基于改进型调度算法的无人机遂行编队飞行研究

摘　要

近年来，无人机领域的相关研究备受瞩目，其在包括测绘勘探、农业灌溉等领域的潜在价值逐渐被挖掘利用，但在实际应用的复杂动态环境中，无人机遂行编队飞行需要考虑多方面的复杂因素。本文着重讨论了在无人机采用纯方位无源定位法的情况下，通过设计改进型无人机调度算法，以实现无人机编队的有效定位与机群调度。

针对主问题一，本文立足已知信息建立了极坐标系模型，结合本题中纯方位无源定位法的原理，通过正确无人机的已知参数来描述偏差无人机的未知参数，求解得到偏差无人机具体的极坐标位置，进而基于上述条件对问题一的子问题进行分析。

针对子问题一，为建立被动接收信号无人机的定位模型，提出单无人机调度方案。本文首先创新性地定义了无人机调度过程中的描述方法：通过定义响应点K、目标点H与预测点P，对偏差无人机的调度进行了细致明确地描述。其次，基于上述定义并结合极坐标模型，设计出基于 Python 程序语言的偏差无人机调度算法，通过不断地循环迭代来使得偏差无人机尽可能逼近其正确的目标点。最终，本文通过修正后的数据集，对该无人机调度算法进行测试与评估，结果表明，本题算法的准确率高达 99.9%。

针对子问题二，为解决除已知的无人机$FY00, FY01$外，还需要引入几架无人机，才能实现对偏差无人机进行有效定位的问题。本文通过逐步引入发射信号无人机的策略，模拟对未知编号无人机的定位，分析是否可以对所有情况下的偏差无人机进行有效定位。最终得出，在已知编号为$FY00, FY01$无人机的情况下，仅需要再引入 1 架无人机，即可实现对偏差无人机的有效定位。

针对子问题三，为提出无人机集群整体的调度方案，本文对原算法循环迭代的方式做出进一步的优化，引入调度中介无人机，采用调度中介循环更替的创新方法，将子问题一中无人机调度算法从单无人机调度扩展到多无人机调度，同时使得各偏差无人机最终的预测点尽可能地逼近其对应的正确目标点。并且，本文通过测试题表中的数据集，对改进型无人机调度算法进行测试与评估，结果表明，该改进算法能够精确地将所有的偏差无人机逐步逼近至其对应的正确目标点，且拟合精度高达 99.9%。

针对主问题二，基于圆形飞行编队的无人机调度算法进一步推广到本题中的锥形飞行编队，本文考虑到该锥形模型的特殊性，在原有算法的基础上，拓展出响应点之间的反馈机制，从而实现动态调整响应点的同时，对先前已调整过的其他响应点进行进一步的优化。最终本文解决了圆形及锥形无人机集群的纯方位无源定位问题。

最后，本文结合实际情况对模型进行了评价与推广，使得模型具有很好的通用性。

关键词：无人机调度算法 纯方位无源定位 极坐标系模型 圆形无人机编队

一、 问题重述

近年来，随着 5G 通信、计算机技术的蓬勃发展与优化算法、人工智能等领域的开拓创新，无人驾驶飞行器（无人机）取得了突破性进展，在包括空中测绘、军事勘探、农业灌溉等领域的利用价值被逐步挖掘[1-4]。但在实际应用的复杂动态环境中，无人机的遂行编队容易受到来自外界的电磁干扰，造成无人机的位置偏差乃至于定位丢失，使得无人机群执行任务变得困难[5, 6]。

为尽量避免电磁干扰、保持无人机编队队形，相关研究者提出了纯方位无源定位的方法：由编队中的某几架无人机发射信号，其他无人机保持电磁静默，被动接受信号。其中，无人机所收到的纯方位信息约定为：该无人机与任意两架可以发射信号的无人机连线之间的夹角。此外，规定编队中的每架无人机均有固定编号，且与编队中其他无人机的两两相对关系保持不变。本文基于上述纯方位无源定位的方法，建立数学模型分析研究以下问题：

问题一：飞行编队由 10 架无人机组成一个圆形编队，其中 9 架无人机（编号 FY01-FY09）均匀分布在以中心无人机（编号 FY00）的位置为圆心的某一圆周上，并且所有无人机均保持在同一高度上飞行。现基于以上条件，解决下列子问题：

（1）位于圆心的无人机（编号 FY00）和位于圆周上的另外 2 架无人机发射信号，其余偏差无人机被动接受信号。当发射信号的 3 架无人机位置无偏差且编号已知时，建立被动接受信号无人机的定位模型。

（2）某个偏差无人机接收到编号 FY00 与 FY01 的两架无人机发射的信号，另外还接收到编队中若干编号未知，即在圆周上的定位未知的无人机发射的信号。如果发射信号的所有无人机位置均无偏差，那么除了编号已知的两架无人机外，还需要引入几架无人机发射信号，才能实现该偏差无人机的有效定位？

（3）现已知 1 架无人机位于圆心，另外 9 架无人机均匀地分布在半径为 100 米的圆周上。当初始时刻的无人机位置略有偏差时，请给出合理、具体的无人机位置调整方案：即经过多次的调整，每次选择无人机 FY00 和圆周上的最多 3 架无人机发射信号，其余的无人机根据接收到的方向信息不断地进行位置调整，进而到达规定的理想位置。最终使得圆周的 9 架无人机都能均匀分布在半径为 100 米的圆周上。其中，10 架无人机各自的初始位置如题所示。

问题二：在现实的飞行中，无人机集群也可以是其他编队队形[7, 8]，例如题中的锥形编队队形，在该队形中，直线上相邻两架无人机的间距相等（如 50 米）。现仍采用纯方位无源定位的方式，设计高效合理的无人机位置调整方案。

二、 模型假设与说明

2.1 模型假设

1. 所有的无人机均既可以发射信号，也可以接收信号；
2. 发射信号的无人机同时可以接收其他无人机发射的信号；
3. 角度信息为无人机集群相互通信的唯一信息；
4. 位置正确的无人机两两之间始终保持相对静止；
5. 不考虑无人机集群在空中飞行时潜在的外界影响因素，如刮风下雨、电磁干扰、飞鸟碰撞等情况。

2.2 假设说明

因模型假设对于本文模型的建立与求解至关重要，将会极大地影响本文模型的科学性与有效性，因此本文首先对模型假设进行必要的说明：

针对假设 1，本文考虑实际情况，认为无人机集群中的任一无人机应该均可以发射信号，也可以接收信号，只是为了避免电磁干扰，人为地设置被动接受信号的无人机保持电磁静默，不发射信号；

针对假设 2，基于上述的说明，可进一步推断，发射信号的无人机可以接收其他的无人机发射的信号，也即发射信号的无人机之间是可以进行信息通信的；

针对假设 3，基于充分理解题中对于纯方位无源定位的诠释，本文假设角度信息为无人机集群相互通信的唯一信息；

针对假设 4，结合模型建立过程中实际考虑，通过假设位置正确的无人机两两之间始终保持相对静止，进而消除本文模型建立过程中的次要因素干扰，确保本文模型的有效性。

针对假设 5，结合实际情况，本文不考虑无人机集群在空中遂行编队飞行时潜在的外界不可抗因素的影响，从而排除小概率事件的影响。

三、 符号说明

符号	说明	单位
O	极坐标系模型的极点	/
B	位于圆周上的已知无人机所在的端点	/
K	响应点，用以描述偏差无人机在调度过程中的动态位置	/
H	目标点，用以描述偏差无人机希望尽可能逼近的优化结果	/
P	预测点，用以描述响应点最后能够达到的一个最优解	/
$\Delta\alpha$	响应夹角与目标夹角的角度差值	度
Δd	响应点与目标点之间的距离差值	米
D_b	所有偏差无人机的偏差距离之和	米

四、 模型的建立与求解

4.1 问题一模型的建立与求解

4.1.1 子问题一模型的分析

问题一要求建立被动接受信号无人机的定位模型，则需要获得偏差无人机对于已知无偏差无人机的相对位置信息。也即本文通过已知无人机的位置，来描述偏差无人机的位置，由已知推算未知，即可对偏差无人机进行定位。

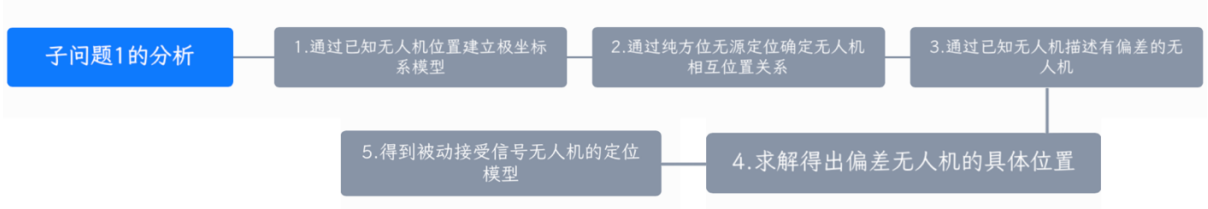


图 1 子问题一分析流程图

如图 1 所示，本文首先通过已知编号的无偏差无人机的位置来建立二维平面上的极坐标系模型，从而可描述已知无人机的具体极坐标定位，再根据无人机之间纯方位无源定位的方法，来确定偏差无人机与无偏差无人机之间的角度关系，进而推得偏差无人机的具体极坐标定位。

在此基础上，考虑到偏差无人机仅能被动接收信号，而无法反馈信息给已知无人机，为实现偏差无人机的正确调度，使其返回对应的正确圆周位置，本文进一步使用 Python 设计偏差无人机的调度算法，结合上述极坐标系模型进行定位计算，最终实现被动接受信号无人机的有效定位与位置调度。

4.1.2 极坐标模型的建立

极坐标系模型指的是在平面内，由极点、极轴、极径组成的二维坐标系模型。在平面上取一个点称为极点O，从极点O出发引一条射线OX称为极轴，并规定极坐标系上的角度取逆时针方向为正，顺时针方向为负。从而，平面上任一端点K的位置，均可用线段OK的长度（记作ρ），以及从射线OX到线段OK的角度（记作θ）来确定，而有序数对(ρ,θ)即称为K点的极坐标定位，记作$K\ (\rho,\theta)$。

如图 2 所示，为了通过数学语言描述接收信号无人机的位置信息，本文建立二维极坐标系模型，以中心无人机$FY00$的位置作为极点O，以O为极点且经过无人机$FY01$所在端点A的射线作为极坐标轴，建立极坐标系。

现假设圆周半径为r，则无人机$FY00$的极坐标为$(0,0)$，无人机$FY01$的极坐标为$(r,0)$，为不失一般性，本文假设圆周上另一架已知无人机编号为$FY0B(B\neq 0,1)$，其所在位置为端点B。由本题中定义的无人机在二维平面的圆周上均匀分布的情况，可知圆周上每架无人机的端点与原点O之间的连线之间的夹角$\tau=[2/9]\pi$（如图 2）。进而可推得无人机$FY0B$的极坐标应为$(r,[B-1]\cdot\tau)$，其中$B\in\{2,3,\dots,9\}$。进一步，可假设偏差无人机$FY0K$，其中$K\in\{2,3,\dots 9\}\backslash\{B\}$，则该无人机的极坐标位置为$(\rho,\theta)$，其端点为$K$。因此，只要确定未知参数$\rho$与$\theta$，即可用数学语言描述偏差无人机的定位信息。

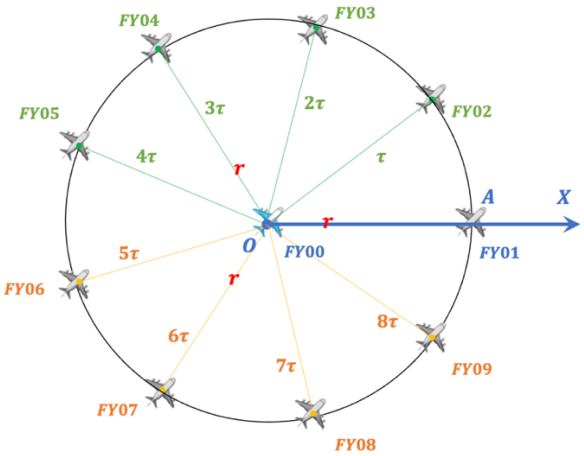


图 2 极坐标模型的建模示意图

如上所述，本文首先通过建立极坐标系来表示无人机在二维平面上的具体位置，进而可以利用无人机纯方位无源定位的方法，确定偏差无人机与已知无人机的相对位置关系。如图 3 中位置状态一所示，令$\angle OKA=\alpha_1,\angle OKB=\alpha_2,\angle AKB=\alpha_3,\angle OBA=\alpha_4,\angle OAB=\alpha_5,\angle AOB=\alpha_6$，结合本文假设 1，再根据纯方位无源定位的原理，可推得$\alpha_1,\alpha_2,\alpha_3,\alpha_4,\alpha_5,\alpha_6$均为已知角度。其次为便于表示，现假设端点$AB$之间的长度为$x$。

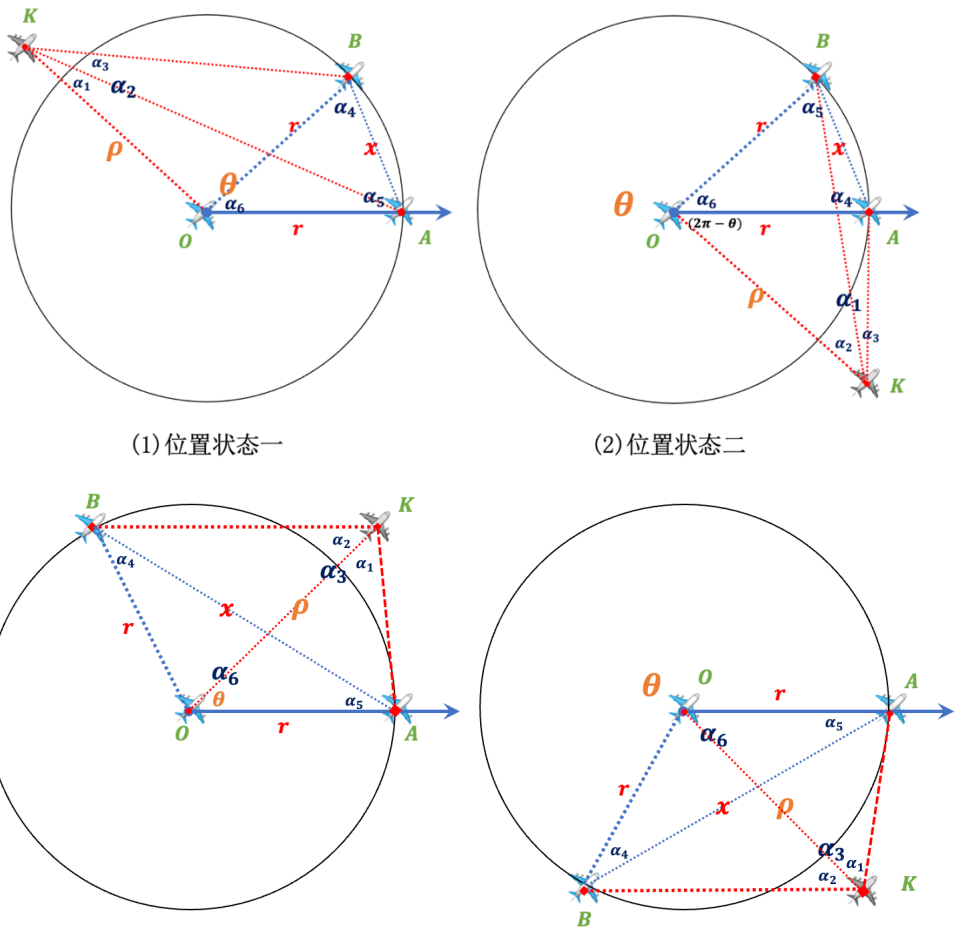


图 3 极坐标系模型中的四种状态示例

如图 3 所示，本文具体列举出了偏差无人机相对于圆周上两架已知无偏差无人机的 4 种位置概略图。将端点O,A,B,K两两连线，即可构建图 3 不同位置状态中$\Delta OBA,\Delta OKB,\Delta OKA$等三角形的关系。进而，结合上文所述，求解本题的关键在于使用已知条件来表示偏差无人机的参数ρ,θ。也即确定ρ,θ后，即可通过极坐标系模型对偏差无人机进行具体的定位。为验证极坐标系模型对表示无人机定位的有效性与一般性，以图 4 的位置状况为例，对此状态下偏差无人机的位置信息进行求解：

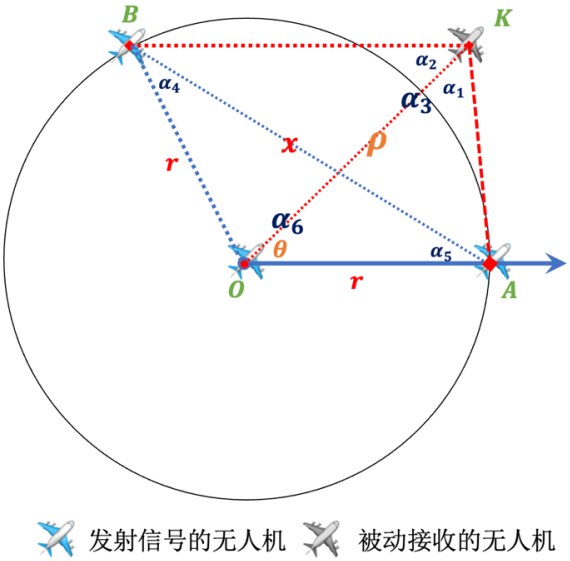


图 4 极坐标系模型中的位置状态演示

如图 4 所示，此时$\alpha_1,\alpha_2,\alpha_3,\alpha_4,\alpha_5,\alpha_6$的角度信息与$r,x$的边长信息均为已知，根据三角形的正弦定理与余弦定理可得：

$$\begin{cases} x^2 = r^2 + r^2 - 2r \cdot r \cdot \cos\alpha_6 \\ \left[\begin{array}{c} \dfrac{x}{\sin\alpha_3} = \dfrac{|AK|}{\sin\alpha_3} = \dfrac{|AK|}{\sin(\alpha_4 - \alpha_2 + \theta)} \\ \dfrac{r}{\sin\alpha_1} = \dfrac{|AK|}{\sin\theta} \end{array}\right. \end{cases} \tag{1}$$

进一步，对式 1 化简求解可得极坐标模型的一般化公式：

$$\begin{cases} \theta = \operatorname{arc\,cot}\dfrac{r \cdot \sin\alpha_3 - x \cdot \sin\alpha_1 \cdot \cos(\alpha_4 - \alpha_2)}{x \cdot \sin\alpha_1 \cdot \sin(\alpha_4 - \alpha_2)} \\ \rho = r(\sin\theta \cdot \cot\alpha_1 + \cos\theta) \end{cases} \tag{2}$$

$$\text{其中} \begin{cases} x = \sqrt{2 \cdot (1 - \cos\alpha_6)} \cdot r \\ \alpha_6 = \angle AOB = (B-1) \cdot \frac{2}{9}\pi \\ B = \{2,3,\ldots,9\} \end{cases} \tag{3}$$

本文考虑到根据无人机编号$FY0B$中B的不同取值($B \in \{2,3,\ldots,9\}\backslash\{k\}$)，端点$O,A,B,K$构建出的三角形关系可能也会有所不同，但本质上，均是通过求解三角形，进而利用已知参数的极坐标信息来表示出未知参数ρ,θ，因此可推知极坐标系模型的推理过程具有一般性。

综上，本文利用已知无偏差无人机的位置建立极坐标系模型，再通过纯方位无源定位方法所得到的三角形关系，推导得出偏差无人机的级坐标(ρ,θ)，实现了在二维平面中对被动接收信号无人机的坐标定位。

4.1.3 极坐标系模型的求解

如上所述，本文通过建立极坐标模型对偏差无人机进行了具体的定位，接下来需要考虑如何设计一个调度算法，来对被动接受信号的偏差无人机的位置进行调整。首先，由题可知无人机之间的传递的方向信息为：该接收无人机与另两架发射信号的无人机连线之间的夹角，即角度为无人机集群通信的唯一信息。并且偏差无人机只能被动接受信号而无法发送信号，即偏差无人机无法发送反馈给位置正确的已知无人机。

因此，可推知无人机是通过对比目前的响应夹角与正确的目标夹角，来调整自身的位置，从而能够不断逼近对应正确的圆周位置。如图 5 所示，本文定义响应夹角为$\angle OKA,\angle OKB,\angle AKB$（图中绿色实线部分），目标夹角为$\angle OHA,\angle OHB,\angle AHB$（图中红色虚线部分）。其中，$K$点位置记为偏差无人机的收到信号的初始响应点，本文定义响应点是随K点动态变化的，而目标点为响应点最终要逼近的目标，理论可推知上响应点只能无限逼近于目标点，而无法到达目标点。

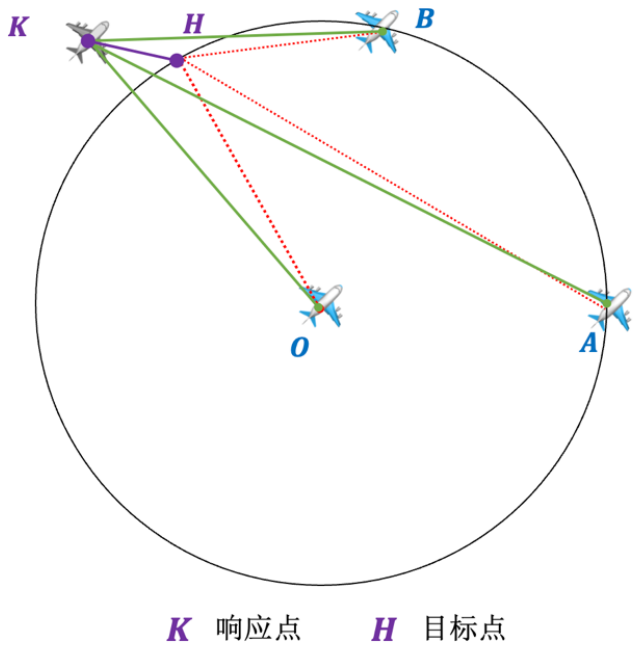


图 5 响应夹角与目标夹角的示意图

由上可知，偏差无人机调度的核心是通过判断响应夹角α_1与目标夹角α_2的差值$\Delta\alpha$，其中$\Delta\alpha=|\alpha_1-\alpha_2|$。如图 5，可令$\angle OKB=\alpha_1,\angle OHB=\alpha_2$，通过不断调整偏差无人机$K$，使其$\Delta\alpha$逼近于 0，即可实现无人机从偏差的响应点$K$移动到正确的目标点$H$。意即，偏差无人机虽然不能发射信号反馈位置，但其能够计算自身目前响应角和目标角之间的差值，通过不断调整以缩小差距，从而不断逼近圆周上的目标正确点。因此，本文基于以上原理，使用 Python 程序语言设计出偏差无人机的调度算法，该算法的伪代码如下表 1 所示（具体代码见附录）：

表 1 偏差无人机调度算法的伪代码

算法1：偏差无人机的调度算法（伪代码）

1 偏差无人机K通过纯方位无源定位获取初始响应夹角；
2 通过极坐标模型计算偏差无人机的响应点极坐标(ρ_1,θ_1)，目标点极坐标(ρ_2,θ_2)与已知无人机的极坐标，如点A为(ρ_A,θ_A)，点B为(ρ_B,θ_B)，其他同理；
3 将极坐标转化为直角坐标，如响应点极坐标(ρ_1,θ_1)转换为(x_1,y_1)即$(\rho_1 cos\theta_1,\rho_1 sin\theta_1)$，目标点极坐标$(\rho_2,\theta_2)$转化$(x_2,y_2)$即$(\rho_2 cos\theta_2,\rho_2 sin\theta_2)$，其他同理；
4 通过直角坐标计算出点与点之间的距离，如$|KO|,|KA|,|KB|$等；
5 通过余弦定理计算出目前的响应夹角，与目标夹角进行对比，得出偏差$\Delta\alpha=|\alpha_1-\alpha_2|$
6 `for i_in_range(1000)` // 循环迭代1000次
    调整响应点直角坐标，即令$(x_1+\Delta x)$，此时y_1不变，此时重新计算偏差$\Delta\alpha$
    `for i_in_range(1000)` // 循环迭代1000次
        再令$(y_1+\Delta y)$，此时再次重新计算偏差$\Delta\alpha$
// 不断循环迭代，尽可能使得$\Delta\alpha$无限逼近于0，即响应夹角无限接近目标夹角
7 取上述迭代过程中的$min\Delta\alpha$所对应的直角坐标，记为预测点(x_3,y_3)
8 将预测点直角坐标(x_3,y_3)即$([x_1+\Delta x],[(y_1+\Delta y])$，转换为极坐标$(\rho_3,\theta_3)$
9 得到偏差无人机最终的预测坐标，即为(ρ_3,θ_3)，此时理论上无限逼近于目标点(ρ_2,θ_2)

如上表 1 与下图 6，为了便于计算偏差无人机与已知无人机之间的距离，需要现将极坐标转换为直角坐标，从而可以使用欧式距离公式$D=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$进行距离计算。进而，本文提出的偏差无人机调度算法通过不断循环迭代，调整偏差无人机响应点对应的直角坐标(x_1,y_1)，直到$\Delta\alpha$无限接近于 0，取得$min\Delta\alpha$时无人机的最终响应点的直角坐标(x_3,y_3)，记为预测点P，转换得到其极坐标为(ρ_3,θ_3)。此时理论上，预测点的极坐标(ρ_3,θ_3)应无限逼近于目标点的极坐标(ρ_2,θ_2)，但无法到达目标点，因此，可认为预测点即为响应点动态变化过程中的最优解。

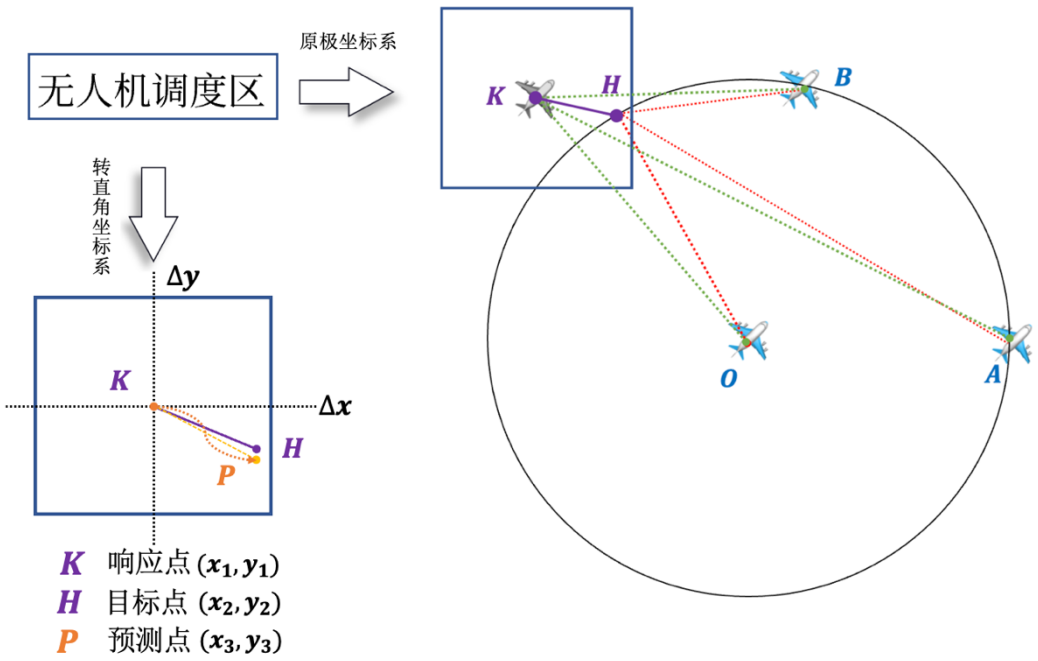


图 6 无人机调度区示意图

在此基础上，可以通过对比预测点坐标与目标点坐标进行比较，从而对本文提出的极坐标系模型与偏差无人机调度算法进行评价，进而可证明本文模型与算法的精度与有效性。例如上图 6 中，可比较目标点H与最终预测点P之间的偏差，偏差越小，则可证明本文提出模型算法的精度越高。

在测试过程中，为了适应本题一(1)中无人机$FY00$与圆周上另外 2 架已知无人机位置均无偏差的情况，如表 2 所示，本文修正了题中提供的无人机初始位置数据集，将编号 2 无人机的初始极坐标修正为$(100,40.00)$，使得数据集符合本题情况。从而可通过测试修正后的数据集，得出各架偏差无人机最终的预测点位置，进而可评估最终预测点和正确目标点之间的偏差，对模型做出评价：

表 2 算法 1 测试数据集的极坐标结果对比表

编号	初始响应点 K	正确目标点 H	最终预测点 P
0	**（0 , 0）**	**（0 , 0）**	**（0.000000 , 0.000000）**
1	**（100 , 0）**	**（100 , 0）**	**（100.0000 , 0.000000）**
2（修正）	**（100 , 40）**	**（100 , 40）**	**（100.0000 , 40.00000）**
3	（112 , 80.21）	（100 , 80）	（99.98508 , 80.01293）
4	（105 , 119.75）	（100 , 120）	（99.96316 , 120.0121）
5	（98 , 159.86）	（100 , 160）	（100.0512 , 159.9875）
6	（112 , 199.96）	（100 , 200）	（100.1042 , 199.9953）
7	（105 , 240.07）	（100 , 240）	（99.99280 , 240.0042）
8	（98 , 280.17）	（100 , 280）	（99.96830 , 279.9655）
9	（112 , 320.28）	（100 , 320）	（100.0293 , 320.0206）

如上表 2 所示，通过采用本文提出的偏差无人机调度算法对上述数据集进行测试，各架无人机最终的预测点P与正确的目标点H的拟合度极高、偏差极小，严谨地证明了本题模型的有效性与算法的合理性。

4.1.4 子问题二模型的分析

由上文所述可知，当$FY00$与另外两架编号确定的已知无人机可以遂行发射信号时，即可定位任意一架已知编号的偏差无人机的极坐标位置(ρ,θ)。但在本问题中，仅有$FY00$和$FY01$的两架发射信号的无偏差无人机的编号已知，基于此情况，接下来需要谈论如何用尽可能少的已知无人机，来实现对未知编号、被动接受信号无人机的定位。

因此，我们从仅有$FY00$和$FY01$两架已知无人机开始，逐步引入新的位于圆周上的已知编号无人机，模拟对未知编号无人机的定位。使用逐步证明的方法，来确认当前尽可能少的无人机，是否可以对偏差无人机实现有效定位。

4.1.5 子问题二模型的建立与求解

在本题中，首先已有$FY00$和$FY01$的两架发射信号的无偏差无人机，如图 7，对当前情况的实例进行分析，由纯方位无源定位的原理可知$\angle OKA$的大小，令$\angle OKA = \alpha_1$。如图 7 所示，此时偏差无人机$FY0K$将会位于在ΔOKA的外接圆上，而圆弧$\overset{\frown}{OA}$对应的圆周角大小均为α_1，因此无法确定K点在圆上的哪个位置，即此时无法对无人机$FY0K$进行有效定位。

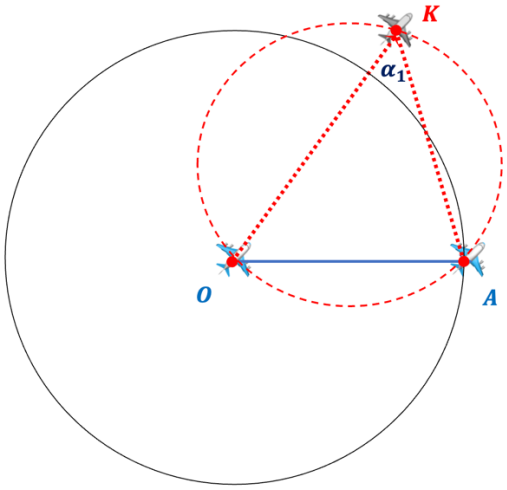


图 7 两架编号已知无人机的定位示意图

进而，如图 8 所示，本文引入第三架已知无人机，为不失一般性，令其编号为$FY0B$，其中$B \in \{2,3,\ldots,9\}$，可知图中$\alpha_1, \alpha_2, \alpha_3$，此时$\alpha_3 = \alpha_1 + \alpha_2$。

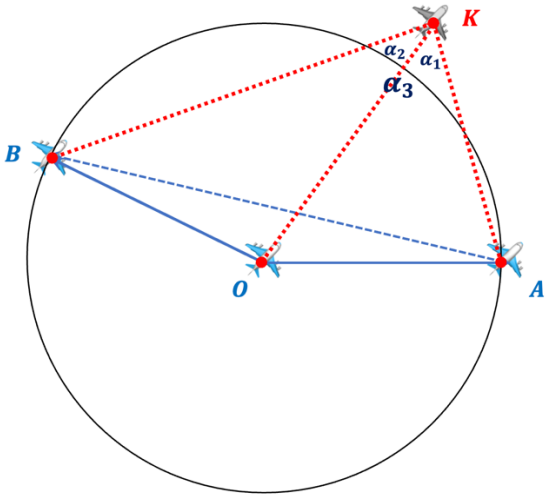
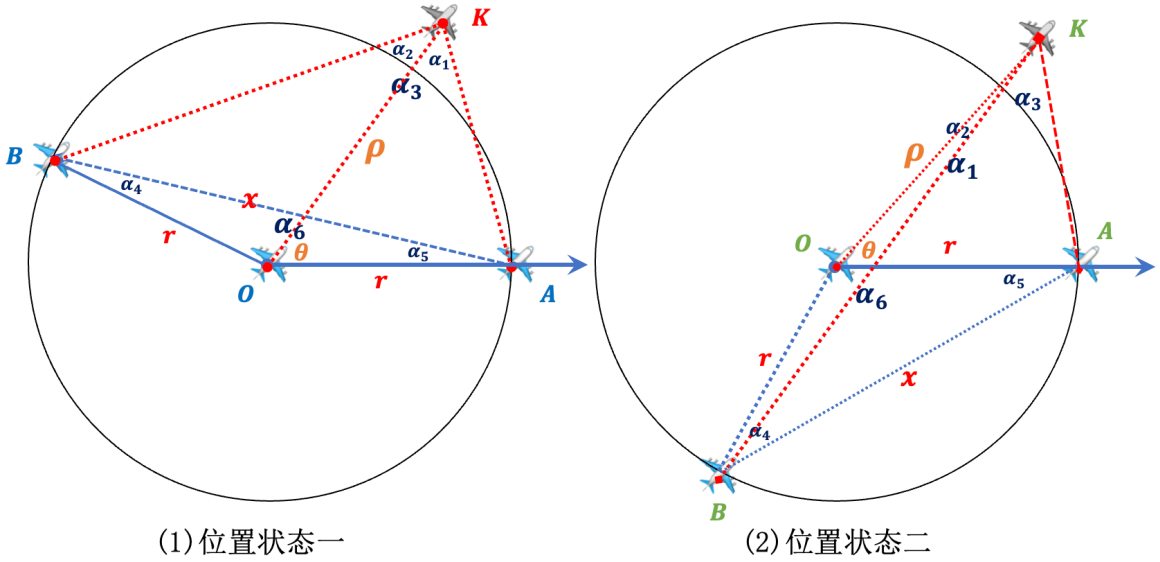


图 8 三架编号已知无人机的定位示意图

由上文分析可知，当存在三架位于圆周上的已知编号无人机发射信号时，即可确认编号已知的任一偏差无人机的定位。但考虑到当前有一架位于圆周上的无人机编号不确定，因此不能采用上文的实例分析，故本文针对该情况进行了一般化分析：

（一）如图 9 的状态一所示，当$2 \le B \le 5$，即$\tau \le \angle \mathrm{AOB} \le 4 \cdot \tau$，此时无人机$FY0B$位于该圆周的上半圆弧内的某个正确定位点。其中，$\angle \mathrm{AOB} = \alpha_6 = (B-1) \cdot \tau$，可推得$B = (\alpha_6/\tau) - 1$，即可以通过计算$B$的具体数值区分出所有$FY0B$的定位情况，故此时

在圆周上半圆弧的讨论具备一般性。在此基础上，根据上文中子问题一的模型，可知当确定了$FY00, FY01, FY0B$三架已知无人机后，便可以使用已知参数对$FY0K$偏差无人机的极坐标(ρ_2, θ_2)进行准确的描述（参照上文公式 2，3）。因此当$2 \le B \le 5$，除$FY00, FY01$外，还需要 1 架已知无人机，即可对偏差无人机实现有效定位。



(1)位置状态一　　(2)位置状态二

图 9 三架编号已知无人机的具体分类定位示意图

（二）如图 9 的状态二所示，当$6 \le B \le 9$，此时无人机$FY0B$位于该圆周的下半圆弧上。但考虑到$\angle AOB = \alpha_6 < \pi$，此时式子$\alpha_6 = (B-1) \cdot \tau$不再适用，若仍考虑$B = (\alpha_6/\tau) - 1$，则发现计算结果与图 9 状态一重复，所上文对响应角定义的启发，本文采用对比响应角的方法来区分相同B取值下的不同模型情况（原理如图 5）。

此时，考察图 9 状态一，响应角之间的相互关系为$\alpha_3 = \alpha_1 + \alpha_2$；考察图 9 状态二，响应角之间的相互关系为$\alpha_1 = \alpha_2 + \alpha_3$。因此，即使 B 点计算结果重复，依旧可以通过对比响应角关系来区分无人机$FY0B$的具体位置。在此基础上，同理可知，此时仍可以使用已知参数对$FY0K$偏差无人机的极坐标(ρ_2, θ_2)进行准确的描述。

综上所述，本文采用了逐步证明的方法，证明了在本题情况下，除了已知无人机$FY00, FY01$外，还需要 1 架无人机发射信号，即可实现对被动接受信息无人机的有效定位。

4.1.6 子问题三模型的分析

综合前文所述，基于子问题一，当存在三架位于圆周上的已知编号无人机发射信号时，即可根据已知参数推导表示出未知偏差无人机的级坐标(ρ, θ)；基于子问题二，除$FY00, FY01$外，仅需要再引入 1 架位于圆周上正确定位点的无人机，即可实现偏差无人机的有效定位。

基于以上结论，接下来考虑子问题三，已知题表给定的 10 架无人机的初始响应点，并知道每架无人机对应的正确目标点，如何通过多次调整，每次选择编号为$FY00$的无人机和圆周上最多 3 架无人机遂行发射信号，使得其余偏差无人机调度到正确的目标点上。其中，关键在于如何实现被动接收信号的无人机仅依据接收到的方向信息，从初始响应点最终逼近到正确的目标点上。

因此，本文基于子问题一中提出的单个偏差无人机的调度算法，结合题中信息，

对算法做出进一步的适应与优化，即使用 Python 设计出改进型偏差无人机群的调度算法，最终给出本问题情况下具体的调整方案。

4.1.7 子问题三模型的建立与求解

本问题给定了 10 架无人机的初始响应点，由均匀分布的条件可推知编号$FY00, FY01$两架无人机位于正确目标点，由上述子问题一、二可以得出，除$FY00, FY01$外，至少还需要 1 架发射信号的无人机，才能对其他偏差无人机实现有效定位与位置调度。基于上述原理并结合题意，此时除了$FY00, FY01$，还需要选取一架偏差无人机作为调度中介，记为$FY0B$，其中$B \in \{2,3,\dots,9\}$。则剩下的其他偏差无人机被动接受信号，记为$FY0K$，其中$K \in \{2,3,\dots 9\}\backslash\{B\}$。

与上文所提出的偏差无人机调度算法的原理一致，其他偏差无人机虽然不能发射信号反馈位置，但其能够计算自身目前响应角和目标角之间的差值，通过不断调整以缩小角度差距，从而不断逼近圆周上的目标正确点，即结果为缩小了无人机实时响应点到正确目标点之间的距离，记此偏差距离为d。

综上，结合本题信息与上文中极坐标系模型的原理，为了实现其他偏差无人机$FY0K$的逼近调整，本文将其中一架偏差无人机作为调度中介$FY0B$，通过$FY00, FY01, FY0B$来初步表示其他偏差无人机在极坐标系中的位置信息(ρ, θ)。如图10所示，按照编号顺序（逆时针）不断更替调度中介的编号，进而可以实现对所有偏差无人机的位置调度逼近。图 10 中，调度中介无人机的编号为红色，其他的偏差无人机的编号为绿色，已知位置正确的无人机的编号为蓝色。

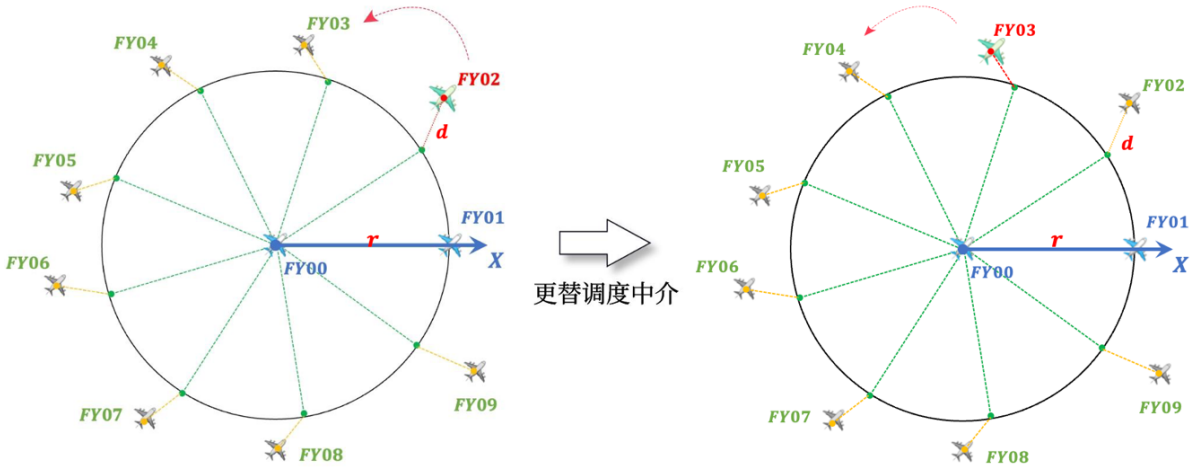


图 10 调度中介的更替示意图

基于上述原理，结合本题特征，现将整体无人机集群作为对象，把迭代范围扩大至无人机集合$FY0B$，其中$(B \in \{2,3,\dots,9\})$。进而，本文基于原单个偏差无人机的调度算法，提出了改进型的针对无人机集群的调度算法，该算法的伪代码如下表 3 所示（具体 Python 代码见附录）：

表 3 改进型偏差无人机群调度算法的伪代码

算法2：改进型偏差无人机群的调度算法（伪代码）

1 调度中介无人机$FY0B$通过纯方位无源定位获取初始响应夹角；

2 通过极坐标模型计算偏差无人机的响应点极坐标(ρ_1,θ_1)，目标点极坐标(ρ_2,θ_2)与已知无人机的极坐标，如点A为(ρ_A,θ_A)，点B为(ρ_B,θ_B)，其他同理；

3 将极坐标转化为直角坐标，如响应点极坐标(ρ_1,θ_1)转换为(x_1,y_1)即$(\rho_1 cos\theta_1,\rho_1 sin\theta_1)$，目标点极坐标$(\rho_2,\theta_2)$转化$(x_2,y_2)$即$(\rho_2 cos\theta_2,\rho_2 sin\theta_2)$，其他同理；

4 通过直角坐标计算出必要点之间的距离，以及各偏差无人机的目标响应点与目标点之间的偏差距离d_k，求出所有的偏差距离d_k之和，记为总偏差距离D_b；

5 此时更替调度中介，即按照编号顺序轮替$FY0B$中B的值，其中$B\in\{2,3,\dots,9\}$

// 更替调度中介后，随机进入新一轮循环迭代的过程

for i_in_range(100) // 循环迭代100次

调整响应点直角坐标，即令$(x_1+\Delta x)$，此时y_1不变，重新计算偏差距离之和D；

for i_in_range(100) // 循环迭代100次

再令$(y_1+\Delta y)$，此时再次重新计算偏差距离之和D_b；

// 不断更替调度中介，进行新一轮的循环迭代，使得总偏差距离D尽可能小，即偏差无人机尽可能逼近各自的最终目标点。

7 取上述迭代过程中的$min\Delta D_b$所对应的所有偏差无人机的直角坐标，记为预测点(x_{kn},y_{kn})，例如偏差无人机$FY02$为(x_{k2},y_{k2})，$FY03$为(x_{k3},y_{k3})，以此类推；

8 将预测点直角坐标(x_{kn},y_{kn})即$([x_1+\Delta x],[(y_1+\Delta y])$，转换为极坐标$(\rho_{kn},\theta_{kn})$；

9 得到各个偏差无人机最终的预测点即为(ρ_{kn},θ_{kn})，此时理论上无限逼近于各自的目标点

如上表 3，同上文已提出的算法 1 原理，先将极坐标转换为直角坐标，从而便于使用欧式距离公式$D=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$求解偏差各架无人机响应点与正确目标点之间的偏差距离d_k，进而可以求和得到总偏差距离D_b，其中偏差距离d_k与总偏差距离D_b之间的关系式为：

$$D_b=\sum d_k \text{ 其中 } (b=\{2,3,\cdots,9\},k=\{2,3,\dots,9\}\backslash\{b\}) \tag{4}$$

$$Z=minD_b \tag{5}$$

再如上图 10 所示，本文提出的算法 2（改进型偏差无人机的调度算法）首先考虑将偏差无人机$FY02$作为响应中介，进行第一轮的算法迭代，然后更替调度中介，将偏差无人机$FY03$作为响应中介，进行更新一轮的迭代计算，以此类推，最终取得总偏差距离的最小值$Z=minD_b$，取此时各个偏差无人机（包括作为调度中介的偏差无人机）的直角坐标，再各自转换为对应极坐标，称为各自的预测点。然后将各自的预测点与其正确目标点进行对比。同上文理，只要迭代循环的次数足够多，则各个偏差无人机的预测点将无限逼近于其目标点，但始终无法抵达目标点。

最终，本文使用改进型偏差无人机的调度算法对子问题三中的数据集进行测试，得到的最终结果如下表 4 所示：

表 4　算法 2 测试数据集的极坐标结果对比表

编号	初始响应点 K	正确目标点 H	最终预测点 P
0	（0 , 0）	（0 , 0）	（0.000000 , 0.000000）
1	（100 , 0）	（100 , 0）	（100.0000 , 0.000000）
2	（98 , 40.10）	（100 , 40）	（100.1709 , 39.79971）
3	（112 , 80.21）	（100 , 80）	（99.92151 , 79.89004）
4	（105 , 119.75）	（100 , 120）	（100.0997 , 120.0331）
5	（98 , 159.86）	（100 , 160）	（100.0512 , 159.9875）
6	（112 , 199.96）	（100 , 200）	（99.82237 , 200.0542）
7	（105 , 240.07）	（100 , 240）	（99.85620 , 240.0252）
8	（98 , 280.17）	（100 , 280）	（99.96830 , 279.9654）
9	（112 , 320.28）	（100 , 320）	（99.90088 , 320.1085）

如表 4 所示，通过采用本文提出的改进型偏差无人机的调度算法对本题数据集进行测试，各架偏差无人机最终的预测点P与正确的目标点H的偏差极小，再一次严谨地证明了本文模型的有效性与算法的优越性。

4.2 问题一模型的总结

综上所述，在问题一模型建立与求解的过程中，本文首先根据题目信息建立了极坐标系模型，结合本题中纯方位无源定位的方法，通过已知无人机的参数来描述偏差无人机的参数，由已知推算未知，从而求解得到偏差无人机具体的极坐标系定位模型。

在此基础上，为了实现偏差无人机的位置调度，本文创新性地定义了无人机调度过程中的描述方法：响应点K、目标点H与预测点P。其中，响应点K为偏差无人机在调度过程中的动态位置描述；目标点H为偏差无人机希望尽可能逼近的优化结果描述；而预测点P为偏差无人机在不断计算、调整的过程中，最后能够达到的一个最优解的描述。通过定义上述的调度过程描述方法，使得本文在公式推理、数值运算的过程中富有逻辑、思路明确，同时让后续的算法设计拥有坚实的理论支撑。

再次，本文在充分理解题意的前提下，结合极坐标系模型，进而使用 Python 程序语言设计出偏差无人机的调度算法，通过不断地循环迭代来尽可能缩小偏差，从而使得偏差无人机逼近其正确目标点，最终实现了被动接受信号无人机的有效定位与位置调度。

最后，针对解决无人机集群整体的调度方案，本文立足于单个偏差无人机的调度算法，进一步结合题目要求，提出了改进型、面向机群整体的偏差无人机调度算法，并对题中数据集进行了测试，结果表明，本文模型的预测值精度均在 99.9%以上，严谨地证明了模型的高效性与科学性。

4.3 问题二模型的建立与求解

4.3.1 问题二模型的分析

综合上文所述，问题一设定的是 10 架无人机组成的圆形遂行编队，且规定该编队圆周上无偏差的已知无人机发射信号，其余的有偏差无人机被动接收信号，通过已知无人机的参数来描述偏差无人机的参数，从而求解得到偏差无人机具体的极坐标系定位模型，并基于极坐标模型设计出高效科学的偏差无人机群调度算法。

现在进一步讨论，在实际生活中，无人机集群也可以是其他编队队形。为求解问题二中锥形编队模型，本文将问题一中的圆形编队模型进一步推广，充分结合题意，对偏差无人机调度算法进行更深入的设计改良，从而设计出基于 Python 的反馈型偏差无人机调度算法，最终给出问题二情况下具体的调整方案。

4.3.2 问题二模型的建立与求解

在本题锥形编队的情况下，结合上述问题一模型求解过程中的考虑，为了便于描述无人机之间的距离及方位关系，从而便于通过已知参数来求解未知参数，本文假定$FY14, FY15$为无偏差无人机，从而可以通过$FY15$的位置为原点O，从O点出发引一条经过无人机$FY14$的射线，记为Y轴，再以垂直于射线OY的方向建立X轴，如图 11 所示，可以建立出平面直角坐标系XOY。

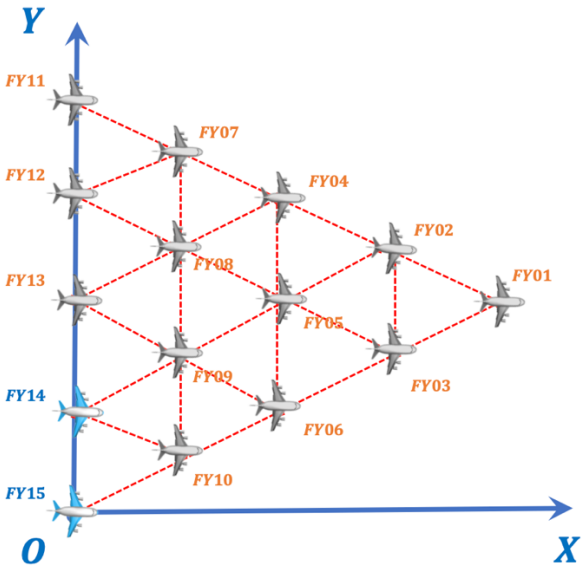


图 11 锥形编队的平面直角坐标系示意图

由题可知，两两相邻的无人机之间距离均为 50 米，因此可以确定图 11 中任意一件无人机的直角坐标。例如$FY15$为$(0,0)$，$FY10$为$(25\sqrt{3}, 25)$，FY01 为$(100\sqrt{3}, 100)$，以此类推。对每个无人机的坐标进行定位后，为符合题意，除了设定正确的$FY14$和$FY15$，需要给其它无人机位置添加随机偏差。例如$FY10$为$(25\sqrt{3}+\Delta x, 25+\Delta y)$，$FY01$为$(100\sqrt{3}+\Delta x, 100+\Delta y)$，以此类推。进而，锥形编队在平面直角坐标系上的位置将出现整体偏差。如图 12 所示，此时以偏差无人机$FY13$为例，画出其偏差情况，可推广到除$FY14$和$FY15$外的所有偏差无人机。

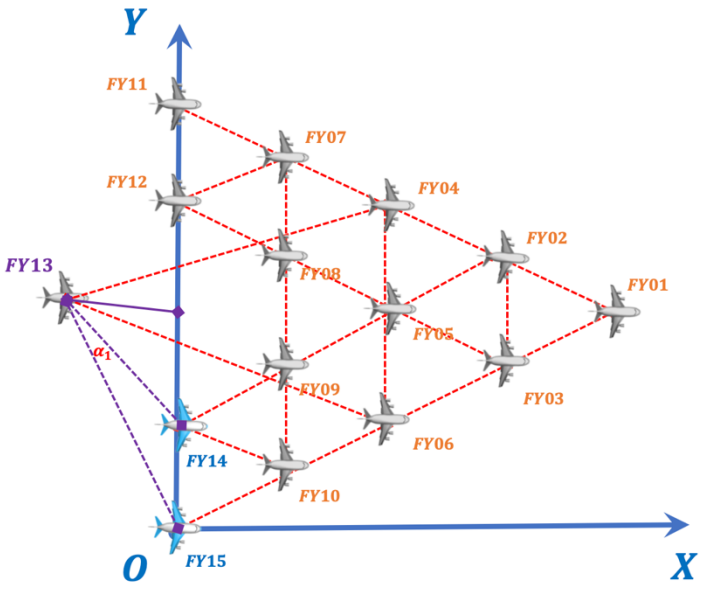


图 12 施加偏移量后平面直角坐标系示意图

上文通过对除$FY14$和$FY15$外的无人机位置添加随机偏差，使得锥形队形产生整体偏差，从而得到符合本题要求的实际状况下的无人机锥形偏差队列。接下来，本文通过进一步拓展本文问题一中的偏差无人机调度算法，并结合本题情况，提出基于 Python 的反馈型偏差无人机调度算法，该算法的伪代码如下表 5 所示（具体 Python 代码见附录）：

表 5 反馈型偏差无人机调度算法的伪代码

算法3：反馈型偏差无人机调度算法（伪代码）
1 基于$FY14$和$FY15$对无人机$FY11, FY12, FY13$的横坐标定位进行调整；
// 不断迭代，尽可能使响应角差$\Delta\alpha$无限逼近于0，即尽可能使得FY13无限接近于目标点
2 基于$FY13, FY14, FY15$对第二排的$FY07, FY08, FY09, FY10$进行调整；
3 基于调整后的$FY07, FY08, FY09$对第一排的$FY11, FY12, FY13$进行反馈调整；
// 反馈迭代调整为本算法的核心，进一步提高调度算法的精度
4 再次基于$FY14$和$FY15$对无人机$FY11, FY12, FY13$的定位进行调整；
// 经过步骤4的调整后，已经可以实现FY13的$\Delta\alpha$非常小，无限逼近于0
5 基于$FY13, FY14, FY15$对锥形队列中的所有无人机进行调整。

如上表 5 所述，问题提出的反馈型偏差无人机调度算法通过不断的循环迭代与反馈调整，最终使得各个偏差无人机的响应点逐渐逼近其对应的目标点，从而实现基于锥形队列的偏差无人机的定位调整。如表 6 所示，为评估反馈型偏差无人机调度算法的有效性，本文通过对锥形队列生成一组随机偏差，使用本算法对随机生成的数据集进行测试，得出以下结果：

表 6 算法 3 测试数据集的直角坐标结果对比表

编号	正确目标点 H	最终预测点 P
$FY15$	（0 , 0）	（0.000000 , 0.000000）
$FY14$	（0 , 50）	（-2.842e-14 , 49.999）
$FY13$	（0 , 100）	（0.016659 , 100.0574）
$FY12$	（0 , 150）	（-0.03138 , 150.0453）
$FY11$	（0 , 200）	（0.022214 , 199.9671）
$FY10$	（43.3012 , 25）	（43.28537 , 24.96887）
$FY09$	（43.3012 , 75）	（43.34051 , 74.97637）
$FY08$	（43.3012 , 125）	（43.19883 , 124.9676）
$FY07$	（43.3012 , 175）	（43.34232 , 175.0641）
$FY06$	（86.6025 , 50）	（86.55988 , 50.08030）
$FY05$	（86.6025 , 100）	（86.62937 , 99.96954）
$FY04$	（86.6025 , 150）	（86.45959 , 149.9678）
$FY03$	（129.9038 , 75）	（129.9148 , 75.02314）
$FY02$	（129.9038 , 125）	（130.0291 , 124.9159）
$FY01$	（173.2050 , 100）	（173.2078 , 100.0122）

由表 6 所示，通过采用本文提出的反馈型偏差无人机群调度算法对随机生成的数据集进行测试，各架偏差无人机最终的预测点 P 与正确的目标点 H 的拟合准确率均超过 99.8%，证明了本反馈型模型的科学性与高效性。

五、 模型的评价、改进与推广

5.1 模型的优点

（1）二维极坐标系模型：此模型将无人机之间的方向信息用相关角度进行描述，只要确定未知参数ρ与θ，即可用数学语言描述偏差无人机的定位信息。

（2）偏差无人机的调度算法：不断减小响应夹角α_1与目标夹角α_2的差值$\Delta\alpha$，使响应点不断逼近正确目标点。通过偏差无人机的调度算法可以得到数据集，得出各架偏差无人机最终的预测点位置，进而可评估最终预测点和正确目标点之间的偏差。对数据集的最终预测点和正确目标点的坐标进行相对误差分析，可以得出 ρ 的平均相对误差为 0.000393，θ 的平均相对误差为 0.000081,二者的平均相对误差均极小，能够严谨地证明了模型的有效性与算法的合理性。

（3）改进型偏差无人机的调度算法：通过足够多次的迭代计算，使各个偏差无人机的预测点无限逼近于其正确目标点，但始终无法抵达目标点。根据数据集结果的对比，可以得出 ρ 的平均相对误差为 0.000391，θ 的平均相对误差为 0.001065，可以得出各架偏差无人机最终的预测点P与正确的目标点H的偏差极小，再一次严谨地证明了本文模型的有效性与算法的优越性。

（4）考虑实际飞行中，无人机集群也可以是其他编队队形的情况，本文给出了锥形编队队形的纯方位无源定位情形，并给出了相应的调整方案，在一定程度上证实了其他编队队形的可行性。

5.2 模型的缺点

（1）求解模型运用的偏差无人机的调度算法及改进型偏差无人机的调度算法类似于一种穷举的方法，这种算法针对此问题无人机个数较少的情况下求出的准确度较高。但如果研究对象数量较大，算法的运行速率可能会减缓，数据结果准确度可能会下降。

（2）模型假定无人机基于自身感知的高度信息，均保持在同一个高度上飞行，因此可以将无人机看作是在二维平面上进行分析，但现实生活中可能是三维立体空间的情况，缺少了对此情况的进一步探讨。

5.3 模型的推广

本题通过设计偏差无人机调度算法，解决无人机编队飞行中的纯方位无源定位的问题，本文建立的模型能够有效定位无人机的位置，以及对有偏差的无人机进行位置的调整，使其不断逼近到正确的目标点。因此，该模型不仅能对无人机进行纯方位无源定位，还可以推广到宇宙空间站、卫星、火箭等应用领域，通过发射信号及接收信号的过程提取方向信息，进行有效定位以及位置的调整。

参考文献

[1] 汲万峰, 王肖飞等. 一种用于无人机军事应用的航路规划及避障方法, CN114859976A [P/OL].
[2] 邱华鑫, 段海滨等. 基于鸽群行为机制的多无人机自主编队 [J]. 控制理论与应用, 2015, 32(10): 1298-1304.
[3] 邱奕鸿, 陈鹏. 无线遥控无人机在消防灭火救援领域的应用研究 [J]. 消防界(电子版), 2022, 8(15): 71-72+76.
[4] 陈锐滨. 基于卫星与地面站的无源定位跟踪算法研究 [D]; 电子科技大学, 2019.
[5] 杨巍. 基于贝叶斯估计的多站纯方位无源定位及优化 [D]; 南京理工大学, 2015.
[6] 王欢, 刘树光等. 基于分层控制结构的有人/无人机编队队形控制 [J]. 电光与控制: 1-8.
[7] 张莞玲, 赵莲莲. 基于无人机群的频率步进雷达数字信号无源定位方法 [J]. 自动化与仪器仪表, 2022, (06): 61-65.
[8] 李林, 崔黎明等. 基于无人机群的电磁干扰信号干扰源定位系统及定位方法, CN115052294A [P/OL].

附录

附录 1

介绍：本文所有代码基于 Python 语言

```
# 本行代码用 python 语言编写。
# 作用：解决 问题一的第一小问

import numpy as np
import cmath
import random
import matplotlib.pyplot as plt

# 传入的是两个飞机的 直角坐标[x,y]，两个列表
# 输出的是两个飞机的距离

def get_dist(drone_a, drone_b):
    # 计算任意两架无人机的距离
    return np.sqrt((drone_a[0] - drone_b[0]) ** 2 + (drone_a[1] - drone_b[1]) **
2)

# 传入的三个列表，里面都是[p,sita]，三个列表
# 输出的是，那个夹角，α
def get_angle(receiver, drone_sender_1, drone_sender_2):
    # 使用余弦定理计算夹角
    a = get_dist(drone_sender_1, drone_sender_2)
    b = get_dist(receiver, drone_sender_1)
    c = get_dist(receiver, drone_sender_2)
    return np.arccos((b ** 2 + c ** 2 - a ** 2) / (2 * b * c)) * 180 / np.pi

# 算δ(α)——>>0，找一个误差最小的情况
def diff(angle_a, angle_b):
 #   print("np.sum(((np.array(angle_a) - np.array(angle_b)) * 100) ** 2) 得到
diff : ",np.sum(((np.array(angle_a) - np.array(angle_b)) * 100) ** 2))
    return np.sum(((np.array(angle_a) - np.array(angle_b)) * 100) ** 2)


# 画散点图用的
def plt_coords(coords):
    x = []
    y = []
    for i in coords:
        x.append(i[0])
        y.append(i[1])
    plt.scatter(x, y)
```

```
    plt.show()


# 表1数据
coords_polar_before = [[0, 0],[100, 0],[100, 40],[112, 80.21],[105,
119.75],[98, 159.86],[112, 199.96],[105, 240.07],[98, 280.17],[112, 320.28]]
# 无人机位置无偏差极坐标

coords_polar_aim = [[0, 0],[100, 0],[100, 40],[100, 80],[100, 120],[100,
160],[100, 200],[100, 240],[100, 280],[100, 320]]

coords_cartesian = []
coords_cartesian_aim = []

# 极坐标列表—》变成直角坐标列表

def polar_to_cartesian(coords):
    # 极坐标转换为直角坐标
    f = np.pi / 180
    return [coords[0] * np.cos(coords[1] * f), coords[0] * np.sin(coords[1] *
f)]

# 直角坐标列表—》变成极坐标列表

def cartesian_to_polar(coords):
    # 直角坐标转换为极坐标
    c_polar  = list(cmath.polar(complex(coords[0], coords[1])))
    c_polar [1] = c_polar [1] * 180 / np.pi
    if c_polar [1] < 0:
        c_polar [1] += 360
    return c_polar


for coords in coords_polar_before:
    coords_cartesian.append(polar_to_cartesian(coords))

for coords in coords_polar_aim:
    coords_cartesian_aim.append(polar_to_cartesian(coords))


plt_coords(coords_cartesian_aim)
plt_coords(coords_cartesian)

# 根据两个夹角来推测位置
```

```
def get_location_from_angle(drone_id, angle, drone_sender_1, drone_sender_2,
drone_sender_3):
    ad_range = 16
    setp = 0.1
    coords_diff = []
    min_diff = 1000000
    min_diff_idx = -1
    for i in np.arange(ad_range * -1, ad_range + setp, setp):
        for j in np.arange(ad_range * -1, ad_range + setp, setp):

            location_right = list(coords_cartesian_aim[drone_id])
            #location 应该为 实际值
            location = list(coords_cartesian[drone_id])

    #        print("理想飞机直角_location : ",location)
            location[0] += i
            location[1] += j
 #          print("location[0] += i  : ",location[0])
 #          print("location[1] += j  : ", location[1])
            diff_result = diff([get_angle(location, drone_sender_1,
drone_sender_2), get_angle(location, drone_sender_1,
drone_sender_3)],[get_angle(location_right,drone_sender_1,drone_sender_2),get_a
ngle(location_right,drone_sender_1,drone_sender_3)])
            # print(" 0 , 1 , 2 的角度 ",get_angle(location, drone_sender_1,
drone_sender_2))
            # print(" 0 , 1 , 3 的角度 ",get_angle(location, drone_sender_1,
drone_sender_3))
            # print("α 初始，",np.sum(np.array(get_angle(location, drone_sender_1,
drone_sender_2))+np.array(get_angle(location, drone_sender_1,
drone_sender_3))))
            # print("a 标准
",np.sum(np.array([get_angle(location_right,drone_sender_1,drone_sender_2),get_
angle(location_right,drone_sender_1,drone_sender_3)])))
            # print("初 diff_result （差值*100 再平方）: ",diff_result)
            # 找更小的 diff_result 值，赋值给 min_diff,然后再找索引
            if min_diff > diff_result:
                min_diff = diff_result
                min_diff_idx = len(coords_diff)
            coords_diff.append([[np.round(location[0], 2), np.round(location[1],
2)], np.round(diff_result, 2)])
            # print("min_diff ",min_diff)
            # print("coords_diff : ",coords_diff)
    return coords_diff[min_diff_idx]
```

```
"""
假设编号为0，1，2的无人机发送信号，它们的位置无偏差。3,4,5,6,7,8,9号无人机的坐标有偏差，
使用表1数据(其中，二号飞机的数据是修正过的)，根据夹角推测每个接受信号的飞机的位置
"""
#7次，从0~6，我们需要3~9
for i in range(7):
    drone_id=i+3
    angle =
[get_angle(coords_cartesian[drone_id],coords_cartesian_aim[0],coords_cartesian_
aim[1]),get_angle(coords_cartesian[drone_id],coords_cartesian_aim[0],coords_car
tesian_aim[2])]

    location_result =
get_location_from_angle(drone_id,angle,coords_cartesian_aim[0],coords_cartesian
_aim[1],coords_cartesian_aim[2])

    print(f"第{i+3}架飞机的推测的位置，极坐标:
{cartesian_to_polar(location_result[0])}")
    print(f"第{i+3}架飞机的实际的位置，极坐标:
{cartesian_to_polar(coords_cartesian[drone_id])}")

print("==========================================================================
===============")
```

附录 2

介绍：本文代码基于 Python 实现

```
# 本代码由 python 编写
# 作用：解决问题一，第三小问；以及问题二

import numpy as np
import cmath
import random
import matplotlib.pyplot as plt

print("问题一，第三小问")
print("\n")

# 传入的是两个飞机的 直角坐标[x,y]，两个列表
# 输出的是两个飞机的距离
def get_dist(UVA_a, UVA_b):
    # 计算任意两架无人机的距离
    return np.sqrt((UVA_a[0] - UVA_b[0]) ** 2 + (UVA_a[1] - UVA_b[1]) ** 2)
```

```
# 传入的三个列表，里面都是[p,sita]，三个列表
# 输出的是，那个夹角，α
def get_angle(receiver, UVA_sender_1, UVA_sender_2):
    # 使用余弦定理计算夹角
    a = get_dist(UVA_sender_1, UVA_sender_2)
    b = get_dist(receiver, UVA_sender_1)
    c = get_dist(receiver, UVA_sender_2)
    return np.arccos((b ** 2 + c ** 2 - a ** 2) / (2 * b * c)) * 180 / np.pi

# 算δ(α)——>>0，找一个误差最小的情况
def diff(angle_a, angle_b):
 #   print("np.sum(((np.array(angle_a) - np.array(angle_b)) * 100) ** 2) 得到
diff : ",np.sum(((np.array(angle_a) - np.array(angle_b)) * 100) ** 2))
    return np.sum(((np.array(angle_a) - np.array(angle_b)) * 100) ** 2)


# 画散点图用的
def plt_coords(coords):
    x = []
    y = []
    for i in coords:
        x.append(i[0])
        y.append(i[1])
    plt.scatter(x, y)
 #   plt.show()

# 表1数据
coords_polar_before = [[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]]
# 无人机位置无偏差极坐标
coords_polar_right = [[0, 0],[100, 0],[100, 40],[100, 80],[100, 120],[100,
160],[100, 200],[100, 240],[100, 280],[100, 320]]
coords_cartesian = []
coords_cartesian_right = []

# 极坐标列表——》变成直角坐标列表
def polar_to_cartesian(coords):
    # 极坐标转换为直角坐标
    f = np.pi / 180
    return [coords[0] * np.cos(coords[1] * f), coords[0] * np.sin(coords[1] *
f)]

# 直角坐标列表——》变成极坐标列表
```

```
def cartesian_to_polar(coords):
    # 直角坐标转换为极坐标
    c_polar  = list(cmath.polar(complex(coords[0], coords[1])))
    c_polar [1] = c_polar [1] * 180 / np.pi
    if c_polar [1] < 0:
        c_polar [1] += 360
    return c_polar


for coords in  coords_polar_before:
    coords_cartesian.append(polar_to_cartesian(coords))

for coords in  coords_polar_right:
     coords_cartesian_right.append(polar_to_cartesian(coords))

plt_coords(coords_cartesian_right)
plt_coords(coords_cartesian)

# 根据两个夹角确定位置
def angle_transfer_to_coords(UVA_id, angle, UVA_sender_1, UVA_sender_2,
UVA_sender_3):
    ad_range= 16
    setp = 0.5
    coords_diff = []
    min_diff = 1000000
    min_diff_idx= -1
    for i in np.arange(ad_range * -1,ad_range+ setp, setp):
        for j in np.arange(ad_range * -1,ad_range+ setp, setp):

            location_right = list( coords_cartesian_right[UVA_id])
            #coords 应该为 实际值
            coords = list(coords_cartesian[UVA_id])

    #        print("理想飞机直角_coords : ",coords)
            coords[0] += i
            coords[1] += j

 #          print("coords[0] += i  : ",coords[0])
 #          print("coords[1] += j  : ", coords[1])

            cost_result = diff([get_angle(coords, UVA_sender_1, UVA_sender_2),
get_angle(coords, UVA_sender_1,
UVA_sender_3)],[get_angle(location_right,UVA_sender_1,UVA_sender_2),get_angle(l
ocation_right,UVA_sender_1,UVA_sender_3)])
```

```
            # print(" 0 , 1 , 2 的角度 ",get_angle(coords, UVA_sender_1,
UVA_sender_2))
            # print(" 0 , 1 , 3 的角度 ",get_angle(coords, UVA_sender_1,
UVA_sender_3))
            # print("α 初始，",np.sum(np.array(get_angle(coords, UVA_sender_1,
UVA_sender_2))+np.array(get_angle(coords, UVA_sender_1, UVA_sender_3))))
            # print("a 标准
",np.sum(np.array([get_angle(coords_right,UVA_sender_1,UVA_sender_2),get_angle(
coords_right,UVA_sender_1,UVA_sender_3)])))
 #           print("初 diff_result （差值*100 再平方）：",diff_result)
            # 找更小的 diff_result 值，赋值给 min_diff,然后再找索引
            if  min_diff > cost_result:
                min_diff = cost_result
                min_diff_idx= len(coords_diff)
            coords_diff.append([[np.round(coords[0], 2), np.round(coords[1], 2)],
np.round(cost_result, 2)])
#           print("min_diff ",min_diff)
  #         print("location_diff : ",location_diff)
    return coords_diff[min_diff_idx]


def adj_coords(UVA_test_coords, sender_a):
    distance_family = []
    min_distance_data = 0
    min_distance_indx = 0
    test_coords_family = []
    testing_coords_middle_family = []
    testing_location_big_family = []
    sum_distance_before = 0

    for i in range(1, len(UVA_test_coords)):
        # 迭代计算 总距离，每个目的点与当前点的距离
        sum_distance_before += get_dist(UVA_test_coords[i],
coords_cartesian_right[i])
    print(f"调整前各无人机与理想位置的距离之和: {sum_distance_before}")


    #  这边的 len(UVA_test_coords)=10 就完事。  for i in [1,10)
    for sender_b in range(1, len(UVA_test_coords)):

        if sender_b == sender_a:
            continue

        distance = 0  # 52.0826
```

```
        sum_distance_after = 0
        for i in range(1, len(UVA_test_coords)):
            if i not in [sender_b, sender_a]:
                # 把 UVA_id 改成 received_UVA 比较好
                receiver_id = i
                angle =
[get_angle(UVA_test_coords[receiver_id],UVA_test_coords[0],UVA_test_coords[send
er_a]),get_angle(UVA_test_coords[receiver_id],UVA_test_coords[0],UVA_test_coord
s[sender_b])]

# 第三小问，仍然是 1+2 就可以了，由 0 ，a ,b 确定其余的 receiver 点，并纠正，即可。这边的
coords 是 receiver 的预测坐标！！！！
                coords =
angle_transfer_to_coords(receiver_id,angle,coords_cartesian_right[0],coords_car
tesian_right[sender_a],coords_cartesian_right[sender_b])

                UVA_test_coords[i][0]=coords[0][0]
                UVA_test_coords[i][1]=coords[0][1]

# b 不变时，收集到遍历过 接收端的 点的坐标，存在 family 里面

test_coords_family.append([UVA_test_coords[i][0],UVA_test_coords[i][1]])
                distance = get_dist(UVA_test_coords[i],
coords_cartesian_right[i])
#                print("刚刚调整后的极坐标  ",cartesian_to_polar(UVA_test_coords[i]))
#                print("dist",distance)
                # print("test_coords_family  ",test_coords_family)


            sum_distance_after = sum_distance_after+distance

#             testing_coords_middle_family.append(test_coords_family)
#             print("testing_coords_middle_family ",testing_coords_middle_family)

        # 将得到的 sum_distance 存到 family 里面

        print(f"我们使用 0, {sender_a}, {sender_b},这三架无人机发射信号来调整无人机位
置，调整后各无人机与目标位置的距离之和: {sum_distance_after}")
        distance_family.append(sum_distance_after)
        # print("distance_family",distance_family)

#
testing_location_big_family.append(testing_coords_middle_family[sender_b-1])
#       print("testing_location_big_family ",testing_location_big_family)
```

```
    min_distance_data=min(distance_family)
    min_distance_indx=distance_family.index(min_distance_data)
 #   print("min_distance_data ",min_distance_data)
 #   print(" min_distance_indx ", min_distance_indx)
    testing_coords_middle_family = test_coords_family[min_distance_indx *
7:(min_distance_indx + 1) * 7]
    # print("testing_coords_middle_family ", testing_coords_middle_family)
    # for h in range(7):
    #     print("之前算出来的极坐标值，",cartesian_to_polar(UVA_test_coords[h]))
#    print("之前的极坐标值，", cartesian_to_polar(UVA_test_coords))

    flag = 0
    for k in range(7):
        if (min_distance_indx) == (k):
#           print("continue")
            flag=1
            continue
    #       UVA_test_coords[k+2+1]=testing_coords_middle_family[k+1]
        if flag == 1:
            UVA_test_coords[k+2]=testing_coords_middle_family[k-1]
        else:
           UVA_test_coords[k+2]=testing_coords_middle_family[k]
#    print("调整之后的直角坐标值，",UVA_test_coords)

    print(f"通过比较发现当我们使用 ,0 ,1 ,{min_distance_indx+2},这三架无人机发射信号来调
整无人机位置，调整后各无人机与目标位置的距离之和:{distance_family[min_distance_indx]}")
    for g in range(10):
        print("调整之后获得的极坐标值，", cartesian_to_polar(UVA_test_coords[g]))

    return UVA_test_coords

    #    print(f"第 {k}次的替换结果 {UVA_test_coords[k+2]}")

    # for i in range(10):
    #     print(test_coords_family[i])


UVA_test_coords = []

for i in range(len(coords_cartesian)):
    UVA_test_coords.append(list(coords_cartesian[i]))
    # print("(coords_cartesian[i])",i," 次",(UVA_coords_cartesian[i]))
```

```
    # print("UVA_testing_coords[i] ",i,"次 ",UVA_testing_coords[i])
# 就是得到了 testing_点 跟 直角坐标点一样的情况。


# 在这里开始执行调用!
# n 为迭代次数
n=1
for i in range(n):
#    print(f"第 {i + 1} 次调整:")
    UVA_test_coords=adj_coords(UVA_test_coords, 1)  # FY00, FY01 发送信号


# print("res ,",UVA_test_coords)


UVA_test_coords_polar = []
for coords in UVA_test_coords:
    UVA_test_coords_polar.append(cartesian_to_polar(coords))

print("-------------------------------------------------------------------------
---------------------------------------")
print("问题二")
print("\n")

# print(UVA_test_coords_polar)
#
# 计算所有误差距离(由 testing 点到 ideal 点的，直角坐标情况)，然后再求和
def calcuate_all_dist(UVA_test_coords):
    distance = 0
    for i in range(1, len(UVA_test_coords)):
        distance += get_dist(UVA_test_coords[i],  coords_cartesian_right[i])
    return distance
#
def w_adj_coords(UVA_test_coords, UVA_sender_1, UVA_sender_2, UVA_sender_3,
receiver):
    print(f"校准前的所有误差值累和为: {calcuate_all_dist(UVA_test_coords)}")
    ad_range= 8
    setp = 0.1
    # id 值，在接收的无人机中遍历
    for UVA_id in receiver:
        coords_diff = []
        min_diff = 1000000
        min_diff_idx= -1
        # 产生一个[-8,8+n*1)的偏移增值
```

```
        for i in np.arange(ad_range * -1,ad_range+ setp, setp):
            for j in np.arange(ad_range * -1,ad_range+ setp, setp):
                # coords 是 testing_点中的，某一点直角坐标
                coords = list(UVA_test_coords[UVA_id])
                # 这边对 某个点的，x,y 的坐标做增量的偏移
                coords[0] += i
                coords[1] += j
                # 这边的 angle 是当前位置的 angle 角【a1,a2】
                angle = [get_angle(coords, UVA_test_coords[UVA_sender_1],
UVA_test_coords[UVA_sender_2]),get_angle(coords, UVA_test_coords[UVA_sender_1],
UVA_test_coords[UVA_sender_3])]

                #又是 diff，计算逼近角度
                angle_cost = diff(angle,[get_angle(coords_cartesian_right[UVA_id],
UVA_test_coords[UVA_sender_1],
UVA_test_coords[UVA_sender_2]),get_angle(coords_cartesian_right[UVA_id],
UVA_test_coords[UVA_sender_1], UVA_test_coords[UVA_sender_3])])
                if angle_cost <  min_diff:
                    min_diff = angle_cost
                    min_diff_idx = len(coords_diff)
                coords_diff.append([coords, angle])
        # 把索引对应的 坐标轴给找到
        UVA_test_coords[UVA_id] = coords_diff[min_diff_idx][0]
 #       print(f"输出了{i}: ",location_diff[min_diff_idx][0])
    print(f"校准后的所有误差值累和为: {calcuate_all_dist(UVA_test_coords)}")



def w_adj_first_column(UVA_test_coords):
    # 这边的 ad_range= 8 就是随机数里面生成的系数
    ad_range= 8
    setp = 0.1
    # 为啥在 2,3,4 中改变？【因为第一列，1~5，而 1,2 不用变，我们就改变其他的点】
    for UVA_id in [2, 3, 4]:
        coords_diff = []
        min_diff = 1000000
        min_diff_idx= -1
        #    x 的增量[-8,8+0.1*n)
        for i in np.arange(ad_range * -1,ad_range+ setp, setp):
            # coords 为 id 中的 testing 中的某个点的 极坐标 【p,sita】，为啥这边只改变一个
值？？？每次都是 p 在变化，sita 都不改变
            # 更新，这边都是对，接收端的飞机，x 值，进行逼近
            coords = list(UVA_test_coords[UVA_id])
            coords[0] += i
```

```
            # 这边的angle中，由改变的增量值找到一个更小的，α，【在这边甚至都不需要 diff 函数，δ(α)，只找最小的 α 值】
            # 通过改变p,从而算出最小的α
            #UVA_test_coords[0], UVA_test_coords[1] 也就是f15,f14
            # coords里面有【x,y】
            angle = get_angle(coords, UVA_test_coords[0], UVA_test_coords[1])
            if angle <  min_diff:
                min_diff = angle
                min_diff_idx= len(coords_diff)
            coords_diff.append([coords, angle])
        UVA_test_coords[UVA_id] = coords_diff[min_diff_idx][0]
        print("第",UVA_id,"架飞机，进行横坐标校准,获得其坐标值:
" ,UVA_test_coords[UVA_id])



# 无人机如锥形编队位置无偏差直角
x = np.sqrt(50 * 50 - 25 * 25)
coords_cartesian_right = [[0, 0],[0, 50],[0, 100],[0, 150], [0, 200],[x,
25],[x, 75],[x, 125],[x, 175],[2 * x, 50],[2 * x, 100],[2 * x, 150],[3 * x,
75],[3 * x, 125],[4 * x, 100]]

plt_coords(coords_cartesian_right)


# 除了FY14和FY15，给其它无人机位置添加随机偏移
coords_cartesian = []
# 15次循环嘛
for i in range(len(coords_cartesian_right)):
    #这边的coords是飞机直角坐标的理想值
    coords =  coords_cartesian_right[i]
    coords_cartesian.append(list(coords))
    # 跳过 f14,f15
    if i in [0, 1]:
        continue
    # 这边的random.random() 就是生成了【0,1】的随机数  ，这个后面乘的 8 可以再优化改进！！！！！！！！！！！！
    # dr_location_car的坐标，一开始其实就是和coords的坐标一样的，只不过，在下面的操作中，跳过f14,f15；给其他飞机加偏移量
    if random.random() >= 0.5:
        coords_cartesian[i][0] = coords[0] + random.random() * 8
    else:
        coords_cartesian[i][0] = coords[0] - random.random() * 8
```

```
    if random.random() >= 0.5:
        coords_cartesian[i][1] = coords[1] + random.random() * 8
    else:
        coords_cartesian[i][1] = coords[1] + random.random() * 8

plt_coords(coords_cartesian)


UVA_test_coords = []
# 仍然 15 次遍历，把刚刚添加了随机数的结果的 copy 放到了 dr_testing_location 里面
for i in range(len(coords_cartesian)):
    UVA_test_coords.append(list(coords_cartesian[i]))
distance = 0
# 再 15 次
for i in range(1, len(UVA_test_coords)):
     distance = distance + get_dist(UVA_test_coords[i],
coords_cartesian_right[i])

print("未开始校准前,各无人机与其对应目标点的距离的总和为 : ",distance)


print("由 FY14、FY15，率先校准第一列的 x 轴坐标，使得第一列能够对齐")
w_adj_first_column(UVA_test_coords)
plt_coords(UVA_test_coords)


print("由处于第一列的 FY13、FY14 和 FY15，开始进行第二列的校准")
w_adj_coords(UVA_test_coords, 0, 1, 2, range(5, 9))
plt_coords(UVA_test_coords)


print("由第二列 FY08、FY09 和 FY10，再开始第一列的校准")
w_adj_coords(UVA_test_coords, 5, 6, 7, [1, 2, 3, 4])
plt_coords(UVA_test_coords)

print("再让第一列与 FY15 对齐")
w_adj_first_column(UVA_test_coords)
plt_coords(UVA_test_coords)

print("由 FY13、FY14 和 FY15，开始对其他无人机的校准")
w_adj_coords(UVA_test_coords, 0, 1, 2, range(5, 15))
plt_coords(UVA_test_coords)
    print("校准后各无人机的直角坐标     ",UVA_test_coords)
```