无人机遂行编队飞行中的纯方位无源定位

摘要

随着科技的发展，在无人机编队飞行中，利用相对角度等信息进行纯方位无源定位成为一个重要的研究方向。本文基于**正弦定理**，提出了一种针对多架无人机在圆形编队中定位的模型。通过几何分析得出，确定三架定位飞机后，即可求解其他飞机的位置。随后，提出了一种多架无人机定位调整的迭代模型，通过多次迭代优化无人机的位置。本文还研究了在纯方位无源定位条件下，无人机在等边三角形阵列中的位置优化模型，并设计了有效的调整方案，为无人机集群控制提供了理论参考。

针对问题一第一小问，本文利用边角之间的关系，巧用**正弦定理**来求无人机的定位模型。首先，通过极坐标方式定义每架无人机的位置，再通过被动接收无人机在k点接收到的三个角度信息，定义了无人机之间的相对位置关系。根据不同飞机的编号顺序，夹角大小等因素，分了三大类共九种情况，利用正弦定理将角度关系转化为边长关系，然后通过将不同的正弦表达式结合，得出k点无人机的极角和半径信息，构建出相应的定位模型。

针对问题一第二小问，本文探讨了利用多架无人机进行定位的模型。当只有两架无人机时，由于信息对称性，定位结果存在重复，而引入第三架无人机后，能够通过接收不同的角度信息来唯一确定无人机的位置。本文通过举例具体分析说明的方式，表明还需要**一架**无人机发射信号，才能实现无人机的有效定位。

针对问题一第三小问，根据第二问得出的结论，再结合欧式距离的计算，选择出最优的一架飞机来构成三架飞机协助定位，根据第一问建立的模型以及本问的实际情况，最终确定本问共6种情况，遍历剩余无人机并根据第一问的模型进行求解，从而更新位置，然后再重新选择一架飞机进行定位，循环迭代并更新位置不断收敛，最终得到最优解。最终总体误差率由原先的**5.21%** 降低至 **3.29%**。

针对问题二，基于问题一中无人机之间已实现相互定位的基础，本文提出了一种无人机阵列优化模型。该模型结合了最优分配、旋转矩阵和平移操作。首先，通过随机生成无人机的初始位置来建立基准。接着，将理想阵列的质心与初始位置的质心对齐，并通过旋转优化调整阵列方向，以减少每个无人机与其目标位置的距离。为实现最优匹配，构建了距离矩阵并采用匈牙利算法解决分配问题。最后，利用局部搜索和交换优化策略，逐步减少无人机的总移动距离，达到整体路径优化的目的，为无人机的定位提供了有效的解决方案。

关键词：无人机编队；无源定位；正弦定理；优化

一、 问题重述

无人机集群在遂行编队飞行时，为避免外界干扰，应尽可能保持电磁静默，少向外发射电磁波信号。为保持编队队形，拟采用纯方位无源定位的方法调整无人机的位置，即由编队中某几架无人机发射信号、其余无人机被动接收信号，从中提取出方向信息进行定位，来调整无人机的位置。编队中每架无人机均有固定编号，且在编队中与其他无人机的相对位置关系保持不变。接收信号的无人机所接收到的方向信息约定为：该无人机与任意两架发射信号无人机连线之间的夹角。例如：编号为 FY01、FY02 及 FY03 的无人机发射信号，编号为FY04 的无人机接收到的方向信息是α_1、α_2和α_3。

请建立数学模型，解决以下问题：

问题 1 编队由 10 架无人机组成，形成圆形编队，其中 9 架无人机（编号 FY01 FY09）均匀分布在某一圆周上，另 1 架无人机（编号 FY00）位于圆心。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。

(1) 位于圆心的无人机（FY00）和编队中另 2 架无人机发射信号，其余位置略有偏差的无人机被动接收信号。当发射信号的无人机位置无偏差且编号已知时，建立被动接收信号无人机的定位模型。

(2) 某位置略有偏差的无人机接收到编号为 FY00 和 FY01 的无人机发射的信号，另接收到编队中若干编号未知的无人机发射的信号。若发射信号的无人机位置无偏差，除 FY00 和 FY01外，还需要几架无人机发射信号，才能实现无人机的有效定位?

(3) 按编队要求，1 架无人机位于圆心，另 9 架无人机均匀分布在半径为 100 m 的圆周上。当初始时刻无人机的位置略有偏差时，请给出合理的无人机位置调整方案，即通过多次调整，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，其余无人机根据接收到的方向信息，调整到理想位置（每次调整的时间忽略不计），使得 9 架无人机最终均匀分布在某个圆周上。利用表 1 给出的数据，仅根据接收到的方向信息来调整无人机的位置，请给出具体的调整方案。

实际飞行中，无人机集群也可以是其他编队队形，例如锥形编队队形（题中图3）。仍考虑纯方位无源定位的情形，设计无人机位置调整方案。

二、 问题分析

2.1 问题一第一小问的分析

该题要求在由 10 架无人机组成的圆形编队中，建立被动接收信号无人机的定位模型，其中位于圆心的无人机（FY00）和编队中另 2 架无人机发射信号，其余位置略有

偏差的无人机被动接收信号，发射信号的无人机位置无偏差且编号已知。由于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，本文参考 [10]的解题思路，通过极坐标方式定义每架无人机的位置，分别用(R, θ)表示。考虑到 [9]中的正弦定理解题方法，本文通过将不同的正弦表达式结合，用被动接收无人机在K点接收到的三个角度定义无人机之间的相对位置关系，通过这些角度计算位置，得出无人机在K点的极角和半径之间的关系，使得复杂的模型关系式变得可解。

2.2 问题一第二小问的分析

该题要求确定还需要发射信号无人机的数量来实现无人机的有效定位，其中编号为 FY00 (0, 0) 和 FY01 (R, 0)，还有若干编号未知的无人机发射信号，发射信号的无人机位置无偏差。因此，本文先假设只有两架无人机进行定位的情况，根据几何对称性和相对角度的信息，这种情况可能会有多个相同的解。因此，本文考虑增加到三架无人机的情况，通过几何关系能唯一确定目标无人机的位置。三架无人机能够提供多个角度信息，在本文的例子中，假设无人机1发射的角度为50°，无人机2发射的角度为70°，此时可以通过三角形的几何关系唯一确定目标无人机（无人机3）的位置。如果接收到的角度信息是50°（来自无人机1）和30°（来自无人机2），同样可以通过几何分析区分并确定目标无人机的位置，使得至少需要三架无人机来确定飞机的位置。

2.3 问题一第三小问的分析

该题要求合理选择发射信号的无人机组合，利用其发送的方向信息，逐步调整和优化其他无人机的位置信息，最终实现均匀分布的编队效果。该题的核心在于选择一架飞机作为主定位飞机，通过它与其他固定飞机共同发射信号，辅助其他飞机计算并调整其位置达到均匀分布的效果。对于剩余的8架飞机，本文通过计算它们与理想位置的欧式距离，选择一架差值最小的飞机来进行定位，然后根据第一小问的模型进行求解，更新其他7架飞机位置，再重新选择一架飞机进行定位，循环迭代并更新位置不断收敛，为防止无效计算，模型设定了在误差率小于2%时停止迭代，或者当迭代次数超过120次时则退出，最终得到最优解。模型最终得到以下结果：大多数飞机的位置在经过多次迭代后逐渐接近它们的理想位置,总体误差率从初始的5.21%降至最终的3.29%，并趋于稳定；绝大多数飞机的误差率收敛于2%以内，几乎与理想位置重合；仅有编号为3和9的飞机未能彻底收敛，但相对于它们的初始位置，位置得到了显著改善。

2.4 问题二的分析

该题要求在纯方位无源定位的情形下，按照锥形编队的特定条件设计无人机的调整方案。综合考虑无人机的定位关系、信号发送与接收机制，参考 [8]的方法，本文在问题一中无人机之间已经实现相互定位的基础上，即无人机的位置可作为输入的已知量，实现其他矩阵的编队队形，构建为等边三角形的锥形编队队形。因此，本文建立一种基于最优分配、旋转矩阵和平移操作的无人机阵列优化模型。通过将一开始随机生成的15个无人机（点）从随机位置调整为理想的等边三角形锥形阵列，同时优化无人机的移动距离为最小值。

三、 模型假设

考虑到实际情况以及不确定因素，以及模型构建的合理性，本文做出以下假设：

1. 由于无人机的飞行高度相同，意味着它们在一个平面内运行，只有平面的二维坐标 (x,y) 需要考虑，不需要考虑高度因素。

2. 本文假设题中所给数据均真实可靠，无异常值的出现。

3. 为简化模型，本题不考虑风向，天气等外在因素对无人机的影响。

四、 符号说明

符号	表示含义
R	圆的半径
r_k	k位置飞机的实际半径
$\theta_i, \theta_j, \theta_k$	点 i, j, k 的极角（弧度）
(R, θ_i)	点 i 的极坐标表示
(R, θ_j)	点 j 的极坐标表示
(R, θ_k)	点 k 的极坐标表示
x, y	平面直角坐标系中的横纵坐标
θ	无人机在圆周上的极角
r	无人机的飞行半径（对于圆周上的无人机，$r = R$）
α_1	o, k, i 三点构成的角度
α_2	i, k, j 三点构成的角度

α_3	j,k,o 三点构成的角度
o	圆心，无人机 FY00 的位置
i,j	固定发出信号的无人机的位置
k	被动接收信息的无人机的位置

五、 模型的建立与求解

5.1 问题一第1小问

5.1.1 模型的建立

对于无人机编队问题，目前已有许多相关研究。鲁 [12]等人针对无人机的编队控制，提出了一种领航-跟随算法，能够使得普通无人机在领航无人机的引导下，实现跟随无人机的控制。 [6]中提出采用全局规划和局部规划来共同对无人机进行编队控制。而 [4]中则提出采用分布式控制策略来解决无人机编队问题，且通过实验证明其比集中控制策略更优。本文参考分布式控制策略思路，通过建立被动接收信号无人机的定位模型，来解决问题一第1小问。

假设题目中该圆其半径为 R，圆心为O，代表一架固定发出信号的飞机的位置。在圆周上有多个点 i,j,k ，i，j点是固定发出信号飞机的位置点，k点是被动接收信息的飞机。每个点的位置由其极坐标 $(R,\theta_i),(R,\theta_j),(R,\theta_k)$ 表示，其中 R 是圆的半径，θ 是极角。

待定位的飞机$FY0k$会接收到三个角度，本文为了后续分类讨论的严谨性，为三个角度按照一定规则顺序进行命名，如下式 (1) 所示。

$$\begin{cases} \angle oki = a_1 \\ \angle ikj = a_2 \\ \angle jko = a_3 \end{cases} \tag{1}$$

在上式1中，假定$\theta_i < \theta_j$恒成立。

根据θ_k 的大小，可以分成如下三种大类：

第一大类：$\theta_k < \theta_i < \theta_j$，此时根据$\theta_k$ 与 θ_i 的夹角大小，以及 θ_k 与 θ_j 的夹角大小，可以进行细分成以下三种情况，如图1所示。

其中，(a)对应的情况为：$\theta_i - \theta_k < \pi, \theta_j - \theta_k < \pi$；(b)图对应的情况为：$\theta_i - \theta_k < \pi, \theta_j - \theta_k > \pi$；(c)图对应的情况为：$\theta_i - \theta_k > \pi, \theta_j - \theta_k > \pi$。

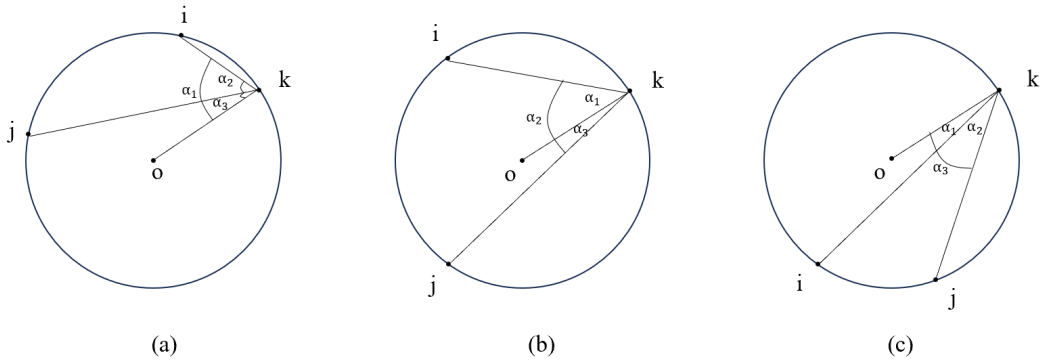


图 1: 第一种大类情况示意图

对于图(a)：其 $\angle koi = \theta_i - \theta_k$，$\angle koj = \theta_j - \theta_k$。

关于正弦定理的相关分析， [11]中进行了较为具体的阐述。在图(a)中连接oi，oj两条边，并对$\triangle koi$、$\triangle koj$分别使用正弦定理，可以得到如下关系：

$$\triangle koi \text{ 的正弦定理:} \quad \frac{R}{\sin\alpha_1} = \frac{r_k}{\sin(\alpha_1 + \theta_i - \theta_k)} \tag{2}$$

$$\triangle koj \text{ 的正弦定理:} \quad \frac{R}{\sin\alpha_3} = \frac{r_k}{\sin(\alpha_3 + \theta_j - \theta_k)} \tag{3}$$

对于图(b)：其 $\angle koi = \theta_i - \theta_k$，$\angle koj = 2\pi + \theta_k - \theta_j$。

此时连接oi，oj两条边，并对$\triangle koi$、$\triangle koj$分别使用正弦定理，可以得到如下关系：

$$\triangle koi \text{ 的正弦定理:} \quad \frac{R}{\sin\alpha_1} = \frac{r_k}{\sin(\alpha_1 + \theta_i - \theta_k)} \tag{4}$$

$$\triangle koj \text{ 的正弦定理:} \quad \frac{R}{\sin\alpha_3} = \frac{r_k}{\sin(\alpha_3 + \theta_k - \theta_j)} \tag{5}$$

对于图(c)：其 $\angle koi = 2\pi + \theta_k - \theta_i$，$\angle koj = 2\pi + \theta_k - \theta_j$。

此时连接oi，oj两条边，并对$\triangle koi$、$\triangle koj$分别使用正弦定理，可以得到如下关系：

$$\triangle koi \text{ 的正弦定理:} \quad \frac{R}{\sin\alpha_1} = \frac{r_k}{\sin(\alpha_1 + \theta_k - \theta_i)} \tag{6}$$

$$\triangle koj \text{ 的正弦定理:} \quad \frac{R}{\sin\alpha_3} = \frac{r_k}{\sin(\alpha_3 + \theta_k - \theta_j)} \tag{7}$$

第二大类：$\theta_i < \theta_k < \theta_j$，此时根据$\theta_k$ 与 θ_i 的夹角大小，以及 θ_k 与 θ_j 的夹角大小，可以进行细分成以下三种情况，如图2所示。

其中，(a)对应的情况为：$\theta_k - \theta_i < \pi$, $\theta_j - \theta_k < \pi$；(b)图对应的情况为：$\theta_k - \theta_i < \pi$,

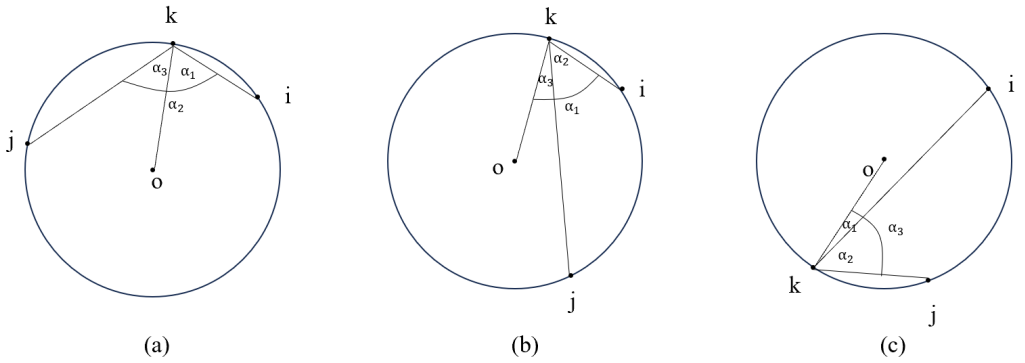


图 2: 第二种大类情况示意图

$\theta_j - \theta_k > \pi$；(c)图对应的情况为：$\theta_k - \theta_i > \pi$, $\theta_j - \theta_k < \pi$。

对于图(a)：其 $\angle koi = \theta_k - \theta_i$，$\angle koj = \theta_j - \theta_k$。

此时连接oi，oj两条边，并对$\triangle koi$、$\triangle koj$分别使用正弦定理，可以得到如下关系：

$$\triangle koi \text{ 的正弦定理:} \quad \frac{R}{\sin\alpha_1} = \frac{r_k}{\sin(\alpha_1 + \theta_k - \theta_i)} \tag{8}$$

$$\triangle koj \text{ 的正弦定理:} \quad \frac{R}{\sin\alpha_3} = \frac{r_k}{\sin(\alpha_3 + \theta_j - \theta_k)} \tag{9}$$

对于图(b)：其 $\angle koi = \theta_k - \theta_i$，$\angle koj = 2\pi + \theta_k - \theta_j$。

此时连接oi，oj两条边，并对$\triangle koi$、$\triangle koj$分别使用正弦定理，可以得到如下关系：

$$\triangle koi \text{ 的正弦定理:} \quad \frac{R}{\sin\alpha_1} = \frac{r_k}{\sin(\alpha_1 + \theta_k - \theta_i)} \tag{10}$$

$$\triangle koj \text{ 的正弦定理:} \quad \frac{R}{\sin\alpha_3} = \frac{r_k}{\sin(\alpha_3 + \theta_k - \theta_j)} \tag{11}$$

对于图(c)：其 $\angle koi = 2\pi + \theta_i - \theta_k$，$\angle koj = \theta_j - \theta_k$。

此时连接oi，oj两条边，并对$\triangle koi$、$\triangle koj$分别使用正弦定理，可以得到如下关系：

$$\triangle koi \text{ 的正弦定理:} \quad \frac{R}{\sin\alpha_1} = \frac{r_k}{\sin(\alpha_1 + \theta_i - \theta_k)} \tag{12}$$

$$\triangle koj \text{ 的正弦定理:} \quad \frac{R}{\sin\alpha_3} = \frac{r_k}{\sin(\alpha_3 + \theta_j - \theta_k)} \tag{13}$$

第三大类：$\theta_i < \theta_j < \theta_k$，此时根据$\theta_k$ 与 θ_i 的夹角大小，以及 θ_k 与 θ_j 的夹角大小，可以进行细分成以下三种情况，如图3所示。

其中，(a)对应的情况为：$\theta_k - \theta_i < \pi$, $\theta_k - \theta_j < \pi$；(b)图对应的情况为：$\theta_k - \theta_i > \pi$,

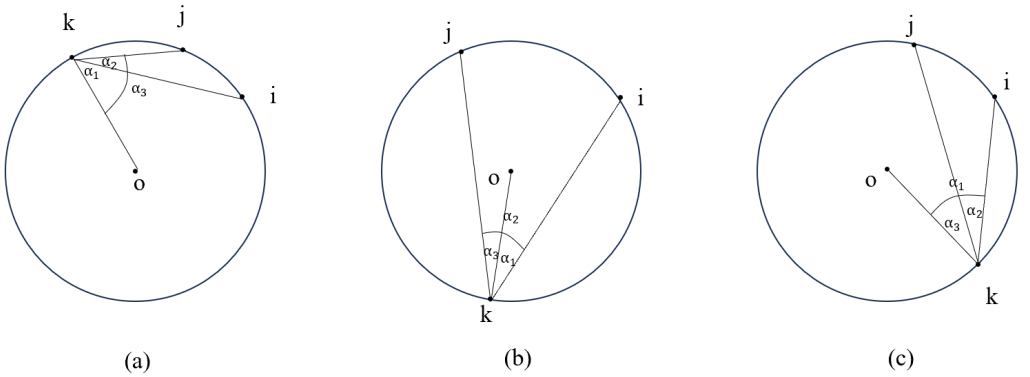


图 3: 第三种大类情况示意图

$\theta_k - \theta_j < \pi$；(c)图对应的情况为：$\theta_k - \theta_i > \pi$, $\theta_k - \theta_j > \pi$。

对于图(a)：其 $\angle koi = \theta_k - \theta_i$，$\angle koj = \theta_k - \theta_j$。

此时连接oi，oj两条边，并对$\triangle koi$、$\triangle koj$分别使用正弦定理，可以得到如下关系：

$$\triangle koi \text{ 的正弦定理:} \quad \frac{R}{\sin\alpha_1} = \frac{r_k}{\sin(\alpha_1 + \theta_k - \theta_i)} \tag{14}$$

$$\triangle koj \text{ 的正弦定理:} \quad \frac{R}{\sin\alpha_3} = \frac{r_k}{\sin(\alpha_3 + \theta_k - \theta_j)} \tag{15}$$

对于图(b)：其 $\angle koi = 2\pi + \theta_i - \theta_k$，$\angle koj = \theta_k - \theta_j$。

此时连接oi，oj两条边，并对$\triangle koi$、$\triangle koj$分别使用正弦定理，可以得到如下关系：

$$\triangle koi \text{ 的正弦定理:} \quad \frac{R}{\sin\alpha_1} = \frac{r_k}{\sin(\alpha_1 + \theta_i - \theta_k)} \tag{16}$$

$$\triangle koj \text{ 的正弦定理:} \quad \frac{R}{\sin\alpha_3} = \frac{r_k}{\sin(\alpha_3 + \theta_k - \theta_j)} \tag{17}$$

对于图(c)：其 $\angle koi = 2\pi + \theta_i - \theta_k$，$\angle koj = 2\pi + \theta_j - \theta_k$。

此时连接oi，oj两条边，并对$\triangle koi$、$\triangle koj$分别使用正弦定理，可以得到如下关系：

$$\triangle koi \text{ 的正弦定理:} \quad \frac{R}{\sin\alpha_1} = \frac{r_k}{\sin(\alpha_1 + \theta_i - \theta_k)} \tag{18}$$

$$\triangle koj \text{ 的正弦定理:} \quad \frac{R}{\sin\alpha_3} = \frac{r_k}{\sin(\alpha_3 + \theta_j - \theta_k)} \tag{19}$$

5.1.2 模型的求解

由模型的建立可知，无人机的定位模型总共可分为9种情况，其中每种情况的等式关系的表现形式均相同，不同点处在于角度大小引起的部分变量改变。因此，以下我们选择第一种大类的子情况(a)进行细致求解，具体步骤如下：

step1: 正弦定理的应用

根据上述模型的建立，该情况的两个表达式为：

$$\frac{R}{\sin\alpha_1}=\frac{r_k}{\sin(\alpha_1+\theta_i-\theta_k)} \tag{20}$$

$$\frac{R}{\sin\alpha_3}=\frac{r_k}{\sin(\alpha_3+\theta_j-\theta_k)} \tag{21}$$

约束条件为：

$$\begin{cases}\theta_k<\theta_i<\theta_j\\ \theta_i-\theta_k<\pi\\ \theta_j-\theta_k<\pi\end{cases} \tag{22}$$

step2: 结合上述两个表达式

通过将两个正弦定理的表达式20和21相除，得：

$$\frac{\sin\alpha_3}{\sin\alpha_1}=\frac{\sin(\alpha_3+\theta_j-\theta_k)}{\sin(\alpha_1+\theta_i-\theta_k)} \tag{23}$$

step3: 利用三角函数的展开式和三角恒等式

使用正弦和余弦公式进行展开，得到：

$$\frac{\sin\alpha_3}{\sin\alpha_1}=\frac{\sin(\alpha_3+\theta_j)\cos(\theta_k)-\cos(\alpha_3+\theta_j)\sin(\theta_k)}{\sin(\alpha_1+\theta_i)\cos(\theta_k)-\cos(\alpha_1+\theta_i)\sin(\theta_k)} \tag{24}$$

step4: 化简求解

通过进一步的化简，得出：

$$\tan\theta_k=\frac{\sin\alpha_1\sin(\alpha_3+\theta_j)-\sin\alpha_3\sin(\alpha_1+\theta_i)}{\sin\alpha_1\cos(\alpha_3+\theta_j)-\sin\alpha_3\cos(\alpha_1+\theta_i)} \tag{25}$$

最终，表达式被化简为：

$$\theta_k = \arctan\left(\frac{\sin\alpha_1\sin(\alpha_3+\theta_j)-\sin\alpha_3\sin(\alpha_1+\theta_i)}{\sin\alpha_1\cos(\alpha_3+\theta_j)-\sin\alpha_3\cos(\alpha_1+\theta_i)}\right) \tag{26}$$

同时根据20式，可以得到：

$$r_k = \frac{R\sin(\alpha_1+\theta_i-\theta_k)}{\sin\alpha_1} \tag{27}$$

step5: 定位模型建立结果

综上所述，该情况对应的定位模型如下：

目标求解模型：

$$\begin{cases}\theta_k = \arctan\left(\dfrac{\sin\alpha_1\sin(\alpha_3+\theta_j)-\sin\alpha_3\sin(\alpha_1+\theta_i)}{\sin\alpha_1\cos(\alpha_3+\theta_j)-\sin\alpha_3\cos(\alpha_1+\theta_i)}\right),\\ r_k = \dfrac{R\sin(\alpha_1+\theta_i-\theta_k)}{\sin\alpha_1}\end{cases} \tag{28}$$

约束条件：

$$s.t.\begin{cases}\theta_k<\theta_i<\theta_j\\ \theta_i-\theta_k<\pi\\ \theta_j-\theta_k<\pi\end{cases} \tag{29}$$

step6: 其他情况求解

根据上述求解过程，同理可得其他八种情况的定位模型，具体结果如下：

第一大类情况(b)：

$$\begin{cases}\theta_k = \arctan\left(\dfrac{\sin\alpha_3\sin(\alpha_1+\theta_i)-\sin\alpha_1\sin(\alpha_3-\theta_i)}{\sin\alpha_1\cos(\alpha_3-\theta_j)+\sin\alpha_3\cos(\alpha_1+\theta_i)}\right),\\ r_k = \dfrac{R\sin(\alpha_1+\theta_i-\theta_k)}{\sin\alpha_1}\end{cases} \tag{30}$$

$$s.t.\begin{cases}\theta_k<\theta_i<\theta_j\\ \theta_i-\theta_k<\pi\\ \theta_j-\theta_k>\pi\end{cases} \tag{31}$$

第一大类情况(c)：

$$\begin{cases}\theta_k = \arctan\left(\dfrac{\sin\alpha_3\sin(\alpha_1+\theta_i)-\sin\alpha_1\sin(\alpha_3-\theta_j)}{\sin\alpha_1\cos(\alpha_3-\theta_j)+\sin\alpha_3\cos(\alpha_1+\theta_i)}\right),\\ r_k = \dfrac{R\sin(\alpha_1+\theta_i-\theta_k)}{\sin\alpha_1}\end{cases} \tag{32}$$

$$s.t.\begin{cases}\theta_k<\theta_i<\theta_j\\ \theta_i-\theta_k>\pi\\ \theta_j-\theta_k>\pi\end{cases} \tag{33}$$

第二大类情况(a)：

$$\begin{cases}\theta_k=\arctan\left(\dfrac{\sin\alpha_1\sin(\alpha_3+\theta_j)-\sin\alpha_3\sin(\alpha_1-\theta_i)}{\sin\alpha_3\cos(\alpha_1-\theta_i)+\sin\alpha_1\cos(\alpha_3+\theta_j)}\right),\\ r_k=\dfrac{R\sin(\alpha_1+\theta_k-\theta_i)}{\sin\alpha_1}\end{cases} \tag{34}$$

$$s.t.\begin{cases}\theta_i<\theta_k<\theta_j\\ \theta_k-\theta_i<\pi\\ \theta_j-\theta_k<\pi\end{cases} \tag{35}$$

第二大类情况(b)：

$$\begin{cases}\theta_k=\arctan\left(\dfrac{\sin\alpha_1\sin(\alpha_3-\theta_j)-\sin\alpha_3\sin(\alpha_1-\theta_i)}{\sin\alpha_3\cos(\alpha_1-\theta_i)-\sin\alpha_1\cos(\alpha_3-\theta_j)}\right),\\ r_k=\dfrac{R\sin(\alpha_1+\theta_k-\theta_i)}{\sin\alpha_1}\end{cases} \tag{36}$$

$$s.t.\begin{cases}\theta_i<\theta_k<\theta_j\\ \theta_k-\theta_i<\pi\\ \theta_j-\theta_k>\pi\end{cases} \tag{37}$$

第二大类情况(c)：

$$\begin{cases}\theta_k=\arctan\left(\dfrac{\sin\alpha_1\sin(\alpha_3+\theta_j)-\sin\alpha_3\sin(\alpha_1+\theta_i)}{\sin\alpha_1\cos(\alpha_3+\theta_j)-\sin\alpha_3\cos(\alpha_1+\theta_i)}\right),\\ r_k=\dfrac{R\sin(\alpha_1+\theta_i-\theta_k)}{\sin\alpha_1}\end{cases} \tag{38}$$

$$s.t.\begin{cases}\theta_i<\theta_k<\theta_j\\ \theta_k-\theta_i>\pi\\ \theta_j-\theta_k<\pi\end{cases} \tag{39}$$

第三大类情况(a)：

$$\begin{cases} \theta_k = \arctan\left(\dfrac{\sin\alpha_1\sin(\alpha_3-\theta_j)-\sin\alpha_3\sin(\alpha_1-\theta_i)}{\sin\alpha_3\cos(\alpha_1-\theta_i)-\sin\alpha_1\cos(\alpha_3-\theta_j)}\right), \\ r_k = \dfrac{R\sin(\alpha_1+\theta_k-\theta_i)}{\sin\alpha_1} \end{cases} \tag{40}$$

$$s.t.\begin{cases} \theta_i < \theta_j < \theta_k \\ \theta_k - \theta_i < \pi \\ \theta_k - \theta_j < \pi \end{cases} \tag{41}$$

第三大类情况(b)：

$$\begin{cases} \theta_k = \arctan\left(\dfrac{\sin\alpha_3\sin(\alpha_1+\theta_i)-\sin\alpha_1\sin(\alpha_3-\theta_j)}{\sin\alpha_1\cos(\alpha_3-\theta_j)+\sin\alpha_3\cos(\alpha_1+\theta_i)}\right), \\ r_k = \dfrac{R\sin(\alpha_1+\theta_i-\theta_k)}{\sin\alpha_1} \end{cases} \tag{42}$$

$$s.t.\begin{cases} \theta_i < \theta_j < \theta_k \\ \theta_k - \theta_i > \pi \\ \theta_k - \theta_j < \pi \end{cases} \tag{43}$$

第三大类情况(c)：

$$\begin{cases} \theta_k = \arctan\left(\dfrac{\sin\alpha_1\sin(\alpha_3+\theta_j)-\sin\alpha_3\sin(\alpha_1+\theta_i)}{\sin\alpha_1\cos(\alpha_3+\theta_j)-\sin\alpha_3\cos(\alpha_1+\theta_i)}\right), \\ r_k = \dfrac{R\sin(\alpha_1+\theta_i-\theta_k)}{\sin\alpha_1} \end{cases} \tag{44}$$

$$s.t.\begin{cases} \theta_i < \theta_j < \theta_k \\ \theta_k - \theta_i > \pi \\ \theta_k - \theta_j > \pi \end{cases} \tag{45}$$

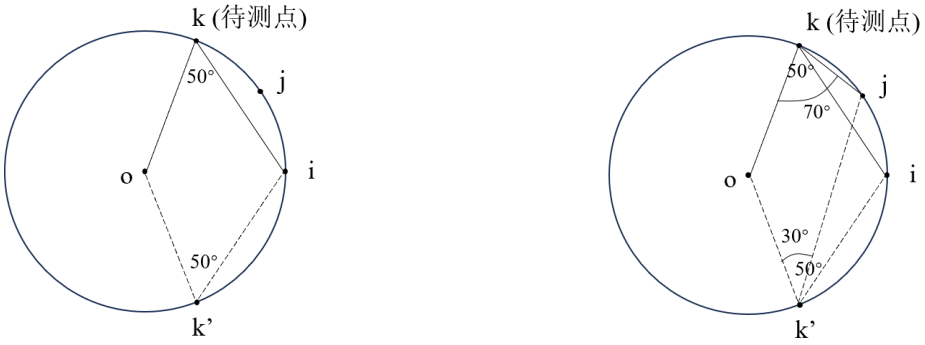
5.2 问题一第2小问

5.2.1 模型的建立

如果只有两架无人机来定位，则由于对称性，一个角度会得到两个相同的结果位置点k和k'。如下图4a所示：

然而，如果有三架无人机来定位，那么可以得到两个角度信息，从而唯一确定飞机的位置。如图4b所示。

其中，飞机 k 知道飞机 i 发射的定位信息是 50°，飞机 j 发射的定位信息是 70°，通



(a) 两架无人机的定位示意图　　(b) 三架无人机的定位示意图

图 4: 多个子图示例

过三角形的几何关系，我们可以唯一确定当前的飞机就是飞机 k。而如果是飞机 k'，则获得的第二个角度信息应是30°而不是 70°，因此我们可以区分并确定无人机的位置。

5.2.2 模型的求解

若增加一架飞机，则飞机k可以获得3个角度信息α_1、α_2、α_3，之后根据模型1 的求解方法，分情况并套用相应的公式，即可唯一确定飞机k的位置。

因此，需要增加一架无人机，才能确定飞机的位置，有三架定位飞机后，利用第一小问建立的定位模型，即可求解飞机的位置。

5.3 问题一第3小问

5.3.1 模型的建立

为了对飞机进行定位调整，由前两问可知，仍需增加一架飞机来发送信息，以便对其他飞机进行定位。因此，我们需要对其他 8 架飞机进行相关信息计算，选择出一架飞机作为定位飞机，对其他 7 架飞机进行一轮更新迭代。

1、选择定位飞机的策略

初始位置与理想位置越接近的飞机，成为定位飞机的效果越好，因此选择距离差最小的飞机作为定位飞机。初始位置与理想位置的距离采用欧式距离来计算，如下式 46 所示：

$$\rho_k = \sqrt{(x_{ki} - x_{kj})^2 + (y_{ki} - y_{kj})^2} \tag{46}$$

其中，x_{ki} 和 y_{ki} 分别表示飞机 k 初始位置的横纵坐标，x_{kj} 和 y_{kj} 分别表示飞机 k 理想位置的横纵坐标。由于题目给的数据是极坐标，因此需要将极坐标转换为直角坐标，再计算欧式距离。转换公式如下式48所示：

$$\begin{cases} x_i = r_i \cos(\theta_i) \\ y_i = r_i \sin(\theta_i) \end{cases} \tag{47}$$

2、更新迭代策略

在第一次选择完定位飞机后，与0、1号共3架飞机，一起对其他7架飞机进行定位。由第一问已知，已知3个点对外发出信息，其他被动接受信息的飞机都能推断出此时自己的位置。该位置由极坐标表示，公式如第一问28所示。

在计算过程中，需要根据三架定位飞机的位置，以及被定位飞机的位置，选择对应情况的公式进行计算。由于在本小问中，0，1号飞机固定作为发送信号的定位飞机，因此其中一架定位飞机$i=1$，另一架通过选择策略得到的定位飞机为j，被定位飞机为k。由于$i=1$，因此k必大于i，所以最后需要考虑的只有第一问中的第二大类，第三大类共6种情况。在本轮迭代中，对7架飞机进行遍历，分别作为被定位飞机k，根据不同情况，选择对应的公式进行计算，得到飞机的位置，并根据欧式距离计算是否更优来决定是否更新调整无人机的位置。最终计算后，可以得到其他7架飞机各自的极坐标信息(θ_k, r_k)。自此，第一轮迭代完成。

下一次迭代时，需要重新计算实际点与理想点的欧式距离，选出距离差最小的作为固定发送信号的飞机，然后重复上述步骤迭代。

3、迭代终止条件

由于每次用于迭代计算的数据都存在一定的误差，无限次迭代可能造成无法收敛，甚至结果越来越差的情况。因此，需要在每次迭代后计算误差率，当所有飞机的位置误差率均小于2%时或者迭代次数超过120次时，认为模型已收敛，退出循环。

5.3.2 模型的求解

根据模型的建立过程，求解步骤如下：

step1: 位置信息

首先，0和1号飞机的位置已知，且根据表格数据可知初始位置与理想位置无偏差。

step2: 选择迭代

其次，由前两问可知，再加一架飞机，便可进行定位，因此在其他8架飞机中，根据其初始位置与理想位置的距离差，选择一架差值最小的飞机来进行定位，对其他7架飞机进行一轮更新迭代。

step3: 更新策略

然后一轮迭代后，除了0和1号飞机固定外，重新选择一架最优飞机进行定位，重复步骤2

step4: 终止条件

每次迭代后都计算误差率，若所有飞机的位置误差率均<2%，认为模型已收敛，退出循环。

根据以上步骤进行求解，每10次迭代便作图记录一次各飞机的位置分布情况，如图5所示：

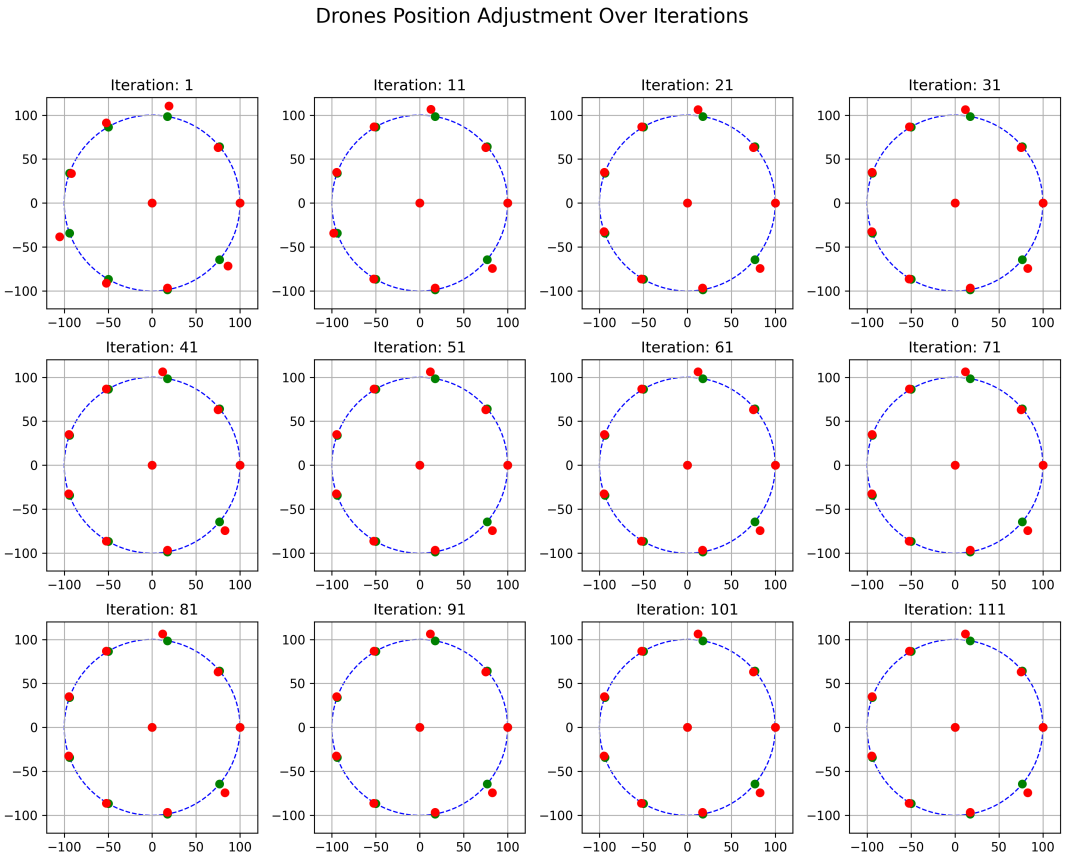


图 5: 迭代结果示意图

其中，绿色点代表各架飞机的理想位置，红色点代表各架飞机的实际位置，蓝色虚线代表半径为100米的圆的边界。由上图可知，在经过多次迭代后，各架飞机的位置逐渐接近理想位置，且误差率逐渐减小，具体总体误差率如图6所示：

由图6可知，总体误差率由原先的5.21%逐渐降至3.29%并趋于稳定。由图5可见，最终大部分点误差率收敛于2%以内，基本与理想位置点重合，只有编号为3和9的飞机无法有效收敛到最佳位置，但相比初始位置，位置也有较大改善。

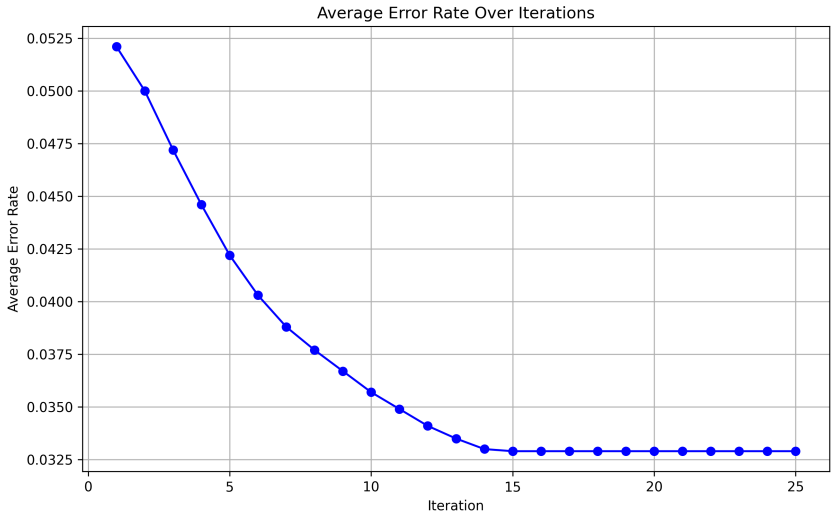


图 6: 总体误差率变化图

5.4 问题二

5.4.1 模型的建立

1、初始位置生成

以等边三角形锥形阵列为研究对象，设置3、6、10、15……等数字类型的无人机个数，本文设定无人机总数为15个，在二维平面上设置在（0，300）的范围内设置15个随机点作为随机情况下的无人机集群初始位置P =p1,p2,p3……p15，因此无人机的每个位置是一个二维向量。

2、理想阵列搭建

然后构建一个理想的等边三角形锥形阵列，即画出等边三角形阵列，设置等边三角形的边长为a，每个点的坐标计算为:

$$\begin{cases} x_{ij} = j \cdot a \\ y_{ij} = i \cdot a \cdot \dfrac{\sqrt{3}}{2} \end{cases} \tag{48}$$

其中i表示层数，j表示该层中的点数。我们仅取前15个点作为理想阵列，记为$T = \{t_1, t_2, t_3, \ldots, t_{15}\}$。

3、平移与旋转优化

本文采用先画出目标阵列，后续平移和旋转处理的方式，使得目标阵列可以让无人机移动较少距离完成对于阵列形状目标的完成。

平移操作即使得理想阵列的质心与无人机初始位置的质心对其。质心代表了整个

位置集的几何中心，通过将质心对齐，可以大大减少之后优化过程中的计算量。质心对齐后，无人机的位置集和理想阵列的位置集在整体上已经较为接近，这使得后续的旋转和匹配操作能够在一个更小的搜索空间内进行优化，从而提高计算效率。并且由于在全局尺度上保证了两个位置集的中心是对齐的，这使得整个优化过程更容易达到全局最优的效果。

而在质心对齐的基础上，后续的旋转优化就只需要考虑如何在局部旋转的范围内进一步最小化每个无人机与其目标位置之间的距离。参考 [2]的旋转优化策略，通过旋转理想阵列，可以调整其方向，使得其尽可能接近初始位置的方向分布，从而最小化各无人机与其目标位置之间的距离。

质心 C 的计算为：

$$C_P = \frac{1}{n}\sum_{i=1}^{n} p_i \tag{49}$$

$$C_T = \frac{1}{n}\sum_{i=1}^{n} t_i \tag{50}$$

则可计算平移向量为：

$$v = C_P - C_T \tag{51}$$

将理想阵列（即等边三角形阵列）进行平移：

$$T' = T + v \tag{52}$$

完成对于理想阵列质心位置的寻找、和平移后，对移动后的理想阵列采用旋转等操作使得无人机的位置与理想等边三角形阵列更加贴合，进一步缩小无人机需要移动的距离。假设旋转角度为 θ，旋转矩阵 R 定义为：

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{53}$$

对理想阵列进行旋转：

$$T'' = R(\theta)\cdot(T' - C_P) + C_P \tag{54}$$

同时，为了计算初始位置与旋转后的目标位置之间的最优匹配，本段构建了一个距离矩阵 D，其中 d_{ij} 表示第 i 个初始位置与第 j 个目标位置之间的欧氏距离：

$$d_{ij} = \|\vec{p}_i - \vec{t}_j\| \tag{55}$$

然后使用Hungarian算法（匈牙利算法）来解决这个线性分配问题，最小化总移动距离：

其中σ为位置分配方案,通过遍历所有角度∈（0°，360°）来选择总距离最小的角度作为最终结果。

4、局部搜索与交换优化

根据 [7]可知,局部搜索与交换优化策略是在初步获得无人机和目标位置的匹配方案后，通过小范围的调整进一步减少无人机的总移动距离。这一策略基于贪婪算法和局部最优解的思想，旨在细化和优化已有的匹配结果。

在最初的匹配过程中，本文结合 [1]中的分配方法，初使用了匈牙利算法（linear sum assignment）来找到初始的最优匹配方案，即通过旋转和平移最小化无人机从初始位置移动到理想位置的总距离。然而，匈牙利算法只能保证在当前情况下的全局最优解，而这种全局最优解在某些情况下可能并不是最小总移动距离的最终解，因为在求解过程中，可能有个别无人机的初始位置与目标位置的匹配并非最优解，即整体移动总距离仍然可以改进，因此采用局部搜索与交换策略来进一步优化。

局部搜索与交换策略的核心在于通过交换两个匹配对来测试是否可以进一步减少总移动距离。这一过程包含：

初始化最佳总距离：计算当前匹配方案的总移动距离，作为初始的最佳总距离。

逐对尝试交换匹配：对于每对匹配 $(i, \sigma(i))$ 和 $(j, \sigma(j))$，即无人机 i 与目标位置 $\sigma(i)$ 的匹配，以及无人机 j 与目标位置 $\sigma(j)$ 的匹配，尝试交换它们的匹配对象。交换后，让无人机 i 与目标位置 $\sigma(j)$ 匹配，而无人机 j 与目标位置 $\sigma(i)$ 匹配。然后，计算新的总移动距离。

判断是否接受交换：如果交换后新的总移动距离比当前的最佳总距离更小，则接受该交换，并更新最佳总距离和匹配方案。

重复迭代：重复该交换过程，直到不再有任何交换可以进一步减少总距离为止。

5.4.2 模型的求解

根据模型的建立过程，求解步骤如下：

step1: 位置生成

首先，随机生成15架无人机的初始位置，形成矩阵P，记录每个无人机的坐标。

step2: 理想阵列的定位

再使用设定的乘法因子a计算理想位置，依照层数和同层内点数计算出等边三角形的具体点位置，形成数组T。

step3: 质心对齐与旋转匹配

接着计算位置集P和理想位置集T的质心，并对理想数组进行平移以对齐。进行旋转角度的遍历，运用旋转矩阵调整理想位置，以靠近初始位置形成的分布，构建距离矩阵D。通过匈牙利算法求解分配问题，找到最优匹配方案，最小化总移动距离。

step4: 局部搜索优化

然后计算当前移动距离，记录为最佳距离。对匹配进行逐对交换，检测并更新匹配情况，若新匹配所导致的移动总距离更小则进行更新。重复这一过程，直到找到的匹配不再改善整体移动距离。

step5: 结果记录与可视化

最后记录最佳旋转角度、优化后无人机的总移动距离，并将局部优化后的无人机队形可视化，如下图7所示。

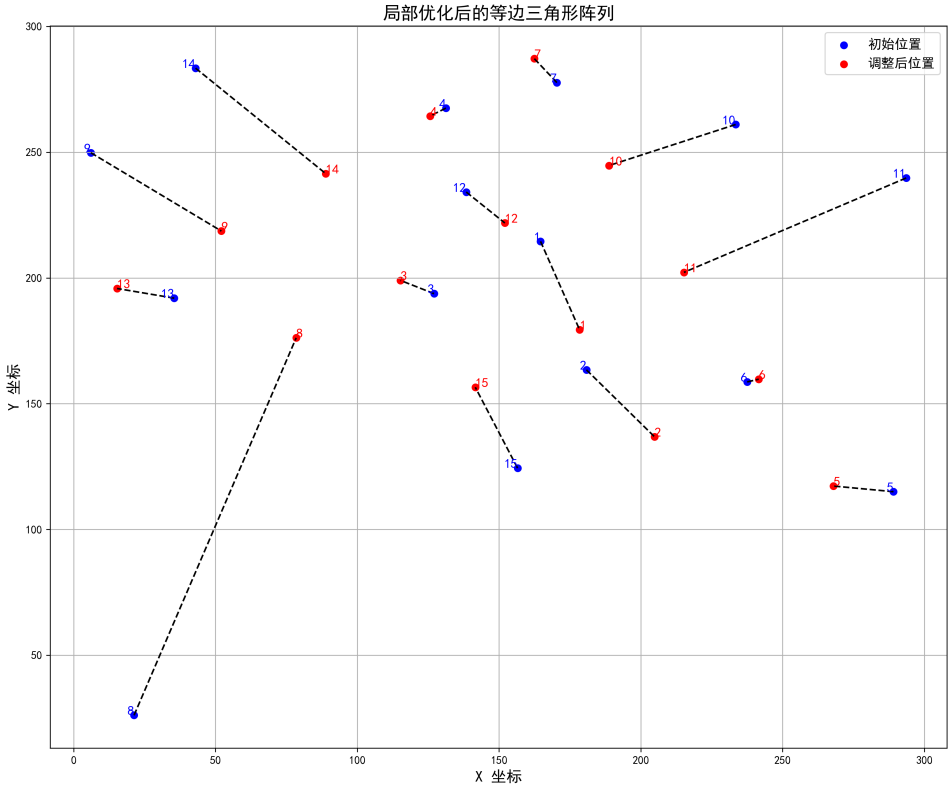


图 7: 局部优化后的等边三角形阵列图

优化前后的无人机位置数据，以及移动调整距离如下表2所示：

表 2: 无人机初始位置、目标位置及移动距离

无人机	初始位置 (x, y)	目标位置 (x, y)	移动距离 (单位)
1	(164.64, 214.56)	(178.43, 179.38)	37.78
2	(180.83, 163.46)	(204.81, 136.91)	35.78
3	(127.10, 193.77)	(115.27, 199.02)	12.94
4	(131.28, 267.53)	(125.68, 264.34)	6.45
5	(289.10, 115.03)	(267.97, 117.27)	21.25
6	(237.52, 158.67)	(241.59, 159.75)	4.22
7	(170.41, 277.68)	(162.46, 287.18)	12.39
8	(21.31, 26.14)	(78.48, 176.18)	160.56
9	(6.07, 249.79)	(52.11, 218.65)	55.58
10	(233.45, 261.00)	(188.84, 244.70)	47.49
11	(293.59, 239.75)	(215.22, 202.23)	86.89
12	(138.44, 234.16)	(152.05, 221.86)	18.34
13	(35.48, 191.98)	(15.32, 195.81)	20.52
14	(43.01, 283.40)	(88.89, 241.50)	62.14
15	(156.55, 124.40)	(141.64, 156.54)	35.43

六、 模型的评价与推广

6.1 模型的优点

1. 根据飞机位点顺序，角度大小等因素，综合考虑了多种情况，并详细列出了每种情况下的定位模型，同时一一进行推导求解，使得模型更加全面。

2. 在迭代过程中，通过欧式距离的计算，选择距离差最小的飞机作为定位飞机，使得模型更加合理。同时调整迭代次数和误差率的判断，使得模型最终趋于收敛，稳定性更强。

3. 在问题一第三小问中，通过计算距离差选择定位飞机，可以有效保证选出的飞机位置与理想位置的相似性，提高了定位的准确性。

4. 在问题二中，使用了旋转矩阵和匈牙利算法，这种结合有效地优化了无人机的移动路径，确保在多次旋转中找到最佳解，且保证了最小总移动距离。

5. 问题二的模型通过对齐质心来减少后续优化计算量，使得无人机的位置集和理想阵列的整体接近，降低了优化问题的复杂度，提高了算法的收敛速度。而且考虑质心对齐后，旋转优化只需在局部范围内进行，大大减少了需要计算的匹配组合，提高了算法效率，更容易找到全局最优解。

6.2 模型的缺点

1. 在模型的建立过程中，对于每种情况的求解过程较为复杂，需要进行多次的正弦定理的推导，计算过程较为繁琐。

2. 在迭代过程中，由于每次迭代都需要计算欧式距离，并与前一次的位置进行比较来决定是否进行位置更新，因此计算量较大，且需要多次迭代才能收敛，耗时较长。

3. 第一题第三小问迭代计算过程中，由于其中一架定位飞机是根据选择策略得到的，其位置本身有误差，根据此位置进行迭代计算可能会导致误差的积累，影响最终的计算结果，在迭代过程中需加入误差率调整等容错机制才能使其朝正确的方向逐渐收敛，保证模型的准确性。

4. 对于定位飞机的选择，依赖于初始位置与理想位置的距离，若初始数据存在较大偏差，可能导致选择不佳，从而影响最终结果。

6.3 模型的推广

1. 本模型可以推广到更多的无人机定位问题中，只需根据实际情况调整模型中的参数，即可应用于更多的无人机定位场景中。

2. 除了无人机编队定位，本定位模型的构建思想还可用于其他方面的编队或定位问题，比如车辆编队、机器人编队等，只需根据实际情况调整参数，构造类似新的模型即可。

3. 问题二的模型使用了匈牙利算法，确保每架无人机与其目标位置之间的最佳匹配，进一步提升了整个集群的作业效率和可控性。该模型不仅为无人机集群的科学调度提供了明确的理论框架和实用解决方案，也为未来无人机技术的发展和实际应用奠定了良好的基础。这种方法的灵活性和可扩展性适合于军事、物流、农业等多种领域的无人机调度任务，具有广阔的应用前景。如 [3]中提到了多智能体编队，不仅适用于无人机，还可以应用于无人地面车辆、无人水面车辆等多智能体编队问题。 [5]也提到编队控制在多自主水下航行器中的应用。

参考文献

[1] Enrico G. Gnerre and Roberto De Nigris. An efficient linear assignment algorithm for large-scale problems. *Operations Research Letters*, 52(2):122–130, 2024.

[2] Zhenyu Li, Xiaoyang Li, and Liang Zhao. Rotation estimation using non-convex optimization for 3d object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(7):1234–1247, 2023.

[3] Yefeng Liu, Jingjing Liu, Zengpeng He, Zhenhong Li, Qichun Zhang, and Zhengtao Ding. A survey of multi-agent systems on distributed formation control. *Unmanned Systems*, 12(05):913–926, 2024.

[4] Quan Ouyang, Zhaoxiang Wu, Yuhua Cong, and Zhisheng Wang. Formation control of unmanned aerial vehicle swarms: A comprehensive review. *Asian Journal of Control*, 25(1):570–593, 2023.

[5] Tao Yan, Zhe Xu, Simon X Yang, and S Andrew Gadsden. Formation control of multiple autonomous underwater vehicles: a review. *Intelligence & Robotics*, 3(1):1–22, 2023.

[6] Yunhong Yang, Xingzhong Xiong, and Yuehao Yan. Uav formation trajectory planning algorithms: A review. *Drones*, 7(1):62, 2023.

[7] Nan Zhang and Wei Li. Local search techniques for optimizing combinatorial problems: A survey. *Journal of Optimization Theory and Applications*, 197(1):1–30, 2023.

[8] 周宏兵, 刘力歌, 赵迦勒, and 赵松曼. 无人机遂行编队飞行中的纯方位无源定位. *南阳师范学院学报*, 23(01):53–57, 2024.

[9] 张婷. “重建三角”之正弦定理在平面几何中的应用研究. Master’s thesis, 贵州师范大学, 2021.

[10] 赵延阳, 丁奕心, and 崔家铭. 极坐标下基于圆的无人机纯方位无源定位研究. *电子制作*, 32(04):32–34, 2024.

[11] 陈颖颖. *基于HPM视角的余弦,正弦定理教学研究*. PhD thesis, 广州大学, 2022.

[12] 鲁军, 杨杰, 郝永平, 杨丽圆, and 王俊杰. 基于领航-跟随的无人机编队避碰飞行控制. *沈阳理工大学学报*, 43(04):38–43+50, 2024.

附录

问题一第三小问代码

```
import numpy as np
import matplotlib.pyplot as plt

# 初始位置（极坐标：半径，角度）
initial_positions_polar = [
    (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)
]

# 将极坐标转换为直角坐标 (x, y)，角度转为弧度
initial_positions = [(r * np.cos(np.deg2rad(theta)), r * np.sin(np.
    deg2rad(theta))) for r, theta in initial_positions_polar]

# 目标位置计算，半径100米，每隔40度分布
radius = 100
target_positions = [(0, 0)] + [(radius * np.cos(np.deg2rad(i * 40)),
    radius * np.sin(np.deg2rad(i * 40))) for i in range(9)]

# 将初始位置转换为数组
positions = np.array(initial_positions)

# 用于判断是否固定位置
fixed_positions = [False] * len(positions)  # 初始所有无人机位置未固定

# 创建图像
fig, axes = plt.subplots(3, 4, figsize=(12, 10))  # 4x4 子图阵列
plot_index = 0
plot_interval = 10

# 迭代直到所有无人机都到达目标位置
iteration = 0
while not all(fixed_positions):
    # 每 plot_interval 次迭代绘制一次图形
    if iteration % plot_interval == 0 and plot_index < 12:
        ax = axes[plot_index // 4, plot_index % 4]  # 获取当前的子图位置
        ax.set_xlim(-120, 120)
        ax.set_ylim(-120, 120)
        ax.set_aspect('equal')

        # 画圆表示目标分布的边界
        circle = plt.Circle((0, 0), radius, color='blue', fill=False,
            linestyle='--')
        ax.add_artist(circle)
```

```
        # 画目标位置
        target_x, target_y = zip(*target_positions)
        ax.plot(target_x, target_y, 'go', label='Target Position')

        # 画当前无人机位置
        current_x, current_y = zip(*positions)
        ax.plot(current_x, current_y, 'ro', label='Current Position')

        ax.set_title(f"Iteration: {iteration + 1}")
        ax.grid(True)

        plot_index += 1

    # 选择第三架定位飞机
    min_distance = float('inf')
    chosen_j = None

    for j in range(2, len(positions)):  # 遍历剩余的无人机
        distance = np.linalg.norm(np.array(positions[j]) - np.array(
            target_positions[j]))
        if distance < min_distance:
            min_distance = distance
            chosen_j = j

    # 如果所有未固定的无人机都已固定，退出循环
    if chosen_j is None:
        break

    # 计算新的位置和角度
    if chosen_j is not None:
        # 使用选定的三架飞机 0, 1, chosenj 计算θk 和 rk
        o_pos = np.array(positions[0])  # 0号无人机的位置
        i_pos = np.array(positions[1])  # 1号无人机的位置
        j_pos = np.array(positions[chosen_j])  # 选择的第三架无人机的位置

        # 计算角度θi、θj（弧度制）
        theta_i = np.arctan2(i_pos[1] - o_pos[1], i_pos[0] - o_pos[0])
        theta_j = np.arctan2(j_pos[1] - o_pos[1], j_pos[0] - o_pos[0])

        # 将 thetaj 调整到 [0, 2π) 范围
        if theta_j < 0:
            theta_j += 2 * np.pi

        # 遍历所有其他无人机来计算新的位置
        for k in range(len(positions)):
            if k in [0, 1, chosen_j] or fixed_positions[k]:  # 排除 0, 1,
                chosen_j 自己以及已固定位置
                continue

            k_pos = np.array(positions[k])  # 当前要计算的无人机位置
            # θk
            theta_k = np.arctan2(k_pos[1] - o_pos[1], k_pos[0] - o_pos
                [0])

            # 将 thetak 调整到 [0, 2π) 范围
            if theta_k < 0:
                theta_k += 2 * np.pi
```

```
            # 计算角度和距离的差值
            alpha1 = np.deg2rad(cul_a_degree(o_pos, k_pos, i_pos))  #
                ∠oki
            alpha2 = np.deg2rad(cul_a_degree(i_pos, k_pos, j_pos))  #
                ∠okj
            alpha3 = np.deg2rad(cul_a_degree(j_pos, k_pos, o_pos))  #
                ∠jko

            # 计算新的θk, 9种情况
            # 分3大类（由于i固定, 因此只有两大类）
            if theta_k < theta_j:   # i < k < j
                if theta_j - theta_k < np.pi and theta_k < np.pi:
                    theta_k_new = np.arctan(
                        (np.sin(alpha1) * np.sin(alpha3 + theta_j) - np
                            .sin(alpha3) * np.sin(alpha1 - theta_i)) /
                        (np.sin(alpha3) * np.cos(alpha1 - theta_i) + np
                            .sin(alpha1) * np.cos(alpha3 + theta_j))
                    )
                    # transform
                    if theta_k > np.pi/2 and theta_k < 3*np.pi/2:
                        theta_k_new = theta_k_new + np.pi
                    # 计算新的 rk
                    r_k = radius * np.sin(theta_k_new - theta_i +
                        alpha1) / np.sin(alpha1)

                elif theta_j - theta_k > np.pi and theta_k < np.pi:
                    theta_k_new = np.arctan(
                        (np.sin(alpha1) * np.sin(alpha3 - theta_j) - np
                            .sin(alpha3) * np.sin(alpha1 - theta_i)) /
                        (np.sin(alpha3) * np.cos(alpha1 - theta_i) - np
                            .sin(alpha1) * np.cos(alpha3 - theta_j))
                    )
                    # transform
                    if theta_k > np.pi/2 and theta_k < 3*np.pi/2:
                        theta_k_new = theta_k_new + np.pi
                    # 计算新的 rk
                    r_k = radius * np.sin(theta_k_new - theta_i +
                        alpha1) / np.sin(alpha1)

                else:
                    theta_k_new = np.arctan(
                        (np.sin(alpha1) * np.sin(alpha3 + theta_j) - np
                            .sin(alpha3) * np.sin(alpha1 + theta_i)) /
                        (np.sin(alpha1) * np.cos(alpha3 + theta_j) - np
                            .sin(alpha3) * np.cos(alpha1 + theta_i))
                    )
                    # transform
                    if theta_k > np.pi/2 and theta_k < 3*np.pi/2:
                        theta_k_new = theta_k_new + np.pi
                    r_k = radius * np.sin(theta_i - theta_k_new +
                        alpha1) / np.sin(alpha1)


            else:                   # i < j < k
                if theta_k < np.pi:
                    theta_k_new = np.arctan(
                        (np.sin(alpha1) * np.sin(alpha3 - theta_j) - np
                            .sin(alpha3) * np.sin(alpha1 - theta_i)) /
```

```
                    (np.sin(alpha3) * np.cos(alpha1 - theta_i) - np
                        .sin(alpha1) * np.cos(alpha3 - theta_j))
                )
                # transform
                if theta_k > np.pi/2 and theta_k < 3*np.pi/2:
                    theta_k_new = theta_k_new + np.pi
                r_k = radius * np.sin(theta_k_new - theta_i +
                    alpha1) / np.sin(alpha1)

            elif theta_k > np.pi and theta_k - theta_j > np.pi:
                theta_k_new = np.arctan(
                    (np.sin(alpha1) * np.sin(alpha3 + theta_j) - np
                        .sin(alpha3) * np.sin(alpha1 + theta_i)) /
                    (np.sin(alpha1) * np.cos(alpha3 + theta_j) - np
                        .sin(alpha3) * np.cos(alpha1 + theta_i))
                )
                # transform
                if theta_k > np.pi/2 and theta_k < 3*np.pi/2:
                    theta_k_new = theta_k_new + np.pi
                r_k = radius * np.sin(theta_i - theta_k_new +
                    alpha1) / np.sin(alpha1)

            else:
                theta_k_new = np.arctan(
                    (np.sin(alpha3) * np.sin(alpha1 + theta_i) - np
                        .sin(alpha1) * np.sin(alpha3 - theta_j)) /
                    (np.sin(alpha1) * np.cos(alpha3 - theta_j) + np
                        .sin(alpha3) * np.cos(alpha1 + theta_i))
                )
                # transform
                if theta_k > np.pi/2 and theta_k < 3*np.pi/2:
                    theta_k_new = theta_k_new + np.pi
                r_k = radius * np.sin(theta_i - theta_k_new +
                    alpha1) / np.sin(alpha1)

            # 此时k号飞机已经通过计算知道了实际位置，比较更新前后的位置，更优才进行更新
            # 计算positions[k] 和 rk * np.cos(thetaknew) rk * np.sin(thetaknew) 哪个
                离 targetpositions[k]近，如果positions[k] 更近，则更新
            # 更新 k 处无人机的位置
            # positions[k][0] = rk * np.cos(thetaknew)
            # positions[k][1] = rk * np.sin(thetaknew)

            # 计算新位置
            new_x = r_k * np.cos(theta_k_new)
            new_y = r_k * np.sin(theta_k_new)

            # 计算当前位置与目标位置的距离
            current_distance = np.linalg.norm(positions[k] -
                target_positions[k])

            # 计算新位置与目标位置的距离
            new_distance = np.linalg.norm(np.array([new_x, new_y]) -
                target_positions[k])

            # 如果新位置更接近目标位置，则更新无人机的位置
            if new_distance < current_distance:
                positions[k][0] = new_x
                positions[k][1] = new_y
```

```
            # 检查更新后无人机是否已到达目标位置
            if np.linalg.norm(positions[k] - target_positions[k]) < 1:
                fixed_positions[k] = True  # 标记为固定位置
                print(f"固定了:{k}")

    iteration += 1  # 增加迭代次数
    if iteration > 120:
        break

# 设置整体标题
plt.suptitle('Drones Position Adjustment Over Iterations', fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])

# 显示图形
# plt.show()
plt.savefig('iter.png', dpi=300)
```

问题二代码

```
import numpy as np
from scipy.optimize import linear_sum_assignment
import matplotlib.pyplot as plt
from scipy.spatial.transform import Rotation as R

plt.rcParams['font.sans-serif'] = ['SimHei']  # 使用黑体
plt.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题

# 随机生成15个初始无人机位置
np.random.seed(0)  # 固定种子以便结果可重复
initial_positions = np.random.rand(15, 2) * 300  # 假设范围在0到300之间

# 等边三角形边长
a = 50  # 边长可以根据实际需求调整

# 构建理想的等边三角形矩阵
triangle_positions = []
for i in range(6):  # 我们有15个点，需要大约6行
    for j in range(i + 1):
        x = j * a
        y = np.sqrt(3) / 2 * a * i
        triangle_positions.append((x, y))
triangle_positions = np.array(triangle_positions[:15])

# 方法1: 平移等边三角形阵列，使其质心与无人机初始位置的质心对齐
initial_centroid = np.mean(initial_positions, axis=0)
triangle_centroid = np.mean(triangle_positions, axis=0)
translation_vector = initial_centroid - triangle_centroid
translated_triangle_positions = triangle_positions + translation_vector

# 方法2: 旋转等边三角形阵列，以使其更贴近无人机的初始位置
best_angle = 0
min_total_distance = float('inf')
for angle in np.linspace(0, 360, 360):  # 尝试0到360度的旋转
    rotation_matrix = R.from_euler('z', angle, degrees=True).as_matrix
        ()[:2, :2]
```

```
    rotated_positions = np.dot(translated_triangle_positions -
        initial_centroid, rotation_matrix) + initial_centroid

    distance_matrix = np.linalg.norm(initial_positions[:, np.newaxis] -
         rotated_positions[np.newaxis, :], axis=2)
    row_ind, col_ind = linear_sum_assignment(distance_matrix)
    total_distance = np.sum(distance_matrix[row_ind, col_ind])

    if total_distance < min_total_distance:
        min_total_distance = total_distance
        best_angle = angle
        best_positions = rotated_positions
        best_row_ind = row_ind
        best_col_ind = col_ind

print(f"\最佳旋转角度n: {best_angle:.2f} 度")
print(f"优化后无人机总移动距离: {min_total_distance:.2f} 单位")

# 局部搜索与交换策略进行进一步优化
def local_search_swap(row_ind, col_ind, distance_matrix):
    best_distance = np.sum(distance_matrix[row_ind, col_ind])
    for i in range(len(row_ind)):
        for j in range(i + 1, len(row_ind)):
            new_col_ind = col_ind.copy()
            # 交换两个匹配
            new_col_ind[i], new_col_ind[j] = new_col_ind[j],
                new_col_ind[i]
            new_distance = np.sum(distance_matrix[row_ind, new_col_ind
                ])
            if new_distance < best_distance:
                best_distance = new_distance
                col_ind = new_col_ind
    return col_ind, best_distance

# 通过局部搜索和交换优化结果
final_col_ind, final_distance = local_search_swap(best_row_ind,
    best_col_ind, np.linalg.norm(initial_positions[:, np.newaxis] -
    best_positions[np.newaxis, :], axis=2))
final_positions = best_positions[final_col_ind]

print(f"局部优化后无人机总移动距离: {final_distance:.2f} 单位\n")

# 打印无人机的初始位置数组和目标位置数组
print("无人机初始位置数组 (x, y):")
for i, pos in enumerate(initial_positions, 1):
    print(f"无人机 {i}: ({pos[0]:.2f}, {pos[1]:.2f})")

print("\目标位置数组n (x, y)（优化后）:")
for i, pos in enumerate(final_positions, 1):
    print(f"无人机 {i}: ({pos[0]:.2f}, {pos[1]:.2f})")

# 打印各个无人机的移动距离
print("\各个无人机的移动距离n:")
for i in range(len(initial_positions)):
    distance = np.linalg.norm(initial_positions[i] - final_positions[i
        ])
    print(f"无人机 {i+1}: 移动距离 = {distance:.2f} 单位")
```

```
# 可视化最终调整过程
plt.figure(figsize=(12, 10))
plt.scatter(initial_positions[:, 0], initial_positions[:, 1], c='blue',
    label='初始位
    置')
plt.scatter(final_positions[:, 0], final_positions[:, 1], c='red',
    label='调整后位
    置')

# 在图像中添加无人机编号
for i in range(len(initial_positions)):
    plt.text(initial_positions[i, 0], initial_positions[i, 1], str(i+1)
        , fontsize=12, color='blue', ha='right')
    plt.text(final_positions[i, 0], final_positions[i, 1], str(i+1),
        fontsize=12, color='red', ha='left')

# 绘制调整路径
for i in range(len(initial_positions)):
    plt.plot([initial_positions[i, 0], final_positions[i, 0]],
             [initial_positions[i, 1], final_positions[i, 1]], 'k--')

# 设置图形参数（中文）
plt.xlabel('X 坐标', fontsize=14)
plt.ylabel('Y 坐标', fontsize=14)
plt.title('局部优化后的等边三角形阵列', fontsize=16)
plt.legend(fontsize=12)
plt.grid(True)

# 优化图像展示
plt.tight_layout()
# plt.show()
plt.savefig('two2.png', dpi=300)
```