Visualizing Planning Dynamically Feasible Trajectories for Quadrotors Using Safe Flight Corridors in 3-D complex Environments

Shengjie Zhang

October 2020

1 Abstract

There is extensive literature on using convex optimization to derive piece-wise polynomial trajectories for controlling differential flat systems with applications to three-dimensional flight for Micro Aerial Vehicles.In this work, I use matlab to realize algorithms in paper[1] ”Planning Dynamically Feasible Trajectories for Quadrotors Using Safe Flight Corridors in 3-D Complex Environments.” [1] has three parts, the first part is path planning using Jump Point Search(JPS)[2].The second part of [1] is Safe Flight Corridor Construction. The third part of [1] is Trajectory Optimization, which is a convex optimizaiton problem. Due to the limitation of length of [1], many detail of implementation of algorithms are not mentioned in [1]. In this report, details and theories about each steps of implementation in [1] will be given. In addition, 3-D simulation and visualization will be provided to illustrate how to realize each part of [1]. In the result part, I compared the results using algorithms in this paper with using close form[3].

2 Motivation

Navigation of a Micro Aerial Vehicle(MAV) in an obstacle-cluttered environment is a challenging problem. There are many research about it, on the other hand, visualizing the algorithm in 3D simulation not only could persuade other people to understand why it is trustworthy, but also help engineers to analysis this algorithm. In this report, I will focus on visualize the algorithm in paper ”Planning Dynamically Feasible Trajectories for Quadrotors Using Safe Flight Corridors in 3-D Complex Environments”. It’s a good opportunity for me to understand how to generate trajectory and know it’s strengthens and shortcomings.

3 Related work

The research about trajectory planning can be divided into two kinds, search-based planning and path-based planning. Search-based[4] planning is well-known to be inefficient for high dimensional planning due to the large number of nodes to expand. The precondition of path-based planning is to find a safe path. For path planning[5], there are two kinds of methods—— deterministic path and randomized path. For deterministic method, there are A*[6], D*[7], ARA*[8] and so on. Since this kind of methods need to search the whole space, it is time-consuming and also have a high requirement for computing resource, if search space is large. For randomized method, there are PRM[9], RRT[10], RRT*[10], FMT[11], BIT[12][13] and etc. This kind of methods is suitable for great search space. Path only has the information about whether this path is safe and waypoints leading to destination, while trajectory has information about velocity and acceleration. After finding path, trajectory generation problem can be formulated as a quadratic programming problem[14] with waypoints. [15] use unconstrained QP, but unconstrained QP can't handle inequality constraints(sub-optimal) and can't guarantee the solution is Optimal. R.deits and R.tedrake proposes MIT[16],but this method is hard to get collision constraint and it's hard to build model about free space. Oleynikova and USenko use local planner[17], it's a fast method to find trajectory, but the result is not globally optimal and not complete.

4 Proposed research directions

4.1 Path Planing

4.1.1 JPS(Jump Point Search[2])

Path planning is computational problem to find a valid collision-free path. In [1], the environment is represented as a map with occupancy grids that can be constructed from sensor data, the grids in this map are devided into free space, target space and obstacle space. Many grid-based algorithms can be used to find a valid collision-free path. For A*, all neighbors of current node are added into openlist, while JPS only adds jump point into the openlist, thus jump point search is fast and requires no preprocessing and introduces no memory overheads. In [1], the author uses JPS on 3-D grid maps with uniform voxels. In order to extend the 2-D algorithm proposed in [2] to 3-D, we just need to add the number of neighhbors from 8 to 26,and for diagonal direction search in 3-D map,we only need to add one more dierction compared with 2-D dimension.In order to illustrate the difference between 2D and 3D JPS, definition of natural neighbor[2] and forced neighbor[2] are given below. In Fig.1 and Fig.2 we use pictures in 3D case to illustrate the difference between the definition of forced neighbor and natural neighbor in 2D and that in 3D. In Fig.3, we give the path found by JPS, red grid dedicates path points, green grids dedicates obstacles.

Definition 1. *A node $n \in$ natural neighbour(x) if:*

1. *Assume there is no obstacles in neighbour(x)*
2. *For straight move,*

$$len(\langle p(x), \cdots, n\rangle \setminus x) > len(\langle p(x), x, n\rangle) \tag{1}$$

For diagonal moves,

$$len(\langle p(x), \cdots, n\rangle \setminus x) \geq len(\langle p(x), x, n\rangle) \tag{2}$$

In definition 1, $p(x)$ is the parental node of x, $\langle p(x), \cdots, n\rangle$ means a path from $p(x)$ to n,$\langle p(x), \cdots, n\rangle \setminus x$ means that x does not appear on this path from $p(x)$ to n. In fig.1, white grid indicates natural neighbor.

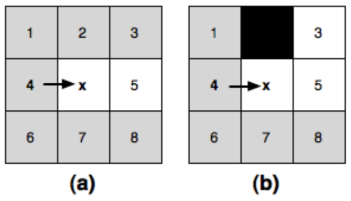


Figure 1: We show one case where a node x is reached from its parent p(x) by a diagonal move. When x is expanded we can prune from consideration all nodes marked grey.White grids indicate natural neighbor.Notice that for natural neighbor, assume there are no obstacles in neighbors, no matter there exists obstacles or not.

Definition 2. *A node $n \in$ forced neighbour(x) if:*

1. *n is not a natural neighbour of x and obstacles exist in neighbour of x.*
2.

$$len(\langle p(x), x, n\rangle) < len(\langle p(x), \cdots, n\rangle \setminus x) \tag{3}$$

In Fig.2, shadow grid indicates forced neighbor, white grids indicate natural neighbor. (Revise after changing the picture) Fig.3 is the result of finding path using JPS.

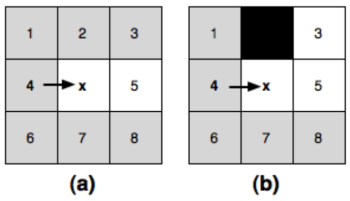


Figure 2: We show one case where a node x is reached from its parent p(x) by a diagonal move. When x is expanded we can prune from consideration all nodes marked grey.Shadow grids indicate forced neighbor.Notice that grid x, forced neighbor only exist when there are obstacles in its neighbor and forced neighbor don't belong to natural neighbor.

4.1.2 Simplify path

Since we use grid 3-D map, during the searching processing, the algorithm searching direction is limited in neighbor grid(only 27 direction), however in practice,quadrotors can go in any direction, so the path points found may be redundant for the quadrotors. The process of how to prune redundant points are described in algorithm1.

Algorithm 1 Simplify path

```
Require: path:way points
  s = 1
  simplePath
  for i from 2 to n do
    if line of point s and point i is not collide with obstacles and line of point
    s and point i+1 is collide with obstacles then
      simplePath.push(s)
      s = i
    end if
  end for
  return simplePath
```

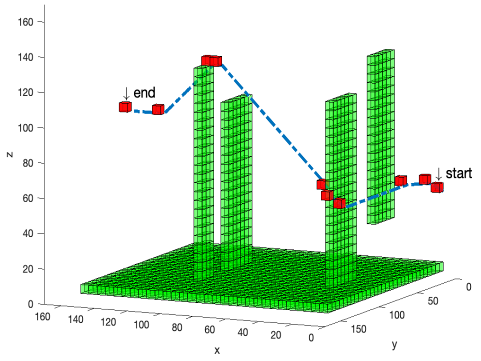
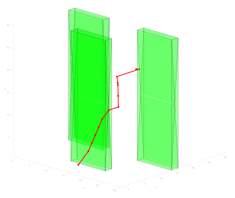
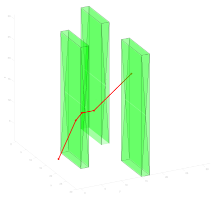


Figure 3: Red girds is the path found by JPS. Green grids indicates obstacles. The start and end points are marked in this picture.This 3D grid map is 30*30 grids and each grid size is 5*5*5.It doesn't matter if quadrotors fly close to obstacles, because obstacles are dilated.



(a) Path without simplification



(b) Path with simplification

Figure 4: Comparison of way points between using simplify path and not

4.2 Safe Flight Corridor Construction

1) Find Sphere
   A piece-wise linear path P found in section 4.3(Fig.5) from start to goal in the free space is denoted as $P = \langle p_0 \rightarrow p_1 \rightarrow ...p_n \rangle$. The i^{th} line segement is represented as $L_i = \langle P_i \rightarrow P_{i+1} \rangle$, we use L as the diameter of a sphere. At this time, it is possible obstacles is inside the sphere(fig.6).

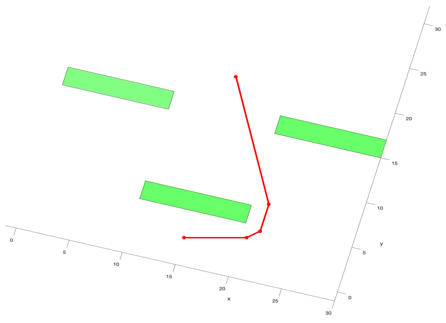
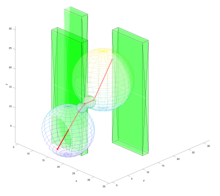
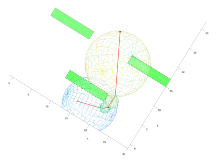


Figure 5: A top view of the piece-wise linear path P. Red line indicates the path, red Point indicates the way point and green grid indicates obstacles.



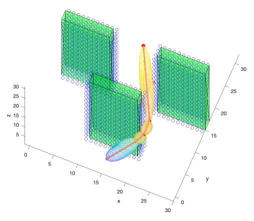
(a) Initial sphere from one side view.



(b) Initial sphere from a top view.

Figure 6: Initial sphere for each pieces of path from two different of view.

2) Shrink minor axes
   It's obviously that in figure6.b, some parts of obstacles are contained in the initial spheres. Start with a sphere, we find the closest point p_i to the center of L_i and adjust the length of short axes such that a new ellipsoid touches this p_i.Repeat the same procedure until no obstacles are contained in the ellipsoid.


(a) Final spheroid from one side view.

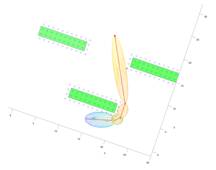

(b) Final spheroid from a top view.

Figure 7: Final spheroid excludes all the obstacles. In practice, the data of obstacles we get from senors is point cloud,thus in this picture, using small blue circles to indicates point cloud. Notice that some blue circles is outside of the obstacle(green bricks),it is normal because in order to make sure the safe of quadrotors, we dilate obstacles firstly to avoid quadrotors hitting the edge of obstacles. In figure.b, the red point indicates that one point of obstacles is exactly on the ellipsoid interface.

3) Dilate ellipsoid and find hyperplane:
Find the intersection point P_0 for the ellipsoid and obstacles. P_0 is on the surface of ellipsoid. Draw a tangent plane through point P_0 and remove the obstacles outside the tangent plane. Then dilate the ellipsoid, find next intersection point, repeat the previous process until all the obstacles are removed. Actually,according to the ellipsoid we find in step3),if we find the closet point(intersection point) from the center of a current ellipsoid, we can get hyperplanes directly. Namely, although this step equals dilating the ellipsoid, there is no need to solve the dilate ellipsoid. Next, we will explain how to get the equation of hyperplane.
The standard equation of ellipsoid in matrix representation is:

$$x^T \Lambda^{-1} x = 1 \tag{4}$$

in equation(4), $\Lambda = \begin{bmatrix} r_1^2 & & & \\ & r_2^2 & & \\ & & \ddots & \\ & & & r_n^2 \end{bmatrix}$ and $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$

x_i indicates the *ith* axis, r_i are half the length of the principal axes.
Now, rotate this ellipsoid and let the principal semi-axes of the ellipsoid aligned with path L_i, assume the rotation matrix is A, we have

$$(A^T x)^T \Lambda^{-1} (A^T x) = x^T (A \Lambda^{-1} A^T) x = 1 \tag{5}$$

Then move the center of this ellipsoid to the center of path L_i, we have:

$$f(x) = (x - x_c)^T (A \Lambda^{-1} A^T)(x - x_c) = 1 \tag{6}$$

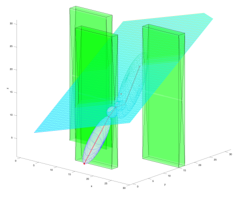
For equation(6), if we dilate the ellipsoid, namely, let $r_i$becomes to λr_i, we only need to change: $\Lambda = \begin{bmatrix} \lambda^2 r_1^2 & & & \\ & \lambda^2 r_2^2 & & \\ & & \ddots & \\ & & & \lambda^2 r_n^2 \end{bmatrix}$.

In addition, a normal vector of a point on the surface of an ellipsoid is:

$$\bigtriangledown f = \left\langle \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right\rangle \tag{7}$$

Notice that after dilating the ellipsoid,$\bigtriangledown f = \frac{1}{\lambda^2} \left\langle \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right\rangle$, which means $\bigtriangledown f$ is not related to λ. Assume the intersection point is $p_0 = (x_0, y_0, z_0)$, it is easy for us to get the equation of the hyperplane through p_0. The equation of this hyperplane is:

$$\bigtriangledown f \cdot (x - x_0, y - y_0, z - z_0) = 0 \tag{8}$$


(a) One hyperplane from one side view.

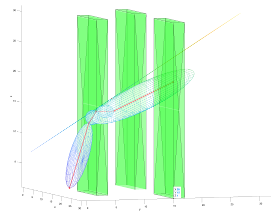

(b) One hyperplane from a different view.

Figure 8: Hyperplane and dilated ellipsoid from two different of view.The red point which is on the hyperplane is intersection point for the ellipsoid and obstacles.In practice, there is no need to draw the dilated ellipsoid.

4) Getting polyhedron:
In step3),we dilate the ellipsoid and get hyperplanes till we move away all of the point cloud of obstacles. According to equation(8), we can write the constraint of hyperplanes as:

$$\begin{bmatrix} \bigtriangledown f_1 \\ \bigtriangledown f_2 \\ \vdots \\ \bigtriangledown f_n \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{9}$$

In equation(9), $\bigtriangledown f_i(x, y, z) = b_i$ is the equation of *ith* hperplane.

4.3 Trajectory Optimization

For smooth trajectories, we have

$$x^*(t) = \underset{x(t)}{argmin} \int_0^T (x^{(n)})^2 dt \tag{10}$$

when n = 1, this equation indicates shortest distance; n = 2, it indicates minimum acceleration;n = 3, it indicates minimum jerk; when n = 4, it means minimum snap[18]. In this paper, we choose minimum-snap trajectory as the cost function:

$$x^*(t) = \underset{x(t)}{argmin} \int_0^T \left\| x^{(4)} \right\|^2 dt \tag{11}$$

From the Euler-Lagrange equations, a necessary condition for the optimal trajectory is:

$$x^{(8)} = 0 \tag{12}$$

Thus, the minimum-snap trajectory is a 7^{th} order polynomial. If the trajectory has k + 1 waypoints(including begin point and end point), we need to design a trajectory x^t such that:

$$\begin{cases} t = [t_0, t_1, \cdots, t_k]^T \\ x = [x_0, x_1, \cdots, t_k]^T \end{cases} \tag{13}$$

Thus, the trajectory will be a $7^{th}-order$ piecewise polynomial with k segments:

$$x(t) = \begin{cases} c_{1,7}t^7 + c_{1,6}t^6 + c_{1,5}t^5 + c_{1,4}t^4 + c_{1,3}t^3 + c_{1,2}t^2 + c_{1,1}t + c_{1,0} \\ c_{2,7}t^7 + c_{2,6}t^6 + c_{2,5}t^5 + c_{2,4}t^4 + c_{2,3}t^3 + c_{2,2}t^2 + c_{2,1}t + c_{2,0} \\ \vdots \\ c_{k,7}t^7 + c_{k,6}t^6 + c_{k,5}t^5 + c_{k,4}t^4 + c_{k,3}t^3 + c_{k,2}t^2 + c_{k,1}t + c_{k,0} \end{cases} \tag{14}$$

We also could write equation(8) as

$$x(t) = \begin{cases} \left[1, t, t^2, ..., t^n\right] \cdot c_1 & t_0 \le t < t_1 \\ \left[1, t, t^2, ..., t^n\right] \cdot c_2 & t_1 \le t < t_2 \\ \vdots & \\ \left[1, t, t^2, ..., t^n\right] \cdot c_k & t_{k-1} \le t < t_k \end{cases} \tag{15}$$

In equation(9), $c_i = [c_{i_0}, c_{i_1}, ..., c_{i_n}]^T$ and n = 7.
For begin point and end point in the trajectory, we have position, velocity and acceleration constraint. For example, at the begin point, position is p_0, velocity is v_0 and acceleration is a_0, we have:
Position constraint:

$$\left[1, t_0, t_0^2, ..., t_0^n, \underbrace{0...0}_{(k-1)(n+1)}\right] c = p_0 \tag{16}$$

Velocity constraint:

$$\left[0, 1, 2t_0, ..., nt_0^{n-1}, \quad \underbrace{0...0}_{(k-1)(n+1)}\right] c = v_0 \tag{17}$$

Acceleration constraint:

$$\left[0, 0, 2, ..., n(n-1)t_0^{n-2}, \quad \underbrace{0...0}_{(k-1)(n+1)}\right] c = a_0 \tag{18}$$

For the joint point between two segments, the position, velocity, and acceleration between adjacent segments have the same value. Take the *ith* segment and the $(i+1)th$ segment for instance, at the joint point we have position constraint:

$$\left[\underbrace{0, ..., 0}_{(i-1)(n+1)}, 1, t_i, t_i^2, ..., t_i^n, -1, -t_i, -t_i^2, ..., -t_i^n, \quad \underbrace{0...0}_{(k-i-1)(n+1)}\right] c = 0 \tag{19}$$

Use all of the constraints, we have equation:

$$\begin{bmatrix} 1, t_0, t_0^2, ..., t_0^n, \quad \underbrace{0...0}_{(k-1)(n-1)} \\ 0, 1, 2t_0, ..., nt_0^{n-1}, \quad \underbrace{0...0}_{(k-1)(n-1)} \\ 0, 0, 2, 2t_0, ..., n(n-1)t_0^{n-2}, \quad \underbrace{0...0}_{(k-1)(n-1)} \\ \vdots \\ \underbrace{0...0}_{(i-1)(n+1)}, 1, t_i, t_i^2, \cdots, t_i^n, \quad \underbrace{0...0}_{(k-1)(n-1)} \\ \vdots \\ \underbrace{0...0}_{(k-1)(n-1)}, 1, t_k, t_k^2, \cdots, t_k^n \\ \underbrace{0...0}_{(k-1)(n-1)}, 0, 1, 2t_k, \cdots, nt_k^{n-1} \\ \underbrace{0...0}_{(k-1)(n-1)}, 0, 0, 2, \cdots, n(n-1)t_k^{n-2} \\ \underbrace{0, ..., 0}_{(i-1)(n+1)}, 1, t_i, t_i^2, ..., t_i^n, -1, -t_i, -t_i^2, ..., -t_i^n, \quad \underbrace{0...0}_{(k-i-1)(n+1)} \\ \underbrace{0, ..., 0}_{(i-1)(n+1)}, 0, 1, 2t_i, ..., nt_i^{n-1}, 0, -1, -2t_i, ..., -nt_i^{n-1}, \quad \underbrace{0...0}_{(k-i-1)(n+1)} \\ \underbrace{0, ..., 0}_{(i-1)(n+1)}, 0, 0, 2, ..., n(n-1)t_i^{n-2}, 0, 0, -2, ..., -n(n-1)t_i^{n-2}, \quad \underbrace{0...0}_{(k-i-1)(n+1)} \end{bmatrix}_{4k+2} c = \begin{bmatrix} p_0 \\ v_0 \\ a_0 \\ \vdots \\ p_i \\ \vdots \\ p_k \\ v_k \\ a_k \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{20}$$

Equation(19) is linear constraint of equation(10), next we will find equation(10) is a quadratic objective function, then we can use quadratic programming to solve the best coefficient in equation(13).
For equation(10), we have:

$$\begin{aligned}
x^*(t) &= \underset{x(t)}{argmin} \int_0^T (x^{(4)})^2 dt \\
&= min \sum_{i=1}^{k} \int_{t_{i-1}}^{t_i} (x^{(4)})^2 dt \\
&= min \sum_{i=1}^{k} \int_{t_{i-1}}^{t_i} ([0,0,0,0,24,\cdots,\frac{n!}{(n-4!)}t^{n-4}] \cdot c_i)^T [0,0,0,0,24,\cdots,\frac{n!}{(n-4!)}t^{n-4}] \cdot c_i \; dt \\
&= min \sum_{i=1}^{k} c_i^T \int_{t_{i-1}}^{t_i} [0,0,0,0,24,\cdots,\frac{n!}{(n-4!)}t^{n-4}]^T [0,0,0,0,24,\cdots,\frac{n!}{(n-4!)}t^{n-4}] dt \; c_i \\
&= min \sum_{i=1}^{k} c_i^T Q_i c_i
\end{aligned} \tag{21}$$

In equation(20),

$$\begin{aligned}
Q_i &= \int_{t_{i-1}}^{t_i} [0,0,0,0,24,\cdots,\frac{n!}{(n-4!)}t^{n-4}]^T [0,0,0,0,24,\cdots,\frac{n!}{(n-4!)}t^{n-4}] dt \\
&= \begin{bmatrix} 0_{4\times 4} & 0_{4\times(n-3)} \\ 0_{(n-3)\times 4} & \frac{r!}{(r-4)!}\frac{c!}{(c-4)!}\frac{1}{(r-4)+(c-4)+1}(t_i^{r+c-7} - t_{i-1}^{r+c-7}) \end{bmatrix}
\end{aligned} \tag{22}$$

c_i is the coefficient of the polynomial of *ith* segment path, r and c is the index of rows and columns in this matrix. Now if we assume $Q = \begin{bmatrix} Q_1 & & & \\ & Q_2 & & \\ & & \ddots & \\ & & & Q_k \end{bmatrix}$,the cost function becomes to $min\ c^T Q c$. In addition, equation(9) and equation(20) will be used as a constraint of this cost function (10).

Notice that from now on, in this section from equation(10) to equation(22), we only consider one dimension, while in practice, quadrotors fly in 3D dimension. Next, we will give equations combining three dimension and we use subscripts x y z to indicate which dimension this variable belongs to.
For cost function, it becomes to:

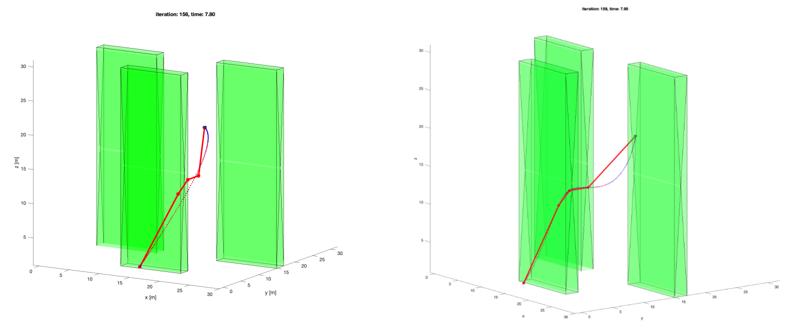
$$f(c_x, c_y, c_z) = min \begin{bmatrix} c_x^T & c_y^T & c_z^T \end{bmatrix} \begin{bmatrix} Q_x & & \\ & Q_y & \\ & & Q_z \end{bmatrix} \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix} \tag{23}$$

Simplify equation(20) to form $Tc = P$,thus for three dimension, we have:

$$\begin{bmatrix} T_x & T_y & T_z \end{bmatrix} \begin{bmatrix} c_x & & \\ & c_y & \\ & & c_z \end{bmatrix} = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} \tag{24}$$

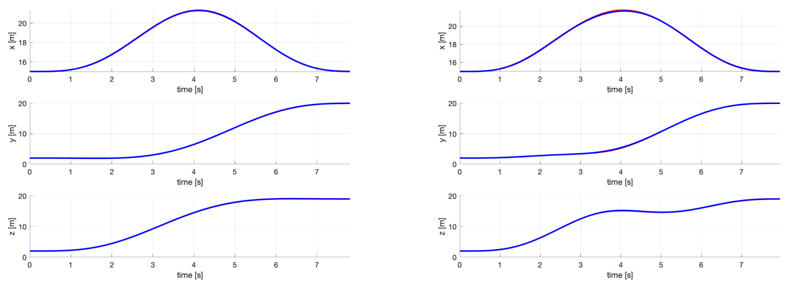
5 Result

Now, for three dimension, we have cost function —— equation(23), two constraint function—— equation(24) and equation(9), solve the matrix c, we can have the equation of the trajectory.



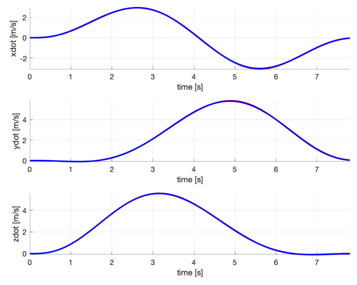
(a) Trajectory generated by SFC (b) Trajectory generated by Close Form

Figure 9: Trajectory generated by SFC and Close Form. Notice that in Figure9(a), the trajectory not pass each waypoints, while in figure9(b), the trajectory generated by close form pass each waypoints.

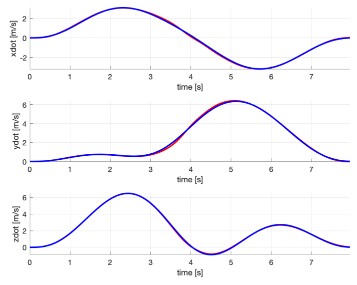


(a) Trajectory tracking(position) of SFC (b) Trajectory tracking(position) of Close Form

Figure 10: Trajectory tracking of position by SFC and Close Form. Blue line is the plan position and red line is actual position.



(a) Trajectory tracking(velocity) by SFC



(b) Trajectory tracking(velocity) by close form

Figure 11: Trajectory tracking of velocity by SFC and Close Form. Blue line is the plan velocity and red line is actual velocity.

5.1 Reference

[1] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C.J. Taylor, et al., “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] D. Harabor and A. Grastien. 2011. “Online Graph Pruning for Pathfinding on Grid Maps.” In Proceedings of the 25th National Conference on Artificial Intelligence (AAAI), San Francisco, USA.

[3]C. Richter, A. Bry, and N. Roy, “Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments,” in Proc. Int. Symp. Robot. Res., 2013

[4]S. Liu, N. Atanasov, K. Mohta, and V. Kumar, “Search-based motion planning for quadrotors using linear quadratic minimum time control,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2017, pp. 2872–2879

[5]A. Gasparetto, P. Boscariol, A. Lanzutti, R. Vidoni, “Path planning and trajectory planning algorithms: a general overview”, in: Mechanisms and Machine Science, 2015, pp. 3–27, doi:10.1007/978-3-319-14705-5_1.

[6]T. Chen, G. Zhang, X. Hu, and J. Xiao, “Unmanned aerial vehicle route planning method based on a star algorithm,” in Proc. 13th IEEE Conf. Ind. Electron. Appl. (ICIEA), Wuhan, China, May/Jun. 2018, pp. 1510–1514.

[7] Matsubara Y, Sakurai Y, Yoshikawa M (2011) “D-Search: an efficient and exact search algorithm for large distribution sets”. Knowl Inf Syst 29(1): 131–157

[8]Li, Bo ; Gong, Jianwei ; Jiang, Yan ; Nasry, Hany ; Xiong, Guangming "ARA+: Improved Path Planning Algorithm Based on ARA" Proceedings of the The 2012 IEEE/WIC/ACM International Joint Conferences Web Intelligence Intelligent Agent Technology, 04 December 2012, Vol.2, pp.361-365

[9] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065), San Francisco, CA, USA, 2000, pp. 521-528 vol.1, doi: 10.1109/ROBOT.2000.844107.

[10] Noreen, I.; Khan, A.; Habib, Z. "A Comparison of RRT, RRT* and RRT*-Smart Path Planning Algorithms."Int. J. Comput. Sci. Netw. Secur. 2016, 16, 20–27.

[11] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," Int. J. Robot. Res., vol. 34, pp. 883–921, 2015.

[12] Menglu Lan et al., "BIT*-based path planning for micro aerial vehicles," IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society, Florence, 2016, pp. 6079-6084, doi: 10.1109/IECON.2016.7792953.

[13] J. D. Gammell, S. S. Srinivasa and T. D. Barfoot, "Batch Informed Trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, 2015, pp. 3067-3074, doi: 10.1109/ICRA.2015.7139620.

[14] M. J. Van Nieuwstadt and R. M. Murray, "Real time trajectory generation for differentially flat systems," 1997.

[15] Adam Bry, Charles Richter, Abraham Bachrach, and Nicholas Roy. 2015. "Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments."The International Journal of Robotics Research 34, 7 (2015), 969–1002.

[16] Deits, R., Tedrake, R. (2015). "Efficient mixed-integer planning for UAVs in cluttered environments. In IEEE international conference on robotics and automation (ICRA)".

[17] http://helenol.github.io/publications/thesis$_2$019$_o$*leynikova.pdf*
Elena Oleynikova, "Mapping and Planning for Safe Collision Avoidance Onboard Micro-Aerial Vehicles".

[18] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," inProc. 2011 IEEE Int. Conf. Robot.Autom.,2011