Speeding up A* Search on Visibility Graphs Defined Over Quadtrees to Enable Long Distance Path Planning for Unmanned Surface Vehicles

Brual C. Shah
Department of Mechanical Engineering
University of Maryland
College Park, MD 20742, USA
brual@umd.edu

Satyandra K. Gupta
Department of Aerospace and Mechanical Engineering
University of Southern California
Los Angeles, CA 90089, USA
skgupta@usc.edu

Abstract

We introduce an algorithm for long distance path planning in complex marine environments. The available free space in marine environments changes over time as a result of tides, environmental restrictions, and weather. As a result of these considerations, the free space region in marine environments needs to be dynamically generated and updated. The approach presented in this paper demonstrates that it is feasible to compute optimal paths using A* search on visibility graphs defined over quadtrees. Our algorithm exploits quadtree data structures for efficiently computing tangent edges in visibility graphs. We have developed an admissible heuristic that accounts for large islands while estimating the cost-to-go and provides a better lower bound than the Euclidean distance-based heuristic. During the search over visibility graphs, the branching factor of A* can be large due to the large size of the region. We introduce the idea of focusing the search by limiting the child nodes to be in certain regions of the workspace. Our results show that focusing the search significantly improves the computational efficiency without any noticeable degradation in path quality. We have also developed a method to estimate bounds on how far the computed path can be from the optimal path when methods for focusing the search are utilized for speeding up the computation.

1 Introduction

Over the last ten years, substantial progress has been made in the development of low-cost unmanned surface vehicles (USVs) (Corfield and Young 2006; Manley 2008). There are a number of civilian applications where deploying USVs can significantly reduce costs, improve safety, and increase operational efficiencies. Representative applications include remote/persistent ocean sensing, marine search and rescue, and industrial offshore supply and support. In this paper, we are interested in path planning over long distances in complex marine environments. Figure 1 shows an example of an environment that consists of hundreds of islands of complex shapes. The available free space in such marine environments changes over time as a result of tides, environmental restrictions, and weather. Low tides may make it infeasible to go through regions with shallow waters. Environmental restrictions may prevent the unmanned surface vehicle from passing through certain protected marine regions for certain periods of times. Weather induced waves may prohibit traveling over certain areas due to high collision risks. As a result of these considerations, the free space region in marine environments needs to be dynamically generated and updated.

Figure 1: Topography of a complex marine environment.

Consider a representative marine region of 100 sq. km. This region may need thousands of complex polygons to represent the land areas (or obstacles) in the marine environment. Roadmap-based methods work well with polygons-based representations (Siegwart, Nourbakhsh, and Scaramuzza 2011) and have been shown to be quite useful in ground applications. However, as mentioned earlier, the free space may change in marine environments. Hence, we cannot justify the computational time and efforts needed to build roadmaps. Instead, we will need to import the obstacle field data from NOAA nautical charts by applying the appropriate height filters based on the tide conditions at the time of the mission. Additional obstacles will need to be identified based on weather and environmental restrictions applicable

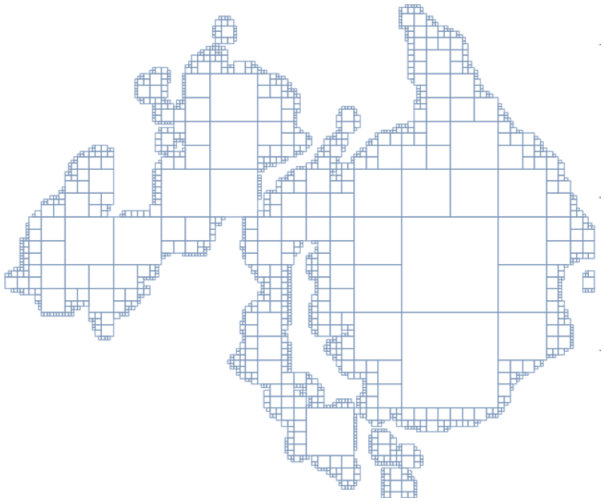

Figure 2: Quadtree representation of a complex polygon.

at the time of the mission. These requirements restrict us to only consider those methods that can compute plans without the need for computing roadmaps.

Grid-based methods can be used to represent complex obstacle fields (Yap 2002; Hart, Nilsson, and Raphael 1968). If a grid is defined over a 100 km by 100 km region using a 10 m resolution in each dimension, the resulting grid will contain 100 million nodes. The large regions typically contain large land masses that may span several kilometers. It appears that quadtrees (Yahja et al. 1998) are better spatial data structures to represent complex marine environments compared to grids (see Figure 2). A 100km by 100km region can be represented with a hundred thousand or fewer nodes in a quadtree. This leads to a reduction of more than a thousand nodes in terms of spatial complexity. Environmental and weather based restricted areas can be easily incorporated in quadtrees as obstacles. Hence, in this paper we will use quadtrees as the spatial data structure to represent the free space.

In this paper, we present an algorithm for long distance path planning in complex marine environments using A* search on visibility graphs defined over quad trees. Section 3 defines the visibility graphs. Visibility graphs have been shown to be useful in computing optimal paths in the presence of complex obstacle fields. However, the computational performance of visibility graphs degrades as the region to be searched becomes large and the number of nodes in the visibility graph increases rapidly. This paper introduces a number of techniques to speed up the A* search process and makes it feasible to compute paths using visibility graphs over large regions. Previous work has shown that the optimal path goes through tangent edges in the visibility graph (Liu and Arimoto 1992). We exploit the data structures and the modified rotational plane sweep algorithm (RPS) (Choset 2005) to compute tangent edges efficiently (see Section 4).

We have presented an improved heuristic to handle large obstacles in the region (see Section 5). Finally, we have described methods for focusing the search by looking for child nodes in certain spatial regions (see Section 6). The path computed using the proposed approach can be used to generate trajectories and support a wide variety of missions (Shah et al. 2015; Švec et al. 2014; Bertaska et al. 2013).

2 Related Work

Path planning is a well-studied problem in robotics and AI communities. Many different approaches have been developed to solve path planning problems (LaValle 2006; Hoy, Matveev, and Savkin 2015). The body of work that is most closely related to the theme of this paper is the path planning problem for a given complete map. We will review methods that deal with known stationary obstacles with no uncertainty in the environment or the outcome of the vehicle actions. Readers are referred to (Stentz 1994; Wang and Julier 2011; Luna et al. 2014) for an overview of planning methods in partially known maps. Methods for planning under uncertainty are discussed in (Dadkhah and Mettler 2012). Planning methods developed for dealing with dynamic obstacles are given in (Van Den Berg et al. 2011; Sezer and Gokasan 2012; Wu and Feng 2012; Kuwata et al. 2014).

Path planning problems over long distances can be divided into two categories. The first category includes problems where configuration spaces associated with the collision free regions of the space can be easily computed explicitly. Path planning problems for unmanned surface vehicles are primarily focused in 2D workspaces (i.e., 3D configuration spaces) and belong to the first category. The second category belongs to the class of problems where explicitly computing configuration space is computationally challenging. Sampling based methods such as Rapidly Exploring Random Trees (RRT) (LaValle and Kuffner Jr 2000) and Probabilistic Road Maps (PRM) (Kavraki et al. 1996) have been successfully used to deal with such problems. In this paper, we will focus on methods that use explicitly computed configuration spaces.

Finding optimal paths requires abstracting the given configuration space into a discrete graph over which a search can be performed to compute the optimal path. If the application requires solving the planning problem multiple times over the same configuration space, then it is useful to construct roadmaps (Siegwart, Nourbakhsh, and Scaramuzza 2011), and Voronoi graphs (Aurenhammer 1991; Bhattacharya and Gavrilova 2008) that associate with the configuration space. Even though this takes significant computational effort upfront, the roadmap and/or Voronoi graphs can be reused over multiple planning instances. If the planning problem is not being solved multiple times, then it is computationally preferable to construct the relevant portions of the search graph on-the-fly. In this paper, we are interested in methods that do not precompute the search graph.

There are three main methods for representing the path planning problem as a graph search. The first class of methods represents the configuration space as uniform grids (Yap 2002; Hart, Nilsson, and Raphael 1968; Koenig and Likhachev 2002; Likhachev et al. 2005) or multi-resolution grids (Yap et al. 2011). At any point in the grid, the ve-

hicle can move only to the adjacent grid points using a fixed number of actions. This method limits branching in the search trees, but leads to a large number of nodes in the search tree. Paths produced by these methods are not smooth and are often not optimal. The second class of methods is based on the idea that the optimal path will move the vehicle in straight line paths between obstacles and it will only pass through visible vertices of the obstacles. The underlying representation used during the search is called a visibility graph (Lee 1978; Lozano-Pérez and Wesley 1979; Rashid et al. 2013). These methods compute optimal paths and significantly reduce the number of nodes in the search graph. However, the branching factor can be high. This leads to computationally slow performance when the spatial region over which the planning is being done is large. Recent development in any-angle search represents a third class of methods. These methods combine features from the above two classes of methods and limit the branching at the search node and yet do not constrain the vehicle to move along the grid edges. These methods are fast and produce significantly better paths than grid-based methods. However, paths produced by these methods may not be optimal. Notable methods belonging to this class are Incremental Phi* (Nash, Koenig, and Likhachev 2009), Theta* (Daniel et al. 2010) and its variants (Nash, Koenig, and Tovey 2010; Uras and Koenig 2015b; Tovey, Koenig, and Nash 2015), and Field D* (Ferguson and Stentz 2006).

Several researchers have used quadtrees as the underlying representation for path planning (Yahja et al. 1998). Recent work using quadtrees to represent the operating environment of the robot includes (Petres et al. 2007; Zhang, Ma, and Liu 2012).

3 Approach

In this paper, we are interested in computing an optimal path τ on a workspace with large maps represented using a quadtree $\mathcal{M}_Q$ (see Figure 12), given the start $\mathbf{n_I}$ and goal $\mathbf{n_G}$ node. Each leaf node in a quadtree is represented as $\mathbf{l} = [\eta^T, d, t] \in \mathcal{M}_Q$, where $\eta = [x, y]^T$ is the center of the node in 2D space, $d \in [0, d_{max}]$ is the current depth of the node ranging from 0 (i.e. the rootnode) up to the maximum depth of the quadtree d_{max}, and t denotes the type of the node, i.e. a free node ($t = 0$), a solid or obstacle node ($t = 1$), and a node with additional branches ($t = 2$). In this paper, the word 'node' (or 'nodes') is used for the nodes of the visibility graph (Lozano-Pérez and Wesley 1979). The quadtree nodes will be referred to as 'leaf nodes' (see Figure 3).

Any shortest path between the start $\mathbf{n_I}$ and the goal $\mathbf{n_G}$ node in the workspace with a set of polygonal obstacles $\mathcal{O}$ is a poly-line path whose inner vertices are vertices of $\mathcal{O}$ (Choset 2005). Two vertices $\mathbf{v}$ and $\mathbf{v}'$ are mutually visible if the line segment connecting $\mathbf{v}$ and $\mathbf{v}'$ does not intersect with the interior of the polygonal obstacle $o_i \in \mathcal{O}$, where o_i is the i^{th} obstacle in the set $\mathcal{O}$. Now, vertices $\mathbf{v}$ and $\mathbf{v}'$ will be nodes $\mathbf{n}$ and $\mathbf{n}'$ in the visibility graph $\mathcal{V}$ with an edge between them. The Visibility graph $\mathcal{V}$ is a graph whose nodes are vertices of polygonal obstacles along with the initial $\mathbf{n_I}$ and the goal $\mathbf{n_G}$ nodes. The edges of the graph represent the

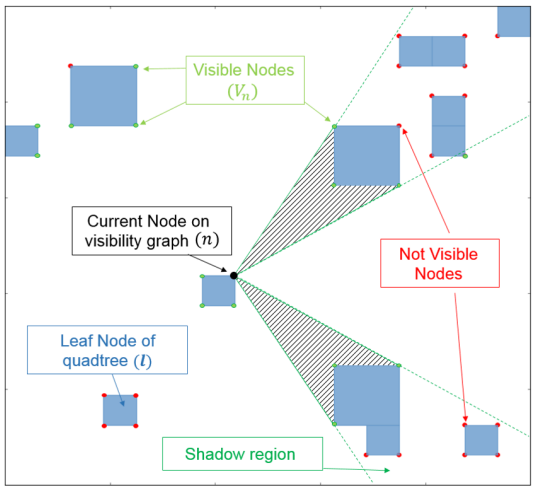

Figure 3: Computation of visible nodes in quadtree.

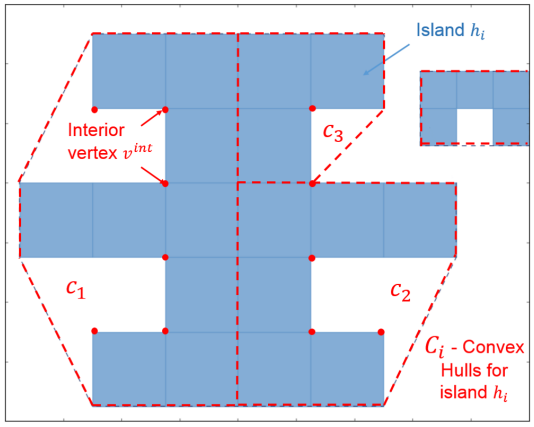

Figure 4: Eliminating visible interior vertices from the visibility graph.

pair of mutually visible vertices. The shortest path between the start $\mathbf{n_I}$ and goal $\mathbf{n_G}$ nodes is the shortest path in the visibility graph $\mathcal{V}$. In our problem, the polygonal obstacles are the solid leaf nodes ($t = 1$) of the quadtree $\mathcal{M}_Q$ and the vertices of these solid leaf nodes are the nodes of the visibility graph $\mathcal{V}$. The set of visible nodes at node $\mathbf{n}$ is denoted as $\mathbf{v^n}$. The example in Figure 3 shows the candidate visible nodes (marked by green) and non-visible nodes (marked by red) of the current node $\mathbf{n}$.

The set of visible nodes $\mathbf{v^n}$ is calculated by iterating over all the N nodes in the visibility graph $\mathcal{V}$ and performing $N - 1$ collision checks to determine the visibility of the nodes. This computation proves to be computationally expensive. The complexity of the visibility graph (i.e. the number of visibility checks per node $\mathbf{n}$) can be reduced by incorporating the concept of tangent graphs (i.e. reduced visibility graph) (Liu and Arimoto 1992). To reduce the complexity of the graph we need to eliminate the nodes that will never be part of the optimal path τ_{opt}. First, we accumulate the connected solid leaf nodes of the quadtree tree which are termed

island $\mathbf{h}_i$. Each map represented by quadtree $\mathcal{M}_Q$ may have several such islands denoted by the set H. Second, we compute the convex hull regions $\mathbf{c}_j \in C_i$ for all the island regions, where $C_i \in \mathcal{C}$ is the set of all the convex hulls for the i^{th} island in H. The convex hull $\mathbf{c}_j$ is computed such that the hull does not intersect with obstacle region $\mathcal{O}$. Thus, each island $\mathbf{h}_i$ can be represented by multiple convex hulls.

Let us consider a simple case shown in Figure 4, where (1) $\mathbf{h}_i \in H$ is an island (i.e. the solid quadtree leaf node). Let, C_i be the set of convex hulls of island $\mathbf{h}_i$, (2) $\tau_{feasible}$ is a feasible path that includes an interior visible node $\mathbf{n}^{int}$ (i.e. interior vertex) of the island $\mathbf{h}_i$ that is not on the boundary of $\mathbf{c}_j \in C_i$, and (3) the start $\mathbf{n_I}$ and the goal $\mathbf{n_G}$ node are outside of both the island $\mathbf{h}_i$ and all the convex hulls in $\mathbf{c}_j \in C_i$ and no other obstacle in $\mathcal{O}$ intersects with $\mathbf{c}_j \in C_i$. In this case, $\tau_{feasible}$ will cross the convex hull twice on its course to reach interior visible node $\mathbf{n}^{int}$ and then to come out of $\mathbf{c}_j$. This path $\tau_{feasible}$ can always be improved by just moving along the convex hull rather than going inside and coming out. There is no other obstacle intersecting with $\mathbf{c}_j$ to prevent this. Therefore, the interior vertex $\mathbf{n}^{int}$ will never be in the optimal path τ_{opt}.

Now, let us consider a more general case as depicted in Figure 5 where (1) $C_i = \{\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3\}$, (2) $\mathbf{c}_j \in C_i$ does not intersect with any other obstacle in $\mathcal{O}$, and (3) $\mathbf{n}^{int}$ is an interior node that belongs to island $\mathbf{h}_i$ but not to $\mathbf{c}_j$. The optimal path τ_{opt} will not pass through $\mathbf{n}^{int}$ if $\mathbf{n_I}$ and $\mathbf{n_G}$ are outside of the convex hulls in C_i. Hence $\mathbf{n}^{int}$ can be removed from the list of potential candidate nodes for visibility graph $\mathcal{V}$. Similarly, in Figure 5, the nodes of the visibility graph marked with red color are interior nodes $\mathbf{n}^{int}$ that lie in the interior of the convex hulls $\mathcal{C}$. Now, as per the theorem stated above, these nodes marked with red color can be eliminated because they will not be included in the optimal path τ_{opt}. Also, the nodes that are shared by the convex hulls within the same island can be eliminated. Each island will have a set of candidate nodes after the elimination of the interior nodes $\mathbf{n}^{int}$ that are used for computing tangent edges.

The interior nodes $\mathbf{n}^{int}$ lying in the same convex hull as the initial node $\mathbf{n_I}$ or the goal node $\mathbf{n_G}$ are not eliminated, thus not altering the optimality of the path. It has been shown that the optimal path only passes through edges that are tangent to the polygonal obstacles in the visibility graph (Liu and Arimoto 1992). Let us denote the graph comprised of only tangent edges (i.e. tangent graph or reduced visibility graph) by $\mathcal{V}_t$. The shortest distance path between $\mathbf{n_I}$ and $\mathbf{n_G}$ is comprised of a combination of shortest straight line paths between two polygons and line segments along the exterior boundary of the polygonal obstacles. Now, the number of visible edges of the current node $\mathbf{n} \in \mathcal{V}_t$ can be restricted to the tangent edges from the current node $\mathbf{n}$. This reduces the size of the visibility graph without altering the quality and optimality of the path.

4 Computation of Edges on the Tangent Graph

Traditionally, the visibility and tangent graph-based path planning approaches precompute the entire structure of the graph with edges connecting the visible nodes. The computation of all the edges of the tangent graph $\mathcal{V}_t$ is computationally expensive. In our approach, the edges of the graph are computed on the fly during the search for the optimal path τ. The search is performed by expanding nodes in the least-cost A* (Hart, Nilsson, and Raphael 1968) fashion according to the cost function $f(\mathbf{n}) = g(\mathbf{n}) + h(\mathbf{n})$, where $g(\mathbf{n})$ is the cost-to-come to node $\mathbf{n}$ from the initial node $\mathbf{n_I}$, and $h(\mathbf{n})$ is the heuristic estimate of the cost-to-go from the current node $\mathbf{n}$ to the goal node $\mathbf{n_G}$.

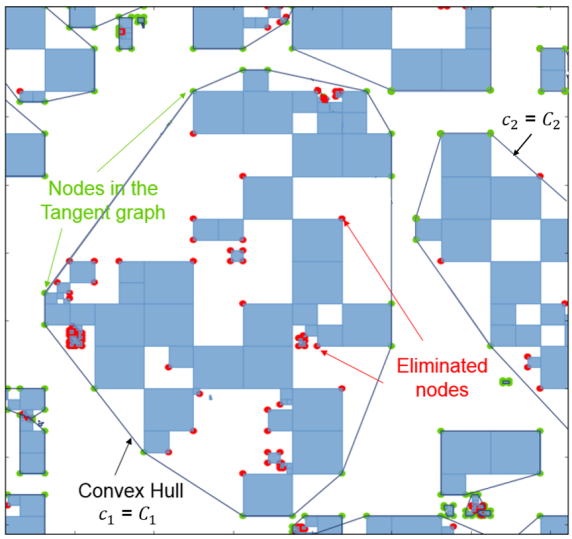

Figure 5: Eliminating nodes in the interior of the convex hull (we assume the start and the goal node are not inside any convex hull).

During the expansion of each node $\mathbf{n} \in \mathcal{V}_t$, we need to determine tangents for each island $\mathbf{h}_i \in H$ and these tangents have to be checked for collisions in order to determine the visible nodes $\mathbf{v_t^n}$. This process of determining the visible nodes $\mathbf{v_t^n}$ is computationally intensive and hence reduces the performance of the search algorithm.

The computational performance of the search can be improved by reducing the number of collision checks required during the determination of visible nodes. The reduction in collision checks is achieved by our implementation of a modified variant of the rotational plane sweep algorithm (RPS) (Choset 2005), in which we compute the angle and distance to each node in $\mathcal{V}_t$ from the current node $\mathbf{n_c}$. During the computation of these angles and distances, we determine maximum and minimum angle for each island $\mathbf{h}_i$ which serve as tangents from the current node. The list of all the tangents are sorted based on their angles with respect to the current node $\mathbf{n_c}$.

Now, these tangents to each island $\mathbf{h}_i \in H$ form cone like structures with the apex at the current node $\mathbf{n}$. We refer to these cones as visibility cones for the island $\mathbf{h}_i$. During the visibility check, we can directly eliminate all the candidate nodes with an angle lying in between the two edges of the cone and distances greater than the tangent nodes for the nearest visible convex hull $\mathbf{h}_i$. This drastically reduces the

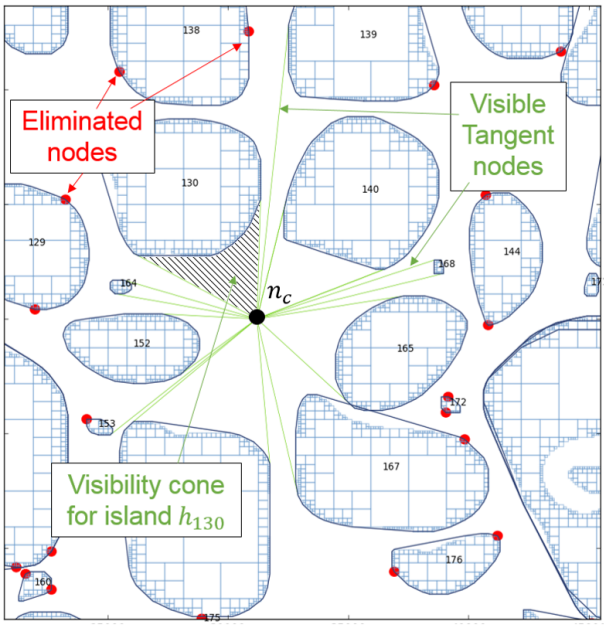

Figure 6: Elimination of non-visible nodes using the computed tangents for the islands in H.

number of collision checks required during the determination of visible tangent nodes and improves the efficiency of the algorithm. For example, in Figure 6 the tangent nodes of island 138 and one of the tangent nodes of island 129 can be directly eliminated by a visibility cone of island 130. Finally, the collision checks for determining visible tangent edges that cannot be eliminated by the modified RPS algorithm are performed by a modified Bresenham's collision test algorithm described in (Choi, Lee, and Yu 2010).

5 A New Heuristic

The performance of the A* based path planning algorithm depends on the estimation of the cost-to-go (or h-cost) from the current state $\mathbf{n_I}$ to the goal state $\mathbf{n_G}$. If the path planner significantly underestimates the cost-to-go, then it has to expand more nodes until it finds an optimal path to the goal. On the other hand, if the path planner overestimates the cost-to-go, then the h-cost is inadmissible and the paths are no longer optimal.

The most widely used heuristic by the path planning algorithms (Švec et al. 2013; 2014; Shah et al. 2014; Koenig and Likhachev 2002; Likhachev et al. 2005) is the Euclidean distance from the current node $\mathbf{n_I}$ to the goal node $\mathbf{n_G}$. This heuristic expands nodes nearest to the goal in a greedy fashion. This assumption works perfectly on maps without any large obstacle regions. In scenarios with large obstacles, the shortest optimal path has to circumvent at least one obstacle before heading towards the goal, unless the goal is in line-of-sight to the initial location. The heuristic based on Euclidean distance will expand all the nodes lying on the perimeter of the obstacle in a greedy fashion until a straight line path is available to the goal. However, in most of the complex marine environments (see Figure 1) we have large islands and the use of the heuristic based on Euclidean distance often degrades the performance of the path planner.

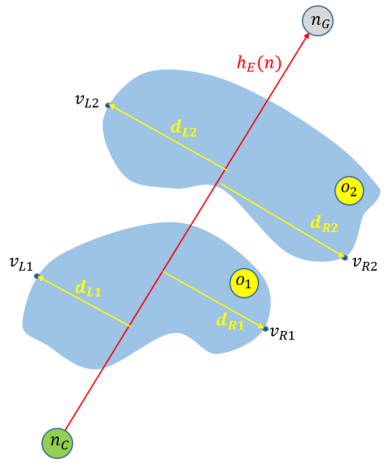

Figure 7: Computation of the heuristic.

We have developed a heuristic that exploits the fact that the optimal path reaching towards the goal has to pass through one of the corners of the obstacle obstructing the straight line path to the goal node. Let us consider a scenario shown in Figure 7. In this example, the straight line path from the current node $\mathbf{n_c}$ to the goal node $\mathbf{n_G}$ intersects two obstacles o_1 and o_2. The Euclidean distance heuristic from $\mathbf{n}$ to $\mathbf{n_G}$ is given by $h_E(\mathbf{n}) = d(\mathbf{n}, \mathbf{n_G})$, where $d(\mathbf{n}, \mathbf{n_G})$ is the straight line distance. The vertices $\mathbf{v}_{R1}$ and $\mathbf{v}_{R2}$ are the rightmost vertices of obstacles o_1 and o_2 and their corresponding orthogonal distances are denoted by d_{R1} and d_{R2}. Similarly, the leftmost vertices are $\mathbf{v}_{L1}$ and $\mathbf{v}_{L2}$ and their corresponding orthogonal distances are denoted by d_{L1} and d_{L2}. The shortest route to the goal will have to pass through at least one of the extreme vertices of obstacle o_1 or o_2 represented in the configuration space, i.e. the path will be triangular with the middle vertex $\mathbf{v}_{Xn} \in \{\mathbf{v}_{R1}, \mathbf{v}_{R2}, \mathbf{v}_{L1}, \mathbf{v}_{L2}\}$, where $X \in \{R, L\}$ and $n \in \{1, 2\}$. Let the extreme vertex corresponding to distance d_{Xn} be denoted by $\mathbf{v}_{Xn}$.

The triangular path length to travel from the right side of the obstacle is given by $h_T^R(\mathbf{n}) = d(\mathbf{n}, \mathbf{v}(max(d_{R1}, d_{R2}))) + d(\mathbf{v}(max(d_{R1}, d_{R2})), \mathbf{n_G})$. Similarly, the path length to travel from the left side is denoted by $h_T^L(\mathbf{n})$. Finally, the admissible triangular heuristic cost is computed as $h_T(\mathbf{n}) = min(h_T^L(\mathbf{n}), h_T^R(\mathbf{n}))$.

6 Focusing A* Search

In order to guarantee optimality, the A* algorithm needs to explore all possible child nodes for a node being expanded. In large spatial regions, there can be a large number of nodes that need to be examined to determine if the straight line between them and the node being expanded belong to the tangent graph. In certain pathological cases, the number of edges on the tangent graph can be very large (see Figure 8). This can lead to poor computational performance during the search.

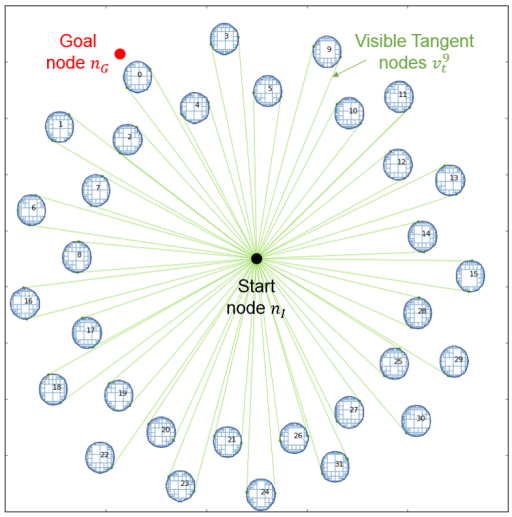

Figure 8: Pathological scenario where a node in tangent graph has a large branching factor.

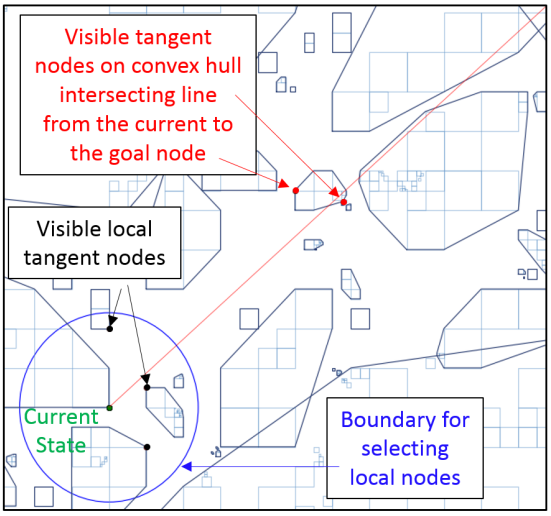

Figure 9: Procedure to add nodes to the focused visibility graph.

In order to improve the computational performance of the algorithm, we can focus the search and examine only certain kinds of edges on the tangent graph. For example, we can search for the edges in a spatial region that lies within a certain radius of the current node. However, if a fixed radius is used, then we may not be able to find any edge to explore if all other obstacles lie outside of the given radius. Therefore, in addition to the radius, we also consider adding the edges that lie on obstacles that intersect with the straight line path from the node being expanded and the goal (see Figure 9). This approach focuses the search and ensures that we do not encounter pathological cases. Also, during the search the child nodes of the current node $\mathbf{n_c}$ are checked for direct line-of-sight connection with the parent node. This enables a line-of-sight connection between the current node and the nodes that lie outside the constrained region.

The path generated by the focused search is not necessarily optimal. Hence we are interested in characterizing the path with respect to the optimal path. Let us assume that L is the length of the path generated using the focused search. Now, let us construct a circle of radius L at the start node $\mathbf{n_I}$ and identify the set of visible nodes $\mathbf{v^n}_{t,L} \in \mathcal{V}_t$ that lie inside the circle of radius L. Any node on the reduced visibility graph (i.e. tangent graph) that is outside of the radius L will have a path length of more than L from $\mathbf{n_I}$, hence it cannot be on the optimal path.

Now we will compute the sum of cost-to-come $g(\mathbf{n})$ and cost-to-go $h(\mathbf{n})$ for all the visible node $\mathbf{n} \in \mathbf{v^n}_{t,L}$. Let L' be the minimum among all the visible nodes in $\mathbf{v^n}_{t,L}$. The optimal path length cannot exceed L' because the optimal path has to go through nodes in $\mathbf{v^n}_{t,L}$. If $L \geq L'$, then L is the optimal solution. If this condition is not satisfied, then L' can be used to compute a bound on how far off L is from the optimal solution. The optimal solution cannot improve the path length given by L by more than $100(L - L')/L$ percent. If this bound is relatively small, then the search can be terminated and L can be returned as the solution.

If L' is much smaller than L, then a second round of the search can be conducted with an adaptive radius of focus. At the start node we can use L as the radius of focus. As the search progresses, this radius can be reduced to L minus the cost-to-come for the node being expanded. Let $\mathbf{n_c}$ be the current node being expanded and $g(\mathbf{n_c})$ be its cost-to-come, then the remaining cost of the optimal path cannot exceed $L - g(\mathbf{n_c})$, therefore there is no need to look for successor nodes that are more than $L - g(\mathbf{n_c})$ distance away from $\mathbf{n_c}$. This search always produces the optimal answer.

7 Results and Discussion

We compare the performance of the developed algorithm with the any-angle path planning algorithm Theta* (Daniel et al. 2010). The implementation of Theta* used for all the simulation experiments is developed by the authors of Theta* and is taken from (Uras and Koenig 2015a). The scenario shown in Figure 11 (quadtree representation) is used to demonstrate the scaling of the developed tangent graph approach with improved heuristics against Theta*. Table 1 provides the computation time of Theta* and our approach. We can see that the computation time of Theta* drastically increases with the increase in pixels (or minimum grid size) used to represent the scene. On the other hand, the computation time of our approach TG+HEU, marginally increases primarily because of the Bresenham's collision test algorithm (Choi, Lee, and Yu 2010). In other words, the developed tangent graph approach is resolution independent and does not depend on the grid size of the scene.

The simulation setup consisted of a randomly generated quadtree for the area of size 100 x 100 km (see Figure 10). The maximum depth of the quadtree was kept at $d_{max} = 13$ i.e. the finest resolution will be $100000/2^{13} = 12.21$ meters. The start and the goal nodes were kept constant at $\mathbf{n_I} = [2000, 2000]^T$ and $\mathbf{n_G} = [98000, 98000]^T$ (in me-

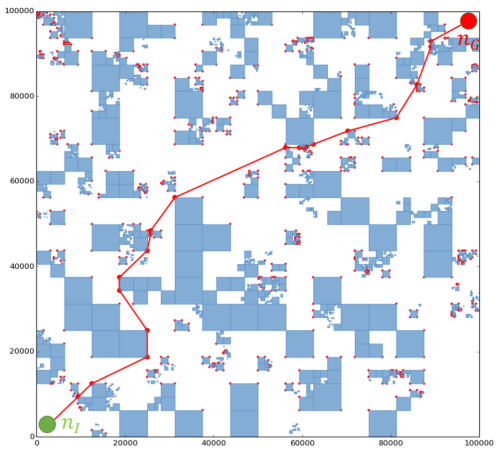

Figure 10: Experimental setup and sample any-angle path from the start node $\mathbf{n_I}$ to the goal node $\mathbf{n_G}$.

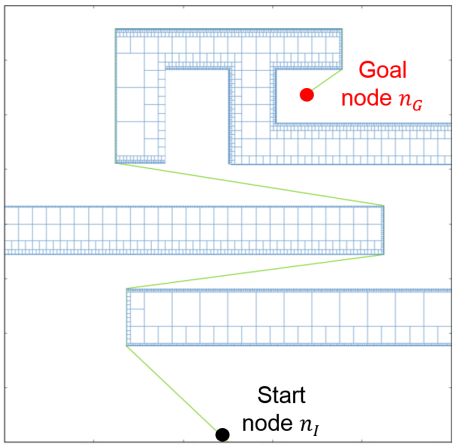

Figure 11: Example scenario to compare the scaling between Theta* and our approach.

Table 1: Computational results for Theta* and tangent graph with the developed new heuristic (see Section 5) (TG+HEU) in the same scenario with different grid sizes.

Map Size (pixels)	Computation Time (in sec)	
	Theta*	TG+HEU
1024 x 1024	1.87	0.19
2048 x 2048	16.69	0.22
4096 x 4096	90.784	0.24

Table 2: Comparison between different variants of developed visibility graphs-based algorithms on scenarios with a varying number of quadtree nodes. VG+ECU: Visibility graph with Euclidean distance as heuristic, TG+HEU: Tangent graph with the developed new heuristic (see Section 5), and FS+HEU: Focused search in tangent graph with the developed heuristic.

# of Quad Nodes	# of Nodes in RVG	Computation Time (in sec)			% reduction in computation time by FS + HEU over VG + ECU	% increase in path length by FS + HEU
		ECU+ VG	TG + HEU	FS + HEU		
6446	919	21.60	10.17	4.92	77.21	0.00
6578	1272	19.79	10.98	3.11	84.30	0.00
7622	1044	16.58	5.78	2.98	82.05	0.00
8689	823	14.96	5.87	2.10	85.97	0.00
20605	1865	121.21	54.81	18.54	84.71	0.00
21075	2678	171.10	79.76	14.86	91.31	0.00
24951	3255	55.80	19.77	1.87	96.64	0.00
23567	1832	131.85	79.10	16.60	87.41	0.00
50534	9553	771.84	416.21	128.37	83.37	0.24
55180	9785	902.62	573.32	242.18	73.17	0.13
54961	11574	686.67	491.36	247.51	63.96	0.10
45082	6652	243.30	123.07	52.97	78.23	0.00
75578	22571	2440.61	1191.60	547.11	77.58	0.06
76507	19837	1209.50	948.21	476.37	60.61	0.00
75017	20427	856.65	475.82	182.92	78.65	0.33
74375	19340	1005.09	562.49	281.43	72.00	0.05
96480	14924	2638.41	1378.00	497.72	81.14	0.01
100657	16144	1339.26	1123.13	551.64	58.81	0.31
96630	14422	1106.41	852.66	368.99	66.65	0.00
101056	17055	1555.00	766.36	377.17	75.74	0.01

ters) respectively. The algorithm is written in Python 2.7 and computed on a Intel(R) Core(IM) i7-2600 CPU @ 3.4 GHz machine with 8GB RAM.

The results presented in Table 2 shows the computational performance of the developed approaches in randomly generated quadtree maps. The size of the quadtree map is varied from 5000 to 100000 leaf nodes by varying the depth of the quadtree $d_{max} = 10$ to 13 and the occupancy of the map. The tangent graph approach combined with the improved heuristics (TG+HEU) enhances the computational performance and reduces the number of expanded states as compared to the visibility graph using the Euclidean distance as heuristics (VG+ECU). This is primarily due to a low branching factor of (TG) as it just examines the tangents of the islands while adding visible edges to the graph. The branching factor is further reduced by focusing the A* search on a tangent graph (FS) which restricts the search for the possible visible edges in the local vicinity and in the line-of-sight to the goal. In our experiments, the local vicinity of FS is determined by a circle of constant radius $r_{foc} = 10$ km. However, the improved computational performance of FS comes at the cost of loss of optimality and increased path length by a maximum of 0.33%.

The computation time and path lengths for Theta* on randomly generated scenarios with nodes fewer than 10000 quadtree nodes are comparable to that of FS+HEU. However, Theta* is unable to compute paths in several scenarios having more than 10000 quadtree leaf nodes (i.e. quadtrees

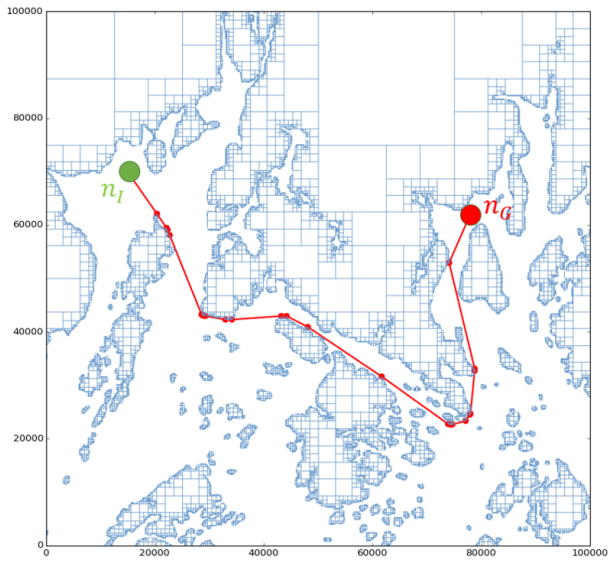

Figure 12: Computed path on a real world scenario.

of depth $d_{max} \geq 12$ which give maps that are 4096 x 4096 pixels in size) because the planner becomes memory intensive and the computer which we used to perform our simulation experiments cannot handle it. In the scenario above 10000 nodes where the Theta* is able to generate paths, we see a significant increase in computation time and path length.

The quadtree of the real marine environment (see Figure 12) is computed using the nautical chart data available from NOAA. Nautical chart data of a 100 x 100 km region is exported into shapefile (.shp) format. We have created a framework where we can read shapefiles of a region and generate the corresponding quadtree of a desired maximum depth (d_{max}). Using this framework we processed the land-regions from the shapefiles to extract the data represented in the form of polygons. The quadtree of depth $d_{max} = 13$ is computed from the extracted polygons and outputted to a text file.

The extracted polygons are then processed to compute the quadtree of depth $d_{max} = 13$ and output it to a text file.

Figure 12 shows the computed path (in red) from the start node $\mathbf{n_I}$ to the goal node $\mathbf{n_G}$ in a real marine environment shown in Figure 1 . The size of the map is 100 x 100 km and the number of quadtree nodes is 66425. The total number of candidate nodes in the tangent graph is 2732. The number of nodes expanded by the path planner FS+HEU is 711 and the computation time is 12.19 seconds. The computation time for Theta* is 96.32 sec and the computed path is the same as that computed by FS+HEU.

8 Conclusion and Future Work

This paper presents an approach for computing paths on large marine domains. The approach presented in the paper demonstrated that it is feasible to compute optimal paths using an A* search on visibility graphs defined over quadtrees. Experimental results indicate that optimal paths can be computed in a reasonable amount of time over a 100 km by 100 km area with a 10 m feature resolution. This was made feasible by developing methods to efficiently compute tangent edges in visibility graphs using quadtree data structure. There can be cases where the branching factor is large during the search over the visibility graph due to the large size of the region. To deal with these cases, we introduced the idea of focusing the search by limiting the child nodes to be in certain regions of the workspace. Our results show that this idea speeds up the computation time significantly without compromising the quality of the path in a significant way. We also developed a method to estimate bounds on how far the computed path can be from the optimal path when methods for focusing the search are utilized for speeding up the computation. Future work will involve dealing with time varying obstacles and the effect of currents on the cost function.

9 Acknowledgment

This work was supported by the National Science Foundation Grant #1526487. Opinions expressed are those of the authors and do not necessarily reflect opinions of the sponsor.

References

Aurenhammer, F. 1991. Voronoi diagramsa survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)* 23(3):345–405.

Bertaska, I.; Alvarez, J.; Armando, S.; von Ellenrieder, K. D.; Dhanak, M.; Shah, B.; Švec, P.; and Gupta, S. K. 2013. Experimental evaluation of approach behavior for autonomous surface vehicles. In *ASME Dynamic Systems and Control Conference (DSCC'13)*.

Bhattacharya, P., and Gavrilova, M. L. 2008. Roadmap-based path planning-using the voronoi diagram for a clearance-based shortest path. *Robotics & Automation Magazine, IEEE* 15(2):58–66.

Choi, S.; Lee, J.-Y.; and Yu, W. 2010. Fast any-angle path planning on grid maps with non-collision pruning. In *Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on*, 1051–1056. IEEE.

Choset, H. M. 2005. *Principles of robot motion: theory, algorithms, and implementation*. MIT press.

Corfield, S., and Young, J. 2006. Unmanned surface vehicles-game changing technology for naval operations. *IEEE Control Enginerring Series* 69:311.

Dadkhah, N., and Mettler, B. 2012. Survey of motion planning literature in the presence of uncertainty: considerations for uav guidance. *Journal of Intelligent & Robotic Systems* 65(1-4):233–246.

Daniel, K.; Nash, A.; Koenig, S.; and Felner, A. 2010. Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research* 533–579.

Ferguson, D., and Stentz, A. 2006. Using interpolation to improve path planning: The field d* algorithm. *Journal of Field Robotics* 23(2):79–101.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on* 4(2):100–107.

Hoy, M.; Matveev, A. S.; and Savkin, A. V. 2015. Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey. *Robotica* 33(03):463–497.

Kavraki, L. E.; Švestka, P.; Latombe, J.-C.; and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on* 12(4):566–580.

Koenig, S., and Likhachev, M. 2002. D* lite. In *AAAI/IAAI*, 476–483.

Kuwata, Y.; Wolf, M.; Zarzhitsky, D.; and Huntsberger, T. 2014. Safe maritime autonomous navigation with COLREGs, using velocity obstacles. *Oceanic Engineering, IEEE Journal of* 39(1):110–119.

LaValle, S. M., and Kuffner Jr, J. J. 2000. Rapidly-exploring random trees: Progress and prospects.

LaValle, S. M. 2006. *Planning algorithms.* Cambridge, U.K.: Cambridge University Press. Available at http://planning.cs.uiuc.edu.

Lee, D.-T. 1978. Proximity and reachability in the plane. Technical report, DTIC Document.

Likhachev, M.; Ferguson, D. I.; Gordon, G. J.; Stentz, A.; and Thrun, S. 2005. Anytime dynamic a*: An anytime, replanning algorithm. In *ICAPS*, 262–271.

Liu, Y.-H., and Arimoto, S. 1992. Path planning using a tangent graph for mobile robots among polygonal and curved obstacles communication. *The International Journal of Robotics Research* 11(4):376–382.

Lozano-Pérez, T., and Wesley, M. A. 1979. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM* 22(10):560–570.

Luna, R.; Lahijanian, M.; Moll, M.; and Kavraki, L. E. 2014. Optimal and efficient stochastic motion planning in partially-known environments. In *AAAI Conf. on Artificial Intelligence*.

Manley, J. E. 2008. Unmanned surface vehicles, 15 years of development. In *OCEANS 2008*, 1–4. IEEE.

Nash, A.; Koenig, S.; and Likhachev, M. 2009. Incremental phi*: Incremental any-angle path planning on grids.

Nash, A.; Koenig, S.; and Tovey, C. 2010. Lazy theta*: Any-angle path planning and path length analysis in 3d. In *Third Annual Symposium on Combinatorial Search*.

Petres, C.; Pailhas, Y.; Patron, P.; Petillot, Y.; Evans, J.; and Lane, D. 2007. Path planning for autonomous underwater vehicles. *Robotics, IEEE Transactions on* 23(2):331–341.

Rashid, A. T.; Ali, A. A.; Frasca, M.; and Fortuna, L. 2013. Path planning with obstacle avoidance based on visibility binary tree algorithm. *Robotics and Autonomous Systems* 61(12):1440–1449.

Sezer, V., and Gokasan, M. 2012. A novel obstacle avoidance algorithm:follow the gap method. *Robotics and Autonomous Systems* 60(9):1123–1134.

Shah, B. C.; Švec, P.; Bertaska, I. R.; Klinger, W.; Sinisterra, A. J.; Ellenrieder, K. v.; Dhanak, M.; and Gupta, S. K. 2014. Trajectory planning with adaptive control primitives for autonomous surface vehicles operating in congested civilian traffic. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'14)*.

Shah, B. C.; Švec, P.; Bertaska, I. R.; Sinisterra, A. J.; Klinger, W.; von Ellenrieder, K.; Dhanak, M.; and Gupta, S. K. 2015. Resolution-adaptive risk-aware trajectory planning for surface vehicles operating in congested civilian traffic. *Autonomous Robots* 1–25.

Siegwart, R.; Nourbakhsh, I. R.; and Scaramuzza, D. 2011. *Introduction to autonomous mobile robots*. MIT press.

Stentz, A. 1994. Optimal and efficient path planning for partially-known environments. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, 3310–3317. IEEE.

Tovey, J. B. C.; Koenig, T. U. S.; and Nash, A. 2015. Path planning on grids: The effect of vertex placement on path length.

Uras, T., and Koenig, S. 2015a. An empirical comparison of any-angle path-planning algorithms. In *Proceedings of the 8th Annual Symposium on Combinatorial Search*. Code available at: http://idm-lab.org/anyangle.

Uras, T., and Koenig, S. 2015b. Speeding-up any-angle path-planning on grids. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Van Den Berg, J.; Guy, S. J.; Lin, M.; and Manocha, D. 2011. Reciprocal n-body collision avoidance. In *Robotics research*. Springer. 3–19.

Švec, P.; Shah, B. C.; Bertaska, I. R.; Alvarez, J.; Sinisterra, A. J.; Ellenrieder, K. v.; Dhanak, M.; and Gupta, S. K. 2013. Dynamics-aware target following for an autonomous surface vehicle operating under COLREGs in civilian traffic. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'13)*.

Švec, P.; Thakur, A.; Raboin, E.; Shah, B. C.; and Gupta, S. K. 2014. Target following with motion prediction for unmanned surface vehicle operating in cluttered environments. *Autonomous Robots* 36(4):383–405.

Wang, H., and Julier, S. 2011. Path planning in partially known environments.

Wu, Z., and Feng, L. 2012. Obstacle prediction-based dynamic path planning for a mobile robot. *International Journal of Advancements in Computing Technology* 4(3).

Yahja, A.; Stentz, A.; Singh, S.; and Brumitt, B. L. 1998. Framed-quadtree path planning for mobile robots operating in sparse environments. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 1, 650–655. IEEE.

Yap, P.; Burch, N.; Holte, R. C.; and Schaeffer, J. 2011. Block a*: Database-driven search with applications in any-angle path-planning. In *AAAI*.

Yap, P. 2002. Grid-based path-finding. In *Advances in Artificial Intelligence*. Springer. 44–55.

Zhang, Q.; Ma, J.; and Liu, Q. 2012. Path planning based quadtree representation for mobile robot using hybrid-simulated annealing and ant colony optimization algorithm. In *Intelligent Control and Automation (WCICA), 2012 10th World Congress on*, 2537–2542. IEEE.