Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Safe navigation of quadrotors with jerk limited trajectory

Shu-peng LAI[†‡1], Meng-lu LAN[2], Ya-xuan LI[3], Ben M. CHEN[1,4]

[1]*Department of Electrical and Computer Engineering, National University of Singapore, Singapore*
[2]*Graduate School for Integrative Science & Engineering, National University of Singapore, Singapore*
[3]*School of Control Science and Engineering, Shandong University, Jinan 250061, China*
[4]*Department of Mechanical and Automation Engineering, Chinese University of Hong Kong, Hong Kong, China*
[†]E-mail: elelais@nus.edu.sg

Received Nov. 12, 2018; Revision accepted Jan. 4, 2019; Crosschecked Jan. 8, 2019

Abstract: Many aerial applications require unmanned aerial systems operate in safe zones because of the presence of obstacles or security regulations. It is a non-trivial task to generate a smooth trajectory satisfying both dynamic constraints and motion limits of the unmanned vehicles while being inside the safe zones. Then the task becomes even more challenging for real-time applications, for which computational efficiency is crucial. In this study, we present a safe flying corridor navigation method, which combines jerk limited trajectories with an efficient testing method to update the position setpoints in real time. Trajectories are generated online and incrementally with a cycle time smaller than 10 μs, which is exceptionally suitable for vehicles with limited onboard computational capability. Safe zones are represented with multiple interconnected bounding boxes which can be arbitrarily oriented. The jerk limited trajectory generation algorithm has been extended to cover the cases with asymmetrical motion limits. The proposed method has been successfully tested and verified in flight simulations and actual experiments.

Key words: Quadrotor; Unmanned aerial vehicle; Motion planning
https://doi.org/10.1631/FITEE.1800719 **CLC number:** V279

1 Introduction

Recently, thanks to the advancement in sensors, controllers, and onboard electronics, unmanned aerial vehicles (UAVs) have increasingly become popular in applications, such as industrial inspection, surveillance, environment mapping, and agriculture. Among all types of UAVs, the quadrotor platform is the most popular because of its agility and low maintenance cost. Moreover, with the development of multiple vehicle scheduling systems (Peng et al., 2017), it is possible to accomplish complex missions with multiple low-cost quadrotors. For many applications, it is normal to have the presence of geometric obstacles or regulations that constrain the vehicle to fly within a specific safe volume. For example, in a site inspection task, a UAV may not be allowed to fly above any residential area because of regulations imposed by the government authority. A common practice is to formulate the problem into a trajectory generation problem, which would consist of reference signals for lower-level controllers to satisfy dynamic constraints of the vehicle and avoid obstacles. However, with all these constraints, it usually leads to a nonlinear and non-convex optimization problem that is difficult to solve in general. For example, obstacles are modeled as soft constraints (Gao et al., 2017) whereas the trajectory is parameterized as a multi-segment polynomial. The resulting optimization problem is solved by a two-layer non-convex optimization process, which does not always guarantee a feasible solution. Florence et al. (2016) proposed a method to limit the size of the optimization

[‡] Corresponding author
ORCID: Shu-peng LAI, http://orcid.org/0000-0003-2597-5392

problem by restricting the input to a finite set of linearly changed accelerations. The resulting optimization problem can be efficiently solved by enumerating through all acceleration changes. However, this technique limits the complexity of possible maneuvers and is restricted to local trajectory planning.

A safe flying corridor (SFC) based approach is commonly adopted to reformulate the non-convex optimization problem into a convex one. It assumes a pre-existing line-segment based nominal plan (red line-segments in Fig. 1) defined by multiple waypoints (red triangles in Fig. 1) which can either be given by a user or automatically generated through methods such as A* pathfinders. The line-segments are enclosed by an SFC (green cuboid in Fig. 1) and a convex optimization problem can be formulated to constrain the trajectory inside the SFC. Chen and Shen (2017) proposed to represent the SFC using multiple axis-aligned rectangles. Then an iterative quadratic programming (QP) problem was formulated to constrain the trajectories. SFC is extended to arbitrarily shaped convex polygons (Liu et al., 2017), but the safety and motion limitations are satisfied with only sampled time points of the trajectory. B-splines are used to enforce the safety and motion limitations over the entire trajectory (Lai et al., 2018). The SFC problems are generally solved as a QP problem, which could be computationally expensive for low-cost vehicles with limited onboard computational power, especially when a rapid replanning is needed. Moreover, to obtain the convex formulation, these methods adopt the time interval segmentation technique that requires the estimation of the time spent in each polyhedron. An inaccurate estimation usually leads to an inadequate or infeasible solution.

However, the jerk limited trajectory can be used as a reference for quadrotors in an obstacle-free environment. It is first used in robotic arms which require a smooth transition in the torque of motors. Haschke et al. (2008) proposed an efficient decision tree based method to generate the velocity, acceleration, and jerk limited trajectory. Kröger (2011) suggested a similar approach with enhanced numerical stability and non-zero end velocity. The jerk limited trajectory is an effective motion primitive for quadrotors to meet their motion limitations (Hehn and D'Andrea, 2015). Lai et al. (2016) used a receding horizon control (RHC) fashion to locally generate obstacle avoidance trajectories. Lopez and How (2017) used the relaxed version with no end-position specification to parameterize its policy space. Furthermore, Ren and Huo (2010) combined these local trajectories with a global planner. Wang et al. (2017) successfully applied the jerk limited trajectories on a vertical-take-off-and-landing (VTOL) vehicle to guide the flight among transitions. It is worth noting that all the aforementioned techniques are developed to handle only symmetrical limitations on velocity, acceleration, and jerk of the trajectory. This is quite inconvenient for quadrotor systems.

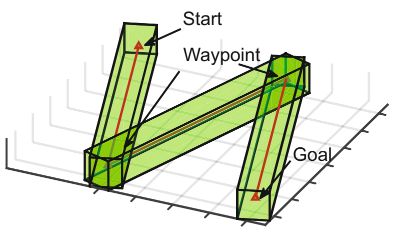


Fig. 1 Line-segments and safe flying corridor
References to color refer to the online version of this figure

In this study, we present a jerk limited trajectory generation method that avoids the complex decision tree and accepts asymmetrical constraints. It performs a bisection search over the switching time directly instead of the achieved velocity as proposed by Ezair et al. (2014). Then the jerk limited trajectory is adopted as motion primitives to guide the quadrotor to fly inside the SFC with guaranteed safety. To the best of our knowledge, it is the first time that the jerk limited trajectory has been used to solve SFC problems.

2 Quadrotor dynamics

A quadrotor is controlled by varying the rotating speeds of its four propellers. Its dynamic model is usually expressed in a body frame B for the rotational movement and a global frame G for its translational movement (Fig. 2). In this study, we focus on the translational model of the vehicle, which can be expressed as

$$\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{bmatrix} = \boldsymbol{R}_B^G \begin{bmatrix}0\\0\\a\end{bmatrix} + \begin{bmatrix}0\\0\\-g\end{bmatrix}, \tag{1}$$

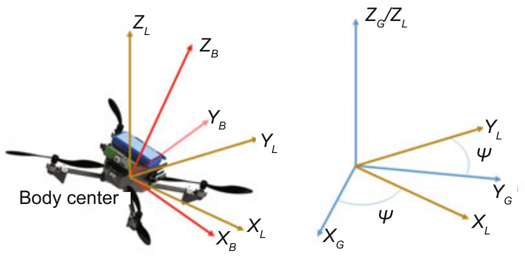


Fig. 2 Coordinate systems

$$\dot{\boldsymbol{R}}_B^G = \boldsymbol{R}_B^G \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}, \tag{2}$$

where x, y, and z denote the vehicle's positions and g is the gravity. The attitude of the vehicle is represented by the rotational matrix $\boldsymbol{R}_B^G$ between its body frame and global frame. The translational acceleration is controlled by vehicle's attitude and normalized collective thrust a. The vehicle's attitude is controlled by its body rates ω_x, ω_y, and ω_z. This simplified model is effective and sufficient for quadrotors in practice. Due to the actuator and sensor limitations, it is commonly assumed that the quadrotor offers limited total thrust and body rates. These limitations can be expressed as constraints on the acceleration and jerk of the vehicle (Hehn and D'Andrea, 2015). Assume the mass of the vehicle is m. Then we can obtain the limitations on the total thrust f as

$$\begin{gathered} \|f\| = \sqrt{\ddot{x}^2 + \ddot{y}^2 + (\ddot{z}+g)^2} \leq \frac{f_{\max}}{m}, \\ \ddot{z} \geq \ddot{z}_{\min} \geq \frac{f_{\min}}{m} - g, \end{gathered} \tag{3}$$

where $f_{\min}$ and $f_{\max}$ are the minimum and maximum available thrusts, respectively. The maximum body rate $\omega_{\max}$ around the body x- and y-axis shall satisfy

$$\sqrt{\dddot{x}^2 + \dddot{y}^2 + \dddot{z}^2} \leq (\ddot{z}_{\min} + g)\omega_{\max}. \tag{4}$$

Let G^a denote the space spanned by acceleration in the global frame, and G^j denote the space spanned by jerk in the global frame. Problem (3) spans a dome shaped constraint volume (CV) in G^a (Fig. 3), while problem (4) spans a sphere in G^j. In practice, cylindrical CVs are frequently used for external restrictions, such as safety regulations (Ang et al., 2018). CVs are intuitive and adjustable for on-site operators and can be made to respect the dynamical differences on the quadrotor's horizontal (h) and vertical (v) axes. For example, an acceleration cylindrical CV can be expressed as

$$\begin{gathered} \sqrt{\ddot{x}^2 + \ddot{y}^2} \leq a_{\mathrm{h}\max}, \\ a_{\mathrm{v}\min} \leq \ddot{z} \leq a_{\mathrm{v}\max}. \end{gathered} \tag{5}$$

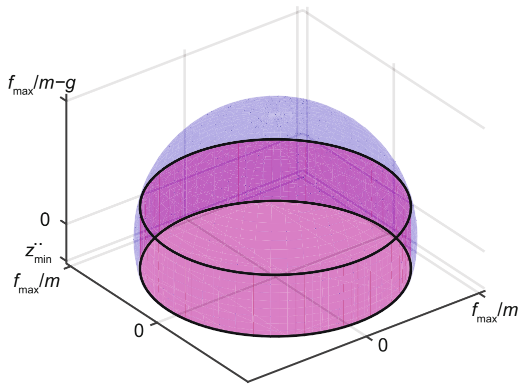


Fig. 3 Constraint volume on acceleration
References to color refer to the online version of this figure

On the horizontal axis, we limit the norm of the acceleration by $a_{\mathrm{h}\max}$. On the vertical axis, an asymmetrical constraint can be set by tuning $a_{\mathrm{v}\min}$ and $a_{\mathrm{v}\max}$ to account for different acceleration limitations for descending and ascending (the red cylinder in Fig. 3). It is fully contained by the dome in Fig. 3 to guarantee the feasibility of the generated trajectory. In theory, any arbitrarily shaped CV can be used as long as it satisfies problems (3) and (4), including the acceleration dome and jerk sphere themselves.

3 Jerk limited trajectory generation

Basic elements in mobile robot application bring the vehicle to the desired location. In this section, we first present a technique to generate jerk, acceleration, and velocity limited trajectory on a single axis. Then we adopt this technique to generate a three-dimensional (3D) trajectory which satisfies the constraints presented in Section 2. The trajectory is used as a reference for a feed-forward and feedback controller. Therefore, properties of the trajectory can be examined before its execution. Because of the limited onboard computational power of the quadrotor and its tight requirement on real-time capability, the trajectory generation algorithm needs to be as efficient as possible. Unlike previous works (Haschke et al., 2008; Kröger and Wahl, 2010), our algorithm does not require a decision tree. It allows asymmetrical constraints to be applied to the velocity, acceleration, and jerk of the trajectory. Similar to the work of Ezair et al. (2014), our algorithm relies on a bisection search procedure. However, we choose to search for desired switching time instead of reached cruise velocity. It allows our algorithm to achieve a more time-efficient double deceleration profile (Haschke et al., 2008) when necessary.

3.1 Single-axis problem

Given a triple integrator system as

$$\dot{p} = v, \dot{v} = a, \dot{a} = j, \tag{6}$$

where p is the position, v the velocity, a the acceleration, and j the jerk which serves as the input. We aim to solve a two-point boundary value problem (TPBVP) with state and input constraints:

$$\begin{cases} p(0) = p_0, \quad p(t_\mathrm{f}) = p_\mathrm{f}, \\ v(0) = v_0, \quad v(t_\mathrm{f}) = 0, \\ a(0) = a_0, \quad a(t_\mathrm{f}) = 0, \\ v_{\min} \le v(t) \le v_{\max}, \ \forall t \in [0, t_\mathrm{f}], \\ a_{\min} \le a(t) \le a_{\max}, \ \forall t \in [0, t_\mathrm{f}], \\ j_{\min} \le j(t) \le j_{\max}, \ \forall t \in [0, t_\mathrm{f}]. \end{cases} \tag{7}$$

To guarantee the existence of a solution, it is assumed that $v_{\min} < 0 < v_{\max}, a_{\min} < 0 < a_{\max}$, and $j_{\min} < 0 < j_{\max}$.

3.1.1 Second-order system

To illustrate our algorithm, we first present the solution to the second-order problem, which is equivalent of bringing system (6) from an arbitrary initial state to a velocity setpoint v_ref in the minimum time:

$$\begin{aligned} \min \quad & t_\mathrm{end} \\ \text{s.t.} \quad & v(0) = v_0, \quad v(t_\mathrm{end}) = v_\mathrm{ref}, \\ & a(0) = a_0, \quad a(t_\mathrm{end}) = 0, \\ & \dot{v}(t) = a(t), \\ & \dot{a}(t) = j(t), \\ & a_{\min} \le a(t) \le a_{\max}, \ \forall t \in [0, t_\mathrm{end}], \\ & j_{\min} \le j(t) \le j_{\max}, \ \forall t \in [0, t_\mathrm{end}]. \end{aligned} \tag{8}$$

Algorithm 1 extends the work of Haschke et al. (2008) to make it possible to include the asymmetrical limitations. In Algorithm 1, we focus on

Algorithm 1 Velocity target solver

```
1: Input: $v_0$, $a_0$, $a_{\max}$, $a_{\min}$, $j_{\max}$, $j_{\min}$, and $v_\mathrm{d}$
2: Output: $P$
3: if $a_0 \ge 0$ then
4:     $v_\mathrm{e} = v_0 + a_0 |a_0/j_{\min}|/2$
5: else
6:     $v_\mathrm{e} = v_0 + a_0 |a_0/j_{\max}|/2$
7: end if
8: $d_\mathrm{a} = \mathrm{sign}(v_\mathrm{d} - v_\mathrm{e})$
9: if $d_\mathrm{a} == 1$ then
10:     $a_\mathrm{c} = a_{\max}$
11: else if $d_\mathrm{a} == -1$ then
12:     $a_\mathrm{c} = a_{\min}$
13: else
14:     $a_\mathrm{c} = 0$
15: end if
16: if $a_\mathrm{c} - a_0 \ge 0$ then
17:     $t_1 = (a_\mathrm{c} - a_0)/j_{\max}$
18:     $j_1 = j_{\max}$
19: else
20:     $t_1 = (a_\mathrm{c} - a_0)/j_{\min}$
21:     $j_1 = j_{\min}$
22: end if
23: $v_1 = v_0 + a_0 t_1 + t_1^2 j_1/2$
24: if $-a_\mathrm{c} \ge 0$ then
25:     $t_3 = -a_\mathrm{c}/j_{\max}$
26:     $j_3 = j_{\max}$
27: else
28:     $t_3 = -a_\mathrm{c}/j_{\min}$
29:     $j_3 = j_{\min}$
30: end if
31: $\bar{v}_3 = a_\mathrm{c} t_3 + t_3^2 j_3/2$
32: $\bar{v}_2 = v_\mathrm{d} - v_1 - \bar{v}_3$
33: if $d_\mathrm{a} == 0$ then
34:     $t_2 = 0$
35: else
36:     $t_2 = \bar{v}_2/a_\mathrm{c}$
37: end if
38: if $t_2 < 0$ then
39:     if $d_\mathrm{a} == 1$ then
40:         $a_n = \sqrt{\left(2(v_\mathrm{d} - v_0) + \frac{a_0^2}{j_{\max}}\right) / \left(\frac{1}{j_{\max}} - \frac{1}{j_{\min}}\right)}$
41:         $t_1 = (a_n - a_0)/j_{\max}$
42:         $t_2 = 0$
43:         $t_3 = -a_n/j_{\min}$
44:     else if $d_\mathrm{a} == -1$ then
45:         $a_\mathrm{n} = -\sqrt{\left(2(v_\mathrm{d} - v_0) + \frac{a_0^2}{j_{\min}}\right) / \left(\frac{1}{j_{\min}} - \frac{1}{j_{\max}}\right)}$
46:         $t_1 = (a_n - a_0)/j_{\min}$
47:         $t_2 = 0$
48:         $t_3 = -a_n/j_{\max}$
49:     end if
50: end if
51: $P \cdot T_1 = t_1$
52: $P \cdot T_2 = t_2 + t_1$
53: $P \cdot T_3 = t_3 + t_2 + t_1$
54: $P \cdot j_1 = j_1$
55: $P \cdot j_2 = 0$
56: $P \cdot j_3 = j_3$
```

determination of the acceleration profile covering the amount of area that equals the desired change in velocity. The acceleration is brought to zero instantly, and the resulting end velocity v_{e} is compared with the desired velocity v_{d}. If v_{e} is smaller than v_{d}, we need a positive cruise direction for acceleration; otherwise, a negative cruise direction is needed (lines 3–8). Haschke et al. (2008) proposed that, to achieve a time-optimal trajectory, the jerk input can be only maximum, minimum, or zero, and there are at most three segments of acceleration profile. Therefore, we determine whether the acceleration profile is triangular- or trapezoidal-shaped, depending on whether it has reached its maximum or minimum. It is tested by bringing the acceleration to either maximum or minimum and immediately to zero, and checking whether the resultant velocity overshoots or undershoots v_{d} (lines 9–37). The resulting trajectory is called "zero acceleration cruise profile (ZACP)." If there is an undershoot, a nonnegative time cruise phase will be necessary, giving a trapezoidal-shaped acceleration profile (line 36); otherwise, if acceleration reduces to zero before reaching maximum or minimum, a triangularly shaped profile (line 38) will be used. For both shapes, we can find closed-form solutions to determine the jerk inputs (j_1, j_2, and j_3) and their corresponding durations (t_1, t_2, and t_3). Finally, we store these parameters in a parameter structure P. The procedure to determine the shape of the acceleration profile is shown in Fig. 4. For ease of reference, we denote the process in Algorithm 1 as

$$P=\mathrm{solveVelocity}(v_0, a_0, a_{\max}, a_{\min}, j_{\max}, j_{\min}, v_{\mathrm{d}}).$$

Given the initial states p_0, v_0, a_0, and P, we can reconstruct the entire trajectory by forward simulating the system (6). We denote the reconstruction process as

$$(p_{\mathrm{s}}, v_{\mathrm{s}}, a_{\mathrm{s}}) = \mathrm{getState}(v_0, a_0, p_0, P, t_{\mathrm{s}}),$$

which gives a point $(p_{\mathrm{s}}, v_{\mathrm{s}}, a_{\mathrm{s}})$ on the trajectory at a specific time point t_{s}.

3.1.2 Third-order system

With the capability to solve the second-order problem, we extend the solution to cover problem (7). The details of the proposed method can be found in Algorithm 2. Similar to the second-order case, we first determine the sign of the cruising velocity by solving a trajectory that immediately brings the system to a stop (line 3). Cruise velocity is determined

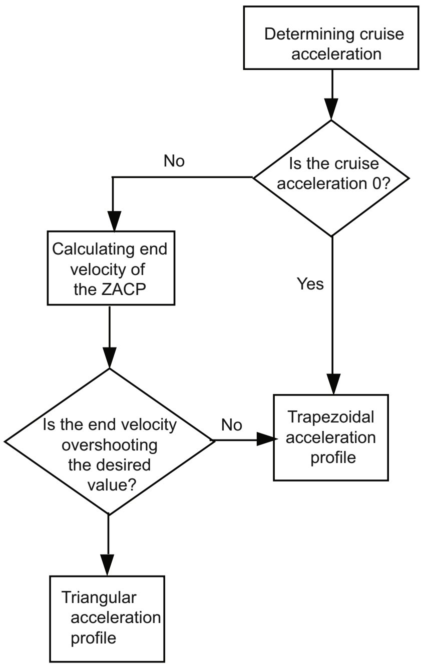

Determining cruise acceleration
Is the cruise acceleration 0?
No
Yes
Calculating end velocity of the ZACP
Is the end velocity overshooting the desired value?
No
Trapezoidal acceleration profile
Triangular acceleration profile

Fig. 4 Shape determination of the acceleration profile

by comparing the stop point p_{sp} to the desired set point p_{d} (lines 3–12). Then we try to create the zero velocity cruise profile (ZVCP) by steering the system to the cruise velocity and immediately to the full stop (lines 13–16). Depending on whether the resulting stop point overshoots or undershoots the setpoint p_{d}, we could determine whether the cruise velocity v_{c} can be reached. If it can be reached, we need to determine the duration by which the velocity stays at v_{c} (line 19); if it cannot be reached, the velocity must start to reduce to zero earlier. That is to say, there exists a switching time t_{pb}, after which the velocity shall decrease to zero. t_{pb} is determined by a bisection search process (lines 22–38). The output of Algorithm 2 consists of three parts: (1) the parameter P_{a} for guiding the system from its initial state towards the cruise velocity v_{c}; (2) the switching time t_{pb} and the cruise time t_{c}; (3) the parameter P_{b} to guide the system to a stop. The procedure to determine the existence of the cruise phase is shown in Fig. 5. To reconstruct the trajectory from these outputs, it is necessary to determine the states at the switching time as

$$(p_{\mathrm{pb}}, v_{\mathrm{pb}}, a_{\mathrm{pb}}) = \mathrm{getState}(v_0, a_0, p_0, P_a, t_{\mathrm{pb}}).$$

If $0 \le t < t_{\mathrm{pb}}$, the trajectory can be reconstructed by $\mathrm{getState}(v_0, a_0, p_0, P_a, t_{\mathrm{pb}})$; otherwise, if $t_{\mathrm{pb}} \le t < t_{\mathrm{c}} + t_{\mathrm{pb}}$, the trajectory is at the cruis-

Algorithm 2 Position target solver

```
1: Input: $p_0, v_0, a_0, v_{\max}, a_{\max}, j_{\max}, v_{\min}, a_{\min}, j_{\min}$,
   and $p_{\mathrm{f}}$
2: Output: $P_{\mathrm{a}}, P_{\mathrm{b}}, t_{\mathrm{pb}}, v_{\mathrm{c}}$, and $t_{\mathrm{c}}$
3: $P = \mathrm{solveVelocity}(v_0, a_0, a_{\max}, a_{\min}, j_{\max}, j_{\min}, 0)$
4: $(p_{\mathrm{sp}}, v_{\mathrm{sp}}, a_{\mathrm{sp}}) = \mathrm{getState}(v_0, a_0, p_0, P, P \cdot T_3)$
5: $d_{\mathrm{p}} = \mathrm{sign}(p_{\mathrm{f}} - p_{\mathrm{sp}})$
6: **if** $d_{\mathrm{p}} == 1$ **then**
7:     $v_{\mathrm{c}} = v_{\max}$
8: **else if** $d_{\mathrm{p}} == -1$ **then**
9:     $v_{\mathrm{c}} = v_{\min}$
10: **else**
11:     $v_{\mathrm{c}} = 0$
12: **end if**
13: $P_{\mathrm{a}} = \mathrm{solveVelocity}(v_0, a_0, a_{\max}, a_{\min}, j_{\max}, j_{\min}, v_{\mathrm{c}})$
14: $(p_{\mathrm{fa}}, v, a) = \mathrm{getState}(v_0, a_0, p_0, P_{\mathrm{a}}, P_{\mathrm{a}} \cdot T_3)$
15: $P_{\mathrm{b}} = \mathrm{solveVelocity}(v_{\mathrm{c}}, 0, a_{\max}, a_{\min}, j_{\max}, j_{\min}, 0)$
16: $(p_{\mathrm{fb}}, v, a) = \mathrm{getState}(v_{\mathrm{c}}, 0, p_{\mathrm{fa}}, P_{\mathrm{b}}, P_{\mathrm{b}} \cdot T_3)$
17: $t_{\mathrm{c}} = 0$
18: **if** $\mathrm{sign}(p_{\mathrm{fb}} - p_{\mathrm{f}}) \cdot d_{\mathrm{p}} \leq 0$ **then**
19:     $t_{\mathrm{c}} = (p_{\mathrm{f}} - p_{\mathrm{fb}})/v_{\mathrm{c}}$
20:     $t_{\mathrm{pb}} = P_{\mathrm{a}} \cdot T_3$
21: **else**
22:     $t_{\mathrm{c}} = 0$
23:     $t_{\mathrm{H}} = P_{\mathrm{a}} \cdot T_3$
24:     $t_{\mathrm{L}} = 0$
25:     **for** counter = 1 : N **do**
26:         $t_{\mathrm{pb}} = (t_{\mathrm{H}} + t_{\mathrm{L}})/2$
27:         $(p_{\mathrm{pb}}, v_{\mathrm{pb}}, a_{\mathrm{pb}}) = \mathrm{getState}(v_0, a_0, p_0, P_{\mathrm{a}}, t_{\mathrm{pb}})$
28:         $P_{\mathrm{b}} = \mathrm{solveVelocity}(v_{\mathrm{pb}}, a_{\mathrm{pb}}, a_{\max},$
                $a_{\min}, j_{\max}, j_{\min}, 0)$
29:         $(p_{\mathrm{fb}}, v, a) = \mathrm{getState}(v_{\mathrm{pb}}, a_{\mathrm{pb}}, p_{\mathrm{pb}}, P_{\mathrm{b}}, P_{\mathrm{b}} \cdot T_3)$
30:         **if** $\mathrm{sign}(p_{\mathrm{fb}} - p_{\mathrm{f}}) \cdot d_{\mathrm{p}} < 0$ **then**
31:             $t_{\mathrm{L}} = t_{\mathrm{pb}}$
32:         **else**
33:             $t_{\mathrm{H}} = t_{\mathrm{pb}}$
34:         **end if**
35:         **if** $|p_{\mathrm{fb}} - p_{\mathrm{f}}| < \varepsilon$ **then**
36:             break
37:         **end if**
38:     **end for**
39: **end if**
```

ing stage with constant velocity v_{c}. If $t_{\mathrm{c}} + t_{\mathrm{pb}} \leq t$, the evolution of the trajectory depends on whether a cruise phase has been presented. If $t_{\mathrm{c}} > 0$, the cruise velocity has been reached, and the trajectory can be evaluated by

$$\mathrm{getState}(v_{\mathrm{c}}, 0, p_{\mathrm{c}}, P_{\mathrm{b}}, t - (t_{\mathrm{c}} + t_{\mathrm{pb}})),$$
$$p_{\mathrm{c}} = p_{\mathrm{pb}} + v_{\mathrm{c}} t_{\mathrm{c}}.$$

On the other hand, if $t_{\mathrm{c}} = 0$, the trajectory is described as

$$\mathrm{getState}(v_{\mathrm{pb}}, a_{\mathrm{pb}}, p_{\mathrm{pb}}, P_{\mathrm{b}}, t - (t_{\mathrm{c}} + t_{\mathrm{pb}})).$$

An example of a jerk limited trajectory with asymmetrical constraints is shown in Fig. 6, where

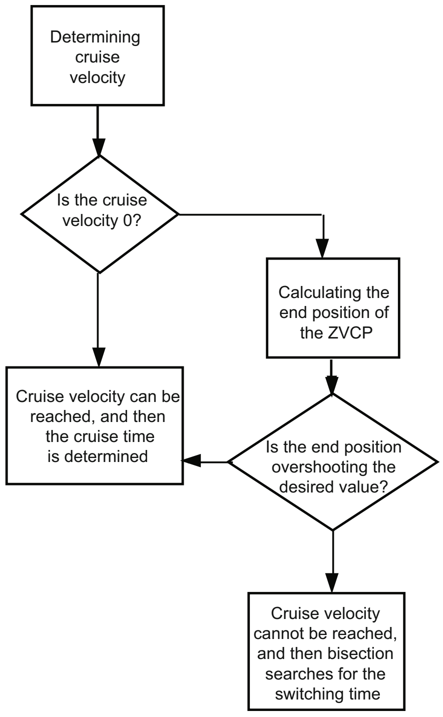

Determining cruise velocity
Is the cruise velocity 0?
Calculating the end position of the ZVCP
Cruise velocity can be reached, and then the cruise time is determined
Is the end position overshooting the desired value?
Cruise velocity cannot be reached, and then bisection searches for the switching time

Fig. 5 Shape determination of the velocity profile

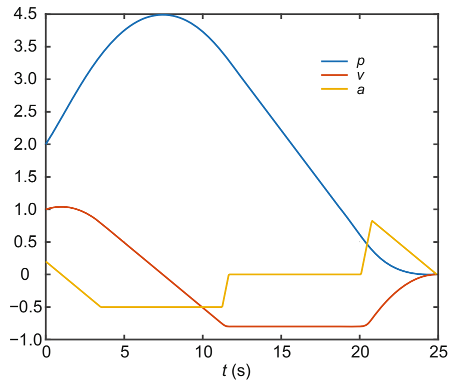


Fig. 6 Jerk limited trajectory with asymmetrical constraints
References to color refer to the online version of this figure

$p_0 = 2$, $v_0 = 1$, $a_0 = 0.2$, $a_{\min} = -0.5$, $a_{\max} = 2$, $v_{\min} = -0.8$, $v_{\max} = 3$, $j_{\min} = -0.5$, $j_{\max} = 3$, and the position set-point is 0.

3.2 Multi-axis problem

For the vehicle moving in a 3D space, the easiest method is to use Algorithm 2 along with each axis of the global frame G. However, we need the trajectory to follow a line-segment based mission, as shown in Fig. 7. Therefore, a synchronization mechanism is needed. Previous methods obtain the time or phase synchronization through post-processing on each axis. In this study, as the trajectory to be generated is as efficient as possible, an axis rotation technique is considered. We create a local frame L_i along with each individual line-segment defined by waypoints w_i and w_{i+1}. Its origin is at w_i, and the x-axis is aligned with the line-segment and the y-axis is perpendicular to the gravity. Then we project the current state onto L_i and generate three jerk limited trajectories on each axis, separately. On the x-axis, the position set-point is chosen as $\|w_{i+1} - w_i\|$, whereas on the y- and z-axis, the set-point is zero. Therefore, the lateral error along with the line-segment $\|w_{i+1} - w_i\|$ will be reduced to zero while the trajectory marches to w_{i+1}.

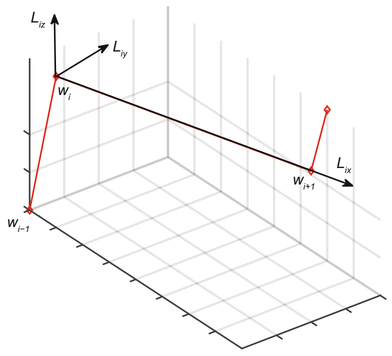


Fig. 7 Line-segment based on a nominal plan

Since the trajectory on each individual axis is individually solved using Algorithm 2, we need to determine the corresponding limitations on velocity, acceleration, and jerk to satisfy the CVs. Taking the acceleration as an example, the constraints it should satisfy when generating the trajectory in the L_i frame can be expressed as

$$\begin{cases} a^{L_i}_{\min_x} \le a^{L_i}_x \le a^{L_i}_{\max_x}, \\ a^{L_i}_{\min_y} \le a^{L_i}_y \le a^{L_i}_{\max_y}, \\ a^{L_i}_{\min_z} \le a^{L_i}_z \le a^{L_i}_{\max_z}, \end{cases} \tag{9}$$

where L_i denotes the frame of the acceleration, and x, y, and z indicate the axes in L_i. Let L^a_i denote the space spanned by $a^{L_i}_x, a^{L_i}_y$, and $a^{L_i}_z$. The constraints in problem (9) span an axis-aligned cuboid in L^a_i. Without any loss of generality, we consider a cylindrical CV defined in G^a as an example. Since frames L_i and G are not aligned, we need to choose the limitations in problem (9), and thus the cylindrical CV fully contains the spanned cuboid (Fig. 8). Therefore, for any trajectory that satisfies problem (9), it fulfills the corresponding CV. Moreover, the same principle can be adopted for the velocity and jerk trajectories.

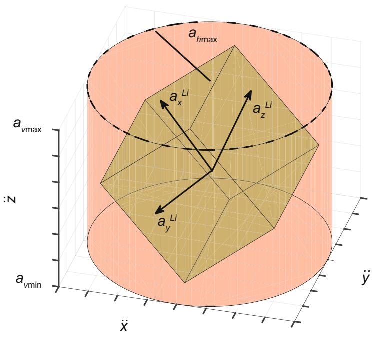


Fig. 8 Cylindrical constraint volume and the selection of axis-decoupled limitations

4 Safe corridor navigation

Assuming that there are N waypoints in total, the aim of the SFC is to enclose $N-1$ line-segments with safety space. In this study, it consists of series of inter-connected bounding boxes (Fig. 9). Unlike the work of Chen and Shen (2017), the bounding boxes used in this study are not required to be axis-aligned in frame G. We denote the oriented bounding box (OBB) that encloses the line-segment $\overline{w_i w_{i+1}}$ as O_i. It is aligned to frame L_i with six positive scalars, $w^i_{x+}, w^i_{x-}, w^i_{y+}, w^i_{y-}, w^i_{z+}$, and w^i_{z-}, defining the width in each direction (Fig. 10). It requires $w^i_{x+} > \|w_{i+1} - w_i\|$; thus, the OBB fully encloses the line-segment. A trivial solution that guarantees the safety of the vehicle is to move along with each line-segment and stop at every waypoint. The resulting trajectory will exactly match the line-segments, thus staying inside the SFC. However, due to the frequent stops, it will be less efficient especially in the case where there are multiple short line-segments. In this study, we present a method to reduce the unnecessary stops to achieve a smoother and faster flight.

4.1 Trajectory switching

Let us denote the trajectory generated in frame L_i towards w_{i+1} as T_i. Then $\dot{T}_i, \ddot{T}_i$, and $\dddot{T}_i$ denote the trajectory's velocity, acceleration, and jerk, respectively. The aim of the proposed method is to perform a trajectory switching from T_i to T_{i+1} before the vehicle reaches the end of T_i and performs a full stop. Assuming that the vehicle is currently tracking T_i, it needs to constantly generate a trajectory T_{i+1} in frame L_{i+1} towards the waypoint w_{i+2} from its current state. If the newly generated T_{i+1} stays inside the SFC and $\dot{T}_{i+1}, \ddot{T}_{i+1}$, and $\dddot{T}_{i+1}$ are contained by their corresponding CVs, T_{i+1} becomes a feasible trajectory and the vehicle immediately switches to it. If T_{i+1} does not pass the checking, the vehicle continues to track the old trajectory T_i. In the worst case, if all T_{i+1}'s fail checking, the vehicle would perform a full stop at the end of T_i. In this case, the proposed algorithm would perform the same as the trivial solution. The details of this process are illustrated in Algorithm 3. For each iteration, we transform the current reference state in frame L_{i+1}. Specifically, the function getCurrentReference returns the current reference state s in the global frame; i.e., the reference position, velocity, and acceleration are tracked by the vehicle. Then we transform the returned state s into the desired frame using the function transform(s, l), which handles both rotation and translation transformation. It is worth noting that the velocity, acceleration, and jerk are all translationally invariant and only rotation is performed. We generate the new trajectory T_{i+1} from the transformed state $\bar{s}$ to the next waypoint in frame L_{i+1}. The function genTrajectory$(\bar{s}, l, t)$ is responsible for generating the new trajectory T_{i+1} and takes in three input parameters: transformed current state $\bar{s}$, frame to generate the trajectory l, and target t expressed in frame L_i. The generation process is built upon Algorithm 2, which generates a three-axis-decoupled trajectory. That is to say, for each x-, y-, and z-axis of frame L_i, Algorithm 2 is executed once. Finally, we use the function check(T) to check whether the generated trajectory T satisfies the safety and feasibility. These will be described in detail later in Algorithms 4 and 5. If the checking criteria have been successfully met, i.e., the trajectory stays inside the safe corridor, we adopt the trajectory and move to the next target. We assume that the first trajectory T_1 is always feasible and safe; however, such an assumption may not be held in real situations. To address this issue, we adopt a method proposed by Lai et al. (2016) to quickly generate a temporal safe trajectory to avoid any possible obstacle nearby. If the initial trajectory T_1 fails in the checking process, we trigger the high-level path planner to re-calculate a nominal plan from the current vehicle state.

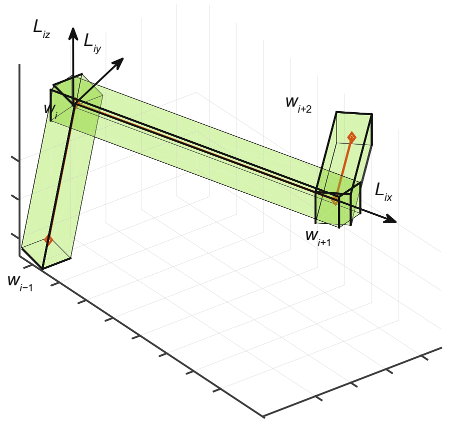


Fig. 9 Safe flying corridor consisting of bounding boxes

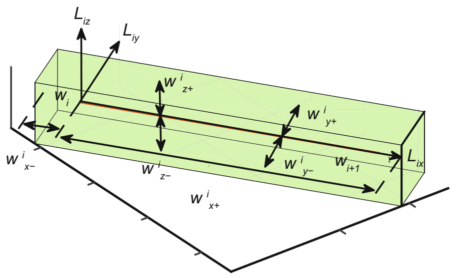


Fig. 10 Dimensions of the bounding box

Algorithm 3 Smooth navigation

```
 1: $i = 0$
 2: while Not reaching $w_N$ do
 3:   if $i < N - 1$ then
 4:     $s$ = getCurrentReference()
 5:     $\bar{s}$ = transform($s$, $L_{i+1}$)
 6:     $T_{i+1}$ = genTrajectory($\bar{s}$, $L_{i+1}$,
            $[\|w_{i+2} - w_{i+1}\|, 0, 0]$)
 7:     if check($T_{i+1}$) then
 8:       $i = i + 1$
 9:       switch to $T_{i+1}$
10:     end if
11:   end if
12: end while
```

4.2 Trajectory checking

Given a trajectory T_{i+1}, the goal is to check whether T_{i+1} is fully inside the SFC, and $\dot{T}_{i+1}, \ddot{T}_{i+1}$, and $\dddot{T}_{i+1}$ are contained by the corresponding CVs.

SFC consists of multiple inter-connected OBBs. One possible approach is to densely sample various points on trajectory and check whether they are all inside the SFC. However, there is no guarantee among these discrete samples. To address this issue, we propose an efficient method to check the trajectory at a continuous time interval by taking a sufficient but not necessary assumption that the trajectory T_{i+1} varies from O_i to O_{i+1}.

Given a single specific OBB O_i, we can check whether a given global-frame trajectory T is inside O_i:

1. Transform T into the frame where O_i is defined, i.e., L_i, and denote the transformed trajectory as T^{L_i}.

2. Find the maximum and minimum along with each individual x-, y-, and z-axis of L_i as $T^{L_i}_{\text{max}_x}$, $T^{L_i}_{\text{min}_x}$, $T^{L_i}_{\text{max}_y}$, $T^{L_i}_{\text{min}_y}$, $T^{L_i}_{\text{max}_z}$, and $T^{L_i}_{\text{min}_z}$.

3. Construct two points in L_i where their coordinates are $\left[T^{L_i}_{\text{max}_x}, T^{L_i}_{\text{max}_y}, T^{L_i}_{\text{max}_z}\right]$ and $\left[T^{i}_{\text{min}_x}, T^{L_i}_{\text{min}_y}, T^{L_i}_{\text{min}_z}\right]$.

4. Check whether these two points are fully contained by O_i.

Since the bounding box O_i is axis-aligned in L_i, the above processes check if the trajectory is fully contained by O_i by finding its extreme values in L_i. It is clear that the trajectory of the double integrator problems consists of at most three segments of the third-order polynomials (Haschke et al., 2008). From Algorithm 2, the trajectory of the triple integrator problem consists of at most two double integrator problems and a cruising phase. Therefore, it has at most seven segments of third-order polynomials. For the three-axis problem, the trajectory possibly has a maximum of 21 segments. Therefore, we could find the maximum and minimum of a trajectory by solving a finite number of second-order polynomial equations, which can be efficiently done because of the existence of the closed-form solution.

For convenience, we denote the above processes with boxCheck(T, O), where T is the trajectory and O is the bounding box. It returns true only if T is fully inside O. To check whether the trajectory is contained by two adjacent OBBs O_i and O_{i+1}, a divide-and-conquer approach is adopted in Algorithm 4. We try to find a split point that resides in both O_i and O_{i+1} to split the trajectory into two parts T_a and T_b (line 3). If no such point exists, the trajectory must not be fully contained by O_i and O_{i+1} (lines 4–7). Then the split trajectories T_a and T_b are checked against their corresponding OBBs (line 8). If both split parts are fully contained by OBBs, the original trajectory T must stay inside the SFC. We adopt a bisection search process to find the split point P_mid in Algorithm 5. For a trajectory T, let $T(\tau)$ denote the corresponding point at time τ and $T(\tau_a : \tau_b)$ the partial trajectory between time τ_a and τ_b. getTotalTime(T) gives the total time required for trajectory T. We bisect the time on trajectory to find the split point P_mid. The bisection search region is initialized to cover the entire trajectory (lines 4–6). Then we start the bisection search process by splitting the trajectory at the medium time τ_mid (lines 8–12). If the medium time point p is already contained by both OBBs, the search stops (lines 13–15). However, if p is contained by only the first OBB (O_i), the search is continued on T_b (lines 16–17). If p is contained in only O_{i+1}, the search continues on the first half of trajectory T_a; otherwise, if it is contained by neither O_i nor O_{i+1}, there exists a point which is outside the SFC and the algorithm returns an empty P_mid.

Algorithm 4 Safe flying corridor check

```
1: Input: $T$, $O_i$, and $O_{i+1}$
2: Output: isInCorridor
3: $[P_\text{mid}, T_\text{a}, T_\text{b}]$ = splitTrajectory($T$, $O_i$, $O_{i+1}$)
4: if $P_\text{mid} = \varnothing$ then
5:     isInCorridor = false
6:     return
7: end if
8: if boxcheck($T_\text{a}$, $O_i$) && boxcheck($T_\text{b}$, $O_{i+1}$) then
9:     isInCorridor = true
10: else
11:     isInCorridor = false
12: end if
```

Algorithm 5 SplitTrajectory

```
1: Input: $T$, $O_i$, and $O_{i+1}$
2: Output: $P_\text{mid}$, $T_\text{a}$, and $T_\text{b}$
3: $P_\text{mid} = \varnothing$
4: $\tau_\text{a} = 0$
5: $\tau_\text{end}$ = getTotalTime($T$)
6: $\tau_\text{b} = \tau_\text{end}$
7: for $i < N_\text{e}$ do
8:     $i = i + 1$
9:     $\tau_\text{mid} = (\tau_\text{a} + \tau_\text{b})/2$
10:     $p = T(\tau_\text{mid})$
11:     $T_\text{a} = T(0 : \tau_\text{mid})$
12:     $T_\text{b} = T(\tau_\text{mid} : \tau_\text{end})$
13:     if $p \in O_i$ && $p \in O_{i+1}$ then
14:         $P_\text{mid} = p$
15:         return
16:     else if $p \in O_i$ && $p \notin O_{i+1}$ then
17:         $\tau_\text{a} = \tau_\text{mid}$
18:     else if $p \notin O_i$ && $p \in O_{i+1}$ then
19:         $\tau_\text{b} = \tau_\text{mid}$
20:     else
21:         return
22:     end if
23: end for
```

With trajectory T_{i+1} fully inside the SFC, we need to check whether its derivatives satisfy CVs. In Section 2, the maximum and minimum of the velocity, acceleration, and jerk have already been selected to be fully contained by the corresponding CVs. However, T_i is generated in L_i and satisfies the limitations defined in L_i, but T_{i+1} and its limitations are generated in L_{i+1}. Therefore, it is possible that the initial condition of T_{i+1} does not satisfy the decoupled velocity, acceleration, and jerk limitations defined in L_{i+1}. In this case, we need to check the feasibility of the derivative trajectories $\dot{T}_{i+1}, \ddot{T}_{i+1}$, and $\dddot{T}_{i+1}$. Similar to the SFC checking case, we propose a method to check whether the trajectory is inside a given cylindrical CV on a continuous time interval. Taking acceleration trajectory $\ddot{T}$ as an example, its feasibility regarding a cylindrical CV can be checked by the following procedure:

1. Construct the horizontal acceleration trajectory $\ddot{T}_{\mathrm{H}}$ from x- and y-axis components of $\ddot{T}$; i.e., $\ddot{T}_x$ and $\ddot{T}_y$ can be expressed as

$$\ddot{T}_{\mathrm{H}} = \sqrt{\ddot{T}_x^2 + \ddot{T}_y^2}.$$

2. Find the maximum of $\ddot{T}_{\mathrm{H}}$ as $h_{\max}$ and check whether it satisfies problem (5):

$$h_{\max} \leq a_{h\max}. \tag{10}$$

3. Find the maximum and minimum of $\ddot{T}_z$ as $\eta_{\max}$ and $\eta_{\min}$, and check whether they satisfy problem (5):

$$a_{v\min} \leq \eta_{\min} \leq a_{v\max}, a_{v\min} \leq \eta_{\max} \leq a_{v\max}. \tag{11}$$

If $\ddot{T}$ satisfies both problems (10) and (11), we argue that the acceleration trajectory satisfies its corresponding cylindrical CV. Similarly, the process mentioned above can be applied to velocity and jerk trajectory. Only when trajectory T_{i+1} passes through SFC checking (Algorithm 4) and its derivatives satisfy problems (10) and (11), does the check() function (line 7 in Algorithm 3) return true and would the vehicle switch to the new trajectory T_{i+1}. Therefore, the trajectory is guaranteed to be both safe and feasible for the underlying controllers.

4.3 Computational efficiency

The major advantage of the proposed approach lies in its efficiency, taking only several microseconds on a normal laptop. In the trajectory generation phase, we adopt a bisection search coupled with a process with a closed-form solution. For the trajectory checking phase, a continuous checking process is adopted to guarantee the soundness of the algorithm. The proposed algorithm takes only a small amount of memory space. Thus, it is expected to be implementation-friendly for a flight controller with a low-end computational capability, such as the Pixhawk flight controllers. We test the efficiency of the single-axis jerk limited trajectory generation algorithm (Algorithm 2). Then we select the initial state and limitations as random variables among the following ranges: $p_0 \in [-100, 100]$, $v_0 \in [-20, 20]$, $a_0 \in [-10, 10]$, $a_{\min} \in [-10, -0.1]$, $a_{\max} \in [0.1, 10]$, $v_{\min} \in [-20, -0.1]$, $v_{\max} \in [0.1, 20]$, $j_{\min} \in [-20, -0.1]$, and $j_{\max} \in [0.1, 20]$. Without loss of generality, the position set-point is at the origin. In the experiment, a total of one billion trajectories are generated without failures. The average time to solve one single-axis trajectory is less than 1 μs, and the size of the execution file is less than 30 KB (Table 1). We compare our algorithm with the Reflexxes library (Kröger, 2011) in solving the single-axis position set-point problem, where our algorithm outperforms the Reflexxes library in terms of both computation efficiency and memory consumption. The comparisons are done on a laptop computer equipped with Intel i5 processor under Windows.

Table 1 Time consumption of the single-axis position set-point problem

Method	Average time (μs)	Executable size (KB)
Our method	0.72	28
Reflexxes	1.96	282

We test the average cycling time of smooth navigation among SFCs. For each computational cycle, the total time of executing the smooth navigation algorithm is 3.91 μs, the time of the trajectory generation phase is 2.68 μs, and the time of the checking phase is 0.90 μs. We randomly generate a nominal plan with nine line-segments. The velocity, acceleration, and jerk limitations are all randomly selected. The experiment is repeated multiple times and the average is taken. The average cycle time is less than 5 μs with both the three-axis trajectory generation and trajectory checking. The experiment has been done on a laptop computer equipped with Intel i7 processor under Linux. Considering that the outerloop controllers of the quadrotors are usually running at a frequency of 20–50 Hz, the computational performance is suitable for any real-time application.

5 Flight experiment

We first demonstrate a simulation showing a flight path that goes through a nine-segment SFC (Fig. 11) and the corresponding trajectory references (Figs. 12 and 13). The horizontal and vertical acceleration limitations are $a_{\mathrm{hmax}} = 2.2$, $a_{\mathrm{vmax}} = 0.8$, and $a_{\mathrm{vmin}} = -0.8$. Fig. 11 shows that the trajectory is fully enclosed by the SFC, and that there are no sharp turns or unnecessary stops at each waypoint. The corresponding position reference can be seen in Fig. 12, which confirms the smooth transition among segments over the entire trajectory. Finally, the acceleration profile is shown in Fig. 13 to satisfy all the constraints mentioned above.

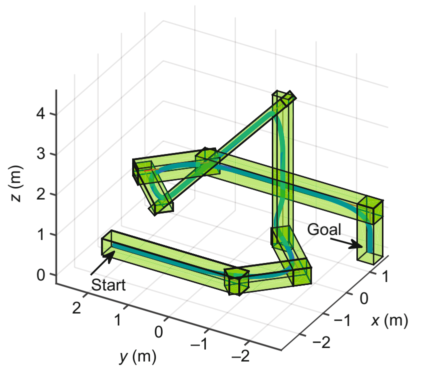


Fig. 11 Flight path passing through a nine-segment safe flying corridor

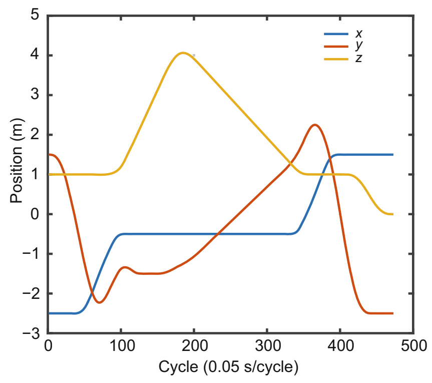


Fig. 12 Position reference generated by the proposed algorithm

References to color refer to the online version of this figure

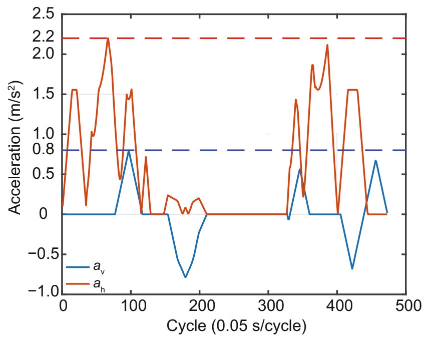


Fig. 13 Acceleration generated by the proposed algorithm

References to color refer to the online version of this figure

The proposed algorithm has been tested by actual flight experiments using a low-cost platform (Fig. 14). The vehicle used is equipped with a Pixhawk flight controller and a Vicon motion capture system providing its position information. The drone has a tip-to-tip size of 26 cm and weighs 726 g. A desktop computer wirelessly provides a reference and measurement signal to the onboard flight controller, which is running at 20 Hz. We compare the execution time of the trajectory obtained by the proposed technique with the trivial approach. When comparing the two methods, we adopt the same limitations on velocity, acceleration, and jerk in both global frame G and each local frame L_i. Specifically, the cylindrical CVs in Table 2 are used. The SFCs, references, and flight results are shown in Fig. 15. In experiment A (Fig. 15a), the SFC is automatically generated to guide the vehicle to fly through multiple pillars. The nominal line-segment path is generated using a traditional A* algorithm. Experiment B (Fig. 15b) is conducted to test the capability of the algorithm to guide the vehicle flying in more complex environments with a rapid height variance. Finally, in experiment C (Fig. 15c) a simulated power-line inspection mission is conducted with a broader mission area and a faster flying speed. Compared with the trivial solution which comes to a full stop at each endpoint, our method saves more than 40% flying time in all three experiments (Table 3).

We demonstrate that through the trajectory switching method, it is possible to dynamically change the desired target. A user could choose an arbitrary destination within an SFC at any moment, and the vehicle will safely proceed towards it. In Fig. 16, the vehicle is first given a target A.

Table 2 State constraints

Experiment	$v_{\mathrm{h\,max}}$ (m/s^2)	$v_{\mathrm{v\,max}}$ (m/s)	$v_{\mathrm{v\,min}}$ (m/s)	$a_{\mathrm{h\,max}}$ (m/s)	$a_{\mathrm{v\,max}}$ (m/s^2)
A	4	0.8	–0.8	2.2	0.8
B	3	0.8	–0.8	1.2	0.8
C	15	0.8	–0.8	2.2	0.8

Experiment	$a_{\mathrm{v\,min}}$ (m/s^2)	$j_{\mathrm{h\,max}}$ (m/s^3)	$j_{\mathrm{v\,max}}$ (m/s^3)	$j_{\mathrm{v\,min}}$ (m/s^3)
A	–0.8	3	3	–3
B	–0.8	3	3	–3
C	–0.8	3	3	–3

Table 3 Flying time consumption

Experiment	Flying time (s)		Improvement
	Our method	Trivial method	
A	11.05	23.70	53%
B	13.05	23.45	44%
C	123.80	211.50	41%

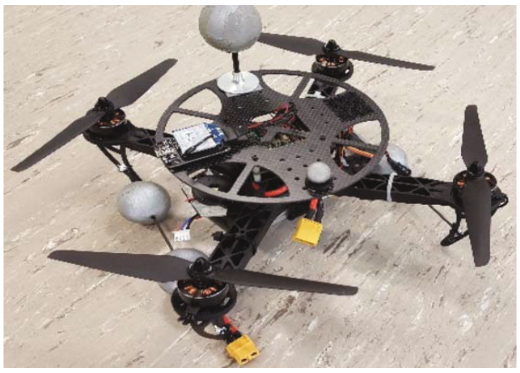
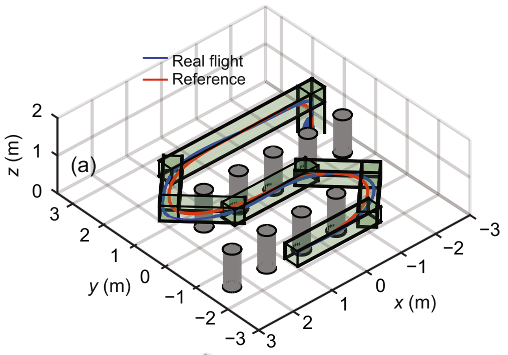
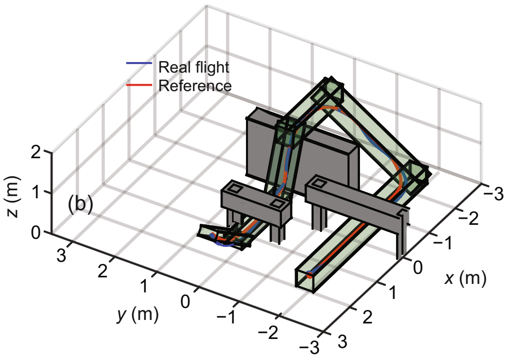


Fig. 14 Platform used for real flight experiments





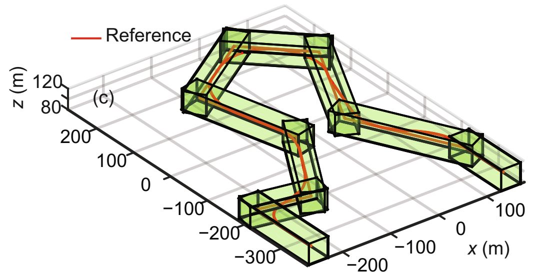


Fig. 15 Real flight experiments: (a) experiment A; (b) experiment B; (c) experiment C

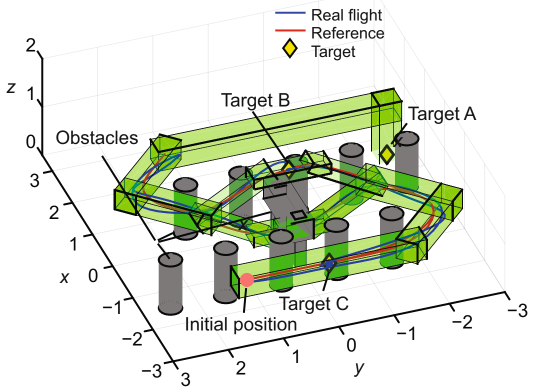


Fig. 16 Dynamically changing targets

However, before reaching its goal, the aim is modified to point B. Once the vehicle reaches B, the goal is ultimately changed to C. For each target switching, the trajectory is immediately re-planned towards the new goal. Since our method can generate the trajectory in microseconds, the instant reaction is to be expected for a flight controller with a low-end computational capability.

6 Conclusions

In this paper, we have developed a computationally efficient algorithm for the quadrotor platform to generate a jerk limited trajectory within a safe corridor. The corridor is conveniently described as a series of inter-connected cuboids. Unlike the trivial solution that requires the vehicle stop at each intersected point of the connected boxes, our algorithm generates a smooth non-stop trajectory with a safety guarantee. The main idea is to incrementally generate a full trajectory by continuously generating a candidate trajectory from the current state of the UAV towards an intermediate waypoint between two cuboids.

To consider the computational efficiency, we have temporarily ignored the complex geometric constraints when generating candidate trajectories. For each candidate trajectory, our algorithm can limit its velocity, acceleration, and jerk in a cylindrical constraint volume, which satisfies the physical limitations of the vehicle. Then a checking algorithm has been proposed to determine the feasibility and safety of the candidate trajectory. Our checking procedure does not require trajectory discretization but guarantees the satisfaction of various constraints on a continuous time interval. Finally, we have noted that the proposed technique has been successfully implemented and tested on actual flight experiments.

References

Ang KZY, Dong XX, Liu WQ, et al., 2018. High-precision multi-UAV teaming for the first outdoor night show in Singapore. *Unmanned Syst*, 6(1):39-65. https://doi.org/10.1142/S2301385018500036

Chen J, Shen SJ, 2017. Improving octree-based occupancy maps using environment sparsity with application to aerial robot navigation. Proc IEEE Int Conf on Robotics and Automation, p.3656-3663. https://doi.org/10.1109/ICRA.2017.7989419

Ezair B, Tassa T, Shiller Z, 2014. Planning high order trajectories with general initial and final conditions and asymmetric bounds. *Int J Rob Res*, 33(6):898-916. https://doi.org/10.1177/0278364913517148

Florence P, Carter J, Tedrake R, 2016. Integrated perception and control at high speed: evaluating collision avoidance maneuvers without maps. Proc 12th Int Workshop on the Algorithmic Foundations of Robotics.

Gao F, Lin Y, Shen SJ, 2017. Gradient-based online safe trajectory generation for quadrotor flight in complex environments. Proc IEEE/RSJ Int Conf on Intelligent Robots and Systems, p.3681-3688. https://doi.org/10.1109/IROS.2017.8206214

Haschke R, Weitnauer E, Ritter H, 2008. On-line planning of time-optimal, jerk-limited trajectories. Proc IEEE/RSJ Int Conf on Intelligent Robots and Systems, p.3248-3253. https://doi.org/10.1109/IROS.2008.4650924

Hehn M, D'Andrea R, 2015. Real-time trajectory generation for quadrocopters. *IEEE Trans Rob*, 31(4):877-892. https://doi.org/10.1109/TRO.2015.2432611

Kröger T, 2011. Opening the door to new sensor-based robot applications—the reflexxes motion libraries. Proc IEEE Int Conf on Robotics and Automation, p.1-4. https://doi.org/10.1109/ICRA.2011.5980578

Kröger T, Wahl FM, 2010. Online trajectory generation: basic concepts for instantaneous reactions to unforeseen events. *IEEE Trans Rob*, 26(1):94-111. https://doi.org/10.1109/TRO.2009.2035744

Lai SP, Wang KL, Qin HL, et al., 2016. A robust online path planning approach in cluttered environments for micro rotorcraft drones. *Contr Theory Technol*, 14(1):83-96. https://doi.org/10.1007/s11768-016-6007-8

Lai SP, Lan ML, Chen BM, 2018. Optimal constrained trajectory generation for quadrotors through smoothing splines. Proc IEEE/RSJ Int Conf on Intelligent Robots and Systems, p.4743-4750. https://doi.org/10.1109/IROS.2018.8594357

Liu SK, Watterson M, Mohta K, et al., 2017. Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-D complex environments. *IEEE Rob Autom Lett*, 2(3):1688-1695. https://doi.org/10.1109/LRA.2017.2663526

Lopez BT, How JP, 2017. Aggressive 3-D collision avoidance for high-speed navigation. Proc IEEE Int Conf on Robotics and Automation, p.5759-5765. https://doi.org/10.1109/ICRA.2017.7989677

Peng KM, Lin F, Chen BM, 2017. Online schedule for autonomy of multiple unmanned aerial vehicles. *Sci China Inform Sci*, 60(7):072203. https://doi.org/10.1007/s11432-016-9025-9

Ren M, Huo XH, 2010. Asynchronous double-precision windows based unmanned aerial vehicle real-time path planning. *Sci China Inform Sci*, 53(2):215-222. https://doi.org/10.1007/s11432-010-0043-7

Wang KL, Ke YJ, Chen BM, 2017. Autonomous reconfigurable hybrid tail-sitter UAV U-Lion. *Sci China Inform Sci*, 60(3):033201. https://doi.org/10.1007/s11432-016-9002-x