Efficient safe corridor navigation with jerk limited trajectory for quadrotors

Shupeng Lai[1], Menglu Lan[2], Ben M. Chen[1]
1. Department of Electrical Computer Engineering, National University of Singapore (NUS).
E-mail: elelais@nus.edu.sg
2. Graduate School for Integrative Science and Engineering, NUS.

Abstract: Quadrotors commonly operate in low attitude and obstacle-strewn environment. To guarantee its safety, it has to be operated inside the certain safe area. This paper presents an efficient technique to generated jerk limited trajectory that leads a quadrotor to travel inside such safe area. By limiting the trajectory's velocity, acceleration, and jerk, it is made feasible regarding the vehicle's physical limits. The underlying trajectory generation algorithm is extremely efficient and takes only several microseconds to produce the results. Finally, the algorithm is tested with real flight experiments.

Key Words: Quadrotor, Trajectory generation

1 Introduction

Safe trajectory generation for quadrotors has been intensively researched in recent years. Because the quadrotors are fragile vehicles that are easily damaged in the collision but often forced to operate in the obstacle-strewn environment due to the nature of its missions. Reaction based methods, such as the potential field [1] and velocity obstacles [2] has been successfully implemented on quadrotors. These methods directly control the inputs of the vehicle but cannot handle input and state constraints. It results in a mismatch between the actual and the predicted behavior of the vehicle and could cause troubles when the flying speed is high, or the obstacles are dense. To address the issue, [3] proposed a trajectory generation and tracking approach. It is further extended in [4] for online obstacle avoidance. In these approaches, a nominal plan is first generated by a global geometrical planner. Then a safe corridor consists of cubes, or other polyhedrons is grown based on the nominal plan. Finally, a smooth trajectory is generated within the corridor which considers all necessary dynamics and state constraints of the vehicle. Accurate tracking of such trajectory can be expected with a cascaded controller [5]. To generate such a smooth trajectory, a common approach is to formulate a quadratic programming problem using polynomial splines [3, 6]. However, the safe corridor, the dynamic and state constraints could introduce a large number of inequality constraints which prolong the solving of the QP. The is undesired in an unknown and dynamic environment where the current trajectory can be rendered as invalid during execution, and fast replan is needed.

On the other hand, the jerk limited trajectory [7], initially designed for robotic arms have been proven well suited for quadrotors [8] as it could satisfy the maximum thrust and body rate limits. In [9] and [10], realtime algorithms have been designed for online obstacle avoidance. However, they adopt an ad-hoc method for choosing the collision-free trajectory through repeatedly test sampled trajectory until the collision-free one is found.

In this paper, we present a method that combines the safe fly corridor with the jerk limited trajectory to allow the quadrotor to fly in obstacle dense environment. The underlying jerk limited trajectory generation algorithm is exceptionally efficient that solves a trajectory in several microseconds. And the trajectory examination algorithm checks for the collision with corridor walls over continuous time-intervals instead of sampled time-points.

The rest of the paper is organized as follows. Section 2 introduces the quadrotor model while Section 3 discusses the generation of jerk limited trajectory on a single axis. In Section 4, we introduce the safe corridor navigation method which utilizes the jerk limited trajectory. Real flight experiment has been done in Section 5. Finally, in Section 6, a conclusion is made on the presented approach.

2 Quadrotor Dynamic Model

A quadrotor model is usually expressed in a body frame for its rotational movement $\mathcal{B}$ and a global frame $\mathcal{G}$ for its translational movement as shown in Figure 1. Here, the

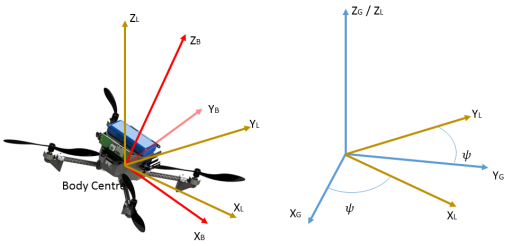

Fig. 1: The coordinate systems

$[\mathbf{x}_\mathrm{B}, \mathbf{y}_\mathrm{B}, \mathbf{z}_\mathrm{B}]$ and $[\mathbf{x}_\mathrm{G}, \mathbf{y}_\mathrm{G}, \mathbf{z}_\mathrm{G}]$ are the axes of the corresponding frame. The quadrotor model can now be expressed as:

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

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

where x, y, z denotes the vehicle's position, g is the gravity and R_B^G is the rotational matrix between body and global

frame. The inputs include $a, \omega_x, \omega_y, \omega_z$ where a is the collective acceleration and $\omega_x, \omega_y, \omega_z$ are the rotational rates about the body axes.

3 Jerk Limited Trajectory

In this section, we present a jerk limited trajectory generation algorithm along a single axis.

3.1 Velocity setpoint

First, we consider the velocity setpoint problem which requires to bring a third order integrator system with an arbitrary initial state to an arbitrary end velocity with the limited jerk and acceleration. It can be expressed as

$$\begin{aligned} \min \quad & t_{\text{end}} \\ s.t. \quad & v(0) = v_0, \quad v(t_{\text{end}}) = v_{\text{ref}} \\ & a(0) = a_0, \quad a(t_{\text{end}}) = 0 \\ & \dot{v}(t) = a(t) \\ & \dot{a}(t) = j(t) \\ & -a_{\max} \leqslant a(t) \leqslant a_{\max}, \ \forall t \in [0, t_{\text{end}}] \\ & -j_{\max} \leqslant j(t) \leqslant j_{\max}, \ \forall t \in [0, t_{\text{end}}] \end{aligned} \tag{3}$$

where v, a, j denotes the velocity, acceleration and jerk on any specific axis and the subscription $\max$ denotes the maximum value, subscription 0 and ref denotes the initial and end conditions, and t_{end} is the total time of the trajectory. A closed-form solution to the problem has been given in [11]. And it is expressed in Algorithm 1. The underlying idea is that to solve the problem in Equation 3, the jerk can only be $\pm j_{\max}$ or zero. First, we try instantly bring the acceleration to zero, check whether the resulted v_{end} is larger or smaller than the desired velocity v_{ref} and determine the cruise direction of the acceleration profile (line 3 – 4). Then we try to bring the acceleration to its maximum magnitude $d_{\text{a}} \cdot a_{\max}$ and immediately down to zero and check whether the resulted velocity overshoots or undershoots the v_{ref} (line 5 – 12). If it overshoots, then the resulted acceleration profile will be wedge-shaped; otherwise, it will be trapezoidal-shaped. For each shape, a set of equations can be solved for a closed-form solution. For convenience, we use the function

$$\mathcal{P} = \text{velT}(v_0, a_0, a_{\max}, j_{\max}, v_{\text{ref}})$$

to denote the Algorithm 1. The acquired $\mathcal{P}$ now contains information on the jerk inputs $\mathcal{P}.j_1, \mathcal{P}.j_2, \mathcal{P}.j_3$ and their corresponding time duration $\mathcal{P}.t_1, \mathcal{P}.t_2 - \mathcal{P}.t_1, \mathcal{P}.t_3 - \mathcal{P}.t_2$. Let p denotes the position, combining the initial conditions p_0, v_0, a_0, it is straight forward to design a function

$$(p_{\text{s}}, v_{\text{s}}, a_{\text{s}}) = \text{StateExam}(v_0, a_0, p_0, \mathcal{P}, t_{\text{s}})$$

to calculate the state of the trajectory $(p_{\text{s}}, v_{\text{s}}, a_{\text{s}})$ at a specific time-point t_{s}.

Algorithm 1 Velocity target solver

1: Input: v_0, a_0 ,$a_{\max}$,$j_{\max}$, v_{ref}
2: Output: $\mathcal{P}$
3: $v_{\text{end}} = v_0 + \dfrac{a_0 \, |a_0|}{2 j_{\max}}$
4: $d_{\text{a}} = \text{sign}(v_{\text{ref}} - v_{\text{end}})$
5: $a_{\text{cruise}} = d_{\text{a}} \cdot a_{\max}$
6: $\Delta t_1 \leftarrow abs(a_{\text{cruise}} - a_0)/j_{\max}$
7: $j_{\text{acc}} \leftarrow j_{\max} \cdot \text{sign}(a_{\text{cruise}} - a_0)$
8: $v_1 \leftarrow v_0 + a_0 \cdot \Delta t_1 + 0.5 \cdot j_{\text{acc}} \cdot \Delta t_1^2$
9: $\Delta t_3 \leftarrow abs(-a_{\text{cruise}})/j_{\max}$
10: $j_{\text{dec}} \leftarrow j_{\max} \cdot \text{sign}(-a_{\text{cruise}})$
11: $\bar{v_3} \leftarrow a_{\text{cruise}} \cdot \Delta t_3 + 0.5 \cdot j_{\text{dec}} \cdot \Delta t_3^2$
12: $\bar{v_2} \leftarrow v_{\text{ref}} - v_1 - \bar{v_3}$
13: **if** $d_{\text{a}} = 0$ **then**
14: $\quad \Delta t_2 \leftarrow 0$
15: **else**
16: $\quad \Delta t_2 \leftarrow \bar{v_2}/a_{\text{cruise}}$
17: **if** $\Delta t_2 < 0$ **then**
18: $\quad a_{\text{cruise}} \leftarrow d_{\text{a}} \cdot \sqrt{d_{\text{a}} \cdot j_{\max} \cdot (v_{\text{ref}} - v_0) + 0.5 \cdot a_0^2}$
19: $\quad \Delta t_1 \leftarrow abs(a_{\text{cruise}} - v_0)/j_{\max}$
20: $\quad \Delta t_2 \leftarrow 0$
21: $\quad \Delta t_3 \leftarrow abs(-a_{\text{cruise}})/j_{\max}$
22: $\mathcal{P}.t_1 \leftarrow \Delta t_1$
23: $\mathcal{P}.t_2 \leftarrow \Delta t_1 + \Delta t_2$
24: $\mathcal{P}.t_3 \leftarrow \Delta t_1 + \Delta t_2 + \Delta t_3$
25: $\mathcal{P}.j_1 = j_1$
26: $\mathcal{P}.j_2 = 0$
27: $\mathcal{P}.j_3 = j_3$

3.2 Position setpoint

Now, we extend the solution in Section 3.1 to a position setpoint problem. The problem has been studied in [11] and [12]. However, the method in [11] has numerical issues while the software in [12] is not open-source available. In this paper, we present an implementation that achieves similar performance on a single axis problem to [12] with detailed pseudo code. Our method runs a binary search for a switching time that separate two different phases. And it is shown efficient and stable for the case of quadrotors. The detail has been given in Algorithm 2. To solve the problem, we first simulate an immediate brake from the current state to determine the cruise velocity direction. The braking trajectory can be solved using Algorithm 1 with $v_{\text{ref}} = 0$. The

Algorithm 2 Position target solver

1: Input: $p_0, v_0, a_0, v_{\max}, a_{\max}, j_{\max}, p_{\text{ref}}$
2: Output: $\mathcal{P}_a, \mathcal{P}_b, t_{\text{pb}}, v_{\text{cruise}}, t_{\text{cruise}}$
3: $\mathcal{P} = \text{velT}(v_0, a_0, a_{\max}, j_{\max}, 0)$
4: $(p_{\text{sp}}, v_{\text{sp}}, a_{\text{sp}}) = \text{StateExam}(v_0, a_0, p_0, \mathcal{P}, \mathcal{P}.t_3)$
5: $d_{\text{p}} = \text{sign}(p_{\text{ref}} - p_{\text{sp}})$
6: $v_{\text{cruise}} = d_{\text{p}} \cdot v_{\max}$
7: $\mathcal{P}_a = \text{velT}(v_0, a_0, a_{\max}, j_{\max}, v_{\text{cruise}})$
8: $(p_{\text{f}}, v, a) = \text{StateExam}(v_0, a_0, p_0, \mathcal{P}_a, \mathcal{P}_a.t_3)$
9: $\mathcal{P}_b = \text{velT}(v_{\text{cruise}}, 0, a_{\max}, j_{\max}, 0)$
10: $(p_{\text{fb}}, v, a) = \text{StateExam}(v_{\text{cruise}}, 0, p_{\text{f}}, \mathcal{P}_b, \mathcal{P}_b.t_3)$
11: $t_{\text{cruise}} = 0$
12: **if** $\text{sign}(p_{\text{fb}} - p_{\text{ref}}) \cdot d_{\text{p}} \leq 0$ **then**
13: $\quad t_{\text{cruise}} = \dfrac{|p_{\text{ref}} - p_{\text{fb}}|}{v_{\text{cruise}}}$
14: $\quad t_{\text{pb}} = \mathcal{P}_a.t_3$
15: **else**
16: $\quad t_{\text{cruise}} = 0$
17: $\quad tH = \mathcal{P}_a.t3$
18: $\quad tL = 0$
19: $\quad$ **for** $counter = 1 : N$ **do**
20: $\quad\quad t_{\text{pb}} = (tH + tL)/2$
21: $\quad\quad (p_{\text{pb}}, v_{\text{pb}}, a_{\text{pb}}) = \text{StateExam}(v_0, a_0, p_0, \mathcal{P}_a, t_{\text{pb}})$
22: $\quad\quad \mathcal{P}_b = \text{velT}(v_{\text{pb}}, a_{\text{pb}}, a_{\max}, j_{\max}, 0)$
23: $\quad\quad (p_{\text{fb}}, v, a) = \text{StateExam}(v_{\text{pb}}, a_{\text{pb}}, p_{\text{pb}}, \mathcal{P}_b, \mathcal{P}_b.t_3)$
24: $\quad\quad$ **if** $\text{sign}(p_{\text{fb}} - p_{\text{ref}}) \cdot d_{\text{p}} < 0$ **then**
25: $\quad\quad\quad tL = t_{\text{pb}}$
26: $\quad\quad$ **else**
27: $\quad\quad\quad tH = t_{\text{pb}}$
28: $\quad\quad$ **if** $|p_{\text{fb}} - p_{\text{ref}}| < \epsilon$ **then**
29: $\quad\quad\quad$ break

resulted stopping point p_{stop} is used to determine the cruise velocity by comparing with the desired target p_{ref} (line 3–6). Then we try to steer the system to reach the cruise velocity v_{cruise} and immediately slowing down to zero speed (line 7–10). The resulting stop point might over or under shoot the p_{ref}, if it undershoots the desired position, then the cruise time is guaranteed to be positive and can be computed in line 13. On the other hand, if it overshoots the desired position, it means the cruise velocity cannot be reached, the system has to slow down to zero speed before reaching v_{cruise}. In other words, we first track the trajectory defined by $p_0, v_0, a_0, \mathcal{P}_a$ for t_{pb} seconds then we start to slow the system to zero velocity. To find the correct t_{pb}, a bisection searching algorithm is adopted (line 16–29). The time-interval we would like to search is $[0, \mathcal{P}_a.t_3]$ as we know if $t_{\text{pb}} = 0$ leads to an undershoot and $t_{\text{pb}} = \mathcal{P}_a.t_3$ leads to an overshoot, then the correct answer must lie between. The searching process is terminated until the final stopping position is close enough to p_{ref}. Using the outputs from Algorithm 2 the resulted trajectory can be reconstructed as follows

1) When $0 \leq t < t_{\text{pb}}$, the trajectory is described by
$$\text{StateExam}(v_0, a_0, p_0, \mathcal{P}_a, t).$$
2) When $t_{\text{pb}} \leq t < t_{\text{cruise}} + t_{\text{pb}}$, the trajectory is at a constant velocity stage and its position is $p = p_{\text{pb}} + v_{\text{cruise}}(t - t_{\text{pb}})$, with
$$(p_{\text{pb}}, v_{\text{pb}}, a_{\text{pb}}) = \text{StateExam}(v_0, a_0, p_0, \mathcal{P}_a, t_{\text{pb}}).$$
The trajectory's velocity equals v_{cruise}, and its acceleration and jerk are all zero.
3) When $t_{\text{cruise}} + t_{\text{pb}} \leq t$, the expression of the trajectory has two different formulations depends on the value of t_{cruise}. If $t_{\text{cruise}} > 0$, it is expressed as
$$\text{StateExam}(v_{\text{cruise}}, 0, p_{1\text{b}}, \mathcal{P}_b, t - (t_{\text{cruise}} + t_{\text{pb}}))$$
$$p_{1\text{b}} = p_{\text{pb}} + v_{\text{cruise}} \cdot t_{\text{cruise}}$$
On the other hand, if there is no cruising phase thus $t_{\text{cruise}} = 0$, it is expressed as
$$\text{StateExam}(v_{\text{pb}}, a_{\text{pb}}, p_{\text{pb}}, \mathcal{P}_b, t - (t_{\text{cruise}} + t_{\text{pb}}))$$

A comparison between our method and the Reflexxes library from [12] is made. For a single axis problem, there is $v_{\max} = 10$, $a_{\max} = 1.5$ and $j_{\max} = 1$. The target point is always $p_{\text{ref}} = 0$ while the initial states are chosen from $p_0 \in [-20, 20], v_0 \in [-20, 20], a_0 \in [-10, 10]$ with a 0.05 increment. 256 million trajectories are generated using both methods using a computer with the Intel I5-4670 CPU running at 3.4 GHz. The resulting file size and the average solving time are given in Table 1. Though slightly slower than the Reflexxes library, the proposed approach reduces the file size and makes it easier to be used on hardware with tight memory space such as the PixHawk [13].

4 Safe Corridor Navigation

In this section, we demonstrate how to use the trajectory generation method in Algorithm 2 to guide the vehicle flying inside a safe corridor. We first introduce the definition of the safe corridor and followed by the discussion on trajectory's feasibility condition within the corridor. Then, we introduce a technique to allow the vehicle to travel through the corridor smoothly.

Table 1: Comparison between reflexxes and our method

	Reflexxes	Our method
Average solving time	2.2 µs	2.4 µs
Size of library file	5072 kB	931 kB
Size of exe file	281 kB	19 kB

4.1 Safe corridor

In this paper, the safe corridor is defined around a nominal plan consists of line-segments. This type of nominal plan are widely available for missions involving obstacle and is usually generated by geometrical shortest path finding algorithms.

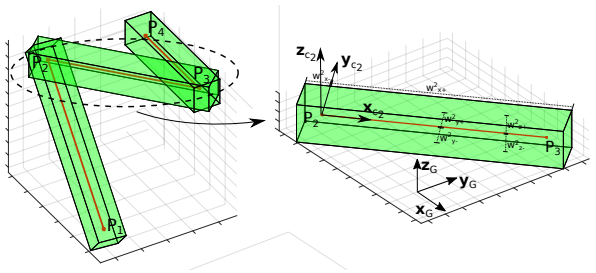

Fig. 2: Nominal plan and flight corridors

As shown in Fig.2, the ith line segment is defined by two points P_i and P_{i+1}. A corridor frame $\mathcal{C}_i$ for the ith line segment, is defined by having its origin at P_i, its x-axis $\mathbf{x}_{\text{C}i}$ aligning $\overrightarrow{P_iP_{i+1}}$ and its y-axis $\mathbf{y}_{\text{C}i}$ parallel to the $\mathbf{x}_{\text{G}} - \mathbf{y}_{\text{G}}$ plane. For each line-segment, we define an oriented bounding box (OBB). The OBB on the ith line-segment is denoted as OBB_i and is defined with 6 positive scalars $w^i_{\text{x}+}, w^i_{\text{x}-}, w^i_{\text{y}+}, w^i_{\text{y}-}, w^i_{\text{z}+}, w^i_{\text{z}-}$ which denotes the width in different directions (the green boxes in Figure 2). It is required for $w^i_{\text{x}+} > \|P_{i+1} - P_i\|$ so that the adjacent OBBs are interconnected. We then call the collection of all OBBs as the safe corridor. For convenience, we also assign the frame information to each OBB as:

$$\text{OBB}_i.\text{frame} = \mathcal{C}_i$$

For the vehicle to travel along the ith line-segment. The trajectory is generated in the frame $\mathcal{C}_i$ whereas the coordinate of the endpoint of the ith line-segment P_{i+1} is $[\|P_{i+1} - P_i\|, 0, 0]$. The Algorithm 2 is then executed three times along the axes $\mathbf{x}_{\text{C}i}, \mathbf{y}_{\text{C}i}, \mathbf{z}_{\text{C}i}$ towards the corresponding targets $\|P_{i+1} - P_i\|, 0, 0$. During the process, we need to choose the velocity, acceleration and jerk limits on each individual axis $v_{\max n}$, $a_{\max n}$, $j_{\max n}$, $n \in \{\mathbf{x}_{\text{C}i}, \mathbf{y}_{\text{C}i}, \mathbf{z}_{\text{C}i}\}$ so that the trajectory satisfy certain feasibility conditions.

4.2 Feasibility condition

The trajectory shall satisfy constraints such as the vehicle's physical limits, safety regulations and sensor specifications. In this paper, we adopt the method in [8] to decouple these limits into constraints on the trajectory's velocity, acceleration and jerk independently. Particularly, we requires these derivatives to satisfy a cylindrical constraint volume (CV). Taking the velocity as an example, it must satisfy

$$\begin{aligned} \sqrt{\dot{x}^2 + \dot{y}^2} &\leq v_{\text{h}\max} \\ v_{\text{v}\min} \leq \dot{z} &\leq v_{\text{v}\max} \end{aligned} \tag{4}$$

and the corresponding CV is shown as the grey cylinder in Figure 3. Similar constraints are also applied on the acceleration and the jerk. These cylindrical CVs reflect the different dynamics between the vertical and horizontal axes of the quadrotor and is intuitive and adjustable to human operators. By properly selecting the height and radius of the cylinders, it can be made to satisfy the vehicle's physical limits [8] and the safety requirements [14].

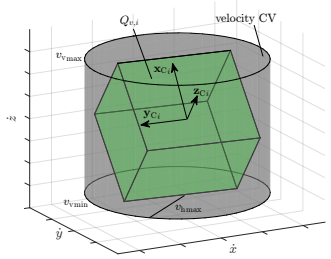

Fig. 3: Constraints on velocity

As discussed in Section 4.1, the task now is to choose the proper value of $v_{\max n}$, $a_{\max n}$, $j_{\max n}$, $n \in \{\mathbf{x}_{Ci}, \mathbf{y}_{Ci}, \mathbf{z}_{Ci}\}$ such that these cylindrical constraints are not violated. We again use the velocity constraint as an example. It is noticed the axes-decoupled limits $v_{\max n}, n \in \{\mathbf{x}_{Ci}, \mathbf{y}_{Ci}, \mathbf{z}_{Ci}\}$ spans a cuboid in $\mathcal{C}_i$, denoted as $Q_{v,i}$ (Figure 3). For convenience, it is given the associated frame

$$Q_{v,i}.\text{frame} = \mathcal{C}_i$$

Since the velocity is translational invariant, we can shift the cylindrical CV of the velocity to the origin of $\mathcal{C}_i$. Then, a sufficient condition for the trajectory to satisfy the velocity related feasibility conditions is that the $Q_{v,i}$ is fully contained inside the cylindrical CV of the velocity. It is achieved by tuning the width, length and height (aka. the value of $v_{\max n}$, $n \in \{\mathbf{x}_{Ci}, \mathbf{y}_{Ci}, \mathbf{z}_{Ci}\}$) of the cuboid. Similarly, we denote the cuboid spanned by $a_{\max n}$, $j_{\max n}$, $n \in \{\mathbf{x}_{Ci}, \mathbf{y}_{Ci}, \mathbf{z}_{Ci}\}$ as $Q_{a,i}$ and $Q_{j,i}$ with the associated frame

$$Q_{a,i}.\text{frame} = \mathcal{C}_i$$
$$Q_{j,i}.\text{frame} = \mathcal{C}_i$$

Since the acceleration and the jerk are also translational invariant, the same sufficient condition can then be applied to them as well.

4.3 Smooth navigation

Assume there are M line segments, the task of the safe corridor navigation is then to guide the vehicle to fly from P_1 to P_{M+1} without leaving the safe corridor. A trivial solution is to move along each line-segment and stop at every point $P_1, P_2, \ldots, P_{M+1}$. The resulting flight path would match the nominal plan exactly thus staying inside the corridor. However, this strategy takes more time and energy due to the frequent stops. In this paper, we then extend this trivial solution so that the unnecessary stops can be removed.

For convenience, we name the trajectory generated in the frame $\mathcal{C}_i$ with the target point P_{i+1} as T_i, and use $\dot{T}_i, \ddot{T}_i, \dddot{T}_i$ to denotes its velocity, acceleration and jerk. Finally, the superscripts $\mathbf{x}_\text{C}, \mathbf{y}_\text{C}, \mathbf{z}_\text{C}$ represents the trajectory's single axis component on the corresponding axis of $\mathcal{C}_i$. For example, $\dot{T}_i^{\mathbf{x}_\text{C}}$ means the single axis component of the velocity profile of the trajectory T_i on the $\mathbf{x}_{Ci}$ axis.

The basic idea of smooth navigation can be summarized as: while the vehicle is tracking trajectory T_i, it constantly generates the trajectory T_{i+1} from its current state. If the trajectory T_{i+1} satisfy the feasibility condition and also stays inside the safe corridor, the vehicle then switches to track T_{i+1}. In this way, the vehicle could efficiently navigate through the corridor without stopping. On the other hand, if the checking fails, the vehicle continues to track T_i. The worst case of the proposed strategy would behave the same as the trivial solution because each trajectory T_i would come to a full stop at its endpoint P_{i+1}. The process is described by the Algorithm 3. The function *getCurrentReference()* returns the currently tracked reference state which includes the position, velocity and acceleration in the global frame. The function *transform(s, c)* transform the state *s* into the desired frame *c*. The transform process consists of a rotation and a translation. For the translational invariant elements, such as the velocity, acceleration and jerk, only the rotation is conducted. The function *genTrajctory($\bar{s}$, c, t)* generates trajectory from the state $\bar{s}$ towards the target *t* by executing Algorithm 2 along the x, y and z axis of the frame *c*. Note the state $\bar{s}$ and target *t* shall already be expressed in the frame *c*. Finally, the *check(T)* function checkes whether the trajectory *T* satisfy the feasibility conditions and stays inside the corridor. In this paper, we assume the first trajectory T_1 satisfies the feasibility condition and stays within the safe corridor. Otherwise, we adopt the method in [9] to generate a temporary trajectory that avoids the obstacles and requires the higher level path planner to provide a new nominal plan. The detail of this method can be found in [9] and is out of the scope of this paper. While the implementation of function *getCurrentReference()*, *transform(s, c)* and *genTrajctory($\bar{s}$, c, t)* is straight forward, *check(T)* remains a more complex process. It involves the checking of the feasibility condition and the checking of corridor violation.

Algorithm 3 Smooth navigation

```
1: i = 0
2: while Not reaching P_{M+1} do
3:     if i < M then
4:         s = getCurrentReference()
5:         s̄ = transform(s, C_{i+1})
6:         T_{i+1} = genTrajctory(s̄, C_{i+1}, [‖P_{i+1} − P_i‖, 0, 0])
7:         if check(T_{i+1}) then
8:             i = i + 1
```

4.3.1 Checking for feasibility condition

For each single-axis component of the trajectory T_{i+1}, namely $T_{i+1}^{\mathbf{x}_\text{C}}, T_{i+1}^{\mathbf{y}_\text{C}}, T_{i+1}^{\mathbf{z}_\text{C}}$, it consists of at most 7 segments of 3rd order polynomials at C_2 continuity [7]. Therefore $\dot{T}_{i+1}^n, \ddot{T}_{i+1}^n$, $n \in \{\mathbf{x}_\text{C}, \mathbf{y}_\text{C}, \mathbf{z}_\text{C}\}$ would consist of at most 7 segments of 2nd and 1st order polynomials.

Taking the velocity trajectory $\dot{T}_{i+1}$ as an example, we can check whether the trajectory $\dot{T}_{i+1}$ stays inside $Q_{v,i+1}$ by calling the method *boxCheck($\dot{T}_{i+1}$, $Q_{v,i+1}$)* in Algorithm 4. It checks whether a given polynomial spline T is inside a cuboid B. It assumes the cuboid B has been assigned with an associated frame, such is the case for all OBB_i, $Q_{v,i}$,

Algorithm 4 boxCheck

1: Inputs: T, B
2: Output: isInside
3: $\bar{T}$ = transform(T, B.frame)
4: $[x_{\min}, x_{\max}]$ = findMinMax($\bar{T}^x$)
5: $[y_{\min}, y_{\max}]$ = findMinMax($\bar{T}^y$)
6: $[z_{\min}, z_{\max}]$ = findMinMax($\bar{T}^z$)
7: isInside = false
8: **if** isInBox($[x_{\max}, y_{\max}, z_{\max}]$, B) **then**
9: **if** isInBox($[x_{\min}, y_{\min}, z_{\min}]$, B) **then**
10: isInside = true

$Q_{a,i}$ and $Q_{j,i}$. It first transforms the entire spline T into the associated frame of B (line 3). If the spline denotes the velocity, acceleration or jerk, only the rotation in the transform process is performed. For the transformed spline $\bar{T}$, we the then find its maximum and minimum value on each axis of the B.frame (line 4 – 6). Finally, for the possible furthest points $[x_{\max}, y_{\max}, z_{\max}]$ and $[x_{\min}, y_{\min}, z_{\min}]$, we check whether they are inside the box B. If both points are contained by B, so does the entire trajectory. If both $\ddot{T}_{i+1}$ and $\dddot{T}_{i+1}$ are also contained in $Q_{a,i}$ and $Q_{j,i}$ then we conclude the trajectory T_{i+1} is feasible. The same idea can also be extended to handle the cylindrical constraints. We first define a new trajectory that is the norm of $[\bar{T}^x, \bar{T}^y]$. Then we check whether its maximum value is inside the cylindrical volume. If the check is successful, we also check whether the extreme value of $\bar{T}^z$ is within the cylindrical volume.

4.3.2 Checking for corridor violation

To check whether the corridor constraint is violated by T_{i+1} is more difficult as the trajectory might passes through multiple OBBs. As a sufficient but not necessary condition, we require the trajectory T_{i+1} to be within OBB_i or OBB_{i+1} by calling *corridorCheck(T_{i+1}, OBB_i, OBB_{i+1})* as in Algorithm 5. It first try to find a middle point P_{mid}

Algorithm 5 corridorCheck

1: Input: T, OBB_a, OBB_b
2: Output: isInCorridor
3: $[P_{\mathrm{mid}}, T_a, T_b]$ = splitTrajectory(T, OBB_1, OBB_2)
4: **if** $P_{\mathrm{mid}} = \varnothing$ **then**
5: isInCorridor = false
6: return
7: **if** boxcheck(T_a, OBB_a) && boxcheck(T_b, OBB_b) **then**
8: isInCorridor = true
9: **else**
10: isInCorridor = false

which is inside both OBB_a and OBB_b and then split the trajectory into two parts T_a and T_b (line 3). If such a point cannot be found, then the trajectory must have leave the safe corridor (line 4 – 6). Then, we check whether T_a and T_b is fully contained by the bounding boxes OBB_a and OBB_b (line 7). To find the point P_{mid}, a bisection search procedure is adopted (Algorithm 5). Given a trajectory T, let $T(\tau)$ denotes the corresponding point at time τ and $T(\tau_a : \tau_b)$ denotes the partial trajectory between time τ_a and τ_b. The function *getTotoalTime(T)* returns the total time of the trajectory T until it comes to a full stop. The bisection is performed over the time length of the trajectory where a middle time τ_{mid} splits the trajectory into two parts T_a and T_b

Table 2: Per-cycle computational time consumption

	Average time consumption
Total	10.20 µs
Trajectory generation	2.55 µs
Switch checking	0.98 µs

with the splitting point p (line 10 – 12). If p is in both the first OBB (OBB_a) and the second OBB (OBB_b), the search is terminated successfully. Otherwise, if p is inside OBB_a but not OBB_b, the search is continued on T_b (line 16 – 17). Similarly, if p is inside OBB_b but not OBB_a, the search is continued on T_a (line 18 –19). Finally, if p is not contained by neither OBBs, the algorithm reports failure with an empty P_{mid}. If T_{i+1} passes both the feasibility checking (Section

Algorithm 6 splitTrajectory

1: Input: T, OBB_a, OBB_b
2: Output: $P_{\mathrm{mid}}, T_a, T_b$
3: $P_{\mathrm{mid}} = \varnothing$
4: $\tau_a = 0$
5: τ_{end} = getTotoalTime(T)
6: $\tau_b = \tau_{\mathrm{end}}$
7: **for** $i < N_{\mathrm{e}}$ **do**
8: $i = i + 1$
9: $\tau_{\mathrm{mid}} = (\tau_a + \tau_b)/2$
10: $p = T(\tau_{\mathrm{mid}})$
11: $T_a = T(0 : \tau_{\mathrm{mid}})$
12: $T_b = T(\tau_{\mathrm{mid}} : \tau_{\mathrm{end}})$
13: **if** $p \in \mathrm{OBB}_a$ && $p \in \mathrm{OBB}_b$ **then**
14: $P_{\mathrm{mid}} = p$
15: return
16: **else if** $p \in \mathrm{OBB}_a$ && $p \notin \mathrm{OBB}_b$ **then**
17: $\tau_a = \tau_{\mathrm{mid}}$
18: **else if** $p \notin \mathrm{OBB}_a$ && $p \in \mathrm{OBB}_b$ **then**
19: $\tau_b = \tau_{\mathrm{mid}}$
20: **else**
21: return

4.3.1) and the corridor checking (Section 4.3.2), the vehicle could start to track this new trajectory T_{i+1} safely.

4.4 Computational performance

The checking for feasibility and corridor violation has been performed analytically. For each control cycle, the algorithm provides the reference for the next cycle and checks whether it is feasible to switch to the next OBB. Together with the efficient trajectory generation method, the overall algorithm achieves realtime performance with small computational consumption. For the corridor shown in Figure 4, our algorithm consumes around 10 microseconds in one control cycle on an Intel I5 CPU (see Table 2). Considering the fact that the translational controller is usually running at 20 to 50 Hz, it is more than enough to achieve a realtime performance. Compared to the method in [15] which solves the entire trajectory in a quadratic programming problem, our algorithm is expected to be more suitable for micro-sized vehicles with limited computational power due to its smaller computational footage.

5 Flight experiment

To demonstrate the effectiveness of our method, real flight experiments and simulation has been performed. The testing vehicle is connected to a desktop computer wirelessly which provides measurement and reference signals. The proposed method is running on the desktop computer at the frequency

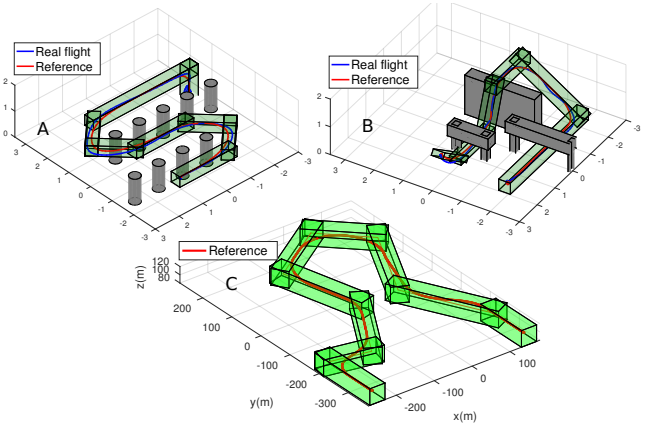

Fig. 4: Experiments

Table 3: Fly time consumption

	Exp A	Exp B	Exp C
Our method	11.05s	13.05s	123.8s
Trivial method	23.7s	23.45s	211.5s
Improvement	53%	44%	41%

of the outer loop controller at 20 Hz. The results are shown in Figure 4. In experiment A, the vehicle is tasked to fly through multiple pillars. The nominal plan is generated with a traditional A* algorithm and gives a corridor consists of 8 OBBs. In experiment B, the vehicle is tasked to fly through corridors with 6 OBBs with rapid height variance. Finally, in experiment C, a simulation study is performed on a task with larger scale and faster flying speed which is commonly seen in the real-life application such as the power line inspection. The state constraints on the velocity, acceleration, and jerk for each experiment is given in Table 4. Compared to the trivial solution which comes to a full stop at each endpoint, our method saves more than 40% of fly time in both three experiments. The video of the real flight experiment A and B can be found at `http://uav.ece.nus.edu.sg/videos_files/2018/jlt.mp4`

6 Conclusion

In this paper, we present a method to guide the quadrotor to fly through a safe corridor with jerk limited trajectory. It extends a trivial solution which stops at every intermediate point to achieve smooth flying while still guarantees to operate entirely inside the corridor. This is done by repeatedly checking the validity of the newly generated trajectory. Our continuous checking algorithm guarantees the satisfaction of various constraints throughout the entire trajectory. On the other hand, the trajectory's velocity, acceleration, and jerk can be limited in a cylindrical constraint volume to satisfy the vehicle's physical limits or safety regulations. As future works, we are testing other efficient trajectory generation and checking algorithm using neural-networks to achieve a more smooth flight experience.

Table 4: State constraints

	Exp A	Exp B	Exp C
$v_{\mathrm{h}\max}$	$4\ \mathrm{m/s}$	$3\ \mathrm{m/s}$	$15\ \mathrm{m/s}$
$v_{\mathrm{v}\max}$	$0.8\ \mathrm{m/s}$	$0.8\ \mathrm{m/s}$	$0.8\ \mathrm{m/s}$
$a_{\mathrm{h}\max}$	$2.2\ \mathrm{m/s^2}$	$1.2\ \mathrm{m/s^2}$	$2.2\ \mathrm{m/s^2}$
$a_{\mathrm{v}\max}$	$0.8\ \mathrm{m/s^2}$	$0.8\ \mathrm{m/s^2}$	$0.8\ \mathrm{m/s^2}$
$j_{\mathrm{h}\max}$	$3\ \mathrm{m/s^3}$	$3\ \mathrm{m/s^3}$	$3\ \mathrm{m/s^3}$
$j_{\mathrm{v}\max}$	$3\ \mathrm{m/s^3}$	$3\ \mathrm{m/s^3}$	$3\ \mathrm{m/s^3}$

References

[1] R. Allen and M. Pavone, "A real-time framework for kinodynamic planning with application to quadrotor obstacle avoidance," in *AIAA Guidance, Navigation, and Control Conference*, 2016.

[2] S. Roelofsen, D. Gillet, and A. Martinoli, "Reciprocal collision avoidance for quadrotors using on-board visual detection," in *2015 IEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2015, pp. 4810–4817.

[3] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 2520–2525.

[4] J. Li, H. Zhan, B. M. Chen, I. Reid, and G. H. Lee, "Deep learning for 2d scan matching and loop closure," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep 2017, pp. 763–768.

[5] S. K. Phang, S. Lai, F. Wang, M. Lan, and B. M. Chen, "Uav calligraphy," in *11th IEEE International Conference on Control Automation (ICCA)*, June 2014, pp. 422–428.

[6] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for quadrotor flight," in *RSS Workshop on Resource-Efficient Integratiton of Perception, Control and Navigation for MAVs*, 2013.

[7] T. Kröger and F. M. Wahl, "Online trajectory generation: Basic concepts for instantaneous reactions to unforeseen events," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 94–111, Feb 2010.

[8] M. Hehn and R. DAndrea, "Real-time trajectory generation for quadrocopters," *IEEE Transactions on Robotics*, vol. 31, no. 4, pp. 877–892, Aug 2015.

[9] S. Lai, K. Wang, H. Qin, J. Q. Cui, and B. M. Chen, "A robust online path planning approach in cluttered environments for micro rotorcraft drones," *Control Theory and Technology*, vol. 14, no. 1, pp. 83–96, Feb 2016.

[10] B. T. Lopez and J. P. How, "Aggressive 3-d collision avoidance for high-speed navigation," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 5759–5765.

[11] R. Haschke, E. Weitnauer, and H. Ritter, "On-line planning of time-optimal, jerk-limited trajectories," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2008, pp. 3248–3253.

[12] T. Kröger, "Opening the door to new sensor-based robot applications – the reflexxes motion libraries," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 1–4.

[13] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "Pixhawk: A system for autonomous flight using onboard computer vision," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 2992–2997.

[14] K. Z. Y. Ang, X. Dong, W. Liu, G. Qin, S. Lai, K. Wang, D. Wei, S. Zhang, P. S. King, X. Chen, M. Lao, Z. Yang, D. Jia, F. Lin, L. Xie, and B. M. Chen, "High-precision multi-uav teaming for the first outdoor night show in singapore," *Unmanned Systems*, vol. 06, no. 01, pp. 39–65, 2018.

[15] J. Chen, T. Liu, and S. Shen, "Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 1476–1483.