Mobile robot navigation for complete coverage of an environment

Ana Šelek * **Marija Seder** * **Ivan Petrović** *

* *Department of Control and Computer Engineering, Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia, (e-mail: {ana.selek, marija.seder, ivan.petrovic}@fer.hr).*

Abstract: Inspired by the Spanning Tree Covering (STC) algorithm of Gabriely and Rimon, a novel algorithm of complete coverage for known environments is developed. Unlike the original STC algorithm, we detect dynamic elements of the environment and efficiently update the solution when changes are observed. The contribution is a path replanning algorithm that reduces overlapping when the part of the environment is changed. Experiments show that the proposed algorithm is capable of planning complete coverage robot paths with 70.27% coverage and without overlapping.

Keywords: Mobile robot, complete coverage, path planning, obstacle avoidance, path replanning

1. INTRODUCTION

Complete coverage path planning, which focuses on covering all reachable areas of the given environment, is one of important tasks for many industrial robots, especially demining, Dakulović and Petrović (2012), floor cleaning, Dakulović et al. (2011), mowing, Weiss-Cohen et al. (2008), and harvesting robots, Ollis and Stentz (1996). The most common representation of the continuous mobile robot environment is a two-dimensional occupancy grid map, which consists of the equally-divided grid of discrete cells. In the preliminary version of the Spanning Tree Coverage (STC) algorithm by Gabriely and Rimon (2002), a coarse grid map is used with completely free of obstacles cells of size of two diameters of the robot's footprint (the robot largest width). These cells are connected by the spanning tree around which the coverage path is created connecting twice as higher resolution subcells without visiting each sub-cell more than once. The coverage path planning algorithm that minimizes overlapping or coverage redundancy influences on improving the cleaning efficiency, Gao et al. (2008). The complete coverage D* (CCD*) algorithm by Dakulović et al. (2011) uses a high resolution grid map representation and have increased coverage rate but increased coverage redundancy. Most coverage path planning algorithms require complete information of the environment and plan the path that can not be changed if the part of the environment changes at some point during the coverage process. Hu et al. (2010) proposed the concept of "fictitious frontier" to tackle the problem imposed by unknown and moving obstacles. Waanders (2011) proposed the efficient update of the environment decomposition as the environment changes. Although there exist replanning algorithms for coverage path planning, the coverage redundancy is increased by the location of dynamic obstacles.

In this paper, our goal is to use ideas of previous algorithms to generate an approach that provides an effective solution to the problem of planning a coverage path in changing environments. Although the high resolution grid map enables higher coverage rate, we use coarse grid map like the STC algorithm to be efficient for large environments and to have minimal coverage redundancy. Since it is hard to cover all the area close to obstacles, the simple wall follower can be applied after the coverage process. We extend the STC algorithm to changing environments ensuring minimal number of coverage overlapping. We compare our results to the CCD* algorithm, which uses the high resolution grid map representation.

The rest of the paper is organized as follows. In Section 2, we describe the STC algorithm and the CCD* algorithm. In Section 3, we show how efficiently our algorithm generates the complete coverage path on the occupancy grid map and how the replanning algorithm works. The experimental results are given in Section 4. The conclusion is discussed in the last section.

2. RELATED WORK

A review and challenges of the most successful coverage path planning algorithms in the last decade can be found in Galceran and Carreras (2013) and Khan et al. (2017). Here we shortly present the work by Gabriely and Rimon (2003) and Dakulović et al. (2011).

2.1 The STC algorithm

Complete coverage path planning algorithm proposed in this paper is based on the STC algorithm by Gabriely and Rimon (2003). The continuous environment is approximated by a discrete grid of $2D$-size cells, where D is the size of the diameter of the circumscribed circle around the robot's footprint. The environment is populated by static obstacles. The robot must use its sensors

to detect obstacles during the coverage process because it does not have an a-priori knowledge of the environment. The first step in the coverage planning is to calculate a Depth First Search (DFS) spanning tree. This algorithm is recursive and uses the idea of backtracking. It involves the exhaustive search of all unvisited nodes by going ahead, if possible, else by backtracking along the visited nodes until a new unvisited neighbor is found. The resulting spanning tree will have a spiral shape, as can be seen in Fig. 1. Next, each cell is divided into four identical sub-cells of the size D. The robot follows a path that circumnavigates the incrementally constructed spanning tree using its on-board sensors. The goal is to cover every grid sub-cell while moving along a path. The robot follows the right side of the spanning tree until it reaches the end of the tree, which is called a leaf. At the leaf cell, the robot turns around to cross to the other side of the spanning tree. The coverage is completed when the robot returns to the starting cell. It's worth noticing that STC algorithm never visits the same sub-cell more than once and thus minimize the coverage time. However, such a recursive algorithm does not guarantee coverage completeness if obstacles are moving in front of the robot.

2.2 The CCD algorithm*

The CCD* algorithm proposed by Dakulović et al. (2011) is based on the D* search of the high resolution grid map of the environment. The robot is represented by the squared mask in the grid map, within which the robot's real footprint can be drawn. The D* search is called from the start node in the grid, resulting with the calculated path cost values g from each reachable node in the grid map to the start node accounting for obstacles in the environment, i.e., the occupied cells. The complete coverage path is produced by following the increase of costs g from the start node taking into account the squared mask of the robot to avoid overlapping. For visiting the overlapped but unvisited nodes, e.g. the corners of obstacles, an extra D* search is executed to compute the path to these nodes. The CCD* algorithm stops when there are no unvisited nodes. The replanning process is initiated when the new static or moving obstacle is detected by the robot's sensors. Then, the D* algorithm computes the new cost values g and the complete coverage path calculation is continued from the current node.

3. COMPLETE COVERAGE

The proposed complete coverage path algorithm is primarily focused on large environments. It is a modified version of the STC algorithm by Gabriely and Rimon (2003). Modifications enable efficient replanning of the coverage path if changes in the environment are detected. We named our algorithm as the Replanning Spanning Tree Coverage (RSTC) algorithm.

3.1 Spanning tree

We use the grid of $2D$-size cells created from the a-priori map of the environment, where each cell that contains the obstacle is occupied. Every free cell is subdivided into four equal area sub-cells of size D. Unlike the STC algorithm,

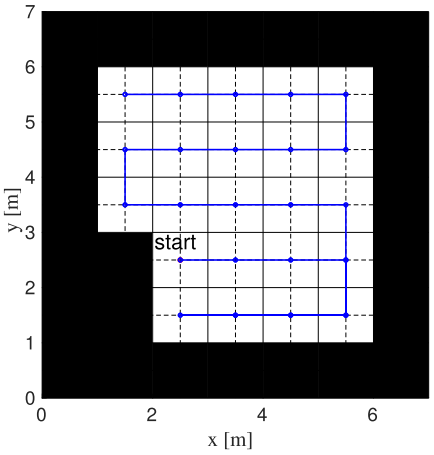


Fig. 1. Spanning tree example where the size of the sub-cell $D = 0.5$ m (robot's size). The starting cell is noted for the spanning tree construction.

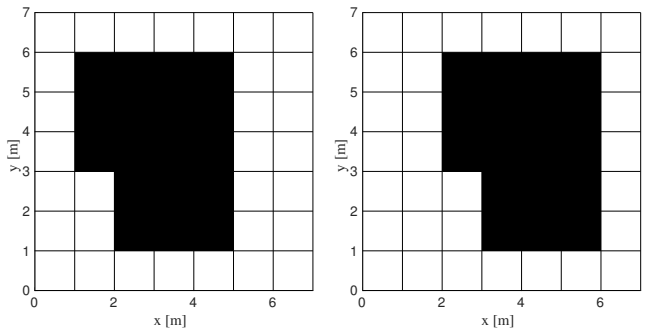
we do not calculate the spanning tree recursively as the robot moves and senses the environment, but instead, we calculate the spanning tree before the robot starts to move from the initial a-priori map of the environment. Figure. 1 shows an example of the grid map of $2D$-size cells and D-size sub-cells with the spanning tree calculated from the start node at (2,2). In our application the size of the cell is 1 m, so the indexes of the nodes correspond to the lower left coordinates in meters.

3.2 Path planning

After calculating the spanning tree from the starting cell, the calculation of the coverage path begins. The goal is to keep the robot circumnavigating around the spanning tree, always at the right side, until it completely covers all sub-cells and returns to the starting sub-cell.

The first step is to determine the direction of the spanning tree for every cell. We use four matrices for encoding each direction called the *north*, *south*, *east*, and *west* matrix. Each matrix contains fields with binary values, where a value is true if the robot can go in a certain direction from the current cell following the right side of the spanning tree, false otherwise. Branching of the spanning tree has at least three directions, and for each one, a certain matrix field is true. All leafs of the spanning tree have only one direction. Example of the direction matrices is shown in Fig. 2 for the spanning tree presented in Fig. 1. From the starting cell at (2,2) the robot is allowed to move only to the cell (3,2) because it is a leaf of the spanning tree and it needs to follow the right side of the spanning tree. So the field of the east matrix at (2,2) is set to true and the same field at all other matrices is false. This principle is applied for all the cells that are connected in the spanning tree.

Path planning starts from the starting sub-cell where the robot is at the beginning. To ensure that the robot follows the path of the sub-cells that circumnavigates around the constructed spanning tree, the angles of each branch need to be calculated. A branch is the straight line segment of the spanning tree, between two intersections of branches or orientation changes. The angle of the current branch



(a) east (b) west

(c) north (d) south

Fig. 2. Four direction matrices that determine the directions of the spanning tree at each cell in the grid map. An occupied field (true) in a matrix means that the robot can go in that direction in the spanning tree, while an unoccupied field (false) means the opposite.

is denoted as α and can be calculated from the direction matrices preferring always the right turn. The angle β is shifted for $-90°$ compared to the angle α and it is used to calculate path that is always on the right side from the spanning tree, see Fig. 3.

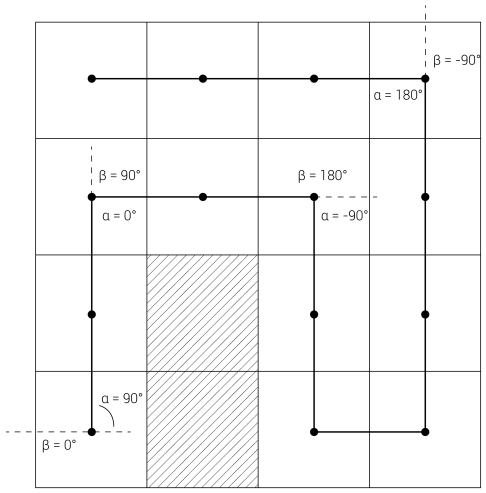


Fig. 3. An example of the spanning tree and the alpha and beta angles of some branches.

When angles α and β are known, the path can be determined. First, we check the direction matrices. If two or more matrices are true for the same field, the robot will go in the direction which ensures the counterclockwise circumnavigation around the spanning tree and does not cross over it. In order to avoid visiting already visited cells of the spanning tree, the binary matrix **visited_path**(i,j)={false, true} is used and its values are stored for each field. Initially, all fields are set to be non-visited (false).

The pseudocode of the complete coverage path is given by Alg. 1. The path is produced from the starting cell. The algorithm stops when the path reaches the start position. The robot's direction and the cell that it needs to visit next is known from the direction matrix and the spanning tree. We calculate angles α and β for each cell to ensure that the robot will follow the right side of the spanning tree. When one of the sub-cells is visited, the value of the field in the matrix **visited_path** is set to true. The path is saved as real coordinates for each cell distanced for $D/2$ from the branch to the right. If a dynamic change of the work-area is detected, the algorithm is interrupted. The field which is currently visited by the robot is set to non-visited in the matrix **visited_path**.

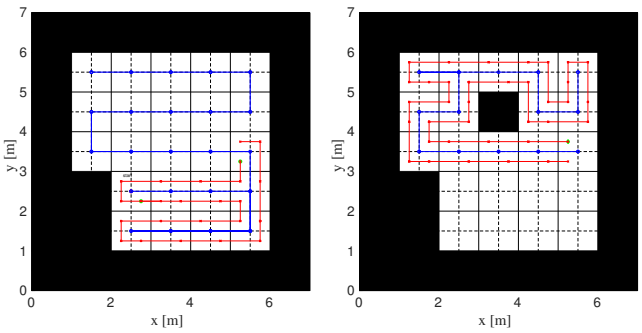
Algorithm 1 Pseudocode for the path planning

```
calculate branch α and β angles
determine starting position (x_s, y_s)
while (x, y) ≠ (x_s, y_s)
    Find new neighbors of the current cell which are connected with the spanning tree
    Go to neighbor cell (x, y) and set the field in the matrix visited_path to true
    x_p = x + D/2 cos(β)
    y_p = y + D/2 sin(β) (Save the path in real coordinates)
    if (dynamic = 1)
        End the path planning algorithm
    end if
end while
End the path planning algorithm
```

3.3 Path replanning

If the robot detects a dynamic change in the environment, the replanning algorithm is executed. Dynamic changes can be detected in neighboring cells of the current cell on the path, see Fig. 4a for an example. All visited cells are not considered as nodes for the new spanning tree calculation and the visited sub-cells are not part of the new path coverage.

After the new spanning tree is determined for the rest of the non-visited grid cells, the path planning algorithm is executed. The path always follows the right side of the spanning tree until the robot comes back to the cell in which replanning started, see Fig. 4b for an example. From this cell, the robot continues to follow the right side of the spanning tree which is already formed at the previous path planning. The robot goes through this visited grid cells by only visiting sub-cells which are not yet visited. The complete coverage algorithm ends when the robot comes to the starting sub-cell.



(a) before replanning (b) after replanning

Fig. 4. Replanning example with the spanning tree (blue), coverage path (red), and the start cell at (2,2). The replanning begins at (5,3), and the new obstacle is at (3,4).

3.4 Path following

The following of the computed coverage path is done by successive selection of subgoals which are set to the motion planner and then executed by the robot. These subgoals are chosen to be the points of the path direction change and have the goal heading in the direction of the next path straight line segment. The motion planning algorithm is used on a much finer occupancy grid map and it includes constraints on the robot motion. This algorithm can adapt to dynamic changes in the environment, Seder et al. (2017). If the selected subgoal is too close to the detected obstacle, a new subgoal is chosen in the local vicinity from the critical one. With this procedure, the robot follows the planned path with a minimal drift and has an additional collision checking performance.

3.5 Computing the coverage rate

Computation of the coverage information requires following parameters to be known: the total area of the environment A_t, the total area of the obstacles A_o and the total area covered by the robot's path A_p. From these three measurements the coverage rate percentage can be computed as:

$$\text{Coverage rate} = \frac{\mathrm{A_p}}{\mathrm{A_t} - \mathrm{A_o}} \cdot 100\%. \tag{1}$$

The total area can be found in the image as the number of all cells. If the image has width w and height h then the total area is A_t=wh. The total area of the obstacles can be found as a number of occupied cells on the map. The occupied cells have binary values which denote whether the robot has visited the cell or not. The total area of the robot path can be found by summing the number of these visited cells.

4. EXPERIMENTAL RESULTS

The proposed algorithm was implemented and tested in Matlab and then Robot Operating System (ROS). To execute the algorithm and get test results a 3D visualizer Rviz and a Pioneer 3DX mobile robot was used. A sick LMS200 laser range finder mounted on the robot was used for the environment perception. To illustrate the functionality of the proposed algorithm the results of the experiment are presented in Fig. 6. - 9. The robot needs to traverse the room completely with known static obstacle configuration (walls, tables, chairs, etc.) and some unknown obstacles that change the environment (humans or changed position of a chair).

Our algorithm works with a known environment, so the first step before executing the complete coverage is to capture the map and detect completely known static obstacle configuration (walls, boxes, etc.) We used sensors on the robot to collect data regarding the environment by the slam_gmapping ROS package. Once the environment had been explored the map was built and the amcl ROS package for localization in the built map was used. We processed the map additionally to align the corridors with the rectangular grid to have better performance of the coverage algorithm, see Fig. 5.

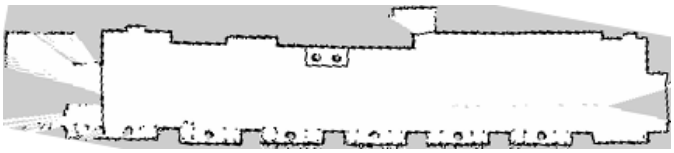


Fig. 5. Map of the environment for the experiment of the complete coverage, where white denotes the empty space, black the obstacles, and gray the unknown area.

4.1 Simulation results

Each static obstacle from the map in Fig. 5 is re-sampled to a lower resolution occupied grid cell. The size of the cells is 1×1 m and every cell is subdivided into four sub-cells of size 0.5×0.5 m which is the robot size. The coverage starts at the cell with the position (7,3). From that cell, the spanning tree construction algorithm is executed (see Fig. 6). The spanning tree is represented as a blue line which connects all center points of each free cell. The path circumnavigates around the constructed spanning tree and is represented by a red line. The robot follows the right side of the spanning tree until it reaches the leaf of the tree. At that cell, the robot turns around and continues to follow the right side of the other side of the tree. The coverage is completed when the robot returns to the starting cell. By following the path, the robot covers every sub-cell precisely once and travels a complete coverage path. For every cell, the algorithm checks all the neighbors of the next cell the robot needs to go. The replanning algorithm is called if the sensors on the robot detect any changes in the environment. This is represented in Fig. 7. The change in the environment is shown as an occupied cell at position (11,2). The replanning algorithm is executed at cell (9,3) and the new spanning tree and the path are recalculated for the non-visited cells, i.e., without the cells (7,3) and (8,3). After the new path covers the unvisited cells, the robot needs to go back to the cell from which the replanning algorithm started and continue from the right side of the spanning tree to the starting cell but only through the sub-cells which were not already visited.

Figure 8 presents how the planned path was followed by the robot in the simulator stage. A minimal drift from

the planned path can be seen, which resulted with the complete coverage of every sub-cell precisely once.

4.2 Results on a real robot

The experiment was done on the real robot Pioneer 3DX. To have a similar scenario to the simulation experiment, a chair was placed on the exactly same place during the coverage process. Comparing the simulation results in Fig. 8 and results on a real robot in Fig. 9, we see a much bigger drift in the actual path driven by the robot. In a real environment, we have a significant problem with the subgoal selection function. Since the sensor readings are noisy, and also the robot localization, then it can happen more frequently than the selected subgoal is too close to the obstacle and a new subgoal is chosen from the vicinity to avoid selecting the occupied one. Because of this, there are some deviations between the calculated (black line) and the real (red line) path in Fig. 9. There were also problems with imprecise localization of the robot at points where it rotates in place. So the robot deviates from the desired path at those points. The ground can also cause problems while following the path because of the slippage. We can see that at the cell (27,3) the subgoal was lost since the corners were occupied and the real path deviates from the calculated path significantly. In Fig. 8 and Fig. 9 the calculated path and the spanning tree from the starting cell to the cell where the replanning algorithm is executed are not plotted for clarity.

4.3 Comparison to the CCD algorithm*

The calculation of the CCD* algorithm on the same environment map is given in Fig. 10. The comparison of CCD* and our RSTC algorithm according to the complete coverage performance without changes in the environment is given in Tab. 1. The total area to be covered was 95.35m^2. The covered area with RSTC algorithm was 67m^2 and a coverage rate is 70.27%. With CCD* algorithm the covered area was 94.2m^2 and the coverage rate is 98.79 %. The coverage rate was calculated using (1). This percentage for RSTC algorithm is lower than using CCD* but can be increased if a wall following post procedure is applied. The advantage of RSTC algorithm is the length of the path for the complete coverage which is 138.63m. This is almost halved comparing to the CCD* algorithm. The biggest advantage of the RSTC algorithm is the overlap rate which is 0% and for CCD* is 88.58%. In Fig. 10 visited cells are noted by different colors according to the number of visits, so some parts of the map are visited nine times.

Table 1.

	RSTC	CCD*
Total trajectory length	**138.63m**	242.76m
Total area to be covered	95.35m^2	95.35m^2
Covered area	67m^2	**94.2 m^2**
Coverage rate	70.27%	**98.79 %**
Overlap rate	**0%**	88.58%

5. CONCLUSION

We presented a complete coverage path planning algorithm based on a low-resolution grid map representation and spiral motion, which can avoid changes in the environment. Because we only used completely free cells, the coverage path is optimal and it doesn't overlap at all. If we want a better coverage rate on a finer grid we can use a wall-following algorithm which will significantly increase the rate but also increase the overlapping. Further work will focus on solving the problem of a low coverage rate using partially-occupied cells and a wall-following algorithm.

6. ACKNOWLEDGEMENT

This work has been supported by the European Regional Development Fund under the project Development of a remotely controlled vehicle for operation in extreme CBRNe conditions (DUV-NRKBE).

REFERENCES

Dakulović, M., Horvatić, S., and Petrović, I. (2011). Complete Coverage D* Algorithm for Path Planning of a Floor-Cleaning Mobile Robot. *Proceedings of the Preprints of the 18th IFAC World Congress, Aug. 28-Sep. 2, Milano, Italy*, 5950–5955.

Dakulović, M. and Petrović, I. (2012). Complete coverage path planning of mobile robots for humanitarian demining. *Industrial Robot: An International Journal*, 484–493.

Gabriely, Y. and Rimon, E. (2002). Spiral-STC: An On-Line Coverage Algorithm of Grid Environments by a Mobile Robot. *IEEE international conference on robotics and automation, ICRA02*, 1, 954–960.

Gabriely, Y. and Rimon, E. (2003). Competitive online coverage of grid environments by a mobile robot. *Computational Geometry*, 24, 197–224.

Galceran, E. and Carreras, M. (2013). A Survey on Coverage Path Planning for Robotics. *Robotics and Autonomous Systems*, 61, 1258–1276.

Gao, X.S., Xu, D.G., and Wang, Y. (2008). Omnidirectional mobile robot for floor cleaning. *Chinese Journal of Mechanical Engineering*, 44.

Hu, G., Hu, Z., and Wang, H. (2010). Complete coverage path planning for road cleaning robot. *International Conference on Networking, Sensing and Control (ICNSC), Chicago*, 643–648.

Khan, A., Noreen, I., and Habib, Z. (2017). On Complete Coverage Path Planning Algorithms for Non-holonomic Mobile Robots: Survey and Challenges. *Journal of Information Science and Engineering*, 33, 101–121.

Ollis, M. and Stentz, A. (1996). First results in vision-based crop line tracking. *in Proc. ICRA96*, 951–956.

Seder, M., Baotić, M., and Petrović, I. (2017). Receding Horizon Control for Convergent Navigation of a Differential Drive Mobile Robot. *IEEE transactions on control systems technology*, 25(2), 653–660.

Waanders, M. (2011). Coverage path planning for mobile cleaning robots. *15th Twente Student Conference on IT , Enschede.*

Weiss-Cohen, M., Sirotin, I., and Rave, E. (2008). Complete coverage path planning of mobile robots for humanitarian demining. *International Conference on Computational Intelligence for Modelling Control and Automation.*

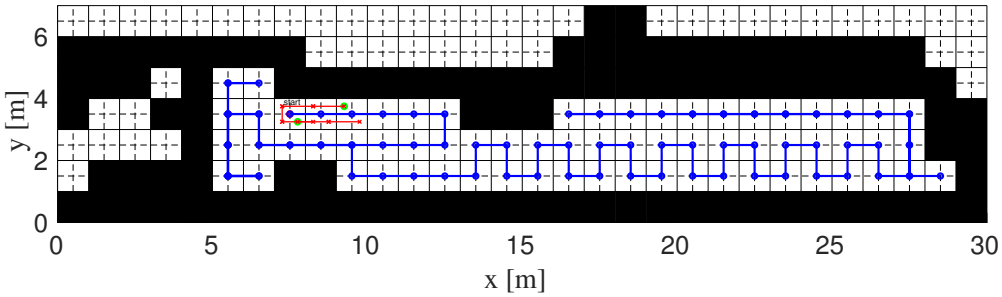


Fig. 6. The complete coverage algorithm execution before dynamic environment change at (11,2).

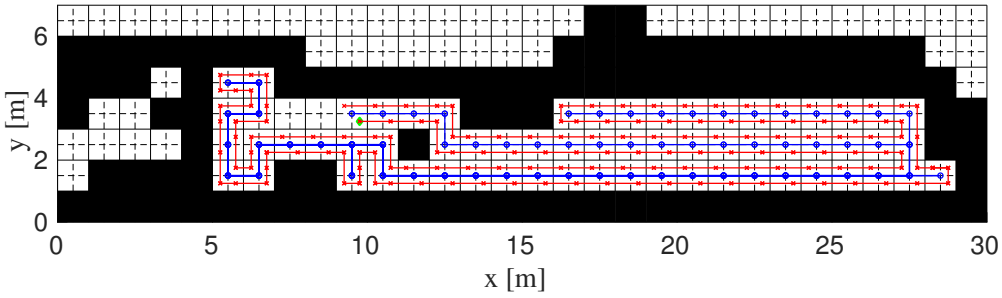


Fig. 7. Example of complete coverage algorithm execution after dynamic environment change at (11,2).

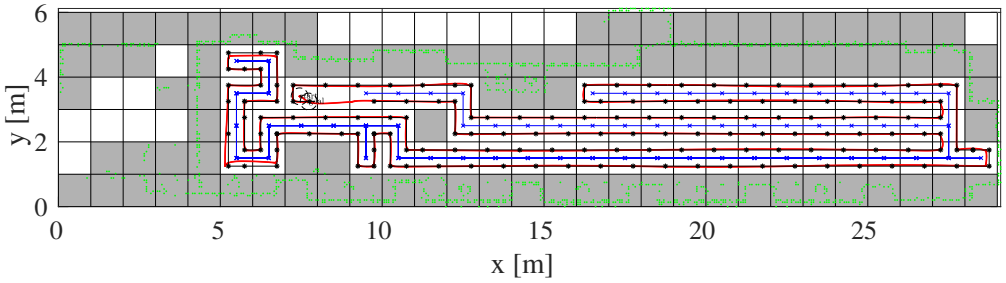


Fig. 8. The simulation of the complete coverage algorithm execution with dynamic change in the environment at (11,2).

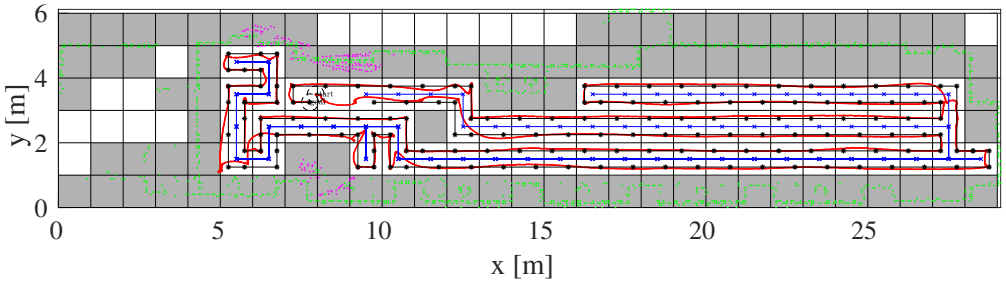


Fig. 9. Experiment of complete coverage on the mobile robot Pioneer 3DX.

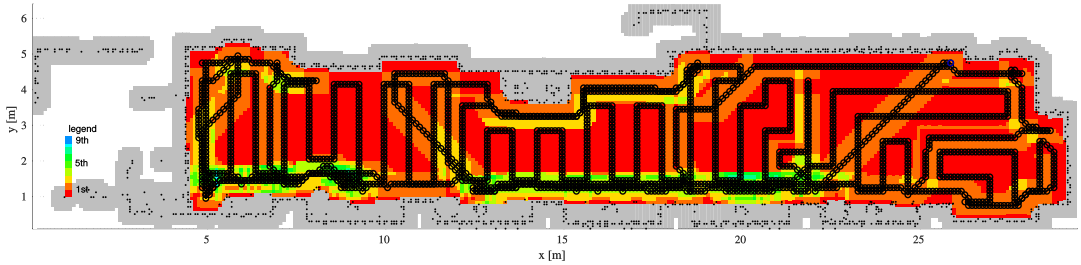


Fig. 10. The complete coverage of the CCD* algorithm with the path and noted redundant numbers of cell visits (from 1 to 9).