CC_R: A Complete Algorithm for Contact-Sensor Based Coverage of Rectilinear Environments

Zack J. Butler

October, 1998

CMU-RI-TR-98-27

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA

© Carnegie Mellon University 1998

This work was supported in part by NSF grant DMI-9523156. The author was supported in part by an NSF Graduate Research Fellowship.

Abstract

Sensor-based coverage is a powerful tool for robots to use to discover their environment, especially in cases when complete knowledge of the environment is required. Current methods have demonstrated the ability to cover virtually arbitrary planar areas but require a remote sensor of finite range, such as sonar. The work presented here handles the case of robots that use only contact sensing to determine the boundaries of their environment. An algorithm CC_R is presented that works in a wide range of rectilinear geometries. A proof of completeness is presented for CC_R which shows that it always causes the robot executing it to cover its environment. Also, possible extensions of CC_R to incorporate a wider variety of environments are discussed.

Contents

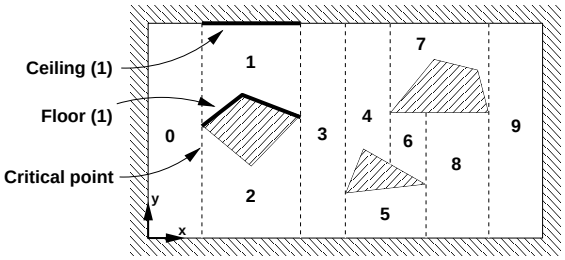

Figure 1: Canonical cell decomposition of a polygonal environment.

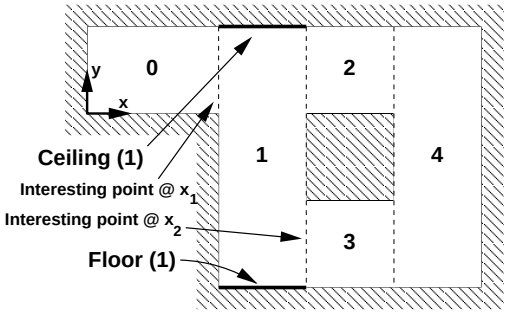

Figure 2: Cell decomposition of a rectilinear environment.

1 Introduction

In certain mobile robotics tasks, an initially unknown environment must be completely explored before the task can begin. In other scenarios, such as searching for meteorites, complete exploration of an area is the task itself. Tasks such as these can be stated as a type of coverage. In general, coverage is defined as passing a sensor or effector over every point in an area, and the "coverage problem" can be defined as determining a continuous path that will produce coverage of an area. While the coverage problem has been solved in the plane for known environments [1], robots performing the tasks of interest will not have a map of the area to be covered. They must instead simultaneously create a map of the environment from their sensor data and plan paths to continue coverage. This is a problem referred to generically as *sensor-based coverage*. This task has garnered some recent attention from the robotics community, but because of the variety of environments and abilities of robots performing coverage, no single current solution satisfies the entire problem domain.

Previous sensor-based coverage methods have concentrated on robots with sonar or other range sensors that can sense objects within a specified radius from the robot's body [2, 3]. These methods also generally operate in a planar environment with polygonal or C^2 obstacles in "general position", i.e. certain reasonable assumptions are made about the geometry of the environment as are described below. For the work presented here, however, the focus is on robots that can use only contact sensing to determine the geometry of their environment. While not as rich a modality as sonar and the like, contact sensing can be very reliable and simple to implement, and can provide sufficient information to produce complete maps.

Among previous coverage methods, cell-decomposition methods such as the one presented in [4] have proven to be the most useful to base the current method on. This particular method arose in coverage of planned environments, inspired by roadmaps used for robot navigation. For planned coverage under this method, the environment is divided into cells based on a 1-D sweep. As a 1-D slice passes through the environment

from left to right, the x location of the first and last points encountered of each obstacle are deemed *critical points*. These points determine the boundaries of cells, which are constructed as shown in Fig. 1 such that each has a continuous *floor* and *ceiling*. (The general position assumption is that no two critical points share an x coordinate.) Coverage of each cell is then performed with *seed-sowing*, in which the robot travels in parallel vertical strips from one side of the cell to the other. Complete coverage is accomplished within this decomposition by simply traveling to every cell, covering each in turn. Paths are planned between cells by making a graph of the adjacency relationships between the cells and searching this graph to find paths. Sensor-based coverage has also been implemented in this framework [5]. This is done by starting with the assumption that the environment can be represented with a single cell. The robot then begins coverage within this cell until a discontinuity is detected in the cell's floor or ceiling. When this occurs, new cells are created appropriately which are themselves covered once the current cell is complete. This method is able to cover unknown generic C^2 environments.

As a first step toward a general coverage algorithm based on contact sensing, however, the work here has been restricted to purely rectilinear environments. In these environments, all boundary and obstacle edges are parallel to the x or y axis of the robot. This allows some simplifications of the algorithm, but introduces some types of geometries not encountered in the general C^2 environment. For example, in a rectilinear environment, the cells are themselves simply rectangles, as seen in Fig. 2. However, the divisions between cells are no longer at simple critical points, but at all *interesting points*, namely, wherever there is a wall parallel to the y axis. Also, the assumption of general position as described above has been eliminated, since allowing for multiple coincident interesting points is not much harder than allowing for a finite-length wall at an interesting point. It should be noted, however, that the current algorithm assumes a minimum width for all cells of the width of the robot, although this is not a fundamental problem. In addition, this work assumes that the robots have no significant dead reckoning error. Although this may not be true for some systems, it is a difficult problem, and one which may be able to be incorporated in a future version of this solution.

2 CC_R: General Description

To address the specific problem outlined above, an algorithm CC_R has been written which solves the coverage problem for a square robot in a rectilinear environment using only contact sensing. A complete description of it is given in Appendix A, but a summary will be given here to provide an understanding of the nature of the algorithm.

One important aspect of CC_R is that it keeps no state other than a cell decomposition C, which is simply a list of cells as defined in Sec. 3 and an associated list H of *placeholders*, line segments placed adjacent to cells to represent entry points to unknown parts of the environment. Therefore, each time a decision is to be made about the cor-

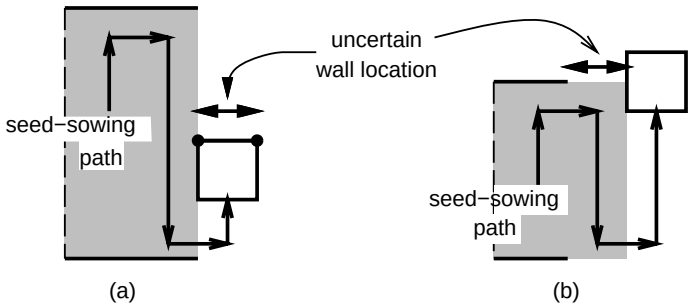

Figure 3: Some of the ways in which interesting points are discovered: (a) unexpected collision (b) unexpected non-collision.

rect movement to make to continue coverage, only the map is used to make this decision, rather than using explicit history or state. Decisions are made at *events*, which are defined as all occasions when the robot experiences a collision in addition to all occasions when the robot has completed its trajectory (as described below). After an event, the *event handler* determines the type of event that has occurred and alters C accordingly. The *map interpreter* then applies a series of rules to C, taking into account the robot's current position, and determines a new direction of travel v along with a distance d_{max}. d_{max} is defined as the maximum possible distance the robot can travel in the direction v before a decision or a change of direction is required. The robot can then use v and d_{max} to generate a smooth trajectory. This method of choosing d_{max} assures that if the trajectory completes without a collision, the map interpreter will need to be rerun to change the robot's direction, and so a event should have taken place.

In the general case, the desired behavior of CC_R is that of seed-sowing. As an arbitrary choice, the strips will be in the $\pm y$ directions, which will produce decompositions like that in Fig. 2, and progress within a cell will be made first in the $+x$ direction, then in $-x$. Seed-sowing is an efficient way to cover a cell once it has been started, and so is performed whenever a cell has a known floor and ceiling and one known and explored side edge. It is also performed initially in the first cell even though neither side edge is known. Seed-sowing will then continue until an interesting point is detected.

Depending on the type of interesting point, it will be discovered and handled differently. Most interesting points are discovered during a vertical motion (as shown in Fig. 3), at which point the exact location of the corner (and therefore the interesting point) are indeterminate. At this point, the map interpreter notices this uncertainty and immediately directs the robot to move such that the corner is localized. For interesting points that are discovered by direct contact with the vertical wall, this secondary step is unnecessary.

Once an interesting point is discovered, the remainder of the cell edge corresponding to that interesting point is explored. This serves two purposes: completion of the current strip of seed-sowing (the one that was interrupted by the discovery of the interesting

point) and discovery of any other cells that may lie next to the current cell. If the interesting point was discovered as shown in Fig. 3a, this exploration will complete coverage of the cell to the left of the interesting point. In the case shown in Fig. 3b, the localization of the interesting point will complete the cell on the left, but the exploration of the newly discovered edge will still continue, giving complete knowledge of the left edge of the new cell.

To properly generate all these behaviors, the map interpreter applies a list of rules to C one at a time, using the first applicable rule to determine v and d_{max}. Special cases such as interesting points that need to be localized or edges that are partially explored are checked first, so that these are able to interrupt seed-sowing until the special case has been dealt with. Seed-sowing rules are then applied if the cell is incomplete but has no special characteristics, and if the cell is complete, rules for that case are invoked as discussed in detail below.

The job of the event handler is then to keep the cell decomposition up to date at all times. When a collision occurs as in Fig. 3a, for example, the event handler knows that the current cell can now extend no further to the right than the right side of the robot. It therefore changes the maximum potential rightward extent of the current cell to reflect this new knowledge. The map interpreter can then use this knowledge to interrupt seed-sowing, causing the robot to travel left, then up, based on the characteristics of the cell (the same rules would be invoked for the mirror-image case, for example, but a rightward motion would be produced initially). Finally, the robot will be moved rightward to cause a collision with the uncertain edge, at which point the event handler sets the exact value of the current cell's right edge. In addition, the event handler creates a placeholder at the time of the initial collision that points to the newly discovered area to the right, and moves that placeholder when the second collision determines its exact x location.

Finally, when the map interpreter finds that the robot is in a cell that has already been completed, it must choose another region to travel to and begin coverage in. For reasons discussed in Sec. 3.3, if there is an incomplete cell in C, it will be selected as the next area to cover. If no incomplete cell exists, a placeholder will be selected as a destination and turned into a small incomplete cell to be covered. For greater efficiency, if the current cell has a placeholder adjacent to it, that placeholder is turned into an incomplete cell and the robot travels into it. If the current cell adjoins only other complete cells, a placeholder is chosen from elsewhere in the environment, and the adjacency relationship of the cells is used to plan a path to the cell that the placeholder is attached to.

To demonstrate CC_R, it has been implemented in simulation and tested in a variety of environments. It has successfully covered all environments that it has been tested in from a variety of initial locations. The simulations have also given some insight into the efficiency of CC_R, and during development, aided in discovering algorithmic flaws. This implementation will also be transferred to a physical robot system for further testing and verification.

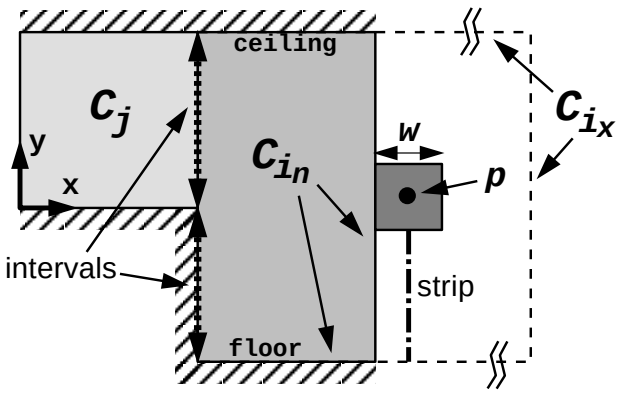

Figure 4: The data structures associated with cell C_i; cell C_j is also shown for clarity.

3 Completeness Proof of CC_R

Having described the operation of CC_R, we now show that it is in fact complete, that is, that a robot executing it will cover any environment with rectilinear boundaries and obstacles and whose canonical cell decomposition contains no cells thinner than the width of the robot.

The proof relies on the following background and definitions:

- The environment representation consists of a cell decomposition C (a list of cells $C_0 \ldots C_n$) and a list of placeholders $H_0 \ldots H_m$.
- A cell C_i (as can be seen in Fig. 4) has area defined by a pair of rectangles: C_{i_n} is its minimum known extent and C_{i_x} its maximum possible extent. Its *floor* and *ceiling* are horizontal, and its other two edges are referred to as *side edges*.
- Side edges have associated *intervals*, line segments which are each adjacent to a specific wall (a "wall interval") or area of free space.
- An edge is *explored* when its disposition (either adjacent to a wall or a specifically denoted area of free space) is known at each point between the ceiling and floor of the cell.
- A *well-opened* cell has a known floor and ceiling and one known and completely explored side edge.
- A *complete* cell is one with zero width or one in which $C_{i_n} = C_{i_x}$, left and right edges have been completely explored, and has been completely covered (with seed-sowing strips).
- The robot has position $p = (p_x, p_y)$ and width w.

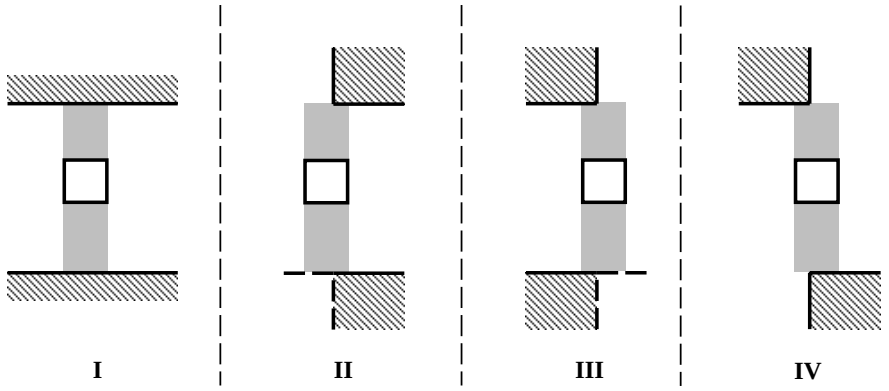

Figure 5: Potential initial conditions; the shaded area is that covered by the first strip of seed-sowing.

- In all figures, the y axis points toward the top of the page, the x axis to the right

To show completeness of CC_R as detailed in Appendix A, we will show that the environment is divided into cells which collectively span the environment, each of which is completely covered. To do this, we will show that:

- From any initial position, at most two cells are created, both well-opened
- Any well-opened cell will be completed when visited
- Every cell completion results in at most one well-opened cell and any number of placeholders
- Any placeholder can be turned into a well-opened cell
- Every incomplete cell will be visited and every placeholder removed

3.1 Initial conditions

To show that any initial position will lead to at most two well-opened cells, four different cases must be considered as shown in Fig. 5.

Case I: The ceiling will be discovered first due to rule 3, then the floor by rule 4. This second move will also be a seed-sowing strip, so that the initial cell will have the width of the robot. Since rules 1-4 no longer apply, seed-sowing will begin to the right. Although this cell is not well-opened, progress will continue as if it was. This is because seed-sowing to the right will take precedence over seed-sowing to the left, and once the right side of the cell is reached, rules 1 and 2 will cover and explore the right edge. This

cell will now be truly well-opened (from the right), as will any cell created during the completion of its right edge.

Case II is very similar to case I. The robot will perform seed-sowing to the right, completing the right side of the cell. However, the assumed leftward extent of the cell will include the initial strip and so be larger than the actual cell. To assure that this is handled correctly, this cell must be returned to and completed from within before it is approached from any other direction. This is in fact assured, as shown at the end of Sec. 3.3.

In case III, the first strip will be made corresponding to the height of the left-hand cell even though the center of the robot is actually in the right-hand cell. However, on the next strip, the robot will travel past where it expects the ceiling to be. It will then behave as in case III of cell completion. After this process, the left-hand cell (C_0) will have a known and explored right-hand side, and therefore be well-opened. It will, however, have width less than that of the robot, but this will not effect coverage as long as (again) C_0 is reentered and completed before it is approached from the left. With the robot now in C_1, this cell will now be made well-opened as described in case II or III of Sec. 3.4.

Case IV also begins with a strip that is smaller than the cell the robot's center is in. However, in this case the initial strip does not actually correspond to any cell in the final decomposition. In this case, the area on the right becomes cell C_1 as in case II above. However, when the robot returns to cell C_0 (the remains of the initial strip) to continue seed-sowing to the left, the first motion in the $-y$ direction will travel past the floor of C_0 and C_j ($j \geq 2$) will be created just as C_1 was created on the other side. When the boundary between C_0 and C_j is localized, C_0 will be shrunk down to zero width, making it a completed cell by definition. (Having a zero-width cell in C does not alter its correctness or the ability to plan paths within it.) C_j can now be made well-opened just as C_1 was, for a total of two well-opened cells. Note that this case also assumes that C_0 will be returned to before being approached from the left, as shown at the end of Sec. 3.3.

3.2 Edge exploration

Before discussing the mechanisms of cell completion, we will first show that any side edge of known location that is known to be a wall at the robot's y location (p_y) will be completely explored. Since rule 2 has an upward bias, the edge will first be explored from p_y to the cell's ceiling. Once the robot has reached the ceiling, rule 2 will move it in the $-y$ direction to a point next to the wall segment from which the exploration started. At this point, exploration will continue down to the floor of the cell the same way as the exploration to the ceiling. Both halves of the exploration will be performed as described below, although for clarity, only upward exploration will be described. Starting at p_y,

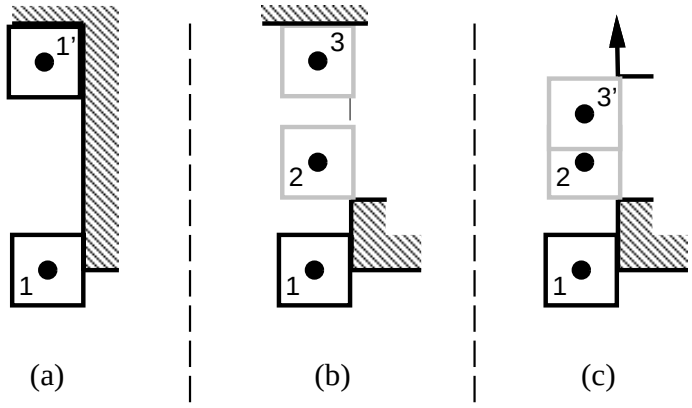

Figure 6: Types of edges to be explored: (a) Case A (b) Case B1 (c) Case B2.

the following two steps will be iterated beginning at location 1 in each of the cases of Fig. 6:

- The disposition of the side edge is known at p_y, so a small step is taken in the $+y$ direction.
- The side at the new y location is unknown, so a small step is taken toward the edge in x, resulting in a collision.

These steps continue until one of two things happens:

- Case A: A y motion results in a collision at the ceiling. This occurs when the wall continues to the ceiling of the cell.
- Case B: An x move does not result in collision. This occurs when there is some free space beyond the cell edge. (i.e. when the initial wall segment does not extend to the ceiling.)

For case A, upon collision at location 1', if a strip exists, it is marked as completed and the covered portion of the cell is extended to the newly explored edge. Next, rule 2 will generate a small x motion (and a collision) to complete the interval corresponding to the right-side wall. Since this interval now extends to the ceiling, this edge is now explored to the ceiling, and the robot will return to the starting point of this exploration to cover the lower portion of the cell if necessary.

In case B, the robot will come to location 2 where its edge is not within the cell. At this point, a placeholder is created for the free space adjacent to the cell and progress continues as follows:

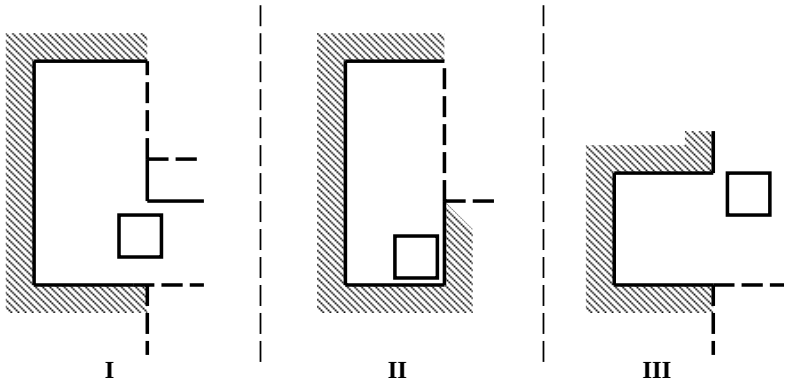

Figure 7: Types of interesting points

- Rule 2 of the algorithm still applies, but rather than retreating from this known interval, it will move in the $+y$ direction while keeping its edge just out of the cell until a collision occurs.
- After a collision (at location 3 or 3'), a zero-length wall interval is created just beyond the current placeholder. This causes rule 2 to now move the robot back entirely within the cell.
- Once entirely within the cell, the robot moves in $+y$ again to continue exploration.

This last move could have two results:

- Case B1: The adjacent free space extends to the cell's ceiling, and so this move will immediately result in collision.
- Case B2: The cell continues past the placeholder, and so the last move does not result in collision.

In case B1, this second collision indicates that the ceiling has been reached (note that the location of the ceiling does not need to be already known for this to occur). The seed-sowing strip is then finished if it exists and the exploration of the edge is complete to the ceiling. In case B2, the exploration of the edge will continue on as directed by rule 2 and be subject to the same analysis just completed. Therefore, any edge satisfying the preconditions outlined above will be completely explored.

3.3 Completing cells

We are now ready to show that all well-opened cells can be completed correctly. First of all, in a well-opened cell, seed-sowing will continue uninterrupted away from the known

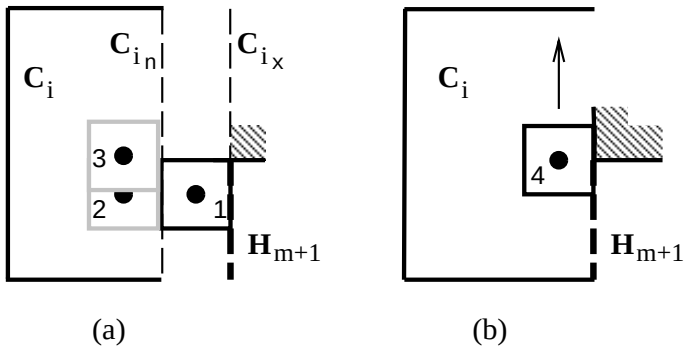

Figure 8: Detail of Case I cell completion: (a) Before and (b) after localization of the interesting point.

side edge until an interesting point is discovered. There are then three types of interesting points that can occur at the right edge of a cell, as shown in Fig. 7. It should be noted that each case can also occur vertically mirrored and/or horizontally mirrored, and will be handled in the same way as described below.

Case I: the interesting point at the right edge of the cell is discovered by an unexpected collision just after the moment pictured. When this collision occurs, the event handler will instantiate a placeholder H_{m+1} next to the current robot location (robot location 1 in Fig. 8a). H_{m+1} will be placed at the new maximum right edge of C_i at this point, but will be moved when the edge is localized. The next few steps that CC_R will take are all generated by rule 1, since the finite edge condition persists until location 4. The motions are as follows (each number refers to a robot location shown, and is the location of the robot before the move described):

1. The robot moves to the left to become completely within C_i, since it is next to a known piece of free space.
2. The robot moves up a small distance past the top of the free space (since the free space is known to extend to the floor of C_i).
3. The robot then drives right until a collision.

At this point (location 4 shown in Fig.8b), the right edge of C_i is set. The x location of H_{m+1} is also set to the x value of the new edge. Then, since the location of the right edge of C_i is known and it is known to be a wall at the current y location, the exploration will finish, completing C_i.

Case II: the robot encounters the right edge of the cell during an x motion of seed-sowing. At this point, the right edge of C_i is set, and a zero-length interval is instantiated corresponding to the shortest wall that could have caused this collision. The right edge is now known at the current y location, but not explored, so exploration (simultaneous

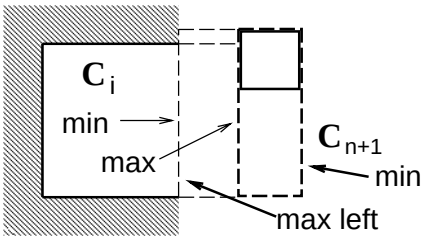

Figure 9: Case 3.

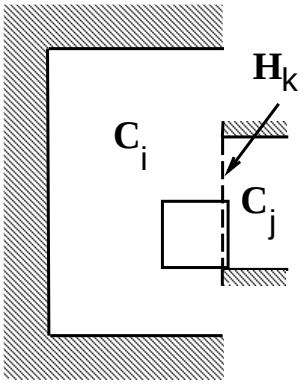

Figure 10: Exploring an edge in the presence of placeholders

with a seed-sowing strip) will be the next step. Since the right side is already explored to the floor of C_i, the exploration need only continue toward the ceiling, completing the cell when done.

Case III: the edge of C_i is detected when the robot drives past the y coordinate of the ceiling, i.e. when a trajectory completes with p no longer within C_{i_x}. At this point, a taller cell C_{n+1} is instantiated as shown in Fig. 9. The robot is now in C_{n+1}, in which rule 1 is applied to the left edge, and since the boundary is unknown at p_y, the robot immediately moves left until a collision occurs. As a result of the collision, the location of the right edge of C_i becomes known, as well as its entire disposition (namely, that it connects to C_{n+1}). Since coverage had already extended at least to this x location, C_i is now complete, with exactly one open cell (C_{n+1}, which is now the robot's current cell) and no placeholders created.

These three cases apply whenever there is only unknown geometry to the right of the cell. However, as long as there are only placeholders (and the completed cells they are attached to) to the right of C_i, the cell will still be correctly completed. In these cases, as shown in Fig. 10, the maximum rightmost extent of C_i will be set by another cell C_j before the seed-sowing reaches the cell boundary. However, since the minimum width of C_i is increased only by seed-sowing, the robot will cover to the right edge before exploring that edge (only at that point will the right edge have known location, a precondition for rule 2). The exploration of the right edge happens as described in Sec. 3.2 with one exception. Whenever the robot discovers free space to its right (as shown in Fig. 10), it recognizes the existence of C_j and looks in C_j for the appropriate placeholder H_k. An interval corresponding to H_k is created in C_i, and the interval pointing to H_k in C_j is changed to point to C_i. H_k is then removed.

The preceding arguments do not apply if another incomplete cell's maximum extent overlaps with C_{i_x}. However, we can guarantee that this will not happen by showing that C will never include more than two incomplete cells and these two cells will not overlap.

First of all, if during the operation of CC_R at time t_0 only one cell C_i is incomplete and C_i is well-opened, there will never be more than one incomplete cell at any time $t > t_0$. This can be shown as follows: if C_i is completed via Case I or II above, there will be no incomplete cells when C_i is complete. A placeholder will then be instantiated as a new incomplete cell, returning to the same situation as at t_0. If, however, C_i is completed via Case III, a new cell will be created and be incomplete, but C_i itself will be immediately completed and the new cell will be well-opened, again leaving C with only one incomplete cell which is well-opened.

We have already shown that initially, at most two well-opened cells will be created. If only C_0 is created initially and its right side is completed via Case I or II, it can then be considered well-opened from its right side, and it will be the only incomplete cell. If C_1 is also created initially (from initial positions III or IV) or equivalently if C_0 has its right side completed via Case III, two well-opened cells will exist with a shared boundary ($C_{0,right} = C_{1,left}$). In all of these cases, the robot will be in C_1. If C_1 is completed via Case I or II, the robot will remain in C_1 after completion and use rule 6 to return to C_0, which will be the only remaining incomplete cell. If C_1 is completed via Case III, the robot will then be in C_2 under the same conditions. Since each new cell opened this way must lie entirely to the right of the previous one, the two incomplete cells will not overlap. When a cell is finally completed without opening a new cell, again rule 6 will direct the robot back to C_0, the only incomplete cell in C, which will become well-opened if it is not already. This assures that there will never be two overlapping incomplete cells in C, allowing the arguments presented in this section to apply to all well-opened cells.

3.4 Opening Cells

When a cell is completed without producing a new open cell and no incomplete cell exists, a placeholder must be instantiated as a cell. We now show that any placeholder will become a well-opened cell under CC_R. There are three types of placeholders, as shown in Fig. 11.

Type I: When the cell is instantiated, its ceiling and floor are both already known due to the known presence of the vertical walls, in addition to the left edge that will be known for all cells built from placeholders. Once the cell is entered, all the preconditions for seed-sowing are met, and it begins covering from the known left edge as a well-opened cell.

Type II: The cell is instantiated with a known ceiling (or floor, for the vertically mirrored case) and left edge. The left edge is given an interval pointing back to the cell from which it was opened and a zero-length wall interval just below it. Once the new cell is entered, rule 2 applies. The robot is next to an free-space interval with a known endpoint (known due to the short wall interval), so it retreats completely into the new cell and moves in $-y$ until it is next to an unknown portion of the left edge. It will

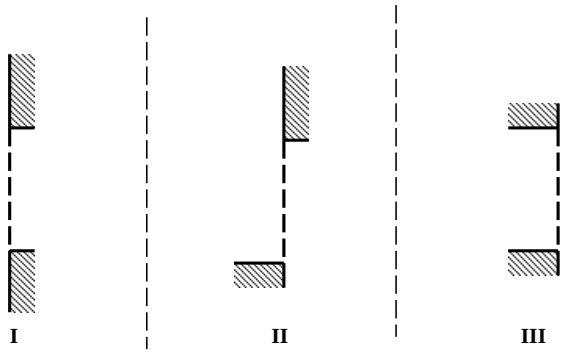

Figure 11: Types of placeholders (dashed lines) to be opened

then make contact with the left edge, extending the zero-length wall, at which point exploration can continue as described in Sec. 3.2. Once this edge is completely explored, the cell will be well-opened and seed-sowing can begin.

Type III: Only the location of the left edge of the cell is known. The left edge will be explored upward, starting from a zero-length wall interval created upon instantiation. The exploration will continue to the ceiling of the cell, creating placeholders as necessary, as described above. Once the ceiling is reached, the robot will move to the unexplored area below the original cell entry and explore to the floor. Once the robot reaches the floor of the new cell, it will have a known and explored left edge and a known floor and ceiling, and will be well-opened.

3.5 Path planning

We have now shown that from any initial condition, CC_R will generate no more than two well-opened cells, each of which will be completed. This completion will also generate placeholders where necessary and at most one well-opened cell, which will itself be immediately completed (since it must be the robot's current cell). However, to ensure complete coverage, once all opened cells have been completed, all placeholders must be opened and completed as cells, one at a time.

When CC_R finds itself in a completed cell without an incomplete cell to go to, a candidate placeholder H_d is selected as a destination and a path is planned to it. This path can always be generated because H_d will be adjacent to a cell, and the adjacency graph of the cells is connected, so a path can always be found from the current cell to the placeholder in that graph. It is also necessary that the same path be planned starting from each cell on the path, since CC_R will stop in each cell to decide its next move. For this reason, the path is planned using depth-first search from the placeholder to the current cell. The robot then moves from cell to cell with a series of linear motions from one cell edge to the next. Finally, before the last move of the path is taken (but after

the decision to make that move), H_d is instantiated as a cell to ensure that the robot is within a cell when that move is complete.

4 Conclusion

Sensor-based coverage has many potential applications in robotics, and although some algorithms have been written that perform this task for common situations, this work seeks to address the specific case of robots with only contact sensing. The algorithm developed, CC_R, performs coverage using only contact sensing in rectilinear environments. It has been proven to be complete for a large class of environments, and work has begun on extending this class to include all rectilinear environments. In addition, CC_R was designed to be extended to general polygonal environments without great changes in the overall structure. Based on this general design, preliminary work has begun on an algorithm CC_P for a circular robot with a ring of contact sensors operating in a polygonal environment. Investigation has also begun into applying CC_R to robots working in a team in such a way that efficiency of coverage is greater than if each robot works separately. Finally, CC_R has been implemented in simulation and shown to work and be reasonably efficient (to the human observer) and work is underway to implement CC_R on an actual robot system.

References

[1] M. Held, *On the Computational Geometry of Pocket Machining.* Springer-Verlag, Berlin, 1991.

[2] A. Pirzadeh and W. Snyder, "A unified solution to coverage and search in explored and unexplored terrains using indirect control," in *Proc. of IEEE Int'l. Conf. on Robotics and Automation*, pp. 2113–2119, April 1990.

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

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

[5] H. Choset. Personal communication, 1998.

A CC_R: Detailed Description

A pseudocode description of CC_R is given in this section to supplement the summary description and proof given earlier. First of all, some definitions are given in addition to those at the beginning of Sec. 3:

- `Known`(C_i,dir) if $C_{i_x,dir} = C_{i_n,dir}$
- `Finite`(C_i,dir) if $0 < |C_{i_x,dir} - C_{i_n,dir}| < \infty$
- `Explored`(C_i,dir) if the intervals on the (dir) side span the edge from floor to ceiling
- `Coveredto`(C_i,dir) if C_c has been covered up to the (dir) edge
- v is the direction of the trajectory that has just ended
- C_c is the cell robot's "current" cell (which gets set only by the map interpreter)

CC_R operates by performing the following at each event:

- If a collision has just occurred:
  - If not(`Known`(C_c,v)):
    - If traveling in y and the robot is not completely within C_c, add a zero-length wall interval just past collision point.
    - Otherwise, it must be the edge of C_c, so set it in C_{c_n} and C_{c_x}.
    - If traveling in the x direction:
      - If a strip is in progress, move it into C_c so it just touches the edge of C_{c_x}.
      - If the new edge has a placeholder neighbor that is now outside C_{c_x}, move it to the appropriate edge of C_c.
  - If traveling in y and the collision is at the known ceiling (or floor) of C_c, add the area of the current strip (if one exists) to the covered area of C_c.
  - If traveling in y and the collision is not at the known ceiling or floor of C_c:
    - If the robot is building a placeholder, finish it and add a zero-length wall interval just past the collision point.
    - Otherwise, add a new placeholder H_{m+1} with known ceiling or floor (due to collision) and neighbor C_c and add an intervals in C_c pointing to H_{m+1}.
  - If traveling in x and `Known`(C_c,v), if there is a wall interval within $w/2$, extend it to include p_y, otherwise create a new zero-length wall interval at p_y.
- Otherwise, a non-collision event occurred (the trajectory completed):
  - If traveling in y and edge of the robot past the floor or ceiling of C_{c_x}, instantiate a cell C_{n+1} around p with width w and an interval pointing to C_c. Also create an interval in C_c and appropriately set the edge of C_{c_n} and C_{c_x} shared with C_{n+1} (as shown in Fig. 9).

- If traveling in x and edge of the robot past the known side edge of C_c:
  - If there is an interval in the v-side edge within $w/2$ of p_y corresponding to a piece of free space, extend that interval to include p_y, and if necessary, extend the cell or holder corresponding to that interval.
  - Otherwise, (there is no nearby interval) check to see if there is another cell C_j adjacent to C_c at this location.
    - If so, then there should be a placeholder there. Point the interval in C_j to C_c instead of the placeholder and create an interval in C_c the length of the placeholder that is pointed to by C_j.
    - Otherwise (the new free space is unexplored), create an interval in C_c at p_y and create a placeholder at $(p_x \pm \frac{w}{2}, p_y)$, both zero length.

- Consistency checking of C — for each cell C_j $(j \neq c)$:
  - If C_{c_n} and C_{j_n} overlap in y, then:
    - If C_c is to the left of C_j, make sure $C_{c_n,right} \leq C_{j_x,left}$ and $C_{j_n,left} \geq C_{c_x,right}$.
    - Vice versa if C_c is to the right of C_j.
  - If C_c and C_j share a known ceiling and floor and abut each other left to right, extend C_c to include the area of C_j including its covered area, and replace the intervals of C_c on the side shared with C_j with C_j's intervals from the other side.

- Then choose a new direction/distance by the first applicable rule [rules with a † are evaluated twice — first with the † representing "right", then with it representing "left"]:

  1. If `Finite`(C_c,†):
     - If no †-side interval contains p_y, move in † direction until collision.
     - If p_y is contained by a †-side wall interval, move in † direction until collision.
     - Otherwise, move to the closer end of the †-side interval containing p_y, unless that side is at the floor or ceiling of C_c.
  2. If `Known`(C_c,†) and not(`Explored`(C_c,†)):
     - If no †-side interval contains p_y, move in † direction until collision.
     - If p_y is contained by a †-side wall interval, move a small amount in $+y$ if the †-side intervals do not extend to $C_{c_x,ceil}$, otherwise move a small amount in $-y$.
     - If p_y is contained by a †-side free space interval:
       - If the interval has a known endpoint on its near side and the robot's edge is outside C_c, move in x to bring the robot inside C_c
       - Otherwise, move in y toward the unknown end of the interval.

3. If not(`Known`(C_c,ceiling)), move in $+y$ until collision
4. If not(`Known`(C_c,floor)), move in $-y$ until collision
5. If not(`Known`(C_c,†)) or not(`Coveredto`(C_c,†)) [seed-sowing]:
   - If not near $C_{c_x,floor}$ or $C_{c_x,ceil}$ move in y to whichever is closer.
   - Otherwise, if necessary, move in x to end up w beyond the †-most edge of the covered portion of C_c.
   - If x motion not necessary, move in y away from the ceiling/floor for a distance just greater than the height of C_c.
6. If C_0 is incomplete, plan a path to it, using depth-first search of the adjacency graph of cells, searching from C_0 to C_c. Move into the next cell on the path.
7. If C_c has an incomplete neighbor, move in to it, instantiating it as a new cell if it is a placeholder
8. If there is any other active placeholder, plan a path to it as described in rule 6 and move into the next cell on the path.