Mobile Robot-Assisted Cellular Environment Coverage

Georgios Siamantas, Konstantinos Gatsis, Antony Tzes

Mobile Robot-Assisted Cellular Environment Coverage

Georgios Siamantas, Konstantinos Gatsis and Antony Tzes

University of Patras, Electrical and Computer Engineering Dept.,
Rion, Patras, Greece
gsiama@upatras.gr,tzes@ece.upatras.gr

Abstract. The robotic coverage problem of a known rectangular cellular environment with obstacles is considered in this article. The robot can move only at directions parallel and perpendicular to the sides of the rectangle and can cover one cell at each time unit. A suboptimal minimum-time complete area coverage path-planning algorithm is presented. This algorithm minimizes a distance cost metric related to its current position and the quadtree-based decomposed blocks of the unexplored space. The efficiency of the suggested algorithm is shown through simulation studies and the implementation of a non-linear controller designed for a two-wheeled robot to achieve tracking of reference trajectories. For the implementation of the controller, the localization of the robotic vehicle was necessary and it was achieved via image processing.

Key words: Robotics, Control, Planning

1 Introduction

Robotic path-planning algorithms tackle the problem of finding a path in an environment that enables a robot to visit or sense with its sensors as much as or all if possible of this environment. Many applications of such algorithms appear in the literature [6]. In the present work we consider a robot placed inside a cellular environment with obstacles. We seek to find a path that minimizes the time that takes the robot to visit each cell of this unexplored environment. The entry and exit points of this path can be different cells. The optimal solution for the suggested l_1-distance cost metric rectilinear environment is NP-hard [5].

Similar works were presented by Zelinsky [3] and Gabriely and Rimon [8]. Zelinsky used the conventional wavefront algorithm to determine a coverage path (distance transform coverage-DTC). In DTC the starting and ending (goal) robot positions are specified in order to create an expanding field of increasing distance values from the end position extending to the whole grid. The robot then finds a cover path from start to goal performing a "pseudo-gradient ascent", on the distance field. Gabriely and Rimon describe a Spanning Tree Coverage (STC) algorithm. The STC works not on the original cellular environment but on a coarsen grid which is a tool-based approximation of the work-area and achieves a covering path in linear time $O(N)$, where N is the number of cells comprising

the approximate area. The DTC method is very well suited for applications where we want to create a covering path in $O(n)$ time (n the grid area) and finish in a specific cell in the grid. But it does not guarantee that this path will be as close to the o as possible. In this paper the proposed coverage method produces close to optimal paths most of the time, overcomes the need of specifying a goal position as required in the DTC algorithm and works directly in the original cellular environment grid in contrast to the STC algorithm.

The paper is organized as follows. The model of the environment, the robot motion and associated metrics and methods are described in Section 2. The proposed algorithm is provided in Section 3 followed by simulation results and robot experiments in Section 4, while conclusions are drawn in the last Section 5.

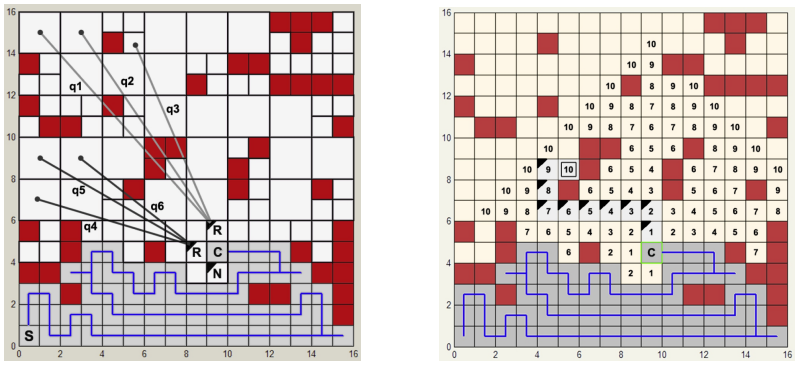
2 Problem statement

2.1 Preliminaries, Concepts

We consider a cellular environment to be a rectangular grid of n x n dimensions where $n = 2^k$, $k > 0$. The grid cells can be empty (free) and thus possible for the robot to visit them or occupied by an obstacle and thus inaccessible to the robot. The topology of the grid is dictated by the number and positions of the obstacles inside the grid. Each grid cell has coordinates of the form: (i, j), where $1 \leq i \leq n$ and $1 \leq j \leq n$. The robot can move under the Manhattan or l_1-metric motion model i.e. parallel or perpendicular to the sides of the rectangle and only one grid cell at each time unit, thus the set of actions (moves) that the robot can make is $U = \{(0,1),(1,0),(0,-1),(-1,0)\}$ where the robot's position is changed at the kth time instant to $X^{k+1}(i,j) = X^k(i,j) + U$. Each empty cell has from 1 to 4 neighbor free cells depending on adjacent obstacles and on whether is located at a corner (1-2 neighbors), an edge (1-3 neighbors) or inside the grid (1-4 neighbors). Each empty cell is accessible from any other empty cell in the grid, i.e. obstacles do not form closed regions of empty space inside the grid. The following notation is used in the remaining of this article: C the set of grid cells, C_{free} the set of free grid cells, C_{obs} the set of cells occupied by obstacles, C_{feas} the set of cells that the robot can visit from the current position, C_{vis} the set of already visited cells, C_{unvis} the set of remaining unvisited cells, $X_{i,j}$ the grid position at coordinates (i, j), X_{cur} the robot's current position coordinates, X_{next} the robot's next move position coordinates, X_I the robot's initial position coordinates.

Under the above environment formulation the minimum-time full area coverage path must intersect itself as less times as possible if not at all. One known way [7] to achieve this (minimizing intersections) is to execute a serpentine like motion consisting of straight moves followed by 180 degree U-turns, the so called boustrophedon (ox turning-like) motion and try to avoid visiting already visited space. In order to achieve as much as possible boustrophedon motion in a known topology environment is to ensure that at each path step the sum of the distances from the next robot position to as many as possible landmarks in the

environment is maximized. An equivalent to this maximization criterion is to minimize the sum of the distances from all the other possible move positions except from the current pick position, e.g. if the robot can move to four adjacent cells then pick one possible move at a time and measure the distance of the other three possible move cells to the landmarks and then move to the cell with the minimum sum of these distances. We used the minimization criterion instead of the maximization because in the l_1 metric cellular environment, the simulation studies indicated a reduced sensitivity when dealing with moves that are equidistant from the landmarks. In our environment formulation all the possible robot positions are countable, so it is possible to sum the distances from the next position to all free cell positions in the grid. Although this is possible it is time consuming for large grid sizes. An equivalent criterion is to decompose free cell space to blocks and then sum the distances from the next position to some cell inside these blocks. The distances should be weighted according to the size of each block to ensure that the mean block distance is approximately the same with that of summing the distances of all the cells inside the block. The concept is shown in Figure 1(a). If the current robot position is in cell C and we choose to move to cell N the two other possible moves are to cells R. We compute the sum of all distances q_1, q_2 etc. from cells R to the decomposed free space blocks and repeat the same computation for each of the 3 possible moves. The criterion to move to the next cell is decided by the one that makes the sum of these distances minimum.



(a) (b)

Fig. 1. (a) Sum of distances computation (b) The propagation and retrace phases.

2.2 Quad-tree decomposition

An n x n grid matrix G is defined such as:

$$G(i,j) = \begin{cases} 1 \text{ if } X_{i,j} \in C_{obs} \text{ or } X_{i,j} \in C_{vis} \\ 0 \text{ if } X_{i,j} \in C_{unvis} \end{cases} \quad (1)$$

where the grid cells that are considered obstacles and the already visited cells (soft obstacles) take value 1 and all other unvisited cells take value 0.

In this paper the Quad Tree Decomposition (QTD) algorithm is used for decomposing the environment grid matrix into rectangular blocks of similar value cells. The QTD is a well established method [4] of decomposing an image into blocks that meet some criterion of homogeneity and has been used extensively in robotics [9]. The motivation for decomposing the grid matrix into blocks is to reduce the number of cell distance calculations by some constant factor. The quad tree method decomposes the grid matrix into two parts. The first part is a tree structure that shows the location and size of each homogeneous region. The second part specifies the cell value at each leaf node of the tree, representing if it is an obstacle, visited space or unvisited space.

Quad tree decomposition is based on the successive subdivision of the grid matrix into four equal-sized quadrants. If the matrix does not consist entirely of 1s or entirely of 0s, then it is subdivided into quadrants, sub-quadrants, and so on, until blocks are obtained that consist entirely of 1s or entirely of 0s; i.e. each block entirely contains cells that belong to C_{obs} and C_{vis} or entirely contains cells that belong to C_{unvis}. The grid quad tree can be characterized as a variable resolution data structure. QTD is an $O(n)$ complexity process, where n is the grid area [9]. The worst case for the QTD in terms of distance calculations occurs when the topology of the grid corresponds to a checkerboard pattern. The amount of distance calculations required is a function of the resolution (i.e., the number of levels in the quad tree), the number of obstacles and visited cells, and their relative positioning inside the grid. Thus, when calculating the sum of distances instead of dealing with all the free grid cells we have to deal only with the nodes of the QTD, a substantial reduction.

2.3 Shortest Path Computation

In order to measure distances in the l_1-metric grid we used an algorithm proposed by [2] that it is widely used in routing two terminal nets in circuit board design problems. This algorithm has $O(n)$ time and space complexity, where n is the area of the grid matrix and guarantees of finding an optimal solution if one exist. The used algorithm consists of two phases [1]. The propagation phase is based on breadth-first search. The search resembles a wave propagating from the starting cell. The start cell is labeled '0' and the wave front propagates to all the unblocked vertices adjacent to it. Every unblocked cell adjacent to the start cell is marked with a label '1' corresponding to its l_1 distance from start. Then, every unblocked cell adjacent to the cells with distance 1 is marked with a label '2' because its distance from the start cell is 2, and so on. This process continues until the target cell is reached or no further expansion of the wave can be carried out. An example of the algorithm is shown in Figure 1(b). Due to the breadth-first nature of the search, this algorithm is guaranteed to find a path between a source and a target cell, if one exists. In addition, it is guaranteed to be the shortest path between these cells [1]. The second phase is the retrace phase where from the target cell we follow back the cell labels in descending distance

order till we reach the starting cell. In our formulation for the sum of distances computation the propagation phase is computed once in each robot's move. The retrace phase is not needed because we only need to find the minimum distance path between the starting and ending cells if such a path exists.

3 Environment Coverage

Let B the set of all QTD blocks, B_{free} the set of QTD blocks of cells with value 0, B_{occ} the set of QTD blocks of cells with value 1, $B_{i,j}$ the position of a block as the upper left cell position $X_{i,j}$ of this block, $A^B_{i,j}$ the area of a block $B_{i,j}$ given from the QTD. $d^{m,n}_{k,l}$ is the l_1-metric distance from $X_{k,l}$ to $X_{m,n}$ and is returned by the aforementioned in Subsection 2.3 algorithm. If there is no path connecting $X_{k,l}$ to $X_{m,n}$ because of the presence of obstacles and/or cells that are already visited then the implemented algorithm returns a very large value to bias the next move properly.

In order to choose which move to select the following quantity is defined:

$$D^{m,n}_{i,j} = \sum_{\forall X_{k,l} \in \{C_{feas} \setminus \{X_{i,j}\}\}} d^{m,n}_{k,l}. \tag{2}$$

This quantity reflects the definition of the sum of the distances from some cell $X_{m,n}$ to all the other possible next moves except the one for which we compute the sum.

Similarly, let:

$$\Delta_{i,j} = \sum_{\forall B_{m,n} \in B_{free}} A^B_{m,n}\, D^{m,n}_{i,j}, \tag{3}$$

where $\Delta_{i,j}$ is the sum of the previously defined D distances to all the free block positions scaled by each block's area as explained in section II.

If the set M is defined as:

$$\{\ \Delta_{i,j} \mid X_{i,j} \in C_{feas}\ \}, \tag{4}$$

then the position coordinates of the robot's next move in order to fulfill the minimization criterion of the proposed algorithm can be set as:

$$X_{next} = \arg \inf_{X_{i,j}} M. \tag{5}$$

In the algorithm *Grid_Cover* shown below function QuadTreeDecomp() performs quad tree decomposition on the given grid. Function FindUnvisitedCellBlocks() finds in the quad tree structure the QTD blocks of cells with value 0 i.e. that correspond to unvisited cells. Function FindAdjacentUnvisitedCells() finds the adjacent to X_{cur} unvisited cells that are not obstacles. Function GridIsCovered() checks if the grid is covered by the robot. Function FindNearestUnvisitedCellTo() finds the nearest to X_{cur} unvisited cell (even if it is necessary to pass over already visited cells) thus implementing a backtracking mechanism

that guarantees the grid's full coverage by the robot. Function FindMinDistancePathTo() finds a minimum distance path to that nearest cell using the suggested shortest path algorithm.

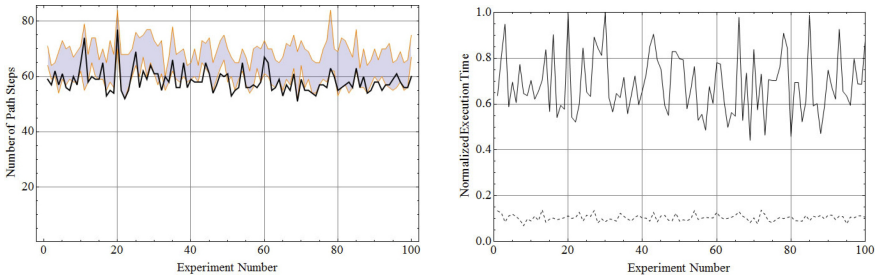
Algorithm *Grid_Cover*(G, X_I)
Input: A grid matrix G; the starting cell position X_I.
Output: An array *Path* with the covering path cell positions.
1. $X_{cur} \leftarrow X_I$
2. $S \leftarrow$ QuadTreeDecomp(G)
3. $B_{free} \leftarrow$ FindUnvisitedCellBlocks(S)
4. $C_{feas} \leftarrow$ FindAdjacentUnvisitedCells(X_{cur})
5. **if** sizeof(C_{feas}) = 1 **then**
   $X_{Next} \leftarrow C_{feas}$; **goto** 8.
6. **if** sizeof(C_{feas}) > 1 **then**
   With C_{feas} and B_{free} compute equations (2)-(4)
   Set X_{Next} according to equation (5); **goto** 8.
7. **if sizeof**(C_{feas}) = 0 **then**
   if GridIsCovered() = True **then**
   return *Path*
   else
   $X_{near} \leftarrow$ FindNearestUnvisitedCellTo(X_{cur})
   Path $\leftarrow$ *Path* + FindMinDistancePathTo(X_{near})
   $X_{Next} \leftarrow X_{near}$
8. Update *Path*, G, X_{cur} with X_{Next}; **goto** 2.

4 Simulations and Experiments

In order to evaluate the performance of the algorithm we define as MNP the minimum number of path steps, as ANP the algorithm number of path steps and as a performance index $PI_{opt} = MNP/ANP$. The numerator refers to the number of robot moves needed to cover optimally the whole grid in minimum time and the denominator refers to the number of moves proposed by the algorithm for full area coverage. So the grid will be covered optimally if PI_{opt}=1. Several tests were performed with different grid areas, obstacle topologies and robot start positions and the algorithm performed acceptably with $PI_{opt} > 0.87$ most of the time. The proposed algorithm shows $O(n^2)$ time complexity where n corresponds to the grid area. This is because wavefront propagation algorithms have an $O(n)$ time complexity and that, without taking QTDs into account, it takes at most $(n^2 + n)/2$ distance propagation related operations to cover the whole area. DTC computed for all possible end positions in the grid (full DTC) has also $O(n^2)$ time complexity.

In Figure 2 we see a comparison of the performance of the proposed algorithm with full DTC. In an 8x8 grid 100 simulation experiments were performed in MATLAB with 15 uniformly randomly distributed obstacles each time. In Figure

2(a) we see that the proposed algorithm (solid line) creates cover paths that have approximately the same number of steps as the minimum-step paths and considerably faster paths than the maximum-step ones created by the full DTC algorithm. Also in Figure 2(b) we see the execution times in the same experiment. The proposed algorithm (dashed line) was almost 7 times faster than full DTC.



(a) (b)

Fig. 2. Simulation results. The gray region represents the minimum and maximum path steps generated by full DTC.

In Figure 3 we see the Amigo differential drive two-wheeled robot that was used to test the proposed algorithm in real time. In order to do this a non-linear controller was implemented, based on [10], in order to achieve tracking of the required by the algorithm reference trajectories. For the implementation of this controller, the localization of the robotic vehicle was necessary and it was achieved via image processing during the experiments. The robot was recognized in images taken by a surveillance camera Figure 3(b), and after appropriate geometric analysis, it was possible to estimate the position and the orientation of the robot. In order to make it easier to recognize the robot's position and orientation inside the image, the robot's external surface was covered by white material and three small black spheres were placed on it, as shown in Figure 3(a). As shown in Figure 3(c) the robot followed the reference path computed by the algorithm with acceptable accuracy.

5 Conclusions

In this work a suboptimum complete area coverage algorithm is presented for rectangular l_1-metric grid environments. The start and finish positions may not be the same and the grid may have an arbitrary number of obstacles in arbitrary positions. The algorithm showed good performance in various grid topologies with acceptable time complexity taking into account its simplicity and hardness of the problem. The algorithm works on the original cellular environment and not on a coarsen approximation grid like the STC algorithm. It also guarantees the grid's full coverage by the robot because of its implemented backtracking

(a) (b) (c)

Fig. 3. (a) Robot in test environment (b) Camera (c) Reference and real robot paths.

mechanism. For comparison we performed various experiments with random topologies and saw that the proposed algorithm produced equally fast cover paths with the minimum-step paths produced by the full DTC algorithm but with considerably less computation time. The full DTC algorithm is an extension of the original DTC algorithm that we derived in order to compare it with the proposed algorithm. In the original DTC the goal position must be specified and produces cover paths with varying step sizes depended on this position. Finally the performance of the algorithm was validated in real time experiments with a differential drive two-wheeled robot moving according to the algorithm output in a laboratory test environment.

References

1. N. Sherwani, *Algorithms for VLSI Physical Design Automation*. Kluwer, 2002, ch.8.
2. C. Y. Lee, "An Algorithm for path connections and its applications," *IRE Transactions on Electronic Computers*, 1961.
3. A. Zelinsky, R.A. Jarvis, J.C. Byrne and S. Yuta, "Planning Paths of Complete Coverage of an Unstructured Environment by a Mobile Robot," *Proc. of Intern. Conf. on Advanced Robotics*, Tokyo Japan, November 1993
4. H. Samet, "Region Representation: Quadtrees from boundary codes," Comm. ACM 23 (March 1980), pp. 163-170.
5. X. Deng, T. Kameda, C. Papadimitriou, "How to learn an unknown environment I: the rectilinear case," *J. ACM 45(2)*, 1998, pp. 215-245.
6. H. Choset, "Coverage for robotics-A survey of recent results," Annals of Mathematics and Artificial Intelligence 31, 2001, pp. 113-126.
7. H. Choset, "Coverage of known spaces: The boustrophedon cellular decomposition," *Autonomous Robots 9*, Kluwer Academic Press, 2000, pp. 247-253.
8. Y. Gabriely and E. Rimon, "Spanning-tree based coverage of continuous areas by a mobile robot," *Annals of Mathematics and Artificial Intelligence* **31**: 77-98, 2001.
9. S. Kambhampati, L. Davis, "Multiresolution Path Planning for Mobile Robots," *IEEE Journal of Robotics and Automation*, Vol. RA-2, No. 3, September 1986
10. G. Klancar, D. Matko, S. Blazic, "Mobile Robot Control on a Reference Path". Proc. 13th Mediterranean Conf. on Control and Automation, Cyprus, June 27-29, 2005