Fast Path Planning of Autonomous Vehicles in 3D Environments

Jonghoek Kim

Electronic and Electrical Department, Sungkyunkwan University, Suwon 16419, Korea; jonghoek@gmail.com

Abstract: Three dimensional path planner is crucial for the safe navigation of autonomous vehicles (AV), such as unmanned aerial vehicles (UAV) or unmanned underwater vehicles (UUV), which operate in three dimensions. In this paper, we develop a novel 3D path planner, which is fast in generating a near-optimal solution path. The planner generates the 3D path considering the size of an AV so that as the AV traverses the constructed path, it does not collide with an obstacle. This paper introduces a 3D path planner with novel concepts, such as a virtual agent and virtual sensors. In order to generate a 3D path to the goal as fast as possible, we let the virtual agent deploy virtual sensors iteratively, such that the connected sensor network can be formed. The constructed sensor network serves as a topological map for the AV, and we find a shortest path from the start to the goal utilizing the network. The virtual agent’s maneuver is biased towards the goal, in order to find a path to the goal as fast as possible. Moreover, the size of the agent is set considering the safety margin of the generated path. Through MATLAB simulations, we demonstrate the outperformance (low computational load and short path length) of our 3D path planner by comparing it with the 3D RRT-star algorithm.

Keywords: fast path planning; collision avoidance; virtual agent; safe path planning; path length; three dimensional path planning

1. Introduction

Recently, various autonomous vehicles (AV), such as unmanned aerial vehicles (UAV) or unmanned underwater vehicles (UUV), perform many missions, such as environment monitoring, target tracking, and UAV photogrammetry [1–3]. Three dimensional path planning is important for safe navigation of an AV, which operates in three dimensions [2,4–6]. Once the 3D path is planned, the AV follows the path utilizing path control [7–9]. However, how to make the AV follow the planned path is not within the scope of our paper.

Simple 2D path planning algorithms cannot handle complex 3D environments, where there exist a lot of structures constraints and uncertainties [10]. Therefore, 3D path planners for an AV are urgently required, especially in complicated 3D environments such as urban environments.

It is desirable that a path planner searches for a safe path as fast as possible. Therefore, this paper proposes a novel 3D path planner that is fast in generating a sub-optimal path. In other words, our path planner considers both the computational load and the generated path length. Since our path planner runs fast, our planner is suitable for on-line path planning of an autonomous vehicle with cheap embedded systems.

There are many papers on 3D path planner for an AV [10]. A-star planner, Theta-star planner, Dijkstra planner, and Voronoi diagram have been implemented as path planners of various AVs [11–15]. A-star planner achieves better performance than Dijkstra planner by utilizing heuristics for guiding its search towards its goal. A-star, Theta-star, or Dijkstra planners [11–15] require that the entire workspace is already covered by multiple node points (grid cells) completely. A-star, Theta-star, or Dijkstra planners then work to find a cost minimum path to the goal [10]. As one increases the volume of the entire workspace,
the number of node points required to cover the workspace also increases. Therefore, calculating a path utilizing A-star, Theta-star, or Dijkstra planners in a large 3D environment may be infeasible due to large computational load.

Our paper considers a 3D workspace without node points. Considering a 2D workspace without node points, a Voronoi diagram has been utilized as a safe path for an AV [16–21]. However, it is not trivial to generate a 3D path utilizing 3D Voronoi diagrams. Two 3D obstacles generate a 3D Voronoi surface that is equidistant from the two obstacles. However, it is not clear how to generate a 3D path on the 3D Voronoi surface.

Considering a 3D workspace without node points, ref. [22] addressed a potential field based 3D path planner in cluttered environments. The path planners in [22] describes obstacles by simulated annealing neural networks. The computational load of the path planner in [22] increases as one increases the total number of planar surfaces of the obstacle. However, adding spheres and cylinders, that are described with second order equations, increases the computational load of the path planner in [22]. The 3D path planner in our paper does not depend on obstacle shapes, which distinguishes our paper from [22]. In other words, our path planner works for obstacles with arbitrary shapes.

Considering a workspace without node points, RRT planners in [23–25] utilized random sampling to generate a feasible path. RRT planners works for obstacles with arbitrary shapes. RRT planners have been utilized in complex environments due to their ability to search high-dimensional input spaces, in order to navigate among static and dynamic obstacles [26]. In RRT planners, sample points are randomly generated in the obstacle-free space. The authors of [24] showed that the path generated by RRT-star converges to the optimum path asymptotically when infinite time is consumed. But, one cannot wait for unspecified time in practice. In other words, it may take too much time until randomly generated sample points cover the entire obstacle-free space. This inspired us to develop a path planner that is fast in generating a near-optimal path considering a 3D workspace without node points.

This paper considers the problem of generating a safe path to the goal in a 3D cluttered workspace without node points. This paper introduces a 3D path planner with novel concepts, such as a virtual agent and virtual sensors. In order to generate a 3D path to the goal as fast as possible, we let the virtual agent deploy virtual sensors iteratively, such that the connected sensor network can be formed. If there is no obstacle between a generated virtual sensor, say \( n_g \), and the goal, then our path planner ends. The constructed sensor network serves as a topological map for the AV, and we find a shortest path from the start to \( n_g \) utilizing the network. As the AV reaches \( n_g \), it moves directly towards the goal, since there is no obstacle along the line segment connecting \( n_g \) and the goal. This direct maneuver is utilized for decreasing the computational load of generating a path to the goal. As far as we know, this direct maneuver towards the goal was not utilized in other path planners, such as RRT-star or A-star planners.

The proposed path planner is fast in generating a near-optimal path considering a cluttered 3D workspace without node points. The virtual agent’s maneuver is biased towards the goal, in order to find a path to the goal as fast as possible. Moreover, the size of the agent is set considering the safety margin of the generated path.

This paper proves that our path planner is assured to find a near-optimal path to the goal in finite time. Through MATLAB simulations, we demonstrate the outperformance (low computational load and short path length) of our 3D path planner by comparing it with the 3D RRT-star planner in [24].

The organization of this paper is as follows: Section 2 introduces assumptions and definitions. Section 3 addresses a fast path planner in three dimensions. Section 4 shows simulation results of our path planner. Section 5 presents the discussion of our paper. Section 6 provides conclusions.
2. Assumptions and Definitions

Let goal define a 3D point in an open area. We assume that the location of the goal is known to the AV. The problem solved in this paper is as follows: generate a safe 3D path to the goal so that the AV can reach the goal as fast as possible while avoiding collision.

Consider the case where we generate a path for the AV, that can be approximated as a sphere with radius $r$. We utilize one virtual agent $R$ to plan the path of the real AV. $R$ is a sphere with radius $r$ centered at $R$. Here, $R \in \mathbb{R}^3$ is the 3D position of $R$.

We plan the AV’s path so that the virtual agent $R$ with radius $r$ does not collide with obstacles as it moves along the generated path. As we increase the radius $r$, we increase the safety margin of the agent. This shows that as we increase $r$, we plan the path while increasing the AV’s safety.

Note that the virtual agent $R$ is utilized to generate a real AV’s path under simulations. Since $R$ moves in simulated (virtual) environments, we can make it move with unbounded speed in simulations. The term “virtual” is utilized to indicate that the agent exists in simulated environments. Note that as a real AV moves along the constructed path, it cannot move with infinite speed due to hardware limits.

We say that $R$ collides with an obstacle if the sphere centered at $R$ meets an obstacle boundary. We plan the path of $R$ so that $R$ with radius $r$ does not collide with an obstacle while traversing the constructed path.

For convenience, let $L(c_1, c_2)$ define a straight line segment connecting two 3D coordinates given by $c_1 \in \mathbb{R}^3$ and $c_2 \in \mathbb{R}^3$ respectively. We say that $L(c_1, c_2)$ is a collision-free path once the following condition is met: consider the case where $R$ with radius $r$ is located at $c_1$ initially. As $R$ moves along $L(c_1, c_2)$, $R$ does not collide with an obstacle.

In order to generate a 3D path to the goal as fast as possible, $R$ deploys virtual sensors iteratively, such that the connected sensor network can be formed. The constructed sensor network serves as a topological map for the AV, and we find a shortest path from the start to the goal utilizing the network.

Each virtual sensor has spherical sensing coverage with radius $r_s$. Let footprint of a sensor $n$ define the sphere whose center corresponds to $n$ and the radius of which is $r_s$. Let footprint boundary $S_n$ define the boundary of the footprint of $n$. Let $n \in \mathbb{R}^3$ indicates the 3D coordinates of sensor $n$.

Recall that $R$ deploys virtual sensors iteratively, such that the connected sensor network can be formed. Let the connectivity graph $I$ be defined by a set $I = (V(I), E(I))$, where $V(I)$ is the vertex set and $E(I)$ is the edge set. Every vertex in $V(I)$ represents a deployed virtual sensor. Every edge, say $(n_i, n_j) \in E(I)$, indicates that $L(n_i, n_j)$ is a collision-free path and that $\|n_i - n_j\| < r_c$ such that $r_c > r_s$. Here, $n_i, n_j \in \mathbb{R}^3$ indicates the 3D coordinates of sensor $n_i$. In MATLAB simulations (Section 4), we utilize $r_c = 5 \times r_s$. In other words, the connection distance is 5 times longer than the footprint radius $r_s$.

For instance, Figure 1 shows an illustration of $I$. $I$ has five vertices $n_1, n_2, n_3, n_4, n_5$. There is a rectangular obstacle in this figure. Also, the length of $r_c$ is depicted at the bottom of this figure. Each edge in $I$ is plotted with dashed line segments in Figure 1. See that each edge length is shorter than $r_c$ and that each edge is a collision-free path.

![Figure 1](image_url)

**Figure 1.** $I$ has five vertices $n_1, n_2, n_3, n_4, n_5$. There is a rectangular obstacle. Also, the length of $r_c$ is depicted at the bottom of this figure. Each edge in $I$ is plotted with dashed line segments. See that each edge length is shorter than $r_c$ and that each edge is a collision-free path.
A virtual sensor \( n_i \) is adjacent to another sensor, say \( n_j \), in the case where \( L(n_i, n_j) \) is a collision-free path and \( ||n_i - n_j|| < r_c \). Each edge, say \( e \in E(I) \), is weighted by \( w(e): e \rightarrow Z^+ \). The weight of \( \{n_i, n_j\} \) is set as \( ||n_i - n_j|| \).

In simulations, \( R \) can move along any edge in \( E(I) \) infinitely fast. Therefore, we say that \( R \) jumps from one sensor in \( I \) to another sensor in \( I \) instantly.

Utilizing the definition of \( E(I) \), \( R \) with radius \( r \) does not collide with an obstacle as it moves along an edge in \( E(I) \). Hence, \( I \) is a collision-free topological map for the agent \( R \). By accessing the weight for every edge in \( I \), \( R \) can find the shortest path from one sensor to every other sensor.

Let \( g \) define the 3D coordinates of the goal. \( S_n \), the footprint boundary of a sensor \( n \), meets \( L(n, g) \) at one point. This point is called the goal-closest point, since it is closest to the goal among all points on \( S_n \). See Figure 2 for an illustration.

![Figure 2. \( S_n \) meets \( L(n, g) \) at one point, called the goal-closest point.](image)

An open surface of a sensor \( n \) defines the set of points on \( S_n \), such that each point in this set is not inside an obstacle. A frontier of \( n \) is the subset of an open surface, such that every point on the frontier is not inside the footprint of any other sensor.

We next introduce how to generate frontier points that are utilized to discretize a frontier with discrete points. We first generate \( q \) points on the footprint boundary of a sensor \( n \) so that they are evenly spaced on the footprint boundary. As \( q \) increases, we obtain densely spaced points on a footprint boundary.

For instance, we address how to generate \( q = 36 \times 36 \) points on a footprint boundary. We rotate a vector \([r_s, 0, 0]^T\) by \( \phi \in \{\pi/18, 2 \times \pi/18, \ldots, 2 \times \pi\} \) centered at the \( z \)-axis. The matrix representing this rotation is

\[
\mathbf{r}(\phi) = \begin{pmatrix}
\cos(\phi) & -\sin(\phi) & 0 \\
\sin(\phi) & \cos(\phi) & 0 \\
0 & 0 & 1
\end{pmatrix}.
\]

Thereafter, we rotate the vector \( \mathbf{r}(\phi) \times [r_s, 0, 0]^T \) by \( \theta \in \{\pi/18, 2 \times \pi/18, \ldots, 2 \times \pi\} \) centered at the \( y \)-axis. The matrix representing this rotation is

\[
\mathbf{r}(\theta) = \begin{pmatrix}
\cos(\theta) & 0 & \sin(\theta) \\
0 & 1 & 0 \\
-\sin(\theta) & 0 & \cos(\theta)
\end{pmatrix}.
\]

The resulting vector after two rotations is \( f \mathbf{p}(\phi, \theta) = \mathbf{r}(\theta) \times \mathbf{r}(\phi) \times [r_s, 0, 0]^T \). The vector \( f \mathbf{p}(\phi, \theta) \) represents the relative position of a point on a footprint boundary with respect to \( n \) (3D coordinates of sensor \( n \)). Since \( \phi \) and \( \theta \) are chosen from 36 values respectively, \( q = 36 \times 36 \) points exist on a footprint boundary. By adding \( n \) to \( f \mathbf{p}(\phi, \theta) \), we generate the 3D coordinates of the \( j \)-th frontier point \((j \in \{1, 2, \ldots, q\}) \) of \( n \).

The goal-closest point and these \( q \) points are termed the candidate frontier points of \( n \). Among these \( q + 1 \) candidate frontier points, we select a set of frontier points, \( f(n) \), on a frontier satisfying the following conditions:

1. A point in \( f(n) \) is not inside the footprint of any other sensor.
2. The line segment connecting \( n \) and a point in \( f(n) \) is a collision-free path for the agent \( R \).
3. The distance between every point in \( f(n) \) and an obstacle boundary is larger than the agent radius \( r \).

In our path plan strategy, \( R \) utilizes a frontier point as a “way-point” for reaching the goal. The first condition shows that a frontier point corresponds to the border between a space covered by footprints and a space covered by no footprints. If every sensor has no frontier point, then footprints of all sensors cover the entire workspace. The second condition indicates that as \( R \) moves from \( n \) to any point in \( f(n) \), \( R \) does not collide with an obstacle boundary. The third condition is required to avoid the case where \( R \) collides with an obstacle when \( R \) arrives at a frontier point. The second and third conditions assure that a frontier point is not inside an obstacle. See Algorithm 1 for generation of both candidate frontier points and frontier points.

Figure 3 illustrates \( R \) deploying a new virtual sensor. In this figure, \( R \) is depicted as a sphere. \( R \) moves inside a tunnel in this figure. The tunnel boundaries are depicted with red curves, and the trajectory of \( R \) is depicted with a yellow curve. The large dots on the trajectory of \( R \) indicate the virtual sensors deployed by \( R \). The footprint boundary of each virtual sensor is depicted as a dotted sphere. Frontier points are marked as large dots on the footprint boundary of one sensor.

![Figure 3. R deploying a new virtual sensor. Frontier points are marked as large dots on the footprint boundary of one sensor.](image)

**Algorithm 1 Path Plan Strategy**

1. `initPhase = TRUE;`
2. Generate a virtual agent \( R \) at the position of a real AV;
3. `repeat`
   4. \( R \) deploys a new virtual sensor, say \( n \), at its position;
   5. if `initPhase == TRUE` then
      6. Let \( n_{init} \) define a virtual sensor deployed at the real AV position;
      7. `initPhase = FALSE;`
   8. end if
   9. Update \( I \) utilizing the newly deployed sensor \( n \);
10. Enable the sensing capability of \( n \);
11. `FrontierGenerate(n)` in Algorithm 2;
12. Every frontier point within \( r_s \) distance from \( n \) is removed;
13. `R.FrontierJump` in Algorithm 3;
14. `until \( L(n_g, g) \)` is a collision-free path;
15. Set \( n \) as \( n_{g} \), representing the sensor nearest to the goal;
16. Generate a shortest path from \( n_{init} \) to \( n_g \) utilizing \( I \);
17. The real AV moves along the generated shortest path;
18. The real AV moves along \( L(n_g, g) \) until reaching \( g \);

3. Fast Path Planner

A time-efficient 3D path planner utilizing one virtual agent \( R \) is addressed in Algorithm 1. We address Algorithm 1 in detail. Algorithms 2 and 3 are sub-algorithms of Algorithm 1.

At the initial phase of Algorithm 1, we generate a virtual agent \( R \) at the position of a real AV. \( R \) deploys a virtual sensor at its position. Let \( n_{init} \) define a virtual sensor deployed at the position of a real AV.
Whenever $R$ deploys a new virtual sensor, the sensor is connected to the network and we update $I$. Furthermore, sensing capability of the sensor is enabled. Recall that each virtual sensor has footprint radius $r_s$.

Algorithm 2 is utilized to generate candidate frontier points. Furthermore, Algorithm 3 is utilized to make $R$ jump to a frontier sensor, say $n_c$, which is closest to the goal $g$. Here, we say that a sensor is a frontier sensor if the sensor has a frontier point. Here, $R$ jumps to a frontier sensor, which is closest to $g$, since we want to find a shortest path to $g$.

**Algorithm 2** FrontierGenerate($n$)

1. Generate the goal-closest point of $n$ and set the goal-closest point as the first candidate frontier point;
2. Generate $q$ candidate frontier points on $S_n$;
3. for $i = 1:1:q + 1$ do
4. if $i$-th candidate frontier point satisfies the conditions for a frontier point then
5. Store the candidate frontier point as a frontier point of $n$;
6. end if
7. end for

**Algorithm 3** R.FrontierJump

1. $R$ finds a frontier sensor, say $n_c$, that is closest to the goal $g$;
2. $R$ jumps to $n_c$;
3. if the virtual sensor $n_c$ has the goal-closest point as a frontier point then
4. $R$ sets the goal-closest point as $f_R$;
5. else
6. $R$ randomly selects one frontier point, which is not the goal-closest point, as $f_R$;
7. end if
8. $R$ jumps to $f_R$;

In Algorithm 3, $n_c$ is a frontier sensor, that is closest to the goal $g$. In Algorithm 3, $f_R$ is a frontier point of $n_c$, that will be visited by $R$ after $R$ jumps to $n_c$. If $n_c$ has the goal-closest point as a frontier point, then $R$ sets the goal-closest point as $f_R$. Otherwise, $R$ randomly selects one frontier point, which is not the goal-closest point, as $f_R$. Due to this random selection, our approach is based on random sampling. Thereafter, $R$ jumps to $f_R$.

In this way, $R$ visits the goal-closest point of $n_c$ before visiting other frontier points of $n_c$. This makes $R$ head towards the goal if possible. This movement is utilized for minimizing the path length to the goal. In this way, the virtual agent’s maneuver is biased towards the goal, in order to find a path to the goal as fast as possible. Our approach is similar to A-star planners [12], since A-star planners utilize heuristics for guiding its search towards the goal.

Once $R$ reaches $f_R$, it deploys a new sensor at $f_R$. Whenever a new sensor is deployed, we enable the sensing capability of the sensor. Thereafter, every frontier point within $r_s$ distance from the sensor is removed.

Algorithm 1 iterates until the newly deployed sensor $n$ satisfies that $L(n,g)$ is a collision-free path. The sensor $n$ is set as $n_{init}$, representing the sensor nearest to the goal. We then generate a shortest path from $n_{init}$ to $n_g$ utilizing $I$. Then, the real AV moves along the generated shortest path. After the AV moves along the generated path, it moves along $L(n_g,g)$ until reaching $g$. As far as we know, this direct maneuver towards the goal was not utilized in other path planners, such as RRT-star or A-star planners.

Algorithm 1 cannot proceed if $R$ cannot find any frontier sensor. The following theorem proves that if $R$ cannot find any frontier sensor, then the entire open space is covered by footprints.

**Theorem 1.** If $R$ cannot find any frontier sensor, then the entire open space is covered by footprints.
Proof. Under the transposition rule, we prove the following statement: if an open space, which has not been covered by footprints, exists, then R can find a frontierSensor.

Suppose that an open space, which has not been covered by footprints, exists. Let O indicate this uncovered open space. Utilizing the definition of a frontier, at least one sensor has a frontier on the boundary of O. Therefore, R can find this frontierSensor.

Theorem 1 shows that if R cannot find any frontierSensor, then the goal must be covered by a footprint of a sensor. Therefore, the path to the goal is constructed before R cannot find any frontierSensor. In this way, Algorithm 1 proceeds until a path to the goal is found.

Theorem 2 proves that the real AV does not collide with an obstacle while moving along the path constructed utilizing Algorithm 1. This shows that Algorithm 1 generates a safe path for the real AV.

Theorem 2. As the real AV moves along the path constructed utilizing Algorithm 1, it does not collide with an obstacle.

Proof. The path to ng is constructed utilizing I. Under the definition of I, every edge, say \{n_1, n_2\} ∈ E(I), indicates that L(n_1, n_2) is a collision-free path. Since the real AV moves along edges of I until meeting ng, the real AV does not collide with an obstacle during the maneuver. Moreover, utilizing the definition of ng, L(ng, g) is a collision-free path. Thus, as the real AV moves along L(ng, g), it does not collide with an obstacle. We proved this theorem.

Theorem 1 shows that if R cannot find any frontierSensor, then the goal must be covered by a footprint of a sensor. Hence, Algorithm 1 continues until a path to the goal is found.

Handle the Case Where a Narrow Passage Exists

In practice, frontier points are generated to discretize a frontier with discrete points. At each sensor n, we generate q points on the footprint boundary of n, so that they are evenly spaced on the footprint boundary. As q increases, we obtain densely spaced points on a footprint boundary.

If q is not sufficiently large, then we may have a case where a frontier point is not generated on the boundary of O (uncovered open space). This case may happen when R is at the entrance of a narrow passage (tunnel) in the workspace.

For instance, let S define the entrance cross area of a narrow tunnel. The area of a footprint boundary is 4πr_s^2. Since q points are evenly generated on a footprint boundary, each point has its associated area as \(\frac{4\pi r_s^2}{q}\). In order to generate a frontier point on S, it is required that \(\frac{4\pi r_s^2}{q} < S\). Thus, we require that

\[ q > \frac{4\pi r_s^2}{S} \]  

(3)

Equation (3) implies that as the footprint radius r_s decreases, we have more probability to make the AV move through a narrow tunnel. (3) further shows that as q increases, we have more probability to make the AV move through a narrow tunnel. If q is sufficiently large, then we can assure of finding a path to the goal utilizing Theorem 1.

If we can access the minimum entrance cross area S in the environment, then we can select q that satisfies (3). In this way, our path planner can handle narrow tunnels.

4. Simulation Results

In this section, we demonstrate the outperformance of our path planner (Algorithm 1) through MATLAB simulations. We utilize \(q = 36 \times 36\), \(r_s = 20\), and \(r_c = 5 \times r_s\). The radius
of the AV is \( r = 2 \) distance units. The goal position is \([170, 170, 170]\). The initial position of the AV is \([30, 30, 30]\).

To demonstrate the performance of Algorithm 1, we compare Algorithm 1 with the 3D RRT-star planner in [24]. We implemented 20 Monte-Carlo (MC) simulations to demonstrate the performance of our planner method rigorously. Let \( L^t \) where \( t \leq 20 \) define the constructed path length utilizing the \( t \)-th MC simulation.

4.1. Cluttered Environments

We present the MATLAB simulation results of the proposed path planner (Algorithm 1). Using MATLAB 2016, we spend 61 s to run 20 MC simulations under Algorithm 1. The mean of \([L^1, L^2, \ldots, L^{20}]\) is 268. Furthermore, the variance of \([L^1, L^2, \ldots, L^{20}]\) is 95.

Figure 4 depicts the path constructed utilizing one MC simulation of Algorithm 1. The obstacles are shown as spheres, and the generated path is depicted with red line segments. In Algorithm 1, the AV moves toward the goal directly if possible. This direct movement is depicted in Figure 4. Since the radius of the virtual agent is \( r \) distance units in the simulations, the constructed path is a safe path for the AV that can be approximated as a sphere with radius \( r \) distance units.

Comparison with the 3D RRT-Star Planner

We compare the performance of the proposed planner with the 3D RRT-star planner in [24]. The 3D RRT-star planner ends when the distance between a sample point and the goal position is shorter than 10 distance units. Two sample points are neighbors if the relative distance between them is shorter than 10 distance units. The step size (expansion of the tree within one time step) in the 3D RRT-star planner is set as 10 distance units.

Once a path is generated, we can further improve the generated path by smoothing it or removing vertices on the path. The Open Motion Planning Library (OMPL) library has a path simplifier and a B-Spline smoother for these tasks. See the following website for the OMPL library: https://ompl.kavrakilab.org/index.html, accessed on 10 April 2022.

For fair comparison with the 3D RRT-star planner, we do not apply a path simplifier or a B-Spline smoother.

Using MATLAB 2016, one spends 644 s to run 20 MC simulations under the 3D RRT-star planner. Under the 3D RRT-star planner, the mean of \([L^1, L^2, \ldots, L^{20}]\) is 384. Furthermore, the variance of \([L^1, L^2, \ldots, L^{20}]\) is 1099.

See that Algorithm 1 runs much faster than the RRT-star planner. Also, Algorithm 1 generates a shorter path than the 3D RRT-star planner. This shows that Algorithm 1 outperforms the 3D RRT-star planner.

Figure 5 depicts the path constructed utilizing one MC simulation of the 3D RRT-star planner. The obstacles are shown as spheres. The generated path is depicted with red line segments.
Figure 5. The path constructed utilizing one MC simulation of the 3D RRT-star planner. The obstacles are shown as spheres. The generated path is depicted with red line segments.

4.2. Highly Cluttered Environments

We next consider highly cluttered environments. We present the simulation results of the proposed path planner (Algorithm 1). Using MATLAB 2016, we spend 113 s to run 20 MC simulations under Algorithm 1. The mean of $[L_1, L_2, \ldots, L_{20}]$ is 280. Furthermore, the variance of $[L_1, L_2, \ldots, L_{20}]$ is 727.

Considering highly cluttered environments, Figure 6 depicts the path constructed utilizing one MC simulation of Algorithm 1. The obstacles are shown as spheres, and the generated path is depicted with red line segments. In Algorithm 1, the AV moves toward the goal directly if possible. This direct movement is depicted in Figure 6.

Figure 6. Highly cluttered environments. The path constructed utilizing one MC simulation of Algorithm 1. The obstacles are shown as spheres. The generated path is depicted with red line segments.

Comparison with the 3D RRT-Star Planner

We next show the simulation results of the 3D RRT-star planner in highly cluttered environments. Using MATLAB 2016, we spend 627 s to run 20 MC simulations under the 3D RRT-star planner. See that Algorithm 1 runs much faster than the RRT-star planner.

Under the 3D RRT-star planner, the results are as follows. The mean of $[L_1, L_2, \ldots, L_{20}]$ is 377. Furthermore, the variance of $[L_1, L_2, \ldots, L_{20}]$ is 842. See that Algorithm 1 generates a shorter path than the 3D RRT-star planner. This shows that Algorithm 1 outperforms the 3D RRT-star planner.

Figure 7 depicts the path constructed utilizing one MC simulation of the 3D RRT-star planner. The obstacles are shown as spheres. The generated path is depicted with red line segments.
Next, Table 1 shows the effect of changing the footprint radius $r_s$. We run the proposed path planner (Algorithm 1) in highly cluttered environments (Figure 6).

In Table 1, $simT$ (sec) indicates the simulation time of running 20 MC simulations utilizing MATLAB. $MeanL$ defines the mean of $[L_1, L_2, \ldots, L_{20}]$. Also, $VarL$ defines the variance of $[L_1, L_2, \ldots, L_{20}]$. As $r_s$ decreases, the computational load $simT$ increases. As $r_s$ decreases, the variance in the generated path length decreases. However, decreasing $r_s$ slightly reduces the average path length. Table 1 shows that our path planner outperforms the 3D RRT-star, regardless of $r_s$.

Table 1. The effect of changing the footprint radius $r_s$.

| $r_s$ | $simT$  | $MeanL$ | $VarL$  |
|-------|---------|---------|---------|
| 40    | 53      | 295     | 1536    |
| 30    | 86      | 292     | 956     |
| 20    | 113     | 280     | 727     |

5. Discussion

The proposed path planner is fast in generating a near-optimal path considering a cluttered 3D workspace. We prove that our fast path planner is assured to find a near-optimal path to the goal in finite time. Since our path planner runs fast, our planner is suitable for on-line path planning of an autonomous vehicle with cheap embedded systems.

Once the path is set, the AV follows the path utilizing path following controls. Many papers [7–9] addressed 3D path following controls for AVs. How to make the AV follow the given path is not within the scope of our paper.

We acknowledge that the proposed planner does not generate an optimal path, considering path length. However, once a safe and near-optimal path is generated utilizing the proposed fast planner, we can further improve the generated path by smoothing it or removing vertices on the path. The Open Motion Planning Library (OMPL) library has a path simplifier and a B-Spline smoother for these tasks. See the following website for the OMPL library: https://ompl.kavrakilab.org/index.html, accessed on 10 April 2022. The authors of [27] reviewed various path smoothing techniques for navigation of an AV. How to smooth the generated path is not within the scope of our paper.

6. Conclusions

This paper introduces a 3D path planner utilizing a virtual agent and virtual sensors. The proposed planner generates a near-optimal and safe path in a time efficient manner. We prove that our planner is guaranteed to find a safe path to the goal in finite time. Through
MATLAB simulations, we demonstrate the outperformance (low computational load and short path length) of our 3D path planner by comparing it with the 3D RRT-star planner.

In our future works, we will demonstrate the outperformance of our 3D path planner utilizing experiments with real AVs. Furthermore, we will extend our path planner to a path planner utilizing multiple virtual agents. Since path planning is done in a parallel manner utilizing multiple virtual agents, we can improve the time efficiency considerably compared to the case where only one virtual agent is utilized.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Dominici, D.; Alicandro, M.; Massimi, V. UAV photogrammetry in the post-earthquake scenario: Case studies in L’Aquila. Geomat. Nat. Hazards Risk 2017, 8, 87–103. [CrossRef]
2. Kim, J. Three-dimensional multi-robot control to chase a target while not being observed. Int. J. Adv. Robot. Syst. 2019, 16, 1–11. [CrossRef]
3. Kim, J. Three dimensional tracking of a maneuvering emitter utilizing doppler-bearing measurements of a constant velocity observer. Signal Process. 2021, 189, 108246. [CrossRef]
4. Pereira, A.A.; Binney, J.; Jones, B.H.; Ragan, M.; Sukhatme, G.S. Toward risk aware mission planning for Autonomous Underwater Vehicles. In Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, San Francisco, CA, USA, 25–30 September 2011; pp. 3147–3153. [CrossRef]
5. Lefèbvre, N.; Schjolberg, I.; Utne, I.B. Integration of risk in hierarchical path planning of underwater vehicles. IFAC-PapersOnLine 2016, 49, 226–231. [CrossRef]
6. Petres, C.; Pailhas, Y.; Patron, P.; Petillot, Y.; Evans, J.; Lane, D. Path Planning for Autonomous Underwater Vehicles. IEEE Trans. Robot. 2007, 23, 331–341. [CrossRef]
7. Yao, X.; Wang, X.; Wang, F.; Zhang, L. Path Following Based on Waypoints and Real-Time Obstacle Avoidance Control of an Autonomous Underwater Vehicle. Sensors 2020, 20, 795. [CrossRef] [PubMed]
8. Caharija, W.; Pettersen, K.Y.; Bibuli, M.; Calado, P.; Zerik, E.; Braga, J.; Gravdahl, J.T.; Serensen, A.J.; Milovanović, M.; Bruzzone, G. Integral Line-of-Sight Guidance and Control of Underactuated Marine Vehicles: Theory, Simulations, and Experiments. IEEE Trans. Control Syst. Technol. 2016, 24, 1623–1642. [CrossRef]
9. Rubi, B.; Pérez, R.; Morcego, B. A Survey of Path Following Control Strategies for UAVs Focused on Quadrotors. J. Intell. Robot. Syst. 2020, 98, 241–265. [CrossRef]
10. Yang, L.; Qi, J.; Xiao, J.; Yong, X. A literature review of UAV 3D path planning. In Proceedings of the 11th World Congress on Intelligent Control and Automation, Shenyang, China, 29 June–4 July 2014; pp. 2376–2381.
11. Muñoz, P.; Rodríguez-Moreno, M. Improving efficiency in any-angle path-planning algorithms. In Proceedings of the 2012 6th IEEE International Conference Intelligent Systems, Sofia, Bulgaria, 6–8 September 2012; pp. 213–218. [CrossRef]
12. Choset, H.; Lynch, K.M.; Hutchinson, S.; Kantor, G.A.; Burgard, W.; Kavraki, L.E.; Thrun, S. Principles of Robot Motion; MIT Press: Cambridge, MA, USA, 2005.
13. Kim, J.; Kim, S.; Choo, Y. Stealth Path Planning for a High Speed Torpedo-Shaped Autonomous Underwater Vehicle to Approach a Target Ship. Cyber Phys. Syst. 2018, 4, 1–16. [CrossRef]
14. Pettie, S. A new approach to all-pairs shortest paths on real-weighted graphs. Theor. Comput. Sci. 2004, 312, 47–74. [CrossRef]
15. Nash, A.; Daniel, K.; Koenig, S.; Felner, A. Theta*: Any-Angle Path Planning on Grids. In Proceedings of the AAAI Conference on Artificial Intelligence, Hyderabad, India, 6–12 January 2007; pp. 1177–1183.
16. Kim, J.; Zhang, F.; Egerstedt, M. A provably complete exploration strategy by constructing Voronoi Diagrams. Auton. Robot. 2010, 29, 367–380. [CrossRef]
17. Choset, H.; Walker, S.; Eiamsa-Ard, K.; Burdick, J. Sensor-Based Exploration: Incremental Construction of the Hierarchical Generalized Voronoi Graph. Int. J. Robot. Res. 2000, 19, 126–148. [CrossRef]
18. Nagatani, K.; Choset, H. Toward robust sensor based exploration by constructing reduced generalized Voronoi graph. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Kyongju, Korea, 17–21 October 1999; pp. 1687–1692.
19. Choset, H.; Nagatani, K. Topological simultaneous localization and mapping (SLAM): Toward exact localization without explicit localization. IEEE Trans. Robot. Auton. 2001, 17, 125–137. [CrossRef]
20. Kim, J. Cooperative Exploration and Protection of a Workspace Assisted by Information Networks. *Ann. Math. Artif. Intell.* 2014, 70, 203–220. [CrossRef]

21. Kim, J. Capturing intruders based on Voronoi diagrams assisted by information networks. *Int. J. Adv. Robot. Syst.* 2017, 14, 1–8. [CrossRef]

22. Kroumov, V.; Yu, J. 3D path planning for mobile robots using annealing neural network. In Proceedings of the 2009 International Conference on Networking, Sensing and Control, Okayama, Japan, 26–29 March 2009; pp. 130–135.

23. Kuffner, J.; LaValle, S.M. Space-Filling Trees: A New Perspective on Incremental Search for Motion Planning. In Proceedings of the IEEE International Conference on Intelligent Robots and Systems, San Francisco, CA, USA, 25–30 September 2011; pp. 2199–2206.

24. Karaman, S.; Frazzoli, E. Sampling-based algorithms for optimal motion planning. *Int. J. Robot. Res.* 2011, 30, 846–894. [CrossRef]

25. Urman, C.; Simmons, R. Approaches for heuristically biasing RRT growth. In Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS 2003), Las Vegas, NV, USA, 27–31 October 2003; pp. 1178–1183.

26. Yershova, A.; LaValle, S.M. Improving motion planning algorithms by efficient nearest-neighbor searching. *IEEE Trans. Robot.* 2007, 23, 151–157. [CrossRef]

27. Ravankar, A.; Ravankar, A.A.; Kobayashi, Y.; Hoshino, Y.; Peng, C.C. Path Smoothing Techniques in Robot Navigation: State-of-the-Art, Current and Future Challenges. *Sensors* 2018, 18, 3170. [CrossRef] [PubMed]