中国研究生创新实践系列大赛
“华为杯”第十六届中国研究生数学建模竞赛

学　　校	华东师范大学
参赛队号	**191026990110**
队员姓名	**1.** 徐鸣睿
	2. 王华杰
	3. 李梅

中国研究生创新实践系列大赛
“华为杯”第十六届中国研究生数学建模竞赛

题　　目　多约束条件下智能飞行器航迹快速规划

摘　　要：

复杂环境下智能飞行器的航迹问题是无人驾驶技术突破的关键所在，要能准确控制航迹，就需要准确定位，而目前定位系统无法精准定位，需要在飞行过程中不断校正定位误差。本文研究智能飞行器在系统定位精度限制下的航迹快速规划问题。

问题一要求能够快速规划飞行器在多约束条件下的飞行航迹，综合考虑飞行距离和误差校正次数尽量少。为了使模型通用性更高，本文使用记忆化深度优先算法建立有向图，然后用 Dijkstra+ 堆优化算法求最短航迹，算法时间复杂度为 $O(V_1 * V_1 * \epsilon + V_2 * log_(V_2))$，其中 V_1 为校正点个数，ϵ 为满足约束条件下最远飞行距离，V_2 为建图后的节点个数。假设飞机可以折线飞行，优先考虑飞行距离然后考虑误差校正次数。在数据集 1 上，建立模型得到一条最短航迹为 103512m 和校正次数为 9 次的最优路径。在数据集 2 上，建立模型得到一条最短航迹为 109336m 和校正次数为 12 次的最优路径。

问题二在问题一基础上，提出飞行器受到结构和控制系统的影响，最小转弯半径为 200m，我们假设飞行器在两点之间飞行时，飞行直线距离大于最小转弯半径，即飞行器能在当前位置沿半径为 200m 的圆转弯，直至能沿该圆的切线方向飞向下一个校正点。在此假设之上，我们使用第一问的算法求出最优的 5 条航迹，将航迹进行修正，选择修正后的最优航迹作为结果。令初始飞行方向为起始点到第一个校正点直线飞行的方向，因此，飞行器航迹修正为沿当前飞行方向以当前校正点为切点的相切圆的圆弧，加上沿相切圆切线方向飞向下一个校正点的航迹。在校正前，在数据集 1 和数据集 2 上得到的最优航迹长度分别为 103512m 和 109336m，经过计算得到修正后的航迹长度为 103652m 和 109534m

问题三在问题一的基础上，加上了概率约束，部分校正点为问题校正点，会有误差校正失败的概率，题目要求出发点成功到达目的地的概率尽可能大。我们在最坏的情况去求最优概率，即假设所有问题校正点均校正失败。我们在解决第一问的算法的基础上改变建有向图的方法。在数据集 1 上，建立模型得到一条概率为 100%，最短航迹为 104823m 和校正次数为 10 次的最优航迹。在数据集 2 上，建立模型得到一条概率为 100%，最短航迹为 161639m 和校正次数为 21 次的最优航迹。

关键字： 多约束条件　通用性　记忆化深度优先搜索算法　Dijkstra+ 堆优化算法　运动轨迹　概率分析

目录

1. 问题重述

复杂环境下飞行器的航迹问题是无人驾驶技术突破的关键所在，需要在满足各种条件下从出发点到目标点选择最优航线，使飞行器消耗最小。要能准确控制航迹，就需要准确定位，而目前定位系统无法精准定位，需要在飞行过程中不断校正定位误差。因此，在飞行过程中对定位误差进行校正是飞行器航迹规划的一项重要任务。

给定飞行区域，出发点为 A 点，目的地为 B 点，航迹约束条件如下：

- 飞行器要实时定位，定位误差分为垂直误差与水平误差，飞行器每飞行 1m，垂直误差和水平误差各增加 δ 个单位。假设当垂直误差和水平误差均小于 θ 个单位时，飞行器能够按照规划路径飞行，并且要求在飞行器到达 B 点时垂直误差和水平误差均小于 θ 个单位；
- 在飞行区域中，存在一些校正点，可用于误差校正。校正点分为垂直校正点和水平校正点，分别只能对垂直误差或水平误差进行校正，校正后误差为 0。最终，飞行器经过若干个校正点到达目标点；
- 飞行器在 A 点时，垂直误差和水平误差都为 0；
- 当飞行器的垂直误差不大于 α_1 个单位，水平误差不大于 α_2 个单位时，才能进行垂直误差校正；
- 当飞行器的垂直误差不大于 β_1 个单位，水平误差不大于 β_2 个单位时，才能进行水平误差校正。

为飞行器建立从 A 点到 B 点的飞行轨迹的一般模型和算法，解决下列问题：

问题一：综合考虑 1) 轨迹长度，2) 校正次数，建立模型和算法优化飞行轨迹；

问题二：由于飞行器在转弯时受结构和控制系统的限制，无法完成即时转弯，假设飞行器最小转弯半径为 200m，设计算法优化飞行轨迹，综合考虑轨迹长度和校正次数，使其尽可能小；

问题三：增加校正点的校正成功几率，假设部分校正点误差校正成功概率为 80%，校正失败后剩余误差为 $min(error, 5)$，其中 $error$ 为校正前误差，不考虑转弯半径的情况下，规划航迹路径，使其成功到达终点的概率最大。

2. 模型假设

- 为了简便运算假设距离为整数；
- 假设飞行器第一次到达校正点为 M，飞行器初始飞行方向为起始点到 M 方向；
- 飞行器在两点之间飞行时，飞行直线距离大于最小转弯半径；
- 假设飞行器最小转弯半径为 200m；
- 在不考虑结构和控制系统限制的条件下，假设飞行器可以折线飞行；

- 假设当垂直误差和水平误差均小于 θ 个单位时，飞行器能够按照规划路径飞行；

3. 符号说明

符号	意义
α_1	垂直校正点对垂直误差的约束
α_2	垂直校正点对水平误差的约束
β_1	水平校正点对垂直误差的约束
β_2	水平校正点对水平误差的约束
θ_1	目的地对垂直误差的约束
θ_2	目的地对水平误差的约束
A	飞行器的航线起始点
B	飞行器的航线目的地
$Point_{vertical}$	垂直校正点
$Point_{horizontal}$	水平校正点
$error$	飞行器飞行过程产生的误差 (单位)
$vertical_{error}$	飞行器飞行过程中的累计垂直误差 (单位)
$horizontal_{error}$	飞行器飞行过程中的累计水平误差 (单位)
$\vec{indirect}$	入射方向，即飞行器向当前校正点飞行方向
$\vec{outdirect}$	出射方向，飞行器离开当前校正点飞行方向
R	最小转弯半径，200m

4. 问题分析

4.1 问题一分析

题目要求飞行器在满足约束条件下，综合考虑航迹长度和校正次数，由于题目没有指定优先级，因此我们以轨迹长度为最高优先级，校正次数为次优先级对问题进行求解。由于本问题不考虑转弯半径的限制，故假设飞机可以折线飞行。我们根据题目给的数据建立有向图，令两个校正点为 P_1 和 P_2，利用题目给的约束条件来判断 P_1 到 P_2 是否可达，令

校正点 P_1 到 P_2 时的误差为 $(vertical_{error}, horizontal_{error})$，如果 P_2 是 $Point_{vertical}$，则需要满足

$$vertical_{error} \leq \alpha_1 \&\& horizontal_{error} \leq \alpha_2 \tag{1}$$

如果 P_2 是 $Point_{horizontal}$，则需要满足

$$vertical_{error} \leq \beta_1 \&\& horizontal_{error} \leq \beta_2 \tag{2}$$

如果 P_2 是目的地，则需要满足

$$vertical_{error} \leq \theta_1 \&\& horizontal_{error} \leq \theta_2 \tag{3}$$

建完有向图之后使用 Dijkstra+ 堆优化最短路径算法求最优航迹。

4.2 问题二分析

由于飞行器在转弯时受结构和控制系统的限制，无法完成即时转弯，约束条件为飞行器最小转弯半径为 200m，即飞行器进行转弯时，飞行轨迹的曲率半径最小为 200m，这就使得第一问中得到的折线航迹不适用，需要对航迹进行修正。在实际情况中，可能会出现问题一中的最优航迹并不是问题二中最优航迹。为了使模型更具有通用性，我们可以利用解决问题一的算法得到最优的 5 条航迹，然后进行修正，对比修正后的结果，将得到的最优的航迹作为第二问的结果。修正后的航迹是沿当前飞行方向并以当前校正点为切点作相切圆，飞行器沿圆弧飞行一段距离后，沿相切圆切线方向飞向下一个校正点。优先考虑航迹长度，然后考虑校正次数，选取最优航迹作为问题二的结果。

4.3 问题三分析

问题三在问题一的基础上，加上了概率约束，部分校正点为问题校正点，这些问题校正点有 80% 的校正成功概率，该问题要求飞行器能成功到达终点的概率尽可能大，我们对校正点做最差假设，假设问题校正点全部校正失败，所以我们将从问题校正点起飞的误差设为 5。如果此假设条件下有航迹能够到达目的地，那么概率就是 100%。如果没有可行解，我们进行节点松弛，将其中一个问题节点的概率调为 80%，即这个节点校正成功，直到有路径能到达终点，即可求得最后的最优概率。

5. 问题建模与求解

接下来的部分，将对之前三个问题分别进行建模并求解。

5.1 问题一的建模与求解

对于问题一，我们先根据数据集用记忆化深度优先搜索算法建立有向图，然后使用 Dijkstra+ 堆优化计算最优航迹，因为记忆化优先深度搜索在最坏条件下的复杂度是

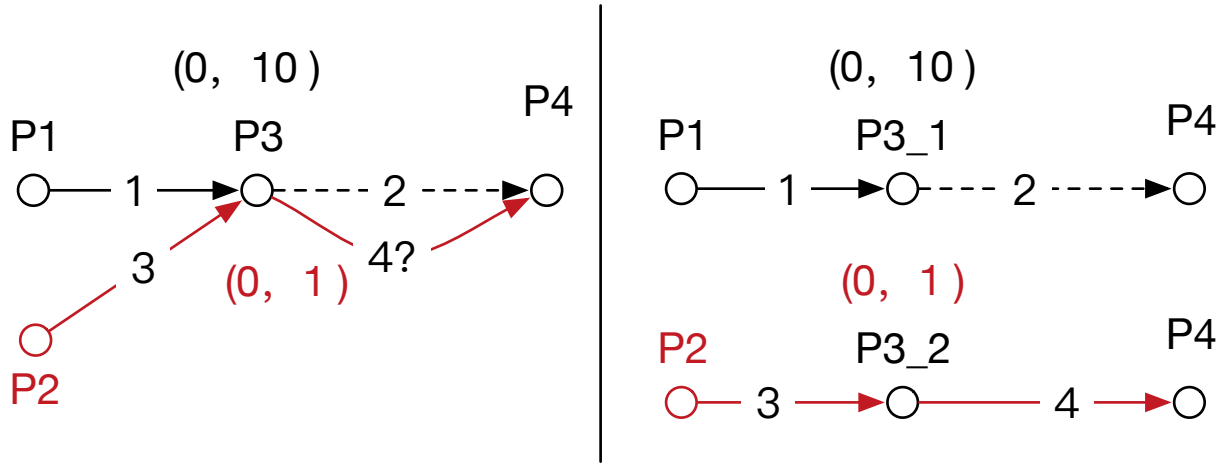

图 1　左部分：记忆化深度优先不根据不同误差拆分点。右部分：记忆化深度优先根据不同误差拆分不同点

$O(V_1 * V_1 * \epsilon)$，其中 V_1 为校正点个数，ϵ 为满足约束条件下最远飞行距离，Dijkstra+堆优化的复杂度为 $O(V_2 * log_(V_2))$，V_2 为建图后的节点个数。所以总的算法复杂度为 $O(V_1 * V_1 * \epsilon + V_2 * log_(V_2))$。

5.1.1 模型建立

假设当前校正点为 $Point_{horizontal}$，则从 $Point_{horizontal}$ 出发的垂直误差为 0。同理，$Point_{vertical}$ 出发的水平误差为 0 。因此从每个校正点开始，我们只要计算当前校正点不能校正的误差类型的误差，将另一个误差置 0，然后判断当前校正点能否到达的其余校正点进行有向图生成。令当前校正点为 $Point_{vertical}$，则从 $Point_{vertical}$ 出发到下一点的误差为 $(error,error + horizontal_{error})$，$horizontal_{error}$ 为 $Point_{vertical}$ 的水平误差，则当 $Point_{vertical}$ 的下一个校正点是 $Point_{vertical}$ 时，根据公式 (1) 必须满足 $error \leq \alpha_1$ 和 $error + horizontal_{error} \leq \alpha_2$，我们才添加 $Point_{vertical}$ 到 $Point_{vertical}$ 的一条有向边。同理，当 $Point_{vertical}$ 下一个校正点为 $Point_{horizontal}$，根据公式 (2) 必须 $error \leq \beta_1$ 和 $error + horizontal_{error} \leq \beta_2$，我们才添加 $Point_{vertical}$ 到 $Point_{horizontal}$ 的一条有向边。当下一个点为 $Point_{end}$ 时，我们根据公式 (3)$error \leq \theta_1$ 和 $error + horizontal_{error} \leq \theta_2$，我们才添加 $Point_{vertical}$ 到 $Point_{end}$ 的一条有向边。在实际算法运算过程中，我们对算法进行了剪枝，即对当前访问过的校正点设置 $visit$ 标志，使下次访问到这个校正点的时候直接返回进行剪枝。但是不同的误差到达固定的校正点会出现不唯一的图，如图1左部分所示，假如深搜过程的路线为 1→2→3→4 由 P1 到达 P3 时的水平误差为 10, 使 P3 无法到达 P4，且此时我们置 P3 $visit$, 等 P2 到达 P3 时误差为 (0,1) 且能到 P4，但是由于 P3 访问过，所以返回上层，将导致无法建边 P3→P4。对于上述问题，我们将 P3 根据不同的误差映射到两个点，如图1右部分所示，将 P3 拆成 P3_1 和 P3_2 重新建图。记忆化深度优先搜索

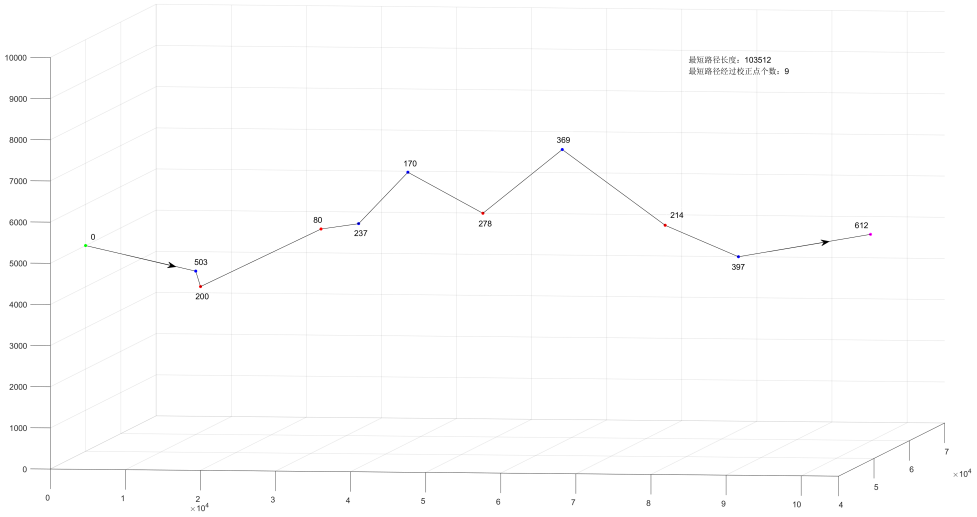

图 2　问题一数据集 1 航迹规划路径图

伪代码如算法 1 所示，具体实现 Python 代码见附录一，模型代码见附件。

Dijkstra+ 堆优化：Dijkstra 算法是实现单源最短路径非常高效的算法，对于节点个数多的时候，加入堆优化可以明显降低复杂度。所以我们用 Dijkstra+ 堆优化算法来求记忆化优先搜索生成的图的最短路径。

Dijkstra 算法的具体实现过程如下：

1）令 S 为已知最短路径的节点的集合，V 为顶点节点，$S-V=T$ 为未知最短路径的节点集合

2）每次从 T 中获取离起始节点最近的节点 P，更新 P 的邻接节点且在 T 中，并将 P 加入到 S 中。

3) 一直重复 2)，直到 T 为空。

其中优先队列用于实现获取 P，使得每次获得节点复杂度为 $log_2(len(T))$

每个节点需要更新邻接节点，因此最终复杂度为 $len(V)*log_2(len(T))$，即 $O(V_2*log_(V_2))$

我们 Dijkstra+ 堆优化是用 C++ 优先队列实现的，具体实现伪代码如算法 2 所示，具体实现 C++ 代码见附录二，模型代码见附件。

5.1.2 问题求解

根据建模，输入数据集 1 的数据后得到最短路径为 103512m，因为需要考虑节点个数，所以我们取所有最短路径，发现最短路径只有一条，所以节点个数最少为 11 个，具体节点坐标如下：[0, 503, 200, 80, 237, 170, 278, 369, 214, 397, 612]，具体线路见图2，具体航迹规划表见表1。

根据建模，输入数据集 2 的数据后得到最短路径为 109336m，因为需要考虑节点个

数，所以我们取所有最短路径，发现最短路径只有一条，所以节点个数最少为 14 个，具体节点坐标如下：[0, 163, 114, 8, 309, 305, 123, 45, 160, 92, 93, 61, 292, 326] 具体线路见图3，具体航迹规划表见表2。

5.2 问题二的建模与求解

5.2.1 模型建立

对于问题二，需要考虑飞行器的转弯半径。根据之前的假设，初始飞行方向为 X 点到第一个校正点直线飞行的方向，并且飞行器在两点之间飞行，飞行直线距离大于最小转弯半径。因此，飞行器最优飞行航迹可以修正为沿当前入射方向 $in\vec{direct}$ 并以当前校正点为切点作相切圆，

表 1　问题一数据集 1 航迹规划结果表

校正点编号	校正前垂直误差	校正前水平误差	校正点类型
0	0	0	出发点 A
503	13.387	13.387	11
200	0.865	14.252	01
80	16.614	15.749	01
237	21.241	4.627	11
170	7.689	12.316	11
278	10.457	22.773	01
369	21.893	11.436	11
214	13.313	24.749	01
397	22.33	9.017	11
612	16.972	25.989	终点 B

飞行器沿圆弧飞行一段距离后，沿相切圆的切线方向飞向下一个节点，离开当前节点时，出射方向 $out\vec{direct}$ 为飞离相切圆处位置到下一个节点的直线方向，如图 4所示。其中，M 点为当前节点，X 点为前一节点，Y 点为下一个节点，则入射方向 $in\vec{direct} = X\vec{M}$，

出射方向 $\vec{indirect} = \vec{NY}$，圆 O 半径为 200m。在这种情况下，可以使飞行器满足最小转弯半径的前提下，从 M 点飞行到 Y 点。当离开当前节点 M 后，Y 点更新为当前节点，入射方向更新为 $\vec{indirect} = \vec{NY}$。

算法 1 Dijkstra+ 堆优化

*** 输入**: $start_{point}$ 起始节点，$edges$ 每个节点的边集合
*** 输出**: $start_{point}$ 到所有点的最短距离和每个节点最短路径的前驱节点
令 priority_queue 为优先队列
将 $start_{point}$ 加入到 priority_queue
while priority_queue 非空 **do**
    $current_point$ ← priority_queue 队头元素
    删除 priority_queue 队头元素
    if $current_{point}$ 访问过 **then**
        continue
    end if
    将 $current_{point}$ 访问标志置为 True
    $edge$ ← $edges$ 中 $current_{point}$ 对应的边集
    while $edge$ 非空 **do**
        $link_{edge}$ ← $edge$ 当前连接边
        将 $edge$ 当前连接边从集合删除
        if $link_{edge}$ 的长度加上 $start_{point} \rightarrow current_{point}$ 的长度 <$start_{point} \rightarrow current_{point}$ 的长度 **then**
            更新 $start_{point} \rightarrow current_{point}$ 为 $link_{edge}$ 的长度加上 $start_{point} \rightarrow current_{point}$ 的长度并且加入到 priority_queue 中
            更新 $link_{point}$ 最短路径的前驱节点为 $current_{point}$
        end if
    end while
end while
Return $start_{point}$ 到所有点的最短距离和每个节点最短路径的前驱节点

算法 2 is_Link()

*** 输入**: now_{id}(当前校正点 id),$vertical_{error}$(上个节点到当前节点后的垂直误差),$horizontal_{error}$(上个节点到当前节点后的水平误差)

*** 输出**: 上个节点是否可达当前节点。

if 当前节点是终点 **then**

  if $vertical_{error} <= \theta$ and $horizontal_{error} <= \theta$ **then**

    Return TRUE,$last_{id}$

  end if

  Return False,-1

end if

if 当前节点是校正节点 **then**

  if $vertical_{error} <= \alpha_1\ if\ T_v\ else\ \beta_1$ and $horizontal_{error} <= \alpha_1\ if\ T_v\ else\ \beta_1$ **then**

    Return TRUE,-1

  end if

  Return False,-1

end if

if T_v then $current_{error} \leftarrow horizontal_{error}$ **else** $current_{error} \leftarrow vertical_{error}$

//将当前 ID 和 error 映射到新的 ID_$error_{id}$ 来建图，防止建图错误

//假如当前 ID 的误差计算过，则直接返回，防止做多余的计算

if $error_{id}$ 访问过 **then**

  Return TRUE,$error_{id}$

end if

$i \leftarrow 1$

$n \leftarrow$ 校正点个数

while $i < n$ **do**

  计算当前节点到 i 节点的 $error$ 来跟下个节点的误差

  if T_v **then**

    $linked$,$next_{id}$=is_Link(i,$error$,$current_{error} + error$)

  else

    $linked$,$next_{id}$=is_Link(i,$error + current_{error}$,$error$)

  end if

  if $linked$ **then**

    将 $error_{id}$ 到 $next_{id}$ 的边建立起来

  end if

end while

Return TRUE,$error_{id}$

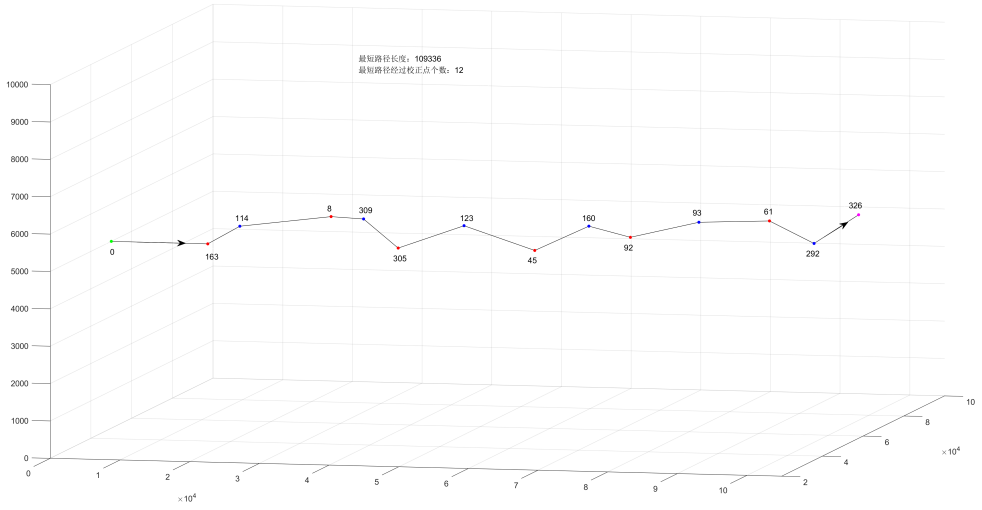

图 3　问题一数据集 2 航迹规划路径图

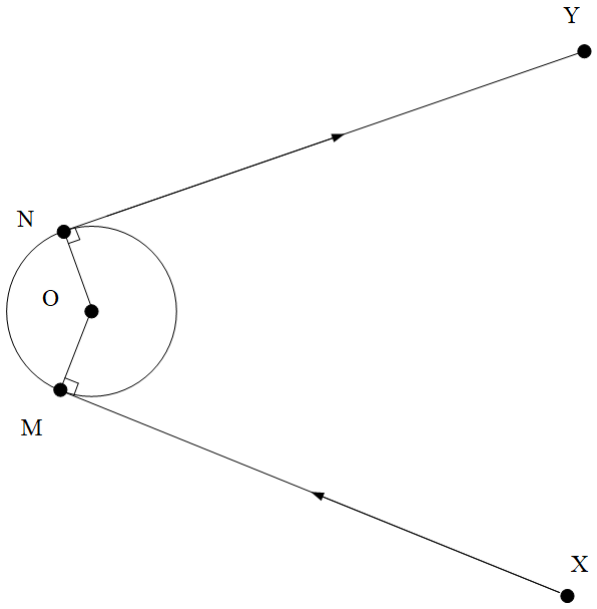

图 4　转弯半径图

在已知三个节点空间坐标的情况下，可以求得飞行器以最小转弯半径进行飞行时的圆心坐标，从而求得转弯结束时开始直线飞行的位置坐标，最终得到飞行轨迹。在列出公式之前，先给出已知条件：如图 4，其中，当前节点 M 坐标为 (x_0, y_0, z_0)，目标点 Y 坐标为 (x_1, y_1, z_1)，入射方向 $\vec{XM} = (a, b, c)$。需要求出圆心 O 的坐标 (x, y, z) 以及停止转弯飞行位置 N 的坐标 (x_2, y_2, z_2)。

先求三个空间节点所形成空间平面的法向量 $\vec{t} = (m, n, k)$ 进行求解，计算公式如下：

表 2　问题一数据集 2 航迹规划结果表

校正点编号	校正前垂直误差	校正前水平误差	校正点类型
0	0	0	出发点 A
163	13.287	13.287	01
114	18.621	5.334	11
8	13.921	19.255	01
309	19.445	5.524	11
305	5.968	11.492	01
123	15.172	9.204	11
45	10.006	19.21	01
160	17.491	7.485	11
92	5.776	13.261	01
93	15.26	9.484	11
61	9.834	19.318	01
292	16.387	6.553	11
326	6.96	13.513	终点 B

$$\vec{t} = (a, b, c) \times (x_1 - x_0, y_1 - y_0, z_1 - z_0). \tag{4}$$

再对圆心坐标进行求解，计算公式如下：

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = R^2; \tag{5}$$

$$(x - x_0, y - y_0, z - z_0) \cdot (a, b, c) = 0; \tag{6}$$

$$(x - x_0, y - y_0, z - z_0) \cdot (m, n, k) = 0. \tag{7}$$

公式 (5) 表示圆心与 M 点距离为 $R = 200$，公式 (6) 表示向量 $\vec{MO}$ 垂直于向量 $\vec{XM}$，公式 (7) 表示向量 $\vec{MO}$ 垂直于平面的法向量 $\vec{t}$。计算出来的结果有两个，因为过直线上一点作已知半径的相切圆有两种情况，在直线两边可以各做一个，但是只有一个圆心坐标是我们所需要的，如图 5 所示，只有与目标点在同侧的圆心坐标为我们所需要的结果，因为这样得到的飞行轨迹更短。只需要将得到的圆心坐标 (x, y, z) 和 (x', y', z') 分别求与目标点 B 的距离，距离短的圆心坐标即为所需结果。

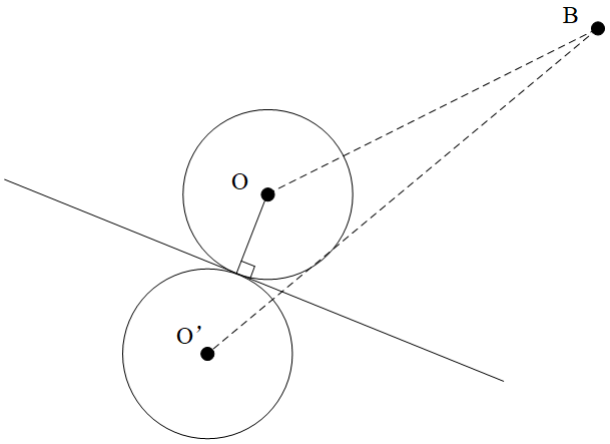

图 5　求解圆心坐标

在得到圆心 O 坐标 (x, y, z) 之后，再对点 N 坐标 (x_2, y_2, z_2) 进行求解，计算公式如下：

$$(x_2 - x_0)^2 + (y_2 - y_0)^2 + (z_2 - z_0)^2 = R^2; \tag{8}$$

$$(x_2 - x_0, y_2 - y_0, z_2 - z_0) \cdot (x_2 - x_1, y_2 - y_1, z_2 - z_1) = 0; \tag{9}$$

$$(x_2 - x_0, y_2 - y_0, z_2 - z_0) \cdot (m, n, k) = 0. \tag{10}$$

如图 4所示，公式 (8) 表示圆心 O 与 N 点距离为 $R = 200$，公式 (9) 表示向量 $\vec{ON}$ 垂直于向量 $\vec{YN}$，公式 (10) 表示向量 $\vec{ON}$ 垂直于平面的法向量 $\vec{t}$。同样可以得到两个结果，因为过圆外一点可以做两条圆的切线，所以切点有两个，取向量 $\vec{NY}$ 与入射方向向量 $\vec{XM}$ 夹角大的切点作为所求点，因为夹角大的切点是从入射点开始沿圆弧飞行先到达的切点。

5.2.2 问题求解

根据问题一的算法进行如下改进得到的最优的 5 个航迹路线，具体代码见附录三。

1）以 Dijkstra 的起点和终点为算法起始点分别计算起点和终点到各个校正点的最短距离。令 S_E 为已经访问过的边集合。

2）遍历所有不在 S_E 中的边，令边对应的节点为 u 和 v，如果起点到 u 的最短距离 $+u \rightarrow v$ 的距离 $+v$ 到终点的最短距离 $\leq$ 当前最短路径, 更新最优边 E_select。

3）将 2）中取得最短距离的最优边 E_select 加入到 S_E 中。

4）根据 S_E 集合得到最优航线。

将校正点处的折线按上述方法进行修正，可以得到修正后的飞行航迹图，先根据飞行轨迹长度进行比较，再根据校正次数进行比较，取最优的航迹为问题二的结果。数据集 1 和数据集 2 得到的航迹规划结果分别见表 3、 4。

表 3　问题二数据集 1 航迹规划结果表

校正点编号	校正前垂直误差	校正前水平误差	校正点类型
0	0	0	出发点 A
503	13.387	13.387	11
200	0.865	14.252	01
80	16.614	15.749	01
237	21.241	4.627	11
170	7.689	12.316	11
278	10.457	22.773	01
369	21.893	11.436	11
214	13.313	24.749	01
397	22.33	9.017	11
612	16.972	25.989	终点 B

得到修正后的飞行航迹图后，使用 matlab 进行绘图，数据集 1 和数据集 2 的飞行轨迹图分别如图 6、 7所示。

5.3 问题三的建模与求解

5.3.1 模型建立

对于问题三，因为校正后得到的剩余误差为 $min(error, 5)$，其中 $error$ 为校正前误差，我们按最差情况进行航迹规划，即到达问题校正点处直接置校正失败且误差直接置为 5，如果按这种情况能够飞到目的地，那么此条航迹成功到达目的地的概率

表 4　问题二数据集 2 航迹规划结果表

校正点编号	校正前垂直误差	校正前水平误差	校正点类型
0	0	0	出发点 A
163	13.287	13.287	01
114	18.621	5.334	11
8	13.921	19.255	01
309	19.445	5.524	11
305	5.968	11.492	01
123	15.172	9.204	11
45	10.006	19.21	01
160	17.491	7.485	11
92	5.776	13.261	01
93	15.26	9.484	11
61	9.834	19.318	01
292	16.387	6.553	11
326	6.96	13.513	终点 B

为 100% 的。因此，假如当前校正点为 $Point_{vertical}$ 且校正成功或者失败时候的误差为 $(vertical_{error}, horizontal_{error})$，我们令 $Point_{vertical}$ 出发时的误差为 $(5, horizontalerror)$，对于 $Ponit_{horizontal}$ 也进行类似的操作。对于模型实现，我们仅需要修改问题一算法2的第 26 行和第 28 行的 $error$，加 5 即可求得最差情况下的有向图，再运行跟 Dijkstra+ 堆优化判断得是否可到达目的地，如果有航迹能到达目的地，那么此航迹成功到达目的地的概率就是 100%，如果没有可行解，我们进行校正点松弛操作，将其中一个问题校正点调整为肯定校正成功，如果经过此校正点的航迹能够到达终点，那么概率就是 80%。重复执行松弛操作直到有航迹能到达目的地，即可求得最优航迹。

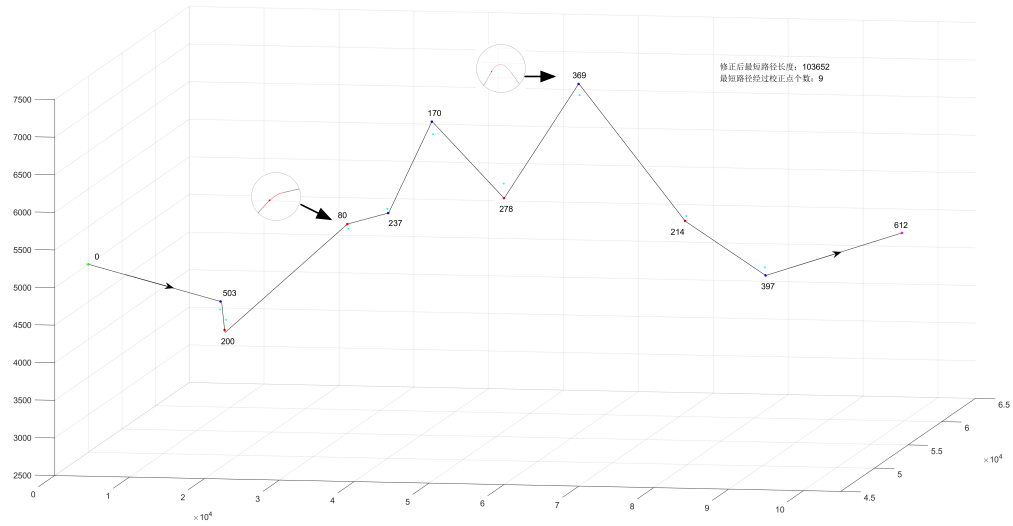

图 6　问题二数据集 1 航迹规划路径图

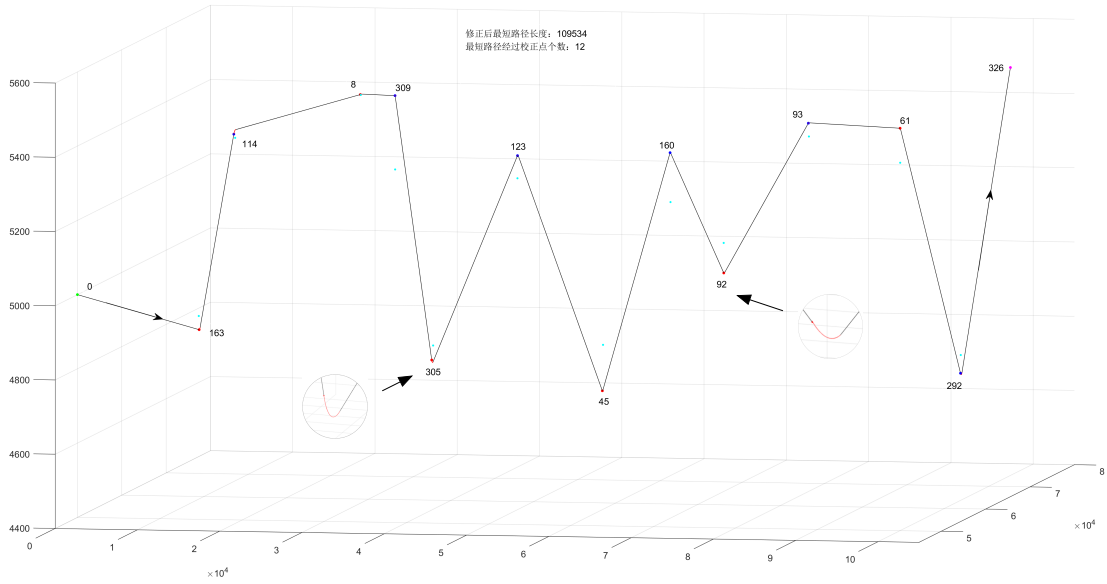

图 7　问题二数据集 2 航迹规划路径图

5.3.2 问题求解

根据建模，输入数据集 1 后，我们发现在将所有问题校正点设为最差情况存在航迹能够到达目的地，则该航迹能到达目的地的概率为 100%, 在航迹能到达目的地的情况下，得到最短航迹为 104823m，因为需要考虑校正点个数，所以我们记录所有最短航迹，发现最短航迹只有一条，该航迹的校正点个数为 10 个，具体节点坐标如下：[0, 503, 69, 506, 371, 183, 194, 450, 113, 485, 248, 612] 具体航迹见图8，具体航迹规划表见表5。

表 5　问题三数据集 1 航迹规划结果表

校正点编号	校正前垂直误差	校正前水平误差	校正点类型
0	0	0	出发点 A
503	13.387	13.387	12
69	13.807	22.194	02
506	21.675	12.868	12
371	15.615	23.483	02
183	22.653	7.038	12
194	13.611	20.649	02
450	19.588	5.977	12
113	6.507	12.484	02
485	13.787	7.28	12
248	4.22	11.5	02
612	23.733	19.513	终点 B

输入数据集 2 后，我们发现在将所有问题校正点设为最差情况存在航迹能够到达目的地，则该航迹能到达目的地的概率为 100%, 在航迹能到达目的地的情况下，得到最短航迹为 161639m，因为需要考虑校正点个数，所以我们记录所有最短航迹，发现最短航迹只有一条，此航迹对应的校正点个数为 21，具体节点坐标如下：[0, 169, 322, 270, 89, 236, 132, 53, 112, 268, 250, 243, 73, 249, 274, 12, 216, 16, 282, 141, 291, 161, 326]

具体航迹见图9，具体航迹规划表见表6。

表 6　问题三数据集 2 航迹规划结果表

校正点编号	校正前垂直误差	校正前水平误差	校正点类型
0	0	0	出发点 A
169	9.27	9.27	02
322	13.418	4.148	12
270	11.34	15.488	02
89	18.89	7.55	12
236	10.127	17.677	02
132	19.831	9.704	12
53	10.259	19.963	02
112	15.461	5.202	12
268	2.157	7.359	02
250	11.763	9.606	12
243	6.958	16.564	02
73	10.5	3.542	12
249	12.846	16.388	02
274	15.686	2.84	12
12	6.436	9.276	02
216	14.238	7.802	12
16	4.216	12.018	02
282	11.655	7.439	12
141	8.1	15.539	02
291	13.585	5.485	12
161	6.466	11.951	02
326	16.612	15.146	终点 B

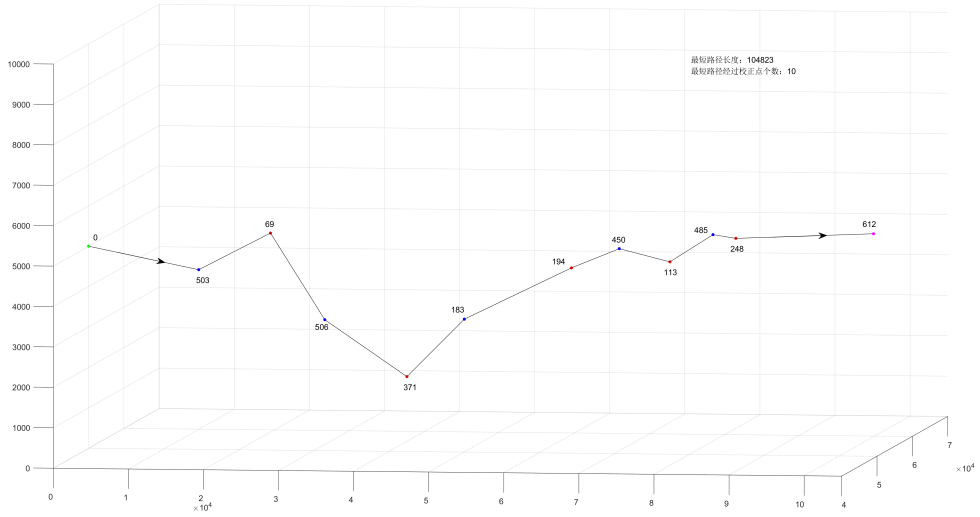

图 8　问题三数据集 1 航迹规划路径图

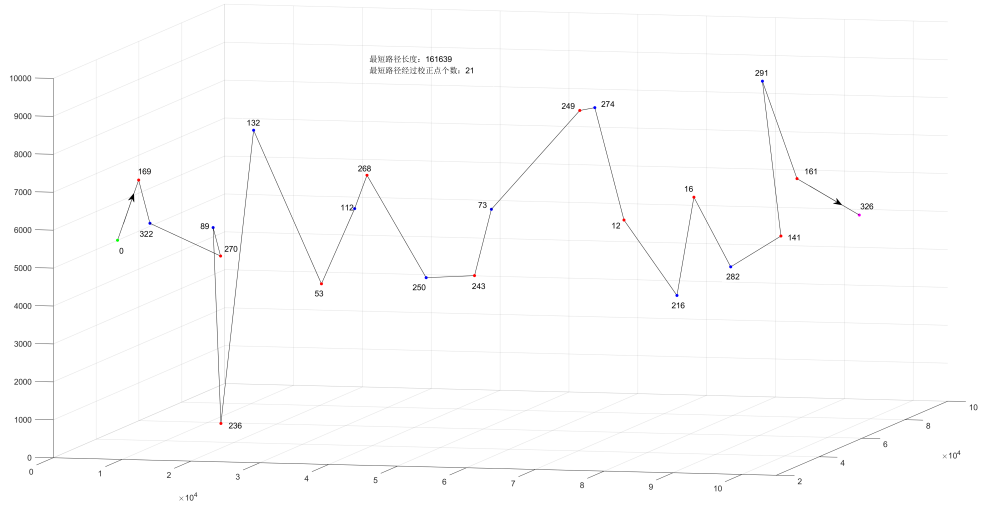

图 9　问题三数据集 2 航迹规划路径图

6. 模型的评价与推广

6.1 模型的评价

6.1.1 模型的优点

- 对于问题一，我们提出了记忆化深度优先搜索算法建有向图，然后利用 Dijkstra+ 堆优化算法求单源最短路径，保证了结果的正确性，且记忆化搜索和堆优化大大降低了模型的复杂度。
- 对于问题一，我们的算法不是针对特定问题而设计的，具有一定的通用性。

- 对于问题二，在求解最优 5 条航迹中，只需分别求得起始点和目的地到各个校正点的最短距离，然后以线性复杂度求任意条最优路径，降低了计算航迹的复杂度。

6.1.2 模型的缺点

- 因为模型假设两点间的距离为整数，会产生一定的误差。
- 因为问题二依赖于问题一的最优 5 条航迹，无法求出新的航迹，具有一定的局限性，可能无法求得最优解。

6.2 模型的推广

该算法复杂度比较低，不仅能运用到本文的多约束调下智能飞行器航迹的快速规划，而且能运用到多个领域，如：汽车无人驾驶，机器人导航系统。

参考文献

[1] 李士波, 孙秀霞, 李海军. 多约束条件下的飞行器航迹规划算法 [J]. 电光与控制, 2007, 14(2):34-37.

[2] Jack Little，Cleve Moler，matlab 官网，https://www.mathworks.com/，2019-9-22

[3] Wenzel Jakob，pybind11 官网，https://pybind11.readthedocs.io/en/stable/，2019-9-21

A 附录

1.1 附录一. 记忆化深度优先算法 Python 代码

```
if now_id==self.len-1:
   if (vertical_val >self.thet or horizontal_val > self.thet):
      return -1, -1
   else:
      if self.vis[now_id][0] == 1:
         return 1, 0
      self.vis[now_id][0]=1
      self.point_num += 1
      self.val2id[now_id][0] = self.s_id
      self.id2val[self.s_id]=now_id
      self.e_id=self.s_id
      self.s_id += 1
      return 1,0
```

```
if self.allow_destination(vertical_val,horizontal_val,now_id) ^ 1:
   return -1,self.inf
val = vertical_val
if self.csv_data[now_id][4] == 1:
   val = horizontal_val
if self.vis[now_id][val] == 1:
   return 1,val
self.vis[now_id][val] = 1
self.point_num+=1
self.val2id[now_id][val]=self.s_id
self.id2val[self.s_id] = now_id
self.s_id+=1
add=0
if self.prob and self.csv_data[now_id][5]==1:
   add=5000

for i in range(1,self.len):
   if i == now_id :
      continue
   loss=(self.dist_data[now_id][i]*self.delta)
   if self.csv_data[now_id][4] == 1:
        # tmp=min_dist
        is_ok,_val=self.get_ans(now_id,i,loss+add,val+loss)
        if is_ok!=-1:

           u = self.val2id[now_id][val]
           v = self.val2id[i][_val]


           if (v in self.is_add[u])==0:
              self.is_add[u][v]=1
              self.e_num+=1
              self.dijkstra.add_edge(u,v,self.dist_data[now_id][i])
   elif self.csv_data[now_id][4]==0:
        # tmp=min_dist
        is_ok,_val=self.get_ans(now_id,i,val+loss,loss+add)
        if is_ok != -1:
           u = self.val2id[now_id][val]
           v = self.val2id[i][_val]
```

```
            # print(u,v)
            if (v in self.is_add[u])==0:
                self.is_add[u][v] =1
                self.e_num+=1
                self.dijkstra.add_edge(u,v,self.dist_data[now_id][i])

return 1,val
```

1.2 附录二.Dijkstra 算法 C++ 代码

```
  void dijkstra(int st,int n){
     for(int i=1; i<=n; i++){
        vis[i] = 0;
        dis[i] = inf;
     }
     dis[st] = 0;
     priority_queue<Node> Q;
     Q.push(Node(st, 0));
     Node nd;

     while(!Q.empty()){
        nd = Q.top(); Q.pop();
        if(vis[nd.id]) continue;
        vis[nd.id] = true;
        for(int i=0; i<V[nd.id].size(); i++){
           int j = V[nd.id][i].first;
           int k = V[nd.id][i].second;
           if(nd.d + k <dis[j] && !vis[j]){
                 dis[j] = nd.d + k;
                 route[j]=nd.id;
                 Q.push(Node(j, dis[j]));
           }
        }
     }

  }
```

1.3 附录三.Dijkstra 算法次优路径计算 C++ 代码

```
void get_next_shortest_path(int n){
   int MIN=1000000000;
   int au=-1;
   int av=-1;
   for(int _id=1;_id<=n;_id++){
      for(int i=0; i<V[_id].size(); i++){
  int j = V[_id][i].first;
  int k = V[_id][i].second;
         bool used=false;
         for(int up=0;up<used_edge.size();up++){
            int u=used_edge[up].first;
            int v=used_edge[up].second;
            if(u==_id&&v==j){
               used=true;
               break;
            }
         }
         if(used){
            continue;
         }
         if(dis[_id]+dis1[j]+k-current_min<=MIN){
            au=_id;
            av=j;
            MIN=dis[_id]+dis1[j]+k-current_min;
         }
      }
   }
   current_min=MIN+current_min;
   used_edge.push_back(make_pair(au,av));
   for(int i=1;i<=n;i++){
      route1[i]=route[i];
   }
   route1[av]=au;


}
```