2014 International Conference on Unmanned Aircraft Systems (ICUAS)
May 27-30, 2014. Orlando, FL, USA

RRT-Based Path Planning for Fixed-Wing UAVs with Arrival Time and Approach Direction Constraints*

Dasol Lee and David Hyunchul Shim, *Member, IEEE*

Abstract— **This paper proposes a path planning algorithm based on rapidly-exploring random trees (RRTs) for fixed-wing unmanned aerial vehicles (UAVs). The algorithm uses a pre-defined motion primitive set to extend the tree, and can be designed to reflect the dynamic capability of a target aircraft. The proposed method produces collision-free, dynamically feasible, and smooth curves. Furthermore, the algorithm can determine the approach direction of each initial and goal position and the arrival time at the goal position by generating an inertial speed command. Estimated dynamic information, including roll, heading, required thrust, and aerodynamic force are also output. The algorithm is validated by performing several simulations and comparing its performance with the response of a nonlinear six-degrees-of-freedom simulation. The results show that this algorithm can be successfully applied to fixed-wing UAV applications.**

I. INTRODUCTION

Producing an efficient flight path is a crucial problem for unmanned aerial vehicle (UAV) operations, and is consequently a widely researched topic. This planning problem can be difficult, because the generated path should obviously be obstacle-free and satisfy other conditions, such as smoothness and dynamic feasibility. Path planning problems are a fundamental area of research in the field of robotics, and a number of algorithms have been proposed [1].

Sampling-based planning algorithms such as probabilistic roadmaps (PRMs) or rapidly-exploring random trees (RRTs) [2] are frequently used to calculate an obstacle-free path. In particular, RRT is widely accepted, and many variants have been developed to handle different situations. For example, a randomized planner that can handle kinodynamic constraints was proposed in [3], and asymptotically optimal planners such as RRT* and PRM* have been reported [4].

However, sampling-based planners can produce jerky, unnatural paths and cause overactuated control during operations. Therefore, smoothing techniques or spline methods have been applied to RRT-based planners [5], and these improve the quality of the generated path.

The application of smoothing techniques to RRT-based planners has led to their widespread use to generate paths for aerial vehicles for which smoothness is necessary. For fixed-wing aircraft applications, an RRT-based path and maneuver planning algorithm was proposed in [6], and a B-spline-based motion planning algorithm for unmanned helicopters in dense 3D environments was presented in [7]. An algorithm dealing with uncertainty was proposed in [8] for UAV applications, and this has been shown to have robust characteristics on uncertain environments.

An interesting research topic involves satisfying arrival time and approach direction constraints toward a specific position, and these constraints are very important to fixed-wing UAVs. Lim *et al*. [9] proposed a UAV guidance law that can determine the arrival time and approach direction to a specific position, and this work is applicable to obstacle-free aerial areas.

The constraints mentioned above are also considered in our proposed algorithm. To determine a collision-free path, an RRT planner is used as a planning algorithm. Based on this, a motion primitive set is used to extend the tree. This motion primitive set can be designed to reflect the dynamic limits of the target aircraft, so that the proposed algorithm can generate dynamically feasible paths and satisfy the specified approach direction and arrival time constraints.

The rest of this paper is organized as follows. First, an RRT-based path planning algorithm that uses a motion primitive set to extend the tree is presented in Section II. Section III describes simulation results for various scenarios, and these are compared with the response from a nonlinear six-degrees-of-freedom (6-DoF) UAV model. Section IV summarizes our conclusions, and the Appendix presents data including the aircraft specification and aerodynamic coefficients.

*This work was supported by Agency for Defense Development(ADD) under the contract UD130041JD

D. Lee is with the Department of Aerospace Engineering, KAIST, Daejeon, South Korea (e-mail: dasol@kaist.ac.kr).

D. H. Shim is with the Department of Aerospace Engineering, KAIST, Daejeon, South Korea (e-mail: hcshim@kaist.ac.kr).

II. ALGORITHM

The proposed algorithm is based on an RRT algorithm specialized for fixed-wing aircraft applications in 2-D space.

Initial and goal configuration are generated by defining the position and heading vector for each configuration. To satisfy both configurations, and specify the arrival time to the goal point, some modifications are made to the basic RRT algorithm.

The input arguments are the altitude map, initial and goal configurations, arrival time, and aircraft specification. The algorithm produces an obstacle-free path, inertial speed command that the aircraft should follow to achieve the specified arrival time, and certain estimated dynamic information including roll angle, heading angle, required thrust, and required aerodynamic force with respect to time.

There are two major advantages of the proposed algorithm. First, the constraints on heading angle and arrival time can be satisfied, and second, the estimated dynamic information can be used to determine the dynamic feasibility for a specific aircraft. Table I gives the required aircraft specification for the algorithm, and Table II gives a summary of the inputs and outputs of the proposed algorithm.

A. Algorithm Overview – Algorithm 1

The main body of the proposed algorithm is represented in **Algorithm 1**. From lines 3–7, a total of $N_{sampling}$ samples are taken to expand the tree using a pre-defined Bézier curve-based motion primitive set. The function **GetCost** on line 8 calculates the cost of all vertices in goal area $\mathbf{V}_{goalarea}$, and the resulting cost $\mathbf{J}$ and other arguments including $\mathbf{A}$, graph $\mathbf{G}$, $\mathbf{X}_{goal}$, and $T_{arrival}$ are used to produce the final results from the function **GetResults**.

Algorithm 1: Main

1: $\mathbf{A} \leftarrow Aircraft\_Specification$; $i \leftarrow 0$;
2: $\mathbf{V} \leftarrow \{\mathbf{X}_{init}\}$; $\mathbf{V}_{goalarea} \leftarrow \varnothing$; $\mathbf{E} \leftarrow \varnothing$;
3: **while** $i < N_{sampling}$ **do**
4: $\mathbf{G} \leftarrow (\mathbf{V}, \mathbf{V}_{goalarea}, \mathbf{E})$;
5: $\mathbf{p}_{rand} \leftarrow \mathbf{GetSample}(i)$; $i \leftarrow i+1$;
6: $(\mathbf{V}, \mathbf{V}_{goalarea}, \mathbf{E}) \leftarrow \mathbf{ExtendTree}(\mathbf{G}, \mathbf{p}_{rand})$;
7: **end while**
8: $\mathbf{J} \leftarrow \mathbf{GetCost}(\mathbf{G}, \mathbf{X}_{goal})$;
9: $(\mathbf{P}_{path}, V_{iner,cmd}, \mathbf{D}_{esti}) \leftarrow \mathbf{GetResults}(\mathbf{A}, \mathbf{G}, \mathbf{J}, \mathbf{X}_{goal}, T_{arrival})$;

TABLE I. REQUIRED AIRCRAFT SPECIFICATION

Notation	Explanation
m_{total}	Total mass
S_{ref}	Reference area
C_{L_0}	Lift coefficient at zero angle of attack
C_{L_α}	Lift curve slope
C_{D_0}	Drag coefficient at zero lift coefficient
K	Equal to $(C_D - C_{D_0}) / C_L^{\,2}$

TABLE II. INPUT AND OUTPUT OF THE ALGORITHM

Notation	Explanation
$\mathbf{A}$	Aircraft specification
$\mathbf{p}_{init}, \mathbf{p}_{goal}$	Initial position vector, goal position vector
$\mathbf{e}_{\psi_{init}}, \mathbf{e}_{\psi_{goal}}$	Initial heading vector, goal heading vector
$\mathbf{X}_{init}, \mathbf{X}_{goal}$	Initial configuration, goal configuration $\mathbf{X}_{init} = \{\mathbf{p}_{init}, \mathbf{e}_{\psi_{init}}\}, \mathbf{X}_{goal} = \{\mathbf{p}_{goal}, \mathbf{e}_{\psi_{goal}}\}$
$T_{arrival}$	Arrival time from $\mathbf{p}_{init}$ to $\mathbf{p}_{goal}$
$\mathbf{P}_{path}$	Obstacle free path
$V_{iner,cmd}$	Inertial speed command
$\mathbf{D}_{esti}$	Estimated dynamic information

B. Bézier Curve Based Motion Primitive Set

In the proposed algorithm, each vertex has a position and heading angle, and the edges of the tree are generated by Bézier curve-based motion primitives. When the tree is extended, the continuity of the heading angle is ensured. The main reason why Bézier curves are used to extend the tree is that a smooth flight path is essential to fixed-wing aircraft.

The basic cubic Bézier curve $\mathbf{B}_{01}(t_{01})$, which can be generated using four point vectors, is represented by the equation below; the output is illustrated in Fig. 1.

$$\mathbf{B}_{01}(t_{01}) = (1-t_{01})^3 \mathbf{P}_i + 3(1-t_{01})^2 t_{01} \mathbf{P}_1 + 3(1-t_{01}) t_{01}^{\,2} \mathbf{P}_2 + t_{01}^{\,3} \mathbf{P}_f \quad (1)$$

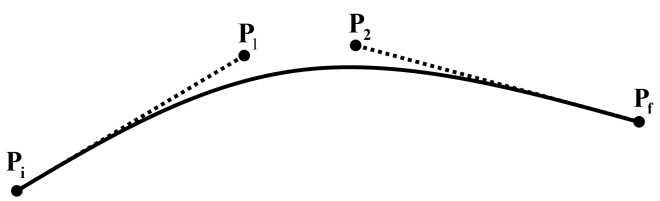

Figure 1. Cubic Bézier Curve

where t_{01} varies from 0 to 1. This basic cubic Bézier curve is modified so that its time property satisfies the arrival time at the goal position. If we substitute t / T_{travel} into t_{01}, we obtain a motion primitive having time property $\mathbf{B}(t)$ with initial position $\mathbf{P}_i$ at time $t = 0$ and final position $\mathbf{P}_f$ at time $t = T_{travel}$. $\mathbf{B}(t)$ is represented by the following equation:

$$\mathbf{B}(t) = \mathbf{P}_i + \frac{3t(\mathbf{P}_1 - \mathbf{P}_i)}{T_{travel}} + \frac{3t^2(\mathbf{P}_2 - 2\mathbf{P}_1 + \mathbf{P}_i)}{T_{travel}^{\ 2}} + \frac{t^3(3\mathbf{P}_1 - 3\mathbf{P}_2 + \mathbf{P}_f - \mathbf{P}_i)}{T_{travel}^{\ 3}} \quad (2)$$

where $\mathbf{P}_i$ is defined by a vertex position vector at which the motion primitive set starts, and the other three vectors $\mathbf{P}_1$, $\mathbf{P}_2$, and $\mathbf{P}_f$ are defined by the following equations:

$$\mathbf{P}_1 = \mathbf{P}_i + D_1 \times \mathbf{e}_{\psi_{\mathbf{P}_i}} \quad (3)$$

$$\mathbf{P}_2 = \mathbf{P}_f - D_1 \times \mathbf{e}_{\psi_{\mathbf{P}_f}} \quad (4)$$

$$\mathbf{P}_f = \mathbf{P}_i + R_{primitive} \times \mathbf{e}_{\psi_{to\mathbf{P}_f}} \quad (5)$$

where $R_{primitive}$ is a motion primitive scale factor that should be adaptively selected to consider the environment density. Therefore, $R_{primitive}$ will take a small value in a dense altitude map. $\mathbf{e}_{\psi_{\mathbf{P}_i}}$ is a unit heading vector for a vertex placed at $\mathbf{P}_i$, and $\mathbf{e}_{\psi_{\mathbf{P}_f}}$, $\mathbf{e}_{\psi_{to\mathbf{P}_f}}$ are defined as:

$$\mathbf{e}_{\psi_{to\mathbf{P}_f}} = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 \\ \sin\theta_1 & \cos\theta_1 \end{bmatrix} \times \mathbf{e}_{\psi_{\mathbf{P}_i}} \quad (6)$$

$$\mathbf{e}_{\psi_{\mathbf{P}_f}} = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 \\ \sin\theta_2 & \cos\theta_2 \end{bmatrix} \times \mathbf{e}_{\psi_{\mathbf{P}_i}} \quad (7)$$

If the aircraft operates with a constant inertial speed, D_1 can be formulated as:

$$D_1 = \frac{\left\|\dot{\mathbf{B}}(t=0)\right\| T_{travel}}{3} \approx \frac{R_{primitive}}{3} \quad (8)$$

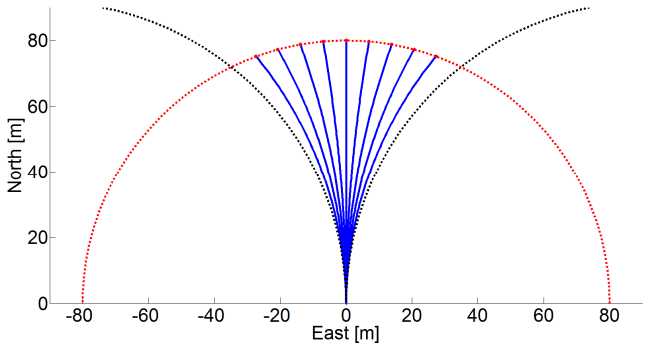

Figure 2. Bézier Curve Based Motion Primitive Set

To generate the motion primitive set, several parameters are needed. First, $\mathbf{P}_i$ and $\mathbf{e}_{\psi_{\mathbf{P}_i}}$ are obtained from the vertex, and $R_{primitive}$, $\boldsymbol{\theta}_{1,set}$, and $\boldsymbol{\theta}_{2,set}$ are pre-defined by the user. T_{travel} does not affect the shape of the motion primitive in geometric space, and this value is assigned by the **CalResults** function in line 7 of **Algorithm 4**.

When designing the motion primitive set, the minimum turning radius $R_{turn,\min}$ should be considered to guarantee dynamic feasibility for a given aircraft. The black dotted circle in Fig. 2 represents this minimum turning radius. This $R_{turn,\min}$ can be defined for the worst case of maximum inertial speed, and is formulated as below:

$$R_{turn,\min} = \frac{V_{\max}^{\ 2}}{A_{lat\max}} \quad (9)$$

where $A_{lat,\max}$ is the maximum lateral acceleration and $V_{\max}$ is the maximum inertial speed.

One example of the motion primitive set generated by this approach is shown in Fig. 2, and the necessary parameter settings are shown below.

Motion Primitive Parameter Values for Fig. 2	
$\boldsymbol{\theta}_{1,set}$	$\{\theta \mid \theta = -25 + 5i, i < 10\}$ [deg]
$\boldsymbol{\theta}_{2,set}$	$\{\theta \mid \theta = -50 + 10i, i < 10\}$ [deg]
$R_{primitive}$	80 m
$V_{\max}$	30 m/sec
$A_{lat\max}$	9.81 m/sec^2

i is positive integer

Algorithm 2: ExtendTree $(\mathbf{G}, \mathbf{p}_{rand})$

1: $\mathbf{V}' \leftarrow \mathbf{V}$; $\mathbf{V}'_{goalarea} \leftarrow \mathbf{V}_{goalarea}$; $\mathbf{E}' \leftarrow \mathbf{E}$;
2: $\mathbf{x}_{nearest} \leftarrow$ **Nearest** $(\mathbf{V}', \mathbf{p}_{rand})$;
3: **if** $\|\mathbf{p}_{nearest} - \mathbf{p}_{goal}\| < R_{goal}$ **then**
4: **return** $\mathbf{G}' = (\mathbf{V}', \mathbf{V}'_{goalarea}, \mathbf{E}')$;
5: **end if**
6: $\mathbf{p}_{ref} \leftarrow$ **Steer** $(\mathbf{p}_{nearest}, \mathbf{p}_{rand})$;
7: $\mathbf{X}_{set} \leftarrow$ **GetPrimitiveSet** $(\mathbf{x}_{nearest})$;
8: $\mathbf{x}_{new} \leftarrow$ **GetNewVertex** $(\mathbf{p}_{ref}, \mathbf{X}_{set})$;
9: **if ObstacleFree** $(\mathbf{x}_{nearest}, \mathbf{x}_{new})$ **then**
10: $\mathbf{V}' \leftarrow \mathbf{V}' \cup \{\mathbf{x}_{new}\}$;
11: $\mathbf{E}' \leftarrow \mathbf{E}' \cup \{\mathbf{x}_{nearest}, \mathbf{x}_{new}\}$;
12: **if** $\|\mathbf{p}_{new} - \mathbf{p}_{goal}\| < R_{goal}$ **then**
13: $\mathbf{V}'_{goalarea} \leftarrow \mathbf{V}'_{goalarea} \cup \{\mathbf{x}_{new}\}$;
14: **end if**
15: **end if**
16: **return** $\mathbf{G}' = (\mathbf{V}', \mathbf{V}'_{goalarea}, \mathbf{E}')$;

C. *Tree Extension – Algorithm 2*

The **ExtendTree** function extends the tree based on the motion primitive set, and each motion primitive represents an edge of the tree. Therefore, the child node position is $\mathbf{P}_f$ and its parent node is placed at $\mathbf{P}_i$.

The input arguments required for the **ExtendTree** function are a graph $\mathbf{G}$ and randomly generated sample position vector $\mathbf{p}_{rand}$. The **Nearest** function on line 2 returns the nearest vertex $\mathbf{x}_{nearest}$, and its position vector $\mathbf{p}_{nearest}$ is closest to $\mathbf{p}_{rand}$. If the distance between $\mathbf{p}_{nearest}$ and $\mathbf{p}_{goal}$ is less than R_{goal}, the **ExtendTree** function is terminated because there is no need to extend the tree within the goal area (defined as a circle centered at $\mathbf{p}_{goal}$ with radius R_{goal}).

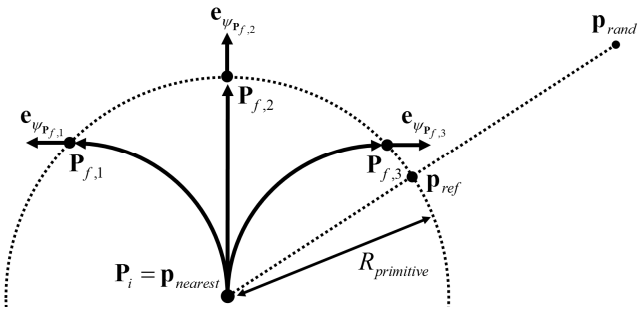

Figure 4. Endpoint Configuration of Motion Primitive Set

If $\mathbf{p}_{nearest}$ is outside the goal area, **ExtendTree** continues to execute, and the **Steer** function will return $\mathbf{p}_{ref}$, which is calculated as follows.

$$\text{If } \|\mathbf{p}_{rand} - \mathbf{p}_{nearest}\| > R_{primitive} \text{ then,}$$

$$\mathbf{p}_{ref} = \mathbf{p}_{nearest} + \frac{\mathbf{p}_{rand} - \mathbf{p}_{nearest}}{\|\mathbf{p}_{rand} - \mathbf{p}_{nearest}\|} R_{primitive} \quad (10)$$

$$\text{If } \|\mathbf{p}_{rand} - \mathbf{p}_{nearest}\| \le R_{primitive} \text{ then,}$$

$$\mathbf{p}_{ref} = \mathbf{p}_{rand} \quad (11)$$

The function **GetPrimitiveSet** on line 7 returns each endpoint configuration of motion primitive set $\mathbf{X}_{set}$, which is described by (see Fig. 4):

$$\mathbf{X}_{set} = \{(\mathbf{P}_{f,i}, \mathbf{e}_{\psi_{\mathbf{P}_{f,i}}}) \mid i = primitive\_index\} \quad (12)$$

To evaluate $\mathbf{X}_{set}$, the position vector of the nearest vertex $\mathbf{p}_{nearest}$ is assigned to $\mathbf{P}_i$, and the heading vector of the nearest vertex $\mathbf{e}_{\psi_{\mathbf{P}nearest}}$ is used as $\mathbf{e}_{\psi_{\mathbf{P}_i}}$ in (3) to form vector $\mathbf{P}_1$. Every motion primitive has the same $\mathbf{P}_i$ and $\mathbf{P}_1$ vectors, but $\mathbf{P}_2$

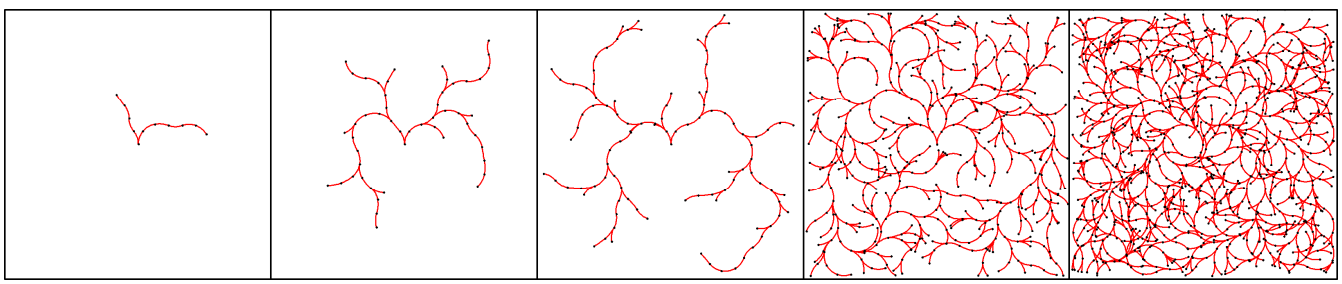

Figure 3. Tree Extension Based on Motion Primitive Set
from the left, 10 nodes (a), 50 nodes (b), 100 nodes (c), 500 nodes (d), and 1000 nodes (e) are extended respectively.

Algorithm 3: GetCost $(\mathbf{G}, \mathbf{X}_{goal})$;

1: $\mathbf{J} \leftarrow \varnothing$;
2: **for** $i = 1$ **to Size** $(\mathbf{V}_{goalarea})$ **do**
3: $\mathbf{X}_{goalarea,i} \leftarrow \mathbf{V}_{goalarea}(i)$;
4: $X_{\text{cost,i}} \leftarrow$ **GetVertexCost** $(\mathbf{X}_{goalarea,i}, \mathbf{X}_{goal})$;
5: $\mathbf{J} \leftarrow \mathbf{J} \cup \{\mathbf{X}_{goalarea,i}, X_{\text{cost,i}}\}$;
6: **for end**
7: $\mathbf{J} \leftarrow$ **Sort** $(\mathbf{J})$; *// Sorting by lowest cost first*
8: **return** $\mathbf{J}$;

and $\mathbf{P}_f$ are computed by the pre-defined $\boldsymbol{\theta}_{1,set}$, $\boldsymbol{\theta}_{2,set}$, and (4)–(7).

Line 8 of **Algorithm 2** returns $\mathbf{x}_{new}$ using the **GetNewVertex** function, and its position vector $\mathbf{p}_{new}$ is the closest to $\mathbf{p}_{ref}$ among the $\mathbf{P}_{f,i}$ vectors. For example, in Fig. 4, $\mathbf{x}_{new}$ will be $\{\mathbf{P}_{f,3}, \mathbf{e}_{\psi_{\mathbf{P}_{f,3}}}\}$.

Vertex $\mathbf{x}_{new}$ and edge $\{\mathbf{x}_{nearest}, \mathbf{x}_{new}\}$ are added if the path from $\mathbf{x}_{nearest}$ to $\mathbf{x}_{new}$ is obstacle-free (lines 10–11 of **Algorithm 2**), and if $\mathbf{p}_{new}$ is in the goal area, $\mathbf{x}_{new}$ is added to $\mathbf{V}'_{goalarea}$ as a new vertex. **ExtendTree** is terminated by returning an updated graph $\mathbf{G}'$, and Fig. 3 illustrates the process of tree extension based on motion primitives.

D. Cost Evaluation for $\mathbf{V}_{goalarea}$ *– Algorithm 3*

The function **GetCost** (**Algorithm 3**) calculates each vertex cost $X_{\text{cost,i}}$. This will return a low value if the vertex can be connected smoothly to the goal vertex. $X_{\text{cost,i}}$ is defined so as to evaluate this 'smoothness to the goal vertex,' and is represented as:

$$X_{\text{cost,i}} = \frac{c_1}{1 + \mathbf{e}_{\psi_{togoal,i}} \bullet \mathbf{e}_{\psi_{\mathbf{P}_{goal}}}} + \frac{c_2}{1 + \mathbf{e}_{\psi_{\mathbf{P}_{goalarea,i}}} \bullet \mathbf{e}_{\psi_{ref,i}}} + c_3 \times tree\_depth(\mathbf{X}_{goalarea,i}) - (c_1 + c_2)/2 \quad (13)$$

$$\mathbf{e}_{\psi_{togoal}} = \frac{\mathbf{p}_{goal} - \mathbf{p}_{goalarea}}{\left\| \mathbf{p}_{goal} - \mathbf{p}_{goalarea} \right\|} \quad (14)$$

$$\mathbf{e}_{\psi_{ref}} = 2 \times \mathbf{e}_{\psi_{togoal}} - \mathbf{e}_{\psi_{\mathbf{P}_{goal}}} \quad (15)$$

$$\mathbf{X}_{goalarea} = \{\mathbf{p}_{goalarea}, \mathbf{e}_{\psi_{\mathbf{P}_{goalarea}}}\} \quad (16)$$

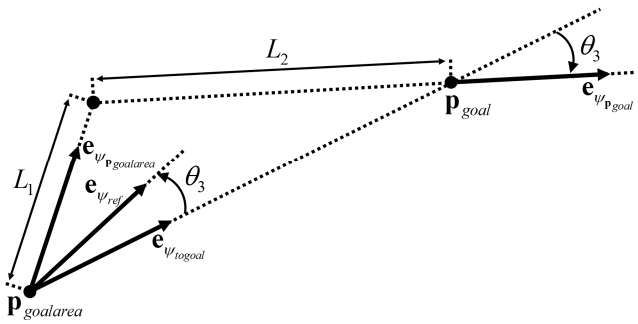

Figure 5. Cost Evalueation of the Vertex within Goal Area

where c_1, c_2, and c_3 are cost weighting factors and $\mathbf{X}_{goalarea}$ is a vertex located in the goal area.

The goal point vector $\mathbf{p}_{goal}$ can be formulated as below (see Fig. 5):

$$\mathbf{p}_{goal} = \mathbf{p}_{goalarea} + L_1 \mathbf{e}_{\psi_{\mathbf{P}_{goalarea}}} + L_2 \mathbf{e}_{\psi_{\mathbf{P}_{goal}}} \quad (17)$$

The connection between $\mathbf{p}_{goal}$ and $\mathbf{p}_{goalarea}$ requires high curvature if L_2 is negative, and low curvature for positive L_2. For this reason, we let the vertex cost take an infinite value for negative L_2. The main process of **Algorithm 3** is cost evaluation by the function **GetVertexCost**. This function calculates each vertex cost, and the output is stored in $\mathbf{J}$ (lines 2–6). On line 7, the cost results are sorted according to $X_{\text{cost,i}}$, and then the **GetCost** function terminates by returning the sorted results in $\mathbf{J}$.

E. Algorithm Results – Algorithm 4

The **GetResults** function in **Algorithm 4** returns the output of the algorithm. First, **ObstacleFree** checks for obstacles, and if the link between $\mathbf{X}_{goalarea,i}$ and $\mathbf{X}_{goal}$ can be generated, vertex $\mathbf{X}_{goal}$ and edge $\{\mathbf{X}_{goalarea,i}, \mathbf{X}_{goal}\}$ are added to the tree. The results are calculated by **CalResults**, then the algorithm generates its output (lines 4–8 of **Algorithm 4**). However, if the collision check for $\mathbf{X}_{goalarea,i}$ fails, the algorithm returns a FAIL message (line 11 of **Algorithm 4**).

The initial output of the **GetResults** function is $\mathbf{P}_{path}$, which can be separated into two parts. The first is the motion primitive part, which connects

Algorithm 4: GetResults $(\mathbf{A},\mathbf{G},\mathbf{J},\mathbf{X}_{goal},T_{arrival})$;

```
V' ← V;   E' ← E;
for i = 1 to Size(J) do
    X_goalarea,i ← J(i);   // Index number of lowest cost is equal to 1.
    if ObstacleFree(X_goalarea,i, X_goal) then
        V' ← V' ∪ {X_goal};
        E' ← E' ∪ {X_goalarea,i, X_goal}
        (P_path, V_iner,cmd, D_esti) ← CalResults(A, E', T_arrival);
        return (P_path, V_iner,cmd, D_esti)
    end if
end if
return FAIL;   // Algorithm fails to find the path.
```

$\mathbf{X}_{init}$ to $\mathbf{X}_{goalarea}$ and is generated by D_1 and T_1 for T_{travel}. The second part is the final edge connecting $\mathbf{X}_{goalarea}$ to $\mathbf{X}_{goal}$, which can be generated using D_2 instead of D_1 and T_2 for T_{travel}, where D_2 is defined in a similar way to D_1. The arrival time at the goal position $T_{arrival}$ and D_2 are formulated as:

$$D_2 = \frac{\left\|\mathbf{p}_{goal} - \mathbf{p}_{goalarea}\right\|}{3} \tag{18}$$

$$T_{arrival} = N_{prim}T_1 + T_2 \tag{19}$$

where N_{prim} is the number of primitives from $\mathbf{X}_{init}$ to $\mathbf{X}_{goalarea}$. To ensure velocity continuity at $\mathbf{X}_{goalarea}$, the condition below should be satisfied.

$$\left\|\dot{\mathbf{B}}(t=T_1)\right\| = \left\|\dot{\mathbf{B}}(t=0)\right\| \Longleftrightarrow \frac{D_1}{T_1} = \frac{D_2}{T_2} \tag{20}$$

Then, T_1 and T_2 are represented by:

$$T_1 = \frac{T_{arrival}D_1}{N_{prim}D_1 + D_2}, \quad T_2 = \frac{D_2}{D_1}T_1 \tag{21}$$

Finally, $\mathbf{P}_{path}$ can be calculated from $\mathbf{P}_i$, $\mathbf{P}_f$, $\mathbf{e}_{\psi_{\mathbf{P}_i}}$, $\mathbf{e}_{\psi_{\mathbf{P}_f}}$, D_1 or D_2, and T_1 or T_2, as summarized in Table III and related illustration is Fig. 6.

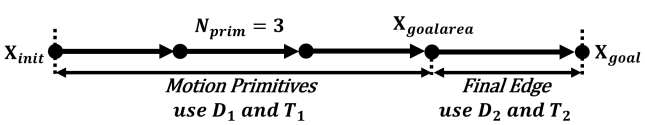

Figure 6. Simple Illustration of the Genterated Path, $\mathbf{P}_{path}$

TABLE III. PARAMETERS REQUIRED TO GENERATE PATH

Required parameter to generate $\mathbf{P}_{path}$	Parameter source, Related equation
Motion Primitive Part, from $\mathbf{X}_{init}$ ***to*** $\mathbf{X}_{goalarea}$	
$\mathbf{P}_i$, $\mathbf{P}_f$	Edge of the tree, E Equations (2), (3), (4)
$\mathbf{e}_{\psi_{\mathbf{P}_i}}$, $\mathbf{e}_{\psi_{\mathbf{P}_f}}$	Edge of the tree, E Equations (3), (4)
D_1	By definition, $D_1 = R_{primitive}/3$ Equation (3), (4)
$\mathbf{P}_1$, $\mathbf{P}_2$	By calculation using (3), (4)
T_1	By calculation using (21)
Final Edge Part, from $\mathbf{X}_{goalarea}$ ***to*** $\mathbf{X}_{goal}$	
$\mathbf{P}_i$, $\mathbf{P}_f$	From edge of $\{\mathbf{X}_{goalarea}, \mathbf{X}_{goal}\}$ Equation (2), (3), (4)
$\mathbf{e}_{\psi_{\mathbf{P}_i}}$, $\mathbf{e}_{\psi_{\mathbf{P}_f}}$	From edge of $\{\mathbf{X}_{goalarea}, \mathbf{X}_{goal}\}$ Equation (3), (4)
D_2	By definition of (18) Equation (3), (4)
$\mathbf{P}_1$, $\mathbf{P}_2$	By calculation using (3), (4)
T_2	By calculation using (21)

The second output of the **GetResults** function is the inertial speed command $V_{iner,cmd}$ that an aircraft should track to achieve the desired arrival time. $V_{iner,cmd}$ is defined as the mean value of $\dot{\mathbf{B}}$, calculated as:

$$V_{iner,cmd} = \frac{\sum_{n=1}^{N_{prim}} \int_0^{T_1} \left\|\dot{\mathbf{B}}_n(t)\right\| dt + \int_0^{T_2} \left\|\dot{\mathbf{B}}_f(t)\right\| dt}{T_{arrival}} \tag{22}$$

where $\dot{\mathbf{B}}_n$ is the first derivative of $\mathbf{B}_n$, which represents the n-th edge, and $\dot{\mathbf{B}}_f$ is the first derivative of the final edge $\mathbf{B}_f$.

The final output of the **GetResults** function is a series of dynamic information. The required aerodynamic force $F_{req,aero}$ is defined by:

$$\mathbf{F}_{req,aero} = m_{total}(\ddot{\mathbf{B}} - \mathbf{g}) - \left[m_{total}(\ddot{\mathbf{B}} - \mathbf{g}) \bullet \mathbf{e}_{\psi} \right] \mathbf{e}_{\psi} \tag{23}$$

$$\mathbf{e}_{\psi} = \frac{\dot{\mathbf{B}}}{\left\|\dot{\mathbf{B}}\right\|} \tag{24}$$

where $\mathbf{g}$ is the acceleration of gravity, and the drag force can be estimated as:

$$C_L = \frac{2\left\|\mathbf{F}_{req,aero}\right\|}{\rho\left\|\dot{\mathbf{B}}\right\|^2 S_{ref}} \tag{25}$$

$$C_D = C_{D_0} + KC_L{}^2 \tag{26}$$

$$\mathbf{F}_{drag} = -\left[\frac{1}{2}\rho\left\|\dot{\mathbf{B}}\right\|^2 S_{ref} C_D\right]\mathbf{e}_\psi \tag{27}$$

where ρ is the air density. The total required force $\mathbf{F}_{req,total}$ and required thrust force $\mathbf{F}_{req,thrust}$ are defined by the following equations:

$$\mathbf{F}_{req,total} = m_{total}(\ddot{\mathbf{B}} - \mathbf{g}) - \mathbf{F}_{drag} \tag{28}$$

$$\mathbf{F}_{req,thrust} = \mathbf{F}_{req,total} - \mathbf{F}_{req,aero} \tag{29}$$

The roll and heading angles are computed from the direction of $\mathbf{F}_{req,aero}$ and $\mathbf{e}_\psi$, respectively.

III. SIMULATION

Simulations were performed to confirm the efficacy of the proposed algorithm. These consisted of two parts.

First, we focused on the obstacle-free path and inertial speed command $V_{iner,cmd}$, and varied these along with the initial position, goal heading angle, and arrival time to the goal position. The second part was intended to identify the validity of $\mathbf{D}_{esti}$ by comparing it to the results of a nonlinear 6-DoF simulation. The nonlinear equation describing aircraft motion was taken from [10]-[11], and the aircraft specifications and aerodynamic coefficients are summarized in the Appendix.

The nonlinear path-following guidance method proposed in [12] was implemented to follow a lateral path. This guidance produces lateral acceleration commands, and these were converted directly to roll angle commands by the equation for a coordinated turn. Altitude, inertial speed, and roll angle controllers were all implemented by PID controller in this simulation environment.

The scaled altitude map used in this simulation contains real terrain data. The parameter settings related to the algorithm, including the motion primitive design parameter, number of samples, initial and goal configurations, are now summarized for each simulation case.

A. Simulation Part I

The proposed algorithm generates $\mathbf{P}_{path}$ and $V_{iner,cmd}$ to satisfy the initial and goal configurations and the arrival time to the goal position. To validate the output, several simulations were carried out, as summarized below.

Simulation cases I and II have the same parameter values, except for those of the initial and goal heading. Figs. 7–10 show the results for case I, and Figs. 11–14 show those for case II. The red line in Fig. 7-(a) and Fig. 11-(a) represent the obstacle-free path $\mathbf{P}_{path}$, and the goal area of radius R_{goal} is represented as a dotted black circle. Green points in Fig. 7-(a) and Fig. 11-(a) denote positions where the motion primitives are linked, and blue lines in Fig. 7-(b) and Fig. 11-(b) represent edges of the graph $\mathbf{G}$.

The generated path $\mathbf{P}_{path}$ satisfies the initial and goal heading conditions, as can be confirmed by Fig. 9 and Fig. 13. Fig. 8 and Fig. 12 show that the arrival time at the goal position is satisfied for both simulation cases I and II.

Simulation Part I, Case I, Parameter Values	
$\mathbf{X}_{init}$	{200m, 50m, -1m, 135deg} North, East, Down, Heading
$\mathbf{X}_{goal}$	{500m, 1500m, -1m, 180deg}
$T_{arrival}$	85sec
$R_{primitive}$, R_{goal}	80m, 160m
$\boldsymbol{\theta}_{1,set}$	$\{\theta \mid \theta = -25 + 5i, i < 10\}$ [deg] i = positive integer
$\boldsymbol{\theta}_{2,set}$	$\{\theta \mid \theta = -50 + 10i, i < 10\}$ [deg]
$V_{\max}$, $A_{lat,\max}$	30m/sec, 9.81m/sec^2
c_1 , c_2 , c_3	2, 4, 1
$N_{sampling}$	50000
Altitude Map	Lake Mead, USA, 1:10 Scale

Simulation Part I, Case II, Parameter Values	
$\mathbf{X}_{init}$	{200m, 50m, -1m, 90deg}
$\mathbf{X}_{goal}$	{500m, 1500m, -1m, 135deg}
Other parameter values are same as simulation part I, case I.	

Simulation Part I, Case III, Parameter Values	
$\mathbf{X}_{init}$	{200m, 50m, -1m, 90deg}
$\mathbf{X}_{goal}$	{500m, 1500m, -1m, 135deg}
$T_{arrival}$	60sec
Other parameter values are same as simulation part I, case I.	

The inertial speed command $V_{iner,cmd}$ to achieve $T_{arrival}$ is shown as a red line on Fig. 10 and Fig. 14, and its value is calculated from (22), where $\left\|\dot{\mathbf{B}}(t)\right\|$ is represented as a blue line. The main reason we calculate $V_{iner,cmd}$ is that the regulation is much easier and more stable for an inertial speed controller, rather than the high-bandwidth tracking that occurs when an inertial speed controller tracks $\left\|\dot{\mathbf{B}}(t)\right\|$.

If the arrival time is changed, a different value of $V_{iner,cmd}$ is produced, and its accuracy can be confirmed by comparing simulation cases II and III. In this case, the new $V_{iner,cmd}$ can be calculated by re-executing the **GetResults** function on line 9 of **Algorithm 1**, which means that there is no need to re-extend the tree. Therefore, the only different argument in the **GetResults** function is the updated arrival time $T_{arrival}$.

Fig. 15 represents the resulting path for an arrival time of 60 s, and $V_{iner,cmd}$ is shown as a red line on Fig. 16. This value has increased because the arrival time in case III was decreased from 85 to 60 s.

B. Simulation Part II

The obstacle-free paths and inertial speed commands were used as the inputs to the nonlinear 6-DoF simulation; related parameter values are summarized below. The results show that the $\mathbf{P}_{path}$ and $V_{iner,cmd}$ generated by our method are appropriate for the guidance law and inertial speed controller to arrive at the goal position at $T_{arrival}$. The generated values of $\mathbf{D}_{esti}$ are accurate, as shown by Figs. 20–23. Fig. 17 shows the resulting path, and the red line in Fig. 18 is the inertial speed response to the commanded $V_{iner,cmd}$, shown as a black line. To follow a lateral path, guidance law produces a roll angle command, shown as a black line in Fig. 20, and its response is represented as a red line. The inertial speed and roll angle are well controlled, and the path produced by the proposed algorithm is smooth and within the dynamic boundary. Therefore, the simulation response track the commanded path well, and the arrival time at the goal position is successfully achieved, as shown in Fig. 19. In the case of a longitudinal path, the altitude controller maintains an altitude command, and these results are shown in Fig. 24, where the red line is the response and the black line is the specified altitude.

Simulation Part II, Parameter Values	
$\mathbf{X}_{init}$	{45m, 1500m, -20m, -45deg}
$\mathbf{X}_{goal}$	{850m, 250m, -20m, -90deg}
Altitude Map	Colorado River, USA, 1:15 Scale
Other parameter values are same as simulation part I, case I.	

A useful property of the proposed algorithm is that it can estimate the roll and heading angle of an aircraft during flight. Fig. 20 shows the estimated value, which is a blue line that is close to a red line, which represents the simulation results, and this estimated value is calculated by the direction of the required aerodynamic force vector, $\mathbf{F}_{req,aero}$. The heading angle estimation is represented as a blue line in Fig. 21 and its simulation response is very close to the estimated value.

A major advantage of the proposed algorithm is that it can estimate the required aerodynamic force and thrust. The blue line in Fig. 22 represents the estimated required aerodynamic force, and it is well estimated, except at some points where the sign of the roll angle changes. In the case of the estimated required thrust, it is a jerky curve because this estimation is based on the magnitude of $\dot{\mathbf{B}}$. However, an advantage is that the simulation response, which is represented as a red line in Fig. 23, exists between the minimum and maximum values of the required thrust. Therefore, we can determine that the actual required thrust will be close to the mean value of the estimated required thrust.

The elapsed time and simulation environment, including the simulation computer specification and simulation program, are summarized in below.

TABLE IV. ELAPSED TIME AND SIMULATION ENVIRONMENT

Part I, Case I	83.32 [sec]
Part I, Case II	80.67 [sec]
Part I, Case III	0.26 [sec] (only the **GetResults** function which is line 9 of **Algorithm 1** is executed)
Part II	91.34 [sec]
Simulation Computer Specification	Intel Core i7-2600 CPU @ 3.4 GHz 4GB RAM Microsoft Windows 7 64bit
Simulation Program	MATLAB

Simulation Part I, Case I

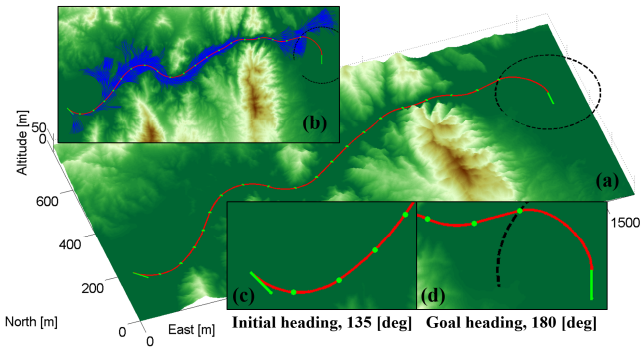

Figure 7. Altitude Map and Resulting Path – Part I, Case I
(a) Red line represents $\mathbf{P}_{path}$ and dotted circle is goal area of radius R_{goal}. Green points are the points where motion primitives are linked. (b) Blue lines are extended edges of the graph. (c) Initial heading angle is equal to 135 [deg]. (d) Goal heading angle is equal to 180 [deg].

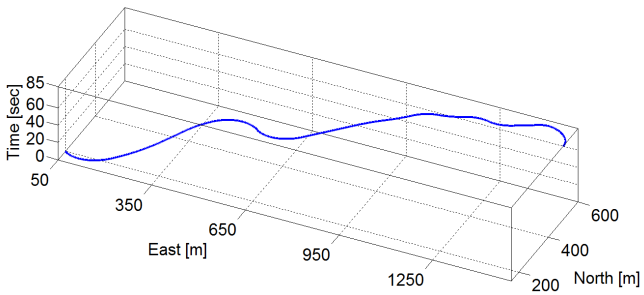

Figure 8. Resulting Path with Respect to Time – Part I, Case I
This shows that the proposed algorithm can satisfy the arrival time at the goal position constraint of 85 s.

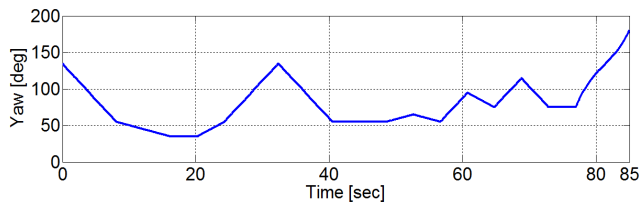

Figure 9. Heading Angle of Resulting Path – Part I, Case I
From this figure, we can confirm that initial and goal heading conditions are satisfied.

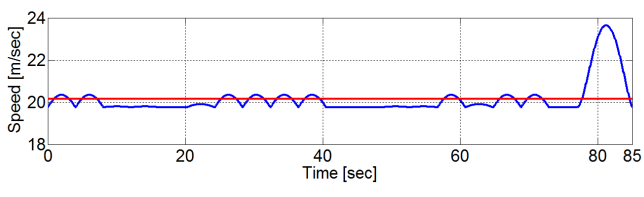

Figure 10. Inertial Speed Command – Part I, Case I
Red line represents inertial speed command, which is the mean value of the blue line calculated by $\|\dot{\mathbf{B}}\|$.

Simulation Part I, Case II

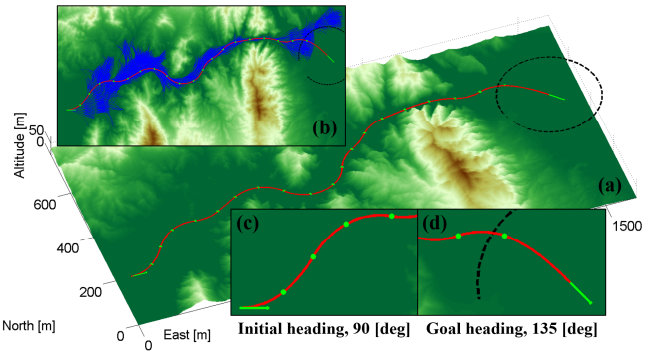

Figure 11. Altitude Map and Resulting Path – Part I, Case II
(a) Red line represents $\mathbf{P}_{path}$ and dotted circle is goal area of radius R_{goal}. Green points are the points where motion primitives are linked. (b) Blue lines are extended edges of the graph. (c) Initial heading angle is equal to 90 [deg]. (d) Goal heading angle is equal to 135 [deg].

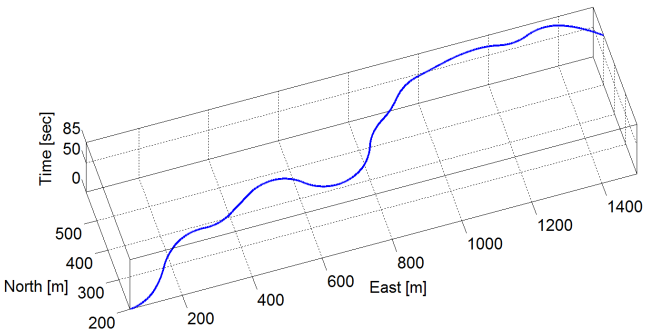

Figure 12. Resulting Path with Respect to Time – Part I, Case II
This shows that the proposed algorithm can satisfy the arrival time at the goal position constraint of 85 s.

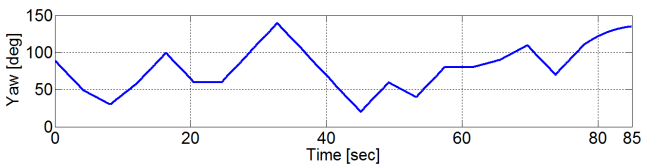

Figure 13. Heading Angle of Resulting Path – Part I, Case II
From this figure, we can confirm that initial and goal heading conditions are satisfied.

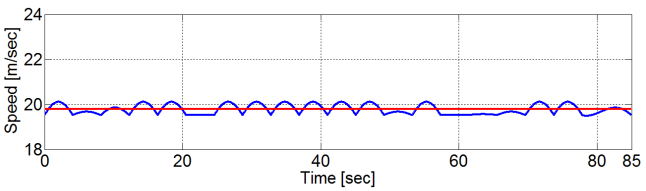

Figure 14. Inertial Speed Command – Part I, Case II
Red line represents inertial speed command, which is the mean value of the blue line calculated by $\|\dot{\mathbf{B}}\|$.

Simulation Part I, Case III

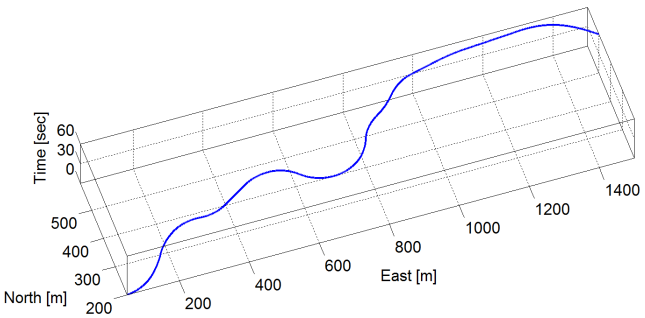

Figure 15. Resulting Path with Respecto to Time – Part I, Case III
The arrival time is 60 s, which differs from that in Case II.

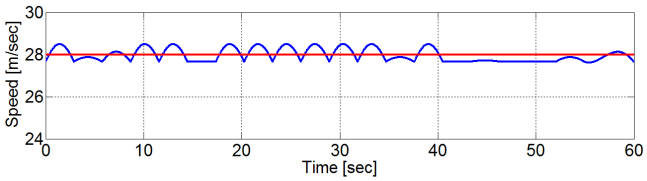

Figure 16. Inertial Speed Command – Part I, Case III
Red line is inertial speed command, which has increased because the arrival time of case III has decreased to 60 s.

Simulation Part II

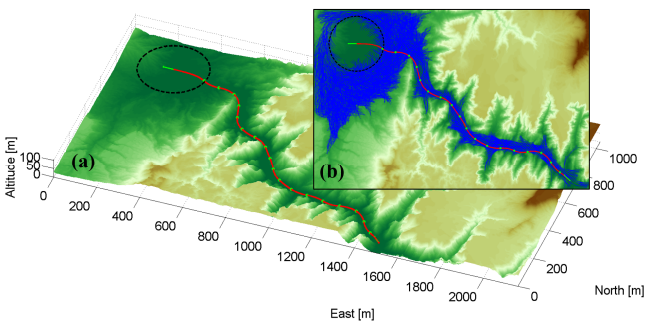

Figure 17. Altitude Map and Resulting Path – Part II
(a) Red line represents $\mathbf{P}_{path}$ and dotted circle is goal area of radius R_{goal}. Green points are the points where motion primitives are linked. (b) Blue lines are extended edges of the graph.

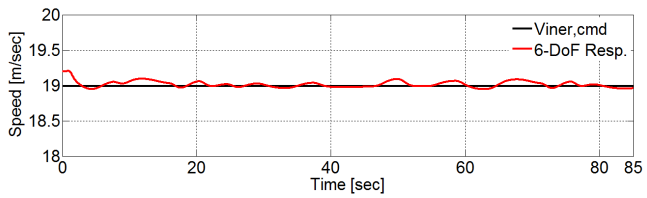

Figure 18. Inertial Speed Command and Its Response – Part II
Black line is inertial speed command and red line is nonlinear 6-DoF simulation response. Inertial speed controller is based on PID controller in this simulation environment.

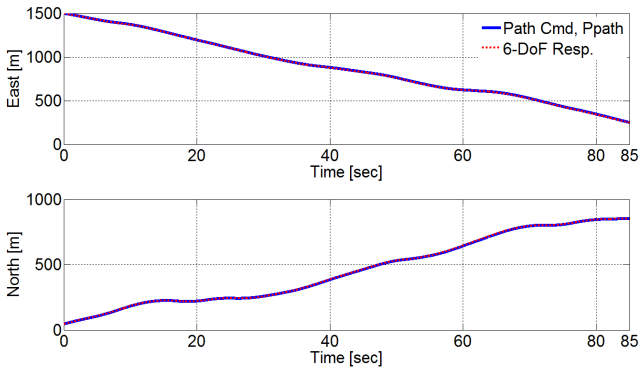

Figure 19. Lateral Path Command, Its Response – Part II

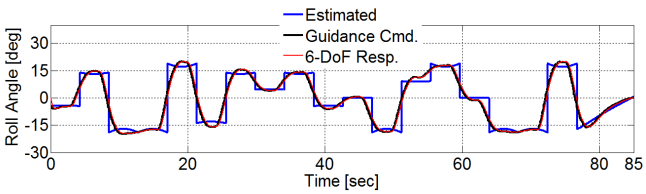

Figure 20. Estimated Roll Angle, Its Response – Part II

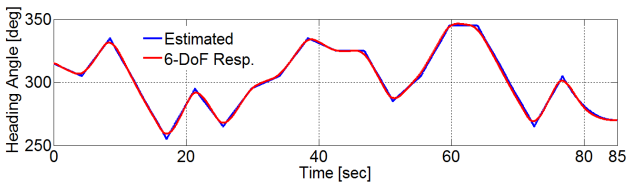

Figure 21. Estimated Heading Angle, Its Response – Part II

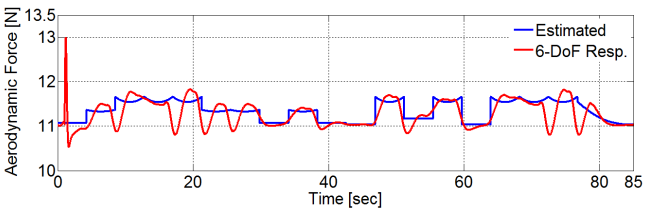

Figure 22. Estimated Aerodynamic Force Required and Its Response – Part II

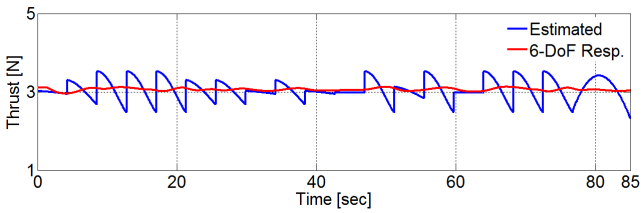

Figure 23. Estimated Thrust Required, Its Response – Part II

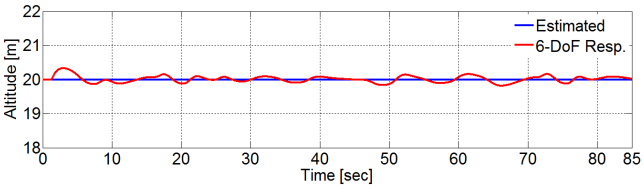

Figure 24. Altitude Command, Its Response – Part II

IV. CONCLUSION

In this paper, we have presented the algorithm which produces smooth and dynamically feasible paths for fixed-wing aircraft. It uses RRT as a base planner and motion primitives to extend the tree. Vertices on the tree contain heading information, and this helps to ensure that the heading angle is continuous. Therefore, smooth paths can be generated and dynamic feasibility is guaranteed, because the motion primitive set is constrained by the maximum lateral acceleration limit.

One of several advantages of the proposed algorithm is that both initial and goal heading conditions can be satisfied. This advantage was demonstrated by simulation I, cases II and III. The second advantage is that the inertial speed command can be calculated to satisfy the arrival time to the goal position, and the results of this algorithm were illustrated in simulation I, case III. The third advantage is that dynamic information including the roll and heading angles, required aerodynamic force, and required thrust can be estimated, and simulation II showed that our method provides accurate values for these variables.

These advantages make the proposed algorithm useful when the initial and goal heading must be considered and the arrival time at the goal position is specified. Furthermore, the estimated dynamic information generated by our method is very useful for determining whether an aircraft has sufficient dynamic capability to follow the generated path.

APPENDIX

The nonlinear 6-DoF simulation model is based on RC-scale fixed-wing aircraft and aerodynamic coefficients are obtained from the AVL program [13].

Aircraft Specification			
m_{total}	1.125 [kg]	C_{L_α}	4.397357
S_{ref}	0.3321 [m^2]	C_{D_0}	0.04
C_{L_0}	0	K	0.0691

Aerodynamic Coefficients			
C_{L_α}	4.397357	C_{l_r}	0.040199
C_{m_α}	-0.492629	C_{n_r}	-0.249136
C_{Y_β}	-0.476408	$C_{Y_{\delta a}}$	0.062774
C_{l_β}	-0.023818	$C_{l_{\delta a}}$	-0.360846
C_{n_β}	0.151514	$C_{n_{\delta a}}$	-0.029085
C_{Y_p}	0.057217	$C_{L_{\delta e}}$	0.724246
C_{l_p}	-0.363469	$C_{m_{\delta e}}$	-1.561560
C_{n_p}	-0.019080	$C_{Y_{\delta r}}$	0.293944
C_{L_q}	7.240289	$C_{l_{\delta r}}$	0.012959
C_{m_q}	-8.840204	$C_{n_{\delta r}}$	-0.167675
C_{Y_r}	0.530628		

ACKNOWLEDGEMENT

This work was supported by Agency for Defense Development(ADD) under the contract UD130041JD

REFERENCES

[1] S.M. LaValle, *Planning Algorithm*. Cambridge University Press, Cambridge, 2006.

[2] S.M. LaValle and J.J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," *Algorithmic and Computational Robotics: New Directions*, pp. 293–308, 2000.

[3] S.M. LaValle and J.J. Kuffner, "Randomized Kinodynamic Planning," *The International Journal of Robotics Research*, vol. 20, no. 5, p. 378, 2001.

[4] S. Karaman and E. Frazzoli, "Sampling-based Algorithms for Optimal Motion Planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[5] K. Yang, S. Moon, S. Yoo, J. Kang, N.L. Doh, H.B. Kim, and S. Joo, "Spline-Based RRT Path Planner for Non-Holonomic Robots," *Journal of Intelligent and Robotic Systems*, vol. 73, issue 1-4, pp. 763-782, Jan. 2014.

[6] E. Koyuncu, N.K. Ure, and G. Inalhan, "Integration of Path/Maneuver Planning in Complex Environments for Agile Maneuvering UCAVs," *Journal of Intelligent and Robotic Systems*, vol. 57, issue 1-4, pp. 143-170, Jan. 2010.

[7] E. Koyuncu and G. Inalhan, "A Probabilistic B-Spline Motion Planning Algorithm for Unmanned Helicopters Flying in Dense 3D Environments," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nice, France, Sept. 22-26, 2008.

[8] M. Kothari and I. Postlethwaite, "A Probabilistically Robust Path Planning Algorithm for UAVs Using Rapidly-Exploring Random Trees," *Journal of Intelligent and Robotic Systems*, vol. 71, issue 2, pp. 231-253, Aug. 2013.

[9] S. Lim and H. Bang, "UAV Guidance Laws to Arrival at Desired Position and Time from Desired Direction," *International Conference on Control, Automation and Systems*, Gyeonggi-do, Korea, Oct. 26-29, 2011.

[10] R.C. Nelson, *Flight Stability and Automatic Control*. McGraw-Hill, 1998, ch. 3.

[11] B.L. Stevens and F.L. Lewis, *Aircraft Control and Simulation*. Wiley, 2003, ch. 2.

[12] S. Park, J. Deyst, and J.P. How, "Performance and Lyapunov Stability of a Nonlinear Path-Following Guidance Method," *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 6, pp. 1718-1728, 2007.

[13] AVL Reference Website, http://web.mit.edu/drela/Public/web/avl/