3-D Trajectory Planning of Aerial Vehicles Using RRT*

P. Pharpatara, B. Hérissé, and Y. Bestaoui

***Abstract*— This brief presents a trajectory planning algorithm for aerial vehicles traveling in 3-D space while avoiding obstacles. The nature of the obstacles can be, for example, radar detection areas, cooperating and non-cooperating vehicles, and so on. Thus, it is a complex trajectory planning problem. The proposed planner is based on the optimal rapidly exploring random tree (RRT*) algorithm. Artificial potential fields are combined with the RRT* algorithm to accelerate the convergence speed to a suboptimal solution by biasing the random state generation. The performance of this framework is demonstrated on a complex missile application in a heterogeneous environment. Indeed, since the air density decreases exponentially with altitude, the maneuverability of the aerial vehicle depending on aerodynamic forces also decreases exponentially with altitude. To face this problem, the shortest paths of Dubins-like vehicles traveling in a heterogeneous environment are used to build the metric. In the simulation results, this framework can find the first solution with fewer iterations than the RRT and the RRT* algorithm. Moreover, the final solution obtained within a given number of iterations is closer to an optimal solution regarding the considered criterion.**

***Index Terms*— Aerial robotics, complex environment, optimal control, optimal rapidly exploring random tree (RRT*) algorithm, path planning algorithm.**

I. Introduction

THE DEVELOPMENT of autonomous aerial vehicles, such as drones, missiles, or unmanned combat aerial vehicles together with advances in technology, has led to increasingly complex autonomous tasks. In particular, advances in embedded computing have allowed aerial vehicles to perform complex trajectory planning algorithms on board the vehicle. These planning algorithms have to face not only static and dynamic obstacles, for example, radar detection areas and cooperating and non-cooperating vehicles, but also the changes of an objective task during the mission. Consequently, such embedded algorithms that can be very demanding regarding numerical computations also need to consider real-time constraints. Then, it is of significance to solve a trajectory planning problem with a good computational efficiency.

A trajectory planning problem can be considered as an optimal control problem leading to a constrained two-point boundary value problem that can be solved using direct or indirect methods [1]. However, solving such a numerical problem can be challenging, especially when obstacles inducing state constraints are considered. Then, other methods have often been preferred, even recently. For example, closed-loop guidance laws [2] relying on a simplified model of the system can be used to choose waypoints along the path in order to avoid obstacles. However, the performance of such an algorithm remains limited. In particular, global optimality with respect to a given criterion cannot be easily achieved. Model predictive control techniques are also widely used [3] by combining open-loop optimal control with feedback control. These techniques provide a good tradeoff between the computational effort and the global performance of the solution. In the robotic field, many trajectory planning methods have been studied. Cell decomposition methods, such as D* [4], and potential field methods [5], [6] are used to find a trajectory for an aerial vehicle. However, the complexity of the considered system model is often limited using such techniques. Sampling-based path planning methods, such as probabilistic roadmap methods [7] or rapidly exploring random trees (RRTs) [8], have been proposed to find collision-free trajectories in complex environments. These methods are often used for path planning of nonholonomic vehicles in environments cluttered by obstacles [9]. It is demonstrated in [10] that this method can be applied to trajectory planning for a two-stage interceptor missile. However, using only the RRT algorithm, an optimal solution cannot be obtained efficiently regarding a specified criterion. Recently, a new algorithm called optimal rapidly exploring random tree (RRT*) [11] has been developed to overcome the optimality problem. Moreover, since it is an incremental algorithm that keeps searching for better solutions, it is ideally suited for real-time applications that need a valid result on-demand. This algorithm has also been extended to systems with differential constraints, such as the nonholonomic constraints [12]. Thus, the asymptotic optimality of the RRT* can be ensured for aerial vehicles, such as airplanes or missiles.

In this brief, a general framework for the trajectory planning algorithm of aerial vehicles traveling in a 3-D space while avoiding obstacles is proposed based on the RRT* algorithm in [11]. In this framework, nonholonomic constraints of such vehicles are considered and artificial potential fields (APFs) [13], [14] are integrated to bias the random

Manuscript received November 17, 2015; revised April 6, 2016; accepted June 12, 2016. Date of publication July 7, 2016; date of current version April 11, 2017. Manuscript received in final form June 15, 2016. Recommended by Associate Editor L. Marconi.

P. Pharpatara was with ONERA–The French Aerospace Lab, Palaiseau F-91761, France. He is now with The Geo-Informatics and Space Technology Development Agency (Public Organization), Bangkok 10210, Thailand (e-mail: pawit@gistda.or.th).

B. Hérissé is with ONERA–The French Aerospace Lab, Palaiseau F-91761, France (e-mail: bruno.herisse@onera.fr).

Y. Bestaoui is with the Laboratoire Informatique, Biologie Intégrative et Systèmes Complexes, Université d'Evry-Val-d'Essonne, Evry 91020, France (e-mail: yasmina.bestaoui@ufrst.univ-evry.fr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCST.2016.2582144

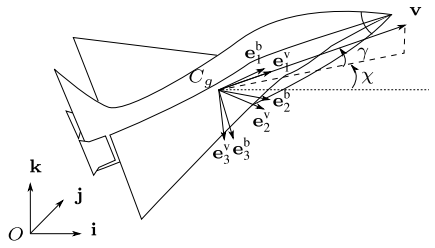

Fig. 1. Vehicle model.

sampling to accelerate the convergence speed to both the first solution and the asymptotic optimal solution. Indeed, it is fascinating to obtain the first trajectory quickly if the objective is to make the algorithm run in real time. Moreover, the approach is designed so that it can easily be combined with other probabilistic methods aiming at improving convergence speed, such as the informed-RRT* [15] or the RRT*-SMART [16] for example. Note that there are very few works that use similar strategies (see [17] for an example) and they often address holonomic problems only. Convergence assumptions discussed in [11] to ensure asymptotic convergence are recalled in the context of nonholonomic systems.

To illustrate the approach, a missile application is presented. Since the maneuverability of such a vehicle decreases with altitude [18], making it hard to predict the trajectory, it is a challenging problem. It extends the work in [18] which solves the 2-D problem for this specific application. Simulation results demonstrate that the algorithm can efficiently find a feasible near-optimal trajectory.

This paper is organized as follows. First, environment, system modeling, and problem formulation are presented in Section II. Then, the RRT* path planner is introduced in Section III along with the integration of the APF in the algorithm. Next, an application of the RRT* framework is described in Section IV as an example. Then, the simulation results are shown, compared with the existing algorithms, and analyzed in Section V. Finally, some concluding remarks are made in Section VI.

II. Problem Statement

A. System Modeling

In this brief, the aerial vehicle is modeled as a rigid body of mass m. Three frames (Fig. 1) are introduced to describe the motion of the vehicle: an earth-centered earth-fixed reference frame $\mathcal{I}$ centered at point O and associated with the basis vectors $(\mathbf{i}, \mathbf{j}, \mathbf{k})$; a body-fixed frame $\mathcal{B}$ attached to the vehicle at its center of mass C_g with the basis vector $(\mathbf{e}_1^b, \mathbf{e}_2^b, \mathbf{e}_3^b)$; and a velocity frame $\mathcal{V}$ attached to the vehicle at C_g with the basis vector $(\mathbf{e}_1^v, \mathbf{e}_2^v, \mathbf{e}_3^v)$, where the translational velocity of the vehicle is denoted $\mathbf{v} = v\mathbf{e}_1^v$ and v is the speed of the vehicle. Position and velocity defined in $\mathcal{I}$ are denoted $\boldsymbol{\xi} = (x, y, z)^\top \in \mathbb{R}^3$ and $\mathbf{v} = (\dot{x}, \dot{y}, \dot{z})^\top \in \mathbb{R}^3$. Denote γ and χ the orientation with respect to $\mathcal{I}$ of the velocity $\mathbf{v}$, where γ is the flight path angle and χ is the azimuth angle.

The dynamics of the aerial vehicle can be written as

$$\begin{cases} \dot{x} = v\cos\gamma\,\cos\chi \\ \dot{y} = v\cos\gamma\,\sin\chi \\ \dot{z} = v\sin\gamma \\ \dot{v} = -g\sin\gamma + \dfrac{F_{a_1}}{m} + \dfrac{F_{p_1}}{m} + \dfrac{T_1}{m} \\ \dot{\gamma} = -\dfrac{g}{v}\cos\gamma + \dfrac{F_{a_3}}{mv} + \dfrac{F_{p_3}}{mv} + \dfrac{T_3}{mv} \\ \dot{\chi} = \dfrac{F_{a_2}}{mv\cos\gamma} + \dfrac{F_{p_2}}{mv\cos\gamma} + \dfrac{T_2}{mv\cos\gamma} \end{cases} \tag{1}$$

where $\gamma \in [-\pi/2, \pi/2]$ and $\chi \in [-\pi, \pi]$. T_1, T_2, and T_3 are the components of the propulsion, F_{a_1}, F_{a_2}, and F_{a_3} are the components of the aerodynamic forces, including the drag force $\mathbf{f}_\mathrm{D}$ and the lift forces $\mathbf{f}_{L_2}$ and $\mathbf{f}_{L_3}$ expressed as follows:

$$\mathbf{f}_D = -\frac{1}{2}\rho(z)SC_D v_a \mathbf{v}_a$$
$$\mathbf{f}_{L_2} = \frac{1}{2}\rho(z)SC_{L_2} v_a^2 \mathbf{e}_2^{v_a}$$
$$\mathbf{f}_{L_3} = -\frac{1}{2}\rho(z)SC_{L_3} v_a^2 \mathbf{e}_3^{v_a}$$

where C_D, C_{L_2}, and C_{L_3} are the aerodynamic coefficients, S is the surface of reference, $\rho(z)$ is the density of air, and v_a is the absolute value of the air velocity $\mathbf{v}_a$ expressed as $\mathbf{v}_a = \mathbf{v} - \mathbf{v}_w$, where $\mathbf{v}_w$ is the wind velocity. $\mathbf{e}_2^{v_a}$ and $\mathbf{e}_3^{v_a}$ are the two unit vectors perpendicular to $\mathbf{v}_a$. Note that if $\mathbf{v}_w = \mathbf{0}$, $\mathbf{e}_2^{v_a} = \mathbf{e}_2^v$ and $\mathbf{e}_3^{v_a} = \mathbf{e}_3^v$. The forces F_{p_1}, F_{p_2}, and F_{p_3} include other perturbation forces acting on the aircraft.

For the control of the vehicle, the aerodynamic forces $\mathbf{f}_{L_2}$ and $\mathbf{f}_{L_3}$ as well as the propulsion forces (T_1, T_2, T_3) are used.

For the environment modeling, the U.S. Standard Atmosphere, 1976 (U.S.-76) can be used. In the lower earth atmosphere (altitude $<$ 35 km), the density of air ρ and atmospheric pressure decreases exponentially with altitude and approach zero around 35 km. As we consider an unmanned aerial vehicle with aerodynamic flight controls, the maneuvering capabilities are linked to the density of air and approach zero at 35 km.

B. Problem Formulation

Let $\mathbf{x}(t) = (\boldsymbol{\xi}^\top, v, \gamma, \chi)^\top \in \mathbb{X} = \mathbb{R}^6$ be the measurable state of the system and $\mathbf{u} \in \mathbb{U}$ be a control input in the set $\mathbb{U}$ of admissible controls including aerodynamic and propulsive forces as described in Section II-A. Then, the differential system (1) can be rewritten as

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \tag{2}$$

where $\mathbf{f}$ is the vehicle system model.

$\mathbb{X} = \mathbb{R}^6$ is the state space. It is divided into two subsets. Let $\mathbb{X}_\mathrm{free}$ be the set of admissible states. $\mathbb{X}_\mathrm{obs} = \mathbb{X} \setminus \mathbb{X}_\mathrm{free}$ is defined as the obstacle region. The initial state of the system is $\mathbf{x}_\mathrm{init} \in \mathbb{X}_\mathrm{free}$.

The path planning algorithm is given a rendezvous set $\mathbb{X}_\mathrm{goal} \subset \mathbb{X}_\mathrm{free}$. In order to achieve its mission, the vehicle

has to reach $\mathbb{X}_{\text{goal}}$ while avoiding obstacles and minimizing a performance criterion J defined as

$$J(t_0, t_f, \mathbf{u}) = \int_{t_0}^{t_f} f^0(\mathbf{x}(t), \mathbf{u}(t))\, dt + k(\mathbf{x}(t_f)) \tag{3}$$

where $f^0 : \mathbb{R}^6 \times \mathbb{R}^3 \to \mathbb{R}$ and $k : \mathbb{R}^6 \to \mathbb{R}$ are C^1 [1]. Note that if the minimal time problem is considered, $f^0 = 1$ and $k = 0$ are chosen, and if the minimal maximum final speed is considered, $f^0 = 0$ and $k = -v(t_f)$ are chosen.

To sum up, the motion planning problem is to find a collision-free trajectory $\mathbf{x}(t) : [0, t_f] \to \mathbb{X}_{\text{free}}$ with $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$, that starts at $\mathbf{x}_{\text{init}}$, reaches the goal region $\mathbb{X}_{\text{goal}}$, i.e., $\mathbf{x}(0) = \mathbf{x}_{\text{init}}$ and $\mathbf{x}(t_f) \in \mathbb{X}_{\text{goal}}$ and minimizes the cost function J.

III. Motion Planning Framework

A. Optimal Rapidly Exploring Random Trees or RRT*

RRT* [11], [12] is an incremental method designed to explore efficiently nonconvex high-dimensional spaces by growing the search tree toward large Voronoi areas [19] with the asymptotic optimality property, i.e., almost-sure convergence to an optimal solution. The principle of the RRT* as a path planner is described in Algorithm 1.

Let G be the exploration tree, V be the set of vertices of the tree, E be the set of connecting edges of the tree, and cost($\{(\mathbf{x}_1, \mathbf{x}_2)\}$) be the minimal cost from $\mathbf{x}_1$ to $\mathbf{x}_2$ according to the specified criterion J defined in (3). Let cost($\mathbf{x}$) also be the total cost to arrive at $\mathbf{x}$, that is cost($\mathbf{x}$) = cost($\{(\mathbf{x}_{\text{init}}, \mathbf{x})\}$).

First, the initial state $\mathbf{x}_{\text{init}}$ is added to the tree G. Then, a state $\mathbf{x}_{\text{rand}} \in \mathbb{X}_{\text{free}}$ is either generated randomly or with help of some heuristics. In this brief, APFs are used as a heuristic (see Section III-B). The `nearest_neighbor` function (see Section III-C) searches the tree G for the nearest vertex to $\mathbf{x}_{\text{rand}}$ according to a user-defined metric d. This state is called $\mathbf{x}_{\text{nearest}}$. In `steer` function, a control input $\mathbf{u}^*$ is selected according to the specified criterion, i.e., such that $J(t_{\text{nearest}}, t_{\text{rand}}, \mathbf{u}^*)$ is minimized.

Then, the system model is integrated from t_{nearest} to t_{rand}, to find a new state $\mathbf{x}_{\text{new}}$, that is

$$\mathbf{x}_{\text{new}} = \mathbf{x}_{\text{nearest}} + \int_{t_{\text{nearest}}}^{t_{\text{rand}}} \mathbf{f}(\mathbf{x}, \mathbf{u}^*)\, dt.$$

A collision test (`collision_free_path` function) is performed: if $\mathbf{x}_{\text{new}}$ and the path between $\mathbf{x}_{\text{nearest}}$ and $\mathbf{x}_{\text{new}}$ lie in $\mathbb{X}_{\text{free}}$, then $\mathbf{x}_{\text{new}}$ is added in V.

Next, the RRT* algorithm tries to find a better parent for $\mathbf{x}_{\text{new}}$, that is a parent providing a lower cost to $\mathbf{x}_{\text{new}}$ than $\mathbf{x}_{\text{nearest}}$. The `near_vertices_parents` function in line 16 will search the tree G for a set of other potential parents in a neighborhood $\mathbb{X}_{\text{near}} \subset V$ of $\mathbf{x}_{\text{new}}$ (see Section III-D). The state $\mathbf{x}_{\text{min}} \in \mathbb{X}_{\text{near}} \cup \{\mathbf{x}_{\text{nearest}}\}$ that is collision-free and minimizes the cost to $\mathbf{x}_{\text{new}}$ is chosen to be its new parent. Therefore, the connecting edge from $\mathbf{x}_{\text{min}}$ to $\mathbf{x}_{\text{new}}$ is added in E.

Afterward, the `near_vertices_children` function in line 18 will search the tree G for a set of potential children in a neighborhood $\mathbb{X}_{\text{near}} \subset V$ of $\mathbf{x}_{\text{new}}$ (see Section III-D).

Algorithm 1 RRT* Path Planner

Function : build_rrt*(in : $K \in \mathbb{N}$, $\mathbf{x}_{\text{init}} \in \mathbb{X}_{\text{free}}$, $\mathbb{X}_{\text{goal}} \subset \mathbb{X}_{\text{free}}$, $\Delta t \in \mathbb{R}^+$, out : G)

1: $G \leftarrow \mathbf{x}_{\text{init}}$
2: cost($\mathbf{x}_{\text{init}}$) $\leftarrow 0$
3: $i = 0$
4: **repeat**
5: $\quad \mathbf{x}_{\text{rand}} \leftarrow$ random_state($\mathbb{X}_{\text{free}}$)
6: $\quad \mathbf{x}_{\text{new}} \leftarrow$ rrt*_extend($G, \mathbf{x}_{\text{rand}}$)
7: **until** $i++ > K$
8: **return** G

Function : rrt*_extend(in : G, $\mathbf{x}_{\text{rand}}$, out : $\mathbf{x}_{\text{new}}$)

9: $V \leftarrow G$.Node
10: $E \leftarrow G$.Edge
11: $\mathbf{x}_{\text{nearest}} \leftarrow$ nearest_neighbor($G, \mathbf{x}_{\text{rand}}$)
12: $(\mathbf{x}_{\text{new}}, \mathbf{u}^*) \leftarrow$ steer($\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{rand}}$)
13: **if** collision_free_path($\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{new}}$) **then**
14: $\quad V \leftarrow V \cup \{\mathbf{x}_{\text{new}}\}$
15: $\quad$ cost($\mathbf{x}_{\text{new}}$) $\leftarrow$ cost($\mathbf{x}_{\text{nearest}}$) + cost($\{(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{new}})\}$)
16: $\quad \mathbb{X}_{\text{near}} \leftarrow$ near_vertices_parents($G, \mathbf{x}_{\text{new}}$)
17: $\quad E \leftarrow$ best_edge($E, \mathbb{X}_{\text{near}}, \mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{new}}$)
18: $\quad \mathbb{X}_{\text{near}} \leftarrow$ near_vertices_children($\mathbf{x}_{\text{new}}, G$)
19: $\quad E \leftarrow$ rewire($E, \mathbb{X}_{\text{near}}, \mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{new}}$)
20: **end if**
21: $G = (V, E)$
22: **return** $\mathbf{x}_{\text{new}}$

Function : best_edge(in : E, $\mathbb{X}_{\text{near}}$, $\mathbf{x}_{\text{min}}$, $\mathbf{x}_{\text{new}}$ out : E)

23: **for all** $\mathbf{x}_{\text{near}} \in \mathbb{X}_{\text{near}} \setminus \{\mathbf{x}_{\text{min}}\}$ **do**
24: $\quad (\mathbf{x}_{\text{new}}, \mathbf{u}^*) \leftarrow$ steer($\mathbf{x}_{\text{near}}, \mathbf{x}_{\text{new}}$)
25: $\quad$ **if** collision_free_path($\mathbf{x}_{\text{near}}, \mathbf{x}_{\text{new}}$) and cost($\mathbf{x}_{\text{new}}$) > cost($\mathbf{x}_{\text{near}}$) + cost($\{(\mathbf{x}_{\text{near}}, \mathbf{x}_{\text{new}})\}$) **then**
26: $\quad\quad$ cost($\mathbf{x}_{\text{new}}$) $\leftarrow$ cost($\mathbf{x}_{\text{near}}$) + cost($\{(\mathbf{x}_{\text{near}}, \mathbf{x}_{\text{new}})\}$)
27: $\quad\quad \mathbf{x}_{\text{min}} \leftarrow \mathbf{x}_{\text{near}}$
28: $\quad$ **end if**
29: **end for**
30: $E \leftarrow E \cup \{(\mathbf{x}_{\text{min}}, \mathbf{x}_{\text{new}})\}$
31: **return** E

Function : rewire(in : E, $\mathbb{X}_{\text{near}}$, $\mathbf{x}_{\text{nearest}}$, $\mathbf{x}_{\text{new}}$ out : E)

32: **for all** $\mathbf{x}_{\text{near}} \in \mathbb{X}_{\text{near}} \setminus \{\mathbf{x}_{\text{nearest}}\}$ **do**
33: $\quad (\mathbf{x}_{\text{near}}, \mathbf{u}^*) \leftarrow$ steer($\mathbf{x}_{\text{new}}, \mathbf{x}_{\text{near}}$)
34: $\quad$ **if** collision_free_path($\mathbf{x}_{\text{new}}, \mathbf{x}_{\text{near}}$) and cost($\mathbf{x}_{\text{near}}$) > cost($\mathbf{x}_{\text{new}}$) + cost($\{(\mathbf{x}_{\text{new}}, \mathbf{x}_{\text{near}})\}$) **then**
35: $\quad\quad \mathbf{x}_{\text{parent}} \leftarrow$ parent($\mathbf{x}_{\text{near}}$)
36: $\quad\quad E \leftarrow E \setminus \{(\mathbf{x}_{\text{parent}}, \mathbf{x}_{\text{near}})\}$
37: $\quad\quad E \leftarrow E \cup \{(\mathbf{x}_{\text{new}}, \mathbf{x}_{\text{near}})\}$
38: $\quad\quad$ cost($\mathbf{x}_{\text{near}}$) $\leftarrow$ cost($\mathbf{x}_{\text{new}}$) + cost($\{(\mathbf{x}_{\text{new}}, \mathbf{x}_{\text{near}})\}$)
39: $\quad$ **end if**
40: **end for**
41: **return** E

For each $\mathbf{x}_{\text{near}} \in \mathbb{X}_{\text{near}}$, if the cost to $\mathbf{x}_{\text{near}}$ passing by $\mathbf{x}_{\text{new}}$ is better than cost($\mathbf{x}_{\text{near}}$) and the path is collision-free, then the `rewire` function will replace the existing connecting edge by the connecting edge from $\mathbf{x}_{\text{new}}$ to $\mathbf{x}_{\text{near}}$.

These steps are repeated until the algorithm reaches K iterations. Thus, the RRT* algorithm will improve the optimality of the solution over time even after the first solution is found.

B. State Generation Using Artificial Potential Fields or APF

The random state $\mathbf{x}_{\text{rand}}$ is generally generated by a uniform distribution in such a way that $\mathbf{x}_{\text{rand}} \in \mathbb{X}_{\text{free}}$. In this brief, a biased random state generation using APFs [13], [14] is introduced.

The APF is a reactive approach where trajectories are not generated explicitly. Instead, the environments generate some forces leading the vehicle to the destination. However, problems, such as local minima and oscillatory movement, can make the goal nonreachable.

The APF can be used to direct $\mathbf{x}_{\text{rand}}$ to $\mathbb{X}_{\text{goal}}$, i.e., the orientation (γ, χ) of $\mathbf{x}_{\text{rand}}$ is generated using the APF. By combining with the RRT*, the disadvantages of the APF can be solved by the randomness of the RRT*, i.e., the vehicle leaves the local minima by trying to go to $\mathbf{x}_{\text{rand}}$ [20], [21]. At the same time, it is expected that the APF also increases the rate of convergence to the optimal solution.

An artificial vector $\mathbf{f}_{\text{APF}}$ used to direct the vehicle is induced by an artificial potential $U \in \mathbb{R}$, i.e., $\mathbf{f}_{\text{APF}} = -\nabla U \in \mathbb{R}^3$. The potentials can be defined in several ways according to the characteristics of the mathematical functions. In this brief, the following artificial potentials and vectors are used.

1) The repulsive potential [14] is usually used around an obstacle for the vehicle to avoid it. It can also be used to guide the vehicle away from the initial state. The repulsive potential and force can be expressed as

$$U_{\text{init}} = \frac{1}{2} K_{\text{init}} \frac{1}{||\boldsymbol{\xi} - \boldsymbol{\xi}_{\text{init}}||_2^2} \tag{4}$$

$$\mathbf{f}_{\text{init}} = K_{\text{init}} \frac{\boldsymbol{\xi} - \boldsymbol{\xi}_{\text{init}}}{||\boldsymbol{\xi} - \boldsymbol{\xi}_{\text{init}}||_2^4} \tag{5}$$

where U_{init} and $\mathbf{f}_{\text{init}}$ are the repulsive potential and vector field centered at $\boldsymbol{\xi}_{\text{init}}$ and K_{init} is a constant.

2) The attractive potential, opposing to the repulsive potential, is used to direct the vehicle to the desired destination. The attractive potential and force can be expressed as

$$U_{\text{goal}} = -\frac{1}{2} K_{\text{goal}} \frac{1}{||\boldsymbol{\xi} - \boldsymbol{\xi}_{\text{goal}}||_2^2} \tag{6}$$

$$\mathbf{f}_{\text{goal}} = -K_{\text{goal}} \frac{\boldsymbol{\xi} - \boldsymbol{\xi}_{\text{goal}}}{||\boldsymbol{\xi} - \boldsymbol{\xi}_{\text{goal}}||_2^4} \tag{7}$$

where U_{goal} and $\mathbf{f}_{\text{goal}}$ are the attractive potential and vector fields centered at $\boldsymbol{\xi}_{\text{goal}}$ and K_{goal} is a constant.

3) The rotational vector field [22] is used instead of the repulsive potential to guide the vehicle around the obstacles in order to reduce the local minimum problem. In Fig. 2, let $\boldsymbol{\xi}_{\text{obs}_c}$ denote the center of gravity of an obstacle, $\mathbb{S}_{\text{obs}}$ denote the set of the obstacle surface, d_{obs} denote a maximum distance of influence from the obstacle, r_{obs} denote the shortest distance of the vehicle from the obstacle, and $\mathbf{t}$, $\boldsymbol{\sigma}$, $\boldsymbol{\lambda}$ are unit vectors defined as

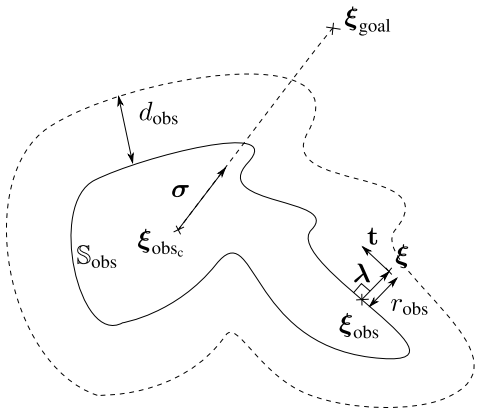

Fig. 2. Example of rotational vector field around the obstacle.

$$\boldsymbol{\sigma} = \frac{\boldsymbol{\xi}_{\text{goal}} - \boldsymbol{\xi}_{\text{obs}_c}}{||\boldsymbol{\xi}_{\text{goal}} - \boldsymbol{\xi}_{\text{obs}_c}||}$$

$$\boldsymbol{\lambda} = \frac{\boldsymbol{\xi} - \boldsymbol{\xi}_{\text{obs}}}{||\boldsymbol{\xi} - \boldsymbol{\xi}_{\text{obs}}||}$$

$$\mathbf{t} = \frac{\boldsymbol{\lambda} \times (\boldsymbol{\sigma} \times \boldsymbol{\lambda})}{||\boldsymbol{\lambda} \times (\boldsymbol{\sigma} \times \boldsymbol{\lambda})||}.$$

Remark 1: If $\boldsymbol{\lambda}$ is collinear to $\boldsymbol{\sigma}$, $\mathbf{t}$ can be chosen arbitrarily. In that case, the most simple solution is to choose a random $\mathbf{t}$ perpendicular to $\boldsymbol{\lambda}$. Other strategies could also be adopted.

The vector field function can be defined as

$$\mathbf{f}_{\text{obs}} = K_{\text{obs}} \left(\frac{d_{\text{obs}} - r_{\text{obs}}}{r_{\text{obs}}} \right) \mathbf{t}, \quad r_{\text{obs}} \leq d_{\text{obs}} \tag{8}$$

$$\mathbf{f}_{\text{obs}} = \mathbf{0}, \quad r_{\text{obs}} > d_{\text{obs}} \tag{9}$$

where $\boldsymbol{\xi}_{\text{obs}} \in \mathbb{S}_{\text{obs}}$ is the nearest point of an obstacle to $\boldsymbol{\xi}$ and K_{obs} is a constant. Note that for a spherical obstacle, $\boldsymbol{\lambda} = \frac{\boldsymbol{\xi} - \boldsymbol{\xi}_{\text{obs}_c}}{||\boldsymbol{\xi} - \boldsymbol{\xi}_{\text{obs}_c}||}$.

Thus, the summation of artificial vector fields expressed as

$$\mathbf{f}_{\text{APF}} = \sum_{i=1}^{n} \mathbf{f}_{\text{i}} = \mathbf{f}_{\text{goal}} + \mathbf{f}_{\text{init}} + \mathbf{f}_{\text{obs}} \tag{10}$$

is used as a basis to generate $(\gamma_{\text{rand}}, \chi_{\text{rand}})$ of $\mathbf{x}_{\text{rand}}$.

Thus, the `random_state` function generates a random state $\mathbf{x}_{\text{rand}} \in \mathbb{X}_{\text{free}}$ by using the uniform distribution as in the original algorithm. The orientation of the randomly generated state is biased toward the destination using the direction of $\mathbf{f}_{\text{APF}}$ with a given margin, that is, $\mathbf{x}_{\text{rand}}$ is chosen in the convex cone shown in Fig. 3 pointing toward $\mathbf{f}_{\text{APF}}$ with apex $\boldsymbol{\xi}_{\text{rand}}$ and apex angle $2\phi_{\text{APF}}$. This makes that our approach differs from [17] as their approach uses $\mathbf{f}_{\text{APF}}$ to update the position of the generated $\mathbf{x}_{\text{rand}}$ before using it and the orientations are not considered. Moreover, a bias toward the goal, used in RRT-*GoalBias* [9], is also used to increase the rate of convergence to a solution. It consists in choosing $\mathbf{x}_{\text{rand}} \in \mathbb{X}_{\text{goal}}$ with a probability p.

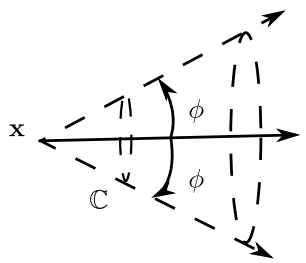

Fig. 3. Definition of convex cone $\mathbb{C}(\mathbf{x}, \phi)$.

C. nearest_neighbor

The nearest state $\mathbf{x}_{\text{near}}$ to $\mathbf{x}_{\text{rand}}$ is determined by a metric. The Euclidean metric is often used in such an algorithm. It finds the shortest line-of-sight distance between two states. Moreover, the kd-tree [23] can be used to enhance the robustness of the algorithm. Thus, it is fast and easy to implement. However, for a nonholonomic vehicle, the Euclidean metric is not appropriate for several reasons. The main reason is that it does not consider the orientation of the vehicle. A suitable metric can be, for example, based on the shortest Dubins' path used in [24] even if it does not perfectly fit the vehicle model. Another suitable metric can be, for example, based on Bézier curves or other methods based on a simplified suboptimal trajectory calculation. Although more complex to compute, the most interesting metric in our framework consists in considering the cost from $\mathbf{x}_{\text{near}}$ to $\mathbf{x}_{\text{rand}}$, i.e., $\text{cost}(\{(\mathbf{x}_{\text{near}}, \mathbf{x}_{\text{rand}})\})$.

D. near_vertices

In the RRT* algorithm described in [11] and [12], the near_vertices function uses the same metric function as nearest_neighbor function to determine the neighborhood $\mathbb{X}_{\text{near}}$ of the state $\mathbf{x}$. This function searches for the vertices in a reachable set containing a ball, centered at $\mathbf{x}$, of volume $\lambda_1 \ln(n)/n$, where n is the number of vertices in V and λ_1 is a positive constant chosen appropriately to ensure asymptotic optimality.

In the present framework, the k-nearest neighbors algorithm is used to determine a set of near vertices $\mathbb{X}_{\text{near}}$ of the state $\mathbf{x}$. It selects the first k-nearest vertices according to the user-defined metric d and returns them to the set $\mathbb{X}_{\text{near}}$. To ensure asymptotic optimality, $k(n)$ is chosen such that $k(n) > \lambda_2 \ln(n)$, where λ_2 is a positive constant (see the Appendix for more details).

The original near_vertices function is divided into two functions in this brief: near_vertices_parents function in line 16, and near_vertices_children function in line 18. The first one searches for the k-nearest vertices to arrive at $\mathbf{x}_{\text{new}}$, while the latter one searches for the k-nearest vertices from $\mathbf{x}_{\text{new}}$ to other vertices.

This APF-biased framework is expected to be able to find a feasible near-optimal solution with less number of iterations than the existing RRT* algorithm. This will be demonstrated in simulations in Section V for the application considered in Section IV.

IV. Application

The performance of the previously mentioned framework is demonstrated using the application of an interceptor missile. The shortest 3-D Dubins' path in a heterogeneous environment as the user-defined metric d (see [25] for details) is used. This shortest 3-D Dubins' path was specifically developed for missile applications. In Sections IV-A–IV-C, the problem statement of the case study is described. Then, the use of Dubins's metric is explained.

A. System Modeling

A simplified hypersonic vehicle is used to simulate results. In this brief, an unpowered interceptor missile during midcourse phase is chosen. For a hypersonic missile, wind speed has such a negligible effect to the system that a zero wind assumption is applied. Then, the translational velocity $\mathbf{v}$ is assumed to coincide with the apparent velocity. Besides, a hypersonic missile is studied. Thus, the gravity can be neglected, which is a strong hypothesis that is only valid for missile-like aircraft flying with a high Mach number in a short distance. Moreover, the drag can be ignored, since the objective is to find the shortest path between two states, i.e., the path of minimum length. Thus, the dynamics of the velocity does not need to be considered. Moreover, the aerodynamic coefficient C_{L_2} is equal to C_{L_3}, because an axisymmetric missile is considered here.

By simplifying (1) according to the mentioned conditions and applying a change of variables from t to curvilinear abscissa $s(t) = \int_0^t v(u)\,du$, the dynamics of a Dubins-like aerial vehicle can be written as

$$\begin{cases} x' = \dfrac{dx}{ds} = \cos\gamma\,\cos\chi \\ y' = \dfrac{dy}{ds} = \cos\gamma\,\sin\chi \\ z' = \dfrac{dz}{ds} = \sin\gamma \\ \gamma' = \dfrac{d\gamma}{ds} = \dfrac{1}{2m}\rho(z)SC_{L_{\max}}\mu = c(z)\mu, \\ \chi' = \dfrac{d\chi}{ds} = \dfrac{1}{2m}\rho(z)SC_{L_{\max}}\dfrac{\eta}{\cos\gamma} = c(z)\dfrac{\eta}{\cos\gamma} \end{cases} \tag{11}$$

where $C_{L_{\max}}$ is the maximum lift coefficient, μ and η are the normalized control inputs bounded by condition $(\mu^2 + \eta^2)^{1/2} \leq 1$, $\rho(z)$ is the air density, and $c(z)$ is the maximum path curvature of the vehicle. Note that μ and η are directly related the angle of attack and the sideslip angle.

The simplified air density function can be expressed as

$$\rho(z) = \rho_0 e^{-z/z_r} \tag{12}$$

where ρ_0 is the air density at standard atmosphere at sea level and z_r is a reference altitude.

As a consequence, the path curvature can be written in the same way as

$$c(z) = c_0 e^{-z/z_r}. \tag{13}$$

where c_0 is the maximum curvature at sea level. At this point, it can be noticed that the maneuverability of the vehicle is exponentially decreasing with altitude, since $c(z)$ is exponentially decreasing. Thus, it is a challenging problem to control such a vehicle at high altitude.

Remark 2: Our objective is to find a reference trajectory for such a vehicle. It is meant to follow the reference trajectory

with its controller. Therefore, the elements related to the controller [26], such as model uncertainties and measurement errors, are not considered here.

B. Problem Formulation

In this application, $\mathbf{x} = (\boldsymbol{\xi}^\top, \gamma, \chi)^\top \in \mathbb{X} = \mathbb{R}^5$ is the measurable state of the system and $\mathbf{u} = (\mu, \eta)^\top \in \mathbb{U}$ is an admissible control input. The set of admissible control inputs is defined as

$$\mathbb{U} = \{\mathbf{u} \in \mathbb{R}^2 : ||\mathbf{u}|| \leq 1\}. \tag{14}$$

The differential system (11) is rewritten as

$$\mathbf{x}' = \mathbf{f}(\mathbf{x}, \mathbf{u}). \tag{15}$$

The obstacle region $\mathbb{X}_{\text{obs}}$ is defined by the exploration space occupied by no-fly zones such as city areas and radar detection zones, and the set of admissible states is the remaining of the exploration space, i.e., $\mathbb{X}_{\text{free}} = \mathbb{X} \setminus \mathbb{X}_{\text{obs}}$.

The path planning starts at the initial state $\mathbf{x}_{\text{init}} \in \mathbb{X}_{\text{free}}$. The destination of the path planning is given by a rendezvous set $\mathbb{X}_{\text{goal}} \subset \mathbb{X}_{\text{free}}$ and defined as

$$\mathbb{X}_{\text{goal}} = \mathbb{P}_{\text{goal}} \times \mathbb{V}_{\text{goal}} \tag{16}$$

where $\mathbb{P}_{\text{goal}}$ is the set of the desired arrival positions and $\mathbb{V}_{\text{goal}}$ is the set of the desired orientations of the vehicle. In this application, they are defined as

$$\begin{aligned} \mathbb{P}_{\text{goal}} &= \{\boldsymbol{\xi}_{\text{rdv}}\} \\ \mathbb{V}_{\text{goal}} &= \{(\gamma, \chi) \in \mathbb{R}^2 : (\gamma, \chi) \in \mathbb{C}(\mathbf{x}_{\text{rdv}}, \phi_f)\} \end{aligned} \tag{17}$$

where $\mathbf{x}_{\text{rdv}}$ is the rendezvous state, and $\mathbb{C}(\mathbf{x}_{\text{rdv}}, \phi_f)$ is the convex cone (see Fig. 3) pointing toward the orientation of the vehicle defined by γ_{rdv} and χ_{rdv} with apex $\boldsymbol{\xi}_{\text{rdv}}$ and apex angle $2\phi_f$, where ϕ_f is a maximal acceptable orientation error defined by the detection range of the embedded radar or infrared sensor.

The objective of path planning algorithm is to find a collision-free trajectory $\mathbf{x}(s) : [0, s_f] \rightarrow \mathbb{X}_{\text{free}}$ with $\mathbf{x}' = \mathbf{f}(\mathbf{x}, \mathbf{u})$, that starts at $\mathbf{x}_{\text{init}}$, reaches the goal region $\mathbb{X}_{\text{goal}}$, i.e., $\mathbf{x}(0) = \mathbf{x}_{\text{init}}$ and $\mathbf{x}(s_f) \in \mathbb{X}_{\text{goal}}$ and minimizes the cost function

$$J = \int_0^{s_f} ds. \tag{18}$$

C. Dubins' Paths for the Metric

Interestingly, the shortest 3-D Dubins' path in a heterogeneous environment developed in [25] solves the optimal problem (18) for the considered model (11) provided that the environment is not cluttered with obstacles. In presence of obstacles, it can still provide locally optimal solutions. That is the reason why the proposed metric is based on this path.

There are two cases to consider when using Dubins' paths as the user-defined metric d.

1) $\mathbf{x}_{rand} \notin \mathbb{X}_{goal}$*:* The shortest CSC (curve-segment-curve) path is used to calculate the nearest neighbor.
2) $\mathbf{x}_{rand} \in \mathbb{X}_{goal}$*:* Since $\mathbb{X}_{\text{goal}}$ is a set, it is more interesting to consider the shortest path between each $\mathbf{x} \in G$ and $\mathbb{X}_{\text{goal}}$ than considering a single $\mathbf{x}_{\text{rand}} \in \mathbb{X}_{\text{goal}}$. Indeed, there can exist a shortest path to another element of $\mathbb{X}_{\text{goal}}$ than the shortest path to $\mathbf{x}_{\text{rand}}$. To this manner, the degenerated form (CS) of CSC path needs to be considered first. Indeed, if the shortest path to the set $\mathbb{X}_{\text{goal}}$ is a CS path, then it arrives with the orientation within the arrival cone $\mathbb{C}$, and thus, the expected shortest path is this CS path. If no CS path arrives in $\mathbb{C}$, the shortest path is necessarily a CSC path whose arrival orientation is one of the extremities of the arrival cone $\mathbb{C}$.

Thus, for each $\mathbf{x} \in G$, the approach first consists in finding the shortest CS path to the desired final position, i.e., $\boldsymbol{\xi}_{\text{rand}} = \boldsymbol{\xi}_{\text{rdv}}$. If the final orientation is in the arrival cone, it is the expected solution. If not, the solution is the shortest CSC path to one of the extremities of $\mathbb{X}_{\text{goal}}$.

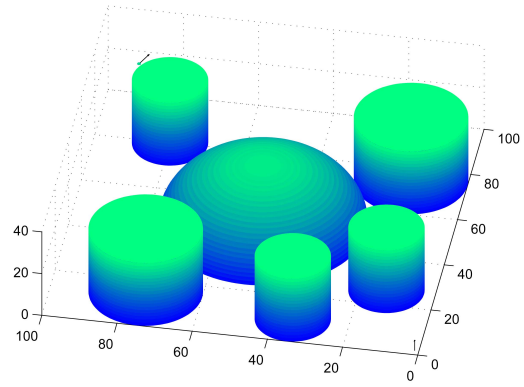

Fig. 4. Illustration of the scenario.

V. Simulation Results

Simulation results are obtained using MATLAB. The scenario shown in Fig. 4 is considered with the following configurations: the initial state $\mathbf{x}_{\text{init}} = (1 \text{ km}, 1 \text{ km}, 1 \text{ km}, \pi/2, 0)^\top$

$$\begin{aligned} \mathbb{X}_{\text{goal}} &= \mathbb{P}_{\text{goal}} \times \mathbb{V}_{\text{goal}} \\ \mathbb{P}_{\text{goal}} &= \{\boldsymbol{\xi}_{\text{rdv}}\} \\ \mathbb{V}_{\text{goal}} &= \{(\gamma, \chi) \in \mathbb{R}^2 : (\gamma, \chi) \in \mathbb{C}(\mathbf{x}_{\text{rdv}}, 5°)\} \end{aligned} \tag{19}$$

with $\mathbf{x}_{\text{rdv}} = (90 \text{ km}, 90 \text{ km}, 25 \text{ km}, 0, -\pi/8)^\top$.

In Fig. 4, the arrows represent the orientations of the initial state $\mathbf{x}_{\text{init}}$ and of the rendezvous state

$$\mathbf{x}_{\text{rdv}} = \left(\boldsymbol{\xi}_{\text{rdv}}^\top, \gamma_{\text{rdv}}, \chi_{\text{rdv}}\right)^\top.$$

$\mathbb{X}_{\text{obs}}$ is represented by the 3-D shaded surfaces (in gradient color). The nature of obstacles can be no-fly zones, such as the area above and around cities, which are represented by cylinders and radar detection zones which are represented by a half sphere.

Here, three different algorithms are considered:

1) RRT;
2) RRT*;
3) RRT* biased by APF with a given margin for the orientation of the randomly generated state $\mathbf{x}_{\text{rand}}$ of 10° ($\phi_{\text{APF}} = 10°$).

For all algorithms, a bias toward the goal consisting in generating $\mathbf{x}_{\text{rand}} \in \mathbb{X}_{\text{goal}}$ with a probability $p = 0.1$ is used. Moreover, the Dubins' path is used for the metric as mentioned in Section IV-C.

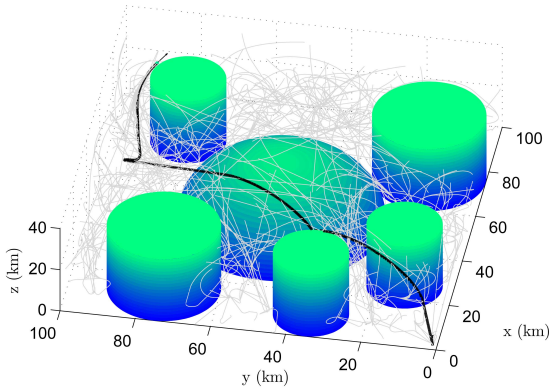

Fig. 5. Exploration trees and results after 1000 iterations of RRT algorithm: path length 200 km.

In the following figures, the exploration tree is represented in light gray and thick solid curve is the final solution found by the algorithms. Fig. 5 shows a result obtained after 1000 iterations by the RRT algorithm. It shows that the RRT algorithm is capable of finding a solution for the problem. However, the obtained trajectory is clearly not the shortest since some loops and turns can be observed. This is because no optimality criteria are considered by the RRT. Moreover, no reconnection is done to improve the path quality.

The other two algorithms are employed on the same scenario. Fig. 6 shows one of the best results obtained by the RRT* and by the RRT* biased by APF after 200 iterations. As we can see from the results, the RRT* algorithm can find a better solution than the RRT. In Fig. 6, the difference between the two RRT* algorithms is that the exploration tree of the RRT* biased by APF [see Fig. 6(b)] tends toward the destination, while the other one expands in every direction [see Fig. 6(a)].

For the in-depth analysis, for each algorithm, 100 Monte Carlo simulations within 200 iterations are simulated to obtain statistic results. Here, the computational efficiency is compared using the number of iterations rather than the computing time, since the code is not fully optimized (MATLAB implementation) to assess its real-time ability. Moreover, the computational effort is mostly spent by the metric function, which can still be optimized. The average iteration needed to obtain the first solution, the average length of the first solution, and the average length of the final solution are observed in Table I.

According to the statistic results in Table I, the first solution can be obtained at almost the same iteration for both RRT and RRT* algorithms. Note also that the optimality of the final solution of the RRT* improves with respect to the number of iterations. The RRT* biased by APF is the fastest to find the first solution. On top of that, both first and final solutions are closer to the optimal solution than the others. The reason is that the heuristic consisting in using the APF helps to bias the search toward the feasible and optimal solutions rather than blindly search the exploration space, i.e., increase the probability of reconnection of the RRT*. Obviously, the RRT* can achieve the same final performance if more iterations are given.

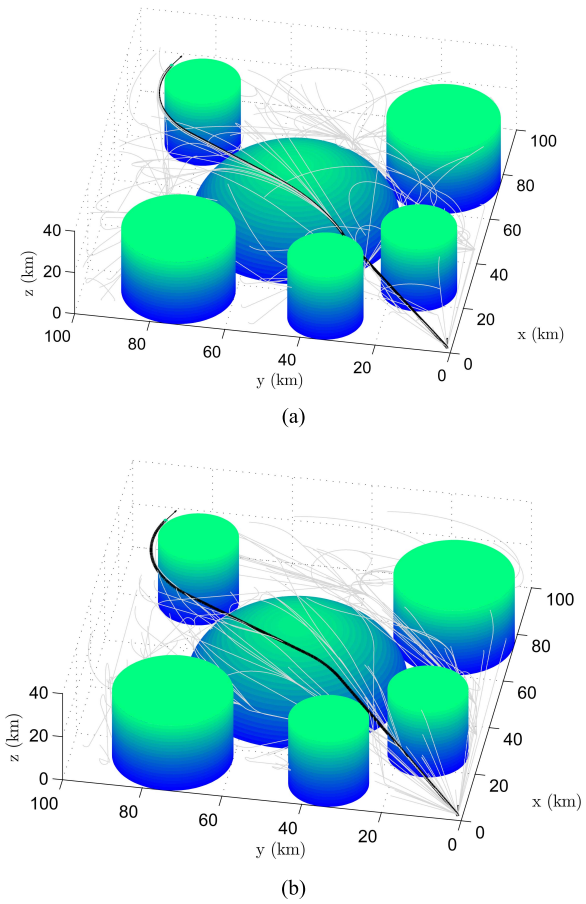

Fig. 6. Exploration trees and results after 200 iterations. (a) RRT*: path length 140.36 km. (b) RRT* biased by APF: path length 139.89 km.

TABLE I
RESULTS OF 100 MONTE CARLO SIMULATIONS WITHIN 200 ITERATIONS

Method	Average iteration for the 1st solution	Average length of the 1st solution	Average length of the final solution
RRT	85	241 km	239 km
RRT*	82	197 km	185 km
RRT* biased by APF	46	169 km	155 km

Monte Carlo simulations of several scenarios, which are not shown here, have also been simulated. For less complex scenarios, i.e., fewer obstacles, the results of the RRT* and the RRT* biased with APF are mostly the same. Since there are few obstacles, the algorithms converge rapidly, because the probability of reconnection is already high. Thus, it is reasonable that the performance does not improve as much in less complex environments for nonholonomic problems.

VI. CONCLUSION

In this brief, an efficient trajectory planning framework for aerial vehicles traveling in 3-D space while avoiding obstacles is presented. The algorithm is based on the RRT* algorithm to find a near-optimal trajectory. Moreover, the APF is integrated into the algorithm to accelerate the convergence speed to the optimal solution. The capability of this framework is

demonstrated on a missile application. The shortest Dubins' paths in a heterogeneous environment are used for the metric. As a result, the approach shows a good performance in simulation results. Compared with the RRT and the RRT* algorithms, better solutions are found with less number of iterations due to the integration of the APF. Although the computational complexity of this method is not rigorously analyzed in this brief, real-time constraints will be easier to verify if the algorithm is implemented on board and the vehicle as less number of iterations is required to find a solution. Moreover, it is important to notice that the algorithm is anytime, since the first admissible solution can be obtained pretty fast, the remaining computing resources being only used to improve the optimality. This property is very interesting for real-time applications. In future work, replanning will be considered to improve the performance of the algorithm in case of dynamic environment or changes in the mission.

APPENDIX
CONVERGENCE ANALYSIS

In this section, a set of sufficient conditions is proposed to guarantee the convergence of the RRT* algorithm for the aerial vehicle path planning. As stated in [11], local controllability of the system, as well as the existence of an optimal trajectory with sufficient free space in its neighborhood, is necessary conditions to ensure asymptotic optimality of the RRT* algorithm. Since the aerial system considered in this brief is assumed to be controllable in its domain of use, local controllability is guaranteed. Therefore, it is only required to assume that there exists an optimal trajectory with free space around it, as explained thoroughly in [11]. Thus, completeness and almost-sure convergence are allowed. The APF bias that is used in the framework does not contradict this assumption, since a given margin ϕ_{APF} described in Section III-B is used for the uniform state generation. Indeed, to ensure free space in the neighborhood, it is only required that any states $\mathbf{x}$ on the optimal path are in the convex cone $\mathbb{C}(\mathbf{x}_{\text{APF}}, \phi_{\text{APF}})$ (see Fig. 3 for a definition of $\mathbb{C}$), where $\mathbf{x}_{\text{APF}}$ is a state on the vector field at position $\boldsymbol{\xi}$ ($\boldsymbol{\xi}_{\text{APF}} = \boldsymbol{\xi}$).

To ensure asymptotic optimality, the `near_vertices` function needs to be analyzed. In [11], it is shown that if the area of research contains a ball of volume $\lambda_1 \ln(n)/n, \lambda_1 > 0$, then the almost-sure convergence is ensured provided that λ_1 is sufficiently large. If the k-nearest neighbors algorithm is used, k needs to verify $k > \lambda_2 \ln(n), \lambda_2 > 0$. Indeed, since a uniform distribution is used for the state generation, the expected number of vertices contained in a ball of volume $\lambda_1 \ln(n)/n$ after iteration n is $\lambda_1 \ln(n)/\lambda_0$, where λ_0 is the total volume of the free space. The proof is straightforward since the number of vertices contained in the ball follows a binomial distribution with parameters n and $\lambda_1 \ln(n)/\lambda_0$ n. Thus, intuitively, both techniques are equivalent and choosing $k > \lambda_1 \ln(n)/\lambda_0$ for the k-nearest neighbors algorithm ensures asymptotic optimality. A thorough proof can be found in [12].

REFERENCES

[1] E. Trélat, "Optimal control and applications to aerospace: Some results and challenges," *J. Optim. Theory Appl.*, vol. 154, no. 3, pp. 713–758, 2012.

[2] C.-F. Lin, *Modern Navigation, Guidance, And Control Processing*. Upper Saddle River, NJ, USA: Prentice-Hall, 1991.

[3] H. J. Kim, D. H. Shim, and S. Sastry, "Nonlinear model predictive tracking control for rotorcraft-based unmanned aerial vehicles," in *Proc. Amer. Control Conf.*, vol. 5. 2002, pp. 3576–3581.

[4] J. Carsten, D. Ferguson, and A. Stentz, "3D field D*: Improved path planning and replanning in three dimensions," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, Oct. 2006, pp. 3381–3386.

[5] R. Daily and D. M. Bevly, "Harmonic potential field path planning for high speed vehicles," in *Proc. Amer. Control Conf.*, 2008, pp. 4609–4614.

[6] X. Chen and J. Zhang, "The three-dimension path planning of UAV based on improved artificial potential field in dynamic environment," in *Proc. Int. Conf. Intell. Human-Mach. Syst. Cybern.*, 2013, pp. 144–147.

[7] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[8] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.

[9] S. M. LaValle and J. J. Kuffner, Jr., "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, no. 5, pp. 378–400, 2001.

[10] P. Pharpatara, R. Pepy, B. Hérissé, and Y. Bestaoui, "Missile trajectory shaping using sampling-based path planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Nov. 2013, pp. 2533–2538.

[11] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *Proc. IEEE Conf. Decision Control*, Dec. 2010, pp. 7681–7687.

[12] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[13] J. R. Andrews and N. Hogan, "Impedance control as a framework for implementing obstacle avoidance in a manipulator," in *Proc. ASME Winter Annu. Meeting Control Manuf. Process. Robot. Syst.*. 1983, pp. 243–251.

[14] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, Mar. 1985, pp. 500–505.

[15] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Sep. 2014, pp. 2997–3004.

[16] F. Islam, J. Nasir, U. Malik, Y. Ayaz, and O. Hasan, "RRT*-smart: Rapid convergence implementation of RRT* towards optimal solution," in *Proc. IEEE Int. Conf. Mechatronics Autom.*, Aug. 2012, pp. 1651–1656.

[17] A. H. Qureshi, K. F. Iqbal, S. M. Qamar, F. Islam, Y. Ayaz, and N. Muhammad, "Potential guided directional-RRT* for accelerated motion planning in cluttered environments," in *Proc. IEEE Int. Conf. Mechatronics Autom.*, Aug. 2013, pp. 519–524.

[18] P. Pharpatara, B. Hérissé, R. Pepy, and Y. Bestaoui, "Shortest path for aerial vehicles in heterogeneous environment using RRT*," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2015, pp. 6388–6393.

[19] G. Voronoi, "Nouvelles applications des paramétres continus à la théorie des formes quadratiques," *J. fur die Reine und Angewandte Math.*, vol. 133, pp. 97–178, 1908.

[20] J. Barraquand and J.-C. Latombe, "A Monte-Carlo algorithm for path planning with many degrees of freedom," in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 3. May 1990, pp. 1712–1717.

[21] J. Barraquand and J.-C. Latombe, "Robot motion planning: A distributed representation approach," *Int. J. Robot. Res.*, vol. 10, no. 6, pp. 628–649, 1991.

[22] P. Falstad. (Feb. 2014). *3-D Vector Fields Applet*. [Online]. Available: http://www.falstad.com/vector3d/directions.html

[23] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[24] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *Amer. J. Math.*, vol. 79, no. 3, pp. 497–516, 1957.

[25] P. Pharpatara, B. Hérissé, and Y. Bestaoui, "3D-shortest paths for a hypersonic glider in a heterogeneous environment," in *Proc. IFAC Workshop Adv. Control Navigat. Auto. Aerosp. Vehicles*, 2015, pp. 186–191.

[26] E. Devaud, H. Siguerdidjane, and S. Font, "Some control strategies for a high-angle-of-attack missile autopilot," *Control Eng. Pract.*, vol. 8, no. 8, pp. 885–892, 2000.