Safe corridor based task interface for quadrotors

Lai Shupeng*, Lan Menglu, and Ben M. Chen
National University of Singapore, 21 Lower Kent Ridge Rd, Singapore

ABSTRACT

Most of the task management interface of quadrotors is based on waypoints. While it is natural for the human to visualize the tasks and easy for quadrotors to execute, there lacks support for functions such as the geo-fencing. In many applications, it is desired to limit the quadrotors' operation area in the safe region. In this paper, we propose an approach that guides the vehicle in a predefined safe corridor without solving the entire trajectory beforehand. It also guarantees the feasibility by limiting the trajectory's velocity, acceleration, and jerk to a predefined range. The effectiveness of the proposed approach is demonstrated with real flight experiment.

1 INTRODUCTION

Quadrotors are used more and more frequently in industrial applications such as inspection, monitoring, and surveillance due to its agility and easy-to-maintain mechanical structure. The quadrotor's mission is usually described by a series of waypoints where it is expected to travel in sequence. Though the waypoint based missions are easy for the human to visualize and edit, it is a non-trivial task to ensure the resulting trajectory is still suitable. As in these applications, the quadrotor is usually required to be operated in a safe area with no apparent obstacles such as tall buildings, and its possible crash will have limited damage. Methods in [1] and [2] achieved this by building a safe-flying corridor connecting the waypoints. The size of the corridor can be adjusted, and the trajectory is restricted inside the corridor through constrained quadratic programming. This approach requires a more powerful on-board computer, especially in the case where a replanning is needed, and the data link is not reliable enough thus planning on a remote computer is not an option. In this paper, we propose a safe corridor based task interface where the user could edit the mission quickly, and the safe trajectory could be generated efficiently which benefits the vehicles with weaker onboard computers. The rest of this paper is organized as the following. In Section 2, the safe-flying corridor used in our interface is introduced. In Section 3, we present an incremental approach to generate jerk, acceleration and velocity limited trajectory that stays inside the safe-flying corridor. And in Section 4, experimental results are analyzed and discussed. Finally, a conclusion is made in Section 5.

*Email address(es): elelais@nus.edu.sg

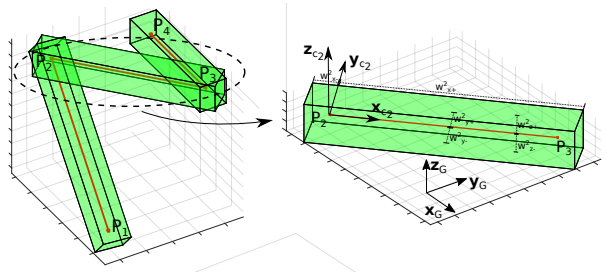

Figure 1: Nominal plan and flight corridors

2 SAFE-FLYING CORRIDOR

Given a list of waypoints, we call the line-segments constructed by connecting the waypoints in sequence as the nominal plan. And the safe-flying corridor is built around such a nominal plan. As shown in Figure 1, the nominal plan is defined by the waypoints P_1 to P_4 in the global frame $\mathcal{G}$. For each line segment defined by P_i and P_{i+1}, a local frame $\mathcal{C}_i$ is defined where its $x_{\mathcal{C}_i}$ axis is aligned to the vector $\overrightarrow{P_iP_{i+1}}$ and its $y_{\mathcal{C}_i}$ axis is perpendicular to the gravity direction. A safe bounding box (the green cuboid in Figure 1) aligned with the local frame is then adapted to enclose the line-segment. In this way, a safe region could be constructed around the nominal plan, and its size can be adjusted by setting the dimension of each safe bounding box. Given a trajectory $\mathbf{T}$, $t \in [t_0, t_f]$ in the 3 dimensional space as

$$\mathbf{T}(t) = \begin{cases} f_x(t) \\ f_y(t) \\ f_z(t) \end{cases}$$

we can check whether it is inside a safe bounding box by:

- Project the trajectory $\mathbf{T}$ into the local frame $\mathcal{C}_i$ of the safe bounding box as

$$\mathbf{T}_{\mathcal{C}}(t) = \begin{cases} f_{x_{\mathcal{C}}}(t) \\ f_{y_{\mathcal{C}}}(t) \\ f_{z_{\mathcal{C}}}(t) \end{cases}$$

- Find the minimum and maximum value of $f_{x_{\mathcal{C}}}, f_{y_{\mathcal{C}}}, f_{z_{\mathcal{C}}}$.
- Check whether all of the minimum and maximum values are inside the safe bounding box. If so, the entire trajectory $\mathbf{T}$ will be enclosed by the safe bounding box.

Using this method, we could decouple the enclosure checking problem into three extreme value searching problems. And if

the trajectory is in the form of a polynomial, the searching could be done efficiently through root finding.

3 Incremental Safe Trajectory Generation

3.1 Jerk limited trajectory

As a basic tool used in the incremental safe trajectory generation process, we present an approach based on the jerk limited trajectory appeared in [3], which is later improved in [4], and shown effective for quadrotors in [5]. In this paper, we present an approach that solves the position set-point problem using a direct bisection search, which does not require to build decision trees as in [3, 4]. And we also allow to set asymmetrical constraints on the velocity, acceleration and jerk.

3.1.1 Problem formulation

Given a triple integrator system

$$\begin{aligned} \dot{p} &= v \\ \dot{v} &= a \\ \dot{a} &= j \end{aligned} \tag{1}$$

where p, v, a, j are the position, velocity, acceleration and jerk respectively and the jerk j also serves as the system's input. The presented algorithm aims to bring the system in Equation 1 from an arbitrary initial state to a position set-point while satisfying constraints on the velocity, acceleration and jerk:

$$\begin{aligned} &p(0) = p_0, \quad p(t_f) = p_f \\ &v(0) = v_0, \quad v(t_f) = 0 \\ &a(0) = a_0, \quad a(t_f) = 0 \\ &v_{\min} \leqslant v(t) \leqslant v_{\max}, \ \forall t \in [0, t_f] \\ &a_{\min} \leqslant a(t) \leqslant a_{\max}, \ \forall t \in [0, t_f] \\ &j_{\min} \leqslant j(t) \leqslant j_{\max}, \ \forall t \in [0, t_f] \end{aligned} \tag{2}$$

To make sure an solution does exist, it is assumed

$$\begin{aligned} v_{\min} < 0 < v_{\max} \\ a_{\min} < 0 < a_{\max} \\ j_{\min} < 0 < j_{\max} \end{aligned}$$

3.1.2 Velocity set-point problem

Here, we first introduce the velocity set-point problem described in [3], the task is to bring the system in Equation 1 from an arbitrary initial state to n velocity set-point:

$$\begin{aligned} &v(0) = v_0, \quad v(t_f) = v_f \\ &a(0) = a_0, \quad a(t_f) = 0 \\ &a_{\min} \leqslant a(t) \leqslant a_{\max}, \ \forall t \in [0, t_f] \\ &j_{\min} \leqslant j(t) \leqslant j_{\max}, \ \forall t \in [0, t_f] \end{aligned} \tag{3}$$

Unlike in [3], in our formulation, we allow asymmetrical limits on the acceleration and jerk. Our solution is based on the one in [3], and it is shown in Algorithm 1. First, we try instantly bring the acceleration to zero, check whether the resulted v_{e} is larger or smaller than the desired velocity v_f and determine the cruise direction of the acceleration profile (line 3 – 8). Then depends on the cruise direction, we try to bring the acceleration either to its maximum or minimum value, and check whether the resulted velocity over or undershoots v_f (line 9 – 37). If it undershoots, there will be an cruise phase with non-negative time endurance (line 36), otherwise we solve for the switching acceleration (line 43 and 48) depending on the cruise sign. The final result is an parameter structure holding the desired jerks and their corresponding endurance. For simplicity, we use the function

$$\mathcal{P} = \mathrm{solveVelocity}(v_0,\, a_0,\, a_{\max},\, a_{\min},\, j_{\max},\, j_{\min},\, v_f)$$

to denote the calculation of $\mathcal{P}$ in algorithm 1. With the initial state p_0, v_0, a_0 and $\mathcal{P}$, it is straight forward to reconstruct the trajectory through the model in Equation 1. The function

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

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

3.1.3 Position set-point problem

Now, we extend the solution to cover the position set-point problem in Equation 2 which has been studied in [3] and [6]. Our method is based on a bisection search to find the solution rather than using decision trees. The detail of our algorithm can be seen in Algorithm 2.

To identify the cruise direction, we first solve for the braking trajectory that immediately brings the velocity and acceleration both to zero (line 3). The resulted stopping point p_{sp} is used to determine the cruise velocity by comparing with the desired target p_f (line 3–12). Then we create the zero cruise profile by steering the system to the cruise velocity and immediately to full stop (line 13–16).

The resulting stop point might over or undershoot the p_f. If it undershoots the desired position, then the non-negative cruise time can be found as in line 19. And if it overshoots the desired position, then the cruise velocity cannot be reached, and we switch the system towards zero speed before reaching v_{c}. The exact switching time is found through a bi-section search (line 22–38). With the parameters for the two different phases $\mathcal{P}_a$, $\mathcal{P}_b$ and the switching time t_{pb}, the trajectory can be constructed using the triple integrator model. An example of such a trajectory with asymmetrical constraints on its velocity, acceleration and jerk is given in Figure 2.

Algorithm 1 Velocity target solver

Input: v_0, a_0 ,$a_{\max}$,$a_{\min}$,$j_{\max}$, $j_{\min}$, v_f
Output: $\mathcal{P}$
if $a0 \geq 0$ **then**
  $v_{\mathrm{e}} = v_0 + a_0 \left| a_0 / j_{\min} \right| / 2$
else
  $v_{\mathrm{e}} = v_0 + a_0 \left| a_0 / j_{\max} \right| / 2$
end if
$d_{\mathrm{a}} = \mathrm{sign}(v_f - v_{\mathrm{e}})$
if $d_{\mathrm{a}} == 1$ **then**
  $a_{\mathrm{c}} = a_{\max}$
else if $d_{\mathrm{a}} == -1$ **then**
  $a_{\mathrm{c}} = a_{\min}$
else
  $a_{\mathrm{c}} = 0$
end if
if $a_{\mathrm{c}} - a_0 \geq 0$ **then**
  $t_1 = (a_{\mathrm{c}} - a_0)/j_{\max}$
  $j_1 = j_{\max}$
else
  $t_1 = (a_{\mathrm{c}} - a_0)/j_{\min}$
  $j_1 = j_{\min}$
end if
$v_1 = v_0 + a_0 t_1 + t_1^2 j_1 / 2$
if $-a_{\mathrm{c}} \geq 0$ **then**
  $t_3 = (-a_{\mathrm{c}})/j_{\max}$
  $j_3 = j_{\max}$
else
  $t_3 = (-a_{\mathrm{c}})/j_{\min}$
  $j_3 = j_{\min}$
end if
$\bar{v}_3 = a_{\mathrm{c}} t_3 + t_3^2 j_3 / 2$
$\bar{v}_2 = v_f - v_1 - \bar{v}_3$
if $d_{\mathrm{a}} == 0$ **then**
  $t_2 = 0$
else
  $t_2 = \bar{v}_2 / a_{\mathrm{c}}$
end if
if $t_2 < 0$ **then**
  if $d_{\mathrm{a}} == 1$ **then**
    $a_{\mathrm{n}} = \sqrt{(2(v_f - v_0) + a_0^2/j_{\max})/(1/j_{\max} - 1/j_{min})}$
    $t_1 = (a_{\mathrm{n}} - a_0)/j_{\max}$
    $t_2 = 0$
    $t_3 = -a_{\mathrm{n}}/j_{\min}$
  else if $d_{\mathrm{a}} == -1$ **then**
    $a_{\mathrm{n}} = -\sqrt{(2(v_f - v_0) + a_0^2/j_{\min})/(1/j_{\min} - 1/j_{max})}$
    $t_1 = (a_{\mathrm{n}} - a_0)/j_{\min}$
    $t_2 = 0$
    $t_3 = -a_{\mathrm{n}}/j_{\max}$
  end if
end if
$\mathcal{P}.T_1 = t_1$
$\mathcal{P}.T_2 = t_2 + t_1$
$\mathcal{P}.T_3 = t_3 + t_2 + t_1$
$\mathcal{P}.j_1 = j_1$
$\mathcal{P}.j_2 = 0$
$\mathcal{P}.j_3 = j_3$

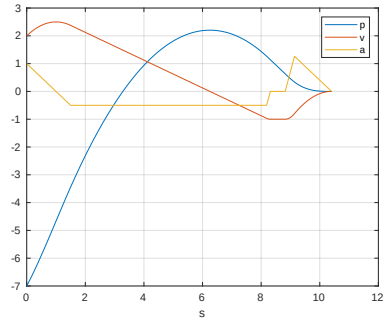

Figure 2: Trajectory with asymmetrical constraints. The position set-point is at zero position.

Algorithm 2 Position target solver

Input: $p_0, v_0, a_0, v_{\max}, a_{\max}, j_{\max}, v_{\min}, a_{\min}, j_{\min}, p_f$
Output: $\mathcal{P}_a, \mathcal{P}_b, t_{\mathrm{pb}}, v_{\mathrm{c}}, t_{\mathrm{c}}$
$\mathcal{P} = \mathrm{solveVelocity}(v_0, a_0, a_{\max}, a_{\min}, j_{\max}, j_{\min}, 0)$
$(p_{\mathrm{sp}}, v_{\mathrm{sp}}, a_{\mathrm{sp}}) = \mathrm{getState}(v_0, a_0, p_0, \mathcal{P}, \mathcal{P}.T_3)$algorithmic
$d_{\mathrm{p}} = \mathrm{sign}(p_f - p_{\mathrm{sp}})$
if $d_{\mathrm{p}} == 1$ **then**
  $v_{\mathrm{c}} = v_{\max}$
else if $d_{\mathrm{p}} == -1$ **then**
  $v_{\mathrm{c}} = v_{\min}$
else
  $v_{\mathrm{c}} = 0$
end if
$\mathcal{P}_a = \mathrm{solveVelocity}(v_0, a_0, a_{\max}, a_{\min}, j_{\max}, j_{\min}, v_{\mathrm{c}})$
$(p_{\mathrm{fa}}, v, a) = \mathrm{getState}(v_0, a_0, p_0, \mathcal{P}_a, \mathcal{P}_a.T_3)$
$\mathcal{P}_b = \mathrm{solveVelocity}(v_{\mathrm{c}}, 0, a_{\max}, a_{\min}, j_{\max}, j_{\min}, 0)$
$(p_{\mathrm{fb}}, v, a) = \mathrm{getState}(v_{\mathrm{c}}, 0, p_{\mathrm{fa}}, \mathcal{P}_b, \mathcal{P}_b.T_3)$
$t_{\mathrm{c}} = 0$
if $\mathrm{sign}(p_{\mathrm{fb}} - p_f) \cdot d_{\mathrm{p}} \leq 0$ **then**
  $t_{\mathrm{c}} = \dfrac{(p_f - p_{\mathrm{fb}})}{v_{\mathrm{c}}}$
  $t_{\mathrm{pb}} = \mathcal{P}_a.T_3$
else
  $t_{\mathrm{c}} = 0$
  $tH = \mathcal{P}_a.T_3$
  $tL = 0$
  for *counter* $= 1 : N$ **do**
    $t_{\mathrm{pb}} = (tH + tL)/2$
    $(p_{\mathrm{pb}}, v_{\mathrm{pb}}, a_{\mathrm{pb}}) = \mathrm{getState}(v_0, a_0, p_0, \mathcal{P}_a, t_{\mathrm{pb}})$
    $\mathcal{P}_b = \mathrm{solveVelocity}(v_{\mathrm{pb}}, a_{\mathrm{pb}}, a_{\max}, a_{\min}, j_{\max}, j_{\min}, 0)$
    $(p_{\mathrm{fb}}, v, a) = \mathrm{getState}(v_{\mathrm{pb}}, a_{\mathrm{pb}}, p_{\mathrm{pb}}, \mathcal{P}_b, \mathcal{P}_b.T_3)$
    if $\mathrm{sign}(p_{\mathrm{fb}} - p_f) \cdot d_{\mathrm{p}} < 0$ **then**
      $tL = t_{\mathrm{pb}}$
    else
      $tH = t_{\mathrm{pb}}$
    end if
    if $|p_{\mathrm{fb}} - p_f| < \epsilon$ **then**
      break
    end if
  end for
end if

To test the efficiency of the proposed method, the initial position, velocity and acceleration is set to be in the range of $[-50, 50]$, $[-10, 10]$ and $[-5, 5]$ with an incremental of 0.05. Without loss of the generality, the position set-point is always zero. And the $v_{\max}, a_{\max}, j_{\max}, v_{\min}, a_{\min}, j_{\min}$ is set as $4, 4, 2, -1, -1, -1$ accordingly. A total of 160880400 trajectories are generated and the average computing time is 0.31 microseconds with an i5-3470S CPU at 2.9 GHz.

3.2 Navigation in the safe corridor

Using the proposed jerk limited trajectory generation algorithm, an online and incremental approach is proposed to generate a safe trajectory that stays fully inside the safe corridor. As in Figure 1, let L_i denotes the line-segment defined by waypoints P_i and P_{i+1}, the corresponding local frame is $\mathcal{C}_i$ and the safe bonding box that enclose L_i is denoted B_i.

3.2.1 Generate trajectory in the local frame

In the local frame $\mathcal{C}_i$, we can generate a trajectory R_i that starts from an arbitrary state and stops at waypoint P_{i+1} by solving the position set-point problem on each axis of $\mathcal{C}_i$ (namely $x_{\mathcal{C}_i}$, $y_{\mathcal{C}_i}$ and $z_{\mathcal{C}_i}$) independently. Since the origin of $\mathcal{C}_i$ is at waypoint P_i and the target is P_{i+1}, the position set-point on the $x_{\mathcal{C}_i}$ axis is $\|P_{i+1} - P_i\|$. And the position set-points on $y_{\mathcal{C}_i}$, $z_{\mathcal{C}_i}$ are 0. Moreover, we have to assign the velocity, acceleration and jerk limits on each axis of $\mathcal{C}_i$, namely $x_{\mathcal{C}_i}$, $y_{\mathcal{C}_i}, z_{\mathcal{C}_i}$. As shown in [5], the physical limits of the quadrotor can be satisfied by limiting the trajectory's velocity, acceleration and jerk separately. However, these desired limits are usually defined in the global frame G and need to be projected into $\mathcal{C}_i$. Let $v_{\mathcal{G}} = [v_{\mathcal{G}_x}, v_{\mathcal{G}_y}, v_{\mathcal{G}_z}]$ denote the velocity in the global frame, a common practice is to have

$$\begin{aligned} \sqrt{v_{\mathcal{G}_x}^2 + v_{\mathcal{G}_y}^2} &< v_{\mathrm{h}_{\max}} \\ v_{\mathrm{v}_{\min}} \leq v_{\mathcal{G}_z} &\leq v_{\mathrm{v}_{\max}} \end{aligned} \tag{4}$$

because the quadrotor have similar dynamics in its horizontal axes compared to the vertical axis. The constraints span a cylindrical volume shown in Figure 3. However, in the $\mathcal{C}_i$ frame, the velocity constraints need to be decoupled into each individual axis. Let $v_{\mathcal{C}_i} = [v_{\mathcal{C}_{i_x}}, v_{\mathcal{C}_{i_y}}, v_{\mathcal{C}_{i_z}}]$ represent the velocity in the local frame, the constraint is

$$v_{\mathcal{C}_{i\lambda_{\min}}} \leq v_{\mathcal{C}_{i\lambda}} \leq v_{\mathcal{C}_{i\lambda_{\max}}}, \forall \lambda \in \{x, y, z\}$$

which spans an axis-aligned cuboid $Q_{v,i}$ in $\mathcal{C}_i$. Therefore, it is necessary to select the limits such that the cuboid is entirely inside the cylinder (see Figure 3). The same axis-decoupling criterion also applies to the acceleration and the jerk. Furthermore, the width of the spanned cuboid (Figure 3) needs to be large enough on all axis, so that the vehicle could move agilely towards any direction at any moment. It is crucial if an evasive maneuver is needed which might deviate from the planned trajectory.

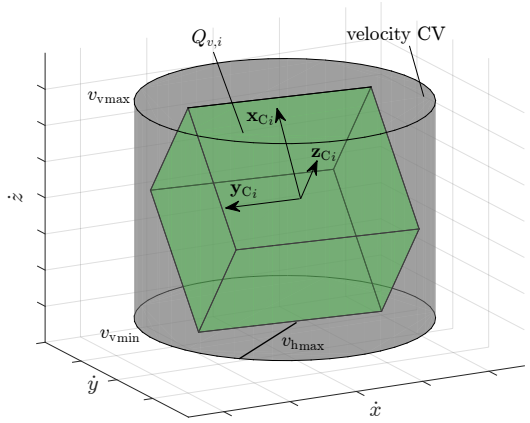

Figure 3: The volume spanned by constraints at $\mathcal{G}$ and $\mathcal{C}_i$.

3.2.2 Continuous navigation

With the capability to reach an arbitrary waypoint in the desired frame, we now consider navigating the quadrotor through multiple safe flying corridors. A trivial approach is to fly along each line-segments and stops at each waypoint. Here, we propose an approach to guide the vehicle inside the safe corridor without stopping at each waypoint. Assume there are total M waypoints and the vehicle is initially inside B_1, the proposed algorithm can be expressed in Algorithm 3. The idea is to repeatedly generate a new trajectory R_{i+1}

Algorithm 3 Smooth navigation

```
1:  i = 0
2:  while Not reaching P_M do
3:      if i < M then
4:          s = get_current_reference()
5:          s̄ = project(s, C_{i+1})
6:          R_{i+1} = generate(s̄, C_{i+1}, P_{i+1})
7:          if exam(R_{i+1}) then
8:              Start tracking R_{i+1}
9:              i = i + 1
10:         end if
11:     end if
12: end while
```

that connects the vehicle from its current reference state s to the new position set-point P_{i+1}, and then exam whether this new trajectory is safe. We first get the current reference state the vehicle is tracking (line 4), then it is projected into the local frame $\mathcal{C}_i$ and the new trajectory is generated from $\bar{s}$ to set-point P_{i+1} (line 5 – 6). And once R_{i+1} is considered safe, the vehicle could start to track it and proceed to try the next waypoint (line 7 – 10). While the implementation of get_current_reference() and project() is straight forward, the generate() function adopts the method in Section 3.2.1. Finally, the exam() function checks whether the tra-

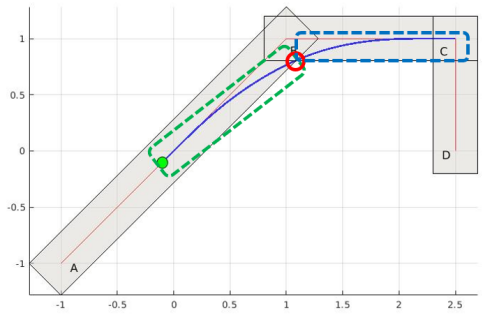

Figure 4: Split the trajectory into two segments, then check whether each individual part is inside a single bounding box.

jectory is fully inside the safe corridor and the satisfaction of constraints in the global frame $\mathcal{G}$.

3.2.3 Safety check

In [1], the author first samples the trajectory at multiple time instances, and then check each sample individually. Here, a continuous checking method is adopted due to the fact the trajectory consists of finite segments of third order polynomials. The process is illustrated in Figure 4 and can be summarized as the following:

1. Find split points that are contained within more than one bounding boxes (the red circle in Figure 4).
2. Split the trajectory into multiple segments (the dotted line rectangle in Figure 4).
3. Check whether at least one bounding box fully contains each of the split segment through finding the extreme values (see Segment 2).

The cylinder-shaped velocity, acceleration and jerk constraints (see Figure 3), can also be checked through finding the extreme values. Taking the velocity constraint from Equation 4 as an example, its satisfaction can be tested by:

1. Project the trajectory into the global frame $\mathcal{G}$.
2. Calculate the horizontal speed profile $v_{\mathrm{h}} = \sqrt{v_{\mathcal{G}_x}^2 + v_{\mathcal{G}_y}^2}$.
3. Find the extreme values of v_{h} and $v_{\mathcal{G}_z}$.
4. Check whether the extreme values is inside the constraint volume.

For a shorter checking time, it is also possible to only consider the trajectory that crosses at most 2 bounding boxes, thus limiting the amount of segments to be examined.

4 Experiments

To test the proposed algorithm, we perform an real experiment flight using an mini-quadrotor. The task (see Figure 5) involving reaching three targets (A, B and C) in sequence inside a pre-generated flying corridor among multiple obstacles. However, before the vehicle reaches target A, the task is modified to reach targets B and C only. The velocity, acceleration and jerk limited trajectory is generated online using the proposed method. And the video of the flight experiment can be found at https://www.youtube.com/watch?v=zAbCmOHj1EI. From Figure 5, our methods respond to the change of targets immediately and generate a trajectory that leads the vehicle to fly back towards target B. The trajectory is entirely inside the safe corridor throughout the process.

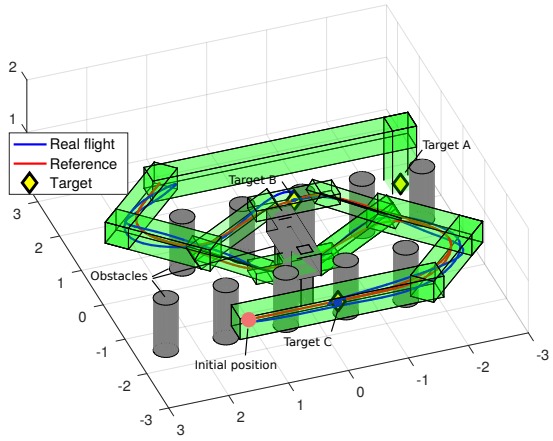

Figure 5: Experiment with multiple targets.

5 Conclusion

In this paper, we present an efficient algorithm to generate velocity, acceleration and jerk limited trajectory with asymmetrical constraints in detail. We then propose an approach to utilize the trajectory generation algorithm to guide a quadrotor to fly smoothly through a pre-generated safe flying corridor without non-necessary stopping. Our approach is efficient and could handle changes in the tasks with real-time responses. Compared to methods which require to solve a constrained quadratic optimization problem, our method is expected to be more suitable for vehicles with limited computational power.

References

[1] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, “Planning dynamically feasible trajectories for quadrotors using safe flight corridors

in 3-d complex environments," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1688–1695, July 2017.

[2] 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.

[3] 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.

[4] T. Krger, "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.

[5] M. Hehn, R. Ritz, and R. D'Andrea, "Performance benchmarking of quadrotor systems using time-optimal control," *Autonomous Robots*, vol. 33, no. 1, pp. 69–88, Aug 2012.

[6] 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.