Enveloping Multiple Obstacles with Hexagonal Metamorphic Robots

Jennifer E. Walter
Vassar College
walter@cs.vassar.edu

Joy W. Kamunyori
Vassar College
jokamunyori@vassar.edu

Abstract

The problem addressed is reconfiguration planning for a metamorphic robotic system composed of any number of hexagonal robots when multiple simple obstacles are embedded in the goal environment. Simple obstacles are defined as obstacles that have even surfaces with no pockets or indentations and that, as a group, form no narrow corridors or isolated areas.

We extend our earlier work on filling multiple pockets in an obstacle to the case where the goal may contain several simple obstacles. In this paper, we present algorithms that determine how many obstacles are in the goal by logically grouping the obstacle cells in the goal into distinct connected components, order them lexicographically from north-west to south-east, and then link them by shortest path bridges to form one large obstacle with multiple pockets. We subsequently use techniques presented in our earlier papers to envelop the composite obstacle.

Index Terms

Metamorphic robots, hexagonal robots, obstacle, pocket, distributed reconfiguration

I. Introduction

A *self-reconfigurable* [6] robotic system is a collection of independently controlled, mobile robots, each of which has the ability to connect, disconnect, and move around adjacent robots. *Metamorphic* robotic systems [3], a subset of self-reconfigurable systems, have the following additional requirements:

- Each module (robot) is required to be identical in structure, motion constraints, and computing capabilities.
- Modules have a regular symmetry so that they can densely pack the plane to form two and three dimensional solid lattices.
- Modules are not independently mobile. They require a substrate lattice in order to achieve locomotion.

Shape changing in these composite systems is envisioned as a means to accomplish various tasks, such as structural support or tumor excision [8], as well as being useful in environments not amenable to direct human observation and control (e.g., interplanetary space, deep oceanic environments). The complete interchangeability of the robots provides a high degree of system fault tolerance, allowing the potential for system "self-repair" [7].

The motion planning problem for a self-reconfigurable robotic system is to determine a sequence of robot motions required to change the shape of the system from a given initial configuration (I) to a desired goal configuration (G).

We focus on a system composed of planar, hexagonal robotic modules as described by Chirikjian [4]. Our strategy for motion planning begins with a centralized planning phase. The actual reconfiguration is distributed and relies on the assumption of global knowledge of the coordinates of cells in G, local contact information at the modules, and that I is a straight chain configuration that intersects G. We have previously applied this approach to the problem of reconfiguring a straight chain to

Kamunyori supported by Vassar College and the CRA-W Distributed Mentorship Program.

an intersecting straight chain [13] or to a goal configuration that satisfies a general "admissibility" condition [12]. In [11], we presented algorithms and heuristics to choose *admissible substrate paths* for module traversal that result in fast reconfiguration. Informally, an admissible substrate path is a sequence of contiguous cells that span the goal configuration in an east to west direction. This path forms a barrier between goal cells to the north and south, allowing efficient module traversal without collision or deadlock. In [10] we presented a definition of directionally-oriented *admissible traversal surfaces*, along with algorithms to plan and perform reconfiguration when a simple obstacle (i.e., one with no indentations or "pockets" in its surface) is embedded in G.

We presented and analyzed algorithms to fill a single obstacle pocket in [9], and algorithms to find and fill multiple obstacle pockets in [14].

The problem we address in this paper is completely enveloping multiple simple obstacles within the goal environment. This problem builds on our previous work by using algorithms previously developed to accomplish this. The multiple obstacles problem is more complex than the single obstacle problem because the obstacles must be grouped together in such a way that they form a single mass within the goal over which modules can traverse during goal-filling. Within this paper, we present algorithms to determine how many obstacles are embedded in G, and then to bridge the obstacles together to form a single large, complex obstacle. We then treat the composite obstacle like a multi-pocket obstacle, using the techniques presented in [14] to fill the pockets.

A. Related work

The problem of enveloping obstacles in the goal environment with metamorphic robots has been addressed by some researchers. Chirikjian [3], for example, proposed a heuristic to attract deformable hexagonal modules to an obstacle in the goal configuration, causing the modules to converge around the obstacle as part of a centralized reconfiguration strategy. Bojinov et al. [1] provide a distributed strategy for grasping objects in the environment using rigid rhombic dodecahedral modules by probabilistically "growing" extensions to envelop the obstacle.

Other work addresses the locomotion of metamorphic robots over irregular traversal surfaces with "hills" or stair-like structures. Butler et al. [2] present a rule set for distributed locomotion of layers of deformable cubic modules over obstacles on the traversal surface. Hosokawa et al. [5] consider obstacles that form a surface such as a stairway to be traversed by rigid square modules.

B. Our approach and problem definition

Our overall objective is to design a distributed algorithm that will cause the modules to move from an initial straight chain configuration, I, in the plane to a known goal configuration, G. This algorithm should ensure that modules do not collide with each other, and the reconfiguration should be accomplished in a minimal number of rounds.

In this paper, we consider how to perform reconfiguration when the goal environment envelops multiple obstacles that do not have pockets in their surfaces, and that do not form narrow corridors or isolated areas when grouped together. An obstacle is a sequence of one or more "forbidden" cells that modules cannot enter. The goal environment has the following admissibility constraints:

- There is a minimum of two cells between the obstacles contained within it,
- The goal is "simple," i.e. it has no pockets or indentations, and

- There exists a shortest path between obstacles that does not go outside the goal

In Section II we describe the system assumptions. Section III gives an overview of our earlier work, and Section IV describes how that work is extended to accomplish the more complex problem of enveloping multiple obstacles. We present results of successful envelopments of multiple obstacles in simulation experiments in Section V. Finally, Section VI provides a discussion of our results and future work.

II. System model

The plane is partitioned into equal-sized hexagonal cells and labeled using the same coordinate system as described by Chirikjian [3].

A. Assumptions about the modules

- Each module is identical in computing capability and runs the same program.
- Each module is a hexagon of the same size as the cells of the plane and always occupies exactly one of the cells.
- Each module knows at all times:
  - its location (the coordinates of the cell that it currently occupies),
  - its orientation (which edge is facing in which direction), and
  - which of its neighboring cells is occupied by another module.

Modules move according to the following rules.

1) Modules move in lockstep rounds.
2) In a round, a module M is capable of moving to an adjacent cell, C_1, iff (cf. Fig. 1)
   (a) cells C_1 and C_2 are currently empty and
   (b) module M has a neighbor S that does not move in the round (called the *substrate*) and S is also adjacent to cell C_1.
3) Only one module tries to move into a particular cell in each round.
4) Modules cannot carry, push, or pull other modules, i.e., a module is only allowed to move itself.
5) Modules are deformable and move by a combination of rotation and changing joint angles.
6) Moving modules never come into contact during a round.

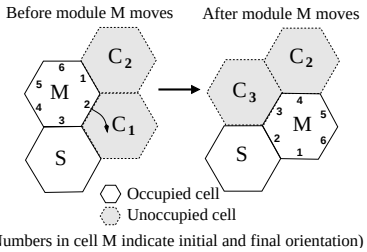

Fig. 1. Before and after module movement. S is substrate module and M is moving module. Numbers in M depict orientation of M before and after move.

III. Reconfiguration planning overview

Motion constraints on the robots dictate the necessity of having at least two contiguous free sides facing the direction the robot will rotate. Another constraint on movement is imposed by our distributed algorithm, which forbids modules from moving if they determine locally that a move may partition the configuration.

Fig. 2 shows the contact patterns of modules that are *free* to move, those that are *blocked* from moving by motion constraints, and those that may cause *partitioning* of the system.

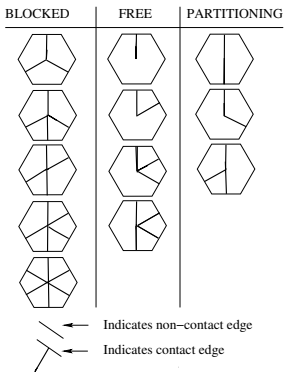

Fig. 2. Contact patterns for modules and pocket cells.

Our results in [13] and [10] show that maximum concurrency without deadlock can be ensured if moving modules are separated by two empty cells while moving over a surface that satisfies the properties of an admissible traversal surface. Informally, a north-facing (south-facing) traversal surface is *east-monotone admissible* if

- modules will move exclusively from west to east over the surface,
- every clockwise (CW) or counter-clockwise (CCW) rotation is in a non-westerly direction, and
- modules separated by two empty cells can move over the surface without creating a deadlocked configuration.

Deadlock avoidance in our algorithm requires that vertical columns in the traversal surface that are traversed on the east side are separated from vertical columns that are traversed on the west side by at least three empty cells. We showed in [12] that our algorithm for reconfiguration will successfully fill any goal configuration that contains an east-monotone admissible traversal surface, also known as an *admissible substrate path*.

A. Reconfiguration planning with one obstacle in G

An obstacle is a sequence of one or more "forbidden cells" that modules cannot enter. We consider obstacles that are composed of hexagons of the same size as the cells of the plane and we assume each hexagon in the obstacle occupies exactly one of the cells in the plane. Any indentation or "pocket" in the obstacle surface is assumed to contain goal cells. Informally, any goal cell found on a straight-line path between the center points of non-adjacent obstacle cells o_i and o_j, such that the straight-line path is normal to one side of both o_i and o_j and passes through only non-obstacle cells, besides o_i and o_j, is a *pocket cell* (cf. Fig. 3). Pocket cells that are adjacent to non-pocket goal cells are said to be on the pocket opening.

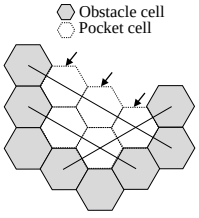

Fig. 3. Pocket cells within obstacle. Arrows point to pocket opening cells.

If the obstacle pocket contains only cells with *free* contact patterns we call the obstacle *admissible*. The *free* contact patterns in Fig. 2 represent non-occupied pocket cells with a sufficient number of contiguous non-occupied sides to allow module entry without partitioning the pocket. Inadmissible obstacles have pockets that contain cells with blocked or partitioning contact patterns or have pockets that are contained completely within the obstacle. It is noteworthy that we use *the same contact patterns during reconfiguration to locally distinguish a movable module as are used in the planning stages to determine whether an obstacle pocket can be filled*.

Our method of enveloping an obstacle containing a single pocket in G requires the following planning steps:

(a) Choose an admissible substrate path that links the westmost column of G to the westmost obstacle column.

(b) Based on the choice of substrate path from part (a), choose a cell in the westmost column of G as an intersection point for the straight chain of modules in I and position I so that it is collinear with the substrate path.

(c) Determine the order in which modules will fill the obstacle pocket.

(d) If the obstacle/pocket mass has any east-facing vertical sections after the coordinates have been ordered, determine the coordinates of goal cells that will be filled by modules to extend the obstacle so that it tapers gradually to the east.

(e) Choose an admissible substrate path that links the eastmost column of the extended obstacle to the eastmost column of G if necessary.

(f) Determine the order to fill cells to the north and south of the substrate path/obstacle mass in vertical columns from east to west.

The above planning steps were the subject of [11], [12], [10] and [9].

B. Filling multiple obstacle pockets

The formal algorithms for filling multiple pockets in an obstacle were presented in [14]. Therefore we will only give a brief, informal description in this paper. As in the single pocket case, each pocket cell in an admissible obstacle with multiple pockets must have a free contact pattern (cf. Fig. 2). To fill multiple obstacle pockets, the pockets are counted, and then the order in which pockets should be filled is determined, as well as the direction in which modules filling each pocket should be traveling.

Fig. 4 shows the order in which the cells in a triple-pocket obstacle are filled by our algorithm when modules choose a rotation direction (CW or CCW), depending on the shortest distance over the obstacle surface.

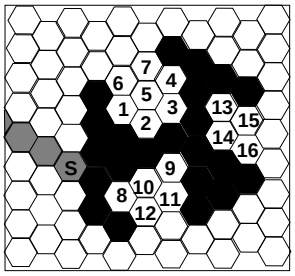

Fig. 4. Obstacle with pocket cells numbered in the order they would be filled by modules sequentially filling pockets.

In [14] we also presented algorithms to fill pockets concurrently, but we will not be dealing with that in this paper.

IV. Reconfiguration Planning For Enveloping Multiple Obstacles

The approach to enveloping multiple obstacles within a goal environment involves finding the shortest-distance paths between obstacles and using these to form "bridges" between the obstacles, creating a composite obstacle. The obstacles must be simple (i.e. no pockets), and must meet the two-cell clearance condition. The composite obstacle is then treated as a multi-pocket obstacle, using pocket-filling techniques to envelop it.

Our method for enveloping multiple obstacles is as follows:

1) Count the number of obstacles within the goal environment.
2) Order the obstacles lexicographically from north-west to south-east. The purpose of ordering is to ensure that obstacles are dealt with from west to east of the goal.
3) Build the shortest substrate path from the west end of the goal to the first obstacle in the ordering, i.e. the obstacle that is furthest north-west in the goal. This first obstacle becomes the initial composite obstacle.
4) Find and save the shortest distance path between any obstacle in the composite and the next obstacle in the goal, and then add this obstacle to the composite. Repeat for as many obstacles as there are in the goal.
5) The obstacles and the shortest distance paths between them now form a large, composite obstacle. Treat this obstacle as a single obstacle with multiple pockets, using previously developed techniques.

Fig. 5 shows an initial configuration in which three obstacles are embedded in the goal environment.

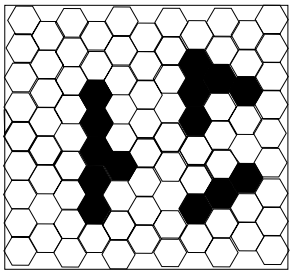

Fig. 5. Three obstacles contained within a goal environment, before the EnvelopMultipleObstacles algorithm is run.

A. Counting Obstacles

The obstacles within the goal are counted using the algorithm CountObstacles in Fig. 6. The algorithm uses breadth-first search to group obstacle cells together into their respective obstacles, retaining only the perimeter cells of these obstacles and discarding the inner cells. Initially, all obstacle cells are in vector *obstacleCells*, and vector of vectors*obstacle* is empty.

B. Ordering Obstacles

The algorithm OrderObstacles in Fig. 7 sorts the cells in each obstacle lexicographically by their x-coordinates, and then sorts the obstacles by their x- and y-coordinates. At end, obstacles are ordered from north-west to south-east. Obstacles are in the vector of vectors *obstacle*.

C. Finding Lattice Distance

The lattice distance between two cells in the plane is found using the Manhattan (also known as 'Taxicab') metric for finding distance within a grid. The algorithm Distance in Fig. 8 finds the lattice distance between 2 cells, using this metric, given

```
Algorithm CountObstacles
 1.  i := 0
 2.  While (obstacleCells is not empty)
 3.      x := obstacleCells(1)
 4.      i++
 5.      remove x from obstacleCells
 6.      Q.enqueue(x)
 7.      While (Q is not empty)
 8.        u := Q.dequeue()
 9.        For each neighbor v of u : v ∈ obstacleCells
10.          Q.enqueue(v)
11.          remove v from obstacleCells
12.        End for
13.        If u has one or more free sides
             add u to obstacle(i)
14.      End while
15.  End while
```

Fig. 6. Pseudocode for Algorithm CountObstacles.

```
Algorithm OrderObstacles(obstacle)
 1.  For every i in obstacle(i)
        Sort obstacle(i) by x-coordinates in ascending order
        Let x(i) := lowest x-coordinate in obstacle(i)
 2.  End for
 3.  Sort obstacle by x(i) values in ascending order
     (where i:= 1,2...  size of obstacle)
 4.  For every group of obstacle cells with equal x(i)
       Sort by y-coordinates in ascending order
 5.  End for
```

Fig. 7. Pseudocode for Algorithm OrderObstacles.

their x- and y-coordinates. The Manhattan metric is tailored for square grids, so it is modified slightly in our algorithm to account for the fact that we are dealing with a hexagonal grid. Unlike a square grid, within which any two non-parallel straight lines that begin in the center of, and are normal to the sides of, the cells of the grid meet at a right angle, such lines would meet at either an acute or obtuse angle in a hexagonal grid.

Let x and y be two cells within the hexagonal grid. Consider a straight line, starting at the center point of x and normal to one side of the cell in either the north-east or south-east direction. Another straight line starting at the center point of y and normal to one side of the cell in either the north or south direction will intersect with the first one at a cell in the same column as y. Let this cell be z. The algorithm works by finding z and determining the angle formed by a path from x to y through z. For every acute-angle path between x and y, there is a shorter obtuse-angle or straight-line path. This shorter path is always of the same distance as that between x and z. This is due to the numbering convention of the hexagonal grid, which must be accounted for when finding lattice distance.

```
Algorithm DISTANCE(a, b)
 1.  x1 := x-coordinate of a; x2 := x-coordinate of b
     y1 := y-coordinate of a; y2 := y-coordinate of b
 2.  If cell b is north or south of cell a
 3.    dist := y2 - y1 - 1
 4.  Else if b is east of a
 5.    dist := x2 - x1 - 1
 6.  Else if b is north-east of a or b is south-east of a
 7.    Find the intersecting cell that is straight NE or SE (as appropriate) of a
       and straight north or south of b.  ycoord := y-coordinate of intersecting cell
 8.    If b is in the adjacent column to a
 9.      dist := ycoord - y2
10.    Else if intersecting cell is cell b
11.      dist := x2 - x1 - 1
12.    Else if the angle formed is acute
13.      dist := x2 - x1 - 1
14.    Else if the angle formed is obtuse
15.      If b is north-east of a
16.        dist := (x2 - x1) + (ycoord - y2 - 1)
17.      Else if b is south-east of a
18.        dist := (x2 - x1) + (y2 - ycoord - 1)
```

Fig. 8. Pseudocode for Algorithm DISTANCE.

D. Finding Paths

Once the shortest lattice distance between two obstacles in the goal is found, a path corresponding to this distance is found. Paths between the obstacles in the goal are found using the algorithm FINDPATHS in Fig. 10. This algorithm works by adding all cells on the straight line from $cell1$ to the $intersecting$ cell to the path. If the angle formed between $cell1$ and $cell2$ through the $intersecting$ cell is obtuse, all cells in the straight line between the $intersecting$ cell and $cell2$ are added to the path. However, if the angle formed is acute, another $intersecting$ cell, which forms an obtuse angle with $cell2$, is found and all cells from $cell1$ to this cell, and then to $cell2$ are added to the path. It is necessary to avoid acute angles in the path in order to avoid cell blockages during module movement.

Fig. 9 shows the paths found between the three obstacles in Fig. 5 after running the FINDPATHS algorithm.

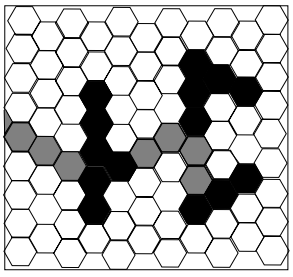

Fig. 9. Obstacles from Fig. 5, with paths between the obstacles to form a composite obstacle as found by algorithm FINDPATHS.

```
Algorithm FINDPATHS(cell1, cell2, intersect)
 1.  If cell2 is east of cell1
 2.    Add all cells in straight horizontal line from cell1 and cell2 to path
 3.  Else if cell2 is north or south of cell1
 4.    Add all cells in straight vertical line from cell1 and cell2 to path
 5.  Else if cell2 is north-east or south-east of cell1
 6.    Add all cells in straight diagonal line from cell1 to intersect to path
 7.    If angle is obtuse
 8.     Add all cells in straight vertical line from intersect to cell2 to path
 9.    Else if angle is acute
10.     Backtrack to find new intersect that forms and obtuse angle
11.     Add all cells in straight diagonal line from intersect to cell2 to path
```

Fig. 10. Pseudocode for Algorithm FINDPATHS.

V. SIMULATION RESULTS

We developed an object-oriented discrete event simulator to test the performance of our multi-obstacle enveloping algorithm on a number of different obstacles embedded in G. For every admissible goal environment tested, all obstacles were enveloped successfully.

VI. CONCLUSIONS AND FUTURE WORK

We have presented algorithms to plan reconfiguration of a system of hexagonal metamorphic robots when multiple simple obstacles are embedded in the goal environment. The algorithms in this paper build on and extend our earlier work by increasing the set of obstacles that can be completely enveloped during reconfiguration.

Our future work involves enveloping complex obstacles, thereby relaxing the restriction that all obstacles be simple.

REFERENCES

[1] H. Bojinov, A. Casal, and T. Hoag. "Emergent structures in modulular self-reconfigurable robots." In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, Vol. 2, pages 1734–1741, 2000.

[2] Z. Butler, K. Kotay, D. Rus, and K. Tomita. "Generic decentralized control for a class of self-reconfigurable robots." In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 809–816, May 2002.

[3] G. Chirikjian. "Kinematics of a metamorphic robotic system." In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 449–455, 1994.

[4] G. Chirikjian, A. Pamecha, and I. Ebert-Uphoff. "Evaluating efficiency of self-reconfiguration in a class of modular robots." *Journal of Robotic Systems*, Vol. 13, No. 5, pages 317–338, May 1996.

[5] K. Hosokawa, T. Tsujimori, T. Fujii, H. Kaetsu, H. Asama, Y. Kuroda, and I. Endo. "Self-organizing collective robots with morphogenesis in a vertical plane." In *IEEE Intl. Conf. on Robotics and Automation*, pages 2858–2863, May 1998.

[6] K. Kotay and D. Rus. "Motion synthesis for the self-reconfiguring molecule." In *IEEE Intl. Conf. on Robotics and Automation*, pages 843–851, 1998.

[7] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji. "M-TRAN: Self-reconfigurable modular robotic system." *IEEE/ASME Trans. on Mechatronics*, Vol. 7, No. 4, pages 431–441, 2002.

[8] A. Pamecha, I. Ebert-Uphoff, and G. Chirikjian. "Useful metrics for modular robot motion planning." *IEEE Transactions on Robotics and Automation*, 13(4):531–545, 1997.

[9] J. Walter, M. Brooks, and N. Amato. "Filling an obstacle pocket with hexagonal metamorphic robots." Submitted Sept. 2003.

[10] J. Walter, B. Tsai, and N. Amato. "Enveloping obstacles with hexagonal metamorphic robots." In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, to appear, Sept. 2003.

[11] J. Walter, B. Tsai, and N. Amato. "Choosing good paths for fast distributed reconfiguration of hexagonal metamorphic robots." In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, pages 102-109, May 2002.

[12] J. Walter, J. Welch, and N. Amato. "Concurrent metamorphosis of hexagonal robot chains into simple connected configurations." *IEEE Transactions on Robotics and Automation*, Vol. 18, No. 6, pages 945-956, 2002.

[13] J. Walter, J. Welch, and N. Amato. “Distributed reconfiguration of metamorphic robot chains.” In *Proc. of ACM Symp. on Principles of Distributed Computing*, pages 171–180, 2000.

[14] J. Walter, M. Brooks, D. Little, N. Amato. ”Enveloping Multi-Pocket Obstacles with Hexagonal Metamorphic Robots”