Mapping for Unknown Environments Using Triangulated Maps

Amna AlDahak, Lakmal Seneviratne, Jorge Dias
Khalifa University of Science, Technology & Research (KUSTAR), UAE
amna.aldahak@kustar.ac.ae, lakmal.seneviratne@kustar.ac.ae, jorge.dias@kustar.ac.ae

***Abstract*—In this paper we present a new geometrical mapping structure that captures both the geometry and the connectivity of the environment. A robot's ability to successfully complete a required task is bound by its knowledge about the operation environment. Thus, the robot must be able to collect information from its surrounding and map it accurately to create a correct and complete representation of the environment. The solution in this paper uses the Gap-Navigation Tree as the underlying structure for the proposed Triangulation-Based exploration which maps the environment using the Dynamic Triangulation Tree structure DTT developed in this study. The efficiency of the proposed strategy is validated experimentally through simulations. The DTT does not only embed the geometry of the environment but also provides a direct mapping of the connectivity of its free space. The proposed algorithm is tested in simulations using various scenarios for exhaustive validation to prove its main advantages, namely ease of construction, compactness and completeness.**

***Index Terms*—Unknown Environment, Mapping, Triangulation.**

I. Introduction

Robot's knowledge about its surrounding environment is a core requirement to successfully achieve its tasks. Acquiring this knowledge becomes more challenging when the robot operates in an unknown environment. An increasing number of challenging applications require robots to explore and map unknown environment. Various number of map representations are available in the literature, some produce topological maps and some geometrical. Topological mapping completely captures the connectivity of the free space but provides no (or minimal) knowledge about its geometry. On the other hand, geometrical representations usually are concerned with the shapes and distances of the environment boundaries and mostly do not directly address the free space connectivity.

In this paper we present a new geometrical map representation that embeds both geometrical details of the environment and free space connectivity. The proposed solution uses the Gap-Navigation Tree (GNT) as the underlying structure for the proposed Triangulation-Based exploration which maps the environment and saves it in a tree-like structure. Using triangles to store the geometry of the environment will significantly reduce the storage space required when compared to the occupancy grids used in many exploration and map building solutions as in [1]. Moreover, the triangles stored in the proposed structure are connected to reflect the connectivity of the environment for easy path generation and navigation.

The remaining of this paper is structured as follows: Section II presents related research efforts. Section III discusses the proposed solution with detailed algorithms for all of its modules. The results presented in Section IV validate the effectiveness of the proposed solution. Finally, Section V draws the conclusion and presents future research directions.

II. Related Efforts

In this section we will present an overview of common map building solutions available in the literature.

One of the most commonly used map representations is the occupancy grid. The main idea in this approach is to divide explored areas to cells of equal size labelled according to their occupancy as obstacle, free or unknown. An early detailed study on its usage can be found in [2] which presented it in a probabilistic frame in an effort to make robotics navigation solutions more practical and applicable in real world scenarios. One of the early solutions for unknown map exploration that uses occupancy grids came years later and was introduced in [3]. This solution modelled the explored environment as a 2D occupancy grid. The work in [4] uses coverage maps which is a modified version of 2D occupancy grid designed to increase the accuracy of the produced map. More recent research was presented in [5] for a single-robot system with 3D occupancy grid. Generally, two problems arise when using occupancy grids. The first is the accuracy of the map which is dependent on the grid resolution, the smaller the cells size the more accurate it is but the larger the memory it requires. The second issue is storage and computational overhead. As the size of the environment increases the total grid size also increases causing an overhead on memory and computation when generating paths for navigation.

To overcome the limitations inherited when using occupancy grid, more researchers have worked toward topological mapping to reduce the memory required and to minimize the computational overhead. An interesting multi-robot solution for fire search and rescue that uses topological mapping was presented in [6]. As indicated by its authors, the topological map was insufficient and their solution can be enhanced by adding geometrical mapping which will create a useful mixture of both toward a more practical implementation.

The Gap Navigation Tree GNT is a very simple yet effective topological mapping which was originally designed for applications that require minimal sensing [7]. Thus, it only sense discontinuities in the incoming sensory readings and refers to them as $gaps$. Our approach uses the GNT as the guide to unexplored areas of the free space but the produced

Dynamic Triangulation Tree DTT in this work contains sufficient information about the partial map and its connectivity. Moreover, our proposed map representation compresses the environment geometrical features through the use of triangles. For a given environment the number of triangles used to represent its geometry can be significantly less than the number of cells used to represent it as an occupancy grid which is verified in Section IV. For a more comprehensive overview of Robotics mapping see [1] and [8].

In the remaining of this section, a detailed overview of the GNT, its information state, and its construction process is included.

A. Gap Navigation Tree (GNT)

The Gap Navigation Tree (GNT) is a dynamic data structure constructed from continuous online sensor readings of a robot while navigating and exploring an unknown environment. When the exploration is complete the GNT completely captures the topology of the environment [7], [9].

The visual events the GNT uses to progressively map the environment have its roots in the well-known *art-gallery* problem which was introduced in [10] in 1987. It presents a map decomposition method that divides a given environment by extending the inflection and bi-tangent rays from its boundaries. This will decompose the environment into a set of cells known as *aspect cells* that will be arranged in an *aspect graph* that reflects the connectivity of the environment. No major changes happen to the robots *visibility region* as it moves within the same aspect cell. Only by crossing boundaries between the cells, major changes occur in its visibility. The same visual events are used in the construction process of the GNT as detailed later.

1) Robot Model: The robot is modelled as a point moving in a 2D simply connected plane R. The robot needs to detect the discontinuities in depth information which is referred to as *gaps*. The robot's sensor, which could be physically implemented using various available sensors, is required to report the circular list of the current gaps from the current position and orientation of the robot.

Behind each gap there is a hidden region that is currently invisible to the robot. As the robot moves in the environment it crosses inflection and bi-tangent rays causing its *visibility region* to change. These events are known as *critical events*.

2) GNT Information Space: Given the robot model defined earlier, the sensing space contains all the different readings that can be obtained from the sensor. Therefore, for any state $x \in R$, the list of reported gaps from the sensor is denoted by $G(x)$. For example in Figure 1, the sensor reading will be $G(x) = [g_1, g_2, g_3]$. The $G(x)$ is updated by the sensors readings; thus when a gap is being chased, the readings need to be updated to include any changes in gaps. Thus the action space contains only the $chase(g)$ command for $g \in G(x)$ where $x \in R$.

3) GNT Construction: As the robot chase gaps and updates its sensors information, the GNT of the environment T is constructed simultaneously. The root of T always refers to the current robot position in the environment. Thus, at the beginning of exploration, the GNT contains the root $T(0)$, which is the initial robot position. The vertices connected to the root will be the current gaps in $G(T(0))$ in their circular order. The illustrated example in Figure 1 depicts a tree containing the root and three vertices for the detected gaps.

During gap chasing, the robots continuously updates T based on the observed visual events. When critical events occur T is updated according to the type of event. The observed critical events could be a one of four. The first is the case of **gap appearance**, which occurs when an already explored area becomes occluded. In this case a *primitive* gap vertex will be added to T. Primitive gaps indicate areas that have been explored and are not candidates for further exploration. The second critical event is **gap disappearance** which causes the removal of the corresponding gap vertex from T. Third is the **gap merge** case which triggered the addition of a new gap vertex with the two merged gaps added as its children. The fourth and last case is the **gap split** which is basically the opposite of the merge. This case causes the original gap vertex to be replaced by the two newly detected gap vertices in T. Given the above description, we can conclude that T encodes

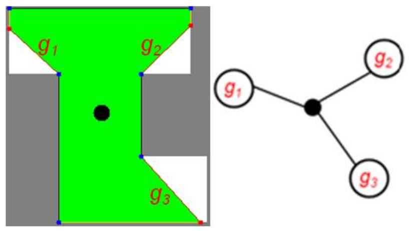


Fig. 1. A robot in a simple environment with the acquired visibility region (left), the corresponding initial T (right)

the sensing history throughout the exploration process. The GNT is complete when all its leaves are primitive vertices which means that all areas have been explored [7], [9].

III. The Proposed Method

In this section we present two main algorithms, namely the exploration and triangulation algorithms. The first uses the GNT to guide the robot to unexplored areas while mapping the environment topologically. The later perform the required updates after each chase command to map the environment geometrically in our proposed data structure, the DTT.

We will start by describing the robot and environment models adopted in this work followed by details of both algorithms.

A. Robot and Environment Model

The robot is modelled as a holonomic robot equipped with 360 degrees sensing range achieved by using two laser range scanners each of which will provide the distances of 180 degrees. Range scanner data will be used to produce the *visibility region*.

The environment R is a two dimensional simply-connected continuous environment. $VisRegion(x)$ is the *visibility region* acquired by the robot at state $x \in R$ containing distances and discontinuities in the sensed region. The discontinuities in the range scanner data are labelled in $VisRegion(x)$ as *gap edges* (see Figure 2).

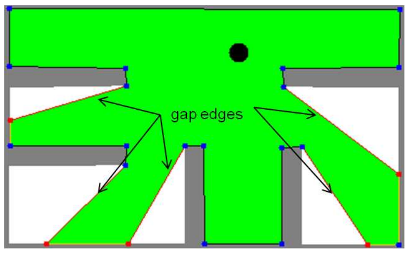


Fig. 2. In this figure the robot is modelled by the black disk in the middle of the green (shaded) region which resembles the 360 degrees *visibility region* of the robot.

B. Incremental Map Building

The two main algorithms , the *Build_Map* algorithm and the *Update_Triangulation* algorithm, introduced in this section topologically map the environment in the GNT and simultaneously produce a complete and compact geometrical map of the environment embedded in the DTT structure.

The main idea in the proposed DTT mapping is to incrementally construct a triangulated geometrical representation of the environment by iteratively dividing the current $VisRegion$ into a series of connected triangles arranged in a simple yet efficient tree structure. Each triangle in the tree embeds information about its geometry and its relation with the surrounding triangles eventually forming a tree structure that can be easily traversed for feasible paths to locations of interests in the mapped environment at any instance of time during mapping despite the incompleteness of the environment map.

In the DTT there are two types of triangles, namely *primitive* and *non-primitive* triangles. Primitive triangles vertices are all boundary vertices, i.e. points and corners on the actual physical boundary of the operation environment. while non-primitive triangles have at least one non-boundary vertex, i.e. a gap vertex at the end of a gap edge in the current (or previously sensed) $VisRegion$. Therefore, non-primitive triangles are those adjacent to unexplored areas of the environment.

Next, we detail the presented algorithms in details and follow that by experimental simulations to evaluate the algorithms performance.

1) Map Building Algorithm: The proposed mapping method is depicted in Figure 3. As can be interpreted from the figure, the mapping starts by initializing the map with the first acquired *visibility region*. After the initialization step we can roughly say that a partial map of the environment exists. Next, the algorithms iteratively selects and chase a gap, acquire the new *visibility region*, updates the constructed map. If more unexplored gaps still exist the algorithm iterates until no more unexplored gaps exist. The *Build_Map* algorithm initializes the GNT with the found list of discontinuities (gaps) extracted from the first acquired $VisRegion$. The DTT is also

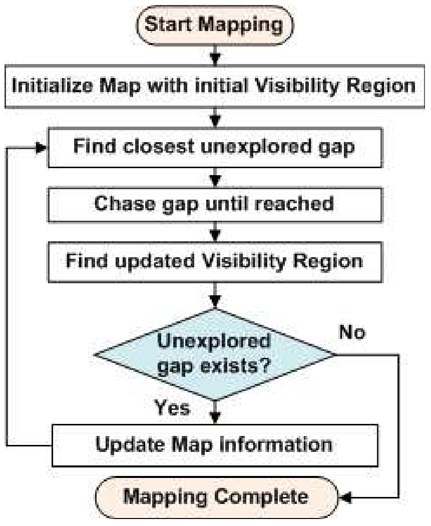

Start Mapping
Initialize Map with initial Visibility Region
Find closest unexplored gap
Chase gap until reached
Find updated Visibility Region
Unexplored gap exists?
No
Yes
Update Map Information
Mapping Complete

Fig. 3. The *Build_Map* algorithm flow chart.

initialized by triangulating the first $VisRegion$ forming the initial tree structure containing non-primitive triangles adjacent to the unexplored areas that are subject for removal and re-triangulation in later iterations.

During each iteration, $VisRegion$ will be passed to the *Update_Triangulation* algorithm to perform the required updates on the current DTT. The closest unexplored gap is selected and the robot is instructed to move toward it. When gap is reached the list of discontinuities $G(x)$ is updated based on the new location. The algorithms iterates until both GNT and DTT are complete.

```
Algorithm Build_Map(G(x))
Input: The initial set of gaps $G(x)$ acquired from the sensory
    information.
Output: The complete $GNT$ and $DTT$.
1.  initialize $GNT$ with $G(x)$
2.  initialize $DTT$ to empty
3.  while $\exists$ $nonprimitive\ g_i \in GNT$
4.      do find $VisRegion(x)$
5.          $DTT$ = Update_Triangulation($DTT$, $VisRegion(x)$)
6.          $g_k$ = closest unexplored gap
7.          chase $g_k$ until gap is reached
8.          update $G(x)$ according to the new sensory reading
9.          update $GNT$ with $G(x)$
10. return $GNT$ and $DTT$
```

Two main updates are executed in each iteration in the *Build_Map* algorithm. One updates the GNT per the rules described in the GNT construction section included earlier. The second updates the triangulation structure the DTT. This update is primarily done through executing the *Update_Triangulation* algorithm in each iteration which uses line sweeping on the vertices of the $VisRegion(x)$ and handles each vertex as it encounters it to incrementally build/update the DTT.

2) Incremental Triangulation Algorithm: The *Update_Triangulation* algorithm Incrementally builds a geometrical map of the environment as the robot moves toward unexplored gaps. If this is the first time this procedure is invoked, i.e. te DTT is empty, then the whole current $VisRegion(x)$ will be triangulated. However, if the DTT is not empty the triangulation starts by a preparation phase which basically find the non-primitive (temprorary) triangles adjacent to the current $VisRegion(x)$ and merge them with the adjacent portion of the $VisRegion(x)$. This step is important to avoid re-triangulating areas that are fixed. In other words, any primitive triangle in the DTT will never be a candidate for re-triangulating, thus avoiding unnecessary computation. This can significantly improve the mapping algorithms performance especially in large environments with relatively smaller connected areas that will require a larger number of triangles to represent them.

Once the area to be triangulated in this iteration is determined, the algorithm proceeds to sorting this area's vertices (its boundary or gap points) in descending order according to their $y-coordinate$. Then the sorted vertices, denoted by v_1, ..., v_n, are handled one after the other to achieve the sweep line effect by testing them for connectivity with each other through performing the test in line 9 of the algorithm. The vertices above the sweep line that are not *connectable* to vertices below the sweep line are not tested for further connection to avoid unneeded computation.

While vertices are getting connected, triangles edges are created and complete triangles are formed. when a triangle is created it will be linked to its neighbouring triangles to maintain the connectivity of the tree structure. When new triangles are formed, they will be immediately flagged as primitive or non-primitive by testing its three vertices, if any is a gap vertex, then this triangle is flagged as non-primitive, otherwise it is primitive. Upon the completion of the triangulation process, the algorithm returns the updated DTT structure to the *Build_Map* algorithm.

Figure 4 illustrates the flow of the *Update_Triangulation* algorithm described above.

Algorithm *Update_Triangulation*(DTT,VisRegion(x))
Input: The existing DTT and the $VisRegion(x)$.
Output: The updated DTT structure.

```
1.  for each $non-primitive$ triangle $t_i \in DTT$ adjacent to
    $VisRegion(x)$
2.      do merge $t_i$ with $VisRegion(x)$
3.  $VisVer$ = sorted vertices of $VisRegion(x)$ in decreasing
    order of $y-coordinate$. If two vertices have the same
    $y-coordinate$, then the left most one comes first. Let
    $v_1$, ..., $v_n$, denote the sorted sequence.
4.  initialize an empty list $L$.
5.  add $v_1$ and $v_2$ to $L$.
6.  for $j \leftarrow 3$ to $n-1$
7.      do add $v_j$ to the end of $L$
8.          for $k \leftarrow size(L)-1$ to 1
9.              do if $v_k$ $in-line-of-sight$ to $v_j$ and $v_k$
                    is not an endpoint to one of the two edges
                    connected to $v_j$ and $\overline{v_k v_j}$ does not intersect
                    edges of $DTT$
10.                 then add $\overline{v_k v_j}$ to $DTT$
11.         for $k \leftarrow 1$ to $size(L)$
12.             do if triangles($v_k$) are all complete
13.                 then remove $v_k$ from $L$
14. return $DTT$
```

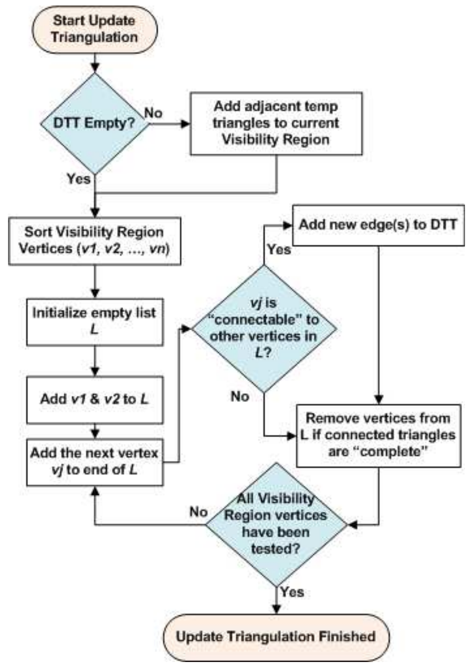
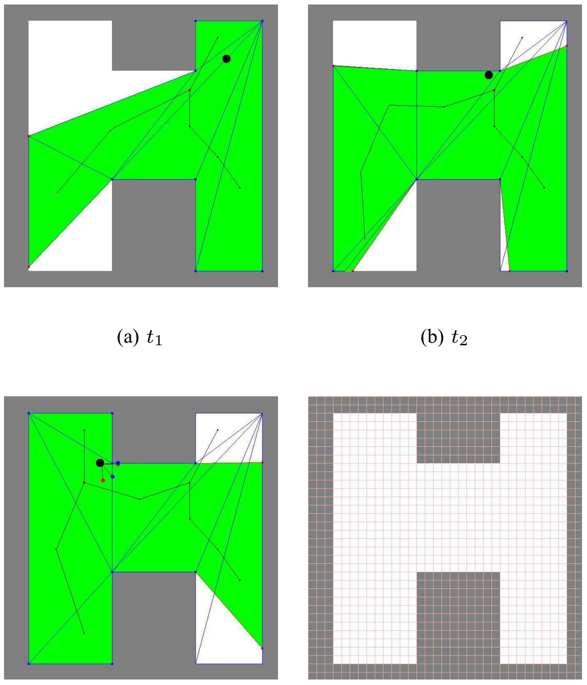

Start Update Triangulation
DTT Empty?
No
Add adjacent temp triangles to current Visibility Region
Yes
Sort Visibility Region Vertices (v1, v2, ..., vn)
Initialize empty list L
Add v1 & v2 to L
Add the next vertex vj to end of L
vj is "connectable" to other vertices in L?
Yes
Add new edge(s) to DTT
No
Remove vertices from L if connected triangles are "complete"
All Visibility Region vertices have been tested?
No
Yes
Update Triangulation Finished

Fig. 4. Update Triangulation algorithm flow chart.

Completion of the previous two algorithms map the environment topologically in the GNT and geometrically in the DTT.

IV. Experiments

This section presents the experiments used to evaluate the performance and measure the compactness of the proposed data structure, the DTT. A simulation environment developed using Java was used to perform this evaluation where both structures were implemented in the simulator. We will start by presenting a simple environment structure illustrating the progress of the mapping process at multiple instances of time and the final result will show the complete resulting DTT with connected edges to clarify its connectivity. This will be followed by a set of five more different environments with varying degrees of difficulties depicting the final DTT in each environment with a medium-resolution occupancy grid to compare the storage size of both structures. In all of our examples the robot is always directed to explore the closest unexplored gap from its current position. However, this naive gap selection method have been replaced with a more sophisticated strategy developed especially to reduce the exploration time to serve in high risk operation environment such as search and rescue. More details on this can be found in [11].


(a) t_1

(b) t_2

(c) t_3

(d) t_4

Fig. 5. The construction of the DTT for a simple environment shown at times t_1, t_2,and t_3 displayed in order, in addition to the corresponding occupancy grid.

Figure 5 (a - c) depicts the first experiments using a very basic and simple environment structure to show the progress of the mapping process at times t_1, t_2, and t_3, respectively. The process starts by initializing the GNT and DTT given the initially acquired $VisRegion$ from the collected range data given the initial robot position as shown in (a). The initial DTT consists of 9 triangles, 6 primitive and 3 non-primitive. The connectivity of the DTT structure is displayed through connecting the centroids of the existing triangles with edges presenting the parent-child relations between them. After initialization, the robot $chase$ the closest gap (since all gaps are unexplored at this stage) which is the top left gap at time t_2 causing changes in the $VisRegion$. However, the areas that were visible at time t_1 and are no longer visible are reprensented in the DTT using primitive triangles, and thus are fixed. Newly visible areas will cause changes in the DTT structure since the adjacent $non-primitive$ triangles will be merged with the areas of the $VisRegion$ that were not previously visible and then triangulated and added to the DTT. The DTT connectivity is updated as displayed in (b). At time t_3, the exploration is completed and the complete geometry of the environment is captured in the final DTT which at this instance of time contains a total of 10 primitive triangles. To show the campactness of the final DTT in (c) with a total of only 10 triangles we compare it to the medium-resultion occupancy grid displayed in (d) for the same environment containing 670 free cells.

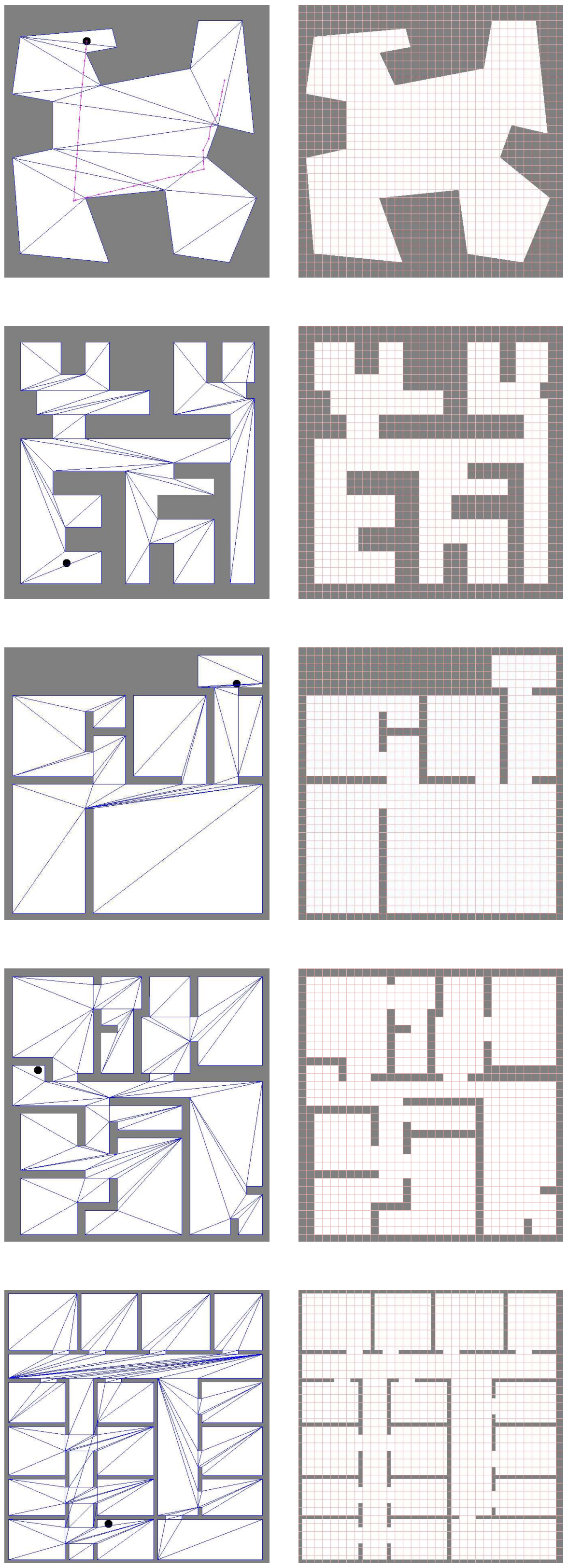


Fig. 6. The final triangulated map (left column) in comparison with occupancy grid (right column) for various environments setups, namely a random environment, maze-like structure, a simple floor plan, a more complicated floor plan, and an office environment, respectively.

Environment Type	Total Storage Size	
	DTT	Grid
Simple	10	670
Random	20	563
Maze	67	566
Simple Floor Plan	91	817
Complicated Floor Plan	115	822
Office Floor Plan	261	742

TABLE I
COMPARISON DATA OF THE DTT AND THE NUMBER OF FREE CELLS IN A MEDIUM-RESOLUTION OCCUPANCY GRID FOR THE ENVIRONMENTS IN FIGURES 5 AND 6.

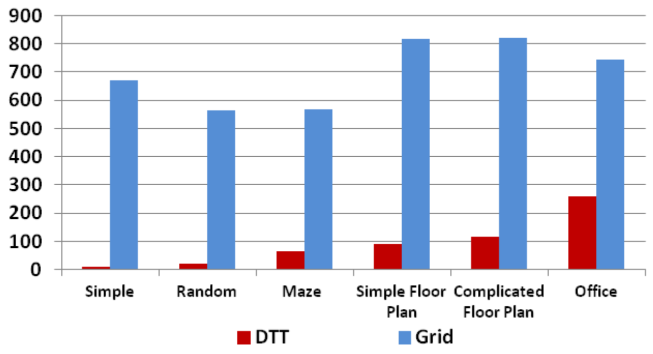


Fig. 7. Graphical representation of the comparison in Table I.

Figure 6 provides a more exhaustive comparison between DTT and occupancy grid storage size in varying complexity scenarios, namely a random environment, maze-like structure, a simple floor plan, a more complicated floor plan, and an office environment, respectively. Table I and Figure 7 summarizes all the presented environments comparison for easy evaluation.

The presented comparison proves that our presented data structure, the DTT, provides a complete mapping of the environment in a very compact representation. The DTT provides not only the geometrical mapping, but facilitate navigation and can easily generate feasible paths in the environment given its connectivity. Searching the DTT for paths is inherently fast due to its tree like structur. Moreover, path generation is not bound by the completion of the environment mapping but can be executed at any instance of time using the existing partial map.

After detailing the above experiments, we can say that our solution encapsulates a number of attractive characteristics:

1) Ease of Construction: The construction process is mainly based on the GNT construction method, chasing gaps.
2) Compactness: The map is represented using triangles saved in the DTT structure which requires a significantly smaller space than the traditional occupancy grid as shown in the analysis of the previously introduced experiments.
3) Completeness: The proposed DTT captures the connectivity of the exploration environment which facilitates path generation for navigation purposes while environment mapping is still in progress or after completion.

V. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a new triangulation-based mapping solution based on the Gap Navigation Tree. The proposed solution maps the environment topologically and geometrically and stores the geometrical map using the Dynamic Triangulation Tree structure DTT developed in this study. The DTT maps the geometry of the environment in addition to its connectivity providing a complete layer of information for environment navigation that can be used by robots and humans alike. The proposed algorithms have been tested for validity and correctness using a simulation program developed using Java. Results are presented to show the main advantages of the proposed algorithm, namely ease of construction, compactness and completeness.

The work in this paper have been extended by proposing an effective frontier selection method for effective environment exploration [11]. In this extension The main objective was to ensure the completion of exploration in the minimum time possible through making optimal frontier selection decisions based on the incrementally constructed map. Path generation for human or robots navigation in the environment, and the possibility of using the triangulated maps to locate risk sources in search and rescue missions by flagging the triangles containing them are some of the potential extensions of this work.

REFERENCES

[1] S. Thrun, “Robotic mapping: A survey,” in *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2003.
[2] A. Elfes, “Using occupancy grids for mobile robot perception and navigation,” *Computer*, p. 57, 1989.
[3] B. Yamauchi, “A frontier-based approach for autonomous exploration,” in *Computational Intelligence in Robotics and Automation, 1997. CIRA’97., Proceedings., 1997 IEEE International Symposium on*, Jul, pp. 146–151.
[4] C. Stachniss and W. Burgard, “Exploring unknown environments with mobile robots using coverage maps,” in *Proc. of the International Conference on Artificial Intelligence (IJCAI)*, 2003.
[5] J. Wettach and K. Berns, “Dynamic frontier based exploration with a mobile indoor robot,” in *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*, June, pp. 1–8.
[6] A. Marjovi, J. Nunes, L. Marques, and A. de Almeida, “Multi-robot exploration and fire searching,” in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, oct. 2009, pp. 1929 –1934.
[7] B. Tovar, R. Murrieta-Cid, and S. LaValle, “Distance-optimal navigation in an unknown environment without sensing distances,” *IEEE Transactions on Robotics*, vol. 23, no. 3, pp. 506 –518, june 2007.
[8] F. Amigoni, S. Gasparini, and M. Gini, “Good experimental methodologies for robotic mapping: A proposal,” 2007.
[9] B. Tovar and S. LaValle, “Searching and mapping among indistinguishable convex obstacles,” in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, may 2010, pp. 3569 –3574.
[10] J. O’Rourke, *Art Gallery Theorems and Algorithms*. New York: Oxford University Press, 1987.
[11] A. Aldahak, L. Seneviratne, and J. Dias, “Frontier-based exploration for unknown environments using incremental triangulation,” in *submitted to IEEE International Symposium on Safety, Security, and Rescue Robotics (under reviewing)*, 2013.