IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 14, 2021, accepted October 22, 2021, date of publication November 2, 2021, date of current version November 11, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3125105

RCD: Radial Cell Decomposition Algorithm for Mobile Robot Path Planning

OMNIA A. A. SALAMA[1], MOHAMED E. H. ELTAIB[2], HANY AHMED MOHAMED[1,3], AND OMAR SALAH[1]

[1]Department of Mechanical Engineering, Faculty of Engineering, Assiut University, Asyut 71516, Egypt
[2]Department of Mechanical Engineering, Faculty of Engineering, Kafrelsheikh University, Kafrelsheikh 33511, Egypt
[3]Mechanical Engineering Department, Higher Technological Institute, 10th of Ramadan City 44634, Egypt

Corresponding author: Omar Salah (omar.salah@aun.edu.eg)

ABSTRACT Finding the optimum path for mobile robots is now an essential task as lots of autonomous mobile robots are widely used in factories, hospitals, farms, etc. Many path planning algorithms have been developed to finding the optimum path with the minimum processing time. The vertical cell decomposition algorithm (VCD) is one of the popular path planning algorithms. It is able to find a path in a very short time. In this paper, we present a new algorithm, called the Radial cell decomposition (RCD) algorithm, which can generate shorter paths and a slightly faster than VCD algorithm. Furthermore, the VCD algorithm cannot be applied directly to obstacles in special cases, like two vertices have the same x-coordinate; on the other hand, the RCD algorithm can be applied to these special cases directly. In addition to that, the RCD algorithm is very suitable for corridor environments, unlike the VCD algorithm. In this paper, the RCD algorithm is described and tested for both cluttered and corridor environments. Furthermore, Two different algorithms A*, and Vertical cell decomposition are compared to the RCD algorithm. Simulation results confirm the effectiveness of the RCD algorithm in terms of path length and processing time.

INDEX TERMS Path planning, radial cell decomposition, vertical cell decomposition.

I. INTRODUCTION

Robots in the industry are evolving quickly from large manually controlled machines to small autonomous mobile robots [1]. Fast performance, high accuracy, flexibility, and safety of autonomous mobile robots led to the proliferation in agriculture [2], medical [3], and industrial applications [4]. Finding the optimal path between two points in an uncontrolled environment is a substantial task that autonomous mobile robots should be able to do [5]. After finding the optimal path, mobile robots can save time and effort by moving heavyweights from one place to another easily and safely without human intervention [6]. The environment around the robot can be static [7] if all obstacles are fixed in place or dynamic if some obstacles are moving or changing their place [8]. For known environments, the map is created before the process of planning a path. Although for unknown environments, the map is built gradually while the robot discovering the environment [9]. A massive number of applications of path planning have motivated researchers to develop different types of methods and algorithms in this field [10]. Most of these methods are trying to find the shortest path [11] from a start location to a destination/ goal position. Although, some research is focusing on finding the safest path [12], while others are focusing on reducing the computational time [13]. Classification of path planning algorithms is based on the algorithm itself and the environment [14]. Some path planning methods are global while others are local [15]. Global methods, also called off-line methods, are considering the entire work-space before the path planning process; while local methods, also called online methods, are considering a small area around the robot [16]. A complete method [17] is a method that guarantees to find a path between two points if one exists, while an incomplete method [18] does not guarantee to find a path even if one exists.

Path planning methods are classified into three main methods [19]; potential fields, sampling-based, and combinatorial methods As shown in Figure1. Potential field path planning is inspired by nature by considering the robot as a charged particle moving in a magnetic field [20]. The robot must be attracted by the goal and repelled by obstacles. The potential function is the sum of an attractive potential attracting the

The associate editor coordinating the review of this manuscript and approving it for publication was Heng Wang.

robot towards the goal and a repulsive potential pushing the robot away from obstacles [21]. One main disadvantage of potential fields is the local minimum trap. However some researchers have overcome this issue by using an optimization based approaches as in [22]. Two important advantages of potential fields are the low calculation cost and the ability to be used with both local and global path planning.

Sampling-based path planning algorithms convert the continuous map into a discrete map consists of a set of nodes that is created randomly and roads that connecting nodes in a certain way [23]. A collision detection function [24] is used to ensure that all nodes and roads are in free space then, a graph search algorithm is used to find the shortest path between any two nodes [25]. Probabilistic Roadmaps, PRM, [26] is a well-known algorithm that randomly creates a predetermined number of nodes. Then, it tries to connect each node to the nearest node or all nodes less than a predetermined distance. Finally, after the roadmap is created, the start and goal configurations are added to the roadmap and a graph search algorithm is used to find the shortest path from a start position to a goal position [27]. This algorithm is helpful in the case of a multi-query problem, where multiple goal positions are required to be added to the [28]. In the case of a single query problem, it is wasteful to create a map that covers the entire free space. Another familiar algorithm is used in single query problems is the Rapidly Exploring Random Trees, RRT, algorithm [29]. This algorithm considers the start and goal positions in the sampling process. A tree rooted at the start position is created where each new node is connected to the closest node within a predetermined distance k. The algorithm continues to add nodes to the tree until it reaches the goal position [30]. Sampling-based methods are more efficient in practical problems and higher dimension configuration spaces, C_{spaces}, [31]. They are probabilistically complete methods, but their performance is low in narrow passages and they are not suitable for dynamic environments [32].

Combinatorial methods, Grid-based methods, [25] are complete and exact algorithms [33] that find a path without approximation. These algorithms encode the topology of the robot's free space globally using some structures while describing the geometry of the free space locally [34]. An important advantage of these methods is that they provide an upper limit on the time needed to solve the problem [35]. First, the free space is decomposed into a finite number of cells. Second, an adjacency graph is created by determining which cells are adjacent. Finally, a graph search algorithm is used to find the shortest path connecting start and goal positions [36]. A* and Dijkstra [37] algorithms are two examples of grid-based methods that divide the free space into a set of cells of equal size, i.e., grids. Each cell is represented as a node in the adjacency graph and the distance between every two nodes is the weight of the edge connecting them [38]. A robot can move only in a lattice around the current cell. The more the number of cells, the more the computation time and the better the resulting path [39]. For large environments, this decomposition results in a huge number of cells which results in a better and shorter path but a slower algorithm [40]. The large number of nodes need very long time to be processed which make these algorithms not suitable for practical problems. Instead of dividing the free space into equal cells, cell decomposition methods divide the free space into a small number of cells which decreases the computation time [41]. Lower computation time and faster execution of the algorithm are two advantages of cell decomposition methods over grid-based methods [42]. Triangulation and vertical cell decomposition are two popular cell decomposition methods [43]. Triangulation is done by first triangulate the free space into a set of 2-simplicial, triangles, then create the adjacency graph by representing each triangle with a node and connect every two adjacent cells by an edge in the graph [44]. This algorithm is complete, but it is complicated to use in practice due to its computational complexity.

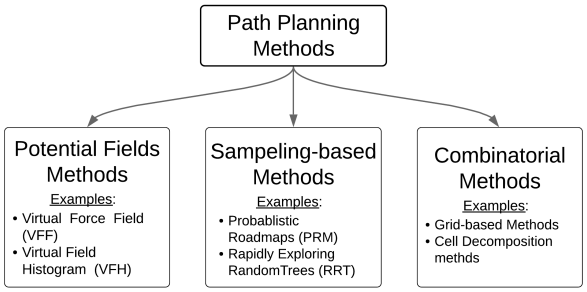


FIGURE 1. Classification of path planning methods.

Vertical cell decomposition depends on the plan sweep algorithm from computational geometry [45]. By partitioning the free space, C_{free}, into a finite number of 2-cells and 1-cells which are either trapezoidal or triangular. Each 1-cell is a vertical line that represents the border between every two 2-cells and each 2-cell is the interior of the trapezoid or the triangle [46]. A topological graph [47], i.e., a roadmap, representing adjacency relations between cells is built from cell decomposition. Two 2-cells are adjacent if they are sharing the same 1-cell in between. Nodes in the graph represent the 2-cells while edges represent the adjacency relations between cells [48]. edges are weighted with Euclidean distance between nodes. Once the Graph is built correctly, two nodes representing the start and goal positions are added to the graph and a graph search algorithm is used to find the shortest path from start to goal across the graph [49]. This algorithm is suitable for large environments, but it does not result in the shortest path if compared to algorithms like A* and Dijkstra. Vertical cell decomposition uses the line sweep algorithm which requires all obstacles to be in general positions, where no obstacle has vertical edges and all vertices have distinct X-coordinate, [35], [50], [51]. Special cases are ignored in cell decomposition planning by performing random perturbations in some random direction. Performing random perturbations can be very frustrating in practice because most of the implementation time is devoted

to fixing such special cases which unnecessarily complicate the solution [35], [50].

In this paper, a new cell decomposition algorithm, Radial Cell Decomposition (RCD), is introduced. RCD algorithm is applied directly to environments that have some obstacles in special cases, have some vertical edges or two points with the same x-coordinate, without the need to convert them to general positions. This saves the time of changing special cases back into general positions and reducing the complexity of cell decomposition algorithms. According to that, the RCD algorithm makes it easier to find a path throw corridor environments than using other cell decomposition algorithms since corridor maps have a lot of vertical edges. The remaining sections of the paper are organized as follows: Section II presents a detailed explanation of the proposed algorithm, RCD algorithm, together with the pseudocode of the algorithm. Section 3 shows examples and simulations of three different algorithms while results of simulations and comparisons are presented in section 4. Finally, conclusions and some remarks are given in section 5.

II. MATERIAL AND METHOD

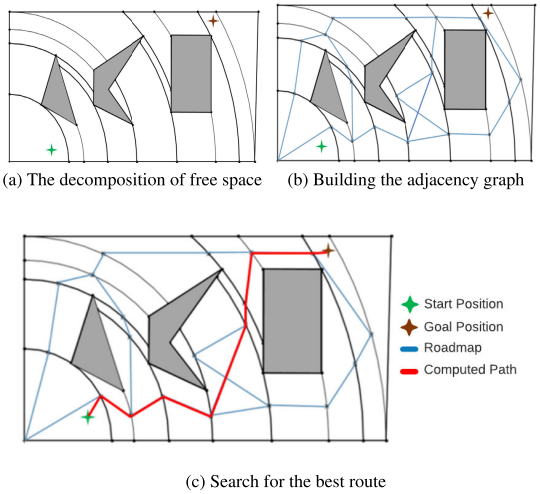
Radial Cell Decomposition is a cell decomposition method that uses the concept of partitioning the free space into a set of cells. A 2D bounded map $\mathcal{W} \subset \mathbb{R}^2$ is used to represent the environment. $\mathcal{W}$ includes a set of static polygonal obstacles $\mathcal{O} = \{O_1, O_2, \ldots, O_m\}$ where m is the cardinality of set $\mathcal{O}$ and a robot $\mathcal{A}$ represented as a point, located initially at start position q_s and assumed to move to a goal position q_g. The set of vertices used to represent obstacles configuration C_{obs} is $V = \{v_1, v_2, \ldots, v_n\}$ where n is the number of vertices V.

Finding a path using RCD algorithm is done throw the following steps which shown in Figure 2:

1) First, computing the decomposition itself, i.e., dividing the free space into a set of bounded cells $\mathcal{C} = \{c_1, c_2, \ldots, c_j\}$ where j is the cardinality of set $\mathcal{C}$ as shown in Figure 2a.
2) Second, building the adjacency graph $\mathcal{G}$ that represents C_{free} and all feasible paths the robot can traverse as shown in Figure 2b.
3) Third, applying A* algorithm on graph $\mathcal{G}$ to find the shortest path between any two nodes as shown in Figure 2c.

Radial cell decomposition (RCD) decomposes the free space into a set of cells $\mathcal{C}$ shaped like part of a disk by drawing arches at obstacle vertices. The algorithm first selects the most left vertex from boundaries polygon to be the center node c_c, center of all developed arches, then, it sorts all polygon vertices V by their distance from c_c. For each vertex $v \in V$ try to extend two arches cw and ccw throw C_{free} until C_{obs} is hit. Each arch extends from vertex v until it hits the first obstacle above or under it. As shown in Figure 3, four possible cases are depending on the possibility of extending an arch in each direction:

- Case1: two arches CW and CCW are extended from v.



(a) The decomposition of free space (b) Building the adjacency graph

(c) Search for the best route

FIGURE 2. The three steps of radial cell decomposition algorithm.

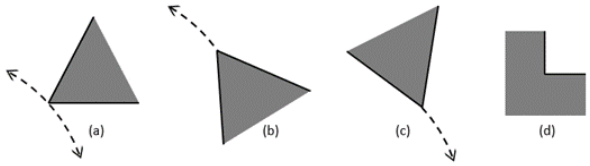


FIGURE 3. The four possible cases for a vertex v: a) extending two arches cw and ccw, b) extend a ccw arch, c) extend a cw arch, and d) no arches can be extended.

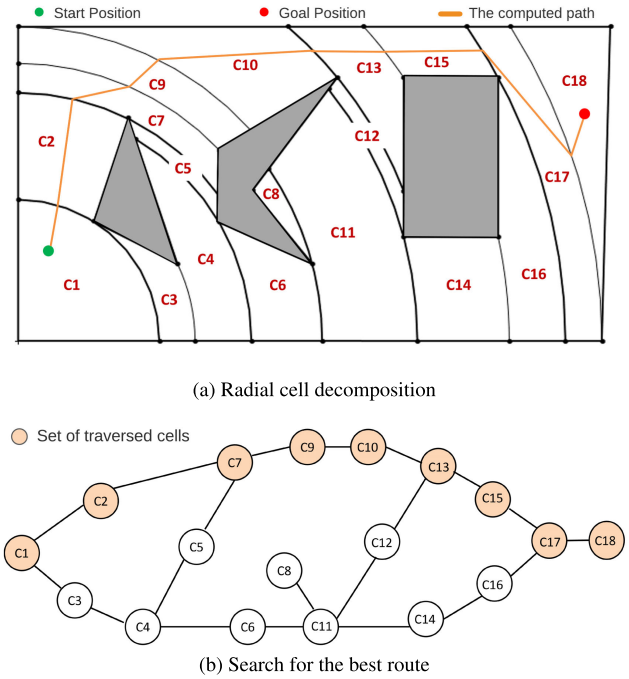
- Case2: one arch CCW is extended from v.
- Case3: one arch CW is extended from v.
- Case4: no arches to extend from v.

Partitioning the C_{free} according to these arches results in Radial Cell Decomposition shown in Figure 4a. The free space is decomposed into 2-cells and 1-cells where each 2-cell is the set of points in the interior of the resulting cell and each 1-cell is the set of points that form the arch shared by two adjacent cells. As shown in Figure 4a, cells $(c_1, c_2, \ldots, c_{18})$ are the free cells resulted from Radial cell decomposition. These cells are represented in the adjacency graph by nodes as shown in Figure 4b. Edges in the graph represents the adjacency relations between the free cells. A path is then computed by searching the graph for the Consecutive nodes that will connect start node to goal. An example of the resulted path is shown in Figure 4.

After decomposition is correctly maintained, an adjacency graph $\mathcal{G}$ is built, as shown in Figure 4b, to solve different path planning queries. Each free cell is represented by a node in the adjacency graph. Two nodes are connected by a road if they are adjacent. Start and goal positions are then connected to the graph. Finally, the adjacency graph is searched for the best route from start to goal using a graph search algorithm like A* algorithm. The computed path shown in Figure 4a is the set of Consecutive nodes that connect start node to goal node which represented in Figure 4b by pink nodes.

A. COMPUTING DECOMPOSITIONS

As mentioned in section II, computing decompositions is the first stage in the problem of finding a path by cell



(a) Radial cell decomposition

(b) Search for the best route

FIGURE 4. Radial decomposition of the free space into 2-cells and 1-cells is shown in (a), and the resulting adjacency graph is shown in (b).

decomposition methods. The input to the algorithm is the boundary description of boundary polygon and obstacles $\mathcal{O}$. In order to collect these information, a camera is installed above the environment to take images of the environment. Using image processing technique images can be analyzed to extract the desired information. The coordinates of boundary polygon and obstacles are extracted. The output is the set of decomposed cells stored in list CC. The following is the detailed explanation of Computing decompositions stage.

1) Given the map of the surroundings, the most left vertex of boundaries polygon is defined as the center node c_c.
2) Information of all polygons in the map is extracted by image processing. These information is then stored in two excel sheets that will be used as inputs to the algorithm. The first excel sheet, named "vertices", contains the ID of each vertex, its coordinates, and the euclidean distance to c_c. The second excel sheet, named "obstacle edges", contains the ID of each edge, the IDs of its endpoints, and the equation of this edge.
3) Sort the set of all vertices V by their distance from center node c_c, this step can take $\mathcal{O}(n\ logn)$ time [52].
4) Loop over the sorted list V, each visit to a vertex $v \in V$ is an event, at each event three main actions happen:
   - An arch A centered at node c_c passing through v and extending until it hits the first obstacle in both directions is created.
   - A list L of some C_{obs} edges that intersects with A is updated. By maintaining L in a balanced binary search tree, updating L at each event will need $\mathcal{O}(logn)$ time instead of $\mathcal{O}(n)$ for insertion and deletion of edges [53].
   - According to the fact that arch A is dividing the free space into smaller cells, a list of closed cells CC, which are the cells that are fully discovered, is updated. Also, a list of opened cells OC is updated. This list contains the cells that are opened and created by A but will be completed and closed by another arch in a future event.
5) For each vertex v, let e_{upper} and e_{lower} to be the two edges containing v. Draw an imaginary circle, C_{im}, from c_c passing throw v. There are four possible cases depending on the position of e_{upper} and e_{lower} to C_{im}. An explanation of each case is shown next:

- **Case 1: both e_{upper} and e_{lower} are outside C_{im}**
  Insert e_{upper} and e_{lower} into list L. Two cases may occur:
  (a) All vertices of the current obstacle lie outside C_{im} as shown in Figure 5. First, create two new cells, one with e_{upper} and one with e_{lower}, and add both to list OC. Second, close the last cell in OC with the new arch A, add it to list CC, and remove it from OC.
  (b) Some vertices of the current obstacle lie inside C_{im} as shown in Figure 6. Create one new cell with e_{upper} and e_{lower} and add it to list OC.
- **Case 2: e_{upper} is outside and e_{lower} is inside C_{im} as shown in Figure 7**
  Delete e_{lower} from list L and insert e_{upper} to list L. Create one new cell with e_{upper} and add it to list OC. Search list OC for the opened cell contains e_{lower}, add it to list CC, and remove it from list OC.
- **Case 3: e_{upper} is inside and e_{lower} is outside C_{im} as shown in Figure 8**
  Delete e_{upper} from list L and insert e_{lower} to list L. Create one new cell with e_{lower} and add it to list OC. Search list OC for the opened cell contains e_{upper}, add it to list CC, and remove it from list OC.
- **Case 4: both e_{upper} and e_{lower} are inside C_{im}**
  Delete e_{upper} and e_{lower} from the list L. Create two new arches A_{cw} and A_{ccw}. Two cases may occur:
  (a) All vertices of the current obstacle lie inside C_{im} as shown in Figure 9. First, Create one new cell with both arches and add it to the list OC. Then, Search list OC for the cell contains e_{upper} and close it with A_{ccw}. Do the same for the cell contains e_{lower}, close it with A_{cw}. Finally, add both cells to list CC and remove them from list OC.
  (b) Some vertices of the current obstacle lie outside C_{im} as shown in Figure 10. Close one cell with e_{upper} and e_{lower}, add it list CC, and remove it from list OC.

The closed list CC is the set of decomposed cells that are used for building the adjacency graph. If the decomposition was applied correctly, list CC should contain all decomposed cells while OC should be empty at the last vertex in V.

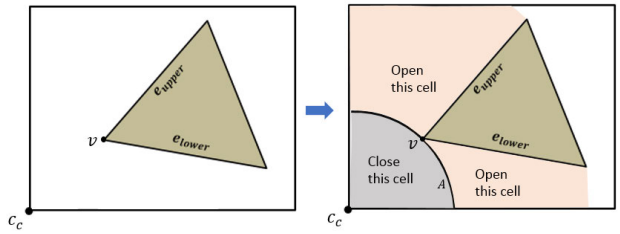


FIGURE 5. Case 1a, two arches are developed at vertex v, one cell is closed by arch A, and two cells are opened, the upper cell is opened by e_{upper} and A_{upper} and lower cell is opened by e_{lower} and A_{lower}.

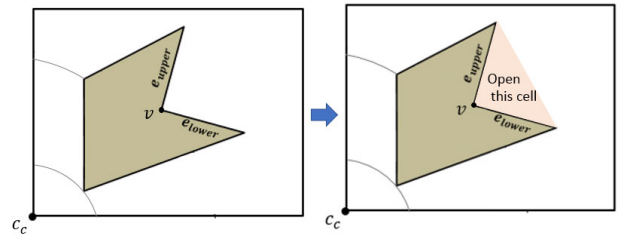


FIGURE 6. Case 1b, the arch is not developed at vertex v, and one cell is opened by e_{upper} and e_{lower}.

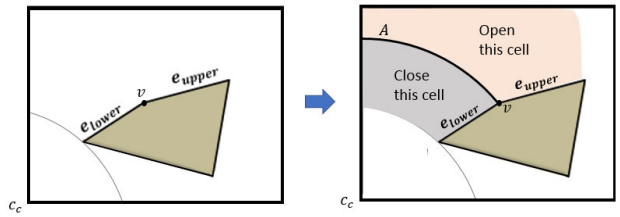


FIGURE 7. Case 2, one upper arch is developed at vertex v, an opened cell contains e_{lower} is closed by arch A, and a new cell is opened by e_{upper} and arch A.

B. BUILDING ADJACENCY GRAPH

The adjacency graph is built using list CC, after the decomposition of the space is completed. This is done by looping over cells in list CC. In this case an extra $\mathcal{O}(n)$ time will be needed. A better approach, that saves time [35], is incrementally building $\mathcal{G}$ at each visit to $v \in V$ while decomposing the free space. Each free cell c_i is represented in $\mathcal{G}$ by a node q_i which can be any point such that $q_i \in c_i$. These nodes can be cells centroids or a point near to the centroid. Each node, i.e., cell c_i is connected by an edge to its neighbor nodes $N(c_i) \subset \mathcal{C}$, $\forall c_i \in \mathcal{C}$ where $N(c_i)$ are the cells that share 1-cell with cell c_i.

There are two approaches to represent the decomposition in a graph $\mathcal{G}$. One simple way is to represent each 1-cell, arch, by a node located at its center. An edge will connect two nodes if both nodes belong to a different border of the same 2-cell c_i. This representation of $\mathcal{G}$ is shown in Figure 11a. Another way is to represent each 2-cell by a node located near to its centroid and each 1-cell by a node located at its center then connect the node representing the 2-cell by nodes in its boundaries, i.e., 1-cells, this representation is shown in Figure 11b. Note that the first representation reduces the number of nodes in $\mathcal{G}$ which reduces the time needed by the A* algorithm to find the shortest path from $\mathcal{G}$. The number of nodes and roads resulted from the second representation are larger, but it solves the problem of discontinuity of the roadmap with corridor environments. It is very effective with corridor environments as shown in section IV. Once $\mathcal{G}$ is correctly built, the A* algorithm can be applied to find the shortest path throw nodes and edges in $\mathcal{G}$.

Algorithm 1: Radial Cell Decomposition

Result: List of Decomposed Cells CC
Input: vertices and edges of obstacles and boundaries polygon C_c = the most left vertex of boundary polygon, v_i = current visited vertex from list V, r_i = distance from C_c to v_i, obs_i = obstacle contains v_i, e_{upper} = upper edge contains vertex v_i, e_{lower} = lower edge contains vertex v_i, T = an empty AVL tree, OC = an empty List of opened cells, CC = an empty List of closed, cells, oc_k = last cell in list OC
Sort list vertices V from lower r_i to higher values
foreach $v_i \in V$ **do**
  if both e_{upper} and e_{lower} were outside the circle of radius r_i **then**
    Insert both e_{upper},e_{lower} to tree T
    if obs_i lies outside the circle of radius r_i **then**
      A_1 = a CW arch of radius r_i from v_i
      A_2 = a CCW arch of radius r_i from v_i
      add A_1, A_2 to oc_k, insert oc_k to CC, and delete it from OC
      create cell oc_1, insert it to OC, $e_{upper} \in oc_1$
      create cell oc_2, insert it to list OC, $e_{lower} \in oc_2$
    else
      create cell oc_i, insert it to list OC, $e_{lower}, e_{upper} \in oc_i$
    end
  else if e_{upper} is outside the circle of radius r_i while e_{lower} is inside the circle **then**
    Insert e_{upper} to T and delete e_{lower} from T
    A_i = a CCW arch of radius r_i from v_i
    Search OC for cell contains e_{lower}, add A_i to it, insert it to CC, and delete it from OC
    discover cell oc_i, insert oc_i to list OC, $A_i \in oc_i$
  else if e_{lower} is outside the circle of radius r_i while e_{upper} is inside the circle **then**
    Insert e_{lower} to T and delete e_{upper} from T
    A_i = a CW arch of radius r_i from v_i
    Search OC for cell contains e_{upper}, add A_i to it, insert it to CC, and delete it from OC
    discover cell oc_i, insert oc_i to list OC, $A_i \in oc_i$
  else if both e_{upper} and e_{lower} are inside the circle of radius r_i **then**
    Delete both e_{upper},e_{lower} from tree T
    if obs_i lies inside the circle of radius r_i **then**
      A_1 = a CW arch of radius r_i from v_i
      A_2 = a CCW arch of radius r_i from v_i
      discover cell oc_i, insert oc_i to list OC, $A_1, A_2 \in oc_i$
      Search OC for two cells contain e_{lower} and e_{upper}, add A_1, A_2 to them, delete them from OC, and insert them to CC
    else
      Search OC for cell contains e_{lower} and e_{upper}, delete them from OC, and insert them to CC
    end
end

C. FINDING THE PATH

Once $\mathcal{G}$ is correctly obtained, a query (q_s, q_g) can be solved directly by any graph search algorithms like the A* algorithm [54]. The two cells containing q_s and q_g are first determined. Let C_0 denote the free cell that contains q_s and q_0

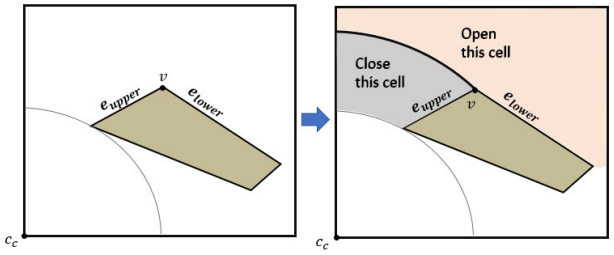


FIGURE 8. Case 3, one lower arch is developed at vertex *v*, one cell is closed, and one cell is opened.

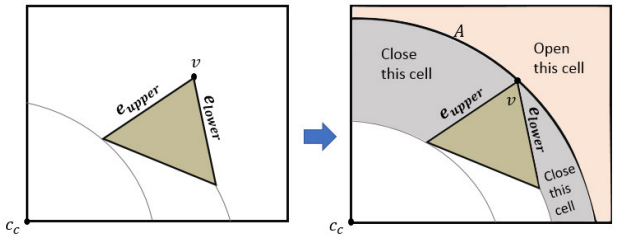


FIGURE 9. Case 4a, two arches are developed at vertex *v*, two opened cells are closed by arch *A*, and one cell is opened by arch *A*.

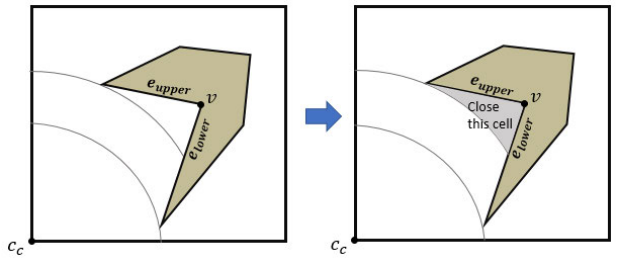


FIGURE 10. Case 4b, no arch is developed at vertex *v*, one opened cell is closed by e_{lower} and e_{upper}.

denote the node represents C_0 in $\mathcal{G}$. Likewise, let C_k denote the free cell that contains q_g and q_0 denote the node represents C_k in $\mathcal{G}$. After connecting (q_s to q_0) and (q_g to q_k), the graph $\mathcal{G}$ is searched for the shortest path between q_s and q_g. The resulting path is the sequence of nodes $q_0, q_1, \ldots q_k$ that are visited by the robot while traveling from point q_s to point q_g as shown in Figure 12. If no path was found, the algorithm will return that no solution exists. A flowchart shown in Figure 13 shows the main steps of the RCD algorithm.

D. ANALYSIS OF RCD ALGORITHM

Starting with n polygonal vertices and n polygonal edges, the time needed by RCD algorithm is $\mathcal{O}(nlogn)$. We will analyze the time taken by each step of the algorithm in terms of n.

- Sorting all polygonal vertices by their distance to center node C_c will take $\mathcal{O}(nlogn)$ using quicksort algorithm [55].
- Extending $\mathcal{O}(n)$ arches CW and CCW is done by finding the intersecting edges. By maintaining edges stored in a binary search tree L, insertion and deletion of an edge will take $\mathcal{O}(logn)$ time. So the time needed for this step is $\mathcal{O}(nlogn)$ for all vertices.
- Building the adjacency graph takes $\mathcal{O}(n)$ time as the number of cells and edges in adjacency graph is $\mathcal{O}(n)$ and for each cell calculating the center node takes constant time.
- (d) By storing closed and opened cells in binary search trees CC and OC, Search, insertion, deletion of cells takes $\mathcal{O}(nlogn)$ time for all $\mathcal{O}(n)$ cells.
- A* algorithm takes $\mathcal{O}(nlogn)$ time [56] to find the shortest path from the adjacency graph.

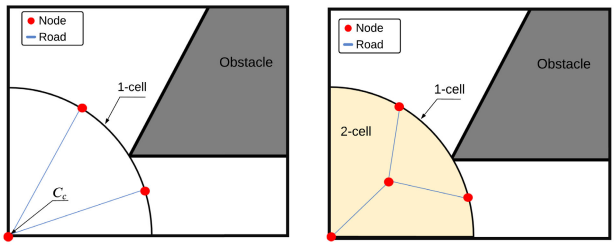


FIGURE 11. Two different representations of the same cell, a) represent each 1-cell by a node located at its center, b) represents each 2-cell by a node located near to its centroid, and each 1-cell by a node at its center.

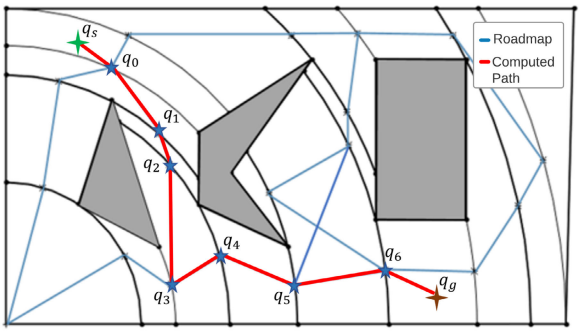


FIGURE 12. The resulting path after applying the A* algorithm to graph $\mathcal{G}$.

III. SIMULATION

Two different environments, cluttered and corridors, are used for simulating RCD algorithm together with the classical A* and Vertical Cell Decomposition. Each environment was tested for series of different queries, start and goal positions, via MATLAB. For each query, a comparison between the three algorithms is based on the length of the path and the processing time needed.

For simplification, a robot $\mathcal{A}$ is assumed to be a point in a 2D bounded workspace $\mathcal{W}$ that includes some stationary polygonal obstacles $\mathcal{O}$. Robot $\mathcal{A}$ is initially located at start position q_s and should move to goal position q_g while avoiding colliding with obstacles. The input to the first algorithm, classical A*, is the binary image of the map where obstacles are represented as 1 and free space as 0. The entire map is discretized into a grid of 100 × 100 nodes and the robot can move diagonally. At each node $x \in X$, A* calculates $f(x)$ for all neighbor nodes to find the node with the best route,i.e., lowest $f(x)$ that is calculated in (1).

$$f(x) = g(x) + h(x) \tag{1}$$

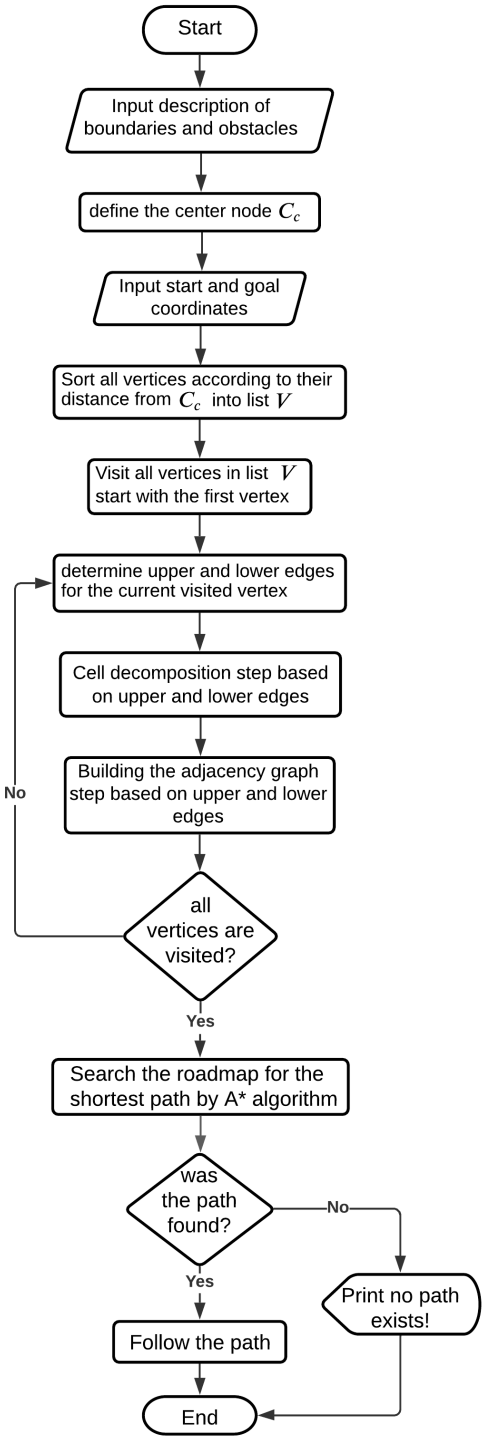


FIGURE 13. Flowchart of the proposed RCD algorithm.

where $g(x)$ denotes the distance from the start node to current node x and the heuristic function $h(x)$ is the Euclidean distance from current node x to the goal node. The output of the algorithm is the length of the path and the time needed by the algorithm to find the best route. Vertical cell decomposition and RCD are decomposing the workspace into a set of cells, i.e., nodes, then create an adjacency graph $\mathcal{G}$. In some cases, some edges from the generated graph $\mathcal{G}$ pass very near to C_{obs}

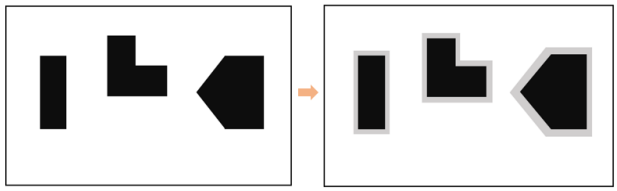


FIGURE 14. Enlarging C_{obs} to ensure safer paths.

which may occur in a collision with obstacles. According to that, obstacles are enlarged by a constant distance to ensure generating a safer path as shown in Figure 14. After enlarging obstacles, the new C_{obs} map is used by the algorithm to find the shortest path. Input is the description of vertices and edges of boundary polygon and obstacles. The output is the sequence of nodes $q_0, q_1, \ldots q_k$ that forms the shortest path from $\mathcal{G}$ that connects q_s to q_g. The resulted path is now guaranteed to be a safe path.

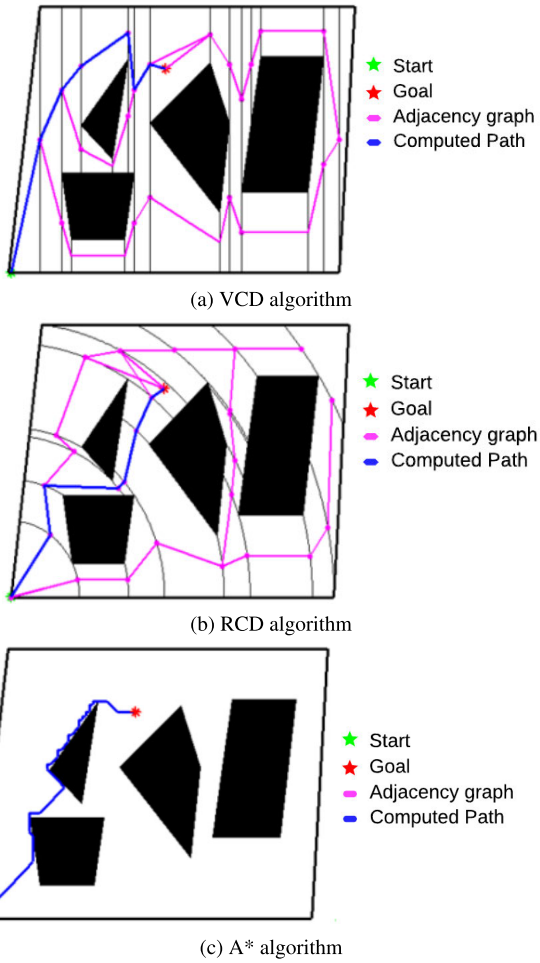
IV. RESULTS AND DISCUSSION

This section compares the Classical A*, Vertical cell decomposition, and RCD decomposition through cluttered and corridor environments in terms of path length and execution time. In all figures, the start configuration is represented as a green star while the goal configuration is represented as a red star, and the resulted path is plotted in blue.

A. CLUTTERED ENVIRONMENTS

In this environment, a set of polygonal obstacles are located randomly in a bounded polygonal environment. Figure 15 shows the computed path of the three algorithms for the same cluttered environment. For VCD and RCD algorithms the roadmap resulted from cell decomposition is colored in pink. The computed path is colored in blue in the three algorithms.

Table 1 shows a comparison between the three algorithms depending on two factors, the length of the resulted route and the processing time. Each case was run 10 times and the average processing time was recorded. As noticed in table1, A* algorithm produces the shortest path but it is very slow if compared to RCD and VCD algorithms. This ability to save processing time is and advantage of cell decomposition methods over grid-based methods. Also, the processing time of A* algorithm is increasing rapidly with the distance between start and goal positions As the number of processed nodes are increasing. On the contrary, the time taken by VCD and RCD algorithms is nearly constant for different queries as the number of nodes are constant. Note that, the number of processed nodes in A* algorithm depends on the size of the map and the resolution of grids. For VCD and RCD algorithms, the number of nodes in a map depends on the number of vertices of obstacles. As shown in table 1, paths generated by RCD algorithm are always shorter than paths generated by VCD algorithm while the processing time of RCD is slightly less than the processing time of VCD. This emphasizes the effectiveness of RCD algorithm in generating shorter paths if compared to VCD algorithm.



(a) VCD algorithm

(b) RCD algorithm

(c) A* algorithm

FIGURE 15. A cluttered environment is used to evaluate the path resulted from (a) VCD algorithm, (b) RCD algorithm, and (c) A* algorithm.

TABLE 1. A cluttered terrain is used to compare between A*, VCD and RCD algorithm in terms of path length and processing time.

	Coordinates		A* Alg.		VCD Alg.		RCD Alg.	
	Start	Goal	Length	T (s)	Length	T (s)	Length	T (s)
1	0,0	5.0,6.1	9.00	2.23	11.61	0.45	9.50	0.44
2	0,0	5.2,5.3	9.11	2.86	12.27	0.46	9.12	0.45
3	0,0	7.6,1.0	7.66	2.06	16.75	0.48	8.24	0.45
4	0,0	6.4,6.6	8.88	2.53	13.02	0.46	12.47	0.45
5	0,0	7.5,8.0	11.00	3.01	18.20	0.45	13.83	0.44
6	0,0	10.2,1.0	10.22	2.78	22.94	0.45	10.93	0.44

As shown in Figure17, four different terrains are used to compare between Vertical cell decomposition (VCD) and Radial cell decomposition (RCD) algorithms. The same start and goal configurations are used in all terrains. The roadmaps resulted from the decomposition of cell is shown in pink while the computed path is represented in blue. For both algorithms, the length of paths and processing time for each terrain in Figure17 are recorded in table 2. Table 2 shows the that the length of the paths generated by RCD algorithm are shorter than those generated by VCD algorithm as shown in the last column. These results proves that Radial decomposition of cells produces shorter paths if compared to vertical decomposition of cells.

TABLE 2. A comparison between VCD and RCD algorithms in terms of path length and processing time.

	Coordinates		VCD Alg.		RCD Alg.		improvement
	Start	Goal	Length	Time (s)	Length	Time (s)	in length %
1	0,0	10.0 , 7.5	15.61	0.45	15.54	0.34	0.44 %
2	0,0	10.0 , 7.5	17.88	0.46	16.66	0.43	6.8 %
3	0,0	10.0 , 7.5	21.38	0.43	17.37	0.41	17.2 %
4	0,0	10.0 , 7.5	19.55	0.43	15.88	0.43	18.7 %

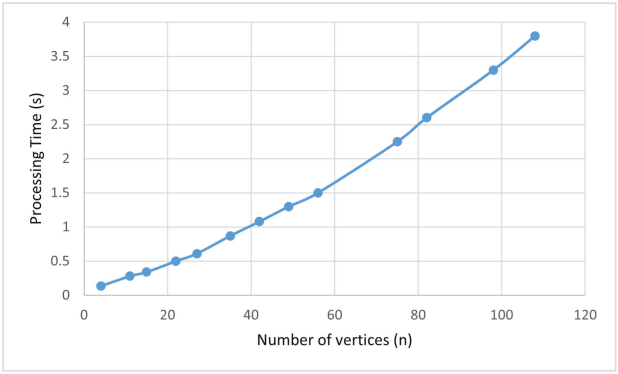


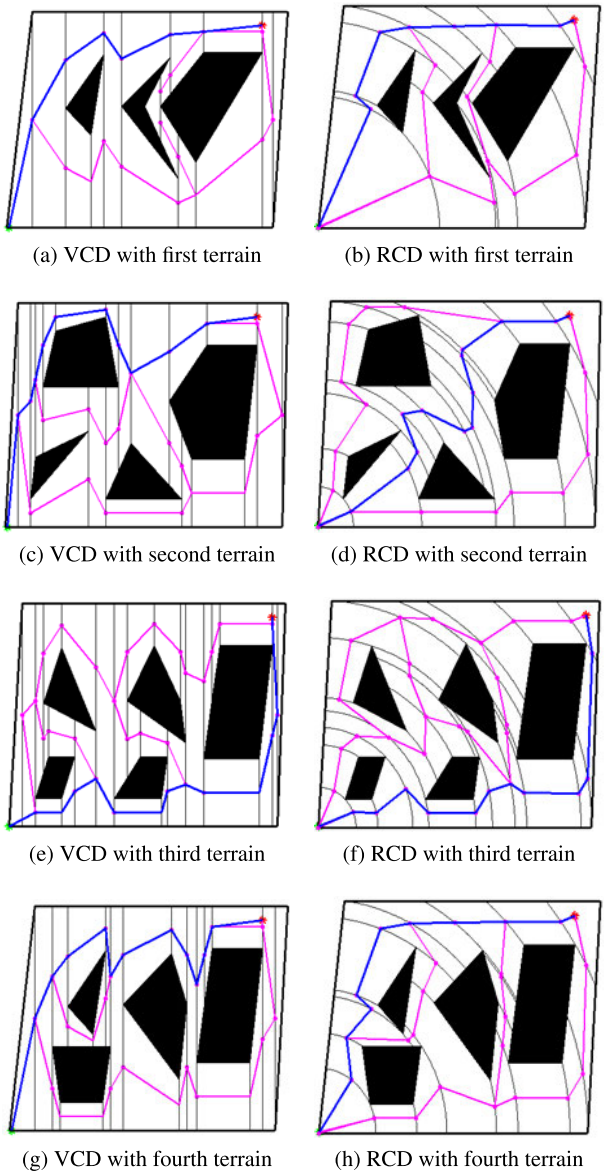
FIGURE 16. Relation between the processing time and number of vertices in an environment.

The tuning parameter of RCD algorithm is the number of vertices of obstacles in the environment. The processing time of RCD algorithm increases as the number of vertices increases. The processing time needed by RCD algorithm is $\mathcal{O}(nlogn)$- as discussed in section II-D. Figure16 shows how the processing time increases when the number of vertices in the environment increases.

B. CORRIDOR ENVIRONMENTS

For corridor environments, Classic A* and RCD algorithms were tested and compared. Vertical cell decomposition is ignored in this scenario since this environment has almost all its edges in a vertical position which will complicate the solution- special cases are ignored in VCD algorithm. On the other hand, RCD algorithm can be applied to corridor environments directly and a good path will be found. The ease of applying RCD algorithm to corridor environments is a great advantage for RCD algorithm over vertical cell decomposition algorithm. When applying RCD algorithm to corridor maps, the second representation of the adjacency graph, discussed in section II-B, will be used to avoid the discontinuity while building the adjacency graph which allows the algorithm to reach any configuration between corridors. Figure 18 shows the resulted path computed by Classical A* and Radial Cell Decomposition for the same corridor map and same start and goal positions.

Table 3 compares between A* and RCD algorithm using different goal positions. The comparison is done in terms of path length and processing time for each algorithm. As shown in Table 3, the processing time of RCD algorithm is very short if compared to the processing time of A* algorithm. Also, the processing time of RCD algorithm is almost constant among different queries, while the processing time of A* algorithm depends on the distance between start and



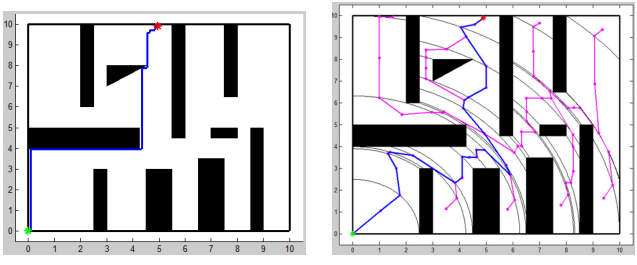
(a) VCD with first terrain (b) RCD with first terrain

(c) VCD with second terrain (d) RCD with second terrain

(e) VCD with third terrain (f) RCD with third terrain

(g) VCD with fourth terrain (h) RCD with fourth terrain

FIGURE 17. Four different cluttered terrains used to compare between the path resulted from VCD and RCD algorithms using same start and goal positions.



(a) A* algorithm (b) VCD algorithm

FIGURE 18. A corridor map is used to compare between paths resulted from a) A*, b) Radial cell decomposition.

goal configurations. Although A* algorithm produces shorter paths than RCD algorithm, it consumes processing time up to 6.5 times RCD algorithm. This makes RCD algorithm more suitable with time sensitive cases.

TABLE 3. A corridor map is used to compare between A* and RCD algorithms using different queries.

	Coordinates		A* Alg.		RCD Alg.	
	Start	Goal	Length	Time (s)	Length	Time (s)
1	0,0	6.5,1.1	11.42	4.81	13.22	1.14
2	0,0	6.3,3.2	9.49	4.58	15.54	1.15
3	0,0	6.9,5.7	12.62	5.41	17.44	1.20
4	0,0	4.9,9.9	14.85	7.89	20.37	1.45
5	0,0	8.9,6.7	15.54	6.54	15.65	1.44
6	0,0	8.9,7.8	16.87	9.43	25.36	1.45

V. CONCLUSION AND FUTURE WORK

This paper proposed a new approach to cell decomposition. RCD algorithm first divides the environment into a set of free cells based on a set of arches drawn from the center point. After decomposition is done properly, an adjacency graph is created from free cells by two procedures; represent each arch by a node in its middle and connect every two nodes if they belong to the border of the same free cell. The second procedure, is by representing each free cell by a node located near to its centroid and each arch by a node in its middle then, connect each free cell node by nodes located in its borders. The resulted adjacency graph is then used by the A* algorithm to find the shortest path from a start position to a goal position.

RCD algorithm was tested and verified together with classical A* algorithm and Vertical cell decomposition in two different environments, cluttered and corridors. Results of cluttered environments showed that paths generated by the RCD algorithm are shorter than those generated by Vertical cell decomposition by 69.8% on average. Also, the computation time of the RCD algorithm is less than the computation time of Vertical cell decomposition by 10-30 ms. Also, results showed that the length of paths found by the classical A* algorithm are shorter than those found by the RCD algorithm by 11.35% on average. However, the computation time of the A* algorithm is up to 6.5 times the computation time of RCD algorithm. Furthermore, The computation time of A* is increasing rapidly as the distance between start and goal positions is increased while it is found to be nearly constant for RCD and Vertical cell decomposition algorithms.

The previous results confirm the advantage of the RCD algorithm over classical A* and Vertical cell decomposition algorithms in terms of computational time in both cluttered and corridor environments which makes it applicable and more effective with large environments. Also, results emphasize the advantage of the RCD algorithm over the Vertical cell decomposition algorithm in terms of both path length and computation time in a cluttered environment. Furthermore, the ease of applying RCD algorithm to obstacles in special case such as corridor environments is a great advantage of RCD algorithm over VCD algorithm.

As stated, RCD algorithm is applied to different static environments using an image of the environment. On the other hand, it is very important to apply the proposed algorithm with dynamic environments. In this case, several images will be taken at a rate that is appropriate to the changing rate of obstacles. As a future work, it is very interesting to verify the

ability of implementing RCD algorithm in dynamic environments. Furthermore, refinement of paths generated by RCD algorithm is an interesting point to consider in future work.

REFERENCES

[1] S. S. Kim, J. Kim, F. Badu-Baiden, M. Giroux, and Y. Choi, "Preference for robot service or human service in hotels? Impacts of the COVID-19 pandemic," *Int. J. Hospitality Manage.*, vol. 93, Feb. 2021, Art. no. 102795, doi: 10.1016/j.ijhm.2020.102795.

[2] L. C. Santos, F. N. Santos, E. J. Solteiro Pires, A. Valente, P. Costa, and S. Magalhães, "Path planning for ground robots in agriculture: A short review," in *Proc. ICARSC*, Apr. 2020, pp. 61–66.

[3] M. Javaid, A. Haleem, A. Vaish, R. Vaishya, and K. P. Iyengar, "Robotics applications in COVID-19: A review," *J. Ind. Integr. Manage.*, vol. 5, no. 4, pp. 441–451, Nov. 2020, doi: 10.1142/S2424862220300033.

[4] M. B. Alatise and G. P. Hancke, "A review on challenges of autonomous mobile robot and sensor fusion methods," *IEEE Access*, vol. 8, pp. 39830–39846, 2020, doi: 10.1109/ACCESS.2020.2975643.

[5] M. G. B. Atia, H. El-Hussieny, and O. Salah, "A supervisory-based collaborative obstacle-guided path refinement algorithm for path planning in wide terrains," *IEEE Access*, vol. 8, pp. 214672–214684, 2020, doi: 10.1109/ACCESS.2020.3041802.

[6] A. Abbadi and V. Přenosil, "Safe path planning using cell decomposition approximation," in *Int. Conf. Dist. Learn. Simul. Commun. (DLSC)*, Brno, Czech Republic, May 2015, pp. 8–14.

[7] M. S. Alam, M. U. Rafique, and M. U. Khan, "Mobile robot path planning in static environments using particle swarm optimization," *Int. J. Electron.*, vol. 3, pp. 2320–4028, 2016.

[8] A. Vemula, K. Muelling, and J. Oh, "Path planning in dynamic environments with adaptive dimensionality," in *Proc. Int. Symp. Comb. Search (SoCS)*, Jul. 2016, pp. 107–116.

[9] J. van den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *Proc. IEEE Int. Conf. Robot Automat. (ICRA)*, May 2006, pp. 2366–2371, doi: 10.1109/ROBOT.2006.1642056.

[10] M. G. B. Atia, O. Salah, and H. Ei-Hussieny, "OGPR: An obstacle-guided path refinement approach for mobile robot path planning," in *Proc. IEEE Int. Conf. Robot. Biomimetics (ROBIO)*, Dec. 2018, pp. 844–849, doi: 10.1109/ROBIO.2018.8665080.

[11] X.-Z. Gao, Z. Hou, X.-F. Zhu, J.-T. Zhang, and X.-Q. Chen, "The shortest path planning for manoeuvres of UAV," *Acta Polytechnica Hungarica*, vol. 10, pp. 214672–214684, Jan. 2013, doi: 10.12700/aph.10.01.2013.1.13.

[12] X. Hu, L. Chen, B. Tang, D. Cao, and H. Hee, "Dynamic path planning for autonomous driving on various roads with avoidance of static and moving obstacles," *Mech. Syst. Signal Process.*, vol. 100, pp. 482–500, Feb. 2018, doi: 10.1016/j.ymssp.2017.07.019.

[13] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," in *Proc. IEEE Int. Conf. Robot. Automat.*, May 2012, pp. 3671–3678, doi: 10.1109/ICRA.2012.6224742.

[14] A. Liaqat, W. Hutabarat, D. Tiwari, L. Tinkler, D. Harra, B. Morgan, A. Taylor, T. Lu, and A. Tiwari, "Autonomous mobile robots in manufacturing: Highway code development, simulation, and testing," *Int. J. Adv. Manuf. Technol.*, vol. 104, nos. 9–12, pp. 4617–4628, Oct. 2019, doi: 10.1007/s00170-019-04257-1.

[15] P. Marin-Plaza, A. Hussein, D. Martin, and A. D. L. Escalera, "Global and local path planning study in a ROS-based research platform for autonomous vehicles," *J. Adv. Transp.*, vol. 2018, p. 10, Feb. 2018, doi: 10.1155/2018/6392697.

[16] Z. Shiller, "Off-line and on-line trajectory planning," in *Motion and Operation Planning of Robotic Systems*, 1st ed, Cham, Switzerland: Springer, 2015, pp. 29–62, ch. 2.

[17] A. Le, M. Arunmozhi, P. Veerajagadheswar, P.-C. Ku, T. H. Minh, V. Sivanantham, and R. Mohan, "Complete path planning for a tetris-inspired self-reconfigurable robot by the genetic algorithm of the traveling salesman problem," *Electronics*, vol. 7, no. 12, p. 344, 2018, doi: 10.3390/electronics7120344.

[18] Q. Luo, H. Wang, Y. Zheng, and J. He, "Research on path planning of mobile robot based on improved ant colony algorithm," *Neural Comput Appl.*, vol. 32, no. 6, pp. 1555–1566, Apr. 2020, doi: 10.1007/s00521-019-04172-2.

[19] L. Yang, J. Qi, D. Song, J. Xiao, J. Han, and Y. Xia, "Survey of robot 3D path planning algorithms," *Nural. Comput. Appl.*, vol. 2016, p. 22, Jul. 2016, doi: 10.1155/2016/7426913.

[20] Y. Rasekhipour, A. Khajepour, S.-K. Chen, and B. Litkouhi, "A potential field-based model predictive path-planning controller for autonomous road vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 5, pp. 1255–1267, May 2017, doi: 10.1109/TITS.2016.2604240.

[21] S. B. Germi, M. A. Khosravi, and R. Fesharakifard, "Adaptive GA-based potential field algorithm for collision-free path planning of mobile robots in dynamic environments," in *Proc. 6th RSI Int. Conf. Robot. Mechatronics (IcRoM)*, Oct. 2018, pp. 28–33, doi: 10.1109/ICRoM.2018.8657601.

[22] F. Bayat, S. Najafinia, and M. Aliyari, "Mobile robots path planning: Electrostatic potential field approach," *Expert Syst. Appl.*, vol. 100, pp. 68–78, Jun. 2018, doi: 10.1016/j.eswa.2018.01.050.

[23] C. I. Vasile, X. Li, and C. Belta, "Reactive sampling-based path planning with temporal logic specifications," *Int. J. Robot. Res.*, vol. 39, no. 8, pp. 1002–1028, Jun. 2020, doi: 10.1177/0278364920918919.

[24] X. Chu, Q. Hu, and J. Zhang, "Path planning and collision avoidance for a multi-arm space maneuverable robot," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 54, no. 1, pp. 217–232, Feb. 2018, doi: 10.1109/TAES.2017.2747938.

[25] S. K. Debnath, R. Omar, N. B. A. Latip, S. Shelyna, E. Nadira, C. K. N. C. K. Melor, T. K. Chakraborty, and E. Natarajan, "A review on graph search algorithms for optimal energy efficient path planning for an unmanned air vehicle," *Indonesian J. Electr. Eng. Comput. Sci.*, vol. 15, no. 2, pp. 743–749, Aug. 2019, doi: 10.11591/ijeecs.v15.i2.pp743-749.

[26] W. Khaksar, T. S. Hong, M. Khaksar, and O. Motlagh, "A low dispersion probabilistic roadmaps (LD-PRM) algorithm for fast and efficient sampling-based motion planning," *Int. J. Adv. Robot. Syst.*, vol. 10, no. 11, p. 397, Jan. 2013, doi: 10.5772/56973.

[27] M. Korkmaz and A. Durdu, "Comparison of optimal path planning algorithms," in *Proc. 14th Int. Conf. Adv. Trends Radioelecrtronics, Telecommun. Comput. Eng. (TCSET)*, Feb. 2018, pp. 255–258, doi: 10.1109/TCSET.2018.8336197.

[28] K. E. Bekris, B. Y. Chen, A. M. Ladd, E. Plaku, and L. E. Kavraki, "Multiple query probabilistic roadmap planning using single query planning primitives," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2003, pp. 656–661, doi: 10.1109/IROS.2003.1250704.

[29] I. Noreen, A. Khan, and Z. Habib, "A comparison of RRT, RRT* and RRT*-smart path planning algorithms," *Int. J. Comput. Sci. Netw. Secur.*, vol. 16, no. 10, pp. 20–27, Oct. 2016, doi: 10.1109/IROS.2003.1250704.

[30] C. Wong, E. Yang, X.-T. Yan, and D. Gu, "Optimal path planning based on a multi-tree T-RRT* approach for robotic task planning in continuous cost spaces," in *Proc. 12th France-Japan 10th Europe-Asia Congr. Mechatronics*, Sep. 2018, pp. 242–247, doi: 10.1109/MECATRONICS.2018.8495886.

[31] D. Devaurs, T. Siméon, and J. Cortés, "Efficient sampling-based approaches to optimal path planning in complex cost spaces," in *Algorithmic Foundations of Robotics XI*. Cham, Switzerland: Springer, 2015, pp. 143–159.

[32] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *IEEE Access*, vol. 2, pp. 56–77, 2014, doi: 10.1109/ACCESS.2014.2302442.

[33] M. N. Zafar and J. C. Mohanta, "Methodology for path planning and optimization of mobile robots: A review," *Proc. Comput. Sci.*, vol. 133, pp. 141–152, Jan. 2018, doi: 10.1016/j.procs.2018.07.018.

[34] S. K. Debnath, R. Omar, S. Bagchi, E. N. Sabudin, M. H. A. S. Kandar, K. Foysol, and T. K. Chakraborty, "Different cell decomposition path planning methods for unmanned air vehicles—A review," in *Proceedings of the 11th National Technical Seminar on Unmanned System Technology*. Singapore: Springer, 2021, pp. 99–111, ch. 8.

[35] S. M. LaValle, "Combinatorial motion planning," in *Planning Algorithms*, 1st ed. Cambridge, U.K.: Cambridge Univ. Press, 2006, pp. 249–252, ch. 6. [Online]. Available: https://lavalle.pl/planning/book.pdf

[36] P. Raja and S. Pugazhenthi, "Optimal path planning of mobile robots: A review," *Int. J. Phys. Sci.*, vol. 7, no. 9, pp. 1314–1320, Feb. 2012, doi: 10.5897/IJPS11.1745.

[37] H. Wang, Y. Yu, and Q. Yuan, "Application of Dijkstra algorithm in robot path-planning," in *Proc. 2nd Int. Conf. Mechanic Autom. Control Eng.*, Jul. 2011, pp. 1067–1069, doi: 10.1109/MACE.2011.5987118.

[38] A. K. Guruji, H. Agarwal, and D. K. Parsediya, "Time-efficient A* algorithm for robot path planning," *Proc. Technol.*, vol. 23, no. 1, pp. 144–149, 2016, doi: 10.1016/j.protcy.2016.03.010.

[39] W. Zeng and R. L. Church, "Finding shortest paths on real road networks: The case for A*," *Int. J. Geograph. Inf. Sci.*, vol. 23, no. 4, pp. 531–543, Apr. 2009, doi: 10.1080/13658810801949850.
[40] V. Kunchev, L. Jain, V. Ivancevic, and A. Finn, "Path planning and obstacle avoidance for autonomous mobile robots: A review," in *Proc. Int. Conf. Knowl.-Based Intell. Inf. Eng. Syst.* Berlin, Germany: Springer, Oct. 2006, pp. 537–544.
[41] B. K. Patle, A. Pandey, D. R. K. Parhi, and A. Jagadeesh, "A review: On path planning strategies for navigation of mobile robot," *Defence Technol.*, vol. 15, pp. 582–606, Aug. 2019, doi: 10.1016/j.dt.2019.04.011.
[42] R. Gonzalez, M. Kloetzer, and C. Mahulea, "Comparative study of trajectories resulted from cell decomposition path planning approaches," in *Proc. 21st Int. Conf. Syst. Theory, Control Comput. (ICSTCC)*, Oct. 2017, pp. 49–54, doi: 10.1109/ICSTCC.2017.8107010.
[43] L. S. C. Pun-Cheng, M. Y. F. Tang, and I. K. L. Cheung, "Exact cell decomposition on base map features for optimal path finding," *Int. J. Geogr. Inf. Sci.*, vol. 21, no. 2, pp. 175–185, Jan. 2007, doi: 10.1080/13658810600852206.
[44] J. Chen, C. Luo, M. Krishnan, M. Paulik, and Y. Tang, "An enhanced dynamic Delaunay triangulation-based path planning algorithm for autonomous mobile robot navigation," *Proc. SPIE*, vol. 7539, pp. 253–264, Jan. 2010.
[45] B. Dugarjav, S.-G. Lee, D. Kim, J. H. Kim, and N. Y. Chong, "Scan matching online cell decomposition for coverage path planning in an unknown environment," *Int. J. Precis. Eng. Manuf.*, vol. 14, no. 9, pp. 1551–1558, Sep. 2013, doi: 10.1007/s12541-013-0209-5.
[46] B.-C. So and J.-W. Jung, "Mobile robot path planning with opposite angle-based exact cell decomposition," *Adv. Sci. Lett.*, vol. 15, no. 1, pp. 144–148, Aug. 2012, doi: 10.1166/asl.2012.4061.
[47] C. P. Bonnington and C. H. Little, "Maps," in *The Foundations of Topological Graph Theory*. New York, NY, USA: Springer, 2012, pp. 23–35.
[48] H. Choset, E. Acar, A. A. Rizzi, and J. Luntz, "Exact cellular decompositions in terms of critical points of Morse functions," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, vol. 3, Apr. 2000, pp. 2270–2277, doi: 10.1109/ROBOT.2000.846365.
[49] D. Glavaški, M. Volf, and M. Bonkovic, "Robot motion planning using exact cell decomposition and potential field methods," in *Proc. WSEAS Int. Conf. Simul. Modeling Optim.*, Jan. 2009, pp. 126–131.
[50] M. Kloetzer and N. Ghita, "Software tool for constructing cell decompositions," in *Proc. IEEE Int. Conf. Automat. Sci. Eng.*, Aug. 2011, pp. 507–512, doi: 10.1109/CASE.2011.6042492.
[51] H. M. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, S. Thrun, and R. C. Arkin, "Cell Decompositions," in *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA, USA: MIT Press, 2005, p. 162, ch. 6.
[52] K. Mehlhorn, "Sorting," in *Data Structures and Algorithms 1: Sorting and Searching*, vol. 1. Berlin, Germany: Springer, 2013, pp. 40–101, ch. 2. [Online]. Available: https://www.worldcat.org/title/data-structures-and-algorithms-1-sorting-and-searching/oclc/310937425
[53] J. A. Storer, "Trees," in *An Introduction to Data Structures and Algorithms*. Boston, MA, USA: Springer, 2012, pp. 127–160, ch. 4, doi: 10.1007/978-1-4612-0075-8.
[54] F. Duchoň, A. Babinec, M. Kajan, P. Beňo, M. Florek, T. Fico, and L. Jurišica, "Path planning with modified a star algorithm for a mobile robot," *Proc. Eng.*, vol. 96, pp. 59–69, Aug. 2014, doi: 10.1016/j.proeng.2014.12.098.
[55] A. Y. Bhargava, "Introduction to algorithms," in *Grokking Algorithms: An Illustrated Guide for Programmers and Other Curious People*. USA: Simon and Schuster, 2016, pp. 10–19, ch. 1. [Online]. Available: https://edu.anarcho-copy.org/Algorithm/grokking-algorithms-illustrated-programmers-curious.pdf
[56] A. Niewola and L. Podsedkowski, "L* algorithm—A linear computational complexity graph searching algorithm for path planning," *J. Intell. Robotic Syst.*, vol. 91, nos. 3–4, pp. 425–444, Sep. 2018, doi: 10.1007/s10846-017-0748-6.

OMNIA A. A. SALAMA received the B.Sc. degree (Hons.) from the Department of Mechanical Engineering, Faculty of Engineering, Mechatronics Section, Assiut University, Assiut, Egypt, in 2016. She was a Teaching Assistant with the Department of Mechanical Engineering, Faculty of Engineering, Assiut University. Her research interests include mechatronics, path planning, robotics, and algorithms.

MOHAMED E. H. ELTAIB received the B.Sc. degree in mechanical design and production engineering from Mansoura University, Egypt, in 1986, the M.Sc. degree in mechanical engineering from Assiut University, Egypt, in 1993, and the Ph.D. degree in the area of tactile sensing for robotics and medical applications from the University of Dundee, U.K., in 2001. In 2001, he joined the Department of Mechanical Engineering, Assiut University, as an Assistance Professor, where he was an Associate Professor, from 2014 to 2019. From 2007 to 2018, he was on a sabbatical leave from Assiut University, where he joined the Mechanical Engineering Department, Qassim University, Qassim, Saudi Arabia. In July2019, he joined the Mechanical Engineering Department, Faculty of Engineering, Kafrelsheikh University, Egypt, as an Associate Professor. His research interests include CAD/CAM, mechatronics, tactile sensors for robotics and medical applications, haptic displays for virtual simulators, adaptive neuro-fuzzy inference systems, smart piezoelectric actuators, piezoelectric-based energy harvesting, nanopositioning, and robotic path planning.

HANY AHMED MOHAMED is currently the Chair with the Mechanical Engineering Department, Higher Technology Institute, Ramadan, Egypt. He has supervised and discussed many master's and doctoral theses (more than 30 theses) and teaching many undergraduate and postgraduate course. He has published more than 70 articles in different journals and participated in the number of 23 research published in scientific conferences. He received the State Incentive Award in Engineering Sciences from the Ministry of Higher Education and Scientific Research, Egypt, in 2003; the Scientific Excellence Award for Best Research in Mechanical Engineering, Assiut University, in 2005; and the participation, attendance, and presentation of research in many international scientific conferences.

OMAR SALAH received the B.Sc. degree (Hons.) from the Department of Mechanical Engineering, Mechatronics Section, Assiut University, in 2007, and the M.Sc. and Ph.D. degrees in mechatronics and robotics engineering from the Innovation Design Engineering School, Egypt-Japan University of Science and Technology (E-JUST), in 2012 and 2015, respectively. He has worked as an Exchange Researcher with Waseda University, Japan. He also worked as a Postdoctoral Research Fellow with LARICS, Zagreb University, Croatia. He is currently a Lecturer with the Department of Mechanical Engineering, Faculty of Engineering, Assiut University. His current research interests include mechatronics, robotics, path planning, and assistive devices.