AN OPTIMIZED CLEANING ROBOT PATH GENERATION AND EXECUTION SYSTEM USING CELLULAR REPRESENTATION OF WORKSPACE

Qile He[1] and Yu Sun[2]

[1]Webber Academy, Calgary, Alberta, Canada
[2]California State Polytechnic University, Pomona, USA

ABSTRACT

Many robot applications depend on solving the Complete Coverage Path Problem (CCPP). Specifically, robot vacuum cleaners have seen increased use in recent years, and some models offer room mapping capability using sensors such as LiDAR. With the addition of room mapping, applied robotic cleaning has begun to transition from random walk and heuristic path planning into an environment-aware approach. In this paper, a novel solution for pathfinding and navigation of indoor robot cleaners is proposed. The proposed solution plans a path from a priori cellular decomposition of the work environment. The planned path achieves complete coverage on the map and reduces duplicate coverage. The solution is implemented inside the ROS framework, and is validated with Gazebo simulation. Metrics to evaluate the performance of the proposed algorithm seek to evaluate the efficiency by speed, duplicate coverage and distance travelled.

KEYWORDS

Complete Coverage Path Planning, Mobile Robots, Graph Theory.

1. INTRODUCTION

As the robot vacuum cleaner technology [1] makes its way into the homes of millions, a need arises to further increase their efficiency. Current robot cleaners often adopt a random walk approach to area coverage [2]. Indeed, given enough time the monte-carlo approach will eventually achieve full coverage of the cleaning area, but it is certainly suboptimal in terms of coverage duplication, coverage speed and coverage completeness. With the recent emergence of cleaning robots equipped with Lidar, it is now possible for the robot to map its environment and plan its path accordingly, to optimize for certain performance metrics. For the application of robotic vacuum cleaners specifically, reasonable metrics to optimize along include speed, and coverage completeness [3].

In order to attack the complete coverage path problem, a representation of the robot's environment is required. Usually, the working space of the robot is divided into small square cells, each assigned a value that indicates how likely, or in our case, whether, an obstacle is present [4]. This approach, termed the occupancy grid, allows a graph representation of the workspace, and correspondingly, path finding and search algorithms can be applied. However, the occupancy grid model as-is is deeply flawed for tasks such as cleaning. The occupancy grid method is discrete, when in reality the set of possible robot positions is continuous. This

David C. Wyld et al. (Eds): ACITY, DPPR, VLSI, WeST, DSA, CNDC, IoTE, AIAA, NLPTA - 2020
pp. 13-25, 2020. CS & IT - CSCP 2020 DOI: 10.5121/csit.2020.101502

discretization is likely to produce variation in performance between a map with orthogonal features and one with features not aligned to the occupancy grid orthogonally. Most existing research avoids this issue because they use mostly orthogonal features in their performance evaluation [5][6]. In addition to lack of consideration for non-orthogonal features, the robot's footprint is not considered at the path generation level. Existing research adopts the method of obstacle inflation, as in marking the neighbouring areas of an obstacle as obstacles also [7][8]. This method addresses robot footprint potentially overlapping with obstacles in real life, but does not address potential duplicate coverage caused by the robot being larger than the occupancy grid cell size. On the evaluation stage, many existing explorations evaluate their algorithms within the occupancy grid model. These papers focus on whether creating the shortest path as represented by the occupancy grid. However, the shortest path approach does not guarantee optimized traversal time. For example, two paths that have identical length in the occupancy grid can have very different traversal time in real life, due to one requiring frequent acceleration and deceleration.

While existing solutions focus on a shortest distance complete coverage path, this paper proposes a more complete solution that addresses various problems that arise when paths represented in the occupancy grid are applied to real-life environments. The effectiveness of this approach is evaluated in lifelike computer simulation of a robot cleaning task scenario. The first problem is that paths on the occupancy grid representation are an ordered collection of poses. In many existing explorations, the robot's footprint is not considered. However, for tasks like room cleaning, the footprint of the robot is crucial to create a path that not only visits all the points on the occupancy grid, but also one that visits all the points in the physical working space with as little duplication as possible. In many cases the occupancy grid cells are smaller than the robot footprint, and the generated path may cause duplicate coverage if it visits all the cells in the occupancy grid. This path planning algorithm accounts for the footprint of the robot. The algorithm is a modified version of Depth First Search (DFS) [15]. Instead of marking each traversed node in the graph representation as visited, the algorithm marks all nodes within the robot's footprint on the occupancy grid as visited. The center node of the footprint is the midpoint of the wheel axis in a differential drive robot. The algorithm will only write the center node's position at each step of the traversal into the path. The proposed algorithm can then recursively visit cells neighboring the center cell and mark the entire robot footprint surrounding that cell as visited, as long as the robot's footprint is clear from visited points or obstacles. When the modified DFS visits a cell that has no available neighbors, it finds the nearest unvisited cell and uses Dijkstra [16] to find a path to that block. Thus, the occupancy grid's resolution does not have to match the robot's size, and can be smaller by the robot's diameter by an arbitrary (ideally integer) positive number of times. With the robot footprint accounted for at the path-generation level, the proposed algorithm significantly reduces duplicate coverage caused by occupancy grid resolution being incompatible with actual robot size, saving time and energy. In addition to optimizations to reduce duplicated coverage, the proposed algorithm also takes into account the effects of target pose distancing on navigation. Given that unit error in heading leads to error in position proportional to distance, and that embedded controllers often have limited memory space and processing capacity to handle a high density of target poses, the target poses generated from the algorithm requires post processing. By pruning points in a straight line, the proposed algorithm increases smoothness of motion and reduces computational load.

Using Gazebo with a cleaning application scenario with complex geometries such as oblique walls, small pillars and acute angle corners, the path generation algorithm's performance is evaluated. A simulated Turtlebot3 Burger completes the planned path in this evaluation. After the path generation algorithm is run, the sum of the number of times nodes are repeated is calculated. The duplication rate can then be calculated by dividing the number of instances of duplicated visits by the total number of points in the path. This happens before the post-processing, and is

therefore a relatively fine-grained measure of duplicated coverage. During the robot run, performance metrics are collected. Total distance traveled is measured by summing the Euclidean distance between the current ground truth pose and the last ground truth pose. Time is also measured. With these two metrics speeds can be calculated. Speed reduction rate measures the extent to which the robot is slowed down by having to decelerate and accelerate, as opposed to operating near the top speed most of the time. This is calculated by dividing the empirical average speed by the maximum speed parameter of Turtlebot3 Burger. These factors are crucial to the effectiveness and satisfaction of robot cleaning products.

The rest of the paper is organized as follows: Section 2 gives the details on the challenges that we met during the design, implementation and evaluation of the algorithm; Section 3 focuses on the details of our solutions corresponding to the challenges that we mentioned in Section 2; Section 4 presents the relevant details about the evaluation of the path planner following by presenting related works in Section 5. Finally, Section 6 summarizes the results of this paper, as well as providing some insights on future works that could further enrich this paper.

2. CHALLENGES

To obtain a path generation and execution solution that is optimized for speed and coverage, a few challenges have been identified as follows.

2.1. Challenge 1: The Need of Optimized Path Post-Processing

DFS generates path points that are equally spaced apart. This creates many superfluous path points that lie on the same straight line. Removing these superfluous points creates significant savings in computation resources.

2.2. Challenge 2: The Complexity of Navigation Control

Traditional closed loop control creates curved trajectories for differential drive robots. This is problematic because the robot can run into obstacles unexpectedly. For maximum conformity to generate paths, the trajectory between any pair of path points needs to be a straight line. For equal angular speed, increased linear speed leads to increased disturbance in position. Therefore, the angular position control loop is scaled inverse to linear speed, such that deviation from the ideal trajectory is minimized.

2.3. Challenge 3: The Automation of the Occupancy Grid Transformation

The occupancy map model allows a high level of abstraction of the real-life environment such that existing algorithms can easily apply. However, translating the results of these abstracted algorithms into practice presents a set of problems. Firstly, gapping as implemented in ROS [12] has faulty coordinate transformation parameters. Therefore, manual calculation is required to set the parameters with which a transform between map coordinates and simulation world coordinates is conducted. Secondly, occupancy grid's limited resolution and poor representation of oblique lines and curves meant that it cannot be relied on for closely tracing the edges of obstacles. Instead, the obstacles as represented on occupancy grid must be inflated by the radius of the robot.

2.4. Challenge4: The Difference between Algorithm-Based Coverage and the Real Coverage

If coverage were to be measured by whether the robot has indeed visited all the points, as represented by the occupancy grid, it would almost always be 100%, by definition of a full coverage path. Therefore. measurements beyond the precision of the occupancy grid is required for a meaningful evaluation of robot performance. By approximating the robot trajectory as a series of short straight lines, the coverage area can be approximated as a series of quadrilaterals, with the four corners being the position of the left and right ends of the robot cleaning tool, at the beginning and the end of the straight line. As the sample rate of robot position increases, the accuracy of this approach increases. The four edges of each quadrilateral is represented by a linear inequality. This representation can then be operated on to calculate if an area has duplicate coverage.

3. SOLUTION

3.1. Overview

The proposed algorithm in this paper is implemented with Python. Before post-processing to reduce density of points in the path, the duplication rate of the path is measured. After a path is generated and pose-processed, it is stored in a YAML file. A Turtlebot3 Burger is simulated in Gazebo, integrated with ROS [13]. A ROS node serves as the navigation stack, by reading the path file and issuing velocity commands that take the robot to the current target pose by the shortest straight line before repeating the same process for the next.

3.2. Path Finding in Action

The simulations are carried out with a model of the Turtlebot3 Burger, a differential drive robot equipped with a laser ranger. Given that differential drive is commonly used in indoor cleaning robots, Turtlebot3 Burger is a good approximation of real-life hardware that relies on full coverage path planning.

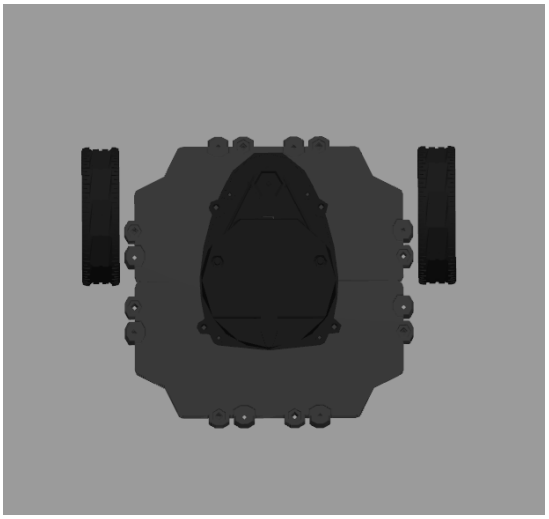

Figure 1: Turtlebot3 Burger

Robot Operation System, or ROS for short, is a framework that facilitates code reuse in robotics by packaging robot subsystem information in a distributed network. The network, or “ROS

runtime graph", consists of nodes broadcasting information, or "messages" about the robot, organized under different topics: heading, speed, sensor data, control commands, etc. Messages are shared through a subscription model, where a node broadcasts its messages independent of receivers. With ROS runtime network, robot subsystem code can be encapsulated, abstracted and reused. Within the ROS framework, a navigation node is written to carry out the planned paths.

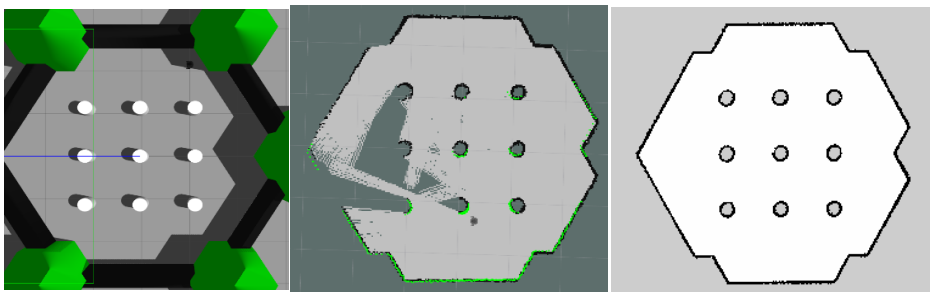

Figure 2: Robot workspace in Gazebo, Rviz LiDAR data view and occupancy grid view

While ROS provides a general framework of a robot system, Gazebo integrates with ROS to provide realistic simulations of real-life robot use cases. With realistic rigid-body dynamics, 3D graphics, and sensor noise generation, Gazebo is a reliable tool for evaluating robot software performance. Using Gazebo, Turtlebot3 Burger, and an example room, is simulated. Additionally, Gazebo messages provide a ground truth for robot position, with which many performance metrics, such as coverage completeness, coverage overlap or coverage time can be calculated.

Rviz is a ROS tool that visualizes ROS topics. Using Rviz, the actual path traveled by Turtlebot3 Burger is visualized.

In order to create a path, a priori knowledge of the robot's environment is required. Turtlebot3 Burger's laser rangefinder is used to carry out this task. When mapping the room, Turtlebot3 Burger travels in straight lines, and makes a turn at a random angle when its bumper sensors detect an obstacle. Given sufficient time, Turtlebot3 Burger can obtain a perspective on all areas in the room to provide a complete map. Meanwhile, odometry provides the relative position of the robot. With this relative position, as well as laser range sensor data, an occupancy grid is created using Gmapping from OpenSLAM. The occupancy grid is a 2D list of occupancy states (occupied, free or indeterminate), indexed by their positions.

3.3. Path Generation

Path planning algorithm is implemented in a python [14] script using Dijkstar library. Before generating the path, the script inflates obstacles and walls by a safety margin of 5 cells. After the inflation, the occupancy grid bmp file is read, and occupancy status of each tile stored in a 2-dimensional list. The robot is represented by a square block of tiles approximating its footprint. The center of this square block is the rotation axis of the differential drive chassis. A modified DFS is implemented, with the aim of having the center cell traverse the occupancy grid. At each move, the entire block of cells is checked for obstacles, such that every movement of the center cell is clear for the entire robot. The center cell's coordinates are then stored in a list, which would be the list of goal points. Meanwhile, the script marks the surrounding area of the visited center cell that is a radius of the block back in the goal points list as visited. Thus, each move of the center cell would not cause the robot footprint to intersect with a section already covered. When DFS runs into a block where no neighbors of the center cell may be added without running

into a visited cell, Dijkstra is used to find the nearest cell by Manhattan distance and generate a path to that cell. After a list of path points is generated, a function iterates through the list to check for collinear sections of goal points and returns a new goal points list that trims the excess points, while maintaining an arbitrary minimum density of goal points to prevent open loop behavior in the navigation stack. As the new goal points list is generated, the heading angle between each point in the path and the next point is calculated, and appended to the current point in the iteration. This list of goal poses is then formatted into YAML and stored.

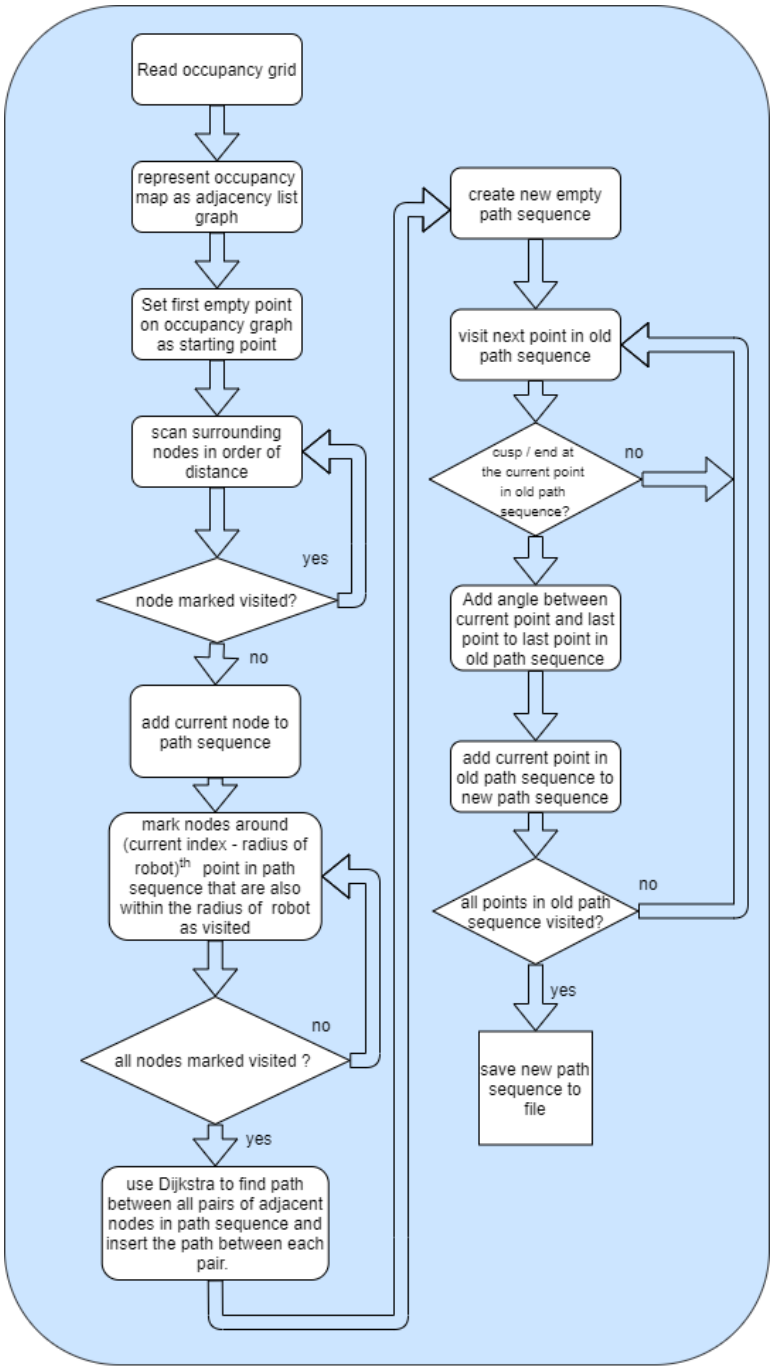

Figure 3: An Excerpt of the Path Generation and Cleaning Algorithm

3.4. Navigation

In order to determine how planned paths, perform despite imperfections induced by real-world physics, a navigation node is written to execute the planned paths in Gazebo. The navigation node reads the list of goal poses from the aforementioned YAML file. The node then calls a function that carries out the navigation between the robot's actual pose and the desired pose. When the robot reaches the vicinity of the goal pose, the navigation function exits and is called again with the next goal pose. Similar to real-life cleaning robots, absolute localization is carried out with odometry with the knowledge of robot initial pose. For each target the navigation node performs two operations. The first is to drive the robot from the initial position to the target, and the second is to rotate the robot to align with the orientation of the target pose. For the first step, the algorithm uses a proportional control loop to minimize two variables: heading error and position error. The target heading during this step is the angle between the initial position and the target position. When this parameter is minimized, only straight movement is required to reach the target. The angular speed in this step is proportional to heading error, and furthermore, to the inverse of linear speed. Because displacement for a given angular speed is proportional to linear speed, the heading adjustment angular speed is proportional to its inverse such that the heading adjustment behavior is consistent for varying distances between target and initial position. The linear speed is proportional to distance error raised to a power of larger than one, such that when the distance is below 1, the robot slows down and eventually stops.

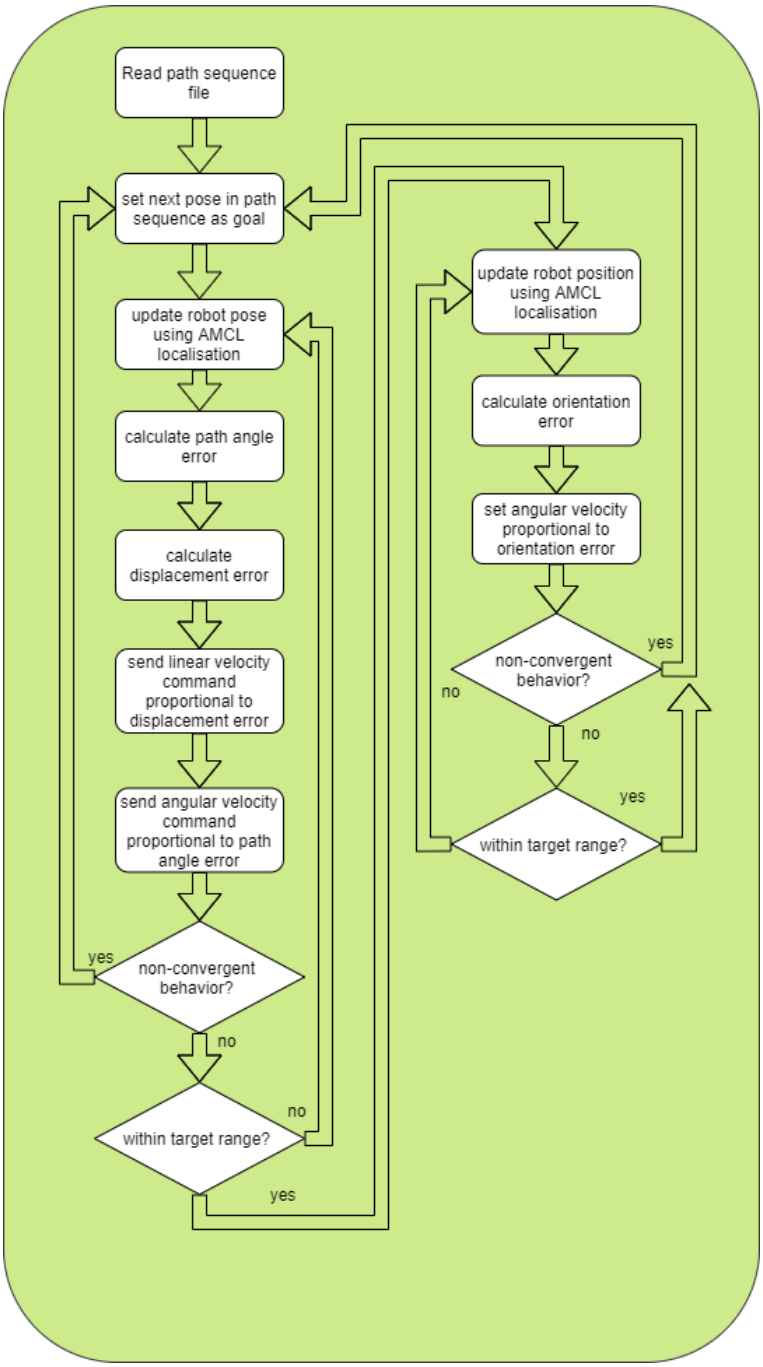

Figure 4: An Excerpt of the Navigation Algorithm

4. EXPERIMENT

The robot carries out the generated path within the Gazebo/ROS simulation environment. The navigation node takes in a desired pose and publishes commands to take the robot to the pose in a straight line maintained by closed loop control of heading. During the run, several pieces of data are calculated: area covered, area duplicated, distance travelled time taken, and poses not reached.

4.1. Occupancy Grid Efficiency

The most straightforward approach to measure the efficiency of the generated path is to utilize the generated points in path with the map dimensions. This shows a very accurate calculation on the coverage, as well as the duplicated points in the path. Even though the grid does not fully represent the actual coverage situation in reality, the efficiency result provides the evaluation from the algorithm foundation.

As shown in Table 1, the generated path only produces a 7.8% duplicated path while covering the 100% space.

Table 1: The Experiment Result of the Occupancy Grid Efficiency

Map	Total Pixels in Path	Duplication Instances	Duplication Rate
Indoor 1	9740	755	7.8%

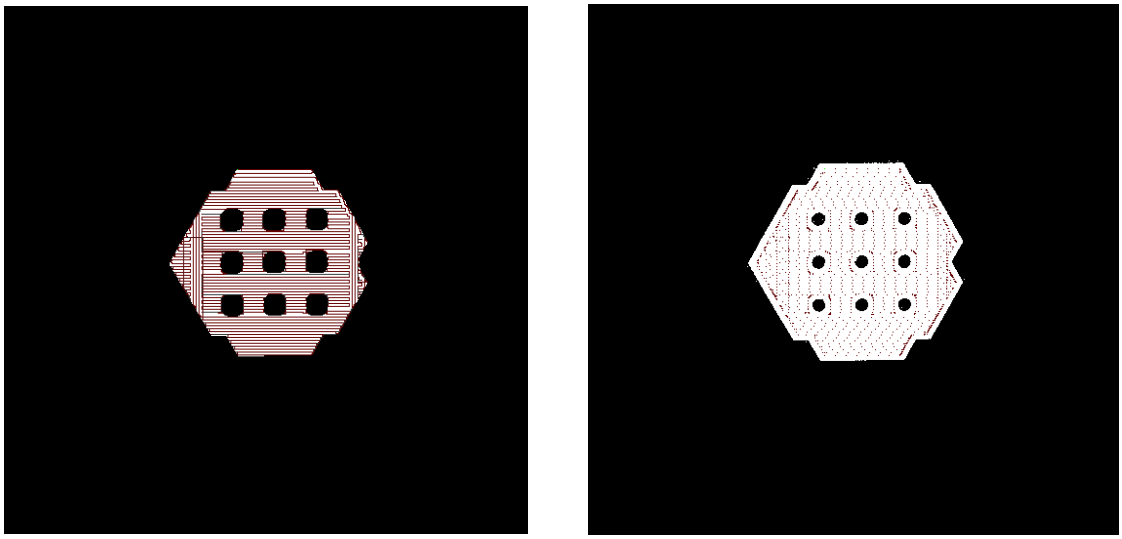

Figure 5: Generated path with duplicated blocks in gray (left) and target points after post-processing (right)

4.2. Simulation-Based Completeness

The area swept over subtracted by area duplicated yields the real covered area. The real covered area compared to the total free area on the map yields the real completeness of the coverage algorithm. This piece of data is obtained by a polygonal approximation of the area covered by the robot, and approaches a high accuracy as the robot position sample rate increases.

4.3. Simulation-Based Efficiency

The average speed of the robot is obtained by calculating the Euclidean distance over a traversal of points in the path sequence and dividing this distance by time taken. The average speed of the robot can be used to determine the amount of acceleration, deceleration and turns the robot goes through. The average speed as a percentage of the maximum speed represents how efficient the planned path is optimized for robot kinematics.

Table 2: The Experiment Result of the Simulation-based Efficiency

Map	Total Distance Traveled (m)	Total Time (s)	Average Speed (m/s)	Speed Reduction Rate
Indoor 1	241	3260	0.074	74%

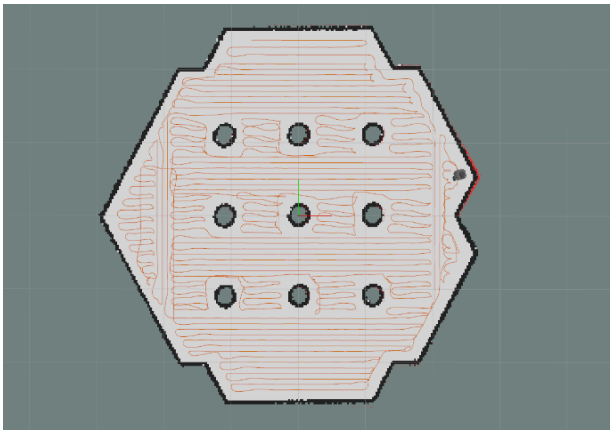

Figure 6: Robot trajectory during execution of generated path

5. DISCUSSION

5.1. Experimental Analysis

The generated path displays occupancy grid completeness by making sure that all points in the inflated occupancy grid are visited. It is also able to do this efficiently, with a duplication rate of 7.8%. It is worth noting that in real application, completeness can sometimes be sacrificed in favor of speed. Much of the duplicated coverage is caused by the robot travelling a long distance to visit one isolated cell. Given that it is difficult for humans to perceive the difference in cleaning results caused by the robot missing a number of small, isolated areas, coverage speed of this algorithm can be further improved by trimming points that are too costly to reach at little expense of completeness. The generated path follows a zig-zag scanning pattern at most places. The zig-zag pattern can be horizontal or vertical in orientation. More importantly, the proposed algorithm does not enforce scanning one row/column at a time, and is able to efficiently fill one local area bounded by obstacles for the most part and backtracking to another divided area. Given that our implementation of DFS weighs neighboring blocks on Manhattan distance, and selects the first unvisited block in the nearest equidistant set of neighbors, the zig-zag pattern is expected. If neighboring blocks during the search are weighted by other preferences, DFS may produce different patterns. For example, a wall- following spiral pattern can be encouraged by decreasing the cost of blocks that are near an obstacle.

The experimental validation shows that the proposed algorithm achieves complete coverage of the inflated occupancy grid even in environments that have a large number of oblique features. Using proportional control, the robot was able to follow the generated path to a precision such that there are no collisions throughout the test. Figure 6 shows the robot's trajectory in a full run. During this run, the robot averages a speed that is 74% of the maximum speed. While this number is subject to influences such as the particular robot's acceleration and deceleration capacity, it is true to Turtlebot3 specifically, due to Gazebo's ability to handle robot kinematic constraints according to manufacturer's specifications.

5.2. Related Works

Oh et al. attempts to create smoother movements by using a triangular cellular decomposition of the map [9]. We use the more traditional square cellular decomposition in the form of the occupancy grid, however, our methods of DFS path generation and duplicate coverage reduction does not depend on the shape of the cells, and can be extended to operate on an arbitrary connected graph as long as the geometric relationship between each node on the graph is defined (in a square cellular decomposition, neighboring cells are in the cardinal directions).

De Carvalho et. al uses a series of movement templates [10], including straight forward, turn and U-turns to carry out their planned paths. They seek to incorporate geometric constraints and kinematic characteristics into path planning for optimal speed. We incorporate the geometric constraint of the robot at the path planning stage by applying a robot shaped mask that marks its vicinity as visited during the path search. Additionally, linear approximation of the path point collection in this paper reduces unnecessary turning when covering and smooths out the path, thus reducing acceleration and deceleration.

Gonzalez et al. proposes Backtracking Spiral Algorithm, which generates a complete coverage path by spiraling from the perimeter of the environment inwards, and backtracking [5] to an unvisited region when the robot reaches the center of a local spiral. As DFS is a greedy algorithm, each path point is likely to be followed by its immediate neighbors on the occupancy grid. As a result, the proposed algorithm tends to continue in a straight line, and thus fills the map with zig-zag patterns. Whereas in the open space the spiral filling pattern is ostensibly better at reducing unnecessary acceleration, deceleration and turns, the superiority of the spiral filling pattern in this regard has not been established for all possible shapes of the workspace.

Khan et al. uses a zig-zag pattern to scan the map [6]. When the zig-zag scan leaves regions uncovered, the paper proposes two-way proximity search to backtrack to the border of the unscanned region. Whereas the above paper enforces scanning in an arbitrary direction, there is no enforced orientation of zig-zag filling in DFS.

Lee et. al improves upon the BSA approach by generating Bezier curves from the BSA path [11]. This is done to increase the smoothness of robot movement and reduce physical coverage time. This paper smooths out paths by constructing polylines from the collection of points in the path. Because DFS favors long straight paths, i.e. zig-zag patterns, the polyline method is more appropriate because it requires less angular acceleration for most of the path, whereas a Bezier curve would turn a set of points approximating a linear shape into a snaking path.

6. CONCLUSIONS AND FUTURE WORK

This paper proposes a novel path-motion planning solution for complete coverage in indoor differential robots. Cellular decomposition is implicitly applied when an occupancy grid is generated from SLAM. Using this cellular representation, an algorithm is used to visit all the cells in the occupancy grid. Beyond cell completeness, this paper seeks to maximize real coverage and reduce duplicate coverage by incorporating the robot's shape into path generation. The algorithm is tested in the ROS/Gazebo environment where the simulated robot carries out the path generated with the proposed algorithm. The robot carries out a test run in the test environment, during which data such as area covered, duplicate coverage and time are collected to validate the feasibility of this paper's proposal. The source of the completed implementation of the algorithm can be found at [17].

Some limitations exist in this paper's solution as-is. Firstly, this paper assumes complete a priori knowledge of the workspace, whereas real life applications may deal with unexpected obstacles. However, collisions can still be avoided using data from the cleaning robot's sensor suite. Moreover, real-time re-planning may be implemented simply by deleting inaccessible cells from the path sequence. Secondly, the a priori workspace knowledge must be obtained using specialized sensors, such as LiDAR or RGB-D stereo camera. Both sensors have seen limited application in indoor robots, but high price precludes these sensors from thoroughly penetrating the market of home cleaning robots. Thirdly, the current path is represented by a poly-line approximated from the cell visit sequence generated by the proposed algorithm. For an accurate execution of the poly-line representation, the robot must decelerate at each node in the poly-line to perform a real pivot turn. This may lead to extra time consumption. This problem can be partially solved by rounding the corners in the poly-line to reduce need for braking.

We would address the current solution's limited ability in dealing with unexpected obstacles by incorporating real-time re-planning capabilities. We would also direct our focus to further post-processing of the generated path for improved smoothness.

REFERENCES

[1] Robot vacuum cleaner, by T. Charles, A. Parker, S. Lau, E. Blair, A. Henginer, E. Ng, E. DiBernardo, R. Witman, M. Stout. (2006, Jan. 26). U.S. Patent 20060020369A1 [Online]. Available: https://assignment.uspto.gov/patent/index.html#/patent/search/resultAssignment?searchInput=20060020369&id=23224-384

[2] Random motion cleaner, by D. E. Gerber, K.L Thomas. (2002, Jun. 6). U.S. Patent 6571415B2 [Online]. Available: http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&p=1&u=/netahtml/PTO/srchnum.html&r=1&f=G&l=50&d=PALL&s1=6571415.PN.

[3] W. Brockman and D. Klingbiel, "Rating the Performance of Robotic Coverage Tasks," Proceedings of the 5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles July 5-7, 2004 Lisboa, Portugal, vol. 22, no. 6, pp. 46-57, June 1989, doi: 10.1109/2.30720.

[4] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," in Computer, vol. 22, no. 6, pp. 46-57, June 1989, doi: 10.1109/2.30720.

[5] E. Gonzalez, O.Alvarez, Y.Diaz, C.Parra and C.Bustacara, "BSA: A Complete Coverage Algorithm," Proceedings of the 2005 IEEE International Conference on Robotics and Automation, Barcelona, Spain, 2005, pp. 2040-2044, doi: 10.1109/ROBOT.2005.1570413.

[6] A. Khan, I. Noreen, H. Ryu, N. L. Doh, and Z. Habib, "Online complete coverage path planning using two-way proximity search," Intel Serv Robotics 10, 229–240 (2017). doi: 10.1007/s11370-017-0223-z

[7] J. Connors and G. Elkaim, "Analysis of a Spline Based, Obstacle Avoiding Path Planning Algorithm," 2007 IEEE 65th Vehicular Technology Conference - VTC2007-Spring, Dublin, 2007, pp. 2565-2569, doi: 10.1109/VETECS.2007.528.

[8] Yuan, R., Zhang, F., Qu, J., Li, G. and Fu, Y., "A novel obstacle avoidance method based on multi-information inflation map," Industrial Robot, Vol. 47 No. 2, pp. 253-265. https://doi.org/10.1108/IR-05-2019-0114

[9] Joon Seop Oh, Yoon Ho Choi, Jin Bae Park and Y. F. Zheng, "Complete coverage navigation of cleaning robots using triangular-cell-based map," in IEEE Transactions on Industrial Electronics, vol. 51, no. 3, pp. 718-726, June 2004, doi: 10.1109/TIE.2004.825197.

[10] R. N. De Carvalho, H. A. Vidal, P. Vieira and M. I. Ribeiro, "Complete coverage path planning and guidance for cleaning robots," ISIE '97 Proceeding of the IEEE International Symposium on Industrial Electronics, Guimaraes, Portugal, 1997, pp. 677-682 vol.2, doi: 10.1109/ISIE.1997.649051.

[11] T. Lee, S. Baek, Y, Choi, and S. Oh, "Smooth coverage path planning and control of mobile robots based on high-resolution grid map representation," Robotics and Autonomous Systems, Vol. 59, No. 10, pp. 801-812. https://doi.org/10.1016/j.robot.2011.06.002

[12] Quigley, Morgan, et al. "ROS: an open-source Robot Operating System." ICRA workshop on open source software. Vol. 3. No. 3.2. 2009.

[13] Santos, Joao Machado, David Portugal, and Rui P. Rocha. "An evaluation of 2D SLAM techniques available in robot operating system." 2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR). IEEE, 2013.
[14] Quigley, Morgan, Brian Gerkey, and William D. Smart. Programming Robots with ROS: a practical introduction to the Robot Operating System. " O'Reilly Media, Inc.", 2015.
[15] Korf, Richard E. "Depth-first iterative-deepening: An optimal admissible tree search." Artificial intelligence 27.1 (1985): 97-109.
[16] Jianya, Yue Yang Gong. "An efficient implementation of shortest path algorithm based on dijkstra algorithm [j]." Journal of Wuhan Technical University of Surveying and Mapping (Wtusm) 3.004 (1999).
[17] Project source code. https://github.com/Jackack/Turtlebot3_complete_coverage