Abstract—This brief presents a trajectory planning algorithm for aerial vehicles traveling in 3-dimensional space while avoiding obstacles. The nature of the obstacles can be, for example, radar detection areas, cooperating and non-cooperating vehicles, etc. Thus, it is a complex trajectory planning problem. The proposed planner is based on the RRT* algorithm. Artificial Potential Fields (APF) 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, 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—Path planning algorithm, optimal control, RRT* algorithm, complex environment, aerial robotics.

I. INTRODUCTION

THE development of autonomous aerial vehicles such as drones, missiles or Unmanned Combat Aerial Vehicles (UCAVs) 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, cooperating and non-cooperating vehicles, etc., but also 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 (MPC) are also widely used [3] by combining open-loop optimal control with feedback control. These techniques provide a good trade-off 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 (PRM) [7] or Rapidly-exploring Random Trees (RRT) [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 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, asymptotic optimality of the RRT* can be ensured for aerial vehicles such as airplanes or missiles.

In this paper, a general framework for trajectory planning algorithm of aerial vehicles traveling in a 3-dimensional 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 (APF) [13], [14] are integrated to bias the random sampling to accelerate the convergence speed of 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-dimensional problem for this specific application. Simulation results demonstrate that the algorithm can efficiently find a feasible near-optimal trajectory.

The 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 existing algorithms, and analyzed in section V. Finally, some concluding remarks are made in the last section.

II. PROBLEM STATEMENT

A. System modeling

In this paper, 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 (ECEF) reference frame $\mathcal{I}$ centered at point $O$ and associated with the basis vectors $(i,j,k)$; a body-fixed frame $\mathcal{B}$ attached to the vehicle at its center of mass $C_\mathcal{B}$ with the basis vector $(e^b_1,e^b_2,e^b_3)$; and a velocity frame $\mathcal{V}$ attached to the vehicle at $C_\mathcal{V}$ with the basis vector $(e^v_1,e^v_2,e^v_3)$ where the translational velocity of the vehicle is denoted $\mathbf{v} = v e^v_1$ and $v$ is the speed of the vehicle. Position and velocity defined in $\mathcal{I}$ are denoted $\mathbf{x} = (x,y,z)^T \in \mathbb{R}^3$ and $\mathbf{v} = (\dot{x},\dot{y},\dot{z})^T \in \mathbb{R}^3$. Denote $\gamma$ and $\chi$ the orientation w.r.t $\mathcal{I}$ of the velocity $\mathbf{v}$, where $\gamma$ is the flight path angle and $\chi$ is the azimuth angle.

The dynamics of the aerial vehicle can be written as

\[
\begin{align*}
\dot{x} & = v \cos \gamma \cos \chi, \\
\dot{y} & = v \cos \gamma \sin \chi, \\
\dot{z} & = v \sin \gamma, \\
\dot{v} & = -g \sin \gamma + \frac{F_{a_1}}{m} + \frac{F_{p_1}}{m}, \\
\dot{\gamma} & = \frac{g}{v} \cos \gamma + \frac{F_{a_2}}{mv \cos \gamma} + \frac{F_{p_2}}{mv \cos \gamma}, \\
\dot{\chi} & = \frac{F_{a_3}}{mv \cos \gamma} + \frac{F_{p_3}}{mv \cos \gamma} + \frac{T_1}{mv},
\end{align*}
\]

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 $f_D$ and the lift forces $f_{l_2}$ and $f_{l_3}$ expressed as follows:

\[
\begin{align*}
\mathbf{f}_D & = -\frac{1}{2} \rho(z) SC_D v_n \mathbf{v}_a, \\
\mathbf{f}_{l_2} & = \frac{1}{2} \rho(z) SC_{L2} v^n a^2 e^v_2, \\
\mathbf{f}_{l_3} & = -\frac{1}{2} \rho(z) SC_{L3} v^n a^2 e^v_3,
\end{align*}
\]

where $C_D$, $C_{L2}$, and $C_{L3}$ are the aerodynamic coefficients, $S$ is the surface of reference, $\rho(z)$ is the density of air and $v_n$ 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. $e^v_2$ and $e^v_3$ are the two unit vectors perpendicular to $\mathbf{v}_a$. Note that if $\mathbf{v}_w = 0$, $e^v_2 = e^b_2$ and $e^v_3 = e^b_3$. The forces $F_{p_1}$, $F_{p_2}$, $F_{p_3}$ include other perturbation forces acting on the aircraft.

For the control of the vehicle, the aerodynamic forces $f_{l_2}$ and $f_{l_3}$ as well as the propulsion forces $(T_1, T_2, T_3)$ are used.

For the environment modeling, the US Standard Atmosphere, 1976 (US-76) can be used. In the lower earth atmosphere (altitude < 35 km), density of air $\rho$ and atmospheric pressure decrease 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 $\mathbf{x}(t) = (\xi^T, v, \gamma, \chi)^T \in \mathbb{X} = \mathbb{R}^6$ be the measurable state of the system, $\mathbf{u} \in \mathbb{U}$ 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

\[
\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}),
\]

where $f$ is the vehicle system model.

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

The path planning algorithm is given a rendezvous set $\mathbb{X}_{\text{goal}} \subset \mathbb{X}_{\text{free}}$. 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

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

where $f^0 : \mathbb{R}^6 \times \mathbb{R}^3 \to \mathbb{R}$ and $k : \mathbb{R} \to \mathbb{R}$ 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 $\dot{x} = f(\mathbf{x}, \mathbf{u})$, that starts at $x_{\text{init}}$, reaches the goal region $\mathbb{X}_{\text{goal}}$, i.e. $\mathbf{x}(0) = x_{\text{init}}$ and $\mathbf{x}(t_f) \in \mathbb{X}_{\text{goal}}$ and minimizes the cost function $J$. Fig. 1: Vehicle model
Algorithm 1 RRT* path planner

Function : build_rrt*(in : K ∈ ℍ, xinit ∈ ℋfree, xgoal ⊂ ℋfree, Δt ∈ ℍ+, out : G)
1:  G ← xinit
2:  cost(xinit) ← 0
3:  i = 0
4:  repeat
5:    xrand ← random_state(ℋfree)
6:    xnew ← rrt*_extend(G, xrand)
7:  until i + + > K
8:  return G

Function : rrt*_extend(in : G, xrand, out : xnew)
9:  V ← G.Node
10:  E ← G.Edge
11:  xnearest ← nearest_neighbor(G, xrand)
12:  (xnew, u*) ← steer(xnearest, xrand)
13:  if collision_free_path(xnearest, xnew) then
14:    V ← V ∪ {xnew}
15:    cost(xnew) ← cost(xnearest) + cost(({xnearest, xnew}))
16:    xnearest ← near_vertices_parents(G, xnew)
17:    E ← best_edge(E, xnear, xnearest, xnew)
18:    xnear ← near_vertices_children(xnew, G)
19:  E ← rewire(E, xnear, xnearest, xnew)
20:  end if
21:  G = (V, E)
22:  return xnew

Function : best_edge(in : E, xnear, xmin, xnew out : E)
23:  for all xnear ∈ Xnear \ {xmin} do
24:    (xnew, u*) ← steer(xnear, xnew)
25:    if collision_free_path(xnear, xnew) and cost(xnew) > cost(xnear) + cost(({xnear, xnew})) then
26:      cost(xnew) ← cost(xnear) + cost(({xnear, xnew}))
27:      xnew ← xnear
28:  end if
29:  end for
30:  E ← E ∪ {xmin, xnew}
31:  return E

Function : rewire(in : E, xnear, xnearest, xnew out : E)
32:  for all xnear ∈ Xnear \ {xnearest} do
33:    (xnear, u*) ← steer(xnew, xnear)
34:    if collision_free_path(xnew, xnear) and cost(xnew) > cost(xnew) + cost(({xnew, xnear})) then
35:      xparent ← parent(xnear)
36:      E ← E ∪ {xparent, xnear}
37:      cost(xnear) ← cost(xnew) + cost(({xnew, xnear}))
38:  end if
39:  end for
40:  return E

III. MOTION PLANNING FRAMEWORK

A. Optimal Rapidly-exploring Random Trees or RRT*

Optimal Rapidly-exploring Random Trees (RRT*) [11], [12] is an incremental method designed to explore efficiently non-convex 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, cost((x1, x2)) be the minimal cost from x1 to x2 according to the specified criterion J defined in equation (3). Let cost(x) also be the total cost to arrive at x, that is cost(x) = cost((xinit, x)).

First, the initial state xinit is added to the tree G. Then, a state xrand ∈ ℋfree is either generated randomly or with help of some heuristics. In this paper, Artificial Potential Fileds (APF) 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 xrand according to a user-defined metric d. This state is called xnearest. In steer function, a control input u* is selected according to the specified criterion, i.e., such that J(tnearest, trand, u*) is minimized.

Then, the system model is integrated from tnearest to trand, to find a new state xnew, that is

\[ x_{\text{new}} = x_{\text{nearest}} + \int_{t_{\text{nearest}}}^{t_{\text{rand}}} f(x, u^*) \, dt. \]

A collision test (collision_free_path function) is performed: if xnew and the path between xnearest and xnew lie in ℋfree then xnew is added in V.

Next, the RRT* algorithm tries to find a better parent for xnew, that is a parent providing a lower cost to xnew than xnearest. The near_vertices_parents function in line 16 will search the tree G for a set of other potential parents in a neighborhood Xnear ⊂ V of xnew (see section III-D). The state xmin ∈ Xnear ∪ {xnearest} that is collision-free and minimizes the cost to xnew is chosen to be its new parent. Therefore, the connecting edge from xmin to xnew 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 Xnear ⊂ V of xnew (see section III-D). For each xnear ∈ Xnear, if the cost to xnear passing by xnew is better than cost(xnear) and the path is collision-free, then the rewire function will replace the existing connecting edge by the connecting edge from xnew to xnear.

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 xrand is generally generated by a uniform distribution in such a way that xrand ∈ ℋfree. In this paper, a biased random state generation using Artificial Potential Fields (APF) [13], [14] is introduced.

The Artificial Potential Field (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 non-reachable.
The APF can be used to direct $x_{\text{rand}}$ to $x_{\text{goal}}$, i.e. the orientation $(\gamma, \chi)$ of $x_{\text{rand}}$ 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 $x_{\text{rand}}$ [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 $f_{\text{APF}}$ used to direct the vehicle is induced by an artificial potential $U \in \mathbb{R}$, i.e. $f_{\text{APF}} = -\nabla U \in \mathbb{R}^3$. The potentials can be defined in several ways according to the characteristics of the mathematical functions. In this paper, the following artificial potentials and vectors are used.

- 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
  \[ U_{\text{init}} = \frac{1}{2} K_{\text{init}} ||\xi - \xi_{\text{init}}||^2, \]
  \[ f_{\text{init}} = K_{\text{init}} (\xi - \xi_{\text{init}}) ||\xi - \xi_{\text{init}}||^{-1}, \]  
where $U_{\text{init}}$, $f_{\text{init}}$ are the repulsive potential and vector field centered at $\xi_{\text{init}}$ and $K_{\text{init}}$ is a constant.

- 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
  \[ U_{\text{goal}} = -\frac{1}{2} K_{\text{goal}} ||\xi - \xi_{\text{goal}}||^2, \]
  \[ f_{\text{goal}} = -K_{\text{goal}} (\xi - \xi_{\text{goal}}) ||\xi - \xi_{\text{goal}}||^{-1}, \]  
where $U_{\text{goal}}$, $f_{\text{goal}}$ are the attractive potential and vector fields centered at $\xi_{\text{goal}}$ and $K_{\text{goal}}$ is a constant.

- 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 $\xi_{\text{obs}}$, denote the center of gravity of an obstacle, $S_{\text{obs}}$ denote the set of the obstacle surface, $d_{\text{obs}}$ denote a maximum distance of influence from the obstacle, $r_{\text{obs}}$ denote the shortest distance of the vehicle from the obstacle, and $t$, $\sigma$, $\lambda$ are unit vectors defined as
  \[ \sigma = \frac{\xi_{\text{goal}} - \xi_{\text{obs}}}{||\xi_{\text{goal}} - \xi_{\text{obs}}||}, \]
  \[ \lambda = \frac{\xi - \xi_{\text{obs}}}{||\xi - \xi_{\text{obs}}||}, \]
  \[ t = \lambda \times (\sigma \times \lambda). \]

**Remark 1:** If $\lambda$ is collinear to $\sigma$, $t$ can be chosen arbitrarily. In that case, the most simple solution is to choose a random $t$ perpendicular to $\lambda$. Other strategies could also be adopted. The vector field function can be defined as
  \[ f_{\text{obs}} = K_{\text{obs}} \left( \frac{d_{\text{obs}} - r_{\text{obs}}}{r_{\text{obs}}} \right) t, \quad r_{\text{obs}} \leq d_{\text{obs}} \]  
  \[ f_{\text{obs}} = 0, \quad r_{\text{obs}} > d_{\text{obs}} \]
where $\xi_{\text{obs}} \in S_{\text{obs}}$ is the nearest point of an obstacle to $\xi$, and $K_{\text{obs}}$ is a constant. Note that for a spherical obstacle, $\lambda = \frac{\xi - \xi_{\text{obs}}}{||\xi - \xi_{\text{obs}}||}$.

Thus, the summation of artificial vector fields expressed as
  \[ f_{\text{APF}} = \sum_{i=1}^{n} f_i = f_{\text{goal}} + f_{\text{init}} + f_{\text{obs}}, \]  
is used as a basis to generate $(\gamma_{\text{rand}}, \chi_{\text{rand}})$ of $x_{\text{rand}}$.

Thus, the random_state function generates a random state $x_{\text{rand}} \in \mathcal{X}_{\text{free}}$ 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 $f_{\text{APF}}$ with a given margin, that is $x_{\text{rand}}$ is chosen in the convex cone illustrated in Fig. 3 pointing toward $f_{\text{APF}}$ with apex $\xi_{\text{rand}}$ and apex angle $2\phi_{\text{APF}}$. This makes our approach differ from [17] as their approach uses $f_{\text{APF}}$ to update the position of the generated $x_{\text{rand}}$ 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 $x_{\text{rand}} \in \mathcal{X}_{\text{goal}}$ with a probability $p$.

### C. nearest_neighbor

The nearest state $x_{\text{near}}$ to $x_{\text{rand}}$ is determined by a metric. The Euclidean metric is often used in such an algorithm. It
D. near_vertices

In the RRT* algorithm described in [11], [12], the near_vertices function uses the same metric function as nearest_neighbor function to determine the neighborhood $X_{\text{near}}$ of the state $x$. This function searches for the vertices in a reachable set containing a ball, centered at $x$, of volume $\lambda_1 \ln(n)/n$ where $n$ is the number of vertices in $V$ and $\lambda_1$ 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 $X_{\text{near}}$ of the state $x$. It selects the first $k$-nearest vertices according to the user-defined metric $d$ and returns them to the set $X_{\text{near}}$. To ensure asymptotic optimality, $k(n)$ is chosen such that $k(n) > \lambda_2 \ln(n)$ where $\lambda_2$ is a positive constant (see Appendix A for more details).

The original near_vertices function is divided into 2 functions in this paper: 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 $x_{\text{near}}$ while the latter one searches for the $k$-nearest vertices from $x_{\text{near}}$ 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-dimensional Dubins’ path in a heterogeneous environment as the user-defined metric $d$ (see [25] for details) is used. This shortest 3D Dubins’ path was specifically developed for missile applications. In the following sections, 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 paper, 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 $v$ 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 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 $C_L_z$ is equal to $C_L_3$ because an axisymmetric missile is considered here.

By simplifying equation (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

$$\begin{align*}
x' &= \frac{dx}{ds} = \cos \gamma \cos \chi, \\
y' &= \frac{dy}{ds} = \cos \gamma \sin \chi, \\
z' &= \frac{dz}{ds} = \sin \gamma, \\
\eta' &= \frac{d\eta}{ds} = \frac{1}{2m} \rho(z) SC \max \mu = c(z) \mu, \\
\chi' &= \frac{d\chi}{ds} = \frac{1}{2m} \rho(z) SC \max \eta \cos \gamma = c(z) \frac{\eta}{\cos \gamma},
\end{align*}$$

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

The simplified air density function can be expressed as

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

where $\rho_0$ 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

$$c(z) = c_0 e^{-z/z_r},$$

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

B. Problem formulation

In this application, $x = (\xi^T, \gamma, \chi)^T \in X = \mathbb{R}^3$ is the measurable state of the system and $u = (\mu, \eta)^T \in U$ is an admissible control input. The set of admissible control inputs is defined as

$$U = \{ u \in \mathbb{R}^2 : ||u|| \leq 1 \}$$
The differential system (11) is rewritten as
\[ x' = f(x, u). \]

The obstacle region \( \mathcal{X}_{\text{obs}} \) 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. \( \mathcal{X}_{\text{free}} = \mathcal{X} \setminus \mathcal{X}_{\text{obs}} \).

The path planning starts at the initial state \( x_{\text{init}} \in \mathcal{X}_{\text{free}} \). The destination of the path planning is given by a rendezvous set \( \mathcal{X}_{\text{goal}} \subset \mathcal{X}_{\text{free}} \) and defined as
\[ \mathcal{X}_{\text{goal}} = \mathbb{P}_{\text{goal}} \times \mathcal{V}_{\text{goal}}, \]
where \( \mathbb{P}_{\text{goal}} \) is the set of the desired arrival positions and \( \mathcal{V}_{\text{goal}} \) is the set of the desired orientations of the vehicle. In this application, they are defined as
\[
\mathbb{P}_{\text{goal}} = \{ \xi_{\text{rdv}} \}, \\
\mathcal{V}_{\text{goal}} = \{ (\gamma, \chi) \in \mathbb{R}^2 : (\gamma, \chi) \in \mathcal{C}(\xi_{\text{rdv}}, \phi_f) \},
\]
where \( x_{\text{rdv}} \) is the rendez-vous state, \( \mathcal{C}(\xi_{\text{rdv}}, \phi_f) \) is the convex cone (see Fig. 3) pointing toward the orientation of the vehicle defined by \( \gamma_{\text{rdv}} \) and \( \chi_{\text{rdv}} \) with apex \( \xi_{\text{rdv}} \) and apex angle \( 2\phi_f \) where \( \phi_f \) 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 \( x(s) : [0, s_f] \rightarrow \mathcal{X}_{\text{free}} \) with \( x = f(x, u) \), that starts at \( x_{\text{init}} \), reaches the goal region \( \mathcal{X}_{\text{goal}} \), i.e. \( x(0) = x_{\text{init}} \) and \( x(s_f) \in \mathcal{X}_{\text{goal}} \) and minimizes the cost function
\[ J = \int_0^{s_f} ds. \]

C. Dubins’ paths for the metric

Interestingly, the shortest 3D 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 \( d \):

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

Thus, for each \( x \in G \), the approach first consists in finding the shortest CS path to the desired final position, i.e. \( \xi_{\text{rand}} = \xi_{\text{rdv}} \). 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 \( \mathcal{X}_{\text{goal}} \).

V. SIMULATION RESULTS

Simulation results are obtained using MATLAB. The scenario shown in Fig. 4 is considered with the following configurations: the initial state \( x_{\text{init}} = (1\text{km}, 1\text{km}, 1\text{km}, \pi/2, 0)^T \),
\[
\mathcal{X}_{\text{goal}} = \mathbb{P}_{\text{goal}} \times \mathcal{V}_{\text{goal}}, \\
\mathbb{P}_{\text{goal}} = \{ \xi_{\text{rdv}} \}, \\
\mathcal{V}_{\text{goal}} = \{ (\gamma, \chi) \in \mathbb{R}^2 : (\gamma, \chi) \in \mathcal{C}(\xi_{\text{rdv}}, 5^\circ) \},
\]
with \( \xi_{\text{rdv}} = (90\text{km}, 90\text{km}, 25\text{km}, 0, −\pi/8)^T \).

In Fig. 4, the arrows represent the orientations of the initial state \( x_{\text{init}} \) and of the rendez-vous state
\[ x_{\text{rdv}} = (\xi_{\text{rdv}}, \gamma_{\text{rdv}}, \chi_{\text{rdv}})^T. \]
\( \mathcal{X}_{\text{obs}} \) is represented by the 3-dimensional 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, 3 different algorithms are considered:
- RRT;
- RRT*;
- RRT* biased by APF with a given margin for the orientation of the randomly generated state \( x_{\text{rand}} \) of \( 10^\circ \) (\( \phi_{\text{APF}} = 10^\circ \)).

For all algorithms, a bias toward the goal consisting in generating \( x_{\text{rand}} \in \mathcal{X}_{\text{goal}} \) with a probability \( p = 0.1 \) is used. Moreover, the Dubins’ path is used for the metric as mentioned in section IV-C.

In the following figures, the exploration tree is represented in light grey 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*
TABLE I: Results of 100 Monte-Carlo simulations within 200 iterations

<table>
<thead>
<tr>
<th>Method</th>
<th>Average iteration for the 1st solution</th>
<th>Average length of the 1st solution</th>
<th>Average length of the final solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>RRT</td>
<td>85</td>
<td>241 km</td>
<td>239 km</td>
</tr>
<tr>
<td>RRT*</td>
<td>82</td>
<td>197 km</td>
<td>185 km</td>
</tr>
<tr>
<td>RRT* biased by APF</td>
<td>46</td>
<td>169 km</td>
<td>155 km</td>
</tr>
</tbody>
</table>

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 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.

VI. CONCLUSIONS

In this paper, an efficient trajectory planning framework for aerial vehicles traveling in 3-dimensional 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 good performance in simulation results. Compared to 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 paper, real-time constraints will
be easier to verify if the algorithm is implemented on board 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 A**

**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, are necessary conditions to ensure asymptotic optimality of the RRT* algorithm. Since the aerial system considered in this paper 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 $\phi_{APF}$ 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 state $x$ on the optimal path are in the convex cone $C(x_{APF}, \phi_{APF})$ (see Fig. 3 for a definition of $C$), where $x_{APF}$ is a state on the vector field at position $\xi$ ($\xi_{APF} = \xi$).

To ensure asymptotic optimality, the near_vertices function need 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 $\lambda_1$ 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 $\lambda_1 \ln(n)/\lambda_0 n$, where $\lambda_0$ 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**