Multi-Robot Coverage Path Planning in 3-Dimensional Environments

Nikolaos Baras, Minas Dasygenis, Nikolaos Ploskas
University of Western Macedonia
Department of Informatics and Telecommunications Engineering
Kozani 50131, Greece
nbaras@outlook.com, mdasyg@ieee.org, nploskas@uowm.gr

***Abstract*—Unmanned Vehicles are being used in several application domains, such as mapping, agriculture, and surveillance. In these application domains, the problem of finding a path that covers the entire Area of Interest (AoI) in a predefined environment is known as Coverage Path Planning (CPP). Even though many works have been focused on solving the CPP problem in 2D environments, the CPP problem in 3D environments has not attracted considerable attention. In this paper, we propose an algorithm capable of solving the CPP problem both in 2D and 3D environments. The algorithm can utilize multiple robots tailoring the coverage path for each robot based on its specifications, i.e., speed and type. We have performed an experimental evaluation of the algorithm in artificially synthetic environments and report the results.**

***Index Terms*—autonomous robot, path planning, coverage path planning, 2D coverage, 3D coverage**

I. Introduction

Coverage Path Planning (CPP) is the problem of determining a path that covers all points of an area or volume of interest while avoiding obstacles. CPP is a fundamental problem in robotics with numerous applications, such as agriculture and farming [1], [2], robot cleaning [3], and underwater operations [4]. CPP algorithms that can find a solution in the case of a single robot have to be revised in order to incorporate the multi-robot dynamics. Even though many works [5], [6] have targeted the CPP in 2D environments, these algorithms cannot be used in 3D environments. In this research, we propose an offline algorithm to solve the CPP problem in 3D environments. Our proposed algorithm is capable of solving the multi-robot CPP problem in two and three dimensional environments, tailoring each robot's path based on its technical specifications such as type and speed. Our algorithm is an extension of the DARP (Divide Areas based on Robots initial Positions) algorithm proposed in [5]. The main novelty of the algorithm is that the DARP algorithm has been extended for 3D environments and that the algorithm takes into account the different type and speed of each robot. We also developed an automated simulation tool to evaluate our algorithm and measure its efficiency.

The structure of the paper is as follows. In Section II, the background of this paper is presented along with related work. Section III presents the proposed algorithm. In Section IV, a comparison between the original DARP and our modified version is presented along with a preliminary computational study. Finally, the conclusions of this paper are outlined in section IV.

II. Related work

Autonomous robots popularity has risen steadily over the last decades [7]. Therefore, the CPP problem has attracted much attention. There are several different approaches attempting to solve the collision-free path planning problems in general, and the CPP in particular. A CPP algorithm for autonomous cleaning robots is introduced in [8]. They use an extension of boustrophedon cellular decomposition [9] combined with Dijkstra's algorithm [10]. Even though this algorithm produces an obstacle free coverage path in single-robot situations, it is inefficient with respect to memory and storage requirements. Gabriely and Rimon [11] proposed a spiral Spanning Tree Coverage (STC) algorithm based on the idea of dividing the environment into cells twice the size of the robot (Approximate Cell Decomposition) [12]. Their algorithm offers complete coverage of the AoI while avoding predefined obstacles. However, the proposed method is capable of utilizing only one robot at a time. Kapoutsis et al. [5] attempted to solve the CPP problem by equally dividing the AoI based on the number of robots available. After the total AoI is divided into multiple sub-areas, each robot uses a standard STC technique to cover its exclusive area. Even though their algorithm works in multi-robot situations, it does not take into account different types of robots and it cannot also be used in covering 3D environments.

Our work differs from the aforementioned ones in two key points: (a) we solve the multi-robot CPP problem in 3D environments, and (b) we take into account specific features of the robots, like speed and type. Moreover, the proposed algorithms solves the multi-robot CPP problem in both 2D and 3D environments using a single unified algorithm. Additionally, the constraint implementation is fully modular, meaning we can select on runtime which constraints apply to each robot. Since we introduce many constraints in the CPP problem, the execution time of the proposed algorithm is larger than the execution time of the aforementioned algorithms.

III. The Proposed Algorithm

The proposed algorithm consists of four main stages (Figure 1). The first stage deals with the initialization of the algorithm. The inputs to the algorithm include: the size and the topology of the environment, obstacle locations and robot specifications (such as type, speed, and initial positions). This data can directly be read from a file. The algorithm then stores all data in memory for faster access during runtime. In the second stage, the connected parts of the environment are being detected and labeled. The output of this stage includes a list of all the connected parts of the AoI. In the third stage, the total AoI is being divided into multiple sub-areas for each robot. This stage outputs two or three dimensional arrays containing information regarding the area assignment. In the case of one single robot, the algorithm directly assigns the total area to the robot without further calculations. Finally, in the fourth stage, for every calculated sub-area, the coverage path is being calculated. Outputs of the algorithm are: a graphical representation of the coverage path for each robot and an output file containing the coverage path.

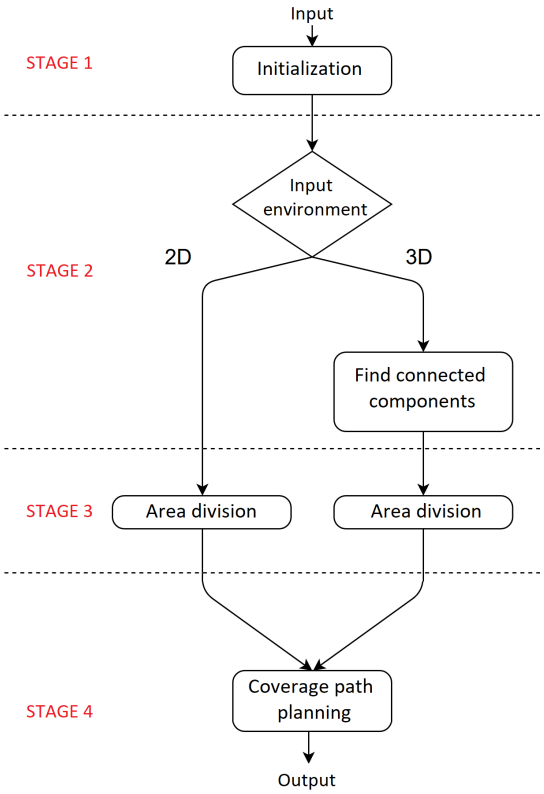
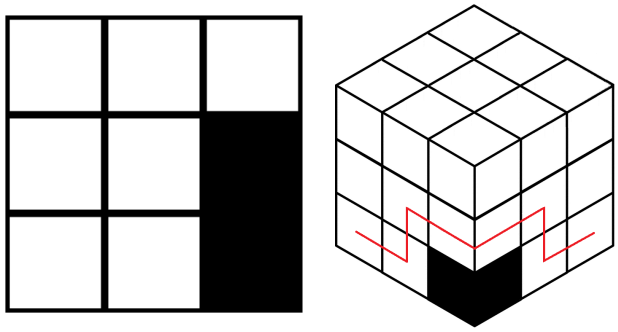


Fig. 1. The stages of the proposed algorithm.

In the **first stage**, the environment, the robots and their respective specifications are initialized. The environment is a three-dimensional Euclidean space consisting of multiple 3D cubicles (which we call votiles in contrast with tiles which 2D environments have). Every votile is placed on a (X, Y, Z) coordinate. A votile is the smallest space that can be occupied by any robot and every votile is equal with each other. Votiles can be obstacles or free space which robots can traverse. In 2D cases votiles have zero height. Obstacles are by definition areas that robots cannot traverse. In 3-D environments the obstacles may be located into various heights, obstructing for example road vehicles while allowing flying vehicles to pass over them. If the algorithm determines that flying over an obstacle is mandatory in order to find the solution for the problem, that path will be included in the final coverage path for the specific robot with that movement capability (Figure 2).



(a) (b)

Fig. 2. A 2D (a) and a 3D (b) environment input to the proposed algorithm. In (b), a robot with flying capability can fly over an obstacle.

Robot specifications include type and speed. In 2D environments, robots are by definition ground only vehicles. However, in 3D environments robots can be either on-ground vehicles, flying vehicles or underwater vehicles. Another characteristic of robots is their movement speed. The movement speed is defined as the rate at which the robot can move from one votile to another. Robots operating on an environment can have different movement speeds. For example, Robot A may have twice the speed of Robot B, meaning, that in the same amount of time, Robot A will cover twice the distance of Robot B. In other words, Robot A will visit twice the votiles compared to Robot B.

In the **second stage**, the non-connected ground areas of the AoI are being detected using a two-pass algorithm [13]. Even though a connection between these areas may not exist on ground level ($Z = 0$), a path connecting these areas may exist in higher levels ($Z > 0$). Using a Breadth First Search (BFS) algorithm, the distance and the path connecting non-connected ground areas are calculated. The BFS algorithm guarantees that the optimal path will always be found, if it exists. If such path does not exist, that means that the area is inaccessible and it is ignored for the rest of the execution. The output of this stage includes: (a) a list of the connected components of the AoI and (b) the distance and the path connecting them.

In the **third stage**, using the modified DARP (Divide Areas based on Robots Initial Positions) algorithm, the AoI is divided into a number of sub-areas each corresponding to a specific robot, so as to guarantee complete coverage of the total area and fully exploit the multi-robot dynamics. Since we are interested in the coverage of the projected 3D environment

(x, y, z) in 2D $(x, y, 1)$, we initially use the DARP algorithm to divide the first height layer of the total AoI in multiple sub-areas for each robot. Initially, DARP uses a votile to robot arrangement. An evaluation matrix E_i is maintained for every robot. This evaluation matrix indicates the distance between every votile in the AoI L and each robot's initial position. In order to calculate this distance we use the Euclidean distance formula (Eq. 1).

$$D_v = \sqrt{(x_v - x_0)^2 + (y_v - y_0)^2}, \forall V \in L \quad (1)$$

where (x_v, y_v) denote the coordinates of each votile and (x_0, y_0) denote the coordinates of the robot's initial position. On every iteration, the assignment matrix A is calculated using (Eq. 2). This matrix assigns each votile to a robot, based on the distance of each votile to each robot's initial position.

$$A_{x,y} = \min_{i \in \{1,\dots,n_r\}} E_i, \forall (x, y) \in L \quad (2)$$

where E_i is the evaluation matrix for the ith robot and n_r is the total number of robots. Subsequently, each robot's exclusive area L_i can be calculated directly by the assignment matrix A using (Eq. 3).

$$L_i = \{(x, y) \in L : (x, y) = i\}, \forall \{i \in (1, ..., n_r)\} \quad (3)$$

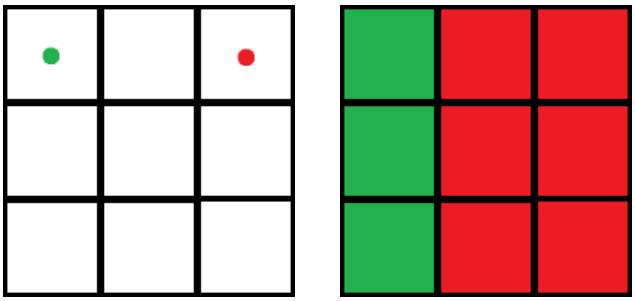
Moreover, the number of assigned votiles per robot k is defined as the cardinality of the L_i set (Eq. 4).

$$k_i = |L_i|, \forall i \in \{1, ..., n_r\} \quad (4)$$

Using the assignment matrix A, DARP will assign one votile to one robot only, and every votile will be assigned to some robot's sub-area. It is assumed that the initial position of each robot is assigned to the corresponding robot's sub-area.

As we already mentioned, at first, the evaluation matrices E_i contain distance information between each votile and the robots. The core idea of the DARP algorithm is that each evaluation matrix E_i can be modified in order to equalize the number of votiles k_i for each robot. In other words, the original DARP attempts to generate equal sized sub-areas for each robot. However, since in our case each robot may have different movement speed, a modification on the area division was required. A weight factor WF_i is introduced to determine the percentage of the total AoI that will be allocated to each robot. Robots capable of moving faster will correspond to a higher value of WF_i. For example, if robot A (green) is twice as fast as robot B (red), $WF = 2$ will correspond to robot A and $WF = 1$ will correspond to robot B. After the **completion** of our proposed algorithm, it is expected that robot A will be allocated twice the area of robot B (Figure 3).

The aforementioned operation guarantees that every votile will be assigned to a robot and that each robot's sub-area will be tailored to its speed. However, it does not guarantee the continuity of each robot's sub-area. DARP solves this issue by performing a procedure in which votiles located closer to each robot's sub-area are being rewarded while votiles located further away are being penalized, constructing gradually a closed-shape region. The very specifics of this procedure, however, are beyond the scope of this paper (see [5]).



(a) (b)

Fig. 3. In our initial setup, having a fast robot (red dot) and a slow robot (green dot) (a), our algorithm will effectively partition the space for each robot resulting in the best completion time (b).

In the **fourth stage** of the algorithm, after the total AoI is divided into multiple sub-areas, the Minimum Spanning Tree (MST) must be calculated for each sub-area. MST is a subset of the edges of a connected edge-weighted graph that connects all the vertices together without any cycles and with the minimum possible total edge weight. Since we only examine the coverage of the bottom layer in 3D environments, we can essentially reduce every 3D CPP problem into a 2D one and find the solution using a Spanning Tree Coverage (STC) technique [11]. Essentially, any MST algorithm can be used [14]–[17]. In our case, we used Prim's algorithm since it has been proven to be the fastest in dense graphs with a number of edges close to the maximal [18]. Our algorithm can be visualized in Figure 4.

IV. Discussion & Simulation Results

As we already mentioned, our proposed algorithm is an offline algorithm, meaning that the knowledge of the environment, robots positions and characteristics is considered already known. In order to evaluate our algorithm and conduct multiple experiments, the development of an automated simulation tool was necessary. Using Python, we developed an environment-generating tool that takes simple user inputs such as map size, robot location, robot type, and percentage of unoccupied votiles and outputs a JSON file describing the generated environment. This tool is capable of generating both two and three dimensonal environments. We then use the generated file as input to our proposed algorithm and automate the simulation process.

The environment we used to test the proposed algorithm was $[X, Y, Z] = 50 \times 50 \times 2$. 20% of its votiles were occupied by obstacles. All obstacles were located in random positions on the first height layer of the environment ($Z = 0$) The robot's initial positions were located at the edges of the environment. For this simulation, we used an Intel Core 2 Quad Q9550 CPU with 4GB of RAM. In Table (I), we can see the simulation results. The number of connected areas and robots only affect the total execution time by a small amount. The time required

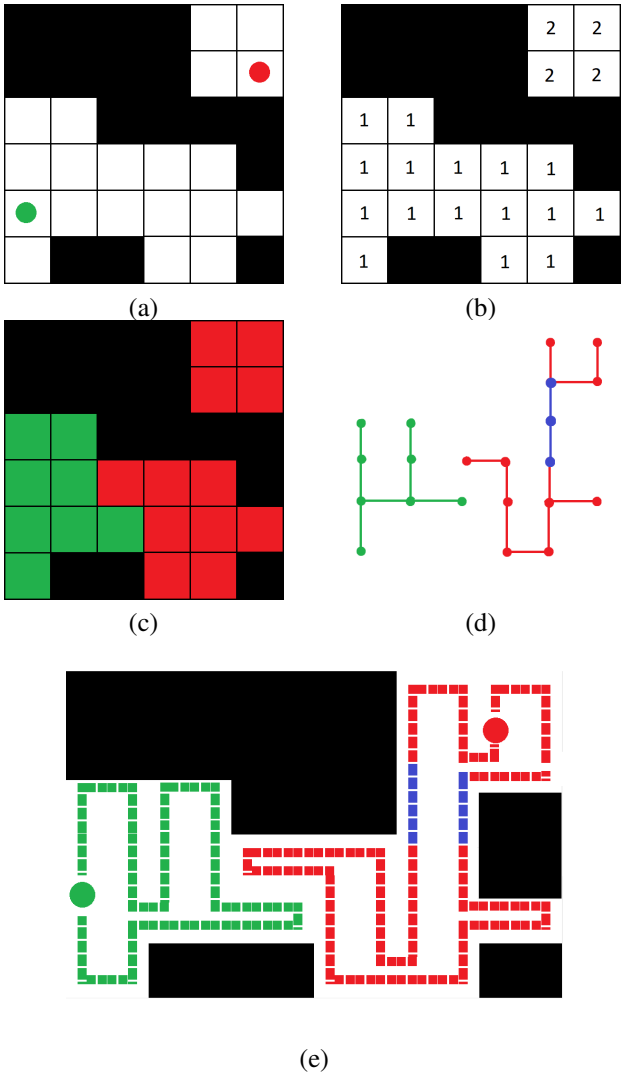


Fig. 4. In this example, our environment dimensions are $\{X, Y, Z\} = \{6, 6, 2\}$. Robot A (green dot) is a slow ground only vehicle and robot B (red dot) is a fast hybrid vehicle (ground and flying capable vehicle). Inaccessible votiles (obstacles) are marked as black squares and free space is marked as white squares. Inaccessible obstacles are located only on the first height layer of the environment. In (a), we can see the initial positions of robots A and B. In (b), we can see the connected component labeling. In (c), the area division is shown. We can see that robot B was assigned more votiles to cover compared to robot A because it is faster than robot A. In (d), we can see the generated MST for each sub-area (the area of the red robot's MST marked with blue color indicates that the robot will fly over the specific obstacle in the second layer of the environment. In (e), we can see the CPP for each robot using our modified DARP algorithm.

for the algorithm to find a solution is heavily dependent on the dimensions of the environment.

Conclusions

The CPP problem is a fundamental problem in robotics with numerous applications. In this paper, we proposed a unique approach that orchestrates the coordination of a multi-robot team, so as to completely cover a two or three dimensional environment taking into account the peculiar features of the robots, like type and speed. Our algorithm is fully modular, meaning we can modify robots' constraints on runtime. Using our simulation tool, we automated the evaluation process of our algorithm and measured its efficiency and scalability.

TABLE I
Simulation results for a 50×50×2 3D environment.

Number of robots	Connected areas	Execution time (s)
1	1	16.21
1	2	18.92
1	3	21.49
2	1	16.55
2	2	19.35
2	3	22.12

References

[1] M. Seder and I. Petrovic, "Complete coverage path planning of mobile robots for humanitarian demining," *Industrial Robot: An International Journal*, vol. 39, pp. 484–493, 2012.

[2] J. Jin and L. Tang, "Optimal coverage path planning for arable farming on 2d surfaces," *Transactions of the ASABE*, vol. 53, pp. 283–295, 2010.

[3] M. Kaur and P. Abrol, "Design and development of floor cleaner robot (automatic and manual)," *International Journal of Computer Applications*, vol. 97, pp. 32–38, 2014.

[4] Y. Deng, P.-P. J. Beaujean, E. An, and E. Carlson, "Task allocation and path planning for collaborative autonomous underwater vehicles operating through an underwater acoustic network," *Journal of Robotics*, vol. 2013, pp. 1–15, 2013.

[5] A. C. Kapoutsis, S. A. Chatzichristofis, and E. B. Kosmatopoulos, "Darp: Divide areas algorithm for optimal multi-robot coverage path planning," *Journal of Intelligent & Robotic Systems*, vol. 86, no. 3, p. 663–680, 2017.

[6] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics and Autonomous Systems*, vol. 61, p. 1258–1276, 2013.

[7] C.-Y. Chan, "Advancements, prospects, and impacts of automated driving systems," *International Journal of Transportation Science and Technology*, vol. 6, no. 3, pp. 208 – 216, 2017.

[8] M. Waanders, "Coverage path planning for mobile cleaning robots," in *15th Twente Student Conference on IT*, 2011.

[9] H. Choset and P. Pignon, "Coverage path planning: The boustrophedon decomposition," in *International Conference on Field and Service Robotics*, 1997.

[10] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, 1959.

[11] Y. Gabriely and E. Rimon, "Spanning-tree based coverage of continuous areas by a mobile robot," *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1, pp. 77–98, 2001.

[12] J.-C. Latombe, *Approximate Cell Decomposition*. Boston, MA: Springer US, 1991, pp. 248–294.

[13] A. Rosenfeld and J. L. Pfaltz, "Sequential operations in digital picture processing," *Journal of the ACM*, vol. 13, pp. 471–494, 1966.

[14] A. Mamun and S. Rajasekaran, "An efficient minimum spanning tree algorithm," in *2016 IEEE Symposium on Computers and Communication (ISCC)*, 2016, pp. 1047–1052.

[15] H. Li, Q. Xia, and Y. Wang, "Research and improvement of kruskal algorithm," *Journal of Computer and Communications*, vol. 05, pp. 63–69, 2017.

[16] K. Sörensen and G. Janssens, "An algorithm to generate all spanning trees of a graph in order of increasing cost," *Pesquisa Operacional*, vol. 25, pp. 219 – 229, 2005.

[17] V. Osipov, P. Sanders, and J. Singler, "The filter-kruskal minimum spanning tree algorithm," in *Proceedings of the Meeting on Algorithm Engineering & Expermiments*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2009, pp. 52–61.

[18] C. F. Bazlamaçcı and K. S. Hindi, "Minimum-weight spanning tree algorithms a survey and empirical study," *Computers and Operations Research*, vol. 28, no. 8, pp. 767 – 785, 2001.