基于人工势场无源定位的无人机编队调整模型

摘要

本文研究的是无人机集群在编队飞行中无源定位的位置调整方案问题，通过考虑发射信号无人机数量的约束条件，仅考虑据接收到的方向信息，建立了**定弦定角模型**、**遍历编队结构**、**人工势场优化模型**和**基于遗传算法的自适应人工势场模型**，最后对模型进行灵敏度分析，对结果进行分析，证明了基于遗传算法改进的人工势场模型对于无人机纯方位无源定位起到良好作用。

针对问题一，对所有无人机所组成的圆形编队结构，由角度信息可以把无人机定位于两个圆周上，由此建立的**定弦定角的模型**能得到**交点坐标的解析解**，以此求解目标无人机在人工建立的坐标系下的具体坐标。

针对问题二，固定 $FY00$ 和 $FY01$ 两架无人机发射信号，若再加入一台发射信号无人机进行定位，对除了 $FY00$ 和 $FY01$ 以外的 8 架无人机进行**遍历分析**，把每一次遍历分析的三架无人机代入到问题一中的三点定位模型去求解，会产生多个可能位置，无法准确定位。考虑仍需要两台无人机进行定位，从剩余 8 台无人机中选出 2 台，共有 $\mathbf{C_8^2 = 28}$ 种选法，将选出的 2 台无人机分别与 $FY00$ 和 $FY01$ 使用定弦定角模型，计算出两组定位坐标，若两组定位坐标相同即完成定位，否则重新选取，直至选取的两组定位坐标相同。根据三圆确定一点即可得出求解的唯一性，得到结果：**仍需要 2 台无人机**。

针对问题三，建立**人工势场优化模型**，对无人机的位置进行迭代优化。首先分享发射信号无人机组合，选择 $\{\mathbf{0147}\}, \{\mathbf{0258}\}, \{\mathbf{0369}\}$ 三组作为发射源，只需对其它无人机使用一种策略即可，使计算量减小。在模型的每一次迭代优化中，将无人机接收到的有效的三个角度信息作为坐标，映射到三维空间中，得到人工构建的三维势场点云模型，对该点云进行**归一化**，即进行三维空间中的**仿射变换**，使特征间的差异更明显。使用 $KD-tree$，应用于三维势场空间坐标查找，以进行线性插值。将查找到的临近点所对应的更新方向加权平均得到差值点势场的更新向量，从而确定了每一角度信息的调整向量。为了让发射位置的无人机也获得位置更新，需要在每次迭代后依次更换无人机的编号组合。模型经过迭代更新后，**误差下降至 0.7%**

针对问题四，建立在问题三基础之上，不同的是问题四中无人机的标准编队队形由旋转对称变为仅有轴对称。首先让顶点互发信息，使用**人工势场优化模型**确定顶点更新策略，让顶点先做微调。再对内部点构建三维势场空间，建立**人工势场优化模型**。由于角度信息减少为 2 个，需要对势场进行修正，使用**遗传算法**优化势场。在顶点调整完后让内部点在**遗传算法**建立的势场中迭代更新。模型经过多次迭代更新后，**误差下降至 1.3%**

关键词: 人工势场模型　遗传算法　定弦定角模型

一、问题重述

1.1 问题背景

随着计算机算力的提升与无线电技术的发展，无人机在技术上越来越成熟，广泛应用民用、军事于各领域之中。[1] 随着电子对抗技术在当今战争中占到越来越重要的地位，为了防止敌方对我方无人机编队的无人机电磁干扰，在多无人机集群不采用 GPS 与地面基站进行定位时，不同无人机之间采用基于方向信息的无源定位。因此实现多无人机集群的无源定位、提高无源定位速度与精度十分有研究意义与价值。

1.2 问题重述

无人机编队飞行中采用某几架无人机发射信号、其余无人机被动接受信号，从中获取无人机位置信息的纯方位无源定位方法来进行集群定位。获取的信息为被动接受信号无人机与任意两架发射信号无人机的夹角。

基于上述背景我们需要建立数学模型解决以下问题：

问题一：如图 1所示，在 10 架无人机（1 架在圆心，其余 9 架均匀分布在圆周）形成的圆形编队中解决当处于圆心的无人机与编队中其余两架已知编号无人机发射信号时，接收信号的无人机定位问题。

问题二：在问题一的背景下，若除了位于中心的 FY00 与圆周上的 FY01 以外其余无人机的编号未知时，解决接收信号无人机的定位问题。

问题三在圆形标准编队的背景下，当初始的发射与接收无人机位置略有偏差时，如何给出合理的无人机位置调整方法，使之调整多次后能够形成 1 架无人机位于圆心，另 9 架无人机均匀分布在某个圆周上的飞行编队。每次只能选择编号为 FY00 与圆周上其他最多三架无人机进行定位。

问题四：提出如图 2所示的锥形无人机编队方案，并在此方案下，考虑无源定位方案下的无人机调整方案。

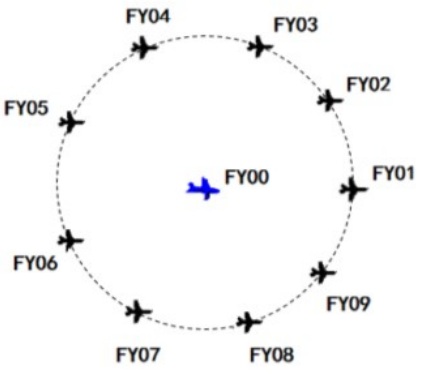

图 1: 圆形无人机编队示意图

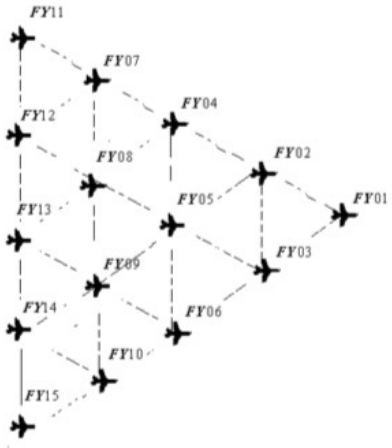

图 2: 锥形无人机编队示意图

二、问题分析

2.1 问题一的分析

问题一是根据位置无偏差的发射信号无人机确定被动接受信号无人机定位的问题。对于被动接收信号无人机接收的每一个角度方位信号，在由中心无人机、发射信号无人机与被动接收信号无人机建立的三角形中。接收方位信号的角度所对应的边长固定为编队飞行的半径 100m。角度与角度所对应的弦长固定，从而转化为定角定弦模型，由该模型性质可知，点的运动曲线在固定的圆周上。在具有三架发射信号无人机时可以获得两个有效的定角定弦模型，在确定两个圆时，利用接收信号信息解决圆的多解性问题。根据两个圆可以确定一个确定坐标的性质，从而求解出该点的坐标信息，从而解决无人机定位问题。

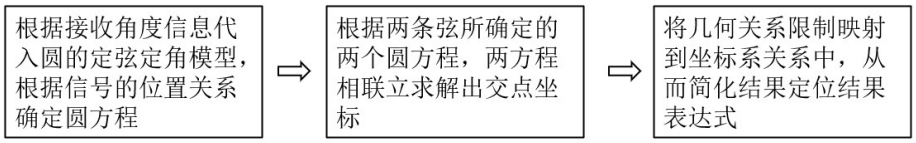

图 3: 问题一分析流程图

2.2 问题二的分析

首先，我们考虑对除了 FY00 与 FY01 外仅有一架发射无人机的情况进行分析，在此情况下会出现无人机多解性的问题，说明获取的信息量还没有达到能够唯一定位的要求。因此我们增设另外一架发射信号的无人机以增加信息维度。在有四架无人机的情况下，由于 FY01 与 FY00 是固定的，因此共有 28 种其余两架无人机的排布方式。接着我们进行每一种情况的遍历求解，将 FY00、FY01 与任意一架编号未知的无人机代入问题一所建立的模型进行接收无人机的定位信息求解。考虑两次分别求解的位置信息，若两次位置信息相同，则该种方案为正确编号与正确位置信息。若在 28 种情况中出现一种以上的结果，则说明无人机的数量还不够，需要继续增设无人机，直至确定最少无人机数量与定位点的精确信息。

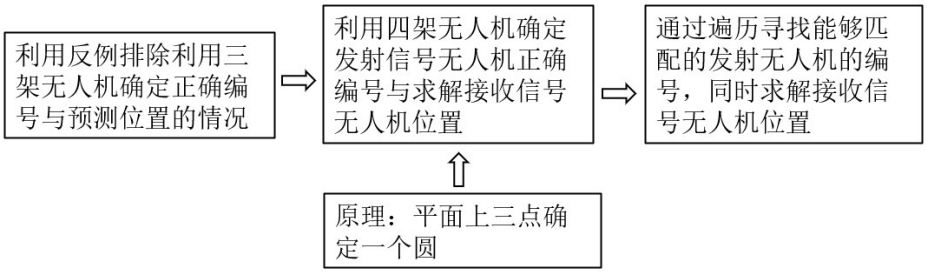

图 4: 问题二分析流程图

2.3 问题三的分析

根据题目意思，在除了中央 FY00 无人机以外的所有无人机位置均略有偏差，需要找到调整策略使得无人机能够调整到标准编队中的正确位置。由于发射与接收无人机的位置均不准确，因此我们只能通过迭代优化的方式对无人机集群进行整体优化，使之超指向正确位置移动。由于我们需要确定无人机迭代更新的方向与距离，因此需要建立标准位置点的人工势场，根据略有偏差点的无人机获取信息角度情况确定该点在势场位置，从而沿势场梯度方向下降，更新所在位置。对于每次迭代，选取圆周上平均间隔的三个点作为信息发射的无人机，对圆周上其余 6 点当成接收信号无人机作位置的迭代更新，进入下一次迭代时，选取圆周上另外间隔相等的三点作为发射信号无人机，再对其余六点的位置进行迭代更新。由于无人机无法知道其余无人机是否到达正确位置，因此我们人为指定迭代次数，经过迭代次数更新后停止迭代，最后计算模型误差。

2.4 问题四的分析

对于问题四，本文考虑分别位于最外侧三个顶点的三架无人机所组成的等边三角形拓扑结构，假设其中两架无人机发射信号，另外一台无人机被动接收信号。对于被动接收信号的无人机，它能够接收来自发射无人机的方向信息，即 α_1 角，根据 α 与等边三角形内角 60 的大小关系来进一步判断该无人机的运动。对于锥形边界上的无人机，把它们作为接收信号的无人机，去接收来自此边界上位于两个顶点无人机发射的角度信息，记作 α_2，由于边界上的无人机应与两个顶点的无人机处在同一条直线上，根据三点共线可以判断得到位于边界上的无人机应往着使 $\alpha_2 = 180$ 的方向进行调整。对于内部的无人机，则接收来自边界和顶点上的无人机协同发送的角度信息来进行调整。类似问题三建立的模型，改变被动接收信号的无人机，重复上面的过程进行迭代。

三、模型假设

为了对模型进行合理简化，本文给出如下假设：

1、不考虑无人机调整过程中的时间。

2、无人机在发射信号的时候无法接收信号，只有被动接收信号的无人机可以接收信号。

3、无人机的位置略有偏差假设为距离正确编队定位不超过 10m

4、无人机仅能接收方位角度信息，无法测量任意两架无人机之间的距离。

5、所有无人机基于自身感知高度信息，均在统一高度飞行。

6、不考虑无人机在飞行过程中的重叠问题

四、符号说明

符号	说明	单位
α	被动接收信号无人机观测角度值	∘
θ	圆周上主动发送信号无人机所成夹角	∘
R	标准圆形无人机编队半径	m

五、模型的建立与求解

5.1 问题一：根据定角定弦模型，确定无人机定位坐标

针对问题一，我们需要建立定弦定角的几何模型，将平面情况抽象成两个定弦定角模型的叠加，并计算两个轨迹圆的正确解析方程，将两个圆的方程表达式相联立，最后求解出该点在平面坐标系下的坐标，实现接收信息无人机的准确定位。由题目信息可知，发射信号的无人机的位置无偏差，因此如图 5可知，处于圆心处的 FY01 与圆周上的其他两家无人机的位置共有四种分布情况，分别对应圆心角为 0°、40°、80°、160°，其余位置的情况均可以通过旋转与翻转转化为如图所示的四种情况。因此，我们将圆心角进一步抽象为 θ 进行问题的求解。

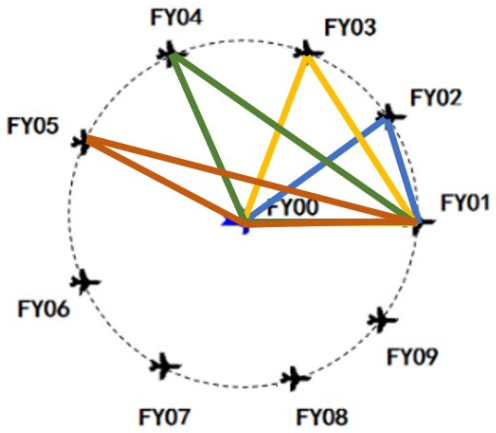

图 5: 发射信号无人机方位的四种情况

5.1.1 模型的建立

如图 6所示，对于固定长度的 BC 段直线。若 A 点与 B、C 两点所成的角度恒为一个定值 α，那么点 A 运动的轨迹在一个圆上。根据正弦定理可以得到轨迹圆的半径 R:

$$\frac{BC}{sin(\alpha)} = 2R \tag{1}$$

其中 BC 为 BC 线段的长度，R 为 A 形成轨迹圆的半径。

在固定 BC 线段的条件下加入坐标系，定弦定角会出现如图 6与 7的两种情况，即移动的点 A 位于固定弦 BC 的上方与下方，若 B 点为坐标系原点，则对应圆心的纵坐标会出现正值与负值两种情况。在后续求解过程中会对轨迹圆的解析式产生影响。

如图 8，根据题意，假设圆周上的两家发射信号无人机分别为 FY0X 与 FY0Y，两架无人机所成角度为 θ。FY00 与 FY0X、FY00 与 FY0Y 之间的距离均为标准编队中的半径 100m。在被动接收信息无人机端，观测的 FY0X 与 FY00 之间的夹角为 α_1，观测的 FY00 与 FY0Y 之间的夹角为 α_2。因此，建立模型 1 为两个定弦定角模型的叠加。

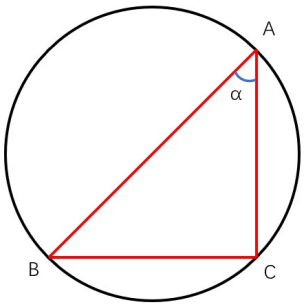

图 6: 定弦定角模型情况一

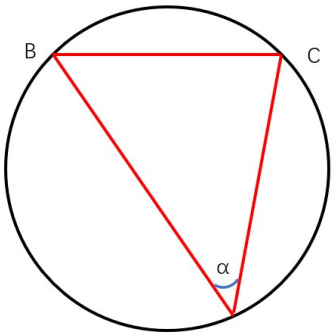

图 7: 定弦定角模型情况二

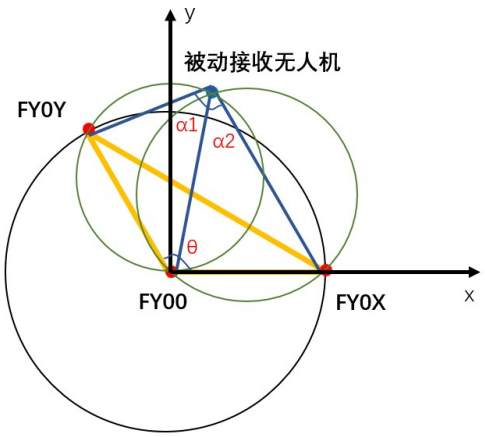

图 8: 问题一建立的模型

以 FY00 与 FY0X 所在直线为 x 轴，以 FY00 与 FY0Y 所在直线为 y 轴，在平面上建立平面直角坐标系。我们分别考虑两个轨迹圆，并分析各轨迹圆属于两个两种情况中的哪种情况，赋予正确的解析坐标式。由观测到的 α_1、α_2、α_3 的相对位置和大小关系可以确定被动接收信息无人机在弦的上方还是下方，从而解决圆的多解问题，在被动接收信息无人机中分析信息来源：

（1）若 FY00 的信息来源位于 FY01 的顺时针方向，则由 FY0Y、FY00、被动接收信号无人机所构成的轨迹圆的方程为：

$$(x - \frac{50\sin(\theta+\alpha_1)}{\sin(\alpha_1)})^2 + (y + \frac{50\cos(\theta+\alpha_1)}{\sin(\alpha_1)})^2 = (\frac{50}{\sin(\alpha_1)})^2 \tag{2}$$

（2）若 FY00 的信息来源位于 FY01 的逆时针方向，则由 FY0Y、FY00、被动接收信号无人机所构成的轨迹圆的方程为：

$$(x + \frac{50\sin(\theta-\alpha_1)}{\sin(\alpha_1)})^2 + (y - \frac{50\cos(\theta-\alpha_1)}{\sin(\alpha_1)})^2 = (\frac{50}{\sin(\alpha_1)})^2 \tag{3}$$

（3）若 FY00 的信息来源位于 FY01 的逆时针方向，则由 FY0Y、FY00、被动接收信号无人机

所构成的轨迹圆的方程为：

$$(x-50)^2+(y-\frac{50}{\tan(\alpha_2)})^2=(\frac{50}{\sin(\alpha_2)})^2 \tag{4}$$

（4）若 FY00 的信息来源位于 FY01 的逆时针方向，则由 FY0Y、FY00、被动接收信号无人机所构成的轨迹圆的方程为：

$$(x-50)^2+(y+\frac{50}{\tan(\alpha_2)})^2=(\frac{50}{\sin(\alpha_2)})^2 \tag{5}$$

5.1.2 模型的求解

通过定弦定角模型所确定的两个轨迹圆组合共有 4 种情况，因此我们选择公式 2与公式 4所确定的轨迹圆进一步分析两圆交点解的计算。

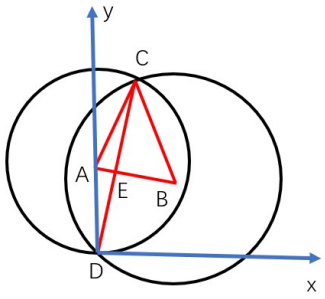

图 9: 两个圆的交点模型

记两个轨迹圆的圆心为 (x_1,y_1) 与 (x_2,y_2)，可以得到：

$$(x_1,y_1)=(50,\frac{50}{\tan(\alpha_2)})$$

$$(x_2,y_2)=(\frac{50\sin(\theta+\alpha_1)}{\sin(\alpha_1)},-\frac{50\cos(\theta+\alpha_1)}{\sin(\alpha_1)})$$

记 K_1 为两个圆心连线形成直线的斜率，可以得到:

$$\begin{cases} K_1=\frac{y_2-y_1}{x_2-x_1}==\dfrac{\sin\alpha_1\cos\alpha_2+\sin\alpha_2\cos(\theta+\alpha_1)}{\sin\alpha_2(\sin\alpha_1-\sin(\theta+\alpha_1))} \\ K_2=-\frac{1}{K_1}=-\dfrac{1}{K1}=\dfrac{\sin\alpha_2(sin(\theta+\alpha_1)-\sin\alpha_1)}{\sin\alpha_1\cos\alpha_2+\sin\alpha_2\cos(\theta+\alpha_1)} \\ L=AB=\sqrt{(x_1-x_2)^2+(y_2-y_1)^2} \end{cases} \tag{6}$$

又由几何关系

$$\begin{cases} CE^2=r_1^2-AE^2 \\ CE^2=r_2^2-EB^2 \\ EB^2=(L-AE)^2 \end{cases} \tag{7}$$

解得：

$$AE = \frac{(r_1^2 - r_2^2 + L^2)}{2L}$$

从而得到 E 点坐标：

$$\begin{cases} x_0 = x_1 + \frac{AE}{L}(x_2 - x_1) \\ y_0 = y_1 + \frac{AE}{L}(y_2 - y_1) \end{cases} \tag{8}$$

由于 D 点所处位置为圆心，因此交点 C 的坐标为 E 坐标的两倍，即为：

$$\begin{cases} x_c = 2(x_1 + \frac{AE}{L}(x_2 - x_1)) \\ y_c = 2(y_1 + \frac{AE}{L}(y_2 - y_1)) \end{cases} \tag{9}$$

最后根据所选取的无人机编号差可以确定所形成的圆圆心角确定 θ 的值，x_c 与 y_c 中的参数 x_1、AE、L 均在公式 (6)-(8) 中利用 α_1、α_2 与 θ 来表示。从而将所有的参数代入表达式中，可以确定最后的接收信号无人机的定位。

5.2 问题二：遍历优化算法解决在发送无人机编号未知情况下接收信号无人机方位解算问题

5.2.1 模型的建立

首先，我们对除了 FY00 与 FY01 的发射信号无人机仅有一架的情况进行建模分析。

如图 10所示，固定 FY00 与 FY01 两点，对剩余的 8 个点进行遍历分析，FY00、FY01、任一发射信号无人机便成为已知条件。对于三个已经确定编号的无人机，可以代入问题一种所建立的定位模型。接收无人机所接收到的方位信息固定，利用情况枚举来确认是否有多解性的可能。图中与每一个蓝点连线的红点代表经过模型一后输出的接收信号无人机的坐标。从图中可知，通过选用不同编号的无人机，对应的预测点有多重情况且无法被排除。因此仅有三架无人机的模型缺少信息维度，无法准确确定目标无人机位置。

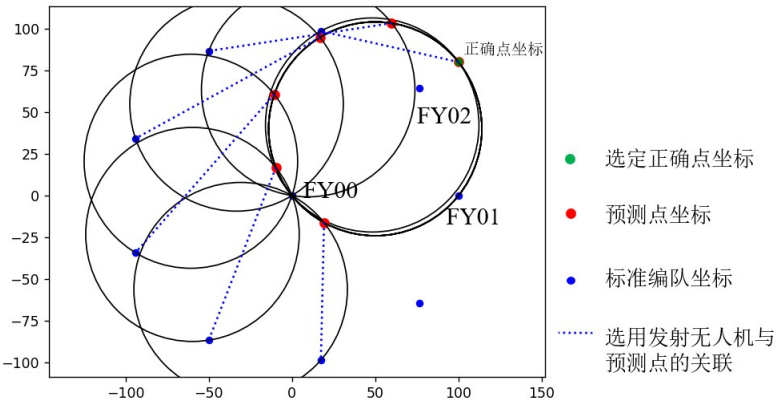

图 10: 基于 3 架无人机出现多解的情况

在排除三架无人机实现准确定位情况的背景下，我们选用四架无人机发射信号，即选用 FY00、FY01 与圆周上任意选取两个编号未知的无人机来接收消息。在圆周上选用两个编号未知的无人机共有 $C_8^2 = 28$ 种情况。

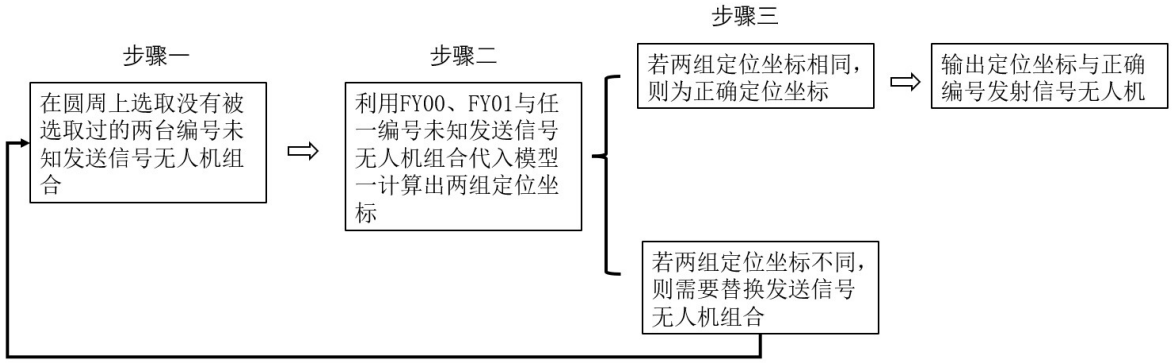

图 11: 模型二流程图

如流程图 11所示，根据平面上三个不重叠且相交的圆唯一确定一个点的定理，我们通过遍历 28 种可能情况，可以得到四个圆，利用圆所确定点的相交性确定正确点的坐标与正确编号组合：

步骤一：在圆周上任意选取两个无人机作为发射信号无人机

如图 12所示，从圆周上任意选取两台无人机（除去已经选择的 FY01）共有 28 种情况，假设选取的无人机的编号为 FY0X 与 FY0Y。

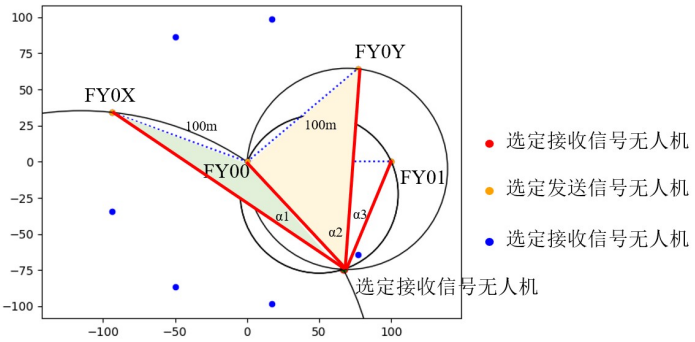

图 12: 模型二

步骤二：利用模型一求解预测坐标

当选定发射无人机的编号时，场景中的无人机共构成两组模型一。我们将 4 架无人机组合成 FY00、FY01、FY0X 的组合与 FY00、FY01、FY02 的组合。由于无人机接收到的角度方位信息确定，因此利用问题一中求解出的定弦定角模型一可以求解出预测无人机在全局位置信息：

FX00、FY01、FY0X 所构建的模型一的角度信息为 α_1，由于编号 FY01 与 FY0X 已知，因此可以通过模型一求解出预测点一位置记为：

$$(x_{p1}, y_{p1}) = Model_1(FY00, FY0X, \alpha_1) \tag{10}$$

FX00、FY01、FY0Y 所构建的模型一的角度信息为 α_2，由于编号 FY01 与 FY0Y 已知，因此可以通过模型一求解出预测点位置记为：

$$(x_{p2}, y_{p2}) = Model_1(FY00, FY0Y, \alpha_2) \tag{11}$$

步骤三：对比两组预测位置，确定预测无人机真实位置

比较步骤二解算出的位置信息，若位置信息相同，即

$$\begin{cases} x_{p1} = x_{p2} \\ y_{p1} = y_{p2} \end{cases} \tag{12}$$

则该点为正确编号的无人机组合所计算出的接收信号无人机的定位信息。

如果两者位置信息不相同，则说明选取的无人机组合不正确，需要重新进入步骤一选取其余未尝试过的无人机编号组合，直至求解出正确的编号组合。

5.2.2 模型的验证

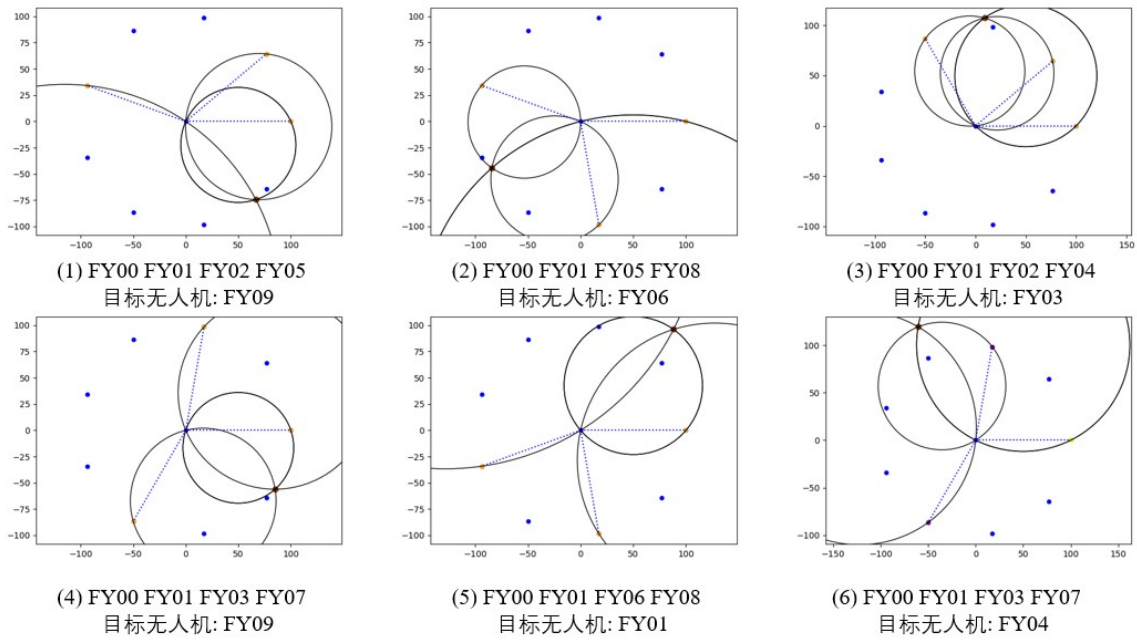

图 13: 模型二中验证不同情况示意图

步骤一：建立验证模型

我们随机选取一架圆周上的无人机并对无人机进行位置在标准编队附近随机偏移处理，作为位置略有偏差的无人机 A。对于选取的点，我们规定无人机在横纵坐标偏移为 $\pm 6m$ 的矩形区间内进行随机数偏移。

接着在圆周上选取其余两架无人机作为编号未知的发射信息无人机，记为 FY0X 与 FY0Y。计算 A 与 FY0X 与 FY0Y 的夹角角度作为已知的角度信息。

步骤二：利用验证模型进行模型检验

在验证阶段，将角度信息代入上述建立模型，如图 13所示为部分不同情况的示意图，图中橙色点代表随机选取的编号未知的无人机，蓝色点代表，而绿色点与红色点重合就代表选取接收信息的无人机的定位与仅通过角度信息经过模型输出的定位坐标相重合。

同时我们也验证 1000 组不同情况，最终得出结果均能与模型假设匹配。

5.3 问题三：利用优化人工势场模型迭代解决无人机位置调整问题

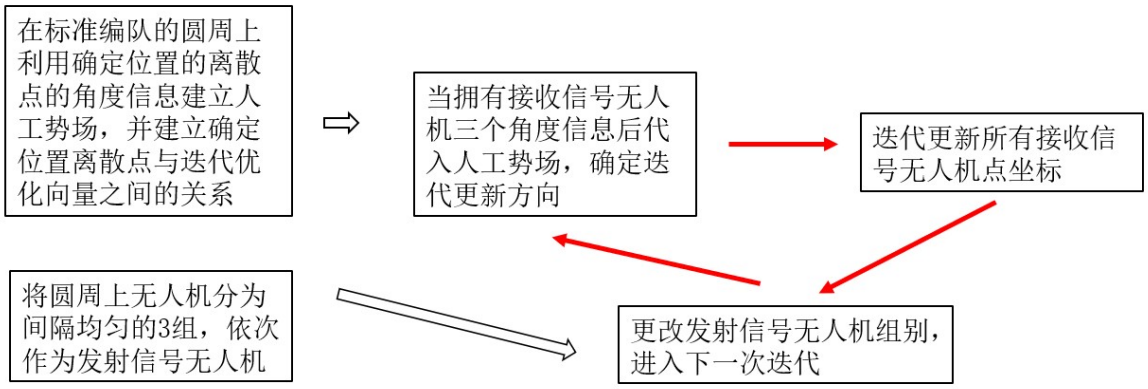

图 14: 模型三流程图

在除了 FY00 以外的无人机位置均有偏差的情况下，无法利用距离信息进行无人机确切位置的解算，由于只有角度信息，没有距离信息。因此缺少必要条件数量，只考虑优化模型对接收信号无人机进行位置的最优化处理。

本文需要构建迭代优化模型，对无人机集群整体进行调整。在模型的每一次迭代过程中，需要确定迭代的目标与在当前状态下模型迭代的方向。由于有效信息仅有三个角度方位，因此需要构建势场模型将三个角度映射到势场三维空间中的坐标点。通过势场的梯度信息确定模型的迭代方向，再将模型迭代方向从势场中映射回二维的平面集合空间，从而确定无人机的飞行方向。

由于在问题三中用于构建势场的无人机并非处于标准编队位置，因此我们需要在每次移动无人机的时候改变发射信号无人机的编号，从而实现所有无人机位置的更新，具体模型与算法实现原理如下：

5.3.1 建立标准位置的势场模型

第一部分：人工势场原理分析

人工势场包括引力场与斥力场，其中目标点对物体产生引力，引导物体朝向目标点运动；而障碍物对物体产生斥力，避免物体与之相撞，物体在路径上每一点的受力均为引力场与斥力场对物体吸引力的合力。[2]

对于引力场而言，常用的引力场函数为

$$U_{att}(q) = \frac{1}{2}\xi\rho^2(q, q_{goal}) \tag{13}$$

其中，ξ 是尺度因子，$\rho(q,q^2)$ 代表物体当前状态与目标位置之间的距离。

有了引力场，物体受到的引力记为引力场的导数

$$F_{att}(q) = -\nabla U_{att}(q) \tag{14}$$

在本问题中，由于飞机可以向任意方向移动，故不存在障碍物，也就不存在斥力场的影响。[3]

第二部分：改进优化人工势场的机理分析

根据无人机仅能接收到角度信息，没有方位定位信息的题目条件，经过测试无法通过足够有效的约束条件，直接进行人工势场方程构建势场的函数。因此我们提出基于改进优化人工势场的模型，该模型基于离散数据点信息，并对离散数据点中间的区域进行二维线性差值处理。从而模拟人工势场，让无人机在通过接收无人机三个角度信息的情况下能够通过差值拟合函数求取迭代更新方向与距离，从而实现无人机位置的一次迭代。

第三部分：人工势场分区分点构建

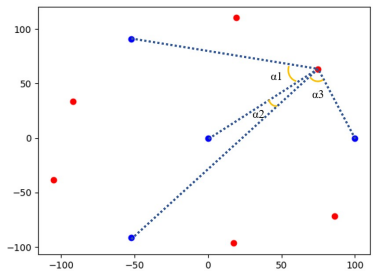

图 15: 观测角度示意图

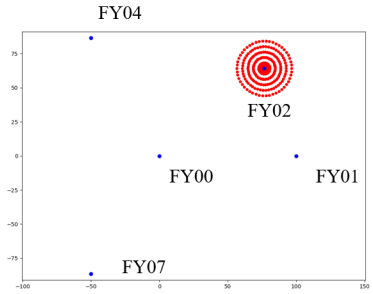

图 16: 在标准编队位置周围构建分区与分点

首先，将接收信息无人机所收集的三个角度方位信息转化为空间中 R^3 中的坐标，如图 15所示，我们将 α_1、α_2、α_3 的坐标作为空间中三维坐标点 $(x_\alpha, y_\alpha, z_\alpha)$。

然后，如图 16所示，针对标准编队（即 9 架无人机均匀分布在半径为 100m 的圆周上）所形成的圆周的每一个点，我们分别在每一个点处以 0 米、0.5 米、1 米、2 米、4 米、8 米、12 米、16 米、20 米为半径形成的圆周上取点，并搜集有效角度信息，将此角度信息以上述方式映射至三维空间 R^3 中，记录该点的最优更新方向，形成如图 17所示的以角度信息构建的人工势场点云模型。

第四部分：迭代更新向量的计算

根据二维线性差值原理，我们需要在映射空间中寻找与 $(x_\alpha, y_\alpha, z_\alpha)$ 最近的四个点，并将此四点在极坐标系中指向中心的向量加权平均得到迭代更新向量。

由于需要做二维线性差值，因此需要将图 17中的点云和横轴与纵轴进行归一化处理，通过三维空间中仿射基底变化实现：

设 E_1, E_2 为两个维度相同的几何向量空间，称从 E_1 到 E_2 的一个映射为仿射映射，若该映射是由一个非奇异的线性变换和一个平移变换所组成。

设 $X_1 \in E_1, X_2 \in E_2, A \in M_n(\mathbb{R}), b \in M_{n,1}$ 为平移变换列矩阵，其中 n 为向量空间的维度。那么从从 E_1 到 E_2 的一个仿射映射可以记作

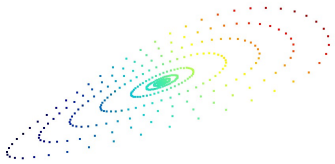

图 17: 通过角度构建的人工势场点云模型

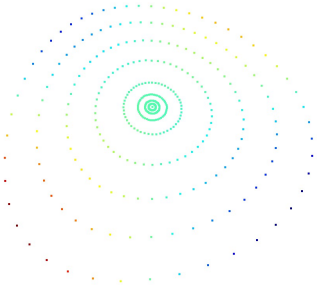

图 18: 经过仿射变化调整后的人工势场点云模型

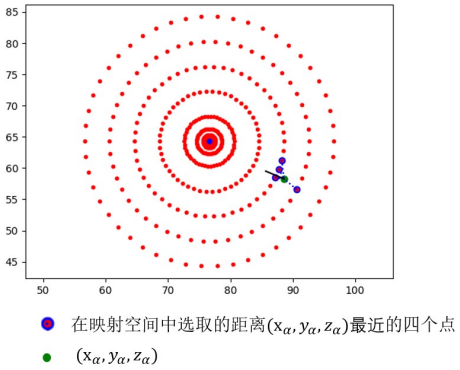

图 19: 通过距离最近四点计算迭代更新向量示意图

$$X_1 = AX_2 + \mathbf{b} \tag{15}$$

由三维空间的仿射变换公式可以写成如下的矩阵表示形式:

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{pmatrix}$$

对于本模型而言，通过比较最外层点云两两之间的距离可得到两个点的最长距离，以此两个点为端点构建向量 $\overrightarrow{MaxVec}$; 得到两个点的最短距离，以此两个点为端点构造向量 $\overrightarrow{MinVec}$，以此两个向量与平面法向量 $\vec{n}$ 组成的基底与标准坐标系的基底进行基底过渡矩阵计算，b 的向量即为从选定的 FY0X 指向 $(0,0,0)$ 的向量:

$$A = \begin{pmatrix} \overrightarrow{MinVec} \\ \overrightarrow{MaxVec} \\ \vec{n} \end{pmatrix}, \quad b = \begin{pmatrix} x_{FY0X} \\ y_{FY0X} \\ z_{FY0X} \end{pmatrix}$$

经过处理后得到如图 18所示的标准映射空间。在此空间中需要寻找 $(x_\alpha, y_\alpha, z_\alpha)$ 距离最近的四个点，采用 KD 树模型的方法进行求解：

针对本模型构建 $KD-tree$ 的原理

已知人工利用角度信息构建的三维映射空间中无序化的 3 维点云，每一个点云节点都是 3 维的数据。

1、初始化分割轴：分别计算被接受信号无人机周围的点 $(\alpha_x, \alpha_y, \alpha_z)$ 对应 x, y, z 坐标角度信息的方差（下面以 x 轴方向上的数据计算方差为例）

$$s^2 = \frac{\sum_{i=1}^{2}(\alpha_{xi} - \overline{\alpha_x})^2}{n}$$

取具有最大方差的坐标轴作为初始分割轴，记作 r_1；

2、确定分割节点：对现有的角度信息按照分割轴维度进行检索，将角度信息从小到大排序后选出中位数数据，并将其当作是分割点，这个节点的分割平面就是通过该点并垂直于具有最大方差坐标轴的一个二维平面。

3、划分双子空间：在当前分割轴中，所有小于中位数的角度信息全部划分到左子空间中；在当前分割轴中，所有大于等于中位数的角度信息全部划分到右子空间中。

4、更新分割轴：利用公式 $r = (r+1)\ \%\ n$ 求出此三维映射空间中的新分割轴。

5、确定子节点：在左子空间的角度信息中进行步骤 2 确定左子节点；在右子空间的数据中进行步骤 2 确定右子节点。

同时将子空间和角度信息进一步划分，如此反复进行上述步骤直到子空间中只包含一个角度信息点 $(\alpha_x, \alpha_y, \alpha_z)$。

这样一来就可以将整个三维映射空间划分成了若干个子空间。考察被接收信号的无人机映射到此空间上的坐标 $(\alpha_1, \alpha_2, \alpha_3)$，以该坐标为中心点，利用 $KD-tree$ 向周围附近空间进行检索，便可找到距离坐标 $(\alpha_1, \alpha_2, \alpha_3)$ 最近的其它四个点的坐标。

通过这四点的坐标，根据在建立人工势场时存储的迭代更新向量可以寻找到对应的四个最近点的的迭代更新向量，记为 $\bar{v}_1 \quad \bar{v}_2 \quad \bar{v}_3\bar{v}_4$。因此，针对 $(x_\alpha, y_\alpha, z_\alpha)$ 所构建的迭代更新向量为：

$$\overrightarrow{\Delta v} = (\bar{v}_1 + \bar{v}_2 + \bar{v}_3 + \bar{v}_4)/4 \tag{16}$$

5.3.2 无人机位置迭代策略

由于圆周上 9 架无人机的位置均不准确，因此在迭代更新的过程中，每一架无人机均需要更新位置。因此，需要制定合理迭代策略以满足模型收敛需求。

步骤一：确定发射信号无人机编号

根据题意，接收信号无人机仅能接收到发送信号无人机的方位角度信息。考虑到势场建立所需要的角度信息对称性，我们在选取发射信号无人机编号时同样考虑模型的对称性。因此，我们将圆周上的 9 架无人机划分为 3 组，分别是：

步骤 2：接收信号无人机获取角度信息，计算迭代更新向量

当有 4 台发射信号无人机时，圆周上其余 6 台为接收信号无人机。每一台接收信号无人机根据自己的获取的 3 个角度信息，通过代入 5.3.1 节中讨论的优化人工势场模型确定各自的迭代更新变量 $\Delta v = (\Delta\rho, \Delta\theta)$，在接受信号无人机端的自身极坐标下更新坐标。

表 1: 圆周上无人机分组表

组合编号	发射信号无人机编号
A 组	FY00、FY01、FY04、FY07
B 组	FY00、FY02、FY05、FY08
C 组	FY00、FY03、FY06、FY09

步骤 3：更换圆周无人机组别，继续进行迭代

在一次迭代过后，接受信号无人机在发射信号无人机的观测条件下已经达到最优情况，然而发射信号无人机的位置略有偏差，因此我们需要更换发射信号的无人机使得接收信号无人机与发射信号无人机之间能够相互迭代，以达到全局均为最优解的情况。

目标函数为:

$$F(x) = \Sigma_{i=0}^{9} \left| R_{FY0i} - \bar{R} \right| \quad \bar{R} = \frac{\Sigma_{i=0}^{9} R_{FY0i}}{9} \tag{17}$$

其中，x 为迭代次数，$\bar{R}$ 为圆周上点的平均半径

因此我们按照 A 组->B 组->C 组->A 组的顺序循环迭代，使得每一次都更新不同架次的无人机，从而达到全局最优，目标函数最小化。

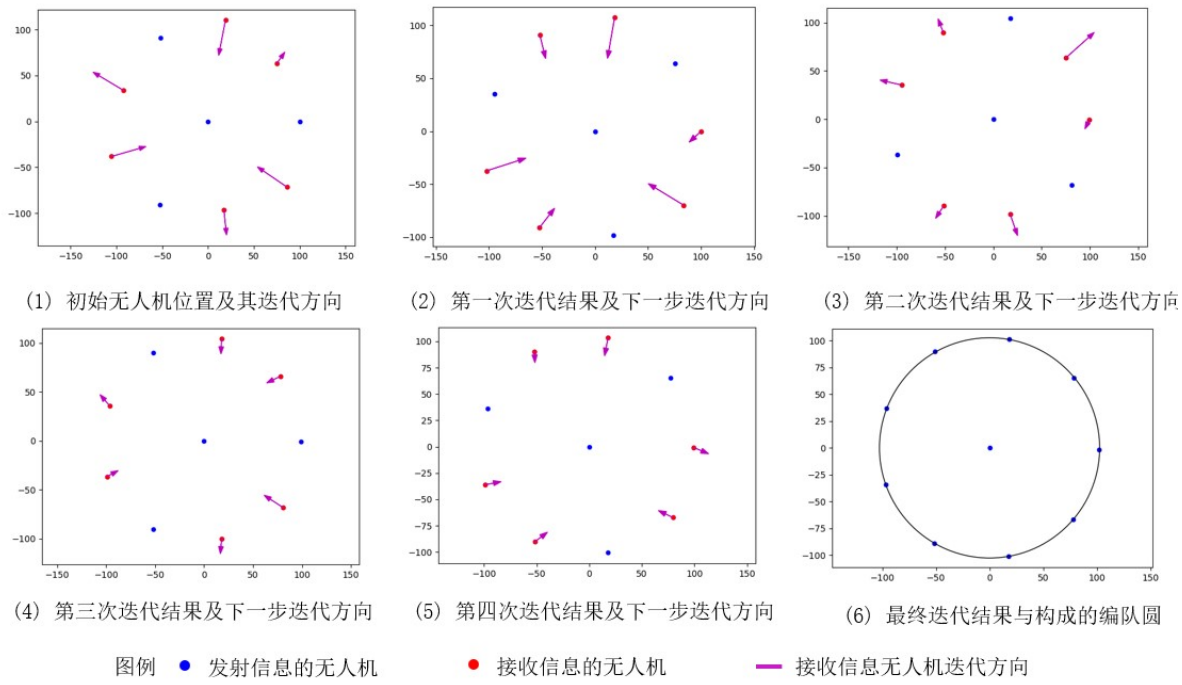

图 20: 无人机位置调整迭代过程图（为增强模型可视化，箭头长度增长 10 倍）

5.3.3 无人机位置迭代求解过程

将题目中给定的所有无人机的初始位置位置进行建模，通过角度计算可得出各接收信息无人机所接收到的三个角度信息。

将角度信息代入无人机迭代更新模型，求解如出表 2所示的无人机位置更新向量。其中，表2中的向量为基于接收信息无人机极坐标系的更新向量。

将更新向量代入上述建立的迭代更新方案模型，求解出如图 20所示的迭代更新方案。

最终迭代 10 次即可获得目标函数误差小于 0.5 的结果，迭代 50 次可以获得目标函数误差小于 0.3 的结果。

表 2: 模型迭代次数与各无人机更新迭代向量（极坐标）关系表

迭代	1	2	3	4	5
*FY*00	(0.000, 0.000)	(0.000, 0.000)	(0.000, 0.000)	(0.000, 0.000)	(0.000, 0.000)
*FY*01	(0.000, 0.000)	(0.789, 110.8)	(0.791, 131.1)	(0.000, 0.000)	(0.316, 216.8)
*FY*02	(3.167, 260.1)	(0.000, 0.000)	(0.556, 27.8)	(0.790, 10.89)	(0.000, 0.000)
*FY*03	(1.583, 283.7)	(1.575, 167.1)	(0.000, 0.000)	(0.790, 37.04)	(0.449, 288.1)
*FY*04	(0.000, 0.000)	(0.790, 237.0)	(0.791, 264.6)	(0.000, 0.000)	(0.317, 345.2)
*FY*05	(3.158, 17.87)	(0.000, 0.000)	(1.585, 146.9)	(0.792, 154.5)	(0.000, 0.000)
*FY*06	(1.583, 52.17)	(1.582, 289.1)	(0.000, 0.000)	(0.791, 337.4)	(0.791, 24.29)
*FY*07	(0.000, 0.000)	(0.317, 244.4)	(0.792, 205.9)	(0.000, 0.000)	(0.236, 39.90)
*FY*08	(3.167, 148.6)	(0.000, 0.000)	(0.792, 267.6)	(0.996, 258.9)	(0.000, 0.000)
*FY*09	(0.792, 222.4)	(3.157, 41.86)	(0.000, 0.000)	(0.317, 272.1)	(0.316, 198.5)

5.3.4 无人机位置迭代结果分析

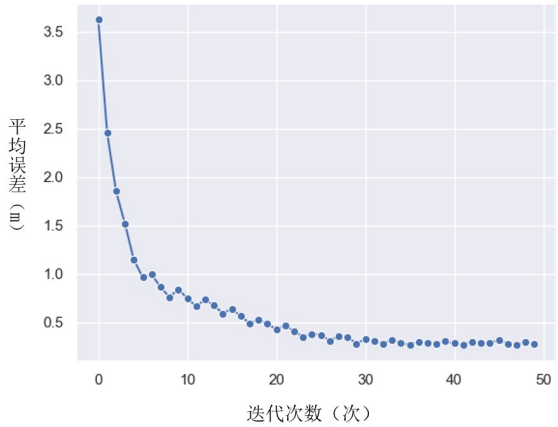

图 21: 无人机位置迭代中平均误差与迭代次数关系图

图 21的横坐标是算法模型迭代的次数，纵坐标是每一次迭代过程中，圆周点与所形成的拟合圆的平均误差。

图 21所示的结果为对这个算法模型迭代 50 次时的误差值 F(x)(见公式 (17))。从图中可以发现此算法在前 6 次迭代过程中，误差下降的速率非常快，从 3.622m 直接下降到了 0.922m。说明该算

法模型可以在迭代次数很小时就表现出强的收敛性，这是该算法的优点。可以看到，在经过 20 次迭代后可以把误差降到了 0.5m 以内，在经过 30 次迭代，误差收敛于 0.3。

5.4 问题四：在锥形无人机编队的情况下，考虑调整算法最优情况

5.4.1 模型的建立

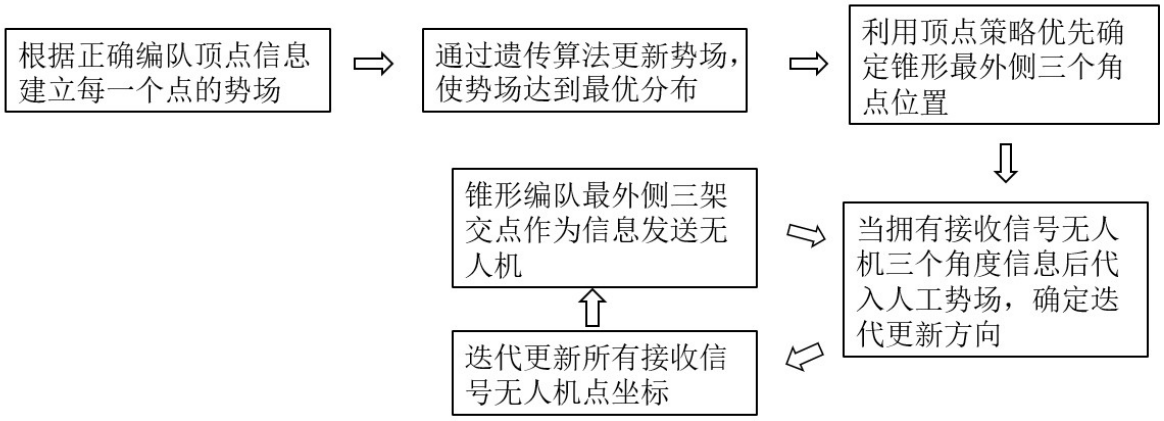

图 22: 锥形无人机编队调整迭代过程图

由图 22可知，延续模型三中建立正确点位置势场的想法，建立编队正确位置的优化人工势场模型。由于锥形图形没有圆形图形一样每一条通过圆心的直线都是对称轴的性质，因此当接收信息的无人机位置有较大偏离时，势场的中心往往不是无人机位置迭代更新的最优解。因此，我们需要对此情况进行特殊处理。本文提出利用遗传算法优化来初始化势场，以保证势场中的矢量指向达到最优分布，提高了模型的收敛性的同时也减小了模型的最终误差。

而对于锥形中发射信号无人机选择的问题，本文提出顶点策略的想法，

第一部分：顶点策略

由于锥形最外层角点离其它点距离较远，是无人机点集的一个凸包, 因此更适合作为发射信息的点。从由顶点向其它点修正信息，首先要保证顶点的位置相对准确。我们让顶点依次发送信息，这样每个顶点能获得其余两个点的方向。我们只需根据两个点的夹角，如果夹角比 60 度大，则前进，否则后退。

$$\begin{cases} \alpha_1 < 60, \textbf{无人机沿着}\alpha_1\textbf{角平分线且使}\alpha_1\textbf{角变大的方向运动} \\ \alpha_1 = 60, \textbf{无人机在瞬时时刻不运动} \\ \alpha_1 > 60, \textbf{无人机沿着}\alpha_1\textbf{角平分线且使}\alpha_1\textbf{角变小的方向运动} \end{cases}$$

借鉴上一问的思路，模拟其余两个顶点位置准确切此顶点的移动方向为两发射源的平均方向，计算不同位置该点所需移动的距离和夹角大小的关系，采用一维线性插值获得移动距离和夹角的方程即可。根据我们的测试，这种方法能让顶点收敛到 $1e-16$ 的精度。

在校准顶点后，每个点都可根据顶点的角度信息进行调整。同样建立人工势场。我们只需要对边上的 2 种点，内部的一种点建立势场模型。

第二部分：利用遗传算法自适应更新势场分布

模型三中建立的势场的迭代更新向量均指向与正确编队位置的圆心处，但是若在锥形编队中无人机偏离正确位置过远时，仅基于优化人工势场的进行迭代向量更新时，会产生一部分不可继续优化的误差，因此我们想利用遗传优化算法来更改势场中固定向量的指向。优化其指向使其不再指向正确位置圆心处，而是指向经过遗传算法训练后迭代更新正确概率最大的一点。

遗传算法的训练步骤如下所示：

1 固定人工选取的模拟势场的点位置不变，将每个点的势场强度向量代入遗传算法迭代，作为初始种群

2 进化过程分为遗传和变异。选择若干向量更改之，即为变异。选择某点的向量设为种群中其它个体此位点的向量，即为遗传。将根据上一代信息获得遗传变异后的个体加入种群。

3 选取 loss 函数为各点临边（共 30 条）的累积误差作为目标函数，每次随机偏移每个飞行器位置作为训练数据，根据每个个体的势场迭代更新 30 此后计算 loss

4 选取新种群目标函数前 50% 小的个体作为新种群

5 重复上述 234 步骤，待 loss 基本收敛

6 将结果输出

5.4.2 模型的求解

我们利用 python 程序（位于附录 problem4.py 中）在平面上依据正确图形随机生成变化范围在指定区间内的随机模型。

首先我们在正确的编队位置依据 5.3 中的人工势场模型建立势场。其次我们利用经过训练的遗传算法自适应势场分布，改变势场中迭代更新的向量方向与向量值，使之分布能使最终结果最优。

其次我们利用顶点策略更新锥形最外侧的三个点，使之能够利用相互之间的角度信息更新自身位置。在顶点更新到位后，转变为信号发射无人机。其余锥形边上点与锥形中的点作为信息接收点，通过正确编队中的势场确定迭代更新变量，从而实现逐步迭代更新，直至到达最优位置。

利用程序模拟锥形无人机编队调整与定位情况如图 23所示：

从图 23中也可以看出：利用本模型进行迭代更新，在 6 次迭代过后能够将原始误差下降 10 倍，到达 0.012767 的误差率。

六、模型的分析与检验

本文对问题三所构建的调整模型进行模型灵敏度分析。由问题三的分析可知，无人机初始点位置偏离正确位置的大小会对误差造成影响。因此我们通过对初始位置偏移最大值进行设定，在偏移最大值范围内产生测试点数据，将测试点数据代入模型三中进行求解。我们设定最大误差在 1m-10m 的范围之内，对范围内的测试点测试，并读取最终稳定误差值。

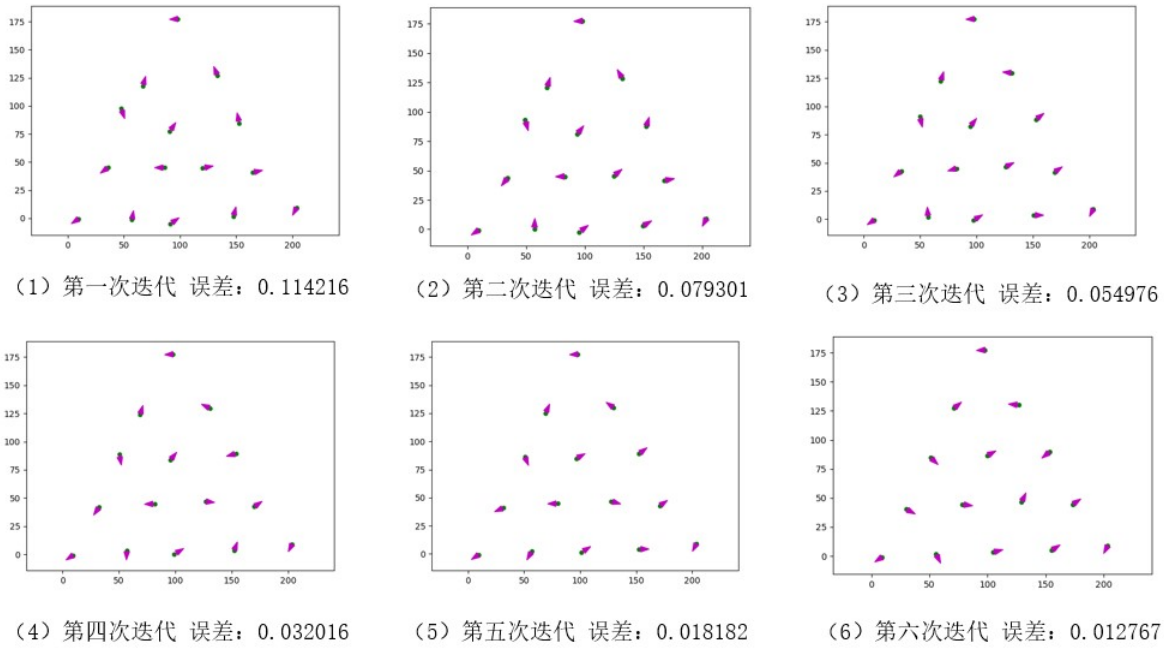

图 23: 锥形无人机编队调整迭代过程图

如图 24，调整后无人机编队的误差在 0.15 到 0.37m 之间波动，波动范围较小。在合理范围内增加产生点的位置偏差，模型所产生的误差在较小范围内波动，说明问题三所构建的模型稳定。

七、模型的评价、改进与推广

7.1 模型的优点

1. 在无人机距离目标越远时，调整的步长越大，因此在只有少数迭代次数的情况下就能快速收敛。

2. 本模型创新性的使用点云和 KD 树进行线性插值，使计算量大大降低。

3. 本模型的泛化性较强，可以用于不同形状的编队，而且只需要少数几个无人机发射信息即可做到所有无人机的调整。

4. 在手动建立的势场表现不好时还可使用遗传算法对势场做进一步优化。

7.2 模型的缺点

1. 本模型尚未对建立势场的选点调参，也未对遗传算法做响应调整。

2. 对于信息源的选取，也未探寻更优的策略。

3. 对于角度特征的映射只进行了简单的放射变换，角度特征在变换后的差异不够明显，导致选取临近点仍有误差。

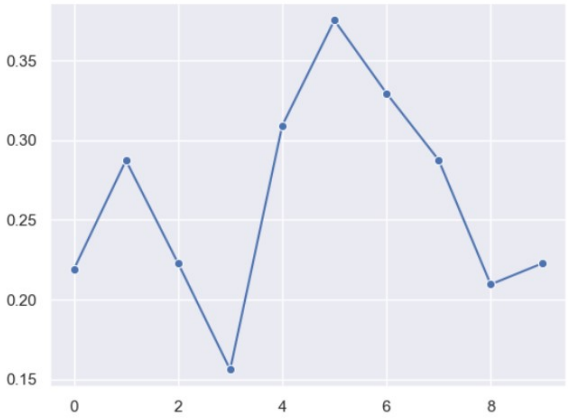

图 24: 敏感度分析图

7.3 模型的拓展与推广

1. 可以进一步探究发射无人机的选取和势场中预先选取点的指定，使调整精度更高。

2. 对于角度特征在空间中的映射，在不惜牺牲计算量的情况下可以使用更复杂的映射，甚至使用神经网络进行映射，使特征空间差异更加明显，更易进行向量检索，进一步提高精度。

3. 还可以使用强化学习更新势场，加速势场的训练。

7.4 模型泛化性

本文中仅把旋转对称的图形和规则多边形编队做为研究对象. 事实上, 对于不规则的编队仍能提供解决方案. 对于一个不规则编队, 可以计算凸包作为顶点, 顶点之间通过分组建立顶点策略, 再以顶点为信息源建立每一个内点的势场. 在做调整时先通过顶点策略确定顶点微调步长, 向信息源构成三角形的外心方向进行微调. 随后顶点作为信息源向其余内点发送角度信息, 让内点在提前建立好的势场中进行调整. 循环此过程即可.

参考文献

[1] 多无人机被动目标定位与跟踪技术研究, 2017.

[2] Cezary Kownacki and Leszek Ambroziak. Local and asymmetrical potential field approach to leader tracking problem in rigid formations of fixed-wing uavs. *Aerospace Science and Technology*, 68:465–474, 2017.

[3] 张鲲, 沈重, 王海丰, 李壮, 高倩, and 李涵雯. 海上侦察船的纯方位无源定位技术研究. *舰船科学技术*, (2):19–21, 2018.

附录

附件清单		
1. 问题一主程序:	problem1.py	用于问题一的坐标解算与做图分析
2. 问题二主程序:	problem2.py	用于问题二验证坐标信息并做图分析
3. 问题三主程序:	problem3.py	用于问题三调整策略与调整策略展示
4. 问题四主程序:	problem4.py	用于问题四调整策略加遗传算法计算

Listing 1: **problem1.py 用于解决问题一中坐标解算与做图**

```python
from sympy import *
from collections import namedtuple
import math
import matplotlib.pyplot as plt
import random
import time
Point = namedtuple('Point', ['x', 'y'])
def addPoint(a: Point, b: Point):
    return Point(a.x + b.x, a.y + b.y)

Point.__add__ = addPoint

def subPoint(a: Point, b: Point):
    return Point(a.x - b.x, a.y - b.y)

Point.__sub__ = subPoint

def divf(x):
    if x == 0:
        return 1e-7
    return x

def dis(a: Point, b: Point):
    return math.sqrt((a.x - b.x) ** 2 + (a.y - b.y) ** 2)

def get_ang(p: Point):
    ang = math.atan(p.y / divf(p.x))
    if p.x < 0:
        ang += math.pi
    if ang < 0:
        ang += 2 * math.pi
    return ang

def get_ang3(p1: Point, p2: Point, p3: Point):
    a1 = get_ang(Point(p1.x - p2.x, p1.y - p2.y))
```

```
36      a2 = get_ang(Point(p3.x - p2.x, p3.y - p2.y))
37      return math.fabs(a2 - a1)
38
39  def roatate(p: Point, theta):
40      d = dis(p, Point(0, 0))
41      theta0 = get_ang(p)
42      return Point(d * math.cos(theta + theta0), d * math.sin(theta + theta0))
43
44  def calc_jiao(a: Point, b: Point, ra, rb):
45      l = dis(a, b)
46      if l > ra + rb:
47          return None
48      k1 = (b.y - divf(a.y)) / (b.x - divf(a.x))
49      k2 = - (b.x - divf(a.x)) / (b.y - divf(a.y))
50      x0 = a.x + (b.x - a.x) * (ra ** 2 - rb ** 2 + l ** 2) / 2 / l ** 2
51      y0 = a.y + k1 * (x0 - a.x)
52      r2 = ra ** 2 - (x0 - a.x) ** 2 - (y0 - a.y) ** 2
53      return Point(x0 - math.sqrt(r2 / (1 + k2 ** 2)), y0 - k2 * math.sqrt(r2 / (1 + k2 ** 2))), \
54             Point(x0 + math.sqrt(r2 / (1 + k2 ** 2)), y0 + k2 * math.sqrt(r2 / (1 + k2 ** 2)))
55
56  def show(a0, ap1, ap2):
57      a2, a1 =  abs(ap1 - a0),  abs(ap2 - a0)
58      theta = (p2 - p1) * 40 / 180 * math.pi
59
60      if ap1 >= a0:
61          a = Point(50, 50 / divf(math.tan(a2)))
62      else:
63          a = Point(50, -50 / divf(math.tan(a2)))
64
65      if a0 >= ap2:
66          b = Point(50 * math.sin(theta + a1) / divf(math.sin(a1)), -50 * math.cos(theta + a1) /
                divf(math.sin(a1)))
67      else:
68          b = Point(-50 * math.sin(theta - a1) / divf(math.sin(a1)), 50 * math.cos(theta - a1) /
                divf(math.sin(a1)))
69
70      ra, rb = math.fabs(50 / divf(math.sin(a2))), math.fabs(50 / divf(math.sin(a1)))
71      tmp = calc_jiao(a, b, ra, rb)
72      if tmp is not None:
73          c, d = tmp
74
75      for fly in flys:
76          plt.scatter(*fly, s=20, c='blue')
77      if tmp is not None:
78          plt.scatter(*c, s=30, c='red')
79          plt.scatter(*d, s=30, c='red')
```

```
80      plt.scatter(*target, s=20, c='green')
81      plt.plot([0, flys[p1].x], [0, flys[p1].y], 'b:')
82      plt.plot([0, flys[p2].x], [0, flys[p2].y], 'b:')
83      draw_circle = plt.Circle(a, ra, fill=False)
84      plt.gcf().gca().add_artist(draw_circle)
85      draw_circle = plt.Circle(b, rb, fill=False)
86      plt.gcf().gca().add_artist(draw_circle)
87      plt.axis('equal')
88      # plt.show()
89
90  r = 100
91  flys = [Point(0, 0)]
92  for i in  range(9):
93      ang = i * 40 / 180 * math.pi
94      flys.append(Point(r * math.cos(ang), r * math.sin(ang)))
95
96  while True:
97      # target = Point(random.randint(-200, 200), random.randint(-200, 200))
98      sel_p = random.choice(flys[1:])
99      target = sel_p + Point((random.randint(0, 2) * 2 - 1) * random.randint(80, 120) / 10,
100                             (random.randint(0, 2) * 2 - 1) * random.randint(80, 120) / 10)
101     p1 = 1
102     p2 = random.randint(2, 9)
103     print(target, p1, p2)
104     a0 = get_ang(flys[0] - target)
105     ap1 = get_ang(flys[p1] - target)
106     ap2 = get_ang(flys[p2] - target)
107     show(a0, ap1, ap2)
108     plt.pause(1)
109     plt.clf()
```

Listing 2: problem2.py 用于问题二中遍历算法与做图

```
1   from sympy import *
2   from collections import namedtuple
3   import math
4   import matplotlib.pyplot as plt
5   import random
6   import time
7
8   Point = namedtuple('Point', ['x', 'y'])
9
10  def addPoint(a: Point, b: Point):
11      return Point(a.x + b.x, a.y + b.y)
12
13  Point.__add__ = addPoint
14
```

```python
def subPoint(a: Point, b: Point):
    return Point(a.x - b.x, a.y - b.y)

Point.__sub__ = subPoint

def averPoint(points:  list[Point]):
    l =  len(points)
    x, y = 0, 0
    for i in  range(l):
        x += points[i].x
        y += points[i].y
    x /= l
    y /= l
    return Point(x, y)

def divf(x):
    if x == 0:
        return 1e-7
    return x

def dis(a: Point, b: Point):
    return math.sqrt((a.x - b.x) ** 2 + (a.y - b.y) ** 2)

def get_ang(p: Point):
    ang = math.atan(p.y / divf(p.x))
    if p.x < 0:
        ang += math.pi
    if ang < 0:
        ang += 2 * math.pi
    return ang

def get_ang3(p1: Point, p2: Point, p3: Point):
    a1 = get_ang(Point(p1.x - p2.x, p1.y - p2.y))
    a2 = get_ang(Point(p3.x - p2.x, p3.y - p2.y))
    return math.fabs(a2 - a1)

def roatate(p: Point, theta):
    d = dis(p, Point(0, 0))
    theta0 = get_ang(p)
    return Point(d * math.cos(theta + theta0), d * math.sin(theta + theta0))

def calc_jiao(a: Point, b: Point, ra, rb):
    l = dis(a, b)
    if l > ra + rb:
        return None
    k1 = (b.y - divf(a.y)) / (b.x - divf(a.x))
```

```
    k2 = - (b.x - divf(a.x)) / (b.y - divf(a.y))
    x0 = a.x + (b.x - a.x) * (ra ** 2 - rb ** 2 + l ** 2) / 2 / l ** 2
    y0 = a.y + k1 * (x0 - a.x)
    r2 = ra ** 2 - (x0 - a.x) ** 2 - (y0 - a.y) ** 2
    return Point(x0 - math.sqrt(r2 / (1 + k2 ** 2)), y0 - k2 * math.sqrt(r2 / (1 + k2 ** 2))), \
           Point(x0 + math.sqrt(r2 / (1 + k2 ** 2)), y0 + k2 * math.sqrt(r2 / (1 + k2 ** 2)))

def get_ans(a0, ap1, ap2, cross):  # corss: p2-p1
    a2, a1 =  abs(ap1 - a0),  abs(ap2 - a0)
    theta = cross * 40 / 180 * math.pi

    if ap1 >= a0:
        a = Point(50, 50 / divf(math.tan(a2)))
    else:
        a = Point(50, -50 / divf(math.tan(a2)))

    if a0 >= ap2:
        b = Point(50 * math.sin(theta + a1) / divf(math.sin(a1)), -50 * math.cos(theta + a1) /
            divf(math.sin(a1)))
    else:
        b = Point(-50 * math.sin(theta - a1) / divf(math.sin(a1)), 50 * math.cos(theta - a1) /
            divf(math.sin(a1)))

    ra, rb = math.fabs(50 / divf(math.sin(a2))), math.fabs(50 / divf(math.sin(a1)))
    tmp = calc_jiao(a, b, ra, rb)
    ans = None
    if tmp is not None:
        c, d = tmp
        if dis(c, Point(0, 0)) >= dis(d, Point(0, 0)):
            ans = c
        else:
            ans = d
    return ans, (a, ra), (b, rb)

r = 100
flys = [Point(0, 0)]
for i in  range(9):
    ang = i * 40 / 180 * math.pi
    flys.append(Point(r * math.cos(ang), r * math.sin(ang)))


def geta0ap1ap2(p1, p2):  # p2 > p1
    a0 = get_ang(flys[0] - target)
    ap1 = get_ang(flys[p1] - target)
    ap2 = get_ang(flys[p2] - target)
    return a0, ap1, ap2
```

```
105
106 while True:
107     # target = Point(random.randint(-200, 200), random.randint(-200, 200))
108     sel_p = random.choice(flys[1:])
109     target = sel_p + Point((random.randint(0, 2) * 2 - 1) * random.randint(80, 120) / 10,
110                            (random.randint(0, 2) * 2 - 1) * random.randint(80, 120) / 10)
111     p1 = 1
112     p2rand = random.sample( range(2, 9), 2)
113     p2rand =  sorted(p2rand)
114     std_a0, std_ap1, std_ap2 = geta0ap1ap2(p1, p2rand[0])
115     _, _, std_ap3 = geta0ap1ap2(p1, p2rand[1])
116     print(target, p2rand)
117     ans, ans_dis = None, 1e15
118     ans_p1, ans_p2 = None, None
119     ans_a1, ans_b1 = None, None
120     ans_a2, ans_b2 = None, None
121     for i in  range(2, 9):
122         for j in  range(i + 1, 9):
123             ans1, a1, b1 = get_ans(std_a0, std_ap1, std_ap2, i - p1)
124             ans2, a2, b2 = get_ans(std_a0, std_ap1, std_ap3, j - p1)
125             tmp_dis = dis(ans1, ans2)
126             if tmp_dis < ans_dis:
127                 ans_dis = tmp_dis
128                 ans = averPoint([ans1, ans2])
129                 ans_p1, ans_p2 = i, j
130                 ans_a1, ans_b1 = a1, b1
131                 ans_a2, ans_b2 = a2, b2
132
133     for fly in flys:
134         plt.scatter(*fly, s=20, c='blue')
135     if ans is not None:
136         plt.scatter(*ans, s=40, c='red')
137     plt.scatter(*target, s=20, c='green')
138     plt.scatter(*flys[1], s=20, c='yellow')
139     plt.scatter(*flys[p2rand[0]], s=20, c='yellow')
140     plt.scatter(*flys[p2rand[1]], s=20, c='yellow')
141     plt.plot([0, flys[p1].x], [0, flys[p1].y], 'b:')
142     plt.plot([0, flys[ans_p1].x], [0, flys[ans_p1].y], 'b:')
143     plt.plot([0, flys[ans_p2].x], [0, flys[ans_p2].y], 'b:')
144     draw_circle = plt.Circle(ans_a1[0], ans_a1[1], fill=False)
145     plt.gcf().gca().add_artist(draw_circle)
146     draw_circle = plt.Circle(ans_b1[0], ans_b1[1], fill=False)
147     plt.gcf().gca().add_artist(draw_circle)
148     draw_circle = plt.Circle(ans_a2[0], ans_a2[1], fill=False)
149     plt.gcf().gca().add_artist(draw_circle)
150     draw_circle = plt.Circle(ans_b2[0], ans_b2[1], fill=False)
```

```
    plt.gcf().gca().add_artist(draw_circle)
    plt.axis('equal')
    # plt.show()
    plt.pause(2)
    plt.clf()
```

Listing 3: problem3.py 用于问题三中迭代算法求解

```
import numpy as np
import matplotlib.pyplot as plt
from collections import namedtuple
import math
import random
import open3d as o3d
import copy


dataset_num = 2000
max_pos_err = 6
generate_rates = [0, 0.5, 1, 2, 4, 8, 12, 16, 20]
generate_num = 50
near_choose = 4

Point = namedtuple('Point', ['x', 'y'])


def addPoint(a: Point, b: Point):
    return Point(a.x + b.x, a.y + b.y)


Point.__add__ = addPoint


def subPoint(a: Point, b: Point):
    return Point(a.x - b.x, a.y - b.y)


Point.__sub__ = subPoint


def divf(x):
    if x == 0:
        return 1e-7
    return x


def roatate(p: Point, theta):
```

```
    d = dis(p, Point(0, 0))
    theta0 = get_ang(p)
    return Point(d * math.cos(theta + theta0), d * math.sin(theta + theta0))


def get_ang(p: Point):
    ang = math.atan(p.y / divf(p.x))
    if p.x < 0:
        ang += math.pi
    if ang < 0:
        ang += 2 * math.pi
    return ang


def dis(a: Point, b: Point):
    return math.sqrt((a.x - b.x) ** 2 + (a.y - b.y) ** 2)


def calc_jiao(a: Point, b: Point, ra, rb):
    l = dis(a, b)
    if l > ra + rb:
        return None
    k1 = (b.y - divf(a.y)) / (b.x - divf(a.x))
    k2 = - (b.x - divf(a.x)) / (b.y - divf(a.y))
    x0 = a.x + (b.x - a.x) * (ra ** 2 - rb ** 2 + l ** 2) / 2 / l ** 2
    y0 = a.y + k1 * (x0 - a.x)
    r2 = ra ** 2 - (x0 - a.x) ** 2 - (y0 - a.y) ** 2
    return Point(x0 - math.sqrt(r2 / (1 + k2 ** 2)), y0 - k2 * math.sqrt(r2 / (1 + k2 ** 2))), \
           Point(x0 + math.sqrt(r2 / (1 + k2 ** 2)), y0 + k2 * math.sqrt(r2 / (1 + k2 ** 2)))


def get_ans(a0, ap1, ap2, cross):  # corss: p2-p1
    a2, a1 =  abs(ap1 - a0),  abs(ap2 - a0)
    theta = cross * 40 / 180 * math.pi

    if ap1 >= a0:
        a = Point(50, 50 / divf(math.tan(a2)))
    else:
        a = Point(50, -50 / divf(math.tan(a2)))

    if a0 >= ap2:
        b = Point(50 * math.sin(theta + a1) / divf(math.sin(a1)), -50 * math.cos(theta + a1) /
            divf(math.sin(a1)))
    else:
        b = Point(-50 * math.sin(theta - a1) / divf(math.sin(a1)), 50 * math.cos(theta - a1) /
            divf(math.sin(a1)))
```

```
84
85      ra, rb = math.fabs(50 / divf(math.sin(a2))), math.fabs(50 / divf(math.sin(a1)))
86      tmp = calc_jiao(a, b, ra, rb)
87      ans = None
88      if tmp is not None:
89          c, d = tmp
90          if dis(c, Point(0, 0)) >= dis(d, Point(0, 0)):
91              ans = c
92          else:
93              ans = d
94      return ans, (a, ra), (b, rb)
95
96
97  def getans3p1x(p1: Point, p2: Point, p3: Point, a1_, a2_):
98      theta0 = get_ang(p1 - p2)
99      a1, a2 = a2_, a1_
100     theta = get_ang(p3 - p2) - theta0
101     l12, l32 = dis(p1, p2), dis(p3, p2)
102     a = Point(l12 / 2, l12 / 2 / divf(math.tan(a2)))
103     b = Point(l32 / 2 * math.sin(theta + a1) / divf(math.sin(a1)), -l32 / 2 * math.cos(theta + a1)
            / divf(math.sin(a1)))
104     ra, rb = math.fabs(l12 / 2 / divf(math.sin(a2))), math.fabs(l32 / 2 / divf(math.sin(a1)))
105     tmp = calc_jiao(a, b, ra, rb)
106     ans = None
107     if tmp is not None:
108         c, d = tmp
109         if dis(c, Point(0, 0)) >= dis(d, Point(0, 0)):
110             ans = c
111         else:
112             ans = d
113         ans = roatate(ans, theta0) + p2
114     return ans, (roatate(a, theta0) + p2, ra), (roatate(b, theta0) + p2, rb)
115
116
117 def showdata(data:  list[Point]):
118     for fly in std_flys:
119         plt.scatter(*fly, s=20, c='blue')
120     for fly in data:
121         plt.scatter(*fly, s=20, c='green')
122
123
124 def get_basic_change():
125     global generate_rates
126     min_dis, min_fa = 1e9, None
127     max_dis, max_fa = 0, None
128     for i in np.linspace(0, math.pi, 200)[:-1]:
```

```
129         pos = std_flys[2] + Point(generate_rates[-1] * math.cos(i), generate_rates[-1] * math.sin(
                i))
130         a0, a1, a2 = geta0a1a2(p0, p1, p2, p3, pos)
131         pos = std_flys[2] + Point(generate_rates[-1] * math.cos(i + math.pi),
132                                   generate_rates[-1] * math.sin(i + math.pi))
133         op_a0, op_a1, op_a2 = geta0a1a2(p0, p1, p2, p3, pos)
134         tmp_dis = (a0 - op_a0) ** 2 + (a1 - op_a1) ** 2 + (a2 - op_a2) ** 2
135         if tmp_dis < min_dis:
136             min_dis = tmp_dis
137             min_fa = np.array([(a0 - op_a0), (a1 - op_a1), (a2 - op_a2)])
138         if tmp_dis > max_dis:
139             max_dis = tmp_dis
140             max_fa = np.array([(a0 - op_a0), (a1 - op_a1), (a2 - op_a2)])
141     generate_angs = []
142     for j in generate_rates:
143         for i in np.linspace(0, 2 * math.pi, generate_num + 1)[:-1]:
144             pos = std_flys[2] + Point(j * math.cos(i), j * math.sin(i))
145             a0, a1, a2 = get_ang(p2 - pos) - get_ang(p0 - pos), get_ang(p1 - pos) - get_ang(p2 -
                    pos), \
146                          get_ang(p2 - pos) - get_ang(p3 - pos)
147             generate_angs.append(np.array([a0, a1, a2]))
148     generate_angs = generate_angs[generate_num - 1:]
149     pcd = o3d.geometry.PointCloud()
150     pcd.points = o3d.utility.Vector3dVector(np.array(generate_angs))
151     pcd.estimate_normals(search_param=o3d.geometry.KDTreeSearchParamHybrid(1, 5))
152     top_fas = np.asarray(pcd.normals)
153     top_fa = np.array([np.median(top_fas[:, 0]), np.median(top_fas[:, 1]), np.median(top_fas[:,
            2])])
154     change_basic = np.concatenate([[min_fa], [max_fa], [top_fa]], axis=0)
155     pos = std_flys[2]
156     a0, a1, a2 = get_ang(p2 - pos) - get_ang(p0 - pos), get_ang(p1 - pos) - get_ang(p2 - pos), \
157                  get_ang(p2 - pos) - get_ang(p3 - pos)
158     centure = np.array([a0, a1, a2])
159     # o3d.visualization.draw_geometries([pcd])
160     print(np.linalg.inv(change_basic.T))
161     print(centure)
162     return np.linalg.inv(change_basic.T), centure
163
164
165 def get_move_vec(policy_vecs, angles, p0_angle, self_num):
166     if self_num % 2 == 0:
167         a0, a1, a2 = angles
168     else:
169         a0, a2, a1 = angles
170         a0 = -a0
171     change_pos = np.matmul(change_basic, (np.array([a0, a1, a2]) - centure).T)
```

```
172     [k, idx, pdis] = pcd_tree.search_knn_vector_3d(change_pos, near_choose)
173     pdis = np.asfarray(pdis)
174     select_vecs = policy_vecs[idx]
175     vec = np.average(select_vecs, axis=0, weights=pdis)
176     vec_d = dis(Point(*vec), Point(0, 0))
177     vec_ang = get_ang(Point(*vec))
178     if self_num % 2 == 0:
179         vec = Point(vec_d * math.cos(vec_ang + p0_angle), vec_d * math.sin(vec_ang + p0_angle))
180     else:
181         vec = Point(vec_d * math.cos(p0_angle - vec_ang), vec_d * math.sin(p0_angle - vec_ang))
182     return vec
183
184
185 def norm_ang(ang):
186     pi = math.pi
187     if ang > 2 * pi:
188         ang -= 2 * pi
189     if ang < 0:
190         ang += 2 * pi
191     return ang
192
193 def geta0a1a2(p0, p1, p2, p3, pos):
194     a0, a1, a2 = get_ang(p2 - pos) - get_ang(p0 - pos), get_ang(p1 - pos) - get_ang(p2 - pos), \
195         get_ang(p2 - pos) - get_ang(p3 - pos)
196     return a0, norm_ang(a1), norm_ang(a2)
197
198 r = 100
199 std_flys = [Point(0, 0)]
200 for i in  range(9):
201     ang = i * 40 / 180 * math.pi
202     std_flys.append(Point(r * math.cos(ang), r * math.sin(ang)))
203
204 dataset = []
205 for i in  range(dataset_num):
206     data = []
207     for fly in std_flys:
208         data.append(fly + Point((random.randint(0, 2) * 2 - 1) * random.randint(0, max_pos_err *
                10) / 10,
209                                 (random.randint(0, 2) * 2 - 1) * random.randint(0, max_pos_err *
                                    10) / 10))
210     dataset.append(data)
211
212 generate_points = []
213 generate_angs = []
214 policy_vecs_beg = []
215 p0 = std_flys[0]
```

```
p1 = std_flys[1]
p2 = std_flys[7]
p3 = std_flys[4]
change_basic, centure = get_basic_change()
for j in generate_rates:
    for i in np.linspace(0, 2 * math.pi, generate_num + 1)[:-1]:
        pos = std_flys[2] + Point(j * math.cos(i), j * math.sin(i))
        a0, a1, a2 = geta0a1a2(p0, p1, p2, p3, pos)
        generate_angs.append(np.matmul(change_basic, (np.array([a0, a1, a2]) - centure).T))
        generate_points.append(pos)
        if j != 0:
            vec = generate_points[-generate_num - 1] - pos
            vec_d = dis(vec, Point(0, 0)) * 0.8
            if j < 1.1:
                vec_d *= 0.8
            if j < 0.51:
                vec_d *= 0.5
            vec_ang = get_ang(vec) - get_ang(p0 - pos)
            vec = Point(vec_d * math.cos(vec_ang), vec_d * math.sin(vec_ang))
            policy_vecs_beg.append(np.array(vec))
        else:
            policy_vecs_beg.append(np.zeros(2))
generate_points = generate_points[generate_num - 1:]
generate_angs = generate_angs[generate_num - 1:]
policy_vecs_beg = np.array(policy_vecs_beg[generate_num - 1:])
pcd = o3d.geometry.PointCloud()
pcd.points = o3d.utility.Vector3dVector(np.array(generate_angs))
pcd_tree = o3d.geometry.KDTreeFlann(pcd)


data = [(0,0),
(100,0),
(98,40.10),
(112,80.21),
(105,119.75),
(98,159.86),
(112,199.96),
(105,240.07),
(98,280.17),
(112,320.28),]
data = [Point(i[0] * math.cos(i[1] / 180 * math.pi), i[0] * math.sin(i[1] / 180 * math.pi)) for i
    in data]

if True:
    data_flys = copy.deepcopy(data)
    p0 = data_flys[0]
```

```
    # data_flys[0] = std_flys[0]
    # data_flys[1] = std_flys[1]
    # data_flys[4] = std_flys[4]
    # data_flys[7] = std_flys[7]
    flys = data_flys[1:]
    p_idx = [0, 3, 6]
    for fly in data_flys:
        plt.scatter(*fly, s=20, c='blue')
    for i in  range(50):
        j = (p_idx[0] + 1) %  len(flys)
        while j != p_idx[0]:
            # print(j)
            if j in p_idx:
                j += 1
                if j ==  len(flys):
                    j = 0
                continue
            pos = flys[j]
            if j - 1 in p_idx:
                p1 = flys[j - 1]
                p2 = flys[j - 4]
                p3 = flys[j - 7]
                a0, a1, a2 = geta0a1a2(p0, p1, p2, p3, pos)
                print([a0, a1, a2])
                vec = get_move_vec(policy_vecs_beg, (a0, a1, a2), get_ang(p0 - pos), 0)
            else:
                p1 = flys[j - 2]
                p2 = flys[j - 5]
                p3 = flys[j - 8]
                print([a0, a1, a2])
                a0, a1, a2 = geta0a1a2(p0, p1, p2, p3, pos)
                vec = get_move_vec(policy_vecs_beg, (a0, a1, a2), get_ang(p0 - pos), 1)
            plt.scatter(*pos, s=20, c='red')
            # plt.plot([pos.x, pos.x + vec[0] * 10], [pos.y, pos.y + vec[1] * 10], 'm')
            plt.arrow(pos.x, pos.y, vec[0] , vec[1] , width = 0.2, head_width = 5, ec = "m", fc =
                'm')
            # plt.plot([pos.x, pos.x + vec[0]], [pos.y, pos.y + vec[1]], 'm')

            flys[j] = pos + Point(*vec)

            j += 1
            if j ==  len(flys):
                j = 0
        plt.axis('equal')
        plt.pause(1)
        plt.clf()
```

```
306             for fly in flys:
307                 plt.scatter(*fly, s=20, c='blue')
308             plt.scatter(*p0, s=20, c='blue')
309             for t in  range( len(p_idx)):
310                 p_idx[t] += 1
311                 if p_idx[t] ==  len(flys):
312                     p_idx[t] = 0
313         r = 0
314         for fly in flys:
315             r += dis(fly, p0)
316         r /=  len(flys)
317         draw_circle = plt.Circle(p0, r, fill=False)
318         plt.gcf().gca().add_artist(draw_circle)
319         print(r)
320         err = 0
321         for fly in flys:
322             err += math.fabs(dis(p0, fly) - r)
323         print(err /  len(flys) / r)
324         r = 0
325         for i in  range( len(flys)):
326             r += dis(flys[i], flys[i - 1])
327         r /=  len(flys)
328         err = 0
329         for i in  range( len(flys)):
330             err += math.fabs(dis(flys[i], flys[i - 1]) - r)
331         print(err /  len(flys) / r)
332 
333 plt.axis('equal')
334 plt.show()
```

Listing 4: problem4.py 用于问题四调整策略加遗传算法计算

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from collections import namedtuple
4  import math
5  import random
6  import open3d as o3d
7  import copy
8  import multiprocessing as mp
9  import pickle
10 
11 dataset_num = 5000
12 max_pos_err = 3
13 generate_rates = [0, 2, 3.5, 5.5, 6.5,]
14 generate_num = 30
15 near_choose = 4
```

```
16
17  change_num = 20
18  calc_loss_num = 10
19  people_num = 48
20
21
22  Point = namedtuple('Point', ['x', 'y'])
23
24
25  def addPoint(a: Point, b: Point):
26      return Point(a.x + b.x, a.y + b.y)
27
28
29  Point.__add__ = addPoint
30
31
32  def subPoint(a: Point, b: Point):
33      return Point(a.x - b.x, a.y - b.y)
34
35
36  Point.__sub__ = subPoint
37
38
39  def divf(x):
40      if x == 0:
41          return 1e-7
42      return x
43
44
45  def roatate(p: Point, theta):
46      d = dis(p, Point(0, 0))
47      theta0 = get_ang(p)
48      return Point(d * math.cos(theta + theta0), d * math.sin(theta + theta0))
49
50
51  def get_ang(p: Point):
52      ang = math.atan(p.y / divf(p.x))
53      if p.x < 0:
54          ang += math.pi
55      if ang < 0:
56          ang += 2 * math.pi
57      return ang
58
59
60  def dis(a: Point, b: Point):
61      return math.sqrt((a.x - b.x) ** 2 + (a.y - b.y) ** 2)
```

```
62
63
64  def calc_jiao(a: Point, b: Point, ra, rb):
65      l = dis(a, b)
66      if l > ra + rb:
67          return None
68      k1 = (b.y - divf(a.y)) / (b.x - divf(a.x))
69      k2 = - (b.x - divf(a.x)) / (b.y - divf(a.y))
70      x0 = a.x + (b.x - a.x) * (ra ** 2 - rb ** 2 + l ** 2) / 2 / l ** 2
71      y0 = a.y + k1 * (x0 - a.x)
72      r2 = ra ** 2 - (x0 - a.x) ** 2 - (y0 - a.y) ** 2
73      return Point(x0 - math.sqrt(r2 / (1 + k2 ** 2)), y0 - k2 * math.sqrt(r2 / (1 + k2 ** 2))), \
74             Point(x0 + math.sqrt(r2 / (1 + k2 ** 2)), y0 + k2 * math.sqrt(r2 / (1 + k2 ** 2)))
75
76
77  def get_ans(a0, ap1, ap2, cross):  # corss: p2-p1
78      a2, a1 =  abs(ap1 - a0),  abs(ap2 - a0)
79      theta = cross * 40 / 180 * math.pi
80
81      if ap1 >= a0:
82          a = Point(50, 50 / divf(math.tan(a2)))
83      else:
84          a = Point(50, -50 / divf(math.tan(a2)))
85
86      if a0 >= ap2:
87          b = Point(50 * math.sin(theta + a1) / divf(math.sin(a1)), -50 * math.cos(theta + a1) /
                divf(math.sin(a1)))
88      else:
89          b = Point(-50 * math.sin(theta - a1) / divf(math.sin(a1)), 50 * math.cos(theta - a1) /
                divf(math.sin(a1)))
90
91      ra, rb = math.fabs(50 / divf(math.sin(a2))), math.fabs(50 / divf(math.sin(a1)))
92      tmp = calc_jiao(a, b, ra, rb)
93      ans = None
94      if tmp is not None:
95          c, d = tmp
96          if dis(c, Point(0, 0)) >= dis(d, Point(0, 0)):
97              ans = c
98          else:
99              ans = d
100     return ans, (a, ra), (b, rb)
101
102
103 def getans3p1x(p1: Point, p2: Point, p3: Point, a1_, a2_):
104     theta0 = get_ang(p1 - p2)
105     a1, a2 = a2_, a1_
```

```
    theta = get_ang(p3 - p2) - theta0
    l12, l32 = dis(p1, p2), dis(p3, p2)
    a = Point(l12 / 2, l12 / 2 / divf(math.tan(a2)))
    b = Point(l32 / 2 * math.sin(theta + a1) / divf(math.sin(a1)), -l32 / 2 * math.cos(theta + a1)
        / divf(math.sin(a1)))
    ra, rb = math.fabs(l12 / 2 / divf(math.sin(a2))), math.fabs(l32 / 2 / divf(math.sin(a1)))
    tmp = calc_jiao(a, b, ra, rb)
    ans = None
    if tmp is not None:
        c, d = tmp
        if dis(c, Point(0, 0)) >= dis(d, Point(0, 0)):
            ans = c
        else:
            ans = d
        ans = roatate(ans, theta0) + p2
    return ans, (roatate(a, theta0) + p2, ra), (roatate(b, theta0) + p2, rb)


def showdata(data:  list[Point]):
    for i in  range( len(std_flys)):
        plt.scatter(*std_flys[i], s=20, c='blue')

    for fly in data:
        plt.scatter(*fly, s=20, c='green')


def norm_ang(ang):
    pi = math.pi
    if ang > 2 * pi:
        ang -= 2 * pi
    if ang < 0:
        ang += 2 * pi
    return ang


def geta0a1a2(p0, p1, p2, p3, pos):
    a0, a1, a2 = get_ang(p2 - pos) - get_ang(p0 - pos), get_ang(p1 - pos) - get_ang(p2 - pos), \
                 get_ang(p2 - pos) - get_ang(p3 - pos)
    return a0, norm_ang(a1), norm_ang(a2)


def get_policy(p1, p2, p3, p):
    min_dis, min_fa = 1e9, None
    max_dis, max_fa = 0, None
    for i in np.linspace(0, math.pi, 200)[:-1]:
        pos = p + Point(generate_rates[-1] * math.cos(i), generate_rates[-1] * math.sin(i))
```

```
        a1, a2 = get_ang(p1 - pos) - get_ang(p2 - pos), get_ang(p2 - pos) - get_ang(p3 - pos)
        a1, a2 = norm_ang(a1), norm_ang(a2)
        pos = p + Point(generate_rates[-1] * math.cos(i + math.pi), generate_rates[-1] * math.sin(
            i + math.pi))
        op_a1, op_a2 = get_ang(p1 - pos) - get_ang(p2 - pos), get_ang(p2 - pos) - get_ang(p3 - pos
            )
        op_a1, op_a2 = norm_ang(op_a1), norm_ang(op_a2)
        tmp_dis = (a1 - op_a1) ** 2 + (a2 - op_a2) ** 2
        if tmp_dis < min_dis:
            min_dis = tmp_dis
            min_fa = np.array([(a1 - op_a1), (a2 - op_a2), 0])
        if tmp_dis > max_dis:
            max_dis = tmp_dis
            max_fa = np.array([(a1 - op_a1), (a2 - op_a2), 0])
    change_basic = np.concatenate([[min_fa], [max_fa], [np.array([0, 0, 1])]], axis=0)
    change_basic = np.linalg.inv(change_basic.T)
    a1, a2 = get_ang(p1 - p) - get_ang(p2 - p), get_ang(p2 - p) - get_ang(p3 - p)
    a1, a2 = norm_ang(a1), norm_ang(a2)
    centure = np.array([a1, a2, 0])
    generate_points = []
    generate_angs = []
    policy_vecs_beg = []
    for j in generate_rates:
        for i in np.linspace(0, 2 * math.pi, generate_num + 1)[:-1]:
            pos = p + Point(j * math.cos(i), j * math.sin(i))
            a1, a2 = get_ang(p1 - pos) - get_ang(p2 - pos), get_ang(p2 - pos) - get_ang(p3 - pos)
            a1, a2 = norm_ang(a1), norm_ang(a2)
            change_ang = np.matmul(change_basic, (np.array([a1, a2, 0]) - centure).T)
            generate_angs.append(change_ang)
            generate_points.append(pos)
            if j != 0:
                vec = generate_points[-generate_num - 1] - pos
                vec_d = dis(vec, Point(0, 0)) * 0.8
                if j < 1.1:
                    vec_d *= 0.8
                if j < 0.51:
                    vec_d *= 0.5
                vec_ang = get_ang(vec) - get_ang(p2 - pos)
                vec = Point(vec_d * math.cos(vec_ang), vec_d * math.sin(vec_ang))
                policy_vecs_beg.append(np.array(vec))
            else:
                policy_vecs_beg.append(np.zeros(2))
    generate_points = generate_points[generate_num - 1:]
    generate_angs = generate_angs[generate_num - 1:]
    policy_vecs_beg = np.array(policy_vecs_beg[generate_num - 1:])
    pcd = o3d.geometry.PointCloud()
```

```
195     pcd.points = o3d.utility.Vector3dVector(np.array(generate_angs))
196     pcd_tree = o3d.geometry.KDTreeFlann(pcd)
197     return generate_points, generate_angs, policy_vecs_beg, pcd_tree, (change_basic, centure)
198 
199 
200 def get_move_vec(policy_vecs, pcd_tree, angles, p0_angle, query, self_num):
201     if self_num % 2 == 0:
202         a1, a2 = angles
203     else:
204         a2, a1 = angles
205     change_ang = np.matmul(query[0], (np.array([a1, a2, 0]) - query[1]).T)
206     [k, idx, pdis] = pcd_tree.search_knn_vector_3d(change_ang, near_choose)
207     pdis = np.asfarray(pdis)
208     select_vecs = policy_vecs[idx]
209     vec = np.average(select_vecs, axis=0, weights=pdis)
210     vec_d = dis(Point(*vec), Point(0, 0))
211     vec_ang = get_ang(Point(*vec))
212     if self_num % 2 == 0:
213         vec = Point(vec_d * math.cos(vec_ang + p0_angle), vec_d * math.sin(vec_ang + p0_angle))
214     else:
215         vec = Point(vec_d * math.cos(p0_angle - vec_ang), vec_d * math.sin(p0_angle - vec_ang))
216     return vec
217 
218 
219 def get_top_policy(policy, angle):
220     a1, a2 = angle
221     a0 = math.fabs(a1 - a2)
222     pos = np.searchsorted(angel_set_v, a0)
223     if pos == 0:
224         d = policy[0]
225     elif pos ==  len(angel_set_v):
226         d = policy[-1]
227     else:
228         d = policy[pos - 1] * (a0 - angel_set_v[pos - 1]) + policy[pos] * (angel_set_v[pos] - a0)
229     ang = (a1 + a2) / 2
230     if ang > math.pi / 2:
231         ang -= math.pi / 2
232     return Point(d * math.cos(ang), d * math.sin(ang))
233 
234 
235 
236 def calc_loss(policy, data_use):
237     data = copy.deepcopy(data_use)
238     top_flys = [0, 10, 14]
239     policy_top, policy_in = policy
240     for i in  range(30):
```

```
241         for j in  range( len(data)):
242             if j in top_flys:
243                 pos_j = top_flys.index(j)
244                 p1, p2 = data[top_flys[pos_j - 1]], data[top_flys[pos_j - 2]]
245                 a1, a2 = get_ang(p1 - data[j]), get_ang(p2 - data[j])
246                 vec = get_top_policy(policy_top, (a1, a2))
247             else:
248                 p1, p2, p3 = (data[t - 1] for t in flys_types[j][1])
249                 a1, a2 = get_ang(p1 - data[j]) - get_ang(p2 - data[j]), get_ang(p2 - data[j]) -
                        get_ang(p3 - data[j])
250                 a1, a2 = norm_ang(a1), norm_ang(a2)
251                 policy = policy_in[flys_types[j][0] - 1]
252                 vec = get_move_vec(policy[0], query_info[policy[1]][0], (a1, a2), get_ang(p2 -
                        data[j]),
253                                    query_info[policy[1]][1], flys_types[j][2])
254             data[j] = data[j] + Point(*vec)
255     r = 0
256     for side in sides:
257         r += dis(data[side[0] - 1], data[side[1] - 1])
258     r /=  len(sides)
259     err = 0
260     for side in sides:
261         err += math.fabs(dis(data[side[0] - 1], data[side[1] - 1]) - r)
262     err /=  len(sides) * r
263     return err
264
265
266 def alldata_loss(args):
267     data_use, policy = args
268     data = copy.deepcopy(data_use)
269     err = 0
270     for d in data:
271         err += calc_loss(policy, d)
272     return err, policy
273
274
275
276 std_flys = [
277         (200, 0),
278         (175, 25 * math.sqrt(3)),
279         (150, 0),
280         (150, 50 * math.sqrt(3)),
281         (125, 25 * math.sqrt(3)),
282         (100, 0),
283         (125, 75 * math.sqrt(3)),
284         (100, 50 * math.sqrt(3)),
```

```
        (75, 25 * math.sqrt(3)),
        (50, 0),
        (100, 100 * math.sqrt(3)),
        (75, 75 * math.sqrt(3)),
        (50, 50 * math.sqrt(3)),
        (25, 25 * math.sqrt(3)),
        (0, 0),
    ]
std_flys = [Point(*i) for i in std_flys]
flys_types = [
        [0, ],
        [1, (1, 15, 11), 0],
        [1, (15, 11, 1), 1],
        [2, (1, 15, 11), 0],
        [3, (1, 15, 11), 0],
        [2, (15, 11, 1), 0],
        [1, (1, 15, 11), 1],
        [3, (11, 1, 15), 0],
        [3, (15, 11, 1), 0],
        [1, (15, 11, 1), 0],
        [0, ],
        [1, (11, 1, 15), 0],
        [2, (11, 1, 15), 0],
        [1, (11, 1, 15), 1],
        [0, ]
    ]
sides = [
        (1, 2),
        (2, 3),
        (1, 3),
        (2, 4),
        (4, 5),
        (5, 2),
        (5, 3),
        (5, 6),
        (6, 3),
        (7, 4),
        (7, 8),
        (8, 4),
        (8, 5),
        (8, 9),
        (9, 5),
        (9, 6),
        (9, 10),
        (10, 6),
        (11, 7),
```

```
        (11, 12),
        (12, 7),
        (12, 8),
        (12, 13),
        (13, 8),
        (13, 8),
        (13, 14),
        (14, 9),
        (14, 10),
        (14, 15),
        (15, 10),
    ]

query_info = []
angel_set_v = []

p1 = Point(25, 0)
p2 = Point(-25, 0)
p = Point(0, 25 * math.sqrt(3))
point_set_v = []
policy_vec_beg_top = []
for j in  range( len(generate_rates) - 1, -1, -1):
    # 小角
    pos = p + Point(0, generate_rates[j])
    point_set_v.append(pos)
    a = get_ang(p1 - pos) - get_ang(p2 - pos)
    angel_set_v.append(a)
    if j == 0:
        policy_vec_beg_top.append(0)
    else:
        policy_vec_beg_top.append(generate_rates[j] - generate_rates[j - 1])
for j in  range( len(generate_rates)):
    # 大角
    if j != 0:
        pos = p - Point(0, generate_rates[j])
        point_set_v.append(pos)
        a = get_ang(p1 - pos) - get_ang(p2 - pos)
        angel_set_v.append(a)
        policy_vec_beg_top.append(generate_rates[j - 1] - generate_rates[j])
angel_set_v = np.array(angel_set_v)
policy_vec_beg_top = np.array(policy_vec_beg_top)

p1 = Point(200, 0)
p2 = Point(0, 0)
p3 = Point(100, 100*math.sqrt(3))
p_pos = [Point(175, 25*math.sqrt(3)), Point(150, 50*math.sqrt(3)), Point(125, 25*math.sqrt(3))]
```

```
377 p_policys = []
378 for p in  range( len(p_pos)):
379     _, _, policy_vec, pcd_tree, query = get_policy(p1, p2, p3, p_pos[p])
380     query_info.append((pcd_tree, query))
381     p_policys.append((policy_vec, p))
382 
383 if __name__ == '__main__':
384     dataset = []
385     for i in  range(dataset_num):
386         data = []
387         for fly in std_flys:
388             data.append(fly + Point((random.randint(0, 1) * 2 - 1) * random.randint(0, max_pos_err
                    * 10) / 10,
389                                     (random.randint(0, 1) * 2 - 1) * random.randint(0, max_pos_err
                                          * 10) / 10))
390         dataset.append(data)
391 
392 
393     all_policy = [(policy_vec_beg_top, p_policys) ]
394     while True:
395         for _ in  range(people_num):
396             policy_change = copy.deepcopy(random.choice(all_policy)[1])
397             for i in random.sample( range(
                    len(policy_change[0][0])), random.randint(0, change_num)):
398                 change_point = random.randint(0,  len(p_pos) - 1)
399                 if random.randint(0, 6) == 1:
400                     # 变异
401                     if random.randint(0, 1) == 1:
402                         # 改角度
403                         d = dis(Point(0, 0), Point(*policy_change[change_point][0][i]))
404                         ang = random.randint(0, 1000000)
405                         policy_change[change_point][0][i][0] = d * math.cos(ang)
406                         policy_change[change_point][0][i][1] = d * math.sin(ang)
407                     else:
408                         # 改长度
409                         d = dis(Point(0, 0), Point(*policy_change[change_point][0][i]))
410                         ang = get_ang(Point(*policy_change[change_point][0][i]))
411                         d *= random.randint(3000, 8000) / 5000
412                         policy_change[change_point][0][i][0] = d * math.cos(ang)
413                         policy_change[change_point][0][i][1] = d * math.sin(ang)
414                 else:
415                     # 杂交
416                     hooker = random.choice(all_policy)[1]
417                     policy_change[change_point][0][i][0] = hooker[change_point][0][i][0]
418                     policy_change[change_point][0][i][1] = hooker[change_point][0][i][1]
419             all_policy.append((policy_vec_beg_top, policy_change))
```

```
420
421         data_use = random.sample(dataset, calc_loss_num)
422         all_loss = []
423         # for ans in mp.Pool(processes=12).imap(alldata_loss, [(data_use, p) for p in all_policy])
                :
424         #     all_loss.append(ans)
425         for t in [(data_use, p) for p in all_policy]:
426             ans = alldata_loss(t)
427             all_loss.append(ans)
428         all_loss.sort(key=lambda x: x[0])
429         all_loss = all_loss[: people_num // 2]
430         print(all_loss[0][0])
431         pickle.dump(all_loss[0][1],  open('policy.pkl', 'wb'))
432         all_policy = [i[1] for i in all_loss]
433         del all_loss
```