Ground Robot Path Planning on 3D Mesh Surfaces Using Bi-directional RRT based on Local Regions

Cebisile Mthabela
School of Mechanical, Industrial and Aeronautical Engineering
University of the Witwatersrand
Johannesburg, South Africa
2035818@students.wits.ac.za

Daniel Withey
Centre for Robotics and Future Production
Council for Scientific and Industrial Research
Pretoria, South Africa
dwithey@csir.co.za

Chioniso Kuchwa-Dube
School of Mechanical, Industrial and Aeronautical Engineering
University of the Witwatersrand
Johannesburg, South Africa
Chioniso.Kuchwa-Dube@wits.ac.za

***Abstract*—The increasing application of ground robots requires efficient path planning algorithms in three-dimensional (3D) environments containing non-spherical topology. Path planning on surface meshes is possible, however, expensive computation of geodesics is required. To reduce the length and, hence, cost of the geodesics, a growing submesh based on local regions is used. Rapidly-exploring Random Trees (RRT) with local regions are computed and compared with the bi-directional variant, based on RRT-Connect. Results show that RRT-Connect with local regions reduces the computational burden for mesh-based path planning.**

Keywords—path planning, RRT, surface mesh, 3D environment, local regions, geodesics, nonspherical topology

I. Introduction

With the advancement of technology and research, the application of mobile robots has spanned into a wide range of fields including computational biology, computer animation and verification [1, 20]. As part of the navigation tasks, path planning is one of the fundamental tasks that a robot has to perform in order to move from one point to another and it is essential for a robot to perform this task effectively. Autonomous mobile robot path planning is not only a fundamental problem in robotics, but it is also a highly studied area because of its wide application in industries. Given a map, a starting position and a goal position, the path planning problem seeks to find an optimal collision-free path from the starting position to the goal position in configuration space.

A number of path planning algorithms have been developed in the past decades and these algorithms are usually categorised into groups based on how the environment is represented and explored. The two most known path planning categories are the grid-based algorithms and sampling-based algorithms. A* [13] and Dijkstra's [10] are both grid-based algorithms. These algorithms have been successfully used to find the shortest path in fairly flat environments, where the uncertainty associated with the environment is normally not considered. However, with the increasing application of robots in industries, robots share space with humans in real life, operating in uneven and unstructured environments. Thus, full 3D path planning is needed.

Path planning for ground robots has been extensively done in 2D environments and most 3D path planning studies have focused on aerial and underwater robots which are not constrained to the ground surface [29, 30]. Both aerial and underwater robots do not operate on the ground surface, hence there is no need to consider traversability on a surface. However, ground robots operate in uneven 3D environments with different running costs for the uphill, downhill and flat surfaces. The discretization of the workspace to a simpler version, such as a 2D environment, can result in the loss of important aspects about the robot's workspace leading to the robot's inability to successfully traverse complex cluttered environments.

Another approach that has been used in an attempt to solve 3D path planning for ground robots is adding elevation to get 2.5D, but still 2.5D is not a fully 3D environment and it is not sufficient for a robot to be able to operate in environments containing nonspherical topology, such as bridges or tunnels. Carsten et al. [5] solved path planning in 3D environments, by extending the Field D* algorithm to operate in 3D occupancy grids. The approach of Carsten et al. [5] was able to compute less costly and less jagged paths in the 3D grid. However, Carsten et al. [5] method was intended for aerial and underwater robots. Using D*, Colas et al. [7] developed a 3D path planning system for ground robots. Their method uses point cloud data for workspace representation; however, it uses a 3D grid representation for the environment, rather than a mesh.

As the dimensions of the workspace of the robot increase, the computation of the path becomes more challenging. Thus, using grid-based methods in high dimensional spaces can become computationally expensive and will take long to find a solution. These methods require grid-based discretization of the workspace which could lead to loss of important aspects about the environment leading to unrealistic solutions. Meanwhile, Sampling-based algorithms, such as RRT, have shown significant improvements in providing solutions in high dimensional spaces containing constraints, and can be faster than grid-based methods, in high-dimensional contexts. Lavalle [20] introduced RRT specifically for holonomic, nonholonomic and kinodynamic constraints in path planning problems. RRT discretely samples the configuration space [11] to find robot configurations within the free space with no need for workspace discretization. RRT was proven to work well in high dimensional spaces [16]; however, RRT tends to take much time to find a solution, if a solution exists, in heavily cluttered environments [28]. To improve the performance of RRT in cluttered environments, a bi-directional version of RRT, namely RRT-Connect, was introduced by Kuffner and LaValle [18]. RRT-Connect grows two RRT trees, one from the start and the other from the goal using the connect heuristic approach. RRT-Connect showed improvement in running time in uncluttered environments [18]. Both RRT and RRT-Connect do not pay any attention to the quality of the path produced. The path produced by them is sub-optimal. As an attempt to improve the path produced by RRT, Karaman and Frazzoli [16] introduced the first asymptotically-optimal sampling-based algorithm, RRT*, which produces less jagged and shorter paths when compared to RRT [23]. RRT* guarantees to return an asymptotically optimal path, if one exists, and it is one of the most used algorithms when dealing with optimal path planning

problems [23]. However, with its unique operations, tree rewiring and best neighbour search, RRT* takes more execution time than RRT.

3D surface meshes have become useful in the representation of large and cluttered environments that are not easy to represent using simple grids [15]. Surface meshes are popular in gaming, as they aid pathfinding in complex and cluttered environments. However, in computer gaming, 2D meshes with polygonal obstacles are mostly used [15]. Triangle meshes enable efficient computation for several navigation procedures [14]. A 3D mesh surface allows navigating through nonspherical topology such as tunnels. Moreover, 3D spaces can be directly represented on a mesh and that makes it easy to identify obstacles and the traversable spaces. Breitenmoser and Siegwart [3] presented a mesh construction approach for path planning for a climbing robot. The work of [3] uses only the triangle strips to implement a graph-based planner limiting the robot to only traverse through the edges of the triangle.

In this paper we will be using a 3D surface mesh for ground robot planning. These robots are known to be operating on the surface and, for a mesh, this requires the computation of discrete geodesic paths which can be computationally expensive in large meshes. A discrete geodesic path can be seen as a straight line connecting the set of points on the unfolded mesh. Finding the shortest paths in 3D surfaces is a well-known problem in computational geometry and arises naturally in applications such as robotics and geographic information systems.

RRT was implemented on a mesh in [22]. This method explores the entire mesh to find nodes that are leading to the goal configuration and, since it is computing geodesic paths, this becomes computationally demanding. Mthabela et al. [22] also introduced the local region method for mesh-based planning, which limits new nodes for tree expansion to within the local regions. This reduces the length of geodesic paths to be computed, leading to reduced computational time to find the desired path. This method is referred to as, RRT on a mesh with local regions (RRT-ML). To further improve the speed of RRT on a mesh, this paper adapts the bi-directional RRT method, RRT-Connect [19], for use on a mesh, and then compares it with RRT-ML, the best performing method from [22]. The RRT-Connect version is referred to as, RRT-CML. RRT-CML is based on the RRT-Connect of [18] and uses local regions to limit new nodes for the tree expansion to within the area of local regions. RRT-CML shows significant improvements in total runtime when compared to the single tree RRT-ML.

The contributions of this research paper are summarized as follows:

- An improved run-time of the mesh-based RRT planner by introducing a local region procedure which reduces the exploration space of the planner.
- Reducing the length and the number of geodesics to be computed in the mesh-based planner.
- Applicability of the proposed method in different 3D Mesh surfaces without being affected by the size of the mesh or the obstacles present in the Mesh.

The remainder of this paper is organised as follows: section II introduces RRT on a mesh surface with local regions (RRT-ML), geodesic distances, RRT-Connect on a mesh surface with local regions (RRT-CML). In section III, the simulation results of the RRT-ML algorithm and the RRT-CML are presented and discussed. The conclusion and future work are given in section IV.

II. METHODS

RRT's are amongst other sampling-based algorithms that have been used to solve path finding problems due to their ability to quickly explore the robot's workspace. Given the start configuration and the goal configuration, RRT incrementally grows the tree simply by randomly sampling the workspace to find feasible configurations and connecting the closest node into the randomly selected node. Fig. 1 adapted from [21] shows the tree expansion process, where q_{new} is being added into the tree. The random node q_{rand} is sampled at each iteration. The random node is only added into the tree if the distance between the random node and the nearest node q_{near} is obstacle free and is within the specified step size, ϵ. Otherwise, a new node q_{new} to be added in the tree is derived using a steering function.

A. Discrete geodesic paths on mesh surfaces

A discrete geodesic path can be seen as a straight line connecting the set of points on the unfolded mesh that determines the local shortest path between the points. The computation of discrete geodesic paths is a well-known problem in computational geometry, and it arises naturally in fields such as robotics and geographic information systems [28]. Computing shortest paths in a 3D mesh surface with obstacles is generally an NP-hard problem [4]. Computation of paths for ground robots on 3D surface meshes is an application of discrete geodesics. Depending on the number of source points and target points, the computation of geodesic paths and distances can be done differently. A geodesic path can be computed between two given mesh points, i.e. only one source point, and one target point, or it can be between a single point and many points. The latter is commonly known as a single-source shortest path problem. The geodesic paths can also be computed between all pairs of points

With the aid of the Triangulated Surface Mesh Shortest Paths package [17] in the Computational Geometry Algorithms Library (CGAL) [12], this paper solves a single point to many points geodesic problem on a mesh surface. Based on the algorithm introduced by Xin and Wang [26], the Triangulated Surface Mesh Shortest Paths package computes the geodesic paths from any source point in a triangle mesh to a selected target point in the mesh by constructing a sequence tree (T) which contains nodes/vertices and edges of the mesh. Thus, locally shortest paths between nodes can be computed. Given a mesh surface M, the algorithm of Xin and Wang [26] finds an exact shortest path λ_s, which is constrained to the surface of M, between the target point t and a source point $s \in S$, where S is the set of all source points.

Fig. 1. RRT tree expansion.

B. RRT on a mesh surface with local regions (RRT-ML)

A local region-based RRT on a mesh (RRT-ML), which allows for the reduction of geodesic length and number of paths was introduced in Mthabela et al. [22] as an attempt to reduce the overall computational time of RRT on a mesh surface. This method uses local regions as the exploration area for RRT. Initially the random points are selected within the local region of the first tree node, which is the area of radius r around the tree node. A local region of a point p consists of triangles/faces that are within the area less than πr^2, centred at p. As the tree grows, local regions around each node in the tree are combined to form a subset of the mesh called sub-mesh. An example of a sub-mesh embedded in the original mesh is depicted in Fig. 2, where the original mesh is the gray area and the sub-mesh is the pink area.

The start and the goal configurations are within the sub-mesh. The sampling of random point q_{rand} is done within the sub-mesh which reduces the length of shortest paths to be computed and when searching for the nearest neighbour q_{near} only the tree nodes that are found within a local region around q_{rand} which is the area of radius $rand_r$ are considered. As a result the number of shortest paths to be computed is reduced. RRT-ML pseudocode is given by Algorithm 1. To speed up the performance of RRT-ML, the goal biasing factor [25], which replaces q_{rand} by q_{goal} with a probability p_{goal} is used, to draw samples towards the goal. The steering function determines an obstacle-free path segment between q_{near} and q_{rand} which is one step size ϵ or less from q_{near}. Any triangle face with a normal vector orientation that is greater than or equal to 30° from the vertical is considered as an obstacle, and, therefore, the path cannot pass through that face.

Planning on a mesh surface is more challenging than planning in 2D, all tree nodes and edges must lie on the mesh surface. Triangulated Surface Mesh Shortest Paths package [17] from CGAL ensures that all paths and tree nodes are fixed on the surface of the mesh while computing the exact geodesic paths. Computation of geodesic path in RRT-ML is performed often, thus causing RRT-ML to take long to find the path since the process of finding geodesic paths on a mesh is computationally expensive [22].

Fig. 2. A union of local regions.

C. RRT-Connect on a mesh surface with local regions (RRT-CML)

RRT-Connect on a mesh can speed up the process of finding the solution. The underlying classical RRT in RRT-Connect still has the tendency to explore a vast area in a robot's workspace, thus resulting in a large number of possibly long geodesic paths to be computed. The performance of RRT-Connect on a mesh can be improved through the introduction of local regions to restrict the sampling space into a subset of the mesh, which reduces the exploration space for the planner into a growing sub-mesh instead of the entire mesh surface. In this section, RRT-Connect with local regions (RRT-CML) is discussed. Since this approach grows two trees alternately, one rooted at the initial configuration and the other at the goal configuration, both initial and goal configurations are initialized with local regions of radius, r. As each tree grows towards the other, two subsets of local regions are created and later intersect. As soon as these local regions intersect, the connection point, which belongs to both trees will be found and the path from the initial configuration to the goal configuration will be obtained. Depending on which tree is being expanded in each iteration, these subsets are interchanged when sampling for a random point. The nearest neighbour search is done through the computation of shortest geodesic paths between tree nodes and the random point. Again this is done in each tree expansion, everything that was done for a single tree in RRT-ML in the previous section is done twice for the RRT-CML.

The EXTEND and REACHED procedure used for the RRT-CML are given by the pseudocode in Algorithm 3 and Algorithm 4, respectively. The function NEWCONFIG (Algorithm 2) used in this method is slightly different from the one used in the basic RRT-Connect [18]. The pseudocode for RRT-CML is given in Algorithm 5. This is the main algorithm taking in a mesh $M=\{R,T\}$ as input, with t_1 and t_2 being proper subsets of T. Since growing two trees alternately, two unions of local regions are also grown alternately. Two lists are created for this purpose, $list_1$ in line 3 of Algorithm 5 initially contains nodes that are within the local region of the starting configuration q_{init} and is grown simultaneously with the first tree (t_1), $list_2$ in line 4 of Algorithm 5 initially contains nodes that are within the local region of the goal configuration q_{goal} is also grown simultaneously with the second tree (t_2). By combining $list_1$ and $list_2$ a SubMesh is obtained, since $list_1$ is a set difference of SubMesh and $list_2$; and $list_2$ is a set difference of SubMesh and $list_1$. In line 7, REACHED function checks if the connection point is found or not found while searching in $list_1$'s local region area. If the connection point is found (line 8), the local region of the point q_{mid} is added into $list_1$ (line 9) then the path is extracted in line 10 and returned in line 11. The planner will stop at this point. Otherwise, if line 9 returned false, the process is repeated from line 12 with t_1 replaced by t_2 and vice-versa and $list_1$ replaced with $list_2$. REACHED function is called again in line 13 to check if the connection point is found now searching in $list_2$'s local region area. If the connection point is found (line 14), the local region of the point q_{mid} is added into $list_2$ (line 15) then the path is extracted in line 16 and returned (line 17) and the planner stops. Otherwise, the process is repeated and if the connection point is not found after N iterations, the empty path is returned in line 21.

III. RESULTS

All experiments were computed using C++ on a Dell OptiPlex-7050 computer with Intel® Core™ i7-7700 CPU @ 3.60GHz processor and 8 GB memory.

For simulations a synthetic mesh which consists of a tunnel and a bridge, representing an environment with non-spherical topology, was initially created using Blender v2.82 [8] and processed in MeshLab [6]. This synthetic mesh has 18362 triangles and 9330 vertices and two different slopes of angles 16° and 35° respectively. To evaluate the methods presented in this paper, we used a set of starting and goal positions. This is motivated by the application of path planning in mobile robots where multiple paths have to be computed as part of navigation. The path for each different starting and goal position pair could have the same distance but the time taken for the planner to find the path may differ, due to the fact that the planner might encounter obstacles in one route that are not necessarily on the other route. For all simulations, the step size $\epsilon=1m$ and the maximum number of iterations is 1500. For the RRT-ML the goal biasing factor is 0.2, representing the probability of using the goal in place of q_{rand}, for new point selection. For the local region method the radius $r=1$ is used for all tree nodes and $r=2$ for each random point q_{rand}. Any face having a normal vector orientation with angle θ from the vertical of 30° or above is considered as an obstacle, therefore the planner cannot pass through that face.

Table I shows the result of RRT-ML and RRT-CML for different sets of starting and ending positions, where average iterations is the total number of iterations that each of these planners took to find the path, averaged over five trials. To evaluate the performance of these methods on a mesh, the total number of iterations taken by each method to find the path is also considered. In Table I the average time per iteration is the time taken by RRT-ML and RRT-CML to complete one iteration and the average time is the total runtime of the planner, each averaged over five trials. The

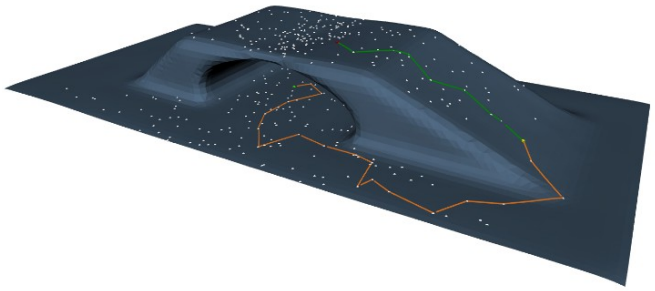
average path length indicates the average distance of the path produced by each of the planners for each case over five trials. Both RRT-ML and RRT-CML were tested on the synthetic mesh described above with the same parameter set up. RRT-CML was also evaluated on a real mesh containing 67263 vertices and 133822 faces. The mesh was created using the colourised 3D LiDAR SLAM point cloud selection collected by Cox et al. [9]. The original dataset has 3521563 vertices. For this simulation, a section of this data containing a ramp with a slope of approximately 5.9° was used.

Fig. 3, shows the tree (orange) grown by RRT-ML. RRT-ML is able find a feasible path in a 3D environment containing nonspherical topology. For the case 2 in Table I, RRT-ML was able to find the path after 40 iterations in 8.61 seconds, on average. However, the average execution time is higher than that for RRT-CML, since goal biasing requires computation of shortest paths to all RRT nodes. The tree is grown on traversable parts of the mesh and growing on obstacles, such as walls, is intentionally avoided.

Fig. 3. RRT-ML tree for Case 4 with a starting point in green and the goal point in red. The orange lines are the RRT tree.

Fig. 4. Path produced by RRT-ML for Case 4 with a starting point in green and the goal point in red.

Growing two trees alternately has had a tremendous result in improving the performance of RRT [18]. Fig. 6



shows the tree produced by RRT-CML on a mesh. Since this method grow two trees alternately at each iteration, the need for goal biasing was removed. Looking at the results in Table I, it is quite clear that the bi-directional RRT version quickly completes its search. Taking case 2, for example, RRT-CML reached the goal point in almost half the iterations required by RRT-ML and in less than half the time, required by RRT-ML. As shown in the table, in all cases, RRT-CML completed the planning tasks in less time than that required by RRT-ML.

RRT-CML was also tested on a real mesh which is different from the synthetic mesh in terms of the total number of faces and vertices it has and the density of points. Fig. 7 shows RRT-CML on a real mesh.

IV. Conclusion

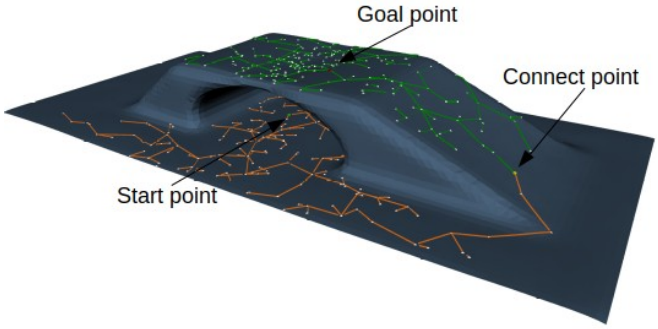
Path planning in 3D environments for ground robots is a challenging task since the path must lie on the surface. Planning on a mesh requires the computation of geodesic paths to ensure that all paths lie on the surface of the mesh. Through the use of CGAL, computation of geodesic paths on a mesh was enabled; however, determining geodesic paths is computationally expensive. By introducing local regions, the exploration space for RRT is reduced to a growing sub-mesh, thus reducing the length and number of geodesics to be computed. As a result, the runtime of the algorithm is reduced.

Fig. 5. RRT-CML tree for Case 4 with a starting point in green and the goal point in red. The first tree is orange and the second tree is green.

Fig. 6. Path produced by RRT-CML for Case 4 with a starting point in green and the goal point in red. The point connecting the two trees, one orange and the other green, is in yellow.

Fig. 7. RRT-CML on a real mesh. The starting point is in green and the goal point is in red. The point where the first tree (orange) and the second tree (green) connect is shown in yellow.

To further reduce the runtime of the planner on a mesh surface, RRT-CML, which grows two trees alternately, removing the need for goal biasing, was developed and demonstrated on a synthetic and on a real mesh. RRT-CML outperformed the RRT-ML method by finding a path in fewer iterations and in less time, on a synthetic mesh. RRT-

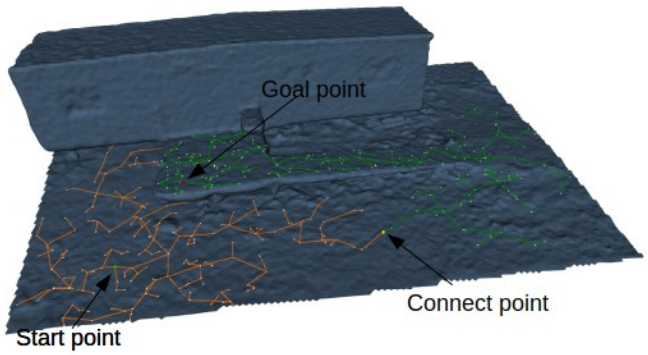


CML operation was also shown in the context of a real 3D surface mesh, showing capabilities to work well in real world problems.

In future research the local region approach could be applied in the asymptotically optimal planner, RRT*, to obtain optimal solutions. The local regions concept may aid the near neighbour search and rewiring processes of the RRT* since it can be used to create the ball of radius k used by RRT*. This will allow nodes to be selected for the nearest neighbour and rewiring processes, allowing the RRT* algorithm to iteratively minimize path cost.

References

[1] A. Bhatia and E. Frazzoli. Incremental search methods for reachability analysis of continuous and hybrid systems. In International Workshop on Hybrid Systems: Computation and Control, pages 142–156. Springer, 2004.



[2] P. Bose, A. Maheshwari, C. Shu, and S. Wuhrer. A survey of geodesic paths on 3d surfaces. Computational Geometry, 44(9):486 – 498, 2011.

[3] A. Breitenmoser and R. Siegwart. Surface reconstruction and path planning for industrial inspection with a climbing robot. In 2012 2nd International Conference on Applied Robotics for the Power Industry (CARPI), pages 22–27, 2012. doi: 10.1109/CARPI.2012.6473354.

[4] J. Canny and J. Reif. New lower bound techniques for robot motion planning problems. In 28th Annual Symposium on Foundations of Computer Science (sfcs 1987), pages 49–60, 1987. doi: 10.1109/SFCS.1987.42.

[5] J. Carsten, D. Ferguson, and A. Stentz. 3D Field D*: Improved path planning and replanning in three dimensions. In Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 3381–3386, 2006.

[6] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In

V. Scarano, R. D. Chiara, and U. Erra, editors, Eurographics Italian Chapter Conference. The Eurographics Association, 2008. ISBN 978-3-905673-68-5.

[7] F. Colas, S. Mahesh, F. Pomerleau, M. Liu, and R. Siegwart. 3D path planning and execution for search and rescue ground robots. In 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 722–727, 2013. doi: 10.1109/IROS.2013.6696431.

[8] B. O. Community. Blender - a 3D Modelling and Rendering Package. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. URL http://www.blender.org.

[9] M. Cox, P. Borges, and T. Lowe. Paintcloud. 3, 2018/06// 2018. URL https://doi.org/10.4225/08/5b31c59fca196.

[10] E. W. Dijkstra. A note on two problems in connexion with graphs. NUMERISCHE MATHEMATIK, 1(1):269–271, 1959.

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

[12] A. Fabri and S. Pion. CGAL: The computational geometry algorithms library. In Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pages 538–539, 2009.

[13] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics, 4(2):100–107, 1968. doi: 10.1109/TSSC.1968.300136.

[14] M. Kallmann. Navigation queries from triangular meshes. In Motion in Games, pages 230–241. Springer, 2010.

[15] M. Kallmann and M. Kapadia. Navigation meshes and real-time dynamic planning for virtual worlds. In ACM SIGGRAPH 2014 Courses. Association for Computing Machinery, 2014. doi: 10.1145/2614028.2615399.

[16] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. The International Journal of Robotics Research, 30(7):846–894, 2011. doi: 10.1177/0278364911406761.

[17] S. Kiazyk, S. Loriot, and E. C. de Verdiere. Triangulated surface mesh shortest paths. In CGAL User and Reference Manual. CGAL Editorial Board, 5.1 edition, 2020. URL https://doc.cgal.org/5.1.1/Manual.

[18] J. J. Kuffner and S. M. LaValle. RRT-Connect: An efficient approach to single-query path planning. In Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065), volume 2, pages 995–1001, 2000. doi: 10.1109/ROBOT.2000.844730.

[19] J.-C. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. The International Journal of Robotics Research, 18(11):1119–1128, 1999. doi: 10.1177/02783649922067753.

[20] S. M. Lavalle. Rapidly-exploring random trees : a new tool for path planning. The Annual Research Report, 1998.

[21] S. M. Lavalle and J. J. Kuffner, Jr. Rapidly-exploring random trees: Progress and prospects. In Algorithmic and Computational Robotics: New Directions, pages 293–308, 2000.

[22] C. Mthabela, D. Withey, and C. Kuchwa-Dube. RRT based path planning for mobile robots on a 3d surface mesh. In 2021 Southern African Universities Power Engineering Conference/Robotics and Mechatronics/Pattern Recognition Association of South Africa (SAUPEC/RobMech/PRASA), pages 1–6, 2021.

[23] I. Noreen, A. Khan, and Z. Habib. A comparison of RRT, RRT* and RRT*-smart path planning algorithms. International Journal of Computer Science and Network Security (IJCSNS), 16(10):20, 2016.

[24] C. Petres, Y. Pailhas, P. Patron, Y. Petillot, J. Evans, and D. Lane. Path planning for autonomous underwater vehicles. IEEE Transactions on Robotics, 23(2):331–341, 2007. doi: 10.1109/TRO.2007.895057.

[25] C. Urmson and R. Simmons. Approaches for heuristically biasing RRT growth. In Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453), volume 2, pages 1178–1183 vol.2, 2003. doi: 10.1109/IROS.2003.1248805.

[26] S.-Q. Xin and G.-J. Wang. Improving Chen and Han's algorithm on the discrete geodesic problem. ACM Transactions on Graphics (TOG), 28(4):1–8, 2009.

[27] S. Xu, D. Honegger, M. Pollefeys, and L. Heng. Realtime 3D navigation for autonomous vision guided MAVs. In 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 53–59, 2015. doi: 10.1109/IROS.2015.7353354.

[28] L. Yang, J. Qi, D. Song, J. Xiao, J. Han, and Y. Xia. Survey of robot 3D path planning algorithms. Journal of Control Science and Engineering, 2016. ISSN 1687-5249. doi: 10.1155/2016/7426913.

[29] Y. Sun, X. Ran, G. Zhang, H. Xu, X. Wang. AUV 3D Path Planning Based on the Improved Hierarchical Deep Q Network. Journal of Marine Science and Engineering. 2020; 8(2):145. https://doi.org/10.3390/jmse802014.

[30] J.L. Sanchez-Lopez, M. Wang, M.A. Olivares-Mendez, M. Molina and H. Voos. A Real-Time 3D Path Planning Solution for Collision-Free Navigation of Multirotor Aerial Robots in Dynamic Environments. J Intell Robot Syst **93,** 33–53 (2019). https://doi.org/10.1007/s10846-018-0809-5