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	中国科学技术大学
参赛队号	19103580043
队员姓名	1.付冠琪
	2.张博
	3.王星起

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

智能飞行器的轨迹优化是飞行器总体设计过程中的重要环节之一，是完成飞行任务的重要保证，也是实现机动飞行的必要条件。飞行器轨迹优化主要是根据具体的飞行目标，并考虑飞行环境中存在的各种约束条件，通过设计控制变量，得到满足各种约束并且性能函数最优的轨迹。由于飞行器具有较强非线性动力学特性以及快速变化的飞行环境带来的多约束，对应的轨迹优化问题变得十分复杂。本文提出了 A*寻优算法模型和启发式算法，针对一个具体的飞行器飞行轨迹案例进行评估，并给出在不同优化目标下的飞行器飞行轨迹和校正点方案。

对于问题一，我们针对飞行校正点的可能组合设计了 0-1 规划模型，最小化飞行器飞行路径长度，并在此基础上最小化飞行器经过的校正点数量。我们使用 python 实现的 A^*算法来求解该模型。数据集一经过 A*算法得到校正次数 12 次，航迹长度为 114818.94 米的初始解。用此初始解经过模拟退火算法得到校正次数为 9 次，航迹长度为 110527.68 米的解。数据集二使用 A*生成校正次数 15 次，行及长度 132192.36 米的初始解。经过模拟退火产生校正次数 9 次，行及长度 117971.67 米的解。

对于问题二，我们在 A^*的基础上增加模拟退火算法，该算法有两点要求：1、初始解的产生策略必须可以产生大量的随机的有效解。2、新解的产生策略必须保证可以有一定量的有效解。用模拟退火算法求解该问题有两个目标：减少校正次数和减少飞行距离。在该问题中由于考虑了飞行器的最小转弯半径，因此，我们要设计飞行器在校正点的转弯飞行模型，该模型由数学几何知识推导，并且做出了模型假设，并用 python 程序实现。数据集一得到校正次数为 9 次，航迹长度为 115609.49 米的解。数据集二得到校正次数为 8 次，航迹长度为 119583.42 米的解。

对于问题三，由于是在第一问的基础上改变某些校正点的校正概率，因此，第一问的解题思路不再适用，我们必须重新考虑飞行器飞行到终点的概率问题，在模拟退火的基础上，我们增加校正点的优先级和校正成功的概率，让飞行器优先经过一定能校正成功的校正点，其次再按问题一的思路解题，该问题也是同时用到 A^*算法和模拟退火算法，用最优策略使飞行器以较大概率飞行到终点的情况下，产生最小飞行距离和经过最少校正点。数据集一得到校正次数为 12 次，航行距离为 120033.42 米的解。数据集二得到校正次数为 12 次，航行距离为 130866.02 米，到达终点概率为 0.367245 的解。

关键字: 轨迹优化 A*算法 模拟退火 最优策略

目录

1. 问题描述

复杂环境下航迹快速规划是智能飞行器控制的一个重要课题。由于系统结构限制，这类飞行器的定位系统无法对自身进行精准定位，一旦定位误差积累到一定程度可能导致任务失败。因此，在飞行过程中对定位误差进行校正是智能飞行器航迹规划中一项重要任务。本题目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。

假设飞行器的飞行区域如图 1 所示，出发点为 A 点，目的地为 B 点。其航迹约束如下：

1) 飞行器在空间飞行过程中需要实时定位，其定位误差包括垂直误差和水平误差。飞行器每飞行 1m，垂直误差和水平误差将各增加 δ 个专用单位，，以下简称单位。到达终点时垂直误差和水平误差均应小于θ个单位，并且为简化问题，假设当垂直误差和水平误差均小于θ个单位时，飞行器仍能够按照规划路径飞行。
2) 飞行器在飞行过程中需要对定位误差进行校正。飞行区域中存在一些安全位置（称之为校正点）可用于误差校正，当飞行器到达校正点即能够根据该位置的误差校正类型进行误差校正。校正垂直和水平误差的位置可根据地形在航迹规划前确定（如图 1 为某条航迹的示意图， 黄色的点为水平误差校正点，蓝色的点为垂直误差校正点，出发点为 A 点，目的地为 B 点，黑色曲线代表一条航迹）。可校正的飞行区域分布位置依赖于地形，无统一规律。若垂直误差、水平误差都能得到及时校正，则飞行器可以按照预定航线飞行，通过若干个校正点进行误差校正后最终到达目的地。

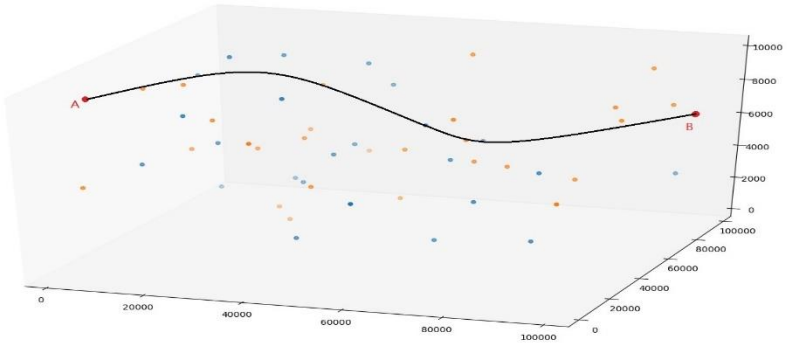

图 1：飞行器航迹规划区域示意图

3) 在出发地 A 点，飞行器的垂直和水平误差均为 0。
4) 飞行器在垂直误差校正点进行垂直误差校正后，其垂直误差将变为 0，水平误差保持不变。
5) 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。
6) 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。
7) 飞行器在转弯时受到结构和控制系统的限制，无法完成即时转弯(飞行器前进方向无法突然改变)，假设飞行器的最小转弯半径为 200m。

请为上述智能飞行器建立从 A 点飞到 B 点的航迹规划一般模型和算法并完成以下问题：

问题 1：分别规划满足条件（1）~（7）时飞行器的航迹，并且综合考虑以下优化目标：（A）航迹长度尽可能小；（B）经过校正区域进行校正的次数尽可能少。讨论算法的有效

性和复杂度。绘出两个数据集的航迹规划路径，并将结果依次填入航迹规划结果表。

问题 2：分别规划满足条件（1）~（8）时飞行器的航迹，并且综合考虑以下优化目标：（A）航迹长度尽可能小；（B）经过校正区域进行校正的次数尽可能少。讨论算法的有效性和复杂度。绘出两个数据集的航迹规划路径，并将结果依次填入航迹规划结果表。

问题 3：飞行器的飞行环境可能随时间动态变化，虽然校正点在飞行前已经确定，但飞行器在部分校正点进行误差校正时存在无法达到理想校正的情况（即将某个误差精确校正为 0），例如天气等不可控因素导致飞行器到达校正点也无法进行理想的误差校正。现假设飞行器在部分校正点能够成功将某个误差校正为 0 的概率是 80%，如果校正失败，则校正后的剩余误差为 min(error,5)个单位（其中 error 为校正前误差，min 为取小函数），并且假设飞行器到达该校正点时即可知道在该点处是否能够校正成功，但不论校正成功与否，均不能改变规划路径。请针对此情况重新规划问题 1 所要求的航迹，并要求成功到达终点的概率尽可能大。绘出两个数据集的航迹规划路径，并将结果依次填入航迹规划结果表。

2. 模型假设

根据题目描述，提出如下假设：

1. 对于飞行器和校验点，皆不考虑其体积对轨迹的影响。
2. 不考虑飞行器的姿态对轨迹的影响。
3. 第一问中，两个优化目标权重相等，第三文中，也为新增加进来的优化目标配给与前两个目标相等大小的权重。
4. 第二问中，飞行器在校正点校正时，假设飞行器以圆弧轨迹接近校正点。

3. 符号定义及数据预处理

3.1 符号定义

表 3-1 模型使用的符号及其含义

符号	含义
	常量
V	垂直校正点集合
H	水平校正点集合
d_{ij}	飞行器从 a_i 到 a_j 的欧式距离
lengthMean	规划路径总长度抽样均值
lengthStddev	规划路径总长度抽样标准差
countMean	规划路径总校正点数抽样均值
countStddev	规划路径总校正点数抽样标准差
posMean	第三问终点可达的概率抽样均值
posStddev	第三问终点可达的概率抽样标准差
	变量
x_i	规划路径是否经过第 i 个校正点
y_{ij}	规划路径中校正点 i，j 是否为连续经过的校正点
$evpre_i$	校正点 i 处校正前垂直误差
$ehpre_i$	校正点 i 处校正前水平误差
$evpost_i$	校正点 i 处校正后垂直误差
$ehpost_i$	校正点 i 处校正后水平误差
length	规划路径轨迹总长度

3.2 数据预处理

为方便对数据的读取和使用，首先将数据文件中的校验点数据按照各点 X 轴坐标值进行升序排列，并编序号。

4 问题一模型求解及算法设计

4.1 问题定义

问题 1. 针对附件 1 和附件 2 中的数据分别规划满足条件（1）~（7）时飞行器的航迹，并且综合考虑以下优化目标：

（A）航迹长度尽可能小；（B）经过校正区域进行校正的次数尽可能少。

并讨论算法的有效性和复杂度。

4.2 数学模型建立及求解

4.2.1 数学模型的建立

针对问题一的条件和假设，我们建立 0-1 规划数学模型。具体构建过程如下：

（1） 目标函数

针对题目要求的两个优化目标，假设两个目标优先级相同，最小化统一量纲后的总轨迹长度与经过的校验点数，目标函数如下图所示，其中x_i表示规划轨迹是否经过第i个校验点，1 是 0 否，length 表示可行解轨迹总长度，lengthMean，lengthStddev，countMean，countStddev 分别表示轨迹长度，经过校验点个数的抽样均值和标准差：

$$\text{Min}\ \frac{length-lengthMean}{lengthStddev}+\frac{\sum_{i\in H\cup V} xi-countMean}{countStdDev} \qquad (4\text{-}1)$$

（2） 约束条件

约束 1：保证飞行器在航行中任何位置的垂直误差和水平误差都小于θ,$evpre_i$,$ehpre_i$表示飞行器经过第i个校验点 X 坐标时的矫正前垂直、水平误差：

$$\forall i \in V \cup H,\ ehpre_i \le \theta,\ evpre_i \le \theta \qquad (4\text{-}2)$$

约束 2：在出发点 A 处，飞行器的垂直误差和水平误差都为 0，分别用ev_A ,eh_a表示：

$$ev_A = 0, eh_a = 0 \qquad (4\text{-}3)$$

约束 3：经过任意垂直校正点时，校正前垂直误差不大于α_1个单位，水平误差不大于α_2个单位，经过任意水平校正点时，校正前垂直误差不大于β_1个单位，

水平误差不大于β_2个单位。x_i表示规划路线是否经过第 i 个校正点：

$$\forall i \in V,\ evpre_i \times x_i \le \alpha_1,\ ehpre_i \times x_i \le \alpha_2 \quad (4\text{-}4)$$

$$i \in V \cup H\ ,\ x_i \in \{0,1\} \quad (4\text{-}5)$$

约束 4：飞行器在垂直误差校正点进行垂直误差校正后，其垂直误差将变为 0，水平误差保持不变，飞行器在水平误差校正点进行水平误差校正后，其水平误差将变为 0，垂直误差保持不变：

$$\forall i \in V, evpost_i = evpre_i - x_i \times evpre_i\ , ehpost_i = ehpre_i \quad (4\text{-}6)$$

$$\forall i \in H, ehpost_i = ehpre_i - x_i \times ehpre_i\ , evpost_i = evpre_i \quad (4\text{-}7)$$

约束 5：y_{ij}表示矫正点 i 和 j 是否为规划路径连续经过的两个矫正点，每个点最多只能有一个后续或前序节点，1 是 0 否：

$$y_{ij} \in \{0\ ,1\} \quad (4\text{-}8)$$

$$x_i - y_{ij} \ge 0 \quad (4\text{-}9)$$

$$x_j - y_{ij} \ge 0 \quad (4\text{-}10)$$

$$\sum_{t \in V \cup H} y_{ti} \le 1 \quad (4\text{-}11)$$

$$\sum_{t \in V \cup H} it \le 1 \quad (4\text{-}11)$$

约束 6：每飞行 1m，垂直误差和水平误差分别增加 δ 单位，d_{ij}表示第 i 个校正点到第 j 个校正点间的欧式距离：

$$evpre_1 = \delta \times d_{A1} \quad (4\text{-}12)$$

$$ehpre_1 = \delta \times d_{A1} \quad (4\text{-}13)$$

$$evpre_i = \sum_{t \in V \cup H} y_{ti} \times （evpost_t + \delta \times d_{ti}） \quad (4\text{-}12)$$

$$ehpre_i = \sum_{t \in V \cup H} y_{ti} \times （ehpost_t + \delta \times d_{ti}） \quad (4\text{-}13)$$

$$\text{length} = \sum_{t \in V \cup H} \sum_{u \in V \cup H} y_{tu} \times d_{tu} \quad (4\text{-}14)$$

4.2.2 数学模型的求解

本文使用 A*结合模拟退火算法求解问题 1。

A*算法是一种广泛用于寻路和图形遍历的计算机算法，该算法是在多个点之间寻找路径的过程。由于其性能和准确性，它得到了广泛的应用。斯坦福研究所的 Peter Hart，Nils Nilsson 和 Bertram Raphael 于 1968 年首次发布了该算法。该算法可以看作是 Edsger Dijkstra 1959 年算法的扩展。通过使用启发式方法来指导搜索，A*可获得更好的性能。A*是一种明智的搜索算法，或“ 最佳优先”搜索，意味着它是根据加权图来表示的：从图的特定起始节点开始，它的目的是查找到具有最小图的给定目标节点的路径费用（最短路程，最短时间等）。它通过维护始于起始节点的路径树并一次扩展一条路径直到满足其终止条

件来做到这一点。在主循环的每次迭代中，A*需要确定要扩展的路径。它基于路径的成本以及将路径一直扩展到目标所需的成本估算值来进行操作。具体来说，A*选择最小化的路径：f(n)=g(n)+h(n)。其中 n 是路径上的下一个节点，g（n）是从起始节点到 n 的路径的成本，h（n）是一种启发式函数，用于估计从 n 到目标的最便宜路径的成本。当 A *选择扩展的路径是从起点到目标的路径或没有适合扩展的路径时，A*终止。启发式功能是特定于问题的。如果启发式函数是允许的，这意味着它绝不会高估达到目标的实际成本，则可以保证 A *返回从起点到目标的最小成本路径。

A*算法的贪心选择同时考虑了当前决策点到下一决策点的实际代价以及下一决策点到终点的代价下限，使规划轨迹尽量靠近起终点的连线，这与问题一要求的两个优化目标都相吻合。

在本题中，若在每一步的决策时仅做使 f*最小的贪心选择，有可能找不到可行解，因此需要将 A*与广度优先搜索算法结合，在广度优先搜索算法的每一层优先遍历使 f*最小的邻居节点，若无法到达终点，则退回上一层选择使 f*次小的邻居节点继续寻找，以此类推。

模拟退火（SA）是一种概率技术，用于逼近给定函数的全局最优值。具体而言，在大搜索空间中针对优化问题近似全局优化是一种元启发法。当搜索空间是离散的（例如，旅行商问题）时，通常使用它。对于在固定时间内找到近似全局最优比找到精确局部最优更为重要的问题，模拟退火可能比梯度下降等替代方法更为可取。其名称和灵感来自冶金学中的退火技术，在模拟退火算法中实现的这种缓慢冷却的概念被解释为随着探索溶液空间而接受较差溶液的概率的缓慢降低。接受较差的解是元启发法的基本属性，因为它允许对全局最优解进行更广泛的搜索。通常，模拟退火算法的工作原理如下。在每个时间步长上，该算法都会随机选择一个与当前解决方案接近的解决方案，然后测量其质量，然后根据其选择的两个概率中的任意一个来决定移至该解决方案还是与当前解决方案保持一致新解决方案比当前解决方案更好或更坏的事实。在搜索过程中，温度从初始正值逐渐降低到零，并影响两个概率：在每个步骤中，转向更好的新解的概率要么保持为 1，要么变为正值。另一方面，转向较差的新解决方案的可能性逐渐变为零。

本文采用的模拟退火算法流程图如下：

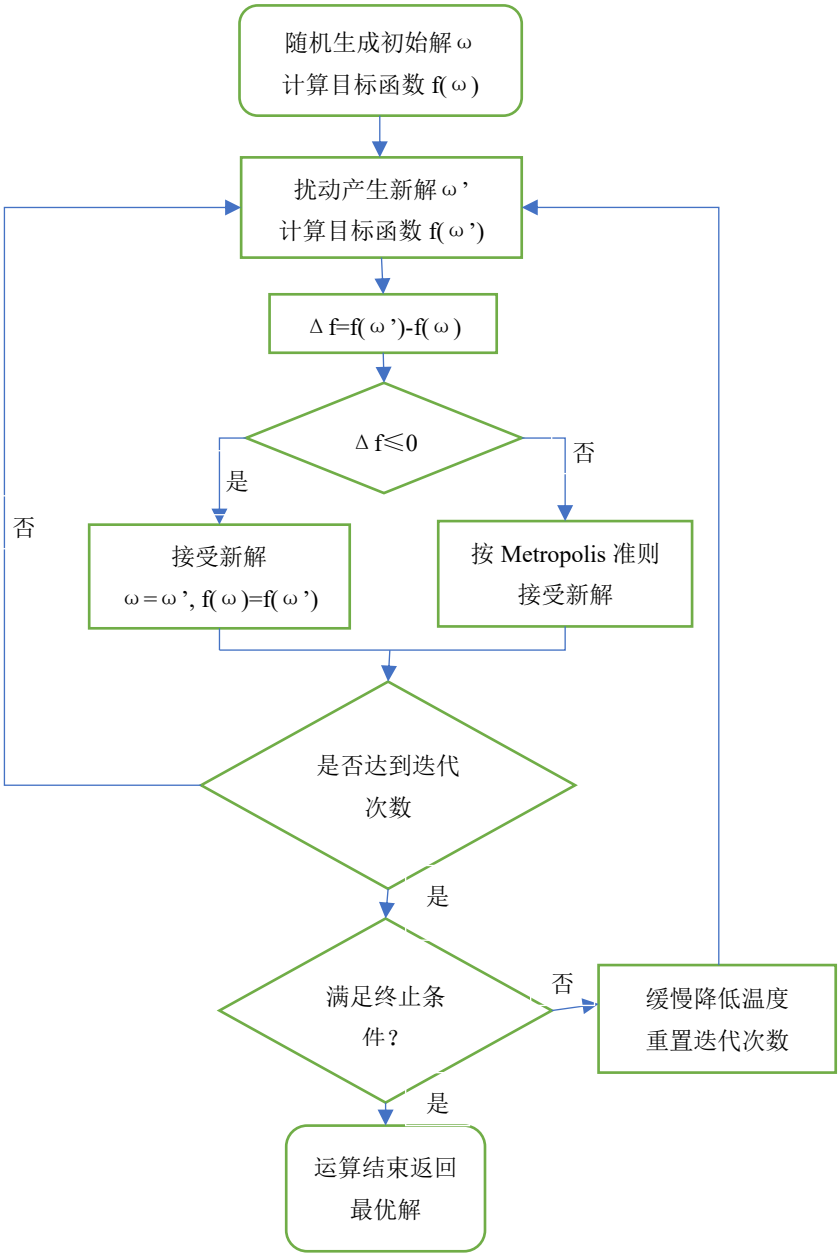

图 4-1 模拟退火算法流程图

在本题中，使用 A*算法生成较优可行解，再将可行解作为模拟退火算法的起点，寻找满意解。经多次调试，发现选择初温 100，恒温步数 10，降温系数 0.9 时算法的优化效果和执行时间皆比较可以接受。

在问题一的数学模型中， 我们先使用 A*算法，求的一组较优的可行解作为模拟退火算法的初始解，再使用模拟退火算法寻找比较满意的解。对于数据集一，使用 A*算法生成经过校正区域校正次数为 12 次，航迹长度为 114818.94 米的初始解。使用此解作为初始解，经过不同初始温度和降温速度的组合测试，最终在初始温度为 100， 恒温步数为 10，降温速度为 0.9 的组合下，模拟退火算法收敛到了局部最优解，规划路径经过校正区域校正次数为 9 次，航迹长度为 110527.68 米；对于数据集二，使用 A*算法生成经过校正区域校正次数为 15 次，航迹长度为 132192.36 米的初始解。以此作为初始解，最终在初始温度为 100， 降温速度为 0.9 的组合下，模拟退火算法收敛到局部最优解，规划路径经过校

正区域校正次数为 9 次，航迹长度为 117971.67 米。具体的数据信息显示如下：

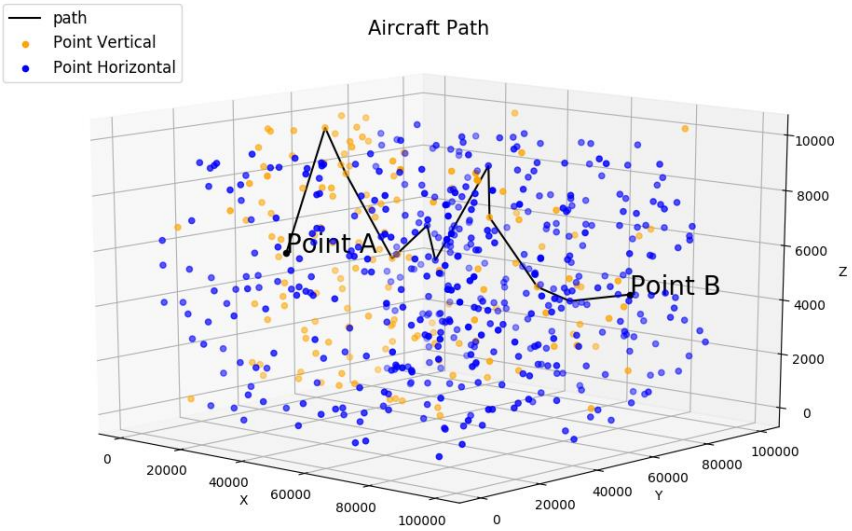

图 4-1 问题一数据集一航迹规划路径图

表 4-1 问题一数据集一航迹规划结果表

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
303	14.139377	14.139377	11
64	18.244164	4.104787	11
80	9.136843	13.24163	01
170	18.718486	9.581643	11
155	3.05888	12.640524	01
250	17.28948	14.230599	01
369	18.907583	1.618103	11
141	12.913907	14.53201	01
397	18.973759	6.059852	11
612	13.578153	19.638005	终点 B

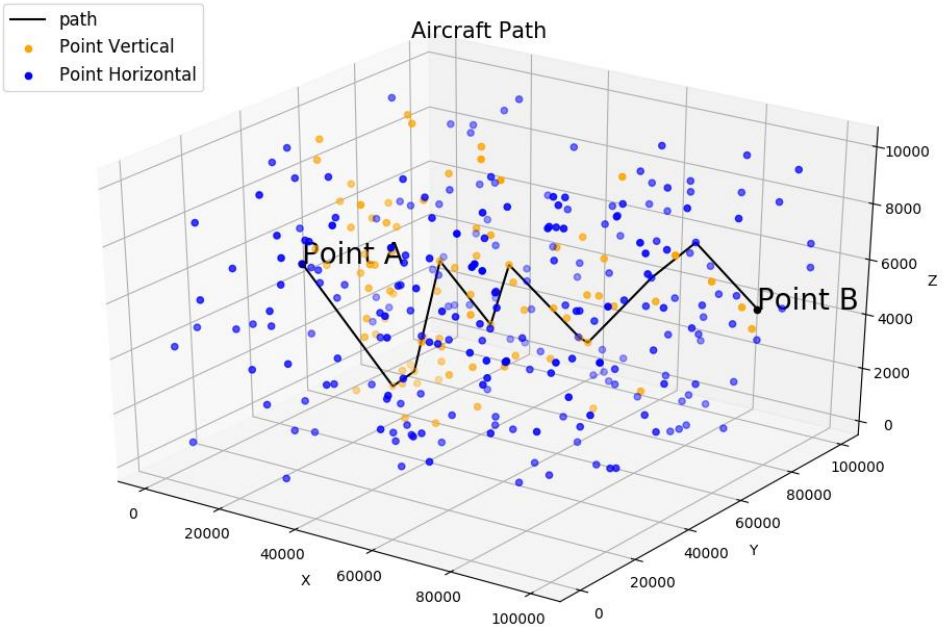

图 4-2 问题一数据集二航迹规划路径图

表 4-2 问题一数据集二航迹规划结果表

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
40	9.906776	9.906776	01
290	12.397956	2.491179	11
8	5.834728	8.325908	01
148	11.737365	5.902636	11
255	4.033647	9.936284	01
41	11.983446	7.949798	11
221	1.8661	9.815899	01
50	8.680184	6.814083	01
96	14.399127	5.718943	01
326	8.467939	14.186882	终点 B

4.3 算法优势与复杂度分析

相比于将“最大 X 轴坐标步长”“最小与终点 B 的距离”作为贪心选择策略，A^*算法规定的贪心选择策略能提供最接近于 AB 点之间连线的路径，与题目所要求的两个优化目标最为匹配。对于此解法，A^*算法的时间复杂度为 O(n)，空间复杂度为 $O(n^2)$。模拟退火算法的时间复杂度为 O(n)，空间复杂度为 $O(n^2)$。

5 问题二模型求解及算法设计

5.1 问题的定义

问题 2. 在问题一的基础上加入飞行器的转弯半径作为飞行器过程的约束条件，飞行器在转弯时受到结构和控制系统的限制，无法完成即时转弯(飞行器前进方向无法突然改变)，假设飞行器的最小转弯半径为 200m。即在每一个决策点处寻求可达点时，需要增加最小转弯半径的条件限制。

5.2 数学模型的建立及求解

5.2.1 数学模型的建立

首先，在决策点处，设置最小转弯半径的约束条件。如图 5-1 所示，在校验点 B 处，判断点 C 是否可以作为飞行器下一经过的矫正点，需要考虑点 B 的前序校正点 A，若点 ABC 三点确定的曲率半径小于 200m，则不将点 C 作为点 B 的可达点。

据此增加约束条件

$$\frac{d_{AB}\times d_{BC}\times d_{AC}}{\sqrt{4*{d_{AC}}^2{d_{AB}}^2-\left({d_{AC}}^2+{d_{AB}}^2+{d_{BC}}^2\right)^2}}>200 \qquad (5\text{-}1)$$

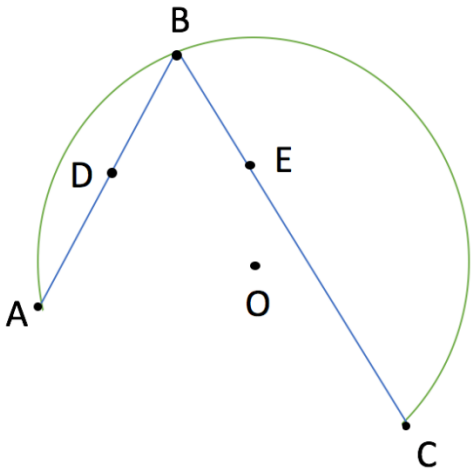

图 5-1 曲率半径的极限情况

其次，题目要求画出飞行器经过各校验点的飞行轨迹。为方便计算轨迹长度和画出轨迹图，本文使用图 5-2 的方式将轨迹进行简化，飞行路径为 A->D->E->C：

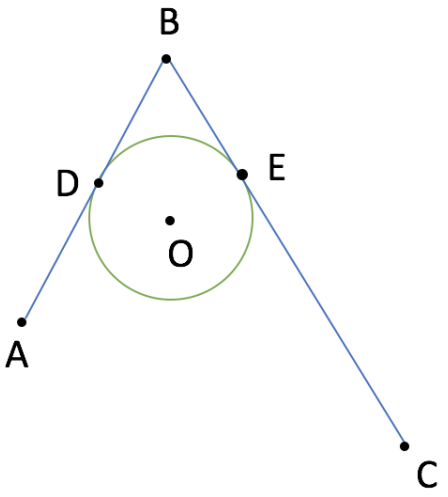

图 5-2 圆弧路径示意图

其中圆 O 的直径为 200m，O 点向量可以通过 BA，BC 向量表示，联立方程组可以解得 O 点坐标

O 点坐标求得后，弧 DE 的轨迹可以用 OD，OE 向量表示：

等间隔的取多组 α 、β 的值，可画出弧 DE 上的多个散点，增加散点个数，即可画出弧上轨迹。

5.2.2 数学模型的求解

在问题二的数学模型中，增加飞行器最小转弯半径的约束。这里我们延续使用问题一中的 A*算法计算出的可行解作为模拟退火算法的初始解。对于数据集一，在初始温度为 100，恒温步数为 10， 降温速度为 0.9 的组合下，得到了满意解：规划路径经过校正区域校正次数为 9 次，航迹长度为 115609.49 米；对于数据集二，在同样组合条件下，我们得到了满意解：规划路径经过校正区域校正次数 8 次，航迹长度为 119583.42 米。具体的数据信息显示如下：

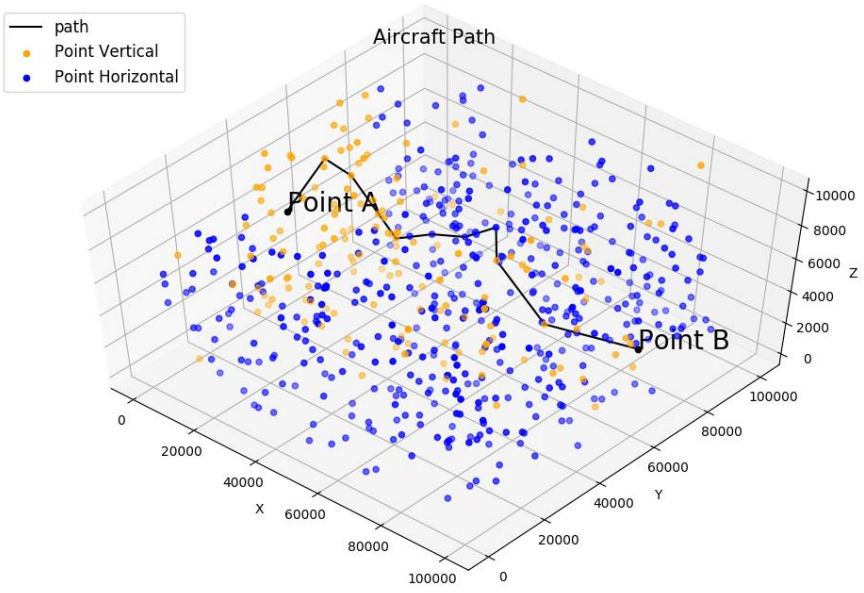

图 5-1 问题二数据集一航迹规划路径图

表 5-1 问题二数据集一航迹规划结果表

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
303	13.432408	13.432083	01
199	18.970213	5.537805	11
80	8.963979	14.501785	01
170	18.066541	9.102561	11
282	5.774023	14.876584	01
250	18.364946	12.590923	01
369	19.902144	1.537197	11
141	12.368212	13.805409	01
397	18.025071	5.756859	11
612	12.899245	18.656104	终点 B

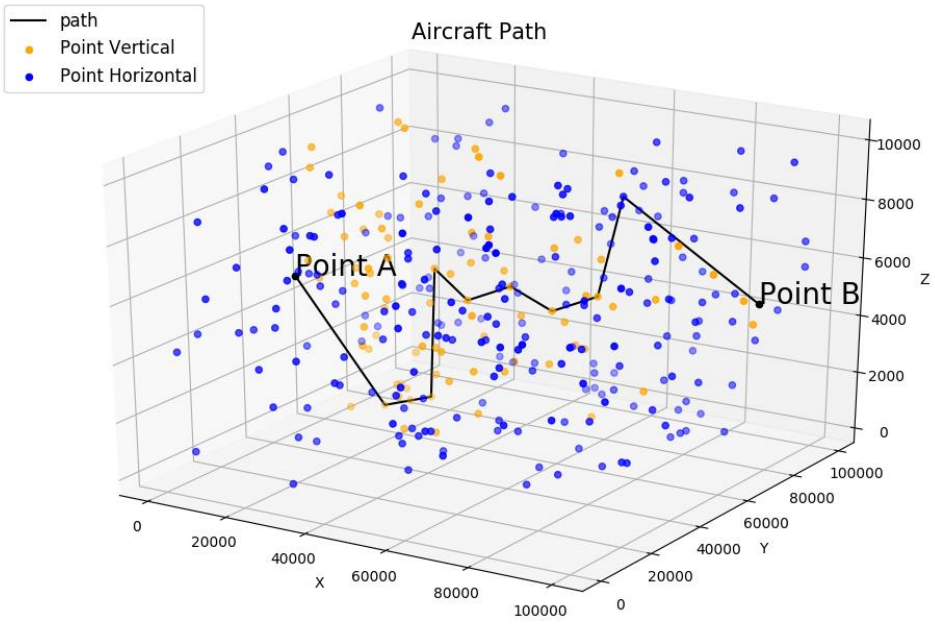

图 5-2 问题二数据集二航迹规划路径图

表 5-2 问题二数据集二航迹规划结果表

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
40	9.70864	9.708641	01
58	14.746183	5.037542	11
8	4.754531	9.792074	01
305	10.376057	5.621525	01
123	14.88621	4.510152	11
45	4.903019	9.413171	01
191	9.680463	4.777444	01
175	14.841105	5.160641	11
326	14.12238	19.283022	终点 B

6 问题三模型求解及算法设计

6.1 问题的定义

问题三：本问题是在问题一的基础上增加飞行器在部分校正点进行误差校正时存在无法达到理想校正的情况。增加该条件后，问题一中的可行解在概率下可能无法到达终点。因此，将可能校验失败的节点处的概率用线性表达式抽象，并将可达终点的概率加入目标函数。

6.2 数学模型的建立及求解

6.2.1 数学模型的建立

对于第三问中的假设，对原有算法进行修改，将路径可达终点的概率 pos 的无量纲量加入目标函数。

$$\text{Min}\ \frac{length-lengthMean}{lengthStddev}+\frac{\sum_{i\epsilon H\cup V} xi-countMean}{countStdDev}+\frac{\text{pos}-posMean}{posStdDev} \qquad (6\text{-}1)$$

6.2.2 数学模型的求解

在问题三的数学模型中，需要考虑到校正点校正概率的问题。我们使用问题一中模拟退火算法的解作为问题三模拟退火算法的初始解。对于数据集一，在初始温度为 100，恒温步数为 10， 降温速度为 0.8 的组合下，得到局部最优解：规划路径经过校正区域校正次数为 12 次，校正距离为 120033.42，成功到达终点的概率为 0.556733。对于数据集二，在初始温度为 100，恒温步数为 10， 降温温度为 0.8 的组合下，得到局部最优解：规划路径经过校正区域校正次数为 12 次，校正距离为 130866.02，成功到达终点的概率为 0.367245。

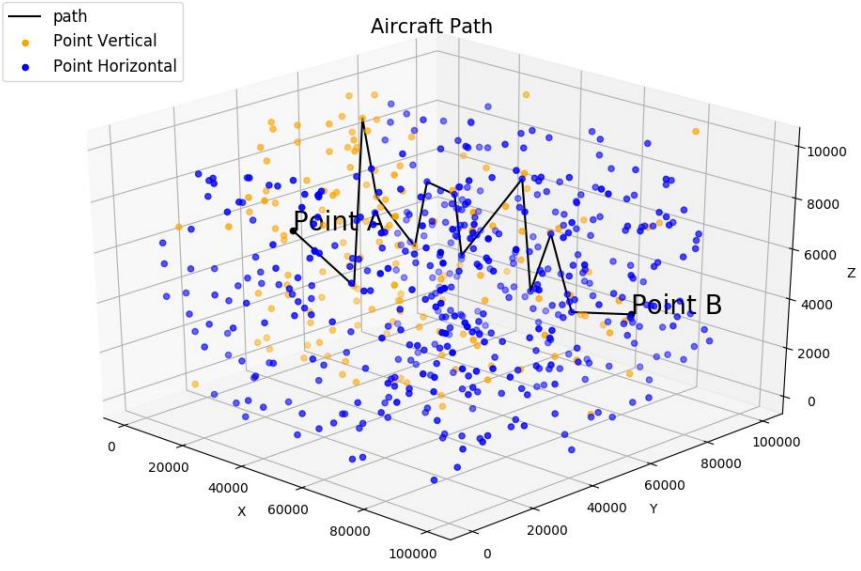

图 6-1 问题三数据集一航迹规划路径图

表 6-1 问题三数据集一航迹规划结果表

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
431	13.826512	13.826512	01
417	19.95347	6.126963	11
294	4.526866	10.653829	01
237	14.155517	9.628651	11
607	4.409425	14.038077	01
540	15.487378	11.077952	11
278	2.541119	13.619071	01
315	15.081109	12.53999	01
450	19.602409	4.521299	11
448	7.403904	11.925204	01
397	18.402845	10.99894	11
612	14.426787	25.425728	终点 B

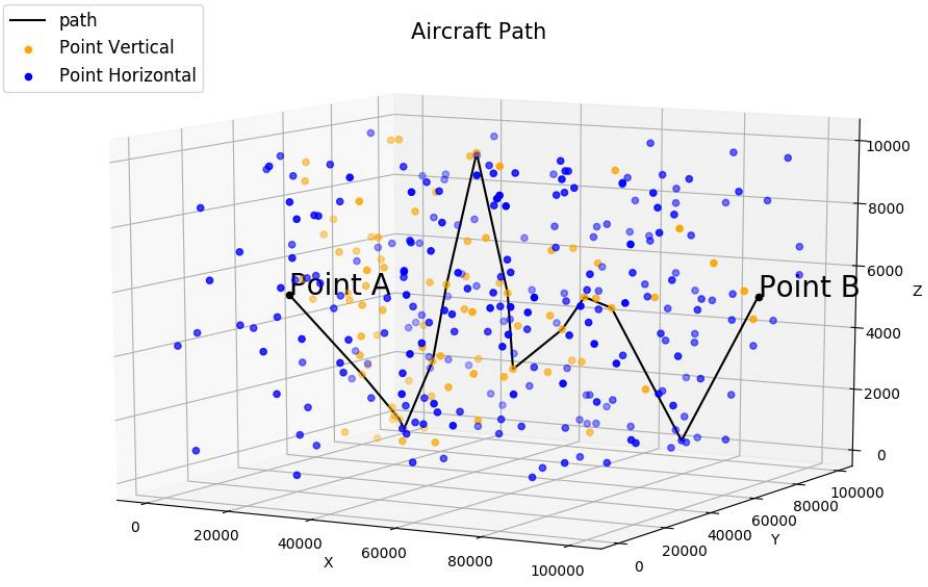

图 6-2 问题三数据集二航迹规划路径图

表 6-2 问题三数据集二航迹规划结果表

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
104	9.427599	9.427599	01
290	14.659527	5.231928	11
209	2.186053	7.417981	01
227	10.936699	8.750646	01
309	14.214521	3.277821	11
54	6.6153	9.893122	01
123	11.623801	5.0085	11
314	4.108947	9.117448	01
49	10.98269	6.873742	01
160	14.890974	3.908283	11
92	3.465698	7.373981	01
287	13.242634	9.776936	11

7 总结分析

第一问我们建立了 0-1 规划数学模型，并且采用去量纲的方法将多目标的优化问题转化为单目标的优化问题。我们首先采用了 A*算法，求出一个比较优的可行解，然后将此作为模拟退火算法的初始解，执行模拟退火算法，最终成功的规划出了在题给约束条件约束下的飞行器飞行路径。第二问在第一问的基础上添加了转弯半径的约束条件，延用第一问 A*算法的初始解，执行模拟退火算法。第三问在考虑到可行解在概率下可能无法到达终点，将可能校验失败的截断出的概率用现行表达式抽象，并将可达终点的概率加入目标函数，执行模拟退火算法。

- **问题一到三结论如下：**

  问题一：

  对于数据集一，规划路径航迹长度为 110527.68 米，校正次数为 9 次；对于数据集二，规划路径长度为 117971.67 米， 校正次数为 9 次

  问题二：

  对于数据集一，规划路径航迹长度为 115609.49 米，校正次数为 9 次；对于数据集二，规划路径长度为 119583.42 米， 校正次数为 8 次

  问题三：

  对于数据集一，规划路径航迹长度为 120033.42 米，校正次数为 12 次，成功到达终点的概率为 0.556733；对于数据集二，规划路径长度为 130866.02 米，校正次数为 12 次，成功到达终点的概率为 0.367245

- **问题一到三总结如下：**

  模拟退火算能够快速的求出最优解，但是不能保证是全局最优解，算法结果也受参数的影响。当问题比较复杂的时候，最好能先找到一个比较优的可行解作为退火算法的初始解。

参考文献

[1] Wikipedia contributors, A* search algorithm, https://en.wikipedia.org/wiki/A*_search_algorithm 2019.9.23

[2] Wikipedia contributors, Simulated annealing, https://en.wikipedia.org/wiki/Simulated_annealing 2019.9.23

[3] 王洪斌,郝策,张平,张明泉,尹鹏衡,张永顺.基于A*算法和人工势场法的移动机器人路径规划[J/OL]. 中国机械工程:1-8[2019-09-23].http://kns.cnki.net/kcms/detail/42.1294.TH.20190909.0950.002.html.

[4] 林宁,邢丽娟,徐珂,秦立峰.改进 A*算法在磁导引 AGV 路径规划中的应用[J].自动化技术与应用,2019,38(07):5-8.

[5] 马云红,张恒,齐乐融,贺建良.基于改进A*算法的三维无人机路径规划[J/OL].电光与控制:1-5[2019-09-23].http://kns.cnki.net/kcms/detail/41.1227.tn.20190624.1645.016.html.

[6] 陈婷.基于模拟退火粒子群算法的含分布式电源配电网故障定位[J].电气技术,2019,20(08):59-63.

[7] Maria Claudia Aguitoni,Leandro Vitor Pavão,Mauro Antonio da Silva Sá Ravagnani. Heat exchanger network synthesis combining Simulated Annealing and Differential Evolution[J]. Energy,2019,181.

[8] Gokcecicek Tasoglu,Gokalp Yildiz. Simulated annealing based simulation optimization method for solving integrated berth allocation and quay crane scheduling problems[J]. Simulation Modelling Practice and Theory,2019,97.

[9] Himanshu Rathore,Shirsendu Nandi,Peeyush Pandey,Surya Prakash Singh. Diversification-based learning simulated annealing algorithm for hub location problems[J]. Benchmarking: An International Journal,2019,26(6).

[10] Elias D. Nino-Ruiz,Xin-She Yang. Improved Tabu Search and Simulated Annealing methods for nonlinear data assimilation[J]. Applied Soft Computing Journal,2019,83.

附录 1：部分代码

```
# 成本大小排序函数
def sort_by_f_star(path, neighbours):
    neighbours['f_star'] = 0
    for i in range(neighbours.shape[0]):
        neighbours.iloc[i, 6] = f_star(path, neighbours.iloc[i, :])
    neighbours = neighbours.sort_values(by='f_star', ascending=False)
    neighbours = neighbours.drop(columns=['f_star'])
    return neighbours

# A*算法
def a_star_path(point, path):
    path = path.append(point)
    neighbours = find_neighbours(point, path)
    if neighbours.shape[0] == 0:
        return False
    if neighbours.iloc[0, 4] == 'B':
        path = path.append(PointB_data.iloc[0, :])
        return path
    neighbours = sort_by_f_star(path, neighbours)
    i = neighbours.shape[0] - 1
    while i >= 0:
        ret = a_star_path(neighbours.iloc[i, :], path)
        if isinstance(ret, bool):
            i -= 1
        else:
            return ret
    return False

# A*算法成本函数
def f_star(path, point):
    return get_path_dis(path, path.shape[0]-1) + \
           cal_dis(point, path.iloc[path.shape[0]-1, :]) \
           + cal_dis(PointB_data.iloc[0, :], point)

# 模拟退火算法代价计算函数
def cal_cost(path):
    w1 = 1
    w2 = 1
    dis = get_path_dis(path, path.shape[0]-1)
    count = path.shape[0]
    return w1*(dis-DIS_MEAN)/DIS_STD + w2*(count-COUNT_MEAN)/COUNT_STD
```

```
# 去量纲操作
def cal_stat(path):
    dis = []
    count = []
    for i in range(100):
        new_path = generate_new(path)
        dis.append(get_path_dis(new_path, new_path.shape[0]-1))
        count.append(new_path.shape[0])
    global DIS_MEAN
    DIS_MEAN = np.mean(dis)
    global DIS_STD
    DIS_STD = np.var(dis)
    global COUNT_MEAN
    COUNT_MEAN = np.mean(count)
    global COUNT_STD
    COUNT_STD = np.std(count)

# 模拟退火算法状态产生函数
def generate_new(path):
    while True:
        alt_point_index = random.randint(1, path.shape[0]-2)                    # 随机选择变异点的下标
        new_path = path.iloc[0:alt_point_index, :]
        point = path.iloc[alt_point_index, :]
        neighbours = find_neighbours(path.iloc[alt_point_index-1, :], new_path)
        temp = neighbours[(neighbours['X'] == point['X']) &
                          (neighbours['Y'] == point['Y']) &
                          (neighbours['Z'] == point['Z'])]
        neighbours.drop(temp.index, inplace=True)
        neighbours = sort_by_f_star(path, neighbours)
        i = neighbours.shape[0] - 1
        while i >= 0:
            ret = a_star_path(neighbours.iloc[i, :], new_path)
            if isinstance(ret, bool):
                i -= 1
            else:
                new_path = new_path.append(ret)
                return new_path

# 模拟退火算法
def algorithm_sa():
    solution = pd.DataFrame(input)
    cal_stat(solution)
    temperature = 100.0
```

```
    cooling_rate = 0.8
    iterations = 1
    cost = cal_cost(solution)

    while temperature > 1.0:
        temp_solution = generate_new(solution)
        current_cost = cal_cost(temp_solution)
        diff = current_cost - cost

        rand = random.random()
        if diff < 0 or rand < math.exp(-diff/(temperature*0.1)):
            solution = temp_solution
            cost = current_cost
            iterations = iterations + 1
        if iterations >= 10:
            temperature = cooling_rate*temperature
            iterations = 1
    solution = generate_new(solution)
    return solution

# 查找可达点函数
def find_neighbours(point, path):
    neighbours = pd.DataFrame(columns=['num', 'X', 'Y', 'Z', 'type', 'flag'])
    if point[4] == 1:
        cur_error_v = 0
    elif point[4] == 0:
        cur_error_h = 0
    elif point[4] == 'A':
        cur_error_v = 0
        cur_error_h = 0
    max_dx_a1 = (ALPHA1-cur_error_v)/DELTA
    max_dx_a2 = (ALPHA2-cur_error_h)/DELTA
    max_dx_b1 = (BETA1-cur_error_v)/DELTA
    max_dx_b2 = (BETA2-cur_error_h)/DELTA
    min_dx = min(max_dx_a1, max_dx_a2, max_dx_b1, max_dx_b2)
    max_dis_to_b = min((THETA - cur_error_v) / DELTA,
                       (THETA - cur_error_h) / DELTA)
    if max_dis_to_b >= cal_dis(PointB_data.iloc[0, :], point):
        return PointB_data
    temp = Origin_data[(Origin_data['X'] == point['X']) &
                       (Origin_data['Y'] == point['Y'])
                       & (Origin_data['Z'] == point['Z'])]
    point_index = temp.index.tolist()[0]
    t = point_index + 1
```

```
    while t < Origin_data.shape[0] and \
            math.fabs(Origin_data.iloc[t, 1] - Origin_data.iloc[point_index, 1]) < min_dx:
        dis = cal_dis(Origin_data.iloc[t, :], Origin_data.iloc[point_index, :])
        if dis <= min_dx:
            check_point = path[(Origin_data.iloc[t, 1] == path['X'])
                               & (Origin_data.iloc[t, 2] == path['Y'])
                               & (Origin_data.iloc[t, 3]== path['Z'])]
            if check_point.empty:
                neighbours = neighbours.append(Origin_data.iloc[t, :])
        t += 1
    return neighbours

# 误差获取函数
def get_error(path, index):
    if index < 1:
        return 0, 0
    cur_error_v = 0
    cur_error_h = 0
    if index == 1:
        cur_error_v = DELTA * cal_dis(path.iloc[0, :], path.iloc[1, :])
        cur_error_h = DELTA * cal_dis(path.iloc[0, :], path.iloc[1, :])
        return cur_error_v, cur_error_h
    i = 1
    while i <= index:
        if path.iloc[i-1, 4] == 'A':
            cur_error_v = DELTA * cal_dis(path.iloc[i-1, :], path.iloc[i, :])
            cur_error_h = DELTA * cal_dis(path.iloc[i-1, :], path.iloc[i, :])
        elif path.iloc[i-1, 4] == 1:
            cur_error_v = 0 + DELTA * cal_dis(path.iloc[i-1, :], path.iloc[i, :])
            cur_error_h += DELTA * cal_dis(path.iloc[i - 1, :], path.iloc[i, :])
        else:
            cur_error_v += DELTA * cal_dis(path.iloc[i-1, :], path.iloc[i, :])
            cur_error_h = 0 + DELTA * cal_dis(path.iloc[i-1, :], path.iloc[i, :])
        i += 1
    return cur_error_v, cur_error_h

# 空间两点计算函数
def cal_dis(point1, point2):
    d_x = point1['X'] - point2['X']
    d_y = point1['Y'] - point2['Y']
    d_z = point1['Z'] - point2['Z']
    return math.sqrt(d_x**2 + d_y**2 + d_z**2)

# 获取路径当前路径长度函数
```

```
def get_path_dis(path, index):
    i = 0
    result = 0
    while i < index:
        result += cal_dis(path.iloc[i, :], path.iloc[i+1, :])
        i += 1
    return result
```