed in C++ on a standard consumer laptop running Ubuntu.

V. CONCLUSIONS

Many robots are slow at turning so the time needed to follow a path depends on the path's length and the number of turns. We presented a multirobot coverage strategy which explicitly considers the number of turns required. Turns are minimized by partitioning the environment into long unit-width rectangles called ranks. Perimeter ranks are parallel to the perimeter of the environment; interior ranks are oriented horizontally or vertically. The interior ranks are constructed using a novel heuristic which minimizes the number of ranks. Coverage paths are generated for m robots by solving a version of the minmax m-TSP on a graph related to the set of ranks. We compared this strategy with one which does not minimize turns on 25 real indoor environments with a combined area of $1285\,\mathrm{m}^2$ mapped by the iRobot Roomba i7+™. For coverage with 1–5 robots, this strategy reduced turns by 6.7% and the coverage time by 3.8% on average.

REFERENCES

[1] M. Held, *On the computational geometry of pocket machining*. Springer Science & Business Media, 1991, vol. 500.

[2] H. Choset and P. Pignon, "Coverage path planning: The boustrophedon cellular decomposition," in *Field and service robotics*. Springer, 1998, pp. 203–209.

[3] E. M. Arkin, S. P. Fekete, and J. S. Mitchell, "Approximation algorithms for lawn mowing and milling," *Computational Geometry*, vol. 17, no. 1-2, pp. 25–50, 2000.

[4] H. Choset, "Coverage for robotics–a survey of recent results," *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1, pp. 113–126, 2001.

[5] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1258–1276, 2013.

[6] A. Xu, C. Viriyasuthee, and I. Rekleitis, "Efficient complete coverage of a known arbitrary environment with applications to aerial operations," *Autonomous Robots*, vol. 36, no. 4, pp. 365–381, 2014.

[7] E. U. Acar, H. Choset, A. A. Rizzi, P. N. Atkar, and D. Hull, "Morse decompositions for coverage tasks," *The International Journal of Robotics Research*, vol. 21, no. 4, pp. 331–344, 2002.

[8] N. Agmon, N. Hazon, and G. A. Kaminka, "The giving tree: Constructing trees for efficient offline and online multi-robot coverage," *Annals of Mathematics and Artificial Intelligence*, vol. 52, no. 2, pp. 143–168, 2008.

[9] W. H. Huang, "Optimal line-sweep-based decompositions for coverage algorithms," in *International Conference on Robotics and Automation (ICRA)*, vol. 1. IEEE, 2001, pp. 27–32.

[10] W. Sheng, N. Xi, H. Chen, Y. Chen, and M. Song, "Surface partitioning in automated CAD-guided tool planning for additive manufacturing," in *International Conference on Intelligent Robots and Systems (IROS)*, vol. 2. IEEE, 2003, pp. 2072–2077.

[11] D. Ding, Z. S. Pan, D. Cuiuri, and H. Li, "A tool-path generation strategy for wire and arc additive manufacturing," *The International Journal of Advanced Manufacturing Technology*, vol. 73, no. 1-4, pp. 173–183, 2014.

[12] S. Bochkarev and S. L. Smith, "On minimizing turns in robot coverage path planning," in *International Conference on Automation Science and Engineering (CASE)*. IEEE, 2016, pp. 1237–1242.

[13] I. Maza and A. Ollero, "Multiple UAV cooperative searching operation using polygon area decomposition and efficient coverage algorithms," in *Distributed Autonomous Robotic Systems 6*. Springer, 2007, pp. 221–230.

[14] Y. Li, H. Chen, M. J. Er, and X. Wang, "Coverage path planning for UAVs based on enhanced exact cellular decomposition method," *Mechatronics*, vol. 21, no. 5, pp. 876–885, 2011.

[15] G. S. Avellar, G. A. Pereira, L. C. Pimenta, and P. Iscold, "Multi-UAV routing for area coverage and remote sensing with minimum time," *Sensors*, vol. 15, no. 11, pp. 27 783–27 803, 2015.

[16] S. Hert and V. Lumelsky, "Polygon area decomposition for multiple-robot workspace division," *International Journal of Computational Geometry & Applications*, vol. 8, no. 04, pp. 437–466, 1998.

[17] H. Bast and S. Hert, "The area partitioning problem," *Canadian Conference on Computational Geometry (CCCG)*, 2000.

[18] M. Ahmadi and P. Stone, "A multi-robot system for continuous area sweeping tasks," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2006, pp. 1724–1729.

[19] C. S. Kong, N. A. Peng, and I. Rekleitis, "Distributed coverage with multi-robot system," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2006, pp. 2423–2429.

[20] I. Rekleitis, A. P. New, E. S. Rankin, and H. Choset, "Efficient boustrophedon multi-robot coverage: An algorithmic approach," *Annals of Mathematics and Artificial Intelligence*, vol. 52, no. 2, pp. 109–142, 2008.

[21] N. Karapetyan, K. Benson, C. McKinney, P. Taslakian, and I. Rekleitis, "Efficient multi-robot coverage of a known environment," in *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1846–1852.

[22] A. C. Kapoutsis, S. A. Chatzichristofis, and E. B. Kosmatopoulos, "DARP: Divide areas algorithm for optimal multi-robot coverage path planning," *Journal of Intelligent & Robotic Systems*, vol. 86, no. 3-4, pp. 663–680, 2017.

[23] T. Oksanen and A. Visala, "Coverage path planning algorithms for agricultural field machines," *Journal of Field Robotics*, vol. 26, no. 8, pp. 651–668, 2009.

[24] C. Papadimitriou, "The Euclidean travelling salesman problem is NP-complete," *Theoretical Computer Science*, vol. 4, no. 3, pp. 237–244, 1977.

[25] S. Lin and B. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research*, vol. 21, no. 2, pp. 498–516, 1973.

[26] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," DTIC Document, Tech. Rep., 1976.

[27] W. Cook. (2016) Concorde TSP solver. [Online]. Available: http://www.math.uwaterloo.ca/tsp/concorde/

[28] K. Helsgaun. LKH version 2.0.7. [Online]. Available: http://www.akira.ruc.dk/~keld/research/LKH/

[29] R. W. Floyd, "Algorithm 97: shortest path," *Communications of the ACM*, vol. 5, no. 6, p. 345, 1962.

[30] E. Kivelevitch, B. Sharma, N. Ernest, M. Kumar, and K. Cohen, "A hierarchical market solution to the min-max multiple depots vehicle routing problem," *Unmanned Systems*, vol. 2, no. 01, pp. 87–100, 2014.

[31] Y. Wang, Y. Chen, and Y. Lin, "Memetic algorithm based on sequential variable neighborhood descent for the minmax multiple traveling salesman problem," *Computers & Industrial Engineering*, vol. 106, pp. 105–122, 2017.

[32] I. Vandermeulen, R. Groß, and A. Kolling, "Balanced task allocation by partitioning the multiple traveling salesperson problem," in *18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 2019.