Shortest path for aerial vehicles in heterogeneous environment using RRT

Pawit Pharpatara, Bruno Hérissé, Romain Pepy, Yasmina Bestaoui

Shortest path for aerial vehicles in heterogeneous environment using RRT*

P. Pharpatara, B. Hérissé, R. Pepy, Y. Bestaoui

Abstract— **This paper presents an algorithm for aerial vehicle trajectory generation based on the optimal Rapidly-exploring Random Tree (RRT*). The trajectory generation for the aerial vehicle is a complex path planning problem since the vehicle flies in a heterogeneous environment. The vehicle must also avoid some obstacles or inaccessible zones such as buildings, mountains and even radar detection zones depending on the mission. The RRT* algorithm is used as a basis to find near-optimal solutions for this problem. The shortest Dubins' path in heterogeneous environment is used to compute a distance and a trajectory between two vehicle states. Simulated results show the capability of the algorithm to find a feasible near-optimal trajectory in terms of path length that anticipates future flight conditions, such as the decrease in maneuverability in high altitude. The results also show the advantages over the numerical methods in avoiding obstacles.**

I. INTRODUCTION

Recently, trajectory planning is a high-demand method for the aerial vehicles such as missiles, drones, and future Unmanned Combat Aerial Vehicles (UCAV). An efficient/optimal trajectory known *a priori* before the mission can increase the probability to complete the mission gradually. Moreover, if changes in mission occur, the vehicles can use remaining time to find the alternative solutions.

The purpose of this paper is to develop an efficient trajectory planning algorithm for an aerial vehicle traveling in 2-dimensional vertical plane while avoiding obstacles.

Trajectory planning in the vertical plane is a challenging problem with many constraints since the vehicle travels in a heterogeneous environment where the air density decreases with altitude. Most aerial vehicles depend on the aerodynamic forces to control their course in the air. As a consequence, the maneuverability of the aerial vehicles decreases with altitude. Moreover, trajectory planning while avoiding obstacles makes the problem more complex and difficult to solve by most existing methods.

One way to find a trajectory for an aerial vehicle is to use classical closed-loop guidance laws to execute the trajectory directly. For example, in missile guidance, many closed-loop optimal guidance laws were proposed [2], [16], [3], [7], [6], [14], [17]. Among these guidance laws, kappa guidance [13] is certainly the most known. The kappa guidance uses optimal control theory on a simplified system and obtains a simple closed-form solution which is easy to implement in the real time system. However, it relies on some restrictive approximations. Moreover, control limitations (saturations) are difficult to satisfy. For such complex systems and missions, the optimal problem needs to be considered globally.

Numerical methods such as pseudo-spectral optimization [18] [15] with a good initialization can be used to find globally optimal solutions. However, there is a trade-off between the precision of the solution and the computational effort. Moreover, obstacles induce state constraints that are very difficult to consider with such methods.

Model Predictive Control (MPC), sometimes called Receding Horizon Predictive Control (RHPC), is one of the numerical methods. The MPC is an alternative way to find a trajectory of an aerial vehicle. It considers problems locally within the calculation horizon. The advantage over global optimization is that solutions can be obtained rapidly and is possible to implement to the real time system. The MPC uses a reference trajectory or waypoints as a reference to generate the control sequence. The MPC uses a vehicle model to predict the future behavior of the vehicle within the optimization horizon. It takes the state errors and control effort into account to minimize an objective function. Even though the MPC considers the future conditions and environment, they are only within the receding horizon. If the reference trajectory is too far from the feasible trajectory, the MPC might have difficulties in finding a feasible trajectory and ends up in some obstacles. Thus, it is preferable that future conditions and environment are considered globally.

Many studies in the robotic field, especially in path planning and control theory aim to find trajectories in complex environments. The sampling-based path planning methods, such as Rapidly-exploring Random Tree (RRT) [11] or Probabilistic Roadmap Methods (PRM) [10], offer solutions for trajectory shaping in complex environments. These are usually used for path planning of a Dubins' car in environments cluttered by obstacles [12]. The main advantage of these techniques is that even a complex system can be considered without need of approximations [19]. Moreover, the application of this method for the interceptor missile trajectory planning is demonstrated in [20]. However, the obtained solution is not optimal in terms of path length.

The optimal RRT (RRT*) [8], [9] was developed to overcome the optimization problem of the RRT algorithm. In this paper, the RRT* algorithm is studied to find an efficient and near-optimal trajectory in terms of path length for an aerial vehicle flying in heterogeneous environment. A metric function based on Dubins' paths in heterogeneous environment [5] is used to compute a distance and a tra-

P. Pharpatara, B. Hérissé and R. Pepy are with Onera - The French Aerospace Lab, Palaiseau, France (email: pawit.pharpatara@onera.fr, bruno.herisse@onera.fr), romain.pepy@onera.fr.

Y. Bestaoui is with IBISC, Université d'Evry-Val-d'Essonne, Evry, France (e-mail: Yasmina.Bestaoui@ufrst.univ-evry.fr).

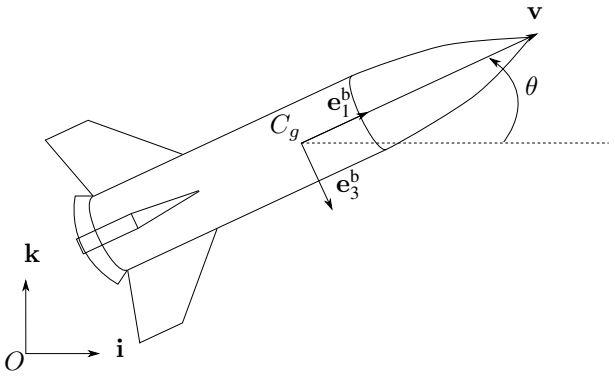


Fig. 1: Vehicle model

jectory between two vehicle states. Terminal constraints are defined with respect to the rendezvous point. The obtained results demonstrate the potential of path planning algorithms in finding a collision-free near-optimal solution for aerial vehicle in heterogeneous environment while other methods may fail to find a proper solution. The obtained trajectory can be used as a reference trajectory to facilitate trajectory tracking by the MPC.

This paper is divided into four parts. First, the environment and system modeling are presented in section II. Then, section III introduces the RRT* path planner along with the shortest path in heterogeneous environment for the aerial vehicle. Then, some simulated results are shown and analyzed in section (section IV). Finally, some concluding remarks are made in the last section.

II. Problem statement

A. Environment modeling

The environment is considered heterogeneous in a 2-dimensional vertical plane because of variation of air density ρ, decreasing exponentially with altitude. The environment model can be expressed as:

$$\rho = \rho_0 e^{-z/z_r} \tag{1}$$

where $\rho_0 = 1.225\text{km/m}^3$ is the air density at standard atmosphere at the sea level and $z_r = 7.5\text{km}$.

B. System modeling

In this paper, a simplified model of an aerial vehicle is used. It is modeled as a rigid body maneuvering in a vertical 2-dimensional plane. Two 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 $(\mathbf{i}, \mathbf{k})$ and a body-fixed frame $\mathcal{B}$ attached to the vehicle at its center of mass C_g with the vector basis $(\mathbf{e}_1^b, \mathbf{e}_3^b)$. $\mathbf{v}$ is the translational velocity of the vehicle in $\mathcal{I}$ which is considered in the direction of $\mathbf{e}_1^b$. Position and velocity defined in $\mathcal{I}$ are denoted $\boldsymbol{\xi} = (x, z)^\top \in \mathbb{R}^2$ and $\mathbf{v} = (v, \theta)^\top \in \mathbb{R}^2$ where $v > 0$ is the magnitude of the velocity and θ is the orientation of the velocity as known as the flight path angle.

To eliminate all the external factor to the problem, a zero wind assumption is applied and the axis of the propulsion of the vehicle is fixed and is in the direction of $\mathbf{e}_1^b$. Then, the translational velocity $\mathbf{v}$ is assumed to coincide with the apparent velocity. Since the air density decreases with altitude and the aerial vehicle depends on the aerodynamic forces to perform a turn using the control surfaces, the maximal curvature, or minimal turning radius, that the vehicle can perform depends on the altitude of the vehicle.

Thus, the dynamics of an aerial vehicle can be written as

$$\dot{x} = v\cos\theta, \tag{2}$$

$$\dot{z} = v\sin\theta, \tag{3}$$

$$\dot{\theta} = vc(z)u, \quad |u| < 1 \tag{4}$$

where $u \in \mathbb{R}$ is the control input ($u \in [-1, 1]$) and $c(z) \in \mathbb{R}_+$ is the maximum curvature that can be performed by the vehicle at the altitude z. In this paper, we are interested in the shortest path problem. Therefore, the optimal control problem consists in minimizing the cost function

$$s_f = \int_0^{t_f} v dt \tag{5}$$

where s_f is the final path length and t_f is the final time.

Since we are interested in the minimum length path, a change of variables from time dt to curvilinear abscissa $ds = vdt$ is used. Thus, the dynamics can be rewritten as:

$$x' = \frac{dx}{ds} = \cos\theta, \tag{6}$$

$$z' = \frac{dz}{ds} = \sin\theta, \tag{7}$$

$$\theta' = \frac{d\theta}{ds} = c(z)u, \quad |u| < 1 \tag{8}$$

Thus, the dynamics of the forward velocity does not need to be specified in this studies.

C. Problem formulation

Let $\mathbf{x}(t) = (\boldsymbol{\xi}, \theta) \in \mathbb{X} = \mathbb{R}^3$ be the measurable state of the system, $u \in \mathbb{U} = [-1, 1]$ be an admissible control input and consider the differential system

$$\mathbf{x}' = f(\mathbf{x}, u), \tag{9}$$

where f is defined in section II-B equations (6), (7), and (8).

$\mathbb{X} = \mathbb{R}^3$ 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 the *obstacle region*.

The initial state of the system is $\mathbf{x}_{\text{init}} \in \mathbb{X}_{\text{free}}$.

The path planning algorithm is given a rendezvous point $\mathbf{x}_{\text{rdv}} = (\boldsymbol{\xi}_{\text{rdv}}, \theta_{\text{rdv}})$. In order to achieve its mission, the vehicle has to reach a goal set $\mathbb{X}_{\text{goal}} \subset \mathbb{X}_{\text{free}}$, shown in Fig. 2, defined as

$$\begin{aligned} \mathbb{X}_{\text{goal}} &= \mathbb{P}_{\text{goal}} \times \mathbb{V}_{\text{goal}}, \\ \mathbb{P}_{\text{goal}} &= \{\boldsymbol{\xi} \in \mathbb{R}^2 : \boldsymbol{\xi} = \boldsymbol{\xi}_{\text{rdv}}\}, \\ \mathbb{V}_{\text{goal}} &= \{\theta \in \mathbb{R} : \theta \in \mathbb{C}(\boldsymbol{\xi}, \theta_{\text{rdv}}, \phi_f)\}, \end{aligned} \tag{10}$$

where $\mathbb{C}(\boldsymbol{\xi}, \theta_{\text{rdv}}, \phi_f)$ is the convex cone pointing toward the flight path angle θ_{rdv} from the ground with apex $\boldsymbol{\xi}$ and apex angle $2\phi_f$, and ϕ_f is a maximal orientation error which can be tolerated by the mission objective.

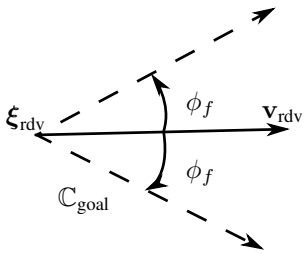


Fig. 2: $\mathbb{X}_{\text{goal}}$

The motion planning problem is to find a collision free trajectory $\mathbf{x}(s) : [0, s_f] \rightarrow \mathbb{X}_{\text{free}}$ with $\mathbf{x}' = f(\mathbf{x}, u)$, that starts at $\mathbf{x}_{\text{init}}$ and reaches the goal region, *i.e.* $\mathbf{x}(0) = \mathbf{x}_{\text{init}}$ and $\mathbf{x}(s_f) \in \mathbb{X}_{\text{goal}}$. Moreover, the obtained trajectory must achieve near-optimality in terms of path length.

III. Motion planning framework

A. Optimal Rapidly-exploring Random Trees or RRT*

Optimal Rapidly-exploring Random Trees (RRT*) [8], [9] is an incremental method designed to efficiently explore non-convex high-dimensional spaces by growing the search tree toward large Voronoi areas [21] 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($\mathbf{x}$) array be the total cost to arrive at $\mathbf{x}$ and c($\{(\mathbf{x}_1, \mathbf{x}_2)\}$), calculated by a metric d in this paper, be the cost from $\mathbf{x}_1$ to $\mathbf{x}_2$.

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 randomly chosen (see section III-B). The `nearest_neighbor` function searches the tree G for the nearest vertex to $\mathbf{x}_{\text{rand}}$ according to a metric d (see section III-C). This state is called $\mathbf{x}_{\text{nearest}}$. In `steer` function, a control input u is selected according to a specified criterion. Equations (6), (7), and (8) are then integrated for a fixed distance Δs (or fixed time Δt) (see section III-D). The newly found state is called $\mathbf{x}_{\text{new}}$ along with the applied control input u. Then, 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 to V (see section III-E).

The `near_vertex_parents` function in line 16 will search the tree G for the set $\mathbb{X}_{\text{near}}$ of near vertices $\mathbf{x}_{\text{near}}$ of $\mathbf{x}_{\text{new}}$ (see section III-F) in order to determine the parent of $\mathbf{x}_{\text{new}}$. The parent $\mathbf{x}_{\text{min}}$ and its connecting edge to $\mathbf{x}_{\text{new}}$ will be determined by `best_edge` function (see section III-G).

After the new vertex and its connecting edge is added to the tree, the `near_vertex_children` function in line 18 will search the tree G for the set $\mathbb{X}_{\text{near}}$ of near vertices $\mathbf{x}_{\text{near}}$ of the state $\mathbf{x}_{\text{new}}$ (see section III-F) in order to determine if $\mathbf{x}_{\text{new}}$ can be a better parent of any vertices in the tree (this is optional depending on method used to determine the near vertices). The suitability of the parent is decided by its cost or objective function. Then, if there are better paths passing by $\mathbf{x}_{\text{new}}$ to any vertices $\mathbf{x}_{\text{near}} \in \mathbb{X}_{\text{near}}$ in the tree, the algorithm will disconnect the old paths and generate the new paths of

Algorithm 1 RRT* path planner

Function : build_rrt*(in : $K \in \mathbb{N}$, $\mathbf{x}_{\text{init}} \in \mathbb{X}_{\text{free}}$, $\mathbb{X}_{\text{goal}} \subset \mathbb{X}_{\text{free}}$, $\Delta s \in \mathbb{R}^+$, 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** $i++ > K$ or ($\mathbf{x}_{\text{new}} \neq$ null and $\mathbf{x}_{\text{new}} \in \mathbb{X}_{\text{goal}}$)
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: $(\mathbf{x}_{\text{new}}, u) \leftarrow$ steer($\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{rand}}, \Delta s$)
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($\mathbf{x}_{\text{nearest}}$) + $c(\{(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{new}})\})$
16: $\quad \mathbb{X}_{\text{near}} \leftarrow$ near_vertex_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_vertex_children($\mathbf{x}_{\text{new}}, G$) (optional)
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** $\mathbf{x}_{\text{near}} \in \mathbb{X}_{\text{near}}$ **do**
24: $\quad$ **if** collision_free_path($\mathbf{x}_{\text{near}}, \mathbf{x}_{\text{new}}$) and cost($\mathbf{x}_{\text{new}}$) > cost($\mathbf{x}_{\text{near}}$) + $c(\{(\mathbf{x}_{\text{near}}, \mathbf{x}_{\text{new}})\}$ **then**
25: $\quad\quad \mathbf{x}_{\text{min}} \leftarrow \mathbf{x}_{\text{near}}$
26: $\quad\quad$ cost($\mathbf{x}_{\text{new}}$) $\leftarrow$ cost($\mathbf{x}_{\text{near}}$) + $c(\{(\mathbf{x}_{\text{near}}, \mathbf{x}_{\text{new}})\}$
27: $\quad$ **end if**
28: **end for**
29: $E \leftarrow E \cup \{(\mathbf{x}_{\text{min}}, \mathbf{x}_{\text{new}})\}$
30: **return** E

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

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

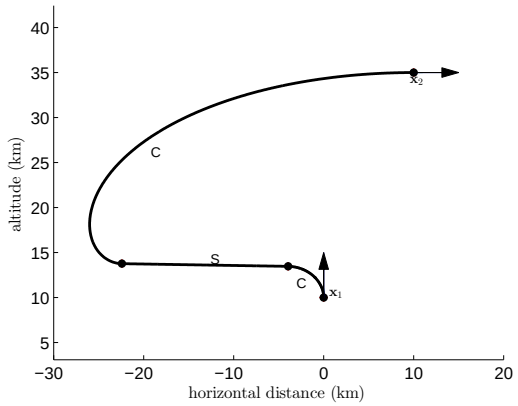


Fig. 3: An example of a Dubins' path of CSC type in heterogeneous environment

the tree in `rewire` function (see section III-H).

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

B. `random_state`

The random state $\mathbf{x}_{\text{rand}}$ is generated by the uniform distribution in such a way that $\mathbf{x}_{\text{rand}} \in \mathbb{X}_{\text{free}} = \mathbb{X} \setminus \mathbb{X}_{\text{obs}}$ where $\mathbb{X}_{\text{obs}} = \mathbb{P}_{\text{obs}} \times \mathbb{V}_{\text{obs}}$ is defined by

$$\begin{aligned} \mathbb{P}_{\text{obs}} = \{\boldsymbol{\xi} \in \mathbb{R}^2 : \quad & \boldsymbol{\xi} \text{ is the position surrounded} \\ & \text{by bounders of obstacles}\} \\ \mathbb{V}_{\text{obs}} = \{\theta \in \mathbb{R}\} & \end{aligned} \tag{11}$$

In this function, a bias toward the goal can be introduced to reduce the number of generated states in order to reach $\mathbb{X}_{\text{goal}}$. This bias, called RRT-*GoalBias* [12], consists in choosing $\mathbf{x}_{\text{rand}} \in \mathbb{X}_{\text{goal}}$ with a probability p.

C. `nearest_neighbor`

$\mathbf{x}_{\text{near}}$ is defined as the nearest state to $\mathbf{x}_{\text{rand}}$ according to a specified metric d. In this paper, the shortest Dubins' path in heterogeneous environment [5] where the curvature of the vehicle decreases exponentially with altitude is used to determine the nearest neighbor. These Dubins' paths have the advantage over the original Dubins' paths [4][1] because they are more realistic for the aerial vehicle traveling in the vertical plane. In [5], it was shown with the same system model as equations (6), (7), and (8) that, analogously to the original Dubins' paths, shortest paths are combinations of curves of maximum curvature C and straight lines S. Therefore, for the problem considered in this paper, paths of CSC type are considered (see an illustration of a CSC path in Fig. 3).

However, when $\mathbf{x}_{\text{rand}}$ is chosen in $\mathbb{X}_{\text{goal}}$ by the biased algorithm discussed in section III-B, it is more interesting to consider the shortest path between any $\mathbf{x} \in G$ and $\mathbb{X}_{\text{goal}}$ than considering a single $\mathbf{x}_{\text{rand}} \in \mathbb{X}_{\text{goal}}$. Indeed,there can exist the 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 arrival orientation within the arrival cone, *i.e.* $\theta_f \leqslant \phi_{\text{pip}} \pm \phi_f$, the shortest path is the CS path. If no CS path arrives in $\mathbb{X}_{\text{goal}}$, the shortest path is necessarily a CSC path whose arrival orientation is one of the extremities of the arrival cone, *i.e.* $\theta_f \in \{\phi_{\text{pip}} + \phi_f, \phi_{\text{pip}} + \phi_f\}$.

Thus, for each $\mathbf{x} \in \mathbb{G}$, the approach first consists in finding the shortest CS path to the desired final position, *i.e.* $\mathbf{x}_{\text{rand}} \in \mathbb{X}_{\text{goal}}$. 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 $\mathbb{X}_{\text{goal}}$.

To improve the performance of the RRT* algorithm, the collision test is also applied in the `nearest_neighbor` function to verify the Dubins' paths if they are collision free. If they are not, the lengths of those Dubins' paths are considered infinity.

D. `steer`

The `steer` function is a node expansion used to move the vehicle from one state to another. It can be randomly generated or computed using a specific criterion. In this paper, it uses the control input u corresponding to the metric d mentioned in section III-C. The control input $u = \pm 1$ for curves of maximum curvature C and $u = 0$ for straight line S. The `steer` function applies the control input u with the system model equations (6), (7), and (8) for distance interval Δs. Then, a new state $\mathbf{x}_{\text{new}}$ is obtained.

In this paper, the RRT-*connect* [12] is applied to improve the performance of the algorithm. Instead of integrating the system for a single step Δs, it keeps integrating the system for ϵ steps if there is no collision along the path. Normally, it is not suitable for non-holonomic problems because it places more faith in the metric and the metric designing of non-holonomic problems is difficult. However, since the metric d mentioned in section III-C is well designed for this problem, the RRT-*connect* can be applied.

E. `collision_test`

The `collision_test` function verifies that the path between two states lies in $\mathbb{X}_{\text{free}}$. If it is collision-free, the path between $\mathbf{x}_{\text{near}}$ and $\mathbf{x}_{\text{new}}$ is added to the tree G.

F. `near_vertex`

In the original RRT* algorithm, the `near_vertex` function uses an euclidean distance to find all the states in the neighborhood of the state $\mathbf{x}$. It is a circle of radius R centered at $\mathbf{x}$. The radius of this circle is fixed at the beginning and will decrease in function of numbers of vertices in the tree. However, this method is not suitable for our framework using the Dubins' paths in heterogeneous environment to determine the distance between two states in the state space. Because if R is too small, there will be very few feasible or reasonable paths connecting two states due to the limited maximum curvature of the vehicle. Moreover, if R is too large, the computational effort will be expensive.

Thus, in this paper, the k-nearest neighbors algorithm is used to determine near vertices of the state $\mathbf{x}$ in order to choose suitable candidate vertices according to the metric d used in `nearest_neighbor` function. This algorithm selects the first k-nearest vertices according to the metric d and returns them to the set $\mathbb{X}_{\text{near}}$.

There are two `near_vertex` functions in this paper: `near_vertex_parents` function in line 16, and `near_vertex_children` function in line 18. The `near_vertex_parents` function searches for the first k-nearest vertices to arrive at $\mathbf{x}_{\text{new}}$ while the latter searches for the first k-nearest vertices from $\mathbf{x}_{\text{new}}$ to other vertices. In case of using the euclidean distance, both functions are the same and the latter is not required.

G. `best_edge`

The `best_edge` function determines a parent of $\mathbf{x}_{\text{new}}$ by calculating a total cost to arrive at $\mathbf{x}_{\text{new}}$ passing by each $\mathbf{x}_{\text{near}}$, *i.e.* cost($\mathbf{x}_{\text{near}}$)+c($\{(\mathbf{x}_{\text{near}}, \mathbf{x}_{\text{new}})\}$). cost($\mathbf{x}_{\text{near}}$) is the actual cost, in this paper, representing the total distance of the trajectory from $\mathbf{x}_{\text{init}}$ to $\mathbf{x}_{\text{near}}$ and c($\{(\mathbf{x}_{\text{near}}, \mathbf{x}_{\text{new}})\}$) is the shortest distance of Dubins' path in heterogeneous environment connecting $\mathbf{x}_{\text{near}}$ and $\mathbf{x}_{\text{new}}$.

If the total cost is less than the total cost of $\mathbf{x}_{\text{new}}$, *i.e.* cost($\mathbf{x}_{\text{new}}$), then $\mathbf{x}_{\text{near}}$ becomes the new parent $\mathbf{x}_{\text{min}}$ of $\mathbf{x}_{\text{new}}$. After each $\mathbf{x}_{\text{near}} \in \mathbb{X}_{\text{near}}$ has been verified, the edge connecting $\mathbf{x}_{\text{nearest}}$ and $\mathbf{x}_{\text{new}}$ is added to E.

H. `rewire`

Opposing to the `best_edge` function, the `rewire` function is used to determine if $\mathbf{x}_{\text{new}}$ can be a new parent of any vertices of the tree. By using the same methodology as the `best_edge` function, if there are paths from $\mathbf{x}_{\text{init}}$ to each $\mathbf{x}_{\text{near}} \in \mathbb{X}_{\text{near}} \setminus \mathbf{x}_{\text{nearest}}$ passing by $\mathbf{x}_{\text{new}}$ with less cost than the actual cost($\mathbf{x}_{\text{near}}$), the parent of $\mathbf{x}_{\text{near}}$ is changed to $\mathbf{x}_{\text{new}}$. The edge connecting $\mathbf{x}_{\text{near}}$ to its previous parent is disconnected and replaced by the edge connecting $\mathbf{x}_{\text{new}}$ to $\mathbf{x}_{\text{near}}$. In other words, the tree is rewired. The cost of all state having $\mathbf{x}_{\text{new}}$ as a new parent are also updated in `updatecost` function in line 36.

IV. Results and analysis

Two scenarios are analyzed with the initial state $\mathbf{x}_{\text{init}} = (0\text{km}, 0\text{km}, \pi/2)$,

$$\begin{aligned} \mathbb{P}_{\text{goal}} &= \{\boldsymbol{\xi} \in \mathbb{R}^2 : \|\boldsymbol{\xi} - \boldsymbol{\xi}_{\text{pip}}\| < 500\text{m}\}, \\ \mathbb{V}_{\text{goal}} &= \{\theta \in \mathbb{R} : \theta \in \mathbb{C}(\boldsymbol{\xi}, \theta_{\text{pip}}, \pi/8)\}, \end{aligned} \tag{12}$$

with $\boldsymbol{\xi}_{\text{pip}} = (30\text{km}, 5\text{km})$ and $\mathbf{v}_{\text{pip}}$ with the flight path angle $-\pi/12$ to the ground for both scenarios.

In this paper, the bias $p = 0.1$ mentioned in section III-B, the integration distance $\Delta s = 1$km, $\varepsilon = 3$ steps mentioned in section III-D and $k = 10$ in *near_vertex* function.

In the following figures, $\mathbb{X}_{\text{goal}}$ is represented as a point with two dashed lines, $\mathbb{X}_{\text{obs}}$ is represented by space surrounded by red dashed curves, the exploration tree is represented in grey and thick solid curve is the solution found by the RRT* algorithm.

In the first scenario, the obstacle is a 180° panning ground radar centered at (10km,0km) with the 8km detection radius. Fig. 4 shows one of the simulation results for scenario 1. They show the improvement of the solutions while the number of iterations increases. Fig. 4(c) shows the shortest length solution with 33.9km found within 300 iterations. Results from 100 Monte-Carlo simulations show that the first solution is acquired around 69^{th} iteration and the average shortest path is 34.2km long. As we can see from the improvement of the obtained solutions over time, the search continues after the first solution is found which is the advantage of this algorithm.

Complexity of the problem increases in the second scenario. Two obstacles are introduced to demonstrate the capability of the RRT* algorithm. This time the obstacles are fixed direction ground radars whose origins are (-8km, 0km) and (19km, 0km).

A simulation result for scenario 2 is shown in Fig. 5. After 400 iterations, several solutions were found but just two solutions are presented in Fig. 5(a). The first obtained solution is represented in thick dotted curve and the last obtained solution is represented in thick solid curve with a length of 43.5km. As it is not clear that these trajectories respect the final constraints in Fig. 5(a), Fig. 5(b) represents the enlarged area around $\mathbb{X}_{\text{goal}}$ which shows that the vehicle makes a small turn at the end of the trajectory to arrive in $\mathbb{X}_{\text{goal}}$ while respecting the final constraints. According to the 100 Monte-Carlo simulations, the mean iteration where the first solution is obtained is 152 and the mean path length solution is 44.2km.

V. Conclusion and perspectives

The RRT* algorithm together with the Dubins' paths in heterogeneous environment is capable of finding a solution for the trajectory planning of the aerial vehicle in vertical plane while avoiding obstacles. The solutions keeps on improving during the remaining time that results in finding a near-optimal solution.

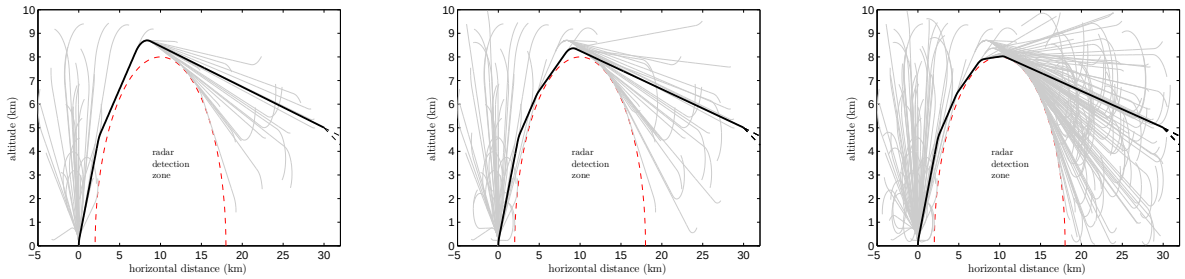
Even if the trajectory obtained from this framework is not totally executable by the complete system because of the simplified model. This trajectory can be used as a reference trajectory to facilitate the control using, for example, the MPC algorithm.

This algorithm shows promising results that the RRT* is suitable for collision-free trajectory generation for aerial vehicles. For an instance, the shortest Dubins' path in heterogeneous environment was proven to be optimal only in 2-dimensional plane. Thus, to be able to solve this problem using this framework, the suitable metric function in 3-dimensional plane must be developed and can be a subject of interest in the future work.

References

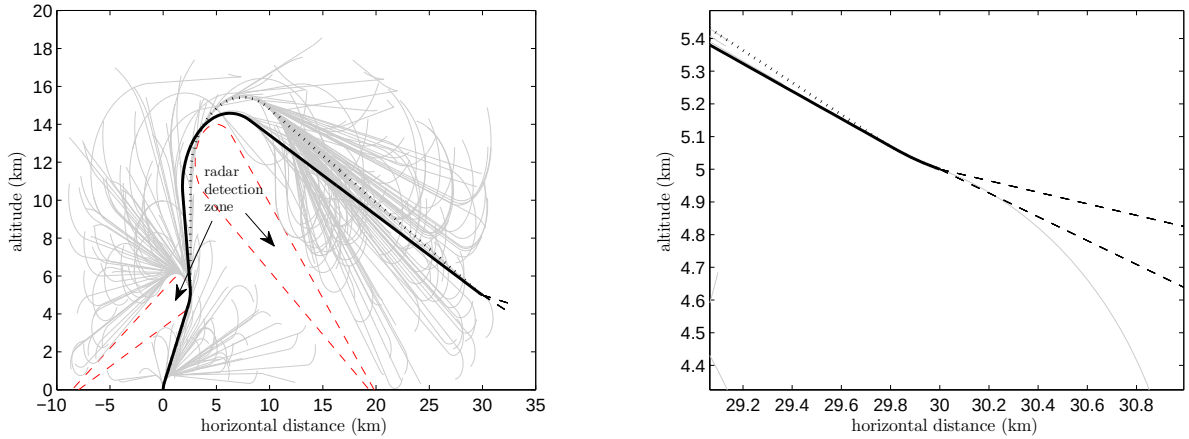
[1] J. D. Boissonnat, A. Cérézo, and J. Leblond. Shortest paths of bounded curvature in the plane. Technical report, Institut National de Recherche en Informatique et en Automatique, 1991.

[2] V. H. L. Cheng and N. K. Gupta. Advanced midcourse guidance for air-to-air missiles. *Journal of Guidance, Control, and Dynamics*, 9(2):135–142, 1986.



(a) 100 iterations (b) 150 iterations (c) 300 iterations

Fig. 4: Exploration tree expansion and results for scenario 1



(a) 400 iterations (b) Enlarged image at $\mathbb{X}_{\text{goal}}$ area

Fig. 5: Simulation result for scenario 2

[3] J. J. Dougherty and J. L. Speyer. Near-optimal guidance law for ballistic missile interception. *Journal of Guidance, Control, and Dynamics*, 20(2):355–362, 1997.

[4] L. E. Dubins. On curves of minimal length with a constraint on average curvature and with presribed initial and terminal position and tangents. *American Journal of Mathematics*, 79:497–516, 1957.

[5] B. Hérissé and R. Pepy. Shortestest paths for the dubins' vehicle in heterogeneous environments. *In Proceedings of the IEEE Conference on Decision and Control*, pages 4504–4509, 2013.

[6] F. Imado and T. Kuroda. Optimal guidance system against a hypersonic targets. In *Proceedings of the AIAA Guidance, Navigation and Control Conference*, 1992.

[7] F. Imado, T. Kuroda, and S. Miwa. Optimal midcourse guidance for medium-range air-to-air missiles. *Journal of Guidance, Control, and Dynamics*, 13(4):603–608, 1990.

[8] S. Karaman and E. Frazzoli. Optimal kinodynamic motion planning using incremental sampling-based methods. *IEEE Conference on Decision and Control*, pages 7681–7687, 2010.

[9] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30:846–894, 2011.

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

[11] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[12] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[13] C. F. Lin. *Modern Navigation Guidance and Control Processing*. Prentice-Hall, Inc., 1991.

[14] C. F. Lin and L. L. Tsai. Analytical solution of optimal trajectory-shaping guidance. *Journal of Guidance, Control, and Dynamics*, 10(1):61–66, 1987.

[15] J. A. Lukacs and O. A. Yakimenko. Trajectory-shape-varying missile guidance for interception of ballistic missiles during the boost phase. *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2007.

[16] P. K. Menon and M. M. Briggs. Near-optimal midcourse guidance for air-to-air missiles. *Journal of Guidance, Control, and Dynamics*, 13(4):596–602, 1990.

[17] B. Newman. Strategic intercept midcourse guidance using modified zero effort miss steering. *Journal of Guidance, Control, and Dynamics*, 19(1):107–112, 1996.

[18] J.-W. Park, M.-J. Tank, and H.-G. Sung. Trajectory optimization for a supersonic air-breathing missile system using pseudo-spectral method. *International Journal of Aeronautical and Space Sciences*, 10:112–121, 2009.

[19] R. Pepy, A. Lambert, and H. Mounier. Reducing navigation errors by planning with realistic vehicle model. In *Proceedings of the IEEE Intelligent Vehicles Symposium*, pages 300–307, 2006.

[20] P. Pharpatara, R. Pepy, B. Hérissé, and Y. Bestaoui. Missile trajectory shaping using sampling-based path planning. In *the IEEE/RCJ International Conference on Intelligent Robots and Systems*, pages 2533–2538, Tokyo, Japan, 2013.

[21] G. Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. *Journal fur die Reine und Angewandte Mathematik*, 133:97–178, 1907.