



An anytime Visibility–Voronoi graph-search algorithm for generating robust and feasible unmanned surface vehicle paths

Marco Schoener¹ · Eric Coyle¹ · David Thompson¹

Received: 17 January 2022 / Accepted: 18 July 2022 / Published online: 23 September 2022
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022

Abstract

While path planning for Unmanned Surface Vehicles (USVs) is in many ways similar to path planning for ground vehicles, the lack of reliable USV models and significant maritime environmental uncertainties requires an increased focus on robustness and safety. This paper presents a novel graph construction method based on Visibility–Voronoi diagrams that allow users to tune path optimality and path safety while considering vehicle dynamics and model uncertainty. The vehicle state is defined as both a 2D location and heading. The method is based on a roadmap generated from a Visibility–Voronoi diagram, and uses motion curves and path smoothing to ensure path feasibility. The roadmap can then be searched using any graph-search algorithm to return optimal paths subject to a cost function. This paper also shows how to generate and search this roadmap in an anytime fashion, which makes the method suitable for local planning where sensors are used to build a map of the environment in real-time. This approach is demonstrated effectively on underactuated systems, with empirical results from USV docking and obstacle field navigation scenarios. These case studies show the path maintains feasibility subject to a simplified vehicle model, and is able to maximize safety when navigating close to obstacles. Simulation results are also used to analyze algorithm complexity, prove suitability for local planning, and demonstrate the benefits of anytime roadmap generation.

Keywords USV · Path planning · Graph search · Feasibility · Anytime algorithm

1 Introduction

Unmanned Surface Vehicles (USVs) are becoming more common in military and civilian applications. Such operations include port security, oceanic surveying, and reconnaissance. With this increased use also comes an increased need for efficient path planning strategies that can safely route the USV through the operating environment.

While path planning is a term used to refer to wide range of navigational algorithms, this paper will focus on developing a spatial plan between two vehicle states for a single agent. Thus, this paper does not consider time-varying trajec-

ries, or multi-agent scenarios. Like many unmanned systems, USVs generally have a well-defined operating area, such as a specific waterway. The set of all vehicle states within this operating area is generally referred to as the *configuration space*, while the set of vehicle states that avoid obstacle collisions is called the *traversable space* or *free space*. However, each vehicle has different motion characteristics. Thus, paths are sought that are both fully within the free space and considered *dynamically-feasible* or *kinematically-feasible*. It is also often desired to achieve an optimal plan subject to a *cost function*, such as minimum distance or minimum time.

While USVs have recently increased in use, unmanned ground vehicles (UGVs) have been in use for a longer period of time and are more common due to the introduction of driverless car technologies. Since both UGVs and USVs can be assumed to be operating on a horizontal plane, planning algorithms are often presented as being applicable to both modalities. However, in practice there are several factors that complicate directly translating UGV planners to USVs. First, generating feasible paths is exceptionally difficult. While there is well-established literature on developing kinematic

✉ Eric Coyle
coylee1@erau.edu

Marco Schoener
marcoschoener@gmail.com

David Thompson
thompd27@gmail.com

¹ Embry-Riddle Aeronautical University, 1 Aerospace Blvd,
Daytona Beach, FL 32114, USA

and dynamic models for UGVs, there is a large amount of uncertainty in how to model USVs and by extension low reliability of these models Fossen (2011); Schoener (2019); Mask (2011). Second, there are often significant errors in the knowledge of the vehicle free space. The lack of infrastructure, the unstructured nature of obstacles (sandbars, floating obstacles, etc.), sensing errors and the non-planar elements of vehicle motion all contribute to this uncertainty. Third, USVs have limited obstacle avoidance capability due to low maneuverability and long stopping distances.

The combination of these factors means that USV path planning should minimize the potential risk of collisions. However, reducing risk comes at the expense of lower quality vehicle paths (as measured by cost). That is, increasing safety leads to staying further from objects, which in turn leads to paths that are longer, take more time to complete, and consume more energy. Thus, *this paper presents a novel USV path planning approach for obtaining feasible paths, while allowing the user to tune path optimality and path safety in the presence of model and environmental uncertainties.* This method is based on a Visibility–Voronoi (V–V) diagram roadmap, which have recently been suggested for USV path planning Niu et al. (2016), and can be searched by traditional graph search algorithms. V–V diagrams are roadmaps where a Visibility graph is created from inflated objects and a Voronoi graph is created inside the inflated objects. The approach presented in this paper improves on existing USV planning and V–V methodologies in the following ways:

Contributions

1. Introduces a new anytime process for generating V–V diagrams and suggests a heuristic for anytime path planning using a V–V diagram
 - Thus the algorithm finds a valid solution as quickly as possible and uses the remaining time to optimize the solution
 - The anytime strategy makes it suitable for online local USV planning, while previous V–V were used for offline global planning
2. Feasible paths are generated for even under-actuated systems, as edges are generated and smoothed using motion curves
3. Each path node now includes position and heading, while prior USV approaches were based only on position
4. Unlike many approaches, results are presented from maritime maps generated on-board a USV through perception techniques

A literature review is presented in Sect. 2, which begins with a general discussion of path planning algorithms and considerations, followed by the techniques that have been presented for generating feasible USV paths. The proposed

roadmap generation method is presented in Sect. 3, along with its anytime variant in Sect. 4. Section 5 presents sample results for a specific case study vehicle, and analyzes real-time performance. Concluding remarks and suggested future work are presented in Sect. 6.

2 Literature review

This literature review will begin with a discussion of general practices for spatial path planning to provide breadth to the techniques used as well as their benefits and challenges for use on USVs. This discussion is organized by the categorizations of path planning algorithms: Roadmaps and Graph Search Theory, Potential Fields, and Sampling-Based Planning (SBP). The general path planning discussion is then followed by the presentation of techniques that have been specifically proposed for USV path planning and a discussion of USV motion modeling.

2.1 Roadmaps and graph search theory

One of the most common approaches to path planning for unmanned systems is to cast the problem in the form of a graph. This approach begins by decomposing the free space into a finite set of vehicle states, called vertices or nodes. Edges between nodes are then generated subject to the requirement that the edges themselves must also lie inside the vehicle's free space. Such graphs are traditionally generated using a Visibility Graph Lozano-Pérez and Wesley (1979), Voronoi Diagram Bhattacharya and Gavrilova (2008), or grid partitioning of the configuration space. However, more advanced techniques generate vertices and edges through sampling-based methods Dunlap et al. (2011); Barraquand and Latombe (1991), (discussed in Sect. 2.3) or attempt to optimize node and edge generation Kumar et al. (2019). The generated graphs are commonly referred to as a *roadmap*.

Once a roadmap through the vehicle's free space has been obtained, the path planning problem is reduced to searching the graph for the best series of edges to follow from the start to goal node. Traditional graph-search algorithms include the Dijkstra algorithm, originally proposed in Dijkstra (1959), and the A* (pronounced "A-star") algorithm, presented in Hart et al. (1968). Both approaches return optimal results (subject to the graph and measured by a cost function), but A* attempts to reduce search time by the introduction of an admissible heuristic. Furthermore, graph search algorithms are considered resolution complete, meaning a solution will be found if one exists as long as the free space is discretized with enough resolution.

The more recent focus of graph search theory has been on improving graph search efficiency. The D* algorithm, presented in Stentz (1995), showed how to efficiently re-plan to

a goal state when node costs are changed. A faster alternative to D*, the D* Lite algorithm, was later presented in Koenig and Likhachev (2005), which incrementally repairs solution paths through a priority queue of nodes where changes occur, referred to as inconsistent nodes. The Anytime Repairing A* (ARA*) algorithm presented in Likhachev et al. (2004), also shows how to use the repairing nature of D* to implement anytime graph searches for path planning applications. ARA* is able to quickly find a valid, often sub-optimal solution, but any remaining time is used to incrementally improve path quality until the optimal solution has been found. Furthermore, the Lifelong Planning A* (LPA*) algorithm presented in Koenig et al. (2004) is able to re-use parts of previous searches regardless of the choice of start and goal states. While searching the graph can be relatively efficient using these approaches, when navigating cluttered environments and accounting for vehicle dynamics Omar et al. (2015), generating the graph itself is often the more computationally expensive task.

2.2 Potential fields

Another class of algorithms used for path planning and navigation are based on Artificial Potential Fields (also called APFs or Potential Fields), which have been implemented on mobile robots since the 1980s Koren and Borenstein (1991). The basic principle of Potential Fields is that a net field is created by a goal that attracts and obstacles that repel the unmanned system. For path planning, the minimum potential route to the goal provides a safe route to the goal state Hwang and Ahuja (1992). However, potential fields are also used for obstacle avoidance where the resultant force magnitude and direction are used to select vehicle direction and speed. One of the better properties about potential fields is the ability to tune the level of safety it provides. To increase safety, an obstacle's repulsive force can be increased, resulting in a vehicle path that will typically move farther from the obstacle and yield longer paths. Conversely, reducing the repulsive force of obstacles will typically decrease the safety distance and yield shorter paths. However, the relationship is not fully deterministic since the net repulsive force depends on object density. Thus settings that work in an object sparse environments are less effective in object dense areas.

Potential fields have also been shown to exhibit a number of undesirable attributes. First, zero resultant force locations other than the goal are difficult for the system to handle natively, sometimes referred to as the local minimum problem. This has been addressed by later works that modify the potential field based on the relationship between the vehicle and obstacle Zhou and Li (2014). Second, the vehicle's path or behavior can be highly oscillatory in cluttered environments, though researchers have presented mitigating strategies to this issue Sfeir et al. (2011). Third, the resulting

path is not guaranteed to be feasible according to a vehicle model. While researchers have shown that pairing potential fields with an optimal controller can allow for planning feasible paths Rasekhipour et al. (2017), accurate vehicles models needed for this approach are difficult to derive for USVs and this work has not been proved experimentally. Additionally, the literature is yet to show how to simultaneously address the local minimum, oscillation, and feasibility concerns of potential field planners.

2.3 Sampling-based planning

Sampling-Based Planning (SBP) reduces the portion of the configuration space considered for planning through sampling the space or control inputs Elbanhawi and Simic (2014). While sampling-based methods have the ability to quickly find solutions to complex planning scenarios, the solutions themselves are generally sub-optimal. Additionally, SBP algorithms do not ensure a solution and are typically only probabilistic complete and/or resolution complete.

One of the first SBP approaches, and one that is still popular today, is a Probabilistic Roadmap (PRM) Barraquand and Latombe (1991). A PRM generates random samples in the configuration space and connects the random samples to the nearest neighbor in the existing roadmap as long as it will not create a collision. While this approach is effective for holonomic vehicles, the paths generated are not guaranteed to be feasible for non-holonomic vehicles such as USVs.

Perhaps the most powerful SBP algorithm in use today is that of a rapidly exploring random tree (RRT), which was originally proposed in LaValle (1998). RRTs begin in a similar fashion to PRMs, by selecting a random sample in the configuration space and finding a nearest neighbor from the existing tree. Rather than creating an edge between these samples, edges are “grown” from the nearest neighbor toward the randomly generated sample. This results in fast exploration of the space and these edges can be grown based on a motion model including motion constraints Kim and Ostrowski (2003), effectively avoiding the problem that PRMs have with non-holonomic vehicles. RRTs are used to directly solve path planning problems by growing separate RRTs between the start and goal states, which when connected contain a valid solution. Efficient re-planning of RRTs has also been studied by Ferguson et al. (2006), where trees are trimmed and reconnected due to changes in the search. While RRTs can efficiently explore the configuration space, the result can be highly sub-optimal. This led to the introduction of the RRT* Karaman and Frazzoli (2011) and anytime RRT* algorithms Karaman et al. (2011) which typically converge toward an optimal solution.

Sampling-Based Model Predictive Control Optimization (SBMPO) also addresses the optimality issue Dunlap et al. (2011); Caldwell et al. (2010). SBMPO differs from RRTs

by sampling the control inputs to the vehicle's motion model, which also eliminates the two-point boundary value problem (BVP) of connecting one state to a future state Dunlap et al. (2011). SBMPO is then paired with the A* graph-search algorithm Hart et al. (1968) to determine node suitability for expansion via additional control input sampling and integration. Additionally, an implicit state grid is used to avoid the local minimum problem that can arise when selecting nodes for expansion based on cost Ericson (2004). Like RRTs, SBMPO is able to generate feasible paths due to its use of a kinematic or dynamic model, though it lacks the time efficiency of RRTs.

2.4 USV path planning

There are significantly fewer path planning methods designed for USVs than ground vehicles. However, USV methods have been proposed from each of the aforementioned path planning categories: graph search algorithms, potential fields, and sampling-based methods. USV algorithms are generally categorized into global and local algorithms Peralta et al. (2020). Global algorithms refer to planning strategies used when complete knowledge of the environment is known, such as areas with known landmasses and few other vessels, and time efficiency is less of a concern. Local algorithms apply when sensors are needed to discover the environmental hazards during navigation.

For global planning, the use of visibility graphs and voronoi diagrams have already been proposed Niu et al. (2016). However, Niu et al. (2019) later showed that a Visibility–Voronoi (V–V) diagram has several advantages over the traditional roadmap approaches. The V–V diagram separates the configuration space into areas close to hazards, where voronoi diagram edges are used, and the remaining area is navigated through visibility diagram edges. The resulting roadmap is able to ensure a tunable level of safety, while optimizing the distance traveled in areas where safety is not a concern. However, this approach does not directly address feasibility. Potential fields have also been shown to have safety benefits Manzini (2017) for global planning in sparse areas, but may have issues with local minimum in obstacle dense environments. The Fast Marching Method (FMM) has been used with USVs to avoid the local minima issue while improving smoothness and safety over global graph search approaches Liu et al. (2015), but the required tuning and computational efficiency is not well-suited for local planning.

In terms of local planning, the work of Chiang and Tapia (2018) attempts to develop a sampling-based path planning algorithm that complies with the rules for maritime vessels developed in the 1972 Convention on the International Regulations for Preventing Collisions, better known as COLREGs. The proposed method is based on a modification of the RRT* algorithm and can avoid stationary and moving

objects. The work of Peralta et al. (2020) compared the USV local planning methods of A*, Potential Fields, RRT*, the Fast Marching Method Sethian (1996) and an updated Fast Marching method on a lake navigation case study. This work concluded that A* should be chosen when optimality is the primary concern and the updated Fast Marching method provided the highest level of security, as determined by author observation on feasibility and safety.

2.4.1 Dynamic modelling of USVs

In order to understand the difficulty in ensuring path feasibility, it is important to briefly discuss the issues associated with dynamic modeling of USVs. It has been shown that obtaining an accurate dynamic model of a boat is a highly complex process, with separate strategies for calm waters and rougher seas (where wave excitation is more significant) Fossen (2011). However, even when focusing on calm waters obtaining an accurate model is challenging. The work of Fossen (2011) is currently viewed by many as the signature resource for such modeling efforts. This work suggests finding the six degree of freedom model using a 3rd-order, truncated Taylor series in order to capture the Hydrodynamic drag and Coriolis effect with a rigid-body ship. The parameters of this model must be experimentally generated by executing a set of predetermined maneuvers, which can be expensive and time consuming. Thus, these modeling efforts are generally best reserved for larger, more expensive vessels or those that will be mass-produced.

There have been works that specifically sought to simplify the modeling process for USVs using a parametric identification approach Mask (2011); Schoener (2019). The work of Mask (2011) used a diagonal 2nd-order, 3 degree of freedom model to capture the primary effects of the control forces of the vehicle. This gave a baseline for control-aided modelling of Surge, Sway and Yaw. The primary disadvantage of this model is there are no cross-coupling effects between these degrees of freedom. For instance, there are no additive drag effects on surge drag from both forward and turning motion, which can be significant when one realizes the difference in side and forward surface area. The work of Schoener (2019) uses a Genetic Algorithm to solve for the high number of model coefficients in its proposed 2nd-order model. In essence, this attempt addresses the issue of accurately determining the highly cross-coupled model coefficients through optimization. While this process was able to capture forward motion and cross-correlated effects, the results were noted to be less accurate than desired and appeared to be on the border of stability due to 2nd order curve-fitting of the coefficients.

In addition to the parametric approaches, researchers have more recently investigated learning methods for USV dynamic modeling Wang et al. (2016); Ensemble (2020). The work of Ensemble (2020) attempts to learn the dynam-

ics directly through an ensemble learning approach, which combines the benefits of multiple weaker learners. The work of Wang et al. (2016) avoids the issues of disturbances and unknown dynamics altogether by using a single-hidden-layer feedforward network to determine control effort and minimize error.

2.4.2 Feasibility through motion curves and smoothing

When an accurate dynamic model is unknown, which is in general true of USVs, a simpler representation of vehicle motion could be sought to ensure feasibility. In ground vehicles, a kinematic model may suffice, but USVs do not have meaningful kinematic models due to the significant effects of vehicle drag. Another approach is to leverage segments of feasible motion, called motion curves, to construct more complex vehicle paths. As long as the transition between the segments is also feasible, then the path itself must be feasible. One such approach is Dubins curves, presented in Dubins (1957), where the shortest path trajectory between two planar vehicle states is found for a vehicle with a minimum turn radius. The resulting paths include only arcs with this minimum turn radius and straight line segments. This work was later extended to include reversing motion by Reeds and Shepp Reeds and Shepp (1990). A similar approach has also been proposed for vehicles that are capable of zero-radius turns Chitsaz et al. (2009). However, these motion curve approaches do not directly show how to avoid collisions and ensure vehicle safety.

Curve interpolation, also known as path smoothing, modifies an existing path to ensure compliance with a set of pre-defined motion constraints. Typical constraints include bounds on vehicle motion (e.g. linear/angular velocity and acceleration) or requiring path velocities and accelerations to be continuous. Smoothing algorithms have been proposed based on Bezier curves Choi et al. (2008), Clothoids Silva and Grassi (2018), and splines Lau et al. (2009). In addition to these approaches, smoothing techniques have also been proposed that avoid collisions with stationary Zhu et al. (2015) or moving obstacles Keller et al. (2014).

3 Primary methodology

3.1 Overview

The USV path planning problem considered in this paper is to generate a local USV path, p , defined by a series of nodes $n = \{x, y, \psi\}$ consisting of a 2D location (x, y) and orientation ψ . The path p must also satisfy the following constraints:

1. p is feasible subject to a model of vehicle motion

2. p should be contained in the free space of the vehicle operating environment OE
3. p should maintain a distance from marine objects of more than d_{thresh} . (Mathematically this is defined as $\text{dist}(p, \mathbf{O}) > d_{thresh}$, where $\text{dist}(p, \mathbf{O})$ returns the minimum distance between p and the set of object boundaries, \mathbf{O}).

Traditionally, the distance d_{thresh} is chosen to satisfy $d_{thresh} = r_{min} + e_{max}$ where r_{min} is the radius of the minimum area bounding circle of the vehicle's 2D geometry and e_{max} is the maximum expected error in the vehicle's ability to follow p . But as e_{max} can be quite large with USVs, this leads to longer paths and can even close off all routes to the goal. Instead, the method presented here uses a roadmap based on a modified V–V diagram to balance the need for appropriate levels of safety with the need to take efficient routes. Unlike other V–V diagram approaches to USV path planning, path feasibility is ensured by generating edges using motion curves and path smoothing. It is also shown that building the roadmap is suitable for local path planning through the use of an anytime variant of roadmap generation.

3.2 Roadmap generation and search

A roadmap R can be viewed as consisting of a set of nodes, and edges between these nodes. Thus

$$R = \{\mathbf{n}, \mathbf{e}\} \quad (1)$$

where \mathbf{n} is the set of all nodes and \mathbf{e} is the set of all edges in R .

Building the modified V–V roadmap used in this paper, requires the user to specify the operating environment of the vehicle, OE , the objects \mathbf{O} contained in OE , and two user-selected distances, d_{safe} and d_{min} . These distances are expected to follow the relationship

$$r_{min} + e_{max} \geq d_{safe} > d_{min} \geq r_{min}. \quad (2)$$

Based on (2), paths that ensure $\text{dist}(p, \mathbf{O}) > d_{min}$ avoid static collisions and have a minimum level of safety, while paths that ensure $\text{dist}(p, \mathbf{O}) > d_{safe}$ are not expected to cause collision due to uncertainties and have the maximum desired level of safety.

The steps needed to generate the modified V–V diagram R are then given by Algorithm 1, where n_s and n_g are nodes that defined the start and goal state respectively.

Algorithm 1 uses three subroutines. The first is buffer (\mathbf{O}, d) , which grows the objects \mathbf{O} by a distance d , resulting in a new set of non-overlapping polygon objects. The second is Voronoi(OE, \mathbf{K}), which computes the voronoi roadmap

Algorithm 1 Building a V–V Roadmap R

Input: $OE, \mathbf{O}, d_{min}, d_{safe}$
Output: R

- 1: $\mathbf{K} = \text{buffer}(\mathbf{O}, d_{min})$ ▷ Minimum clearance bounds
- 2: $\mathbf{B} = \text{buffer}(\mathbf{O}, d_{safe})$ ▷ Desired safety bounds
- 3: $V = \{\mathbf{n}_V, \mathbf{e}_V\} = \text{Voronoi}(OE, \mathbf{K})$
- 4: $V' = \{\mathbf{n}_{V'}, \mathbf{e}_{V'}\} = V \cap B$ ▷ Voronoi roadmap inside \mathbf{B}
- 5: $\mathbf{e}_B = \text{edge}(\mathbf{n}_B, \mathbf{n}_B, \mathbf{B})$ ▷ Generate edges between nodes
- 6: $\mathbf{e}_s = \text{edge}(n_s, \mathbf{n}_B, \mathbf{B})$
- 7: $\mathbf{e}_g = \text{edge}(\mathbf{n}_B, n_g, \mathbf{B})$
- 8: $R = \{\{\mathbf{n}_{V'}, \mathbf{n}_B\}, \{\mathbf{e}_{V'}, \mathbf{e}_B, \mathbf{e}_s, \mathbf{e}_g\}\}$ ▷ Final roadmap nodes and edges

of OE , given object definitions \mathbf{K} based on the method presented in Fortune (1986). Lastly, the function $\text{edge}(\mathbf{n}_1, \mathbf{n}_2, \mathbf{B})$ grows dynamically feasible edges between the nodes in \mathbf{n}_1 and those in \mathbf{n}_2 and returns all edges that do not intersect \mathbf{B} . When Algorithm 1 completes, finding p is simply reduced to an A* graph search on R . As such, p is optimal subject to R and the cost function criteria.

While Algorithm 1 is generalized to use any method of dynamically feasible edge generation, the results presented in this paper will use Dubins curves Dubins (1957), which, as noted in Sect. 2.4.2, finds the shortest path between two states for a vehicle with a minimum turn radius r . Thus, we can effectively replace $\text{edge}(\mathbf{n}_1, \mathbf{n}_2, \mathbf{B})$ with dubins curve edge generation subroutine $\text{Dubins}(\mathbf{n}_1, \mathbf{n}_2, \mathbf{B}, r)$.

While $\text{Dubins}(\mathbf{n}_B, \mathbf{n}_B, \mathbf{B}, r)$ would generate feasible edges between grown objects \mathbf{B} , it has quadratic computational complexity and requires the complex process of checking each generated curve for collision with \mathbf{B} . Thus, a more efficient alternative to $\text{Dubins}(\mathbf{n}_B, \mathbf{n}_B, \mathbf{B}, r)$ should be considered. The work of Niu et al. (2016), which used linear edges for V–V diagram generation, suggests only generating the edges that are considered tangent to an object in \mathbf{B} . However, this concept must be re-considered for Dubins edge generation.

To find a more efficient alternative to generating all Dubins edges, realize the edges within \mathbf{e}_B that are part of the distance-optimal path p will satisfy one of four cases:

1. $e \in \mathbf{e}_B$ connects adjacent nodes on the boundary of a grown object in \mathbf{B} .
2. $e \in \mathbf{e}_B$ is linear and tangent to the bounds of an object in \mathbf{B} at both endpoints.
3. $e \in \mathbf{e}_B$ connects two nodes n_1 and n_2 , where $n_1, n_2 \in \mathbf{n}_B$ and $n_1, n_2 \in \mathbf{n}_{V'}$, denoted $\mathbf{n}_{B,V'} = \mathbf{n}_B \cap \mathbf{n}_{V'}$
4. $e \in \mathbf{e}_B$ is tangent to a boundary in \mathbf{B} at one end and connected to a node in $\mathbf{n}_{B,V'}$ at the other end

All four of these cases are illustrated in the scenario of Fig. 1.

Recognition of these cases allows $\text{Dubins}(\mathbf{n}_1, \mathbf{n}_2, \mathbf{B}, r)$ to be replaced with two more efficient routines. The first of

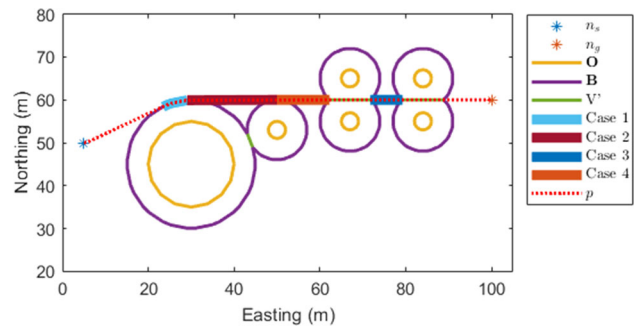


Fig. 1 Roadmap R and resulting distance optimal path p , showing the edges that will be generated by new functions tangents (Case 1 and 2) and VoronoiTangents (Case 3 and 4). Edges not highlighted are generated by Voronoi segments, or Dubins curves involving n_s or n_g

these is $\text{tangents}(\mathbf{n}_B, \mathbf{B})$, which generates tangential edges for the objects in \mathbf{B} (Cases 1 & 2 above). The second routine is $\text{VoronoiTangents}(\mathbf{n}_B, \mathbf{n}_{V'})$, which generates tangent edges between the nodes in \mathbf{n}_B and $\mathbf{n}_{B,V'}$, as well as edges from nodes in $\mathbf{n}_{B,V'}$ to different nodes in $\mathbf{n}_{B,V'}$ (Cases 3 & 4 above). While the Dubins routine would generate an edge to be checked for possible collision between every pair of input nodes, there are significantly fewer tangential edges. Since every edge is checked for collision before returning the final set of edges, the tangent functions are significantly more computationally efficient. With the introduction of these tangent functions, the resulting vehicle roadmap R can then be found using Algorithm 2, and finding p is again reduced to an A* graph search on R .

Algorithm 2 V–V Roadmap using Dubins Curves

Input: $OE, \mathbf{O}, d_{min}, d_{safe}, r$
Output: R

- 1: $\mathbf{K} = \text{buffer}(\mathbf{O}, d_{min})$
- 2: $\mathbf{B} = \text{buffer}(\mathbf{O}, d_{safe})$
- 3: $V = \{\mathbf{n}_V, \mathbf{e}_V\} = \text{Voronoi}(OE, \mathbf{K})$
- 4: $V' = \{\mathbf{n}_{V'}, \mathbf{e}_{V'}\} = V \cap B$
- 5: $\mathbf{e}_B = \text{tangents}(\mathbf{n}_B, \mathbf{B})$ ▷ Generate tangent edges
- 6: $\mathbf{e}_{B,V'} = \text{VoronoiTangents}(\mathbf{n}_B, \mathbf{n}_{V'})$
- 7: $\mathbf{e}_s = \text{Dubins}(n_s, \mathbf{n}_B, \mathbf{B}, r)$
- 8: $\mathbf{e}_g = \text{Dubins}(\mathbf{n}_B, n_g, \mathbf{B}, r)$
- 9: $R = \{\{\mathbf{n}_{V'}, \mathbf{n}_B\}, \{\mathbf{e}_{V'}, \mathbf{e}_B, \mathbf{e}_{B,V'}, \mathbf{e}_s, \mathbf{e}_g\}\}$
- 10: **return** R ▷ Final Roadmap nodes and edges

To better illustrate the steps of Algorithm 1, consider the scenario of Fig. 2a where the operating environment OE , obstacles \mathbf{O} , starting node $n_s = \{5, 5, 90^\circ\}$ and goal node $n_g = \{90, 50, 90^\circ\}$ are as shown. Using values of $d_{min} = 1.5$ and $d_{safe} = 5$ the set of polygon objects \mathbf{K} and \mathbf{B} are found and shown in Fig. 2b (Steps 1 & 2). The voronoi diagram V is then generated based on the limits of the environment OE and \mathbf{K} , as shown in Fig. 2c (Step 3). The intersection of V and \mathbf{B} then yields V' as shown in Fig. 2d (Step 4). Tangent

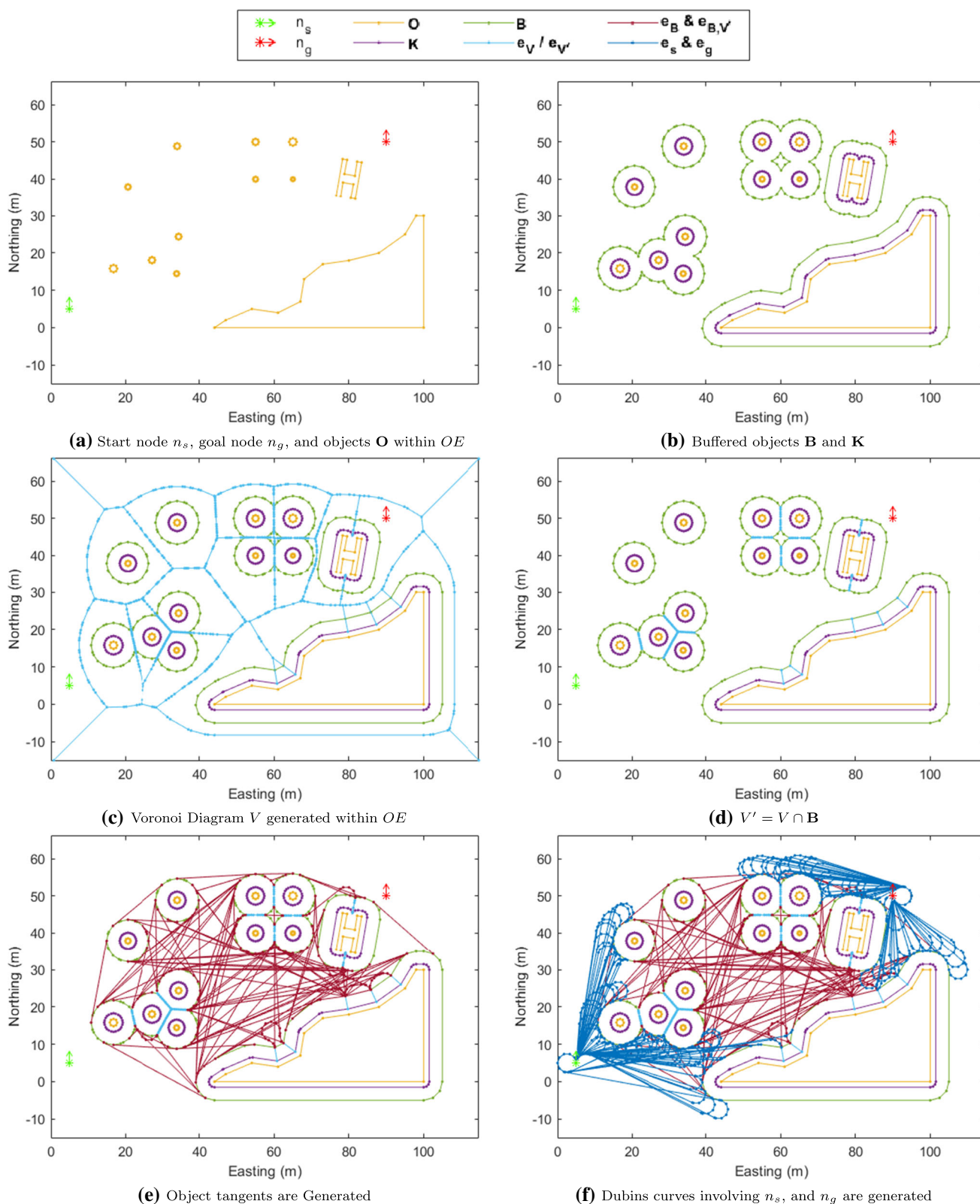


Fig. 2 Steps toward building the Visibility–Voronoi and dubins motion curve roadmap

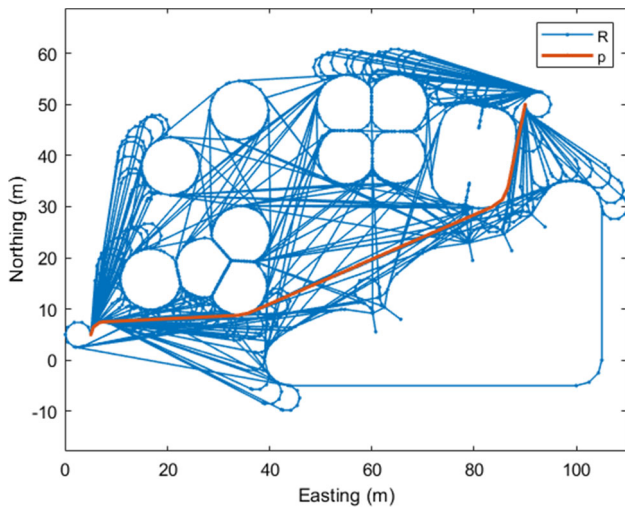


Fig. 3 The final Roadmap R and path p generated by Algorithm 2 for the same scenario as Fig. 2

and voronoi tangent edges are then generated and shown in Fig. 2e (Steps 5 & 6). Next, dubins curves are generated from the start node to the nodes of \mathbf{n}_B and from the nodes of \mathbf{n}_B to the goal node as shown in Fig. 2f (Steps 7 & 8). The roadmap R (returned in Step 9), as well as the distance optimal path p from n_s to n_g , is then given in Fig. 3.

3.3 Path smoothing to ensure feasibility

The set of edges found in $\mathbf{e}_{B,V}, \mathbf{e}_{start}, \mathbf{e}_{goal}$ are all dynamically feasible according to Dubins curves, as are the tangent edges that are a subset of \mathbf{e}_B . However, the rest of the edges in \mathbf{e}_B (which lie on the boundaries of \mathbf{B} , and the edges in $\mathbf{e}_{V'}$

are not guaranteed to be feasible. In fact, when more than two edges in $\mathbf{e}_{V'}$ share a common node $n_i \in n_{V'}$ sharp angles are created by consecutive adjacent voronoi edges as shown in Fig. 4. To address this concern it is suggested that a smoothing function be used to ensure feasibility. While a smoothing function could be used on the entire roadmap, here only p is smoothed to improve efficiency. As Algorithm 2 is based on a vehicle with a minimum turn radius, a minimum curvature smoothing function, such as the one presented in Parlange and Indiveri (2010) is suggested (see Fig. 4).

3.4 Planning close to objects

While Algorithm 1 or Algorithm 2 is effective when n_s and n_g are far from objects, if n_s or n_g are inside \mathbf{B} , there will be no valid edges connecting to these nodes. Similarly, if n_s or n_g is close to \mathbf{B} there may be no Dubins curve involving these states that does not intersect \mathbf{B} . Realize that n_s and n_g will in general be inside or close to \mathbf{B} whenever the vehicle needs to dock, or a new plan is requested while navigating an edge shared with \mathbf{B} .

To address this issue, it is suggested that the boundary used to check collisions of edges be based on $\mathbf{B}' = (\mathbf{B} - \mathbf{SG}) \cup \mathbf{K}$ rather than \mathbf{B} , where \mathbf{SG} is area to be removed from \mathbf{B} . The choice of \mathbf{SG} should be dependent on the chosen motion curve. Here \mathbf{SG} is chosen as two circular regions with diameter d_{SG} centered on n_s and n_g , where d_{SG} is the larger value of $3 * r$ (three times the minimum turn radius of the USV) and d_{safe} . This ensures n_s and n_g can be connected to the rest of the roadmap as long as it is dynamically feasible to do so without collision. Note that \mathbf{SG} can be viewed as a algorithm parameter, which may need to be tuned when using

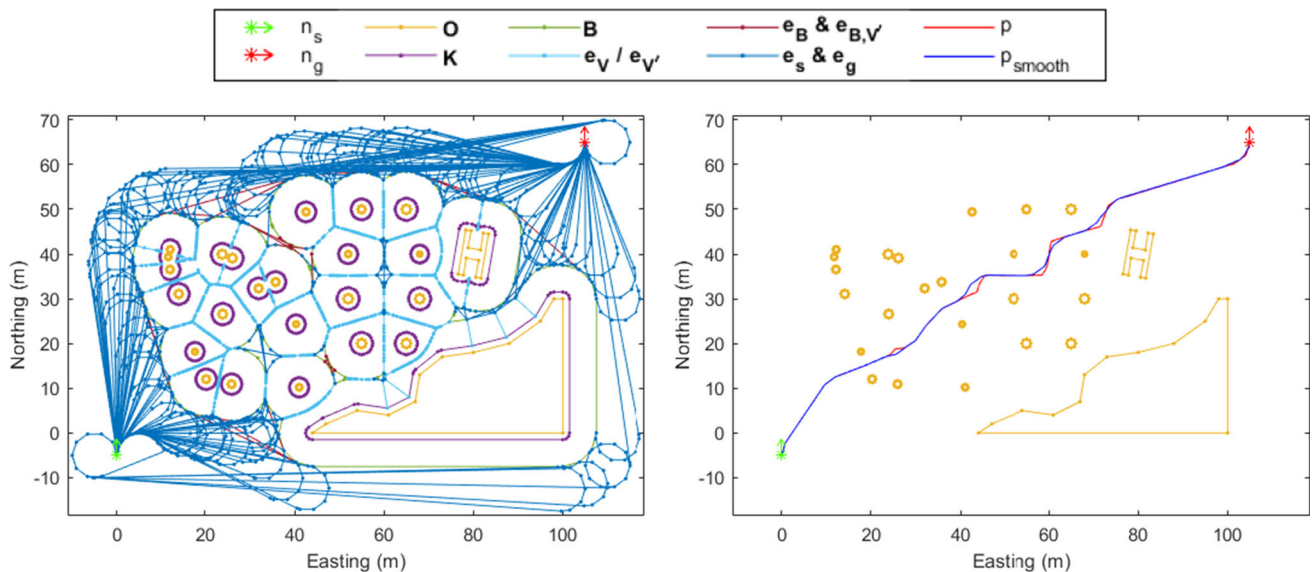


Fig. 4 Left: Example roadmap that requires smoothing of Voronoi edges $\mathbf{e}_{V'}$ for feasibility. Scenario uses settings of $d_{safe} = 7.5$ m, $d_{min} = 1.5$ m, and $r = 5$ m. Right: Original path p from roadmap compared to smooth path p_{smooth} that results from Dubins Curve smoothing of p

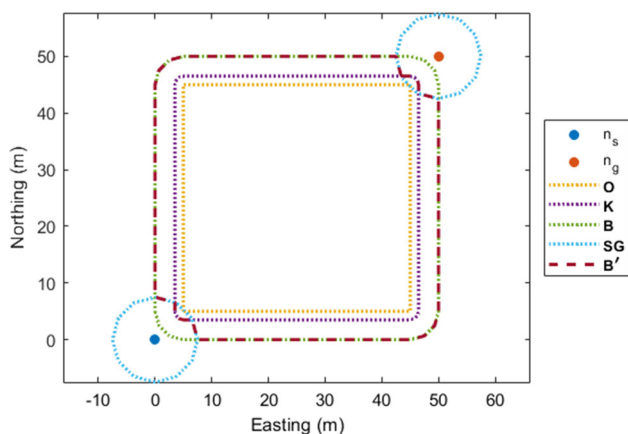


Fig. 5 Visual representation of computing $\mathbf{B}' = (\mathbf{B} - \mathbf{SG}) \cup \mathbf{K}$ when the start and/or goal are close to an object. Here $d_{safe} = 5$ m, $d_{min} = 1.5$ m, and $r = 2.5$ m

motion curves other than Dubins. An example calculation of \mathbf{B}' is found in Fig. 5.

4 Anytime methodology

While the roadmaps produced by Algorithm 2 satisfy the need for feasible and safe USV paths, generating the roadmap has the potential to be computationally intensive if OE encompasses a large number of objects or highly complex objects. Thus, generating the full USV roadmap may require more time than the USV can allow before needing to act on a plan. One way to address this issue is to alter roadmap generation to be an anytime process.

Generating R as an anytime process is based on a few key insights about the relationship between the roadmap R and optimal path p generated in Algorithm 2.

1. R could be built incrementally by adding objects in \mathbf{O} to R sequentially rather than as a single batch
2. The distance optimal path p is more likely to contain edges and nodes generated from objects close to the most direct route from n_s to n_g .
3. During the building process, there is no value in adding an object in \mathbf{O} to R unless it has the potential to result in a lower cost path than current best path from n_s to n_g found in R .

The first insight may at first imply the roadmap can be built one object $o \in \mathbf{O}$ at a time, but in actuality R should be built one safety boundary $b \in \mathbf{B}$ at a time. This is because a given b may contain any number of objects in \mathbf{O} and it is the multiple nature of these objects that leads to adding voronoi edges between objects to the roadmap. The second insight suggests that a priority queue can be established which determines the

order in which objects in \mathbf{O} should be added to R based on the distance to the line segment connecting n_s to n_g . Based on the third insight, the cost of the best path in R should be monitored and an admissible heuristic established for determining which objects should remain in the priority queue during the building process. When using a distance-based cost function, the path length forms this cost bound and the sum of the minimum distance from n_s and n_g to the object's safety boundary forms an admissible heuristic.

Before presenting the anytime V–V Roadmap generation algorithm, consider that collision detection in Algorithm 2 requires the potentially complex batch process $\mathbf{B} = \text{buffer}(\mathbf{O}, d_{safe})$. However, in the field of collision detection, the use of shape primitives, such as axis-aligned bounding boxes (AABBs), is often suggested to efficiently rule out collisions before conducting the more complex process of intersection with polygon bounds such as those found in **B** Cai et al. (2014). Thus for anytime roadmap generation a list of collision detection geometries \mathbf{CD} will be maintained, which will initially be composed of the AABB of each object o_i in \mathbf{O} grown by a distance d_{safe} . During algorithm execution if the AABB of an object o_i causes a collision with a roadmap edge or o_i is selected to generate roadmap nodes and edges, the function $\text{overlaps}(o_i, \mathbf{o}, d_{safe})$ is used to generate the the list of all objects in \mathbf{O} , denoted \mathbf{o}_i , that are $2d_{safe}$ distance-reachable from o_i . An object o_j is $2d_{safe}$ distance-reachable from o_i if an ordered set of objects from o_i to o_j exists such that adjacent objects in the set are never separated by a distance of more than $2d_{safe}$. The AABB of each object in the set \mathbf{o}_i is then removed from \mathbf{CD} and replaced by $B_i = \text{buffer}(\mathbf{o}_i, d_{safe})$, where B_i will be a single polygon boundary.

Based on the presented insights and collision detection strategy, the anytime procedure for roadmap generation is defined by Algorithm 3, where $q = \text{sort}(\mathbf{O})$ sorts the objects starting with the object closest to the line segment from n_s to n_g . Collision checks utilize the full set of collision detection geometries \mathbf{CD} , and not just the collision detection geometries corresponding to objects currently in the roadmap.

If Algorithm 3 is interrupted, then \hat{p} contains the best path currently found in the roadmap. If the algorithm is allowed to finish, \hat{p} will be equivalent to the optimal path p . While Algorithm 3 is presented as a sequential algorithm, the architecture lends itself to parallelization of the main loop and parallel computations within each loop iteration through the use of queues for generating edges and check generated edges for collisions before adding them to the roadmap. Only sequential processing is considered in this paper however, and an example Anytime V–V Roadmap Generation is shown in Fig. 6.

From Fig. 6 it can be seen that after the first iteration of the main loop from Algorithm 3, edges have been generated due to the selection of only a single object and added to R .

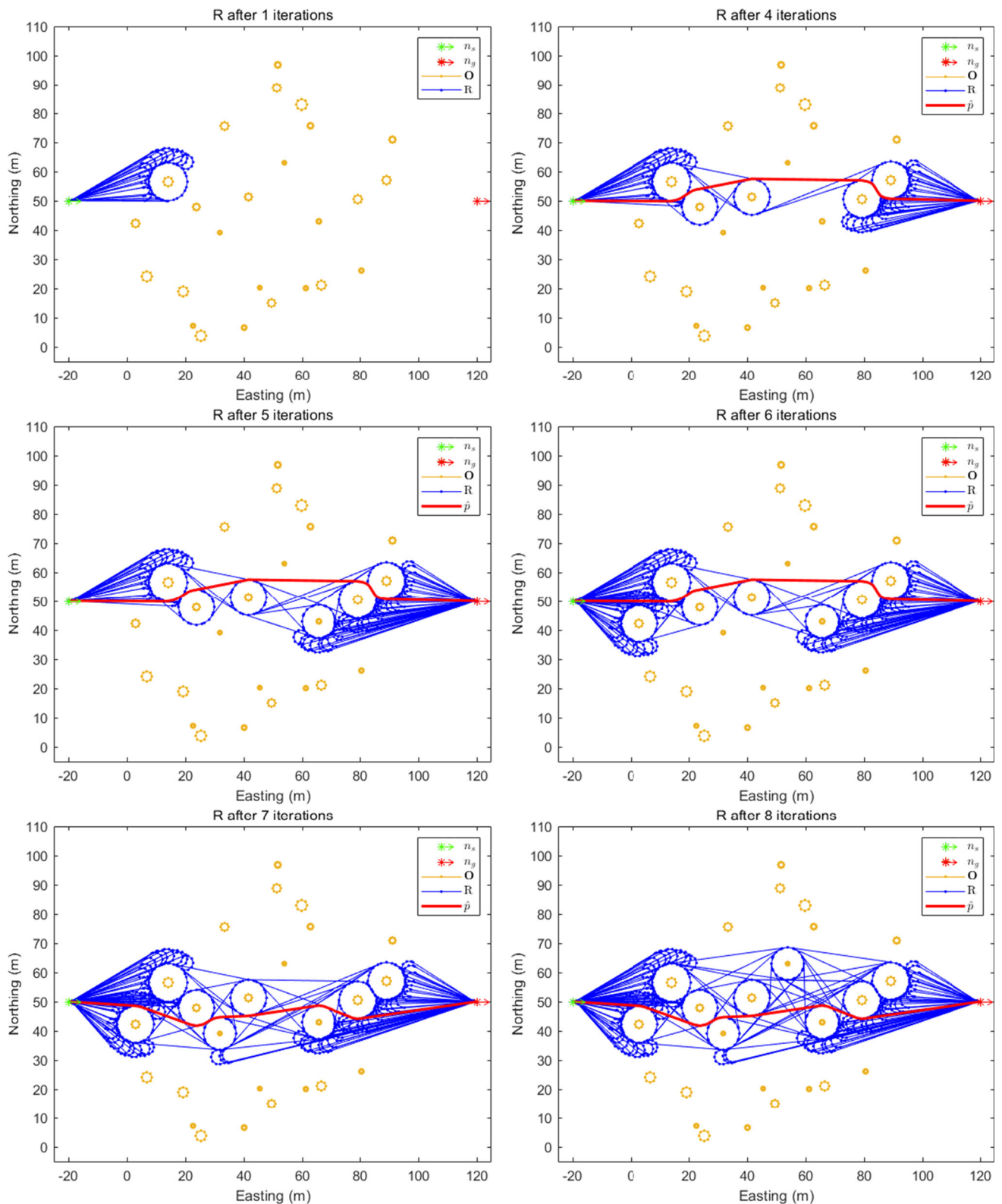


Fig. 6 Progression of Anytime V-V Roadmap building using example scenario with $n_s = \{-20m, 50m, 0^\circ\}$ to $n_g = \{120m, 50m, 0^\circ\}$, $d_{min} = 1.5$ m, $d_{safe} = 5$ m, and $r = 2.5$ m

Algorithm 3 Anytime V–V Roadmap Generation using Dubins Curves

Input: $OE, \mathbf{O}, d_{min}, d_{safe}, r$
Output: R

- 1: Initialize R, \mathbf{B} ▷ Both empty
- 2: $\mathbf{CD} = \text{AABB}(\mathbf{O})$
- 3: Initialize Queue $q = \text{sort}(\mathbf{O})$
- 4: **while** $q \neq \emptyset$ **do**
- 5: $o_{best} = q.\text{front}()$ ▷ first element
- 6: $\mathbf{o}_{best} = \text{overlaps}(o_{best}, \mathbf{O}, d_{safe})$
- 7: $q.\text{remove}(\mathbf{o}_{best})$ ▷ remove element
- 8: $\mathbf{CD}.\text{remove}(\text{AABB}(\mathbf{o}_{best}))$
- 9: $B_{best} = \text{buffer}(\mathbf{o}_{best}, d_{safe})$
- 10: $\mathbf{CD}.\text{push}(B_{best}), \mathbf{B}.\text{push}(B_{best})$ ▷ add element
- 11: $\mathbf{e}_{s,best} = \text{Dubins}(n_s, \mathbf{n}_{B_{best}}, \mathbf{CD}, r)$
- 12: $\mathbf{e}_{best,g} = \text{Dubins}(\mathbf{n}_{B_{best}}, n_g, \mathbf{CD}, r)$
- 13: $\mathbf{e}_{best} = \text{tangents}(\mathbf{n}_{B_{best}}, \mathbf{n}_{\mathbf{B}}, \mathbf{CD})$
- 14: $\mathbf{K}_{best} = \text{buffer}(\mathbf{O}_{best}, d_{min})$
- 15: $V'_{best} = \{\mathbf{n}_{V'_{best}}, \mathbf{e}_{V'_{best}}\} = \text{Voronoi}(B_{best}, \mathbf{K}_{best})$
- 16: $\mathbf{e}_{\mathbf{B},V'} = \text{VoronoiTangents}(\mathbf{n}_{\mathbf{B}}, \mathbf{n}_{V'_{best}})$
- 17: $\mathbf{e} = \{\mathbf{e}_{\mathbf{B}_{best}}, \mathbf{e}_{s,best}, \mathbf{e}_{best,g}, \mathbf{e}_{V'_{best}}, \mathbf{e}_{\mathbf{B},V'}\}$
- 18: $R = R \cup \{\{\mathbf{n}_{\mathbf{B}_{best}}, \mathbf{n}_{V'_{best}}, \mathbf{e}\}\}$ ▷ update roadmap
- 19: $\hat{p} = \text{astar}(R)$
- 20: $\hat{d} = \text{pathlength}(\hat{p})$
- 21: **for all** $o_i \in q$ **do**
- 22: **if** $(\text{dist}(n_s, o_i) + \text{dist}(o_i, n_g)) \geq \hat{d}$ **then**
- 23: $q.\text{remove}(o_i)$ ▷ remove element
- 24: **end if**
- 25: **end for**
- 26: **end while**
- 27: **return** R

Furthermore, there is no path within R between n_s and n_g until the fourth algorithm iteration. It can also be seen after the 4th iteration that all objects chosen for edge generation are close to the line between n_s and n_g , which is the dubins curve heuristic for this case. Thus, if the algorithm was interrupted at any point after the completion of iteration 4, then a valid path would be returned. However, by giving the algorithm more time, it is able to improve the path quality in iteration 5 and again in iteration 7. Once iteration 8 is completed, the algorithm is completes due to an empty queue q , and the \hat{p} can now be assured to be the optimal path p .

5 Results

As previously discussed, the V–V work of Niu et al. (2019) is the most closely related published work to the method presented in this paper, but its authors state it is intended for global planning. The results shown below, which are broken into two studies of algorithm performance, are intended to show the method presented in this paper is suitable for local planning. The first is a case study of paths obtained when avoiding real marine objects and algorithm parameters consistent with a medium-sized (16-foot long) USV. The case study is discussed in Sect. 5.1. The second study uses



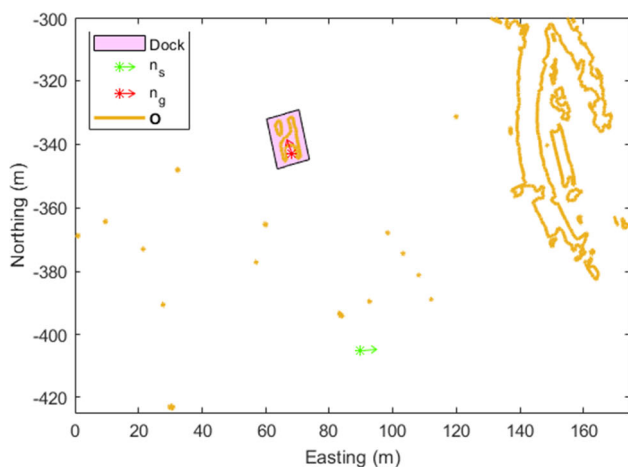
Fig. 7 The Minion Research Platform is a 16-foot long USV that is equipped with a high fidelity GPS/INS and four multi-beam Lidar sensors

randomly generated scenarios to analyze the computational complexity of the presented algorithm and applicability to local planning where plans are obtained in real-time. These simulation results are presented in Sect. 5.2. The computer used to produce the processing times noted in Sects. 5.1 and 5.2 has a Intel i7-8650U CPU @ 1.9 GHz and 16 GB of RAM. All timing results are for single-threaded performance.

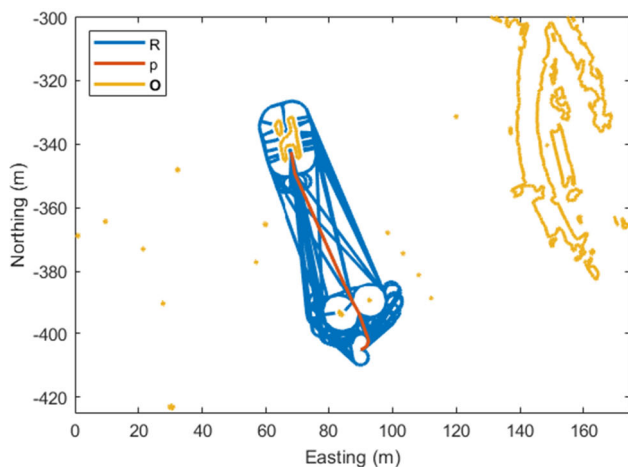
5.1 Case study: Maritime RobotX Challenge

The case study presented here is based on the Maritime RobotX Challenge, which is an international, university-level student competition designed to foster student interest in autonomous and robotic systems operating in the maritime domain, with an emphasis on the science and engineering of cooperative autonomy. This competition involves tasks derived from existing Naval operations and challenges, such as USV docking, complex navigation, and environmental perception. All competing USVs are based on the Marine Advanced Research 16-foot WAM-V platform, which is the vessel studied in Schoener (2019) and Mask (2011). This competition motivated the development of the path planning algorithm presented in this paper due to tasks that require navigation close to obstacles, but a need to handle uncertainties in vehicle model and environment.

The ERAU-owned WAM-V USV, named Minion (see Fig. 7), is designed to allow the operator to select vehicle control by either differential thrust or azimuth control of the propulsion motors. However, both control schemes result in under-actuated systems, yielding a minimum turn radius that varies with vehicle speed. This means Dubins curve edge generation is appropriate when Minion operates at a pre-defined vehicle speed. As an under-actuated system, the USV has an inability to compensate for some unknown disturbances or modeling errors. The vehicle is also equipped with four Lidar sensors for environmental perception, which map objects using the scheme defined in Thompson et al. (2019). Thus, the V–V path planning scheme proposed in this paper is appropriate for this USV and its perception suite. Minion



(a) Docking Scenario from 2018 Maritime RobotX Challenge



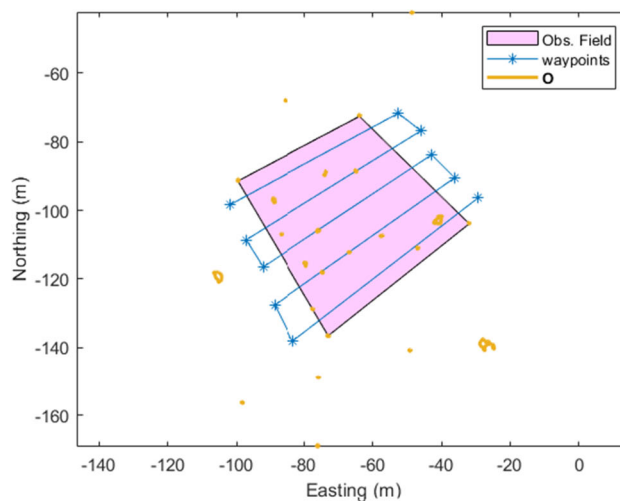
(b) Roadmap R and path p found using Algorithm 3

Fig. 8 A docking scenario from the 2018 Maritime RobotX Competition. In this scenario $n_s = \{90m, -405m, 6^\circ\}$ and $n_g = \{68.25m, -342.5m, 73^\circ\}$ and all obstacles are created from empirical data. Algorithm 3 is implemented using settings of $d_{safe} = 5m$, $d_{safe} = 1.5m$, and $r = 2.5m$

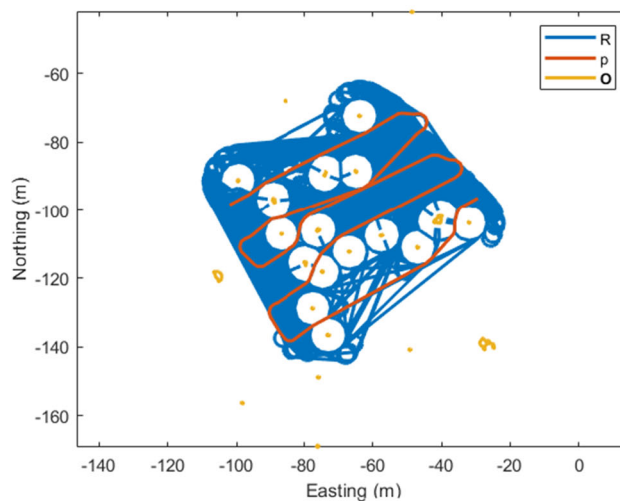
is just under 3 m wide, so $d_{min} = 1.5m$ is chosen. Also, Minion's typical operational speed is 1.5 m/s, which corresponds to a minimum turn radius of just under 2.5 m, thus $d_{safe} = 5m$ and $r = 2.5m$ are chosen.

The results presented in this paper will utilize planning scenarios Minion encountered when attempting the docking and obstacle field navigation tasks at the 2018 Maritime RobotX Challenge. During competition, the vehicle state and Lidar measurements were recorded. By replaying these logs and executing the perception algorithm presented in Thompson et al. (2019), the originally perceived environment can be re-created, as seen in Figs. 8a and 9a.

During the docking scenario, the vehicle must navigate into one of two docking bays on a H-shaped, moored, float-



(a) Obstacle Field Scenario from 2018 Maritime RobotX Challenge



(b) Roadmap R and path p found using Algorithm 3

Fig. 9 A obstacle field scenario from the 2018 Maritime RobotX Competition using empirical data. Algorithm 3 is implemented using settings of $d_{safe} = 5m$, $d_{safe} = 1.5m$, and $r = 2.5m$ over a series of waypoints

ing dock. Given that Minion is just under 3 m wide and the docking bays are only 3.2 m wide, this is a scenario where the planning algorithm must plan close to an object (the dock) but stay as far from the object as it can and still succeed at the task. After identifying the docking bay and its orientation using the methodology presented in Barnes et al. (2018), Algorithm 3 produces the plan shown in Fig. 8b. The plan first navigates between the two buoys located at approximately $(84E, -394N)$ and $(93E, -389.5N)$ while following the voronoi edges to maximize safety. The path then connects to the voronoi edges to lead it into the center of the bay, which maximizes safety and likelihood of success when entering the dock. The optimal vehicle path is returned in 114msecs in this scenario.

The RobotX Challenge’s obstacle field scenario requires the vehicle to make several “lawn-mower” style passes through an obstacle field marked by four cylindrical, white buoys. The obstacles within the field are spherical in nature and vary from 0.5 to 1.2 m in diameter. Figure 8a shows the perceived objects, outlines the area marked by the four cylindrical buoys, and shows the 10 target waypoints for completing the lawnmower pattern.

Algorithm 3 is applied to each consecutive pair of waypoints, and the path between each pair of waypoints is concatenated to give a complete “lawn-mower” pattern p as shown in Fig. 9b. It can be seen that the final vehicle path p closely mirrors the originally designated pattern, but deviates in order to maintain safety. This is understandable considering the waypoints were assigned manually without consideration of the objects inside the obstacle field. When p is less than $d_{safe} = 5$ m from an object, p follows the voronoi edges between the objects. However, the path is never less than $d_{min} = 1.5$ m from an object. Algorithm 3 returned the Fig. 9b result, which consists of 9 separate paths (one path between each consecutive pair of the 10 waypoints), in a total of 451 msec.

5.2 Simulated environment

Given the need to consider edges between each node pair in R , it is reasonable to assume the order of complexity is $O(N^2)$ for Algorithm 1, where N is the number of nodes used in R . If each object $o \in \mathbf{O}$ has a nearly equivalent number of nodes, it can be further assumed that Algorithm 2 would have a complexity of $O(m^2)$, where m is the number of objects in \mathbf{O} . However, since Algorithm 3 only adds nodes and vertices to R for objects likely to influence the optimal path p , Algorithm 3 is expected to have a complexity of $O(\tilde{m}^2)$, where \tilde{m} is the number of objects for which nodes are added to R before the optimal path is found. Thus, $\tilde{m} \leq m$. Furthermore, if Algorithm 3 returns before completion, then an estimate of the optimal path \hat{p} may be found which required \hat{m} objects to be added to the roadmap, where $\hat{m} \leq \tilde{m}$. So, it is theorized in order to return a non-empty path \hat{p} the complexity is quadratic ($O(\hat{m}^2)$), but the quadratic complexity results in significantly slow execution only when the local environment is highly complex.

To test the theorized complexity of Algorithm 3, and determine if the algorithm is efficient enough for local planning, a series of simulations are conducted. These simulations are based on a 500 m \times 500 m operating area OE , with random starting state, n_s , and random goal state, n_g . A thousand random scenarios are then generated for the case where OE contains 0, 25, 50, 100, 200, 250, 300, 400, and 500 obstacles. Objects are non-overlapping regular hexadecagons (16-sided polygon) circumscribed by a circle with radius between 0.5m and 2m and the object location within OE is random. For

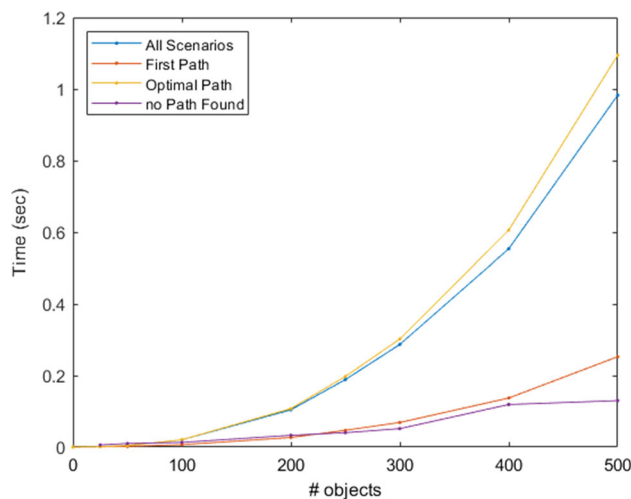


Fig. 10 Average timing for 1000 random scenarios of non-overlapping 16-sided objects

each scenario the algorithm completion time and time to find the first path estimate \hat{p} is recorded. The completed simulations are then sorted based on if a solution exists. The average algorithm completion time for all scenarios and scenarios where a solution does not exist is shown in Fig. 10. Furthermore, when a solution does exist, Fig. 10 shows both the average time to find the the first path estimate \hat{p} , and the time to find the optimal path p .

Figure 10 is strong evidence that the Algorithm 3 has a quadratic complexity of $O(\tilde{m}^2)$. The completion time of all scenarios (Blue line), in fact has a coefficient of determination $r^2 = 0.998$ for a quadratic fit. Furthermore, there is also a strong quadratic relationship when looking at the time to find the first estimate of the optimal path (Red line) and the time to find the optimal path (Orange Line), with coefficients of determination of 0.996, and 0.998 respectively. However, the benefits of the anytime approach are clearly shown as well, as the first path is in on average returned much faster than the time it takes to return the known optimal path.

The one surprising result of Fig. 10 is the algorithm complexity when a valid path does not exist appears to be linear instead of quadratic with a coefficient of determination of 0.916. To explain this phenomenon, consider that a path will not exist if and only if n_s or n_g is so close to objects that feasible edges cannot be created from n_s or to n_g without collision. Recall that Algorithm 3 prioritizes adding objects to R based on distance to n_s and n_g . Thus, if there are objects within a $2r$ distance of n_s or n_g , then they will be added to R first. Furthermore, once these objects are added to R a edge involving n_s and n_g should exist if it will ever exist. If it does not exist at that time, the algorithm can terminate. As there is a physical limit to how many objects can be within $2r$ of n_s and n_g , the complexity is not quadratic or linear based on the number of objects, but bounded when a path will not exist.

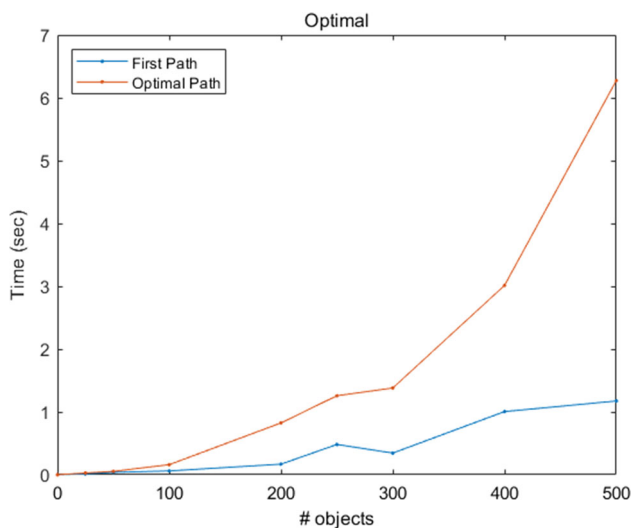


Fig. 11 Worst case timing results from 1000 random scenarios of non-overlapping 16-sided objects

However, the complexity will still be quadratic based with respect to the total number of nodes on the objects within $2r$ of n_s and n_g .

While algorithm complexity is important to understanding performance, the time required to obtain a path in real-world scenarios is often a more practical measure for determining applicability to local planning. Additionally, returning a valid path is often a higher priority than returning an optimal path. Thus, Fig. 11 presents the worst-case completion time for obtaining an optimal path and obtaining any path among all randomly generated scenarios.

The results of Fig. 11 indicate that even in a worst-case scenario, the proposed algorithm can be effective for local planning. To understand this, consider that a USV will often move at a slow rate of speed. For example, the Minion USV, for which this algorithm was originally developed, typically operates at 1.5 m/s. While finding the optimal path was shown to take up to 6.27 s in the most obstacle dense environment, finding a valid solution only requires 1.17 s. This worst case time corresponds to traveling 1.8 m while moving at 1.5 m/s and the results of Fig. 10 indicates that this distance will typically be much shorter with a 250 ms average time yielding a travel distance of 0.375m. However, if the algorithm chose to wait on the optimal path, then it could travel close to 10m while waiting on a solution. So there is tremendous value in the anytime approach.

While the results shown here are based on the number of obstacles, increasing the number of nodes should also yield a quadratic complexity. This is due to the original hypothesis of $O(N^2)$ complexity, where N is the number of nodes used in R . More obstacles means more nodes, but more complex obstacle geometries also increases the total number of nodes. Thus, a decimation algorithm such as the

Douglas-Peucker algorithm Douglas and Peucker (1973) or Visvalingam algorithm Visvalingam and Whyatt (1993) is highly recommended for use with this algorithm to decrease the complexity of the obstacles themselves.

6 Conclusion

This paper builds on existing USV path planning and Visibility–Voronoi (V–V) diagram approaches in several key ways. First, the presented approach is presented as an anytime process. This anytime approach prioritizes node and edge generation in areas where the optimal path is expected to lie, as defined by a heuristic. This often enables the algorithm to return a feasible, near-optimal path even when interrupted before completion due to the relatively sparse nature of the USV environment. Second, the presented method differs from existing USV and V–V approaches by using motion curves and path smoothing to ensure path feasibility subject to a simplistic vehicle model. This is especially important for USVs, which are primarily under-actuated systems. Third, unlike prior V–V approaches, all roadmap nodes include a definition of vehicle heading, which improves complex maneuvering capabilities. Lastly, it is shown that this new V–V planning approach allows the user to balance uncertainties in the vehicle model and environment with the need for better paths (as measured by a cost function). This is done through the use of two safety boundaries, one that ensures a minimum level of safety at all times, and a second that is only violated when necessary to find a solution or achieve a significantly better path in terms of cost. Furthermore, it is shown that the algorithm has a quadratic complexity when a valid path exists and a near-linear complexity when a valid path does not exist. Unlike many prior approaches that are validated using only simulated scenarios, real-time maps from a USV perception suite are used in addition to random simulations to prove the effectiveness of the presented approach.

6.1 Future work

Several efficiency-based improvements should be considered in future work. First, the authors believe significant computational improvements can be achieved through parallelization of the presented algorithm, both in how objects are added to the roadmap and in how edges are generated and subsequently checked for collision. Second, future work should also consider methods to reduce the number of edges generated, as most motion curve generated edges involving the start and goal node are thrown out due to object collision. Third, treating collision detection as a 3D problem could lead to improvements in path safety and cost. Lastly, efficient methods of re-planning should be investigated.

Future work should also consider the use of other motion curves. While the general approach presented here is agnostic to the choice of motion curve, several of the efficiency-based improvements used in the anytime version of the algorithm are based on Dubins curves. As a result, the use of other motion curves may require new assumptions, simplifications, and heuristics. The use of other motion curves will also require the use of a different smoothing function.

Acknowledgements The author's contributions to this work were partially supported by the Department of Defense (DoD) Science, Mathematics, and Research for Transformation (SMART) Scholar Program.

References

- Barnes, J. E., Bloom, N. D., Cronin, S. P., Grady C. D., J. L. H., Helms, M. R., Hendrickson, J. J., Middlebrooks, N. R., Moline, N. D., III, Romney, J. S., Schoener, M. A., Schultz, N. C., Thompson, D. J., Zuercher, T. A., Reinholtz, C. F., Coyle, E. J., Currier, P. N., Butka, B. K., & Hockley, C. J. (2018). Design of the minion research platform for the 2018 maritime robotx challenge. Tech. rep., Embry-Riddle Aeronautical University, Department of Mechanical Engineering.
- Barraquand, J., & Latombe, J. C. (1991). Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research*, 10(6), 628–649. <https://doi.org/10.1177/027836499101000604>.
- Bhattacharya, P., & Gavrilova, M. L. (2008). Roadmap-based path planning: Using the voronoi diagram for a clearance-based shortest path. *IEEE Robotics Automation Magazine*, 15(2), 58–66. <https://doi.org/10.1109/MRA.2008.921540>.
- Cai, P., Indhumathi, C., Cai, Y., Zheng, J., Gong, Y., Lim, T. S., & Wong, P. (2014). *Collision detection using axis aligned bounding boxes* (pp. 1–14). Springer. https://doi.org/10.1007/978-981-4560-32-0_1
- Caldwell, C. V., Dunlap, D. D., & Collins, E. G. (2010). Motion planning for an autonomous underwater vehicle via sampling based model predictive control. In *Oceans 2010 MTS/IEEE seattle* (pp. 1–6). <https://doi.org/10.1109/OCEANS.2010.5664470>
- Cheng, C., Xu, P. F., Cheng, H., Ding, Y., Zheng, J., Ge, T., et al. (2020). Ensemble learning approach based on stacking for unmanned surface vehicle's dynamics. *Ocean Engineering*, 207, 107388. <https://doi.org/10.1016/j.oceaneng.2020.107388>.
- Chiang, H. T. L., & Tapia, L. (2018). Colreg-rrt: An rrt-based colregs-compliant motion planner for surface vehicle navigation. *IEEE Robotics and Automation Letters*, 3(3), 2024–2031. <https://doi.org/10.1109/LRA.2018.2801881>.
- Chitsaz, H., LaValle, S. M., Balkcom, D. J., & Mason, M. T. (2009). Minimum wheel-rotation paths for differential-drive mobile robots. *The International Journal of Robotics Research*, 28(1), 66–80. <https://doi.org/10.1177/0278364908096750>.
- Choi, J. w., Curry, R., & Elkaim, G. (2008). Path planning based on bézier curve for autonomous ground vehicles (pp. 158–166). <https://doi.org/10.1109/WCECS.2008.27>
- Dijkstra, E. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1), 269–271. <https://doi.org/10.1007/BF01386390>.
- Douglas, D. H., & Peucker, T. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10, 112–122.
- Dubins, L. E. (1957). On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3), 497–516.
- Dunlap, D., Caldwell, C., Collins, E. J., & Chuy, O. (2011). Motion planning for mobile robots via sampling-based model predictive optimization, chap. 11 (pp. 211–232). IntechOpen. <https://doi.org/10.5772/17790>
- Elbanhawi, M., & Simic, M. (2014). Sampling-based robot motion planning: A review. *IEEE Access*, 2, 56–77. <https://doi.org/10.1109/ACCESS.2014.2302442>.
- Ericson, C. (2004). *Real-time collision detection*. CRC Press Inc.
- Ferguson, D., Kalra, N., Stentz, A. (2006). Replanning with rrts. In *Proceedings 2006 IEEE international conference on robotics and automation*, 2006. ICRA 2006 (pp. 1243–1248). <https://doi.org/10.1109/ROBOT.2006.1641879>
- Fortune, S. (1986). A sweepline algorithm for voronoi diagrams. In *Proceedings of the second annual symposium on computational geometry, SCG '86* (pp. 313–322). Association for Computing Machinery, New York, NY, USA. <https://doi.org/10.1145/10515.10549>
- Fossen, T. I. (2011). *Handbook of marine craft hydrodynamics and motion control*. Wiley. <https://doi.org/10.1002/9781119994138>
- Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100–107. <https://doi.org/10.1109/TSSC.1968.300136>.
- Hwang, Y., & Ahuja, N. (1992). A potential field approach to path planning. *IEEE Transactions on Robotics and Automation*, 8(1), 23–32. <https://doi.org/10.1109/70.127236>.
- Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7), 846–894. <https://doi.org/10.1177/0278364911406761>.
- Karaman, S., Walter, M. R., Perez, A., Frazzoli, E., & Teller, S. (2011). Anytime motion planning using the rrt*. In *2011 IEEE international conference on robotics and automation* (pp. 1478–1483). <https://doi.org/10.1109/ICRA.2011.5980479>
- Keller, M., Hoffmann, F., Hass, C., Bertram, T., & Seewald, A. (2014). Planning of optimal collision avoidance trajectories with timed elastic bands. *IFAC Proceedings Volumes*, 47(3), 9822–9827. <https://doi.org/10.3182/20140824-6-ZA-1003.01143> (19th IFAC World Congress).
- Kim, J., & Ostrowski, J. (2003). Motion planning a aerial robot using rapidly-exploring random trees with dynamic constraints. In *2003 IEEE international conference on robotics and automation (Cat. No.03CH37422)* (Vol. 2, pp. 2200–2205). <https://doi.org/10.1109/ROBOT.2003.1241920>
- Koenig, S., & Likhachev, M. (2005). Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, 21(3), 354–363. <https://doi.org/10.1109/TRO.2004.838026>.
- Koenig, S., Likhachev, M., & Furcy, D. (2004). Lifelong planning a*. *Artificial Intelligence*, 155(1), 93–146. <https://doi.org/10.1016/j.artint.2003.12.001>.
- Koren, Y., & Borenstein, J. (1991) Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings of 1991 IEEE international conference on robotics and automation* (Vol. 2, pp. 1398–1404). <https://doi.org/10.1109/ROBOT.1991.131810>
- Kumar, R., Mandalika, A., Choudhury, S., & Srinivasa, S. S. (2019). LEGO: leveraging experience in roadmap generation for sampling-based planning. CoRR [arxiv:1907.09574](https://arxiv.org/abs/1907.09574)
- Lau, B., Sprunk, C., & Burgard, W (2009). Kinodynamic motion planning for mobile robots using splines (pp. 2427–2433). <https://doi.org/10.1109/IROS.2009.5354805>

- LaValle, S. (1998). Rapidly-exploring random trees: A new tool for path planning. The annual research report.
- Likhachev, M., Gordon, G. J., & Thrun, S. (2004). Ara*: Anytime a* with provable bounds on sub-optimality. In S. Thrun, L. Saul, & B. Schölkopf (Eds.), *Advances in neural information processing systems* (Vol. 16, pp. 767–774). MIT Press.
- Liu, Y., Song, R., & Bucknall, R. (2015). A practical path planning and navigation algorithm for an unmanned surface vehicle using the fast marching algorithm. In *OCEANS 2015 - Genova* (pp. 1–7). <https://doi.org/10.1109/OCEANS-Genova.2015.7271338>
- Lozano-Pérez, T., & Wesley, M. A. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. *Communication of ACM*, 22(10), 560–570. <https://doi.org/10.1145/359156.359164>.
- Manzini Nicholas, A. (2017). Usv path planning using potential field model. <https://calhoun.nps.edu/handle/10945/56152>
- Mask, J. L. (2011). System identification methodology for a wave adaptive modular unmanned surface vehicle.
- Niu, H., Lu, Y., Savvaris, A., & Tsourdos, A. (2016). Efficient path planning algorithms for unmanned surface vehicle. *IFAC-PapersOnLine*, 49(23), 121–126. <https://doi.org/10.1016/j.ifacol.2016.10.331> (10th IFAC Conference on Control Applications in Marine SystemsCAMS 2016).
- Niu, H., Savvaris, A., Tsourdos, A., & Ji, Z. (2019). Voronoi-visibility roadmap-based path planning algorithm for unmanned surface vehicles. *Journal of Navigation*, 72(4), 850–874. <https://doi.org/10.1017/S0373463318001005>.
- Omar, R., Hailma, C. K. N., & Elia Nadira, S. (2015). Performance comparison of path planning methods. *ARNP Journal of Engineering and Applied Sciences*, 10, 8866–8872 (Omar, R., Hailma, C.K.N., Elia Nadira, S Omar, R., Hailma, C.K.N., Elia Nadira, S).
- Parlangeli, G., & Indiveri, G. (2010). Dubins inspired 2d smooth paths with bounded curvature and curvature derivative. *IFAC Proceedings Volumes*, 43(16), 252–257. <https://doi.org/10.3182/20100906-3-IT-2019.0004> (5. 7th IFAC Symposium on Intelligent Autonomous Vehicles).
- Peralta, F., Arzamendia Lopez, M., Gregor, D., Gutiérrez, D., & Toral, S. (2020). A comparison of local path planning techniques of autonomous surface vehicles for monitoring applications: The ypacarai lake case-study. *Sensors*. <https://doi.org/10.3390/s20051488>.
- Rasekhipour, Y., Khajepour, A., Chen, S. K., & Litkouhi, B. (2017). A potential field-based model predictive path-planning controller for autonomous road vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 18(5), 1255–1267. <https://doi.org/10.1109/TITS.2016.2604240>.
- Reeds, J. A., & Shepp, L. A. (1990). Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, 145(2), 367–393.
- Schoener, M. A. (2019). Global estimation methodology for wave adaptation modular vessel dynamics using a genetic algorithm.
- Sethian, J. A. (1996). A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4), 1591–1595. <https://doi.org/10.1073/pnas.93.4.1591>.
- Sfeir, J., Saad, M., & Saliah-Hassane, H. (2011). An improved artificial potential field approach to real-time mobile robot path planning in an unknown environment. In *2011 IEEE international symposium on robotic and sensors environments (ROSE)* (pp. 208–213). <https://doi.org/10.1109/ROSE.2011.6058518>
- Silva, J. A. R., & Grassi, V. (2018). Clothoid-based global path planning for autonomous vehicles in urban scenarios. In *2018 IEEE international conference on robotics and automation (ICRA)* (pp. 4312–4318). <https://doi.org/10.1109/ICRA.2018.8461201>
- Stentz, A. T. (1995). The focussed d* algorithm for real-time replanning. In *Proceedings of 14th international joint conference on artificial intelligence (IJCAI '95)* (pp. 1652–1659).
- Thompson, D., Coyle, E., & Brown, J. (2019). Efficient lidar-based object segmentation and mapping for maritime environments. *IEEE Journal of Oceanic Engineering*, PP10.1109/JOE.2019.2898762, 1–11. <https://doi.org/10.1109/JOE.2019.2898762>.
- Visvalingam, M., & Whyatt, J. D. (1993). Line generalisation by repeated elimination of points. *The Cartographic Journal*, 30(1), 46–51. <https://doi.org/10.1179/000870493786962263>.
- Wang, N., Sun, J. C., Er, M. J., & Liu, Y. C. (2016). A novel extreme learning control framework of unmanned surface vehicles. *IEEE Transactions on Cybernetics*, 46(5), 1106–1117. <https://doi.org/10.1109/TCYB.2015.2423635>.
- Zhou, L., & Li, W. (2014). Adaptive artificial potential field approach for obstacle avoidance path planning. *2014 Seventh International Symposium on Computational Intelligence and Design*, 2, 429–432. <https://doi.org/10.1109/ISCID.2014.144>.
- Zhu, Z., Schmerling, E., & Pavone, M. (2015). A convex optimization approach to smooth trajectories for motion planning with car-like robots. In *2015 54th IEEE conference on decision and control (CDC)* (pp. 835–842). <https://doi.org/10.1109/CDC.2015.7402333>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.



Eric Coyle received his Ph.D. in Mechanical Engineering from Florida State University in 2010 and his B.S. in Mechanical Engineering from Clemson University in 2006. He is currently a Professor of Mechanical Engineering at Embry-Riddle Aeronautical University (ERAU), where he has been employed since 2011. He is also the Director of the Robotics and Autonomous Systems Lab and a member of the Association for Unmanned Systems International (AUVSI). Dr.

Coyle has served as the lead advisor of the Maritime RobotX Challenge team at ERAU since 2013, which developed the AUVSI Excellence in Technology award-winning USV, Minion, and advises several other robotic student competition teams. Dr. Coyle has lead research efforts funded by ONR, DON, DOA, FDOT, and multiple industry partners. His research focuses on perception, machine learning, autonomy, and unmanned system design and his current research projects also include developing situational awareness for USVs and multi-domain sensor fusion.



Marco Schoener received his B.S. and M.S. degrees in Mechanical Engineering from Embry-Riddle Aeronautical University in 2016 and 2019 respectively. He is currently a Mechanical Engineer working for the Ground Vehicle Systems Center, under the Department of the Army, located in Warren, Michigan. He was a 4-year member of the Embry-Riddle Aeronautical University (ERAU) Maritime RobotX Competition Team, where he was the lead programmer, and a 4-year member of

the ERAU Intelligent Ground Vehicle Competition team, and team lead. His research interests include autonomy, path planning, and human-machine interfaces.



David Thompson received his M.S. degree in Mechanical Engineering in 2017 and his B.S. degree in Aerospace Engineering in 2015 from Embry-Riddle Aeronautical University (ERAU). He is currently a PhD candidate in the Ph.D. in Mechanical Engineering program at ERAU. He has worked as a graduate intern at both APTIV and Leidos Surveillance and Reconnaissance in perception and autonomy related fields. Mr. Thompson is a 6-year member of the Embry-Riddle

Aeronautical University (ERAU) Maritime RobotX Competition Team and former team lead, which developed the AUVSI Xcellence in Technology award-winning USV, Minion. His research interests include computer vision, deep learning, perception, and autonomy.