Robotica: (2021) volume 39, pp. 317–333. © The Author(s), 2020. Published by Cambridge University Press.
doi:10.1017/S0263574720000387

# Efficient and Safe Motion Planning for Quadrotors Based on Unconstrained Quadratic Programming

Yanhui Li* and Chao Liu

*Key Laboratory of Dynamics and Control of Flight Vehicle, Ministry of Education, School of Aerospace Engineering, Beijing Institute of Technology, Beijing, 100081, P.R. China*

(Accepted May 3, 2020. First published online: June 19, 2020)

**SUMMARY**
An autonomous motion planning framework is proposed, consisting of path planning and trajectory generation. Primarily, a spacious preferred probabilistic roadmap algorithm is utilized to search a safe and short path, considering kinematics and threats from obstacles. Subsequently, a minimum-snap and position-clearance polynomial trajectory problem is transformed into an unconstrained quadratic programming and solved in a two-step optimization. Finally, comparisons with other methods based on statistical simulations are implemented. The results show that the proposed method achieves computational efficiency and a safe trajectory.

KEYWORDS: Path planning; Trajectory generation; Probabilistic roadmap; Two-step optimization.

## 1. Introduction

Quadrotor has received much attention in recent years due to its promising applications in dangerous and confined environment, such as surveillance, detection, and delivery. Motion planning module plays an important role in these autonomous missions. Generally speaking, there are two crucial issues to be considered, the front-end path planning and the back-end trajectory generation. Path planning algorithms generate a short and collision-free path from a beginning point to a final point, while trajectory generation algorithms create a feasible trajectory, considering dynamic properties and time information.

Search-based methods are typical representatives of motion planning. Search-based methods discretize the configuration space and convert the path-finding problem to graph searching. Common methods like Anytime A* (ARA*),[1] Jump Point Search (JPS),[2] and hybrid A*[3] have been used for mobile robots in many applications. However, it is impractical for a dynamic system with six degrees of freedom to explore the full state space using incremental search-based techniques. Liu[4] formulates the trajectory generation problem as a minimum-time optimal control-based graph search. A linear model of the quadrotor is established, followed by searching time-optimal motion primitives in a discrete space. The work is extended to enable aggressive flight with attitude constraints in cluttered environments, and a hierarchical refinement process is proposed to accelerate planning in higher dimensions by prior lower-dimensional trajectories.[5] Furthermore, motion uncertainty, constraints on field-of-view, and dynamic environments are taken into consideration.[6] However, the complexity of the search grows exponentially as the space size increases. Also, there are no explicit criterions for the discretization of control input. To achieve better time allocation, Gao[7] proposes a fast marching-based searching method on Euclidean Signed Distance Field (ESDF) map, by building a reasonable velocity field according to the distance of obstacles. Ding[8] proposes a real-time B-spline based kino-dynamic (RBK) search algorithm, which transforms a position-only shortest path search (such as A*

* Corresponding author. E-mail: liyanhui8711@sina.com

https://doi.org/10.1017/S0263574720000387 Published online by Cambridge University Press

and Dijkstra) into an efficient kinodynamic search, by exploring the properties of B-spline parameterization. To cope with the potential sub-optimality and grid discretization in the RBK search, an elastic optimization is used to refine the trajectory. Since search-based method usually depends on discretization of the configuration space or state space, it is not computationally efficient to explore a large amount of states in high-dimensional spaces. One way to cope with the difficulty is to use an efficient heuristic function.

Sampling-based methods are widely used to solve path planning problems. Rapidly exploring random tree (RRT) method is proposed by LaValle.[9] This method allows to search nonconvex high-dimensional spaces, randomly build a space-filling tree, and grow toward the target. However, as the number of samples increases, RRT converges almost surely to a nonoptimal value. In order to make the sampling-based method asymptotic optimal, a rapidly exploring random graph (RRG) and a probabilistic roadmap algorithm (PRM) are developed.[10] RRG and PRM are variants of the RRT algorithm, as they connect new samples not only to the nearest node but also to all other nodes within a nearby region. Gao[11] utilizes RRG algorithm to generate an initial path directly on voxel map. In his method, a K-d tree built by the voxels is used to evaluate the safe volume of a sampling point, and nodes that have enough safe volume are added to the graph. Finally, A* is used to search a minimum distance path on the graph. Based on Gao's work, Lin implements the informed random sampling scheme to improve the efficiency of sampling as well as the optimality of the path.[12] A novel variant of the RRG algorithm, called RRT*, is proposed by Karaman,[13] which inherits the asymptotic optimality of the RRG algorithm while maintaining a tree structure. Richter[14] utilizes the RRT* algorithm to find a collision free path, considering only the kinematics of the vehicle.

The path found by the front end cannot be directly executed by vehicles since it may be discontinuous or contain unnatural swerves. Therefore, the path is further optimized to generate a safe, continuous, and dynamically feasible trajectory, which is considered as executable. There is plenty of research on algorithms and applications of trajectory generation for quadrotors, from bio-inspired method to polynomial optimization method. Zhang and Xie[15–19] have developed tau theory and its variants in a rapid point to point trajectory generation of perching for the quadrotor, not taking obstacles into consideration. As the quadrotor is proved to be a differentially flat system, its trajectory can be described as piecewise polynomials.[20] When collision avoidance is taken into account, more constraints need to be added to guarantee safety. To ensure the trajectory collision-free with more likehood, it is useful to build a Safe Flight Corridor (SFC). The straight-line path is enclosed by the SFC, and a convex optimization problem can be formulated to constrain the trajectory inside the SFC. A mixed integer method (MILP/MIQP) is proposed to ensure a collision-free trajectory.[21–23] While it is not efficient since solving MILP/MIQP takes seconds to minutes. To efficiently generate an obstacle-free trajectory from a straight-line path, geometric constraints have been proposed. Mellinger[20] implements a specific corridor for the straight line, and the generated trajectory is constrained in the corridor. Oleynikova[24] formulates the trajectory optimization problem as a nonlinear optimization over the penalty of the safety and the smoothness and a loco resample method to generate collision-free trajectory in her further work.[25] Furthermore, to maximize obstacle clearance and preserve topological connections within the space, Oleynikova[26] extracts a 3D Generalized Voronoi Diagram from an ESDF map and then constructs a skeleton sparse graph. Path planning over this graph has orders of magnitude speed-up than other methods. Lai[27] uses B-splines to enforce the safety and motion limitations over the entire trajectory. In his another work,[28] safe zones are represented with multiple interconnected bounding boxes which can be arbitrarily oriented, and the jerk limited trajectory generation has been extended to cover the cases with asymmetrical motion limits. Gao[11] uses the point cloud map to directly find a collision-free flight corridor that consists of multiple overlapping free space balls and turns the polynomial trajectory generation problem into a quadratically constrained quadratic programming (QCQP). The trajectory is represented as piecewise Bézier curves by using the Bernstein polynomial basis, and the optimization problem is solved as a convex program.[29] By using Bézier curves, the position and kinodynamics of the trajectory are entirely constrained within the flight corridor. This method treats all the free space in the flight corridor equally, which may result in a trajectory close to obstacles. Therefore, Lin[12] proposes the gradient information in the environments to add a penalty based on the distance to obstacles, thus pushing the optimized trajectory away from obstacles. Liu[30] derives an efficient convex decomposition method to build a SFC from a piecewise linear skeleton, and the SFC provides a set of linear inequality constraints in the quadratic programming (QP). Huh[31] constructs a convex probabilistically safe corridor using

https://doi.org/10.1017/S0263574720000387 Published online by Cambridge University Press

tangent hyperplanes of confidence ellipsoids of GMMs, and then projects the sample goal onto probabilistically safe corridors, which efficiently steers a safe direction and stepsize. The SFC problems are generally solved as a constrained QP problem. To obtain the convex formulation, these methods adopt the time allocation technique that requires the estimation of the time for each segment, and an inaccurate estimation usually leads to an inadequate or infeasible solution. Unconstrained QP is proposed by Richter,[14] which can be solved in closed-form. If the trajectory intersects with obstacles, additional intermediate waypoints are added, and the trajectory is generated again. Both numerical stability and computational efficiency are improved in this way. However, it requires many iterations to generate a collision-free trajectory, especially when it is in a narrow environment.

According to the above literatures, it is sensible to generate a spacious SFC in the process of path planning. Because this can ensure computational efficiency as well as collision-free for the trajectory generation optimization. In this paper, our motion planning framework also consists of two parts, the front-end path planning and the back-end trajectory generation. In path planning, we focus on the strategy of finding a straight-line path with a spacious corridor. Unlike most of the literations,[11,20,30] in which the SFC is constructed after a shortest path generated, we try to construct it when searching the path. Our method is similar to motion primitive methods,[5,7] but avoids uniform-cost grid map and ensures computational efficiency in 3D environment. In the planning process, a sparse graph is generated based on PRM, and each edge is described with a property of the minimum radius from obstacles. When searching a straight-line path with A* algorithm, a more spacious path is searched by utilizing a cost function that trades off between the threats from obstacles and the distance. We will use SP-PRM as an abbreviation of spacious-preferred probabilistic roadmap in the following. In trajectory generation process, a minimum snap and position-clearance polynomial trajectory problem is transformed into an unconstrained QP and solved in closed-form efficiently. If the trajectory collides with obstacles, it can be pulled close to the path due to the control function of position-clearance.

The paper is structured as follows. In Section 2, the SP-PRM algorithm is introduced and implemented to generate a sparse graph, followed by a spacious path. Minimum snap and position-clearance trajectory generation based on polynomial optimization is presented in Section 3. Section 4 shows the related results and comparisons of simulations. Finally, conclusions are given in Section 5.

## 2. Path Planning

### *2.1. Map Representation based on voxblox*

Map representation of the environment is central to the algorithms of path planning and trajectory generation. Our map representation depends on a dense voxel map of Euclidean Signed Distance Field (ESDF), which is built using voxblox.[32] In this map space, each point is stored with its distance to the nearest obstacle. In the scenario of this paper, the ESDF map is built and processed off-line, and our work is to create a sparse topological graph and search a safe and short straight-line path, using the SP-PRM algorithm.

### *2.2. Generation of a sparse graph*

In order to obtain a sparse graph, a sampling-based method is adopted to generate points that connect others, including the beginning point and the goal point. Based on the advantage of the ESDF map, the safe radius of an arbitrary point in the map can be easily obtained. In addition, we can locate the point with the smallest safe radius for an arbitrary edge, using a specific interpolation method. The primary pseudocode of the algorithm is presented in Tables I and II. In Table I, a graph $\mathcal{G}$ is initialized with the beginning point and the goal point. In this graph, a point $p$ consists of two parts, the position $p.c$ and its safe radius $p.r$. The while function attempts to sample $N$ points randomly and find valid connections with their neighbors. During each pass in the loop, a new position $p_m.c$ is generated by the sample function in the first place. According to the ESDF map, the safe radius of the new position, $p_m.r$, which can determine whether it is a valid position, is obtained by the radius function. Once the new position is valid, its $K$ nearest neighbors in graph $\mathcal{G}$ can be find by neighbors function. For each of the neighbors, a check function is used to determine whether it can be connected with the new position. What is worth mentioning, the minimum safe radius of the connection line is obtained by check function at the same time.

https://doi.org/10.1017/S0263574720000387 Published online by Cambridge University Press

Table I. Main loop.

| Sample and connect points for PRM |
|---|
| Requires Voxblox Map $\mathcal{M}$ |
| $\mathcal{G} \leftarrow p_b, \mathcal{G} \leftarrow p_g, i \leftarrow 0$ |
| **while** $i < N$ **do** |
| $p_m.c \leftarrow$ **sample**(), $p_m.r \leftarrow$ **radius**$(p_m.c, \mathcal{M})$ |
| **if** $p_m.r > r_s$ |
| $\mathcal{G} \leftarrow \mathcal{G} \cup p_m$ |
| **for** $p_n \in$ **neighbors**$(p_m, \mathcal{G})$ |
| **if check**$(p_m, p_n, r_{\min})$ |
| **connect**$(p_m, p_n)$ |
| **end if** |
| **end for** |
| **end if** |
| $i \leftarrow i + 1$ |
| **end while** |

Table II. Interpolation process.

| Check whether to generate an edge, and obtain a minimum safe radius |
|---|
| $p_m.r \leftarrow$ **radius**$(p_m.c, \mathcal{M})$, $p_n.r \leftarrow$ **radius**$(p_n.c, \mathcal{M})$ |
| $r_{\min} \leftarrow$ **min**$(p_m.r, p_n.r)$ |
| $p.c \leftarrow p_m.c, r \leftarrow p_m.r$ |
| $d \leftarrow$ **distance**$(p_m.c, p_n.c)$ |
| **while** $r < d$ **do** |
| $p.c \leftarrow \dfrac{(p_n.c \times r + (d - r) \times p.c)}{d}$, $p.r \leftarrow$ **radius**$(p.c, \mathcal{M})$ |
| **if** $p.r \leq r_s$ |
| **return false** |
| **end if** |
| $d \leftarrow d - r, r \leftarrow p.r$ |
| $r_{\min} \leftarrow$ **min**$(r_{\min}, r)$ |
| **end while** |
| **return true** |

The pseudocode of the check function is presented in Table II. In this process, two points are given as the inputs, and a linear interpolation is used to obtained a new position step by step. Specifically speaking, a linear ratio of interpolation is calculated as $\frac{r}{d}$, where $r$ is a safe radius of the start point, and $d$ is a distance between the two points in each cycle. If the new interpolated point has a safe enough radius for the quadrotor, take it as the start point in the next cycle. Otherwise, break this function, which means that the connection is invalid. In addition, $r_{\min}$ equals the minimum safe radius when all cycles are over.

https://doi.org/10.1017/S0263574720000387 Published online by Cambridge University Press

*2.3. A* algorithm through sparse graph*

As presented above, a sparse graph has been created. Since there are multiple feasible paths from the beginning point to the goal point in the graph, a search-based algorithm A* is utilized to find an optimal one, considering a specific objective function. A* is a typical heuristic search algorithm, and it is a crucial issue how to design cost function of this algorithm. As is known to all, a cost function value consists of two parts: an actual cost function and a heuristic function.

$$f(p) = g(p) + h(p) \tag{1}$$

where $p$ is the next point on the path, $g(p)$ is the actual cost of the path from the beginning point to $p$, and $h(p)$ is a heuristic function that estimates the cost of the optimal path from $p$ to the goal. If the heuristic function does not overestimate the actual cost to the goal, A* algorithm is guaranteed to find an optimal-cost path. As the cost of a path is the sum of its edge costs, it is significant to define the edge cost. In general, it is in Euclidean metric.[11] While we make a different definition in this paper. For each edge, whose start vertice and end vertice are $p_b$ and $p_g$, respectively, the cost function $c(e)$ is defined as follows:

$$c(e) = d(e)w(e) + \max(0, z_g - z_b) \tag{2}$$

$$w(e) = w_1(e)w_2(e) \tag{3}$$

where $d$ is the edge length, $w(e)$ is the edge weight, and $z$ is the vertical component of a point. The edge function $w(e)$ contains two parts, $w_1(e)$ and $w_2(e)$. The major advantage of $w_1(e)$ is to ensure a relatively wide straight-line path which tends to keep away from obstacles, while $w_2(e)$ is responsible that the straight-line path can be optimized to an obstacle-free trajectory in a high probability, by constraining the length of each line. As shown in Eqs. (4) and (5), both $w_1(e)$ and $w_2(e)$ depend on the minimum safe radius $r_{\min}(e)$ of the edge, which is obtained in the generation process of the graph.

$$w_1(e) = \begin{cases} \ln\left(\frac{r_u - r_s}{r_{\min}(e) - r_s}\right) + 1, & \text{if } r_s < r_{\min}(e) < r_u \\ 1, & \text{if } r_{\min}(e) \geq r_u \end{cases} \tag{4}$$

$$w_2(e) = a_1 \left(\frac{d(e)}{r_{\min}(e) - r_s} - a_2\right)^2 + 1 \tag{5}$$

where $r_s$ is the minimum radius that the quadrotor can pass through, $r_u$ is the maximum radius that can affect the flight of the quadrotor, and $a_1$ and $a_2$ are constant numbers. The probability of a collision with obstacles increases, as the value of $r_{\min}(e)$ decreases. Therefore, a smaller $r_{\min}(e)$ makes a bigger $w_1(e)$, which lead the algorithm to search a path keeping away from obstacles.

In a real flight, the quadrotor follows a smooth trajectory, not a straight-line path. The former is optimized from the latter, which will be introduced in Section 3. Generally speaking, an optimized trajectory has an overshoot relative to the straight line. The overshoot is probably larger than the secure radius of the line and results in an intersection with obstacles. It is more likely to intersect with obstacles for the trajectory from a longer straight line in a narrow scenario. On the other hand, a shorter straight line in a wide scenario is less efficient. Therefore, to avoid both cases, $w_2(e)$ is proposed to adapt the length of edge to the environment. Intuitive descriptions of the functions are presented in Fig. 1. As a result, the quadrotor has to get through the edge in a lower speed, which adds to the time cost. In the second part of this equation, it takes an additional power cost if there is an increase of height in the edge. The definition of $w(e)$ is defined in equation. Finally, we choose to use the sum of Euclidean distance and Manhattan distance in positive vertical direction as a heuristic function, that is,

$$h(v) = d(p, p_g) + \max(0, z_g - z) \tag{6}$$

where $d$ and max means the Euclidean distance and positive vertical Manhattan distance between two points, respectively. Finally, a simple 2D and a 3D verifications of our method compared with PRM are presented in Fig. 2.

https://doi.org/10.1017/S0263574720000387 Published online by Cambridge University Press

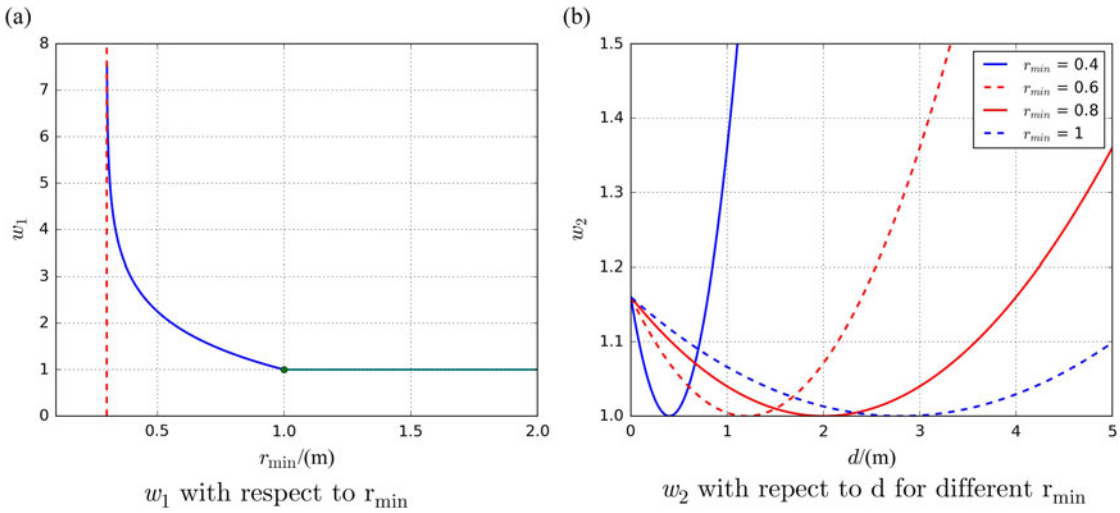



Fig. 1. Descriptions of weights with specified parameters. In (a) $r_u = 1$ and $r_s = 0.3$. In (b) $a_1 = 0.01$ and $a_2 = 4$.

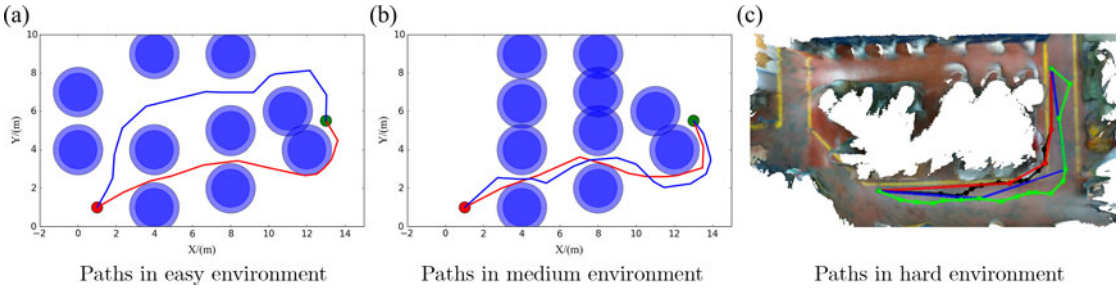



Fig. 2. Paths generated by SP-PRM compared with PRM in different maps. Paths in blue lines are generated by SP-PRM, while those in red lines are generated by PRM. The simple 2D simulations are presented in (a) and (b). (a) With a spacious and a narrow corridor. (b) With a narrow corridor only. Obviously, there is a more spacious corridors to choose in (a) for SP-PRM. As there is only a narrow corridor, the path generated by SP-PRM stays as far away from obstacles as possible. (c) A 3D machine hall map based on voxblox.[32] The original paths may be in zig-zag patterns in a 3D map, green lines and black lines corresponding to SP-PRM and PRM. After random path shortening process,[33] the blue lines are the final path for SP-PRM and the red ones for PRM. Obviously, the blue path has a more safe corridor than the red one.

## 3. Trajectory Generation

In this section, an analytical method is proposed to optimize a minimum-snap as well as position-clearance polynomial trajectory. Before the optimization, the straight-line path has been obtained based on the SP-PRM method mentioned in Section 2, which has a considerable spacious corridor. To ensure computational efficiency and numerical stability, the optimization problem is transformed into an unconstrained QP.

### *3.1. Cost function*

As presented in ref. [20], the quadrotor dynamics with four inputs is differentially flat. Thanks to the differential flatness property of the dynamic model, it is possible to reduce the full state space of the quadrotor to the position and yaw angle and their derivatives. With the advantage of differentiability, polynomial trajectory is really a good choice to be used as a representation of the quadrotor dynamics. We focus on generating a minimum-snap as well as position-clearance polynomial trajectory.

*3.1.1. Cost function based on minimum-snap.* According to ref. [14], the cost function of the trajectory optimization is derived from the square of the snap. They are meaningful since the input

https://doi.org/10.1017/S0263574720000387 Published online by Cambridge University Press

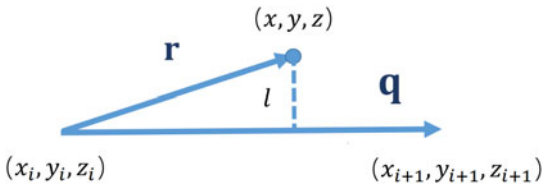



Fig. 3. Distance from a point to a straight line. $\mathbf{q}$ is the vector of a segment line, which equals $(x_{i+1} - x_i, y_{i+1} - y_i, z_{i+1} - z_i)$, and $\mathbf{r}$ is the vector from the start point of the segment line to an arbitrary point of the corresponding trajectory, expressed as $(x - x_i, y - y_i, z - z_i)$.

variables are algebraically related to the snap. By concatenating cost matrices in a block-diagonal matrix, the minimum-snap cost function can be respectively described as follows:

$$J_1 = \int_0^T (P^{(4)}(T))^2 = \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix}^T \begin{bmatrix} \mathbf{Q}_1^{(4)}(T_1) & & \\ & \ddots & \\ & & \mathbf{Q}_M^{(4)}(T_M) \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix} = \mathbf{p}^T \mathbf{Q}^{(4)} \mathbf{p} \tag{7}$$

In this expression, $\mathbf{p}_i = \begin{bmatrix} p_{i1} \dots p_{iN} \end{bmatrix}^T \in \mathbf{R}^{N \times 1}$ is a vector of the N coefficients in the $i$th polynomial segment, and $\mathbf{Q}_i^{(4)}(T_i) \in \mathbf{R}^{N \times N}$ is the corresponding cost matrices with time duration $T_i$. Here we make $N = 10$, so they can be expressed with basic matrices.

$$\mathbf{Q}_i^{(4)}(T_i) = \int_0^{T_i} \mathbf{B}_4 \mathbf{B}_4^T, \quad \mathbf{B}_4 = \left[0, 0, 0, 3!, \frac{4!}{1!}T, \frac{5!}{2!}T^2, \frac{6!}{3!}T^3, \frac{7!}{4!}T^4, \frac{8!}{5!}T^5, \frac{9!}{6!}T^6\right]^T \tag{8}$$

*3.1.2. Cost function based on position-clearance.* As the SP-PRM method is implemented in Section 2, the straight-line path has a considerable safe corridor. Due to this fact, it is safe to keep the trajectory close to the straight-line path. Therefore, the cost function for position-clearance can be described as the distance between the trajectory and the path. In order to simplify things, we just deduce the equation for the $i$th segment. We use a vector-based method to calculate the distance, and the schematic diagram is shown in Fig. 3.

According to the figure, the squared distance from the trajectory point to the line can be described and arranged as Eqs. (9) and (10).

$$l^2 = |\mathbf{r}|^2 - \left(\frac{\mathbf{rq}}{|\mathbf{q}|}\right)^2 = \frac{f_1^2}{f_0} + \frac{f_2^2}{f_0} + \frac{f_3^2}{f_0} = P_1 + P_2 + P_3 \tag{9}$$

$$\begin{cases} f_1 = (y_{i+1} - y_i)x + (x_i - x_{i+1})y + (x_{i+1}y_i - x_i y_{i+1}) \\ f_2 = (z_{i+1} - z_i)x + (x_i - x_{i+1})z + (x_{i+1}z_i - x_i z_{i+1}) \\ f_3 = (z_{i+1} - z_i)y + (y_i - y_{i+1})z + (y_{i+1}z_i - y_i z_{i+1}) \\ f_0 = (x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 + (z_{i+1} - z_i)^2 \end{cases} \tag{10}$$

To solve one of the three dimensions, we assume the other two known. For example, to solve $\mathbf{p}_{x_i}$, $\mathbf{p}_{y_i}$, and $\mathbf{p}_{z_i}$ are already known. For simplification, we take the solution of $\mathbf{p}_x$ as an example. Then $\mathbf{p}_y$ and $\mathbf{p}_z$ are already known, which means that $P_3$ does not contain unknown variables and can be ignored when taking derivatives. Therefore, $P_1$ can be written as a form corresponding to Eq. (11).

$$\begin{cases} P_1 = (\mathbf{s}_i \mathbf{p}_{x_i} + \Delta \mathbf{p}_i)^T \mathbf{Q}_i (\mathbf{s}_i \mathbf{p}_{x_i} + \Delta \mathbf{p}_i) \\ \Delta \mathbf{p}_i = \dfrac{(x_i - x_{i+1})\mathbf{p}_{y_i} + (x_{i+1}y_i - x_i y_{i+1})\mathbf{p}_{o_i}}{\sqrt{f_0}} \\ \mathbf{s}_i = \dfrac{y_{i+1} - y_i}{\sqrt{f_0}} \mathbf{I}^{N \times N} \\ \mathbf{Q}_i = \int_0^{T_i} \mathbf{B}\mathbf{B}^T \end{cases} \tag{11}$$

https://doi.org/10.1017/S0263574720000387 Published online by Cambridge University Press

where $\mathbf{p}_{o_i} = [1, 0, \ldots, 0]^T \in R^{10\times 1}$ and $B = [1, T, T^2, T^3, T^4, T^5, T^6, T^7, T^8, T^9]$. Similarly, $P_2$ can be expressed in the same form. Finally, by using block-diagonal matrices $\mathbf{S}$ and $\mathbf{Q}$, the position-clearance cost function is formulated as:

$$\mathbf{S} = \begin{bmatrix} \mathbf{s}_1 & & \\ & \ddots & \\ & & \mathbf{s}_M \end{bmatrix} \quad \mathbf{Q} = \begin{bmatrix} \mathbf{Q}_1(T_1) & & \\ & \ddots & \\ & & \mathbf{Q}_M(T_M) \end{bmatrix} \tag{12}$$

$$\begin{aligned} J_2 &= (\mathbf{Sp} + \Delta\mathbf{p})^T \mathbf{Q}(\mathbf{Sp} + \Delta\mathbf{p}) \\ &= \mathbf{p}^T\mathbf{S}^T\mathbf{QSp} + \Delta\mathbf{p}^T\mathbf{QSp} + \mathbf{p}^T\mathbf{S}^T\mathbf{Q}\Delta\mathbf{p} + \Delta\mathbf{p}^T\mathbf{Q}\Delta\mathbf{p} \end{aligned} \tag{13}$$

Considering weight parameters $W_1$ and $W_2$ to trade-off smoothness and clearance, the formulation of the complete cost function can be finally written as:

$$J = W_1 J_1 + W_2 J_2 \tag{14}$$

*3.2. Unconstrained quadratic programming*

As there are two endpoints of each segment, fixed or free constraints of position, velocity, acceleration, jerk, and snap are implemented on each endpoint. Therefore, we can deduce the constraints equations for the $i$th segment.

$$\mathbf{A}_i\mathbf{p}_i = \mathbf{c}_i, \mathbf{A}_i = \begin{bmatrix} \mathbf{A}_0 \\ \mathbf{A}_T \end{bmatrix}_i, \mathbf{c}_i = \begin{bmatrix} \mathbf{c}_0 \\ \mathbf{c}_T \end{bmatrix}_i \tag{15}$$

Here, $\mathbf{A}_i$ is a mapping matrix between coefficients and endpoint derivatives, $\mathbf{A}_0$ for the beginning point, and $\mathbf{A}_T$ for the end point of the $i$th segment. As mentioned above, each endpoint has five constraints, no matter fixed or free. Therefore, $\mathbf{A}_i$ is formulated as follows.

$$\mathbf{A}_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1! & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2! & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3! & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4! & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{16}$$

$$\mathbf{A}_{T_i} = \begin{bmatrix} 1 & T_i & T_i^2 & T_i^3 & T_i^4 & T_i^5 & T_i^6 & T_i^7 & T_i^8 & T_i^9 \\ 0 & 1! & \frac{2!}{1!}T_i & \frac{3!}{2!}T_i^2 & \frac{4!}{3!}T_i^3 & \frac{5!}{4!}T_i^4 & \frac{6!}{5!}T_i^5 & \frac{7!}{6!}T_i^6 & \frac{8!}{7!}T_i^7 & \frac{9!}{8!}T_i^8 \\ 0 & 0 & 2! & \frac{3!}{1!}T_i & \frac{4!}{2!}T_i^2 & \frac{5!}{3!}T_i^3 & \frac{6!}{4!}T_i^4 & \frac{7!}{5!}T_i^5 & \frac{8!}{6!}T_i^6 & \frac{9!}{7!}T_i^7 \\ 0 & 0 & 0 & 3! & \frac{4!}{1!}T_i & \frac{5!}{2!}T_i^2 & \frac{6!}{3!}T_i^3 & \frac{7!}{4!}T_i^4 & \frac{8!}{5!}T_i^5 & \frac{9!}{6!}T_i^6 \\ 0 & 0 & 0 & 0 & 4! & \frac{5!}{1!}T_i & \frac{6!}{2!}T_i^2 & \frac{7!}{3!}T_i^3 & \frac{8!}{4!}T_i^4 & \frac{9!}{5!}T_i^5 \end{bmatrix} \tag{17}$$

$\mathbf{c}_i \in \mathbf{R}^{10\times 1}$ is a vector containing 10 constraints, 5 for each point of the $i$th segment. In order to keep continuity, constraints at the end of the $i$th segment must match those at the beginning of the $(i+1)$th segment.

$$\mathbf{c}_{T_i} = \mathbf{c}_{0_{i+1}} \tag{18}$$

https://doi.org/10.1017/S0263574720000387 Published online by Cambridge University Press

Therefore, the whole constraints can be formulated by concatenating mapping matrices in a block-diagonal matrix in Eq. (19).

$$\begin{bmatrix} \mathbf{A}_1 & & \\ & \ddots & \\ & & \mathbf{A}_M \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix} = \begin{bmatrix} \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_M \end{bmatrix} \tag{19}$$

Now, by constituting (7), (13), and (19) into (14), we can get a new quadratic cost function. If we assume $T_i$ as known parameters, then the cost function is only determined by the constraints of all endpoints.

$$\begin{aligned} J = {} & W_2 \left( \mathbf{c}^T \mathbf{A}^{-T} \mathbf{S}^T \mathbf{Q} \mathbf{S} \mathbf{A}^{-1} \mathbf{c} + \Delta \mathbf{p}^T \mathbf{Q} \mathbf{S} \mathbf{A}^{-1} \mathbf{c} + \mathbf{c}^T \mathbf{A}^{-T} \mathbf{S}^T \mathbf{Q} \Delta \mathbf{p} + \Delta \mathbf{p}^T Q \Delta \mathbf{p} \right) \\ & + W_1 \mathbf{c}^T \mathbf{A}^{-T} \mathbf{Q}^{(4)} \mathbf{A}^{-1} \mathbf{c} \end{aligned} \tag{20}$$

Generally speaking, five constraints are all fixed for the start point and the goal point, while the position is fixed and its four derivatives are free for each intermediate point. With this in mind, we can reorder these variables, by putting fixed constraints together as $\mathbf{c}_F$ and those free as $\mathbf{c}_P$. Here, $\mathbf{c}_F$ is predefined known before the trajectory optimization, and $\mathbf{c}_P$ are the variables to be optimized. $\mathbf{c}^T$ can be reordered as:

$$\mathbf{c}^T = \begin{bmatrix} \mathbf{c}_F \\ \mathbf{c}_P \end{bmatrix}^T \mathbf{O} \tag{21}$$

In this expression, $\mathbf{O}$ is a permutation matrix with ones and zeros to accomplish this reordering. Now constitute (21) into (20) and group the new augmented matrices into a single matrix.

$$J = W_2 \left( \begin{bmatrix} \mathbf{c}_F \\ \mathbf{c}_P \end{bmatrix}^T \mathbf{H} \begin{bmatrix} \mathbf{c}_F \\ \mathbf{c}_P \end{bmatrix} + \mathbf{E} \begin{bmatrix} \mathbf{c}_F \\ \mathbf{c}_P \end{bmatrix} + \begin{bmatrix} \mathbf{c}_F \\ \mathbf{c}_P \end{bmatrix}^T \mathbf{E}^T + \mathbf{B} \right) + W_1 \begin{bmatrix} \mathbf{c}_F \\ \mathbf{c}_P \end{bmatrix}^T \mathbf{H}^{(4)} \begin{bmatrix} \mathbf{c}_F \\ \mathbf{c}_P \end{bmatrix} \tag{22}$$

In this expression, $\mathbf{H} = \mathbf{O}\mathbf{A}^{-T}\mathbf{S}^T\mathbf{Q}\mathbf{S}\mathbf{A}^{-1}\mathbf{O}^T$, $\mathbf{E} = \Delta\mathbf{p}^T\mathbf{Q}\mathbf{S}\mathbf{A}^{-1}\mathbf{O}^T$, $\mathbf{B} = \Delta\mathbf{p}^T\mathbf{Q}\Delta\mathbf{p}$, $\mathbf{H}^{(4)} = \mathbf{O}\mathbf{A}^{-T}\mathbf{Q}^{(4)}\mathbf{A}^{-1}\mathbf{O}^T$. Furthermore, (22) can be written in a partitioned form according to indices of the fixed and the free derivatives.

$$\begin{aligned} J = W_2 \Bigg( & \begin{bmatrix} \mathbf{c}_F \\ \mathbf{c}_P \end{bmatrix}^T \begin{bmatrix} \mathbf{H}_{FF} & \mathbf{H}_{FP} \\ \mathbf{H}_{PF} & \mathbf{H}_{PP} \end{bmatrix} \begin{bmatrix} \mathbf{c}_F \\ \mathbf{c}_P \end{bmatrix} + \begin{bmatrix} \mathbf{E}_F \\ \mathbf{E}_P \end{bmatrix}^T \begin{bmatrix} \mathbf{c}_F \\ \mathbf{c}_P \end{bmatrix} + \\ & \begin{bmatrix} \mathbf{c}_F \\ \mathbf{c}_P \end{bmatrix}^T \begin{bmatrix} \mathbf{E}_F \\ \mathbf{E}_P \end{bmatrix} + \mathbf{B} \Bigg) + W_1 \begin{bmatrix} \mathbf{c}_F \\ \mathbf{c}_P \end{bmatrix}^T \begin{bmatrix} \mathbf{H}_{FF}^{(4)} & \mathbf{H}_{FP}^{(4)} \\ \mathbf{H}_{PF}^{(4)} & \mathbf{H}_{PP}^{(4)} \end{bmatrix} \begin{bmatrix} \mathbf{c}_F \\ \mathbf{c}_P \end{bmatrix} \end{aligned} \tag{23}$$

Finally, we deduce the Jacobian of J with respect to the free variables $\mathbf{c}_p$ and assign it to zero, then the solution can be found.

$$\mathbf{c}_P^* = -\left( W_2 \mathbf{H}_{PP} + W_1 \mathbf{H}_{PP}^{(4)} \right)^{-1} \left[ \left( W_2 \mathbf{H}_{PF} + W_1 \mathbf{H}_{PF}^{(4)} \right) \mathbf{c}_F + W_2 \mathbf{E}_P \right] \tag{24}$$

Before the solution, there are two problems to be considered. The first is time allocation. According to (8) and (16), $A_i$ and $Q_i$ are matrix consisting of $T_i$ in each segment. Therefore, we need to allocate time for each segment in advance before optimization. A naive time allocation method is to map the path into time domain using trapezoid velocity profile, but may result in large velocity and acceleration out of threshold. Inspired by Richter,[14] we also map the path into time domain using sigmoid velocity profile. The approach is described in Eq. (25).

$$t = \frac{2d}{v_{\max}} \left( 1 + k \frac{v_{\max}}{a_{\max}} e^{-\frac{2d}{v_{\max}}} \right) \tag{25}$$

https://doi.org/10.1017/S0263574720000387 Published online by Cambridge University Press

$k$ is a magic number decided by the user. A large $k$ leads to conservative dynamics and a small one corresponds to aggressiveness. The second problem is dynamic constraints, while it is hard to optimize if we formulate them as nonlinear inequalities. Therefore, we do not consider the constraints when solving the QP. After the trajectory is obtained, an time scaling method[30] is implemented to ensure the dynamic constraints, keeping the geometry of the trajectory no change.

### *3.3. Solution strategy*

*3.3.1. Two-step optimization framework.* According to Mellinger[20] and Richter,[14] the three dimensions are decoupled in both the cost function and the constraints so the problem can be separated into three optimization problems. However, it is different in our research because the partial cost function $J_2$ is coupled in three dimensions. To solve the optimization problem in our research, we propose a two-step optimization strategy as follows:

1. Optimize the problem with the cost function $J_1$ only, which is the same as ref. [14]. After the optimization, we can obtain an initial solution to each dimension. If the initial trajectory is collision free, it is taken as the final trajectory and the optimization stops. Otherwise, step 2 will be implemented.
2. Reoptimize the problem based on the total cost function $J_1$ and $J_2$. As the initial trajectory is obtained in the former step, the coefficients of three dimensions, $p_x$, $p_y$, and $p_z$ are already known. Therefore, a new $p_x$ is solved in the second-step optimization, coupled with known $p_y$ and $p_z$. Then this dimension is updated after the solution. Similarly, repeat the process for the other two dimensions. Finally, the second-step optimization stops until all the three dimensions are updated.

The first-step optimization is only based on the minimum-snap cost function, and its solution is taken as the initial trajectory. After the first-step optimization, the initial trajectory is probably away from the searched path and collides with obstacles. If this situation happens, the second-step optimization can quickly pull it close to the path and regenerates a much safer trajectory. More details will be demonstrated in the following Section 3.3.2.

*3.3.2. Cost nondimensionalization and iteration.* Note that the cost function consists of two parts according to (14), it is a critical problem how to arrange the weight for each part. Generally speaking, the value of $J_2$ is much larger than that of $J_1$ after the first-step optimization. There are two reasons: the first and most decisive one is that we just take minimizing $J1$ into consideration in the first-step optimization; the second one depends on the maximum velocity and acceleration of the quadrotor and the geometric shape of the path. Due to the above reasons, the difference between $J_1$ and $J_2$ depends much on different situations, which make it not easy to decide the values of $W_1$ and $W_2$. To obtain an analytical solution, we make the cost function dimensionless. Therefore, $W_1$ and $W_2$ can be arranged as follows:

$$\begin{cases} W_1 = \dfrac{1}{\hat{J}_1} \\ W_2 = \dfrac{1}{\hat{J}_2}\eta \end{cases} \qquad \eta \in [0, \infty) \tag{26}$$

where $\hat{J}_1$ and $\hat{J}_2$ are the respective snap and position cost values in the current solution, and $\eta$ is an factor to determine the approaching rate between $\hat{J}_1$ and $\hat{J}_2$. Readers should distinguish that $J_1$ and $J_2$ are cost functions for the next solution, described as matrices with unknown coefficients, while $\hat{J}_1$ and $\hat{J}_2$ are cost values under the current solution. According to (26), we just give an appropriate value of $\eta$ to solve the second-step optimization with different requirements. To demonstrate how the different values of $\eta$ work, we implement a simple 2D simulation in a corridor-like environment. In the simulation, we choose $v_{\max} = 2$, $a_{\max} = 2$, $\eta = (0, 0.5, 1, 5, 10, 50, 100)$. The optimization results are shown in the following Fig. 4(a) and Table III.

https://doi.org/10.1017/S0263574720000387 Published online by Cambridge University Press

Table III. Optimized parameters with different values of $\eta$.

| $\eta$ | **0** | **0.5** | **1** | **5** | **10** | **50** | **100** |
|---|---|---|---|---|---|---|---|
| $\hat{J}_1$ | 0.73 | 0.80 | 0.88 | 1.20 | 1.41 | 2.26 | 2.78 |
| $\hat{J}_2$ | 56.91 | 27.25 | 18.17 | 6.42 | 4.19 | 1.20 | 0.60 |
| $v$(m/s) | 2.00 | 2.00 | 2.00 | 2.00 | 2.00 | 2.00 | 2.00 |
| $a$(m/s$^2$) | 0.81 | 0.87 | 0.92 | 0.93 | 0.91 | 0.89 | 0.89 |
| $t$(s) | 29.32 | 27.66 | 27.12 | 27.54 | 28.10 | 29.11 | 29.38 |

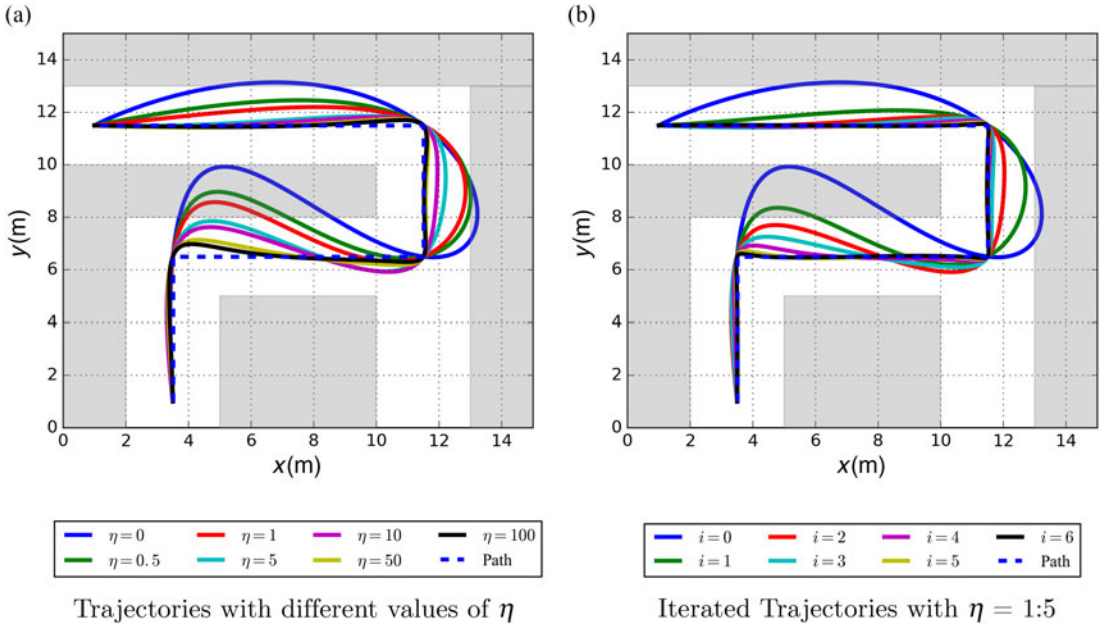



Fig. 4. The dashed line is the straight-line path. As the path has a drastical change in directions, there is a large overshoot in the first-step optimized trajectory which is the blue curve with $\eta = 0$. (a) It is obvious that the second-step optimization can pull the trajectory close to the path, and the value of $\eta$ can determine the amplitude of the change. As can be seen in the figure, the trajectory becomes collision-free when $\eta = 5$, 10, 50, and 100. (b) $\eta = 1.5$ is chosen for iterations. The trajectory converges to the path in exponential rate and its amplitude approximately becomes to a half after each iteration. It is obvious that the trajectory becomes collision-free after two iterations and almost coincides with the line after five iterations.

From the results of the table, We can summarize some qualitative conclusions:

1. After the optimization with $\eta$, $\hat{J}_2$ is approximately changed to $\left(\frac{\hat{J}_2}{1+2x}, \frac{2\hat{J}_2}{1+2x}\right)$. Ten is kind of the boundary in this simulation, while this value differs in different situations. However, we do not need to know the specific value.
2. The final trajectory is scaled until either $v$ or $a$ meets its constraint according to.[30] Inspite of the same allocated time, the final scaled time is different after the optimizations with different $\eta$. There seems to exist a minimum value of the scaled time, and it occurs when the optimized value of $J$ is equals to 2.
3. When the value of $\hat{J}_2$ has a $n$-times change, the amplitude of the overshoot has an approximately $\sqrt{n}$-times change. This is because $\hat{J}_2$ is an integral of the squared distance.

These conclusions are instructive for us to choose the value of $\eta$ and implement the optimization. For example, if we want to approximately change the overshoot to a half, we can make $\eta$ equal to 1.5. Based on this principle, we can accomplish the second-step optimization in an iterative process. The pseudocode is shown in Table IV.

https://doi.org/10.1017/S0263574720000387 Published online by Cambridge University Press

Table IV. Optimization process.

**OptForSmooth**()
**for** i ← 0 to n:
  **if CollisionFree**():
    **break**
  **else**:
    $\hat{J}_1 \leftarrow$ **Compute**$J_1()$, $\hat{J}_2 \leftarrow$ **Compute**$J_2()$
    $W_1 \leftarrow \frac{1}{\hat{J}_1}, W_2 \leftarrow \frac{\eta}{\hat{J}_2}$
    **OptForSmoothAndPosition**()
  **end if**
  $i \leftarrow i + 1$
**end for**

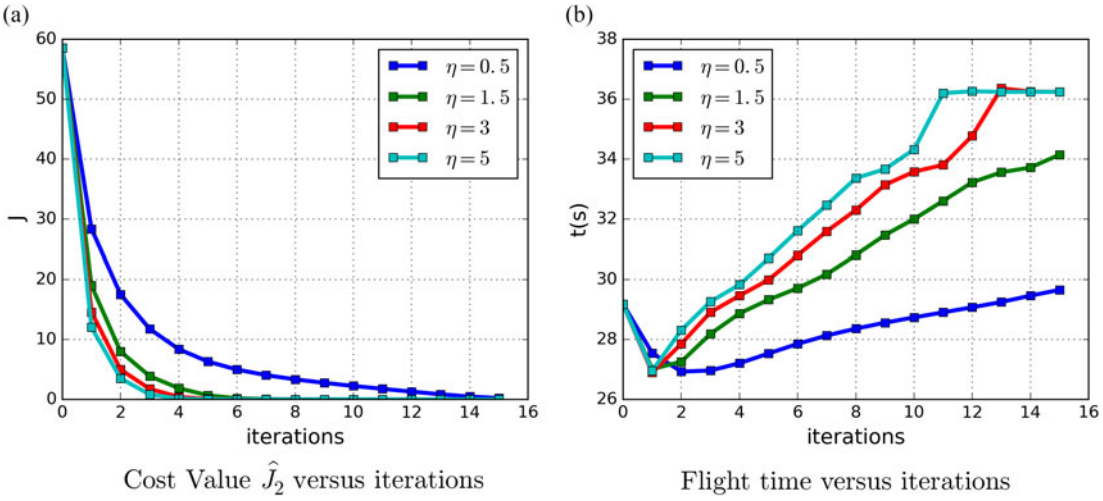



Fig. 5. (a) $\hat{J}_2$ changes in an exponential rate and it converges in several iterations. (b) There is a minimum value for the flight time, and we can get an approximate one for it after one or two iterations, depending on $\eta$. When the trajectory converges to the path, the flight time also converge to that of the velocity ramp method.[26]

OptForSmooth function is used as the first-step optimization only with the cost function $J_1$. Then a small and appropriate value of $\eta$ is chosen, and the second-step optimization is implemented iteratively. In each loop, it stops if the trajectory is collision free. Otherwise, values of $\hat{J}_1$ and $\hat{J}_2$ under the current solution are computed, followed by the update of $W_1$ and $W_2$ according to (26). Finally, the trajectory is optimized by OptSmoothAndPosition function, taking both $J_1$ and $J_2$ functions into consideration. In addition, the loops should stop when satisfying a number of times $n$. In this process, the value of $\hat{J}_2$ converges in an exponential rate as shown in Fig. 4(b). In addition, the cost value $\hat{J}_2$ and the flight time $t$ over iterations are also presented in Fig. 5.

*3.3.3. Control for each segment.* As we have known in the above section, $\eta$ has a direct impact on the change of $\hat{J}_2$. According to Fig. 4(a), each segment becomes collision-free with different values of $\eta$ (0 for the first segment, 5 for the second, 1 for the third, and 0.5 for the last). It is more reasonable to choose different values of $\eta$ for each segment. Therefore, we formulate a vector for $\eta$, which is $\eta = [0, 5, 1, 0.5]$. The optimized trajectory is shown in Fig. 6, compared with the corresponding trajectory with single value of $\eta$. It is another key point how to choose an appropriate vector of $\eta$. As we have obtained a secure radius $r_{\min}$ for each segment in Section 2, it is instructive for us to decide

https://doi.org/10.1017/S0263574720000387 Published online by Cambridge University Press

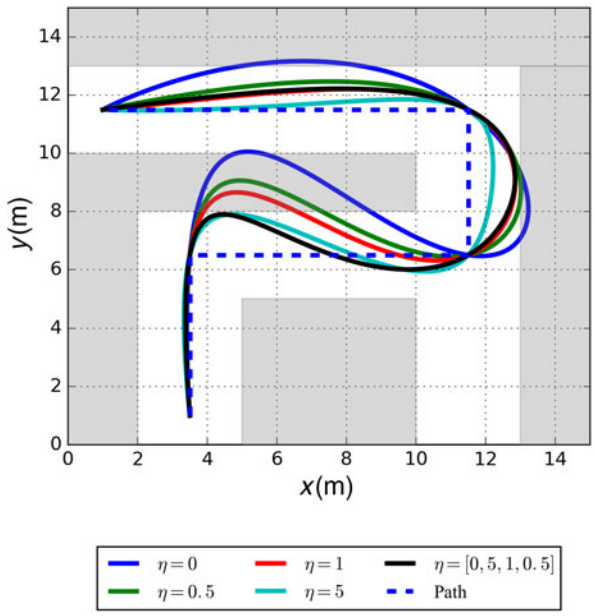



Fig. 6. Optimized trajectory for each segment. The black curve is the optimized trajectory with $\eta = [0, 5, 1, 0.5]$. It approximately coincides with the blue curve ($\eta = 0$) in the first segment, the cyan curve ($\eta = 5$) in the second segment, the red curve ($\eta = 1$) in the third segment, and the green curve ($\eta = 0.5$) in the fourth segment. An appropriate vector of $\eta$ plays an effective and flexible role in the optimization process.

a value for the corresponding segment. Suppose that $r_i$ is the secure radius, and $R_i$ is the maximum distance from the trajectory to the path in the $i$th segment. We will choose a value of $\eta_i$ for each segment, as an approximate formula.

$$\eta = \frac{R_i(R_i - r_i)}{r_i^2} \tag{27}$$

In this equation, we just assume $\hat{J}_2$ is proportional to $r^2$ as mentioned above. Although it is an empirical formula lack of proof, we do not need to be too precise for the iterated second-step optimization. Analytical researches will be carried out in the future work.

## 4. Results and Simulations

As shown in the above sections, the proposed method is validated in some 2D simulations and proved to be significant. Without loss of generality, we will evaluate it on real 3D datasets and compare with other methods. Thanks to researchers of ETHZ lab for their contributions to the open-source code,[34] we can fastly integrate our proposed method in it and compare with other methods. The simulation is done on a laptop equipped with a 2.6 GHz Intel Core i7-6700HQ CPU and 8 GB RAM.

The evaluation benchmark is from the work of Oleynikova.[26] Three kinds of datasets, named "Shed," "Rubble," and "Machine-Hall," are available for simulations, and we choose the "Machine-Hall" dataset in our research. Based on the complete global map, we compare different global planning strategies and evaluate their performances. It is noted that a global planning strategy consists of a front-end path searching method and a back-end trajectory optimization method, so four methods are chosen for the front-end in Table V and 3 for the back-end in Table VI, finally combining 12 kinds of global planning methods.

Before the simulations, we should limit the number of added waypoints in the splitting approaches. Because more additional points require more times of optimizations and increase the computational complexity of the QP problem. Furthermore, more waypoints seem to result in more flight time of the final scaled trajectory. Therefore, we choose five additional waypoints at most in our simulations. In addition, we also limit the number of iterations to five in the two-step optimization method.

In order to guarantee statistical property, we run 100 simulations for each global planning strategy. In each simulation, the start point and end point are randomly selected with a minimum of 2 m apart.

https://doi.org/10.1017/S0263574720000387 Published online by Cambridge University Press

Table V. The front-end path planning methods.

| Front-end method | Summary information |
|---|---|
| *RRT** | Probabilistically optimal planner, run for 2.0 s |
| Skeleton | A sparse topological planner, to compare with our method[26] |
| PRM | Probabilistic roadmap, to compare with our method. Preplanning for 2.0 s and real-time planning for 0.1 s |
| SP-PRM | Our proposed method, the same setup with PRM |

Table VI. The back-end trajectory optimization methods.

| Back-end method | Summary information |
|---|---|
| Loco | Local trajectory optimization, with visibility waypoint resampling[26] |
| Polynomial | Minimum-snap trajectory optimization with unconstrained quadratic programming and a splitting approach to ensure collision-free[14] |
| Two-step opt | Our proposed method, minimum-snap and position-clearance trajectory optimization, with iterations for collision-free |

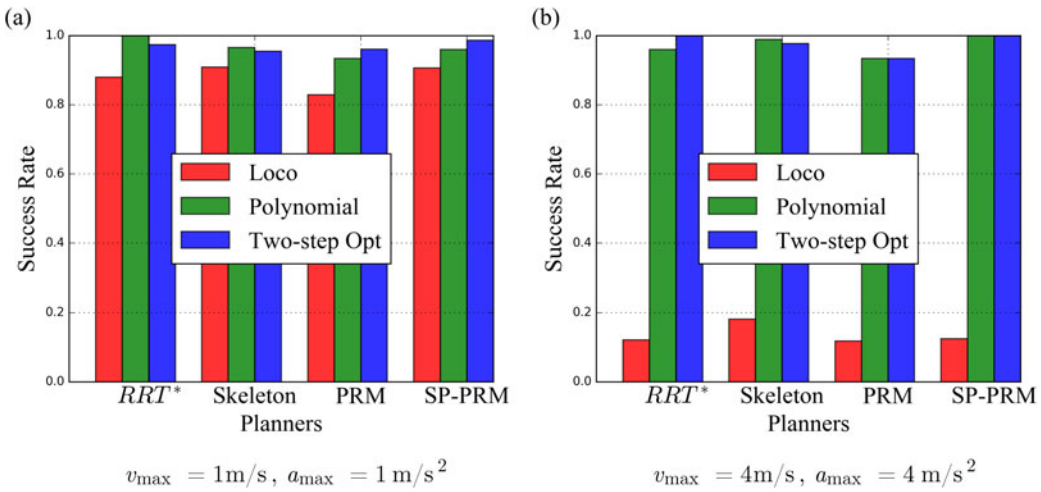



Fig. 7. Success rate of 12 global planning strategies on Machine Hall dataset. (a) With a conservative initial condition, our two-step optimization method is able to give smooth dynamically feasible trajectories as well as the "Polynomial" method, a little better than the "Loco" method. In addition, paths generated by the SP-PRM planner provide a better condition for the two-step optimization, resulting in a higher success rate. (b) With an aggressive initial condition, our two-step method and the "Polynomial" method perform well as in (a), while the "Loco" method becomes out of use.

In addition, the quadrotor is regarded as a sphere with a radius of 0.5 m, not considering attitude angles. The corresponding success rate for each planning method is shown in Fig. 7.

In addition to the success rate, the run time and the flight time are two significant indicators to evaluate the three smoothing methods. In the following, we simulate 100 times for a specified flight task. In each simulation, the start point and the end point are fixed, and the straight-line paths are generated by PRM and SP-PRM, followed by the three different smoothing methods. Corresponding paths and trajectories of a typical simulation in the "Machine Hall" dataset are presented in Fig. 8. In addition, the average of run time and flight time is shown in Fig. 9.

As we can see in the figure, the two-step optimization method runs a little faster than the polynomial method and almost an order of magnitude faster than the loco method. As to the flight time, the loco method performs the best, and the two-step optimization is slightly better than the polynomial method. It is probably because the trajectory generated by the loco method is further away from obstacles, which can be seen in Fig. 8. In summary, the proposed method can generate a fast and safe trajectory with an efficient computation.

https://doi.org/10.1017/S0263574720000387 Published online by Cambridge University Press

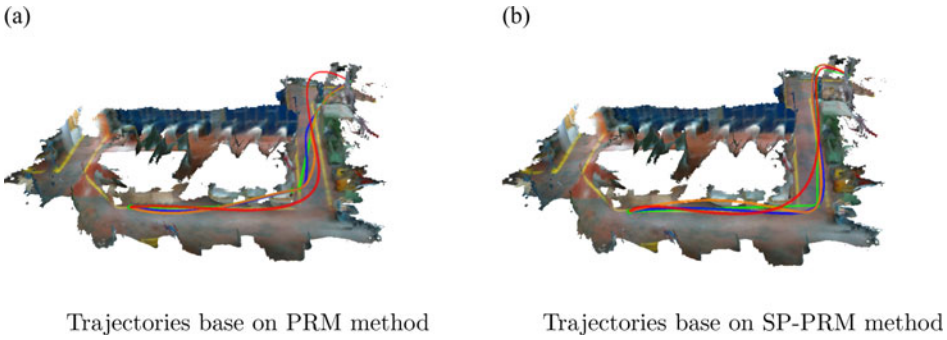



Fig. 8. Optimized trajectories in the "Machine Hall" dataset. The green line is the path, and the red, orange, and blue curves are trajectories optimized by loco, polynomial, and two-step methods, respectively. (a) The path generated by PRM is close to the obstacles, so the optimized trajectories are more likely to collide with them. (b) The path generated by SP-PRM has a considerable safe corridor and the trajectories becomes safer and easier to avoid collisions after optimization. It is also noted that the trajectories optimized by the "loco" method have no significant difference between (a) and (b). That is because this method resample new waypoints and tends to keep away from obstacles, which has the same purpose as SP-PRM.

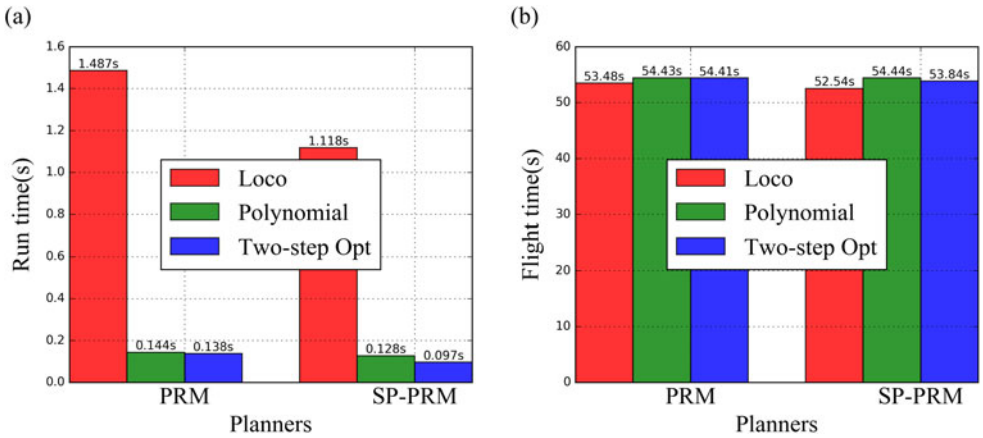



Fig. 9. Performance parameters for different methods. As can be seen, our proposed method in blue bars makes a good showing. (a) Average run time for different methods. (b) Average flight time for different methods.

## 5. Conclusion

We propose a framework of motion planning for the quadrotor in a known environment. In this framework, path planning and trajectory generation are discussed as a front-end and a back-end, respectively. A safe straight-line path is created based on a SP-PRM algorithm, followed by a minimum-snap and position-clearance trajectory generation which is solved by an unconstrained QP. Due to the penalty of position-clearance, the trajectory can be quickly pulled close to the path, ensuring collision-free. Comparisons based on statistical simulations are designed to demonstrate the validity of our method. SP-PRM prefers to search a path with spacious corridor, but not the most ideal one due to the short cutting process. Finally, the minimum-snap and position-clearance optimization makes a significant contribution to achieving computational efficiency and safe trajectory.

In the future, we will extend the work to a more appropriate map like ref. [35] for the best SFC. We also plan to do more analysis how to adaptively trade-off the weights between minimum-snap and position-clearance costs. Furthermore, we aim to challenge this method in extreme situations such as complex and dynamic environments and finally implement the whole framework in a real flight of the quadrotor.

## Acknowledgments

The research leading to these results has received funding from the National Natural Science Foundation of China (program theme: Dynamic modelling and intrinsic control of multibody aircraft considering unilateral constraint; program NO.: 11572036).

https://doi.org/10.1017/S0263574720000387 Published online by Cambridge University Press

## References


1. M. Likhachev, G. Gordon and S. Thrun, "ARA*: Anytime A* with Provable Bounds on Sub-Optimality," *Advances in Neural Information Processing Systems* (2004) pp. 767–774
2. D. Harabor and A. Grastien, "Online Graph Pruning for Path Finding on Grid Maps," *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence* (2011) pp. 1114–1119.
3. D. Dolgov, S. Thrun, M. Montemerlo and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments. *Int. J. Robot. Res.* **29**(5), 485–501 (2010).
4. S. Liu, N. Atanasov, K. Mohta and V. Kumar, "Search-based Motion Planning for Quadrotors using Linear Quadratic Minimum Time Control," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2017) pp. 2872–2879.
5. S. Liu, K. Mohta, N. Atanasov and V. Kumar, "Search-based motion planning for aggressive flight in SE(3)," *IEEE Robot. Autom. Mag.* **3**(3), 2439–2446 (2017).
6. S. Liu, K. Mohta, N. Atanasov and V. Kumar, "Towards Search-based Motion Planning for Micro Aerial Vehicles," arXiv:1810.03071 (2018).
7. F. Gao, W. Wu, Y. Lin and S. Shen, "Online Safe Trajectory Generation for Quadrotors Using Fast Marching Method and Bernstein Basis Polynomial," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (2018) pp. 344–351.
8. W. Ding, W. Gao, K. Wang and S. Shen, "Trajectory Replanning for Quadrotors Using Kinodynamic Search and Elastic Optimization," arXiv:1903.01139 (2019).
9. S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Technical Report (1998) pp. 1–4.
   S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," (1998).
10. S. Karaman and E. Frazzoli, "Sampling-based Motion Planning with Deterministic u-Calculus Specifications," *Proceedings of the IEEE Conference on Decision and Control (CDC)* (2009) pp. 2222–2229.
11. F. Gao and S. Shen, "Online Quadrotor Trajectory Generation and Autonomous Navigation on Point Clouds," *Proceedings of the IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)* (2016) pp. 139–146.
12. Y. Lin, F. Gao, T. Qin, W. Gao, T. Liu, W. Wu, Z. Yang, and S. Shen, "Autonomous aerial navigation using monocular visual-inertial fusion," *J. Field. Robot.* **35**(1), 23–51 (2017).
13. S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.* **30**(7), 846–894 (2011).
14. C. Richter, A. Bry and N. Roy, "Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments," *Proceedings of the International Symposium on Robotics Research (ISRR)* (2016) pp. 649–666.
15. X. Pu, M. Ou and Z. Zhen, "A Bio-inspired Approach for UAV Landing and Perching," *Proceedings of AIAA Guidance, Navigation, and Control* (2013).
16. Z. Zhen, X. Pu and M. Ou, "Bio-inspired Trajectory Generation for Uav Perching," *Proceedings of the International Conference on Advanced Intelligent Mechatronics* (2013) pp. 997–1002.
17. Z. Zhen, S. Zhang, X. Pu and M. Ou, "Bioinspired 4D trajectory generation for a UAS rapid point-to-point movement," *J. Bionic. Eng.* **11**(1), 72–81 (2014).
18. Z. Yang, F. Zhou and L. Ping, "A Bio-inspired Collision-free 4D Trajectory Generation Method for Unmanned Aerial Vehicles Based on Tau Theory," *Proceedings of the Control Conference* (2015) pp. 6961–6968.
19. Z. Yang, Z. Fang and P. Li, "Decentralized 4D trajectory generation for uavs based on improved intrinsic tau guidance strategy," *Int. J. Adv. Robot. Syst.* **13**(3), 1–13 (2016).
20. D. Mellinger and V. Kumar, "Minimum Snap Trajectory Generation and Control for Quadrotors," *Proceedings of the IEEE International Conference on Robotics and Automation* (2011) pp. 2520–2525.
21. D. Mellinger, A. Kushleyev and V. Kumar, "Mixed-integer Quadratic Program Trajectory Generation for Heterogeneous Quadrotor Teams," *Proceedings of the IEEE International Conference on Robotics and Automation* (2012) pp. 477–483.
22. S. Tang and V. Kumar, "Mixed Integer Quadratic Program Trajectory Generation for a Quadrotor with a Cable-suspended Payload," *Proceedings of the International Conference on Robotics and Automation* (2015) pp. 2216–2222.
23. R. Deits and R. Tedrake, "Efficient Mixed-integer Planning for Uavs in Cluttered Environments," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (2015) pp. 42–49.
24. H. Oleynikova, M. Burri, Z. Taylor, J. Nieto and E. Galceran, "Continuous-Time Trajectory Optimization for Online UAV Replanning," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots Systems* (2016) pp. 5332–5339.
25. H. Oleynikova, Z. Taylor, A. Millane, R. Siegwart and J. Nieto, "A Complete System for Vision-Based Micro-Aerial Vehicle Mapping, Planning, and Flight in Cluttered Environments," arXiv:1812.03892 (2018).
26. H. Oleynikova, Z. Taylor, R. Siegwart and J. Nieto, "Sparse 3D Topological Graphs for Micro-Aerial Vehicle Planning," arXiv:1803.04345 (2018).
27. S. Lai, M. Lan, B. M. Chen, "Optimal Constrained Trajectory Generation for Quadrotors through Smoothing Splines," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (2018) pp. 4743–4750.


https://doi.org/10.1017/S0263574720000387 Published online by Cambridge University Press

28. F. Gao, W. Wu, W. Gao and S. Shen, "Flying on point clouds: Online trajectory generation and autonomous navigation for quadrotors in cluttered environments," *J. Field. Robot*. **36**(4), 710–733 (2019).
S. Lai, M. Lan, Y. Li.-x and B. Chen, "Safe navigation of quadrotors with jerk limited trajectory," *Front. Inf. Tech. Electron. Eng.* **20**(1), 107–119 (2019).
29. F. Gao, W. Wu, W. Gao and S. Shen, "'Flying on point clouds: Online trajectory generation and autonomous navigation for quadrotors in cluttered environments," *J. Field. Robot.* **36**(4), 710–733 (2019).
30. S. Liu, M. Watterson, K. Mohta, K. Sun, C. J. T. S. Bhattacharya and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments," *IEEE Robot. Autom. Mag.* **2**(3), 1688–1695 (2017).
31. J. Huh, O. Arslan and D. Lee, "Probabilistically Safe Corridors to Guide Sampling-Based Motion Planning," arXiv:1901.00101 (2019).
32. H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart and J. Nieto, "Voxblox: Incremental 3D Euclidean Signed Distance Fields for On-board Mav Planning," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2017) pp. 1366–1373.
33. J. Mainprice, E. Sisbot, L. Jaillet and J. Cortes, "Planning Human-Aware Motions Using a Sampling-Based Costmap Planner," *Proceedings of the IEEE International Conference on Robotics and Automation* (2011) pp. 5012–5017.
34. H. Oleynikova, https://github.com/ethz-asl/mav_voxblox_planning.
35. R. Geraerts, "Planning Short Paths with Clearance Using Explicit Corridors," *Proceedings of the IEEE International Conference on Robotics and Automation* (2010) pp. 1997–2004.

https://doi.org/10.1017/S0263574720000387 Published online by Cambridge University Press