无人飞行器基础路径规划

韩志超　　杨晨　　潘能

【摘要】 我们对无人飞行器的运动问题从路径规划和轨迹生成两个方面和进行了研究与仿真分析。在路径规划部分，我们分别测试了 A*算法、JPS 算法、RRT*算法在不同环境下的性能，其中 JPS 算法的综合性能最优；在轨迹生成方面，minimum-snap 的方法存在一定的问题，比如不能保证避障，数值稳定性差，我们从其缺点出发着手改进了该算法，另外我们还采用了贝塞尔曲线加飞行走廊的方法进行轨迹生成，在动态避障和无人机跟踪测试中的整体效果非常好。

【关键词】 路径规划，轨迹生成，飞行走廊，贝塞尔曲线

Basic path planning of drones

Zhichao Han　Chen Yang　Neng Pan

Abstract We carried out research and simulation analysis on the motion problem of UAV from two aspects of path planning and trajectory generation. In the path planning part, we tested the performance of the A* algorithm, JPS algorithm, and RRT* algorithm in different environments. Among them, the comprehensive performance of the JPS algorithm is optimal. In terms of trajectory generation, the minimum-snap method has certain problems. For example, obstacle avoidance cannot be guaranteed, and the numerical stability is poor. We started to improve the algorithm from its shortcomings. In addition, we also used the method of Bezier curve and flight corridor for trajectory generation. In dynamic obstacle avoidance and drone tracking tests The overall effect is very good.

Key words path-planning, Trajectory generation, Flight Corridor, Bezier Trajectory

1　引言

我们实现了无人飞行器完整的路径规划与轨迹生成。首先我们实现了三种路径规划算法并且比较了它们的效率。接着我们复现了传统的 Minimum-Snap[1]轨迹生成并且发现其中问题进行改进。实验表明，我们的改进基本完善了传统 Minimum-Snap 存在的问题，但是仍然有所不足。因此之后我们又引入了飞行走廊和贝塞尔曲线进行轨迹生成，并提出了一种高效，快速的飞行走廊生成方式，非常适合用来动态规划。最后我们将我们的代码在动态避障和无人机跟踪两个场景下面进行测试，实验证明我们的方法生成轨迹具有很强的实时性与光滑性。

2　路径规划

我们先后采用了 A*算法、RRT*算法、JPS 算法来实现无人机运动的路径规划。

2.1　A*规划

A*算法的核心是构造代价函数：

$$F(n) = G(n) + H(n)$$

$G(n)$是从起点移动到当前点的代价；$H(n)$是从当前点移动到终点的估计代价。

算法流程伪代码如下：

Algorithm 1: A* Search

```
Begin
    openlist.add(startnode)
    While !isempty(openlist)
        node = openlist.mininode
        if node == endnode
            break
        for nnodes around node
            if nnode is unexpanded
                f(n) = g(n) + h(n)
                openlist.add(nnode)
                nnode.father = node
            else if f(n) > f(n-1) + d
                f(n) = f(n-1) + d
                nnode.father = node
End
```

A*算法路径规划的仿真效果如图 1 所示：

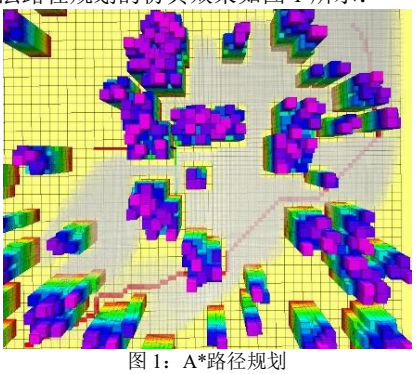

图 1：A*路径规划

红线表示 A*路径，灰格表示 A*搜索空间

此外我们对 A*算法的启发函数进行了测试对比，我们固定地图，取相同的起点终点，对三种启发式函数（曼哈顿距离、欧式距离、对角线距离）进行了测试，其中的三组测试结果如下：

表 1：测试地图一

启发函数	时间/ms	路程/m	节点数
曼哈顿距离	0.932	7.476	127
欧式距离	6.001	7.195	7556
对角线距离	2.261	7.195	2403

表 2：测试地图二

启发函数	时间/ms	路程/m	节点数
曼哈顿距离	2.588	6.535	1655
欧式距离	3.576	6.495	5154
对角线距离	2.875	6.495	2839

表 3：测试地图三

启发函数	时间/ms	路程/m	节点数
曼哈顿距离	1.926	9.464	1304
欧式距离	10.912	9.114	21526
对角线距离	7.429	9.114	16328

根据测试结果总结结论如下：

最优性方面：欧式距离和对角线距离都是最优，曼哈顿距离不具有最优性。

时间效率方面：曼哈顿距离最快，其次是对角线距离，最后是欧式距离。

内存占用方面：曼哈顿占用最少，其次是对角线距离，最后是欧式距离。

稳定性：三种启发式函数都有稳定性。

2.2　JPS 规划

JPS 算法的核心是解决 A*算法中对称性问题(Tie Breaker)

为此 JPS 定义了跳点概念(Jump Point)，顾名思义，跳点即运动方向改变的点，通过检索跳点位置，而在非跳点节点保持之前的运动方向不变，

就可以避免过度搜索。
另外一个概念叫强迫邻居(Forced Neighbor)，凡是有强迫邻居的点则为跳点，强迫邻居的出现会增加当前点的最优前进方向(即经过当前点的最优且唯一的路径的方向)，此时物体在当前节点可能改变运动方向。
算法流程伪代码如下：

Algorithm 2: Jump Point Search

```
Begin
    openlist.add(startnode)
    While !isempty(openlist)
        node = openlist.mininode
        if node == endnode
            break
        for directions from node
            if nnode in direction has forced
            neighbor
                if nnode is unexpanded
                    f(n) = g(n) + h(n)
                    openlist.add(nnode)
                    nnode.father=node
                else if f(n) > f(n-1) + 1
                    f(n) = f(n-1) + 1
                    nnode.father=node
End
```

JPS 算法路径规划的仿真效果如图 3 所示：

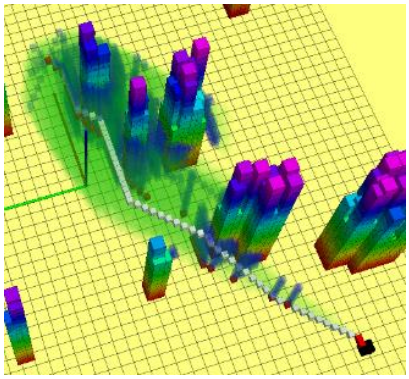

图 2：JPS 路径规划
红格表示 JPS 路径点，蓝格表示 JPS 搜索空间，白格表示 A*路径，绿格是 A*搜索空间

2.3　RRT*规划

RRT*算法的核心是使用随机采样的方法使得路径树不断生长，直到到达终点，同时采用代价函数来选取拓展节点领域内的最小节点为父节点并重新连接节点保证渐进最优性。
算法流程伪代码如下：

Algorithm 3: RRT* Search

```
Begin
    nodetree.add(startnode)
    While endnode is far from nodetree
        generate random node xrand
        connect xrand and nearest node in
        nodetree xnearest
        search potential parent node in circle
        centered in xrand with radius ri
        for all potential nodes
            if cost(potentialnode + xrand) <
            cost(parentnode + xrand)
            &&isfree(potentialnode, xrand)
                xrand.parent=potentialnode
        nodetree.add(xrand)
End
```

RRT*算法的路径规划仿真效果如图 3 所示：

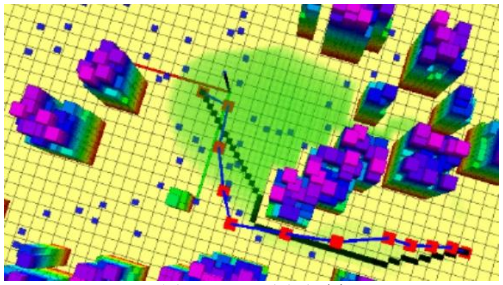

图 3：RRT*路径规划
蓝线表示 RRT*搜路径，蓝格表示 RRT*搜索空间，黑格表示 A*路径，绿格表示 A*搜索空间

2.4　算法对比

我们分别使用了 A*，JPS，RRT*三种算法进行路径规划，并选取随机地图，取相同的起点终点分别对三种路径规划算法进行测试，这里我们随机选出三组测试样例如下：

表 4：测试地图一

算法	时间/ms	路程/m	节点数
A*	3.197	6.121	3312
JPS	1.837	6.121	267
RRT*	0.041	6.393	9

表 5：测试地图二

算法	时间/ms	路程/m	节点数
A*	11.415	12.278	12424
JPS	10.954	12.278	3066
RRT*	98.43	13.808	1938

表 6：测试地图三

算法	时间/ms	路程/m	节点数
A*	5.76	8.766	10664
JPS	6.319	8.766	1684
RRT*	2.494	13.98	407

根据测试结果总结结论如下：
最优性方面：JPS 与 A*得到的路径都是最优，RRT*不是最优。
时间效率方面：JPS 和 A*时间效率差距不是太大，而如果地图简单，RRT*往往最快，但是地图复杂，RRT*往往耗费最多时间。
内存占用方面：JPS 内存占用远远小于 A*，而 RRT*内存占用一般来讲也是很小的。
稳定性：A*和 JPS 都比较稳定，但是 RRT*稳定性不算好，当地图复杂，RRT*效率迅速下降。

2.5　路径简化

路径规划生成的原始节点数量庞大，直接用于轨迹生成过于冗余，对于轨迹生成的光滑性与时间效率不利，因此需要进行化简。
为了简化路径，保留路径上的关键节点，我们采用了两种路径简化的方法：拐点法和 Line of Sight。

拐点法是从起点开始，寻找第一个无法与上个拐点，及它的上一个点共线的点，将它的上一个点设置为拐点。原理图如图 4 所示：

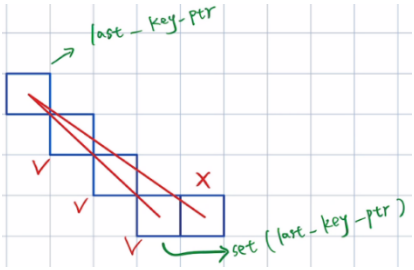

图 4：找拐点法原理

算法流程伪代码如下：

Algorithm 4: Find Inflection Points

Begin
curnode = startnode
While curnode != endnode
flag=checkinLine(curnode, curnode→last, lastkeynode)
if flag
curnode = curnode→next
else
lastkeynode = curnode→last
keynodes.add(lastkeynode)
End

找拐点法的仿真效果如图 5 所示：

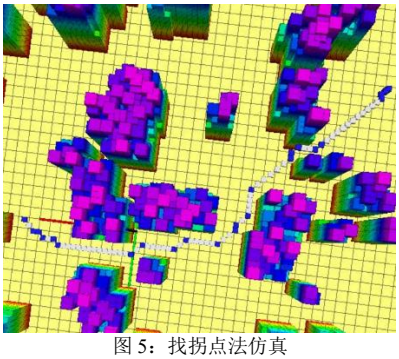

图 5：找拐点法仿真

白格表示路径搜索算法得到的路径，蓝格表示简化关键点

Line of Sight 是从起点开始，寻找离自己最远的连线无碰节点，将它设作关键点，再将当前点更新为该关键点，循环更新整个路径。

原理图如图 6 所示：

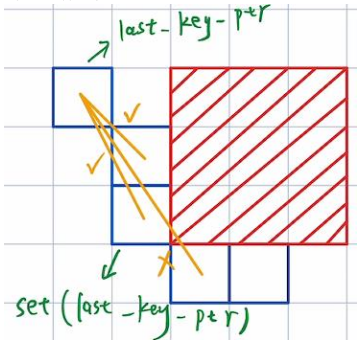

图 6：Line of Sight 原理

算法流程伪代码如下：

Algorithm 5: Line of Sight

Begin
curnode = startnode
While curnode != endnode
While curnode != endnode
flag=collisioncheck(curnode, lastkeynode)
if flag
maxnode = curnode
curnode = curnode→next
lastkeynode = maxnode
curnode = maxnode
keynodes.add(lastkeynode)
End

Line of sight 的仿真效果如图 7 所示：

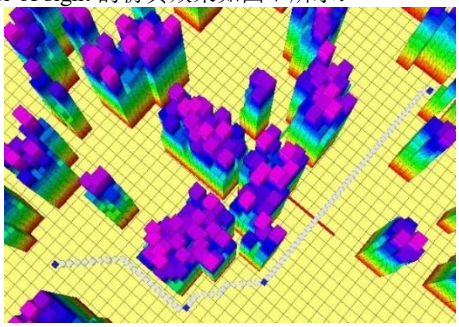

图 7：Line of Sight 仿真

白格表示路径搜索算法得到的路径，蓝格表示简化关键点

对比知道 Line of Sight 得到的化简点化简得更加充分，后续工作中使用 Line of Sight 进行化简

3　轨迹生成

由于路径搜索算法得到的路径点没有考虑机器人的运动学约束，无法直接用于机器人运动控制，因此需要进行轨迹生成。

对于相邻的两个路径节点，我们采用 n 阶（n 取 7）多项式来拟合两节点之间的路径，而轨迹规划的目的就是求取轨迹的多项式系数：

$$f(t)=\begin{cases} f_1(t)\doteq\sum_{i=0}^{N}p_{1,i}t^i & T_0\le t\le T_1\\ f_2(t)\doteq\sum_{i=0}^{N}p_{2,i}t^i & T_1\le t\le T_2\\ \vdots & \vdots\\ f_M(t)\doteq\sum_{i=0}^{N}p_{M,i}t^i & T_{M-1}\le t\le T_M \end{cases}$$

我们希望轨迹满足一系列的约束条件，比如：希望设定起点和终点的位置、速度或加速度，希望相邻轨迹连接处平滑（位置连续、速度连续等），设定最大速度、最大加速度等，甚至希望轨迹在规定空间内（corridor）等等。

有了路径简化之后的一系列节点，我们先后采用了 Minimum-Snap 曲线、飞行走廊结合 Bezier 曲线的方法来实现无人机运动的轨迹生成。

3.1　Minimum-Snap 曲线

Minimum-Snap 顾名思义，最小化目标是 Snap(加加加速度)平方的积分和，从而降低无人机飞行的能量损耗，优化问题可以建模为：

$$\min J=\min\int_{T_{i-1}}^{T_i}(f_i^4(t))^2dt \tag{1}$$

$$f_i(T_i)^k=x_i^k \tag{2}$$

$$f_i^k(T_i)=f_{i+1}^k(0) \tag{3}$$

传统方法一般使用迭代来求解，本文中我们为了提升求解效率，采用了闭式求解

闭式求解的具体的流程如下：

找到映射矩阵M，将多项式系数映射到具有实际意义的物理量。

$$M_jP_j=d_j \tag{4}$$

$$J = \begin{bmatrix} p_1 \\ \vdots \\ p_M \end{bmatrix}^T \begin{bmatrix} Q_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & Q_M \end{bmatrix} \begin{bmatrix} p_1 \\ \vdots \\ p_M \end{bmatrix} \quad (5)$$

其中M_j代表第 j 段轨迹的映射矩阵，P_j代表第 j 段轨迹的系数向量，d_j代表第 j 段轨迹的实际物理量（位置，速度，加速度，Jerk）

求取选择矩阵 C 来将变量排序为自由变量（d_p）和约束变量（d_F）

$$C^T \begin{bmatrix} d_F \\ d_p \end{bmatrix} = \begin{bmatrix} d_1 \\ \vdots \\ d_M \end{bmatrix} \quad (6)$$

$$J = \begin{bmatrix} d_F \\ d_p \end{bmatrix}^T C M^{-T} Q M^{-1} C^T \begin{bmatrix} d_F \\ d_p \end{bmatrix} = \begin{bmatrix} d_F \\ d_p \end{bmatrix}^T \begin{bmatrix} R_{FF} & R_{FP} \\ R_{PF} & R_{PP} \end{bmatrix} \begin{bmatrix} d_F \\ d_p \end{bmatrix} \quad (7)$$

转化为无约束最优化问题，进行求极值点。

$$J = \begin{bmatrix} d_F \\ d_p \end{bmatrix}^T \begin{bmatrix} R_{FF} & R_{FP} \\ R_{PF} & R_{PP} \end{bmatrix} \begin{bmatrix} d_F \\ d_p \end{bmatrix} \quad (8)$$

$$\frac{\partial J}{\partial d_p} = 0 \quad (9)$$

$$d_p^* = -R_{pp}^{-1} R_{FP}^T d_F \quad (10)$$

根据极值点可以反推得到各段轨迹的多项式系数

多数情况下，该方法可以生成较好的轨迹：

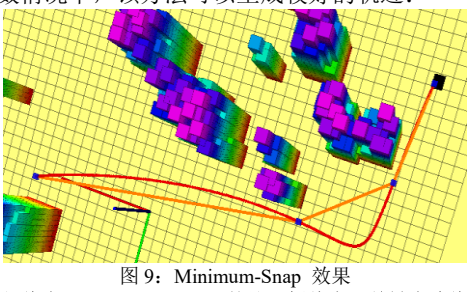

图 9：Minimum-Snap 效果

红线表示 Minimum-Snap 轨迹，橙线表示关键点连线

但是该方法生成的轨迹并不能确保无碰撞，如下图所示：

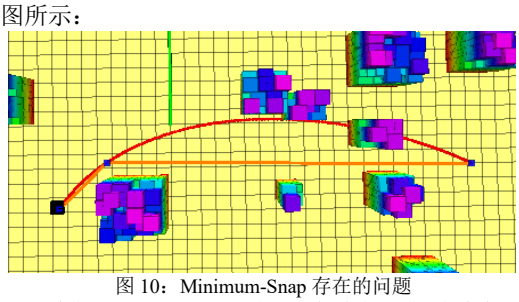

图 10：Minimum-Snap 存在的问题

红线表示 Minimum-Snap 轨迹，橙线表示关键点连线

此外 Minimum-Snap 在两段轨迹间长度差距大时存在数值不稳定性，对此对传统 Minimum-Snap 做了以下改进：

①数值稳定性可通过标准化处理来优化

$$f(t) = \sum_{i=0}^{n} p_i \tau^i = \sum_{i=0}^{n} p_i (kt)^i \quad (0 < \tau < 1) \quad (11)$$

②为了解决避障问题，我们首先提出了等分添加约束点的方法，在过长的关键点间添加等分约束点，来近似“拉直”轨迹，效果如下：

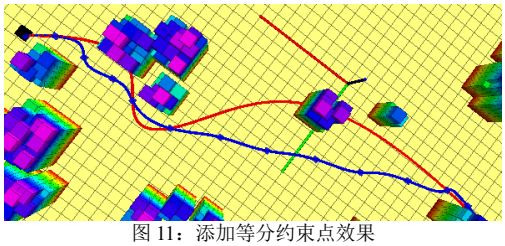

图 11：添加等分约束点效果

红线表示原始 Minimum-Snap 轨迹（有碰），蓝线表示等分添加约束点后得到的轨迹（无碰）

这种优化方法简单，并且运行很快，但是无法从根本上确保避障，并且可能会添加不必要的冗余约束点，不利于轨迹生成与数值稳定性

为此，我们进一步提出了迭代式添加约束点算法，策略是：最开始不添加任何约束点，进行轨迹生成。轨迹生成之后进行每一段轨迹的检测。如果发现某一段轨迹碰撞，则在轨迹两端点中间添加约束点。重复上述过程。伪代码如下

Algorithm 6:
Recursively Insert Constrained Points

```
Begin
    While true
        flag = true
        for all segments in trajectory
            if isoccupied(segment)
                flag = false
                insert(segment, node)
        if flag
            break
    generateTrajectory(trajectory)
End
```

效果图如下：

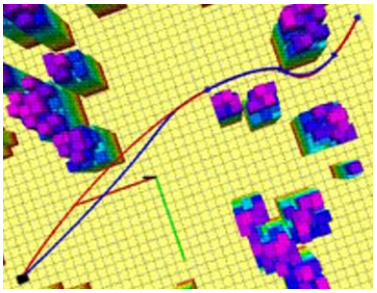

图 12：迭代添加约束点效果

红线表示原始 Minimum-Snap 轨迹（有碰），蓝线表示迭代添加约束点后得到的轨迹（无碰）

迭代式添加约束点解决了等分添加约束点的约束点冗余问题，但仍无法做到本质避障，并且在拐角处容易添加过多约束点，因此我们考虑使用性能更佳的飞行走廊结合贝塞尔曲线的轨迹生成方法。

3.2　飞行走廊结合贝塞尔曲线

我们利用飞行走廊生成一个绝对安全区域，接着再利用贝塞尔曲线的性质将轨迹约束在飞行走廊内，实现轨迹生成。这样便可以充分利用空间信息，在实现根本避障的同时使得轨迹更加光滑。

3.2.1　飞行走廊

我们根据简化后的路径节点生成了飞行走廊。所谓飞行走廊也就是飞行器的安全联通区域。

我们提出了一种高效生成飞行走廊的算法，思想是从起点开始向后续节点生成走廊，直到发生碰撞，将起点与碰撞点的上一个节点生成的走廊加入走廊集合，再从该走廊的终点开始继续向后生成走廊，

直到到达目标点，最后对走廊集合进行膨胀，来增加走廊间的重叠，流程图如图 13 所示

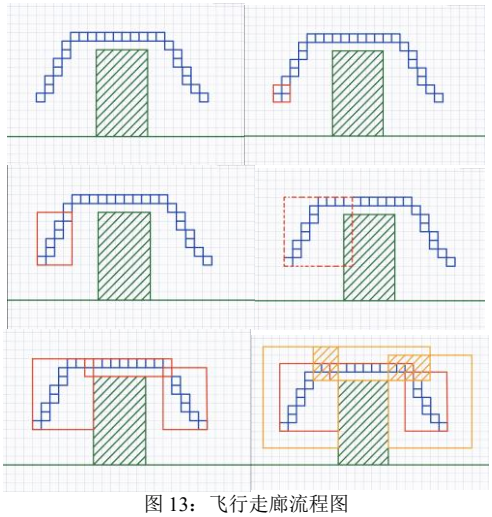

图 13：飞行走廊流程图

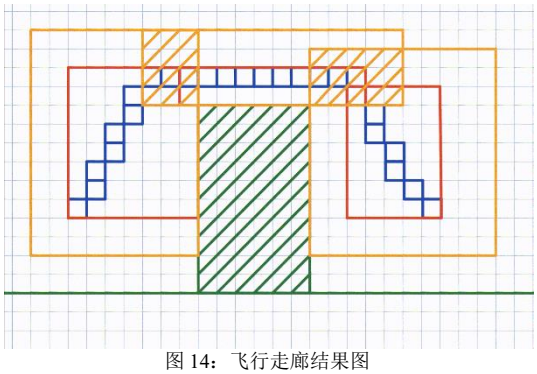

图 14：飞行走廊结果图

蓝框表示搜索得到的路径点，绿框表示障碍物
红框表示初态飞行走廊，橙框表示最终的飞行走廊即膨胀后

伪代码如下：

Algorithm 7: Flight Corridor Generation

```
Begin
    keypoint = startnode
    While keypoint != endnode
        nextpoint = keypoint.next
        While obstaclefree(keypoint,next)
            nextpoint = nextpoint.next
        corridor.add(keypoint,nextpoint)
        keypoint = nextpoint
    expand(corridor)
End
```

仿真效果如图 15 所示：

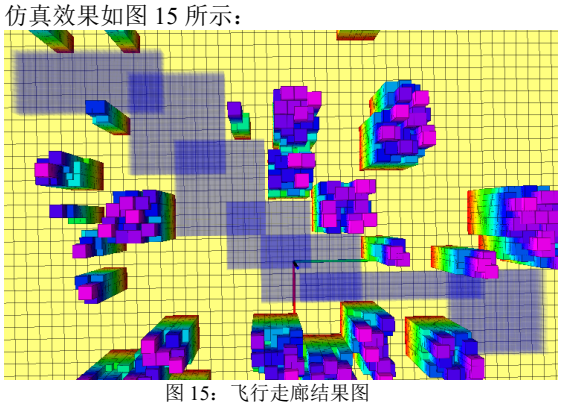

图 15：飞行走廊结果图

由我们的方法生成的走廊效率很高，典型值在 1~2ms，显著快于论文[2]中的方法，但最优性略逊。

3.2.2 贝塞尔曲线

贝塞尔曲线形式如下：

$$B_j(t) = \sum_{i=0}^{n} c_j^i b_n^i(t) \tag{12}$$

$$b_n^i(t) = \binom{n}{i} t^i (1-t)^{n-i} \tag{13}$$

为了使得贝塞尔曲线的每一段轨迹的时间被放缩到（0，1）之间，做如下改进：

$$B_j(t) = \sum_{i=0}^{n} c_j^i b_n^i(t) \tag{14}$$

$$b_n^i(t) = \binom{n}{i} \left(\frac{t}{T}\right)^i \left(1-\frac{t}{T}\right)^{n-i} \tag{15}$$

与之前 Minimum-Snap 轨迹生成相似，我们的优化目标仍然是 Snap。但是与 Minimum-Snap 最大的不同在于，之前 Minimum-Snap 强制要求轨迹经过中间节点(waypoint)，但是这里，并没有这个要求。这里只要求整条轨迹可以被约束在飞行走廊里面即可，而这，可以轻松借助贝塞尔曲线的性质来实现。因此，整个优化问题可以被如下建模：

$$\mathrm{J} = \sum_{j}^{M} \int_{0}^{T_j} (B_j^4(t))^2 \tag{16}$$

连续性约束：

$$c_{u,j}^n = c_{u,j+1}^0 \ (1) \tag{17}$$

$$\frac{\mathrm{n}}{\mathrm{T}}\left(\mathrm{c}_{u,j}^{\mathrm{n}} - {c^{n-1}}_{u,j}\right) = \frac{\mathrm{n}}{\mathrm{T}}\left(\mathrm{c}_{u,j+1}^{1} - {c^{0}}_{u,j+1}\right) \tag{18}$$

$$\left(c_{u,j}^n - 2c_{u,j}^{n-1} + c_{u,j}^{n-2}\right) = \left(c_{u,j+1}^2 - 2c_{u,j+1}^1 + c_{u,j+1}^0\right) \tag{19}$$

$$\frac{n(n-1)(n-2)}{T^3}\left(c_{u,j}^n - 3c_{u,j}^{n-1} + 3c_{u,j}^{n-2} - c_{u,j}^{n-3}\right) = \frac{n(n-1)(n-2)}{T^3}\left(c_{u,j+1}^3 - 3c_{u,j+1}^2 + 3c_{u,j+1}^1 - c_{u,j+1}^0\right) \tag{20}$$

不等式约束：

$$\beta_{u,j}^- \le c_{u,j}^i \le \beta_{u,j}^+ \tag{21}$$

$$v_m^- \le \frac{n}{T}\left(c_{u,j}^i - {c^{i-1}}_{u,j}\right) \le v_m^+ \tag{22}$$

$$a_m^- \le \frac{n(n-1)}{T^2}\left(c_{u,j}^n - 2c_{u,j}^{n-1} + c_{u,j}^{n-2}\right) \le a_m^+ \tag{23}$$

$$i = 0,1,2 \ldots n, u = x, y, z, j = 1,2, \ldots, M$$

式(17)，(18)，(19)，(20)分别代表在关键点上的位置，速度，加速度，Jerk 的连续性约束。而式(21)代表位置的不等式约束，意为将轨迹约束在实现定义好的飞行走廊内。式（22），（23）代表速度和加速度的不等式约束，因为考虑到实际无人机的运动能力有所限制，因此添加此约束。

轨迹生成仿真效果如图 16 所示：

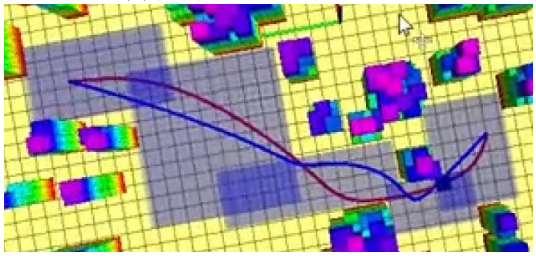

图 16：贝塞尔曲线+飞行走廊的轨迹生成

蓝线表示迭代添加约束点的 Minimum-Snap 轨迹，
红线表示贝塞尔曲线轨迹，蓝框表示飞行走廊

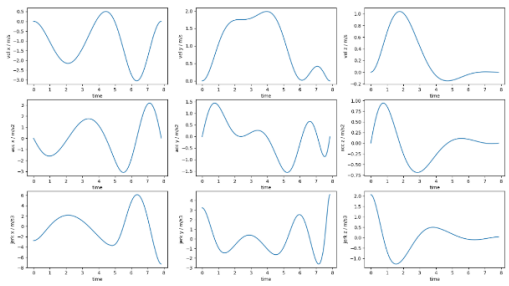

图 17：贝塞尔曲线三个方向上的速度、加速度、Jerk 变化
三行分别表示无人机速度，加速度，加加速度，
三列分别表示 x,y,z 方向

其中飞行器三个方向的速度、加速度、Jerk 都实现了连续变化，且位于约束条件内。

贝塞尔曲线结合飞行走廊相较于 Minimum-Snap 具有以下优势：

1. 信息量大幅度增加，从之前的一系列关键点到目前的一系列有重叠部分的凸几何体。
2. 将一部分的等式约束变成了不等式约束，增加了轨迹的自由度，有利于轨迹生成。
3. 对时间的分配不敏感，
4. 从根本上去保证避障的有效性，不需要去迭代增加约束点。
5. 通过贝塞尔曲线控制点约束，还可以对速度，加速度，Jerk 做约束，更加符合实际。

接着，为了验证轨迹生成的实时性与光滑性，我们将我们的代码在动态避障和无人机跟踪两个场景下面进行测试。

动态避障：我们将地图按 1Hz 的频率随机刷新，测试无人机的飞行性能，实验证明，即使对于动态障碍物，我们的实时轨迹生成也能很好应对。

无人机跟踪：我们将无人机的路径规划终点设置为动态运动的目标的实时位置，以 10Hz 的频率刷新无人机的轨迹，实现跟踪，效果如图 18 所示。

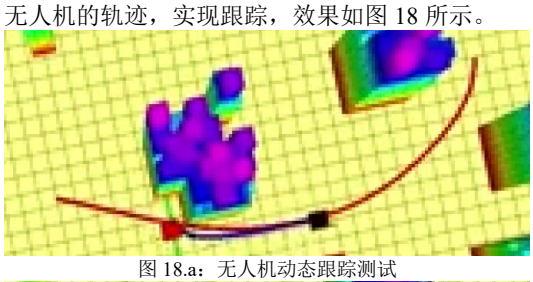

图 18.a：无人机动态跟踪测试

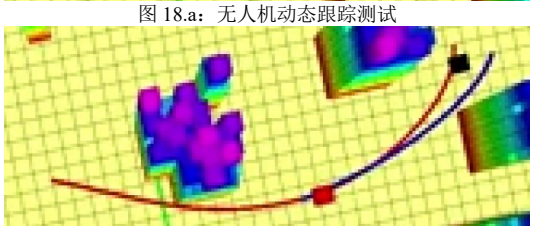

图 18.b：无人机动态跟踪测试

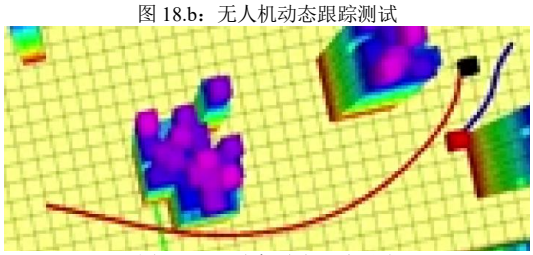

图 18.c：无人机动态跟踪测试

结果显示我们的轨迹生成策略有较好的重规划能力。实际通过多次实验测试，发现 Minimum-Snap 轨迹生成的时间平均值在 2~5ms 之间，而贝塞尔曲线轨迹生成时间约在 10~20ms，虽然比较慢，但是仍然是可以接受的。

结论：

在对无人飞行器基础路径规划的研究中，我们分别从路径规划和轨迹生成两方面入手展开仿真分析：在路径规划中，我们分别对 A*算法中的启发函数进行了性能对比，并测试了 A*算法、JPS 算法、RRT*算法的性能。其中欧式距离和对角线距离作为启发函数都具有最优性，且整体性能上来看，对角线距离更为实用；A*算法和 JPS 算法规划出来的路径都是最优的，从时间效率、内存占用、稳定性上来考虑，JPS 算法的表现更好。

在轨迹生成中，我们分别使用了 Minimum-Snap 和贝塞尔曲线结合飞行走廊的方法来确保轨迹的形状符合动力学约束。Minimum-Snap 的测试运行时间是 2~5ms，但是前者在避障和稳定性上存在问题，为此我们做出了相应的改进，包括数值稳定性优化、插点法避障、迭代法避障，但仍不能达到完美。因此之后我们采用了贝塞尔曲线结合飞行走廊的方法进行轨迹生成，贝塞尔曲线结合飞行走廊的运行时间比 Minimum-Snap 稍微要长，但仍然能满足实时性的要求，它不仅从根本上保证了避障性能，而且更方便对轨迹做出约束，同时由于更加充分利用空间信息，使得轨迹质量明显由于 Minimum-Snap。而且在动态避障测试和无人机跟踪测试中都展现出优秀的性能。在实际测试中，路径规划和贝塞尔曲线轨迹生成的总时间花费典型值为 10~20ms，可以满足实时规划的需求。

References

[1]. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," 2011 IEEE International Conference on Robotics and Automation, Shanghai, 2011, pp. 2520-2525, doi: 10.1109/ICRA.2011.5980409.

[2] F. Gao, L. Wang, B. Zhou, X. Zhou, J. Pan and S. Shen, "Teach-Repeat-Replan: A Complete and Robust System for Aggressive Flight in Complex Environments," in IEEE Transactions on Robotics, doi: 10.1109/TRO.2020.2993215.

[3] M. M. de Almeida and M. Akella, "New numerically stable solutions for minimum-snap quadcopter aggressive maneuvers," 2017 American Control Conference (ACC), Seattle, WA, 2017, pp. 1322-1327, doi: 10.23919/ACC.2017.7963135.

[4] F. Gao, L. Wang, K. Wang, W. Wu, B. Zhou, L. Han, and S. Shen, “Optimal trajectory generation for quadrotor teach-and-repeat,” IEEE Robotics and Automation Letters (RA-L), 2019.

[5] F. Gao and S. Shen, “Online quadrotor trajectory generation and utonomous navigation on point clouds,” in Proc. of the IEEE Intl. Sym.on Safety, Security, and Rescue Robotics (SSRR), lausanne, switzerland,2016, pp. 139–146.

[6] F. Gao, W. Wu, W. Gao, and S. Shen, “Flying on point clouds: Online trajectory generation and autonomous navigation for quadrotors in cluttered environments,” Journal of Field Robotics, 2018

[7] F. Blochliger, M. Fehr, M. Dymczyk, T. Schneider, and R. Siegwart, “Topomap: Topological mapping and navigation based on visual slam maps,” in Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA),2018, pp. 1–9 environments[M]//Robotics Research. Springer International Publishing, 2016: 649-666.

[8] S. Liu et al., "Planning Dynamically Feasible Trajectories for Quadrotors Using Safe Flight Corridors in 3-D Complex Environments," in IEEE Robotics and Automation Letters, vol. 2, no. 3, pp. 1688-1695, July 2017,doi:10.1109/LRA.2017.2663526.

[9] Richter C, Bry A, Roy N. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor