# 无人机遂行编队飞行中的纯方位无源定位

## 摘要

随着无人机的广泛应用，定位技术成为关键问题。相比成本高、技术复杂的有源定位，无人机无源定位通过接收目标无人机的电磁信号来确定无人机位置，即通过提取出方向信息进行定位，来调整无人机的位置。对于无人机纯方位无源定位问题，本文提出多阶段定位与优化模型。

**针对问题一**，建立了基于余弦定理的小偏差定位模型：以圆心 FY00 为原点，FY01 为 x 轴基准，将接收无人机位置参数化为理想位置叠加微小偏差($\Delta x$, $\Delta y$)；利用 FY00、FY01 及一架编号已知无人机发射信号形成的三个方向夹角，通过余弦定理构建非线性方程组并通过线性化近似求解偏差量，实现被动接收机的高精度定位。

**针对问题二**，本文的思路为采取几何定位模型，基于圆周角定理，证明每个方向角测量值唯一对应一个外接圆轨迹。当仅知 FY00、FY01 编号时，引入一架未知编号发射无人机，通过求解三圆方程组（圆心由弦的中垂线与半径公式确定）实现被动接收机有效定位，严格论证除 FY00、FY01 外仅需 **1 架**额外发射机即可保证位置唯一性。

**针对问题三**，鉴于题目没有对圆周半径做硬性要求，我们选择优化算法：粒子群优化算法 (PSO) ，通过逐轮迭代的方式，将 9 架初始位置不精确的无人机，逐步引导至一个以中心无人机（FY00） 为圆心、且成员间均匀分布的圆形编队。粒子群算法兼顾了全局最优和个体最优，且收敛速度快，适合无人机实时调整的应用场景。

**针对问题四**，题目给出了锥形的编队场景，具体任务同问题三一样是调整无人机至正确的位置。因此，我们仍然选择粒子群优化算法，通过定义一个几何结构的适应度函数，来量化任一队形与其理想 锥形构型的差距，不断迭代来寻找全局最优。我们也自己编缀了验证数据，证明了本方案的可行性。

**关键词：** 纯方位无源定位，圆周角定理，粒子群优化，适应度函数，锥形坐标系变换

# 一、 问题重述

## 1.1 问题背景

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（如图 1 所示）。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 $\alpha 1$，$\alpha 2$ 和 $\alpha 3$。

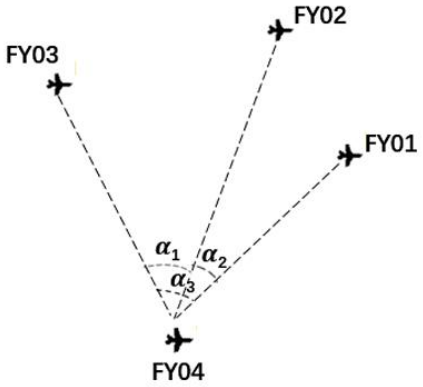



**图 1 无人机接收到的方向信息示意图**

## 1.2 问题重述

基于上述背景，要求建立数学模型解决以下问题：

问题的无人机阵型分为两类:

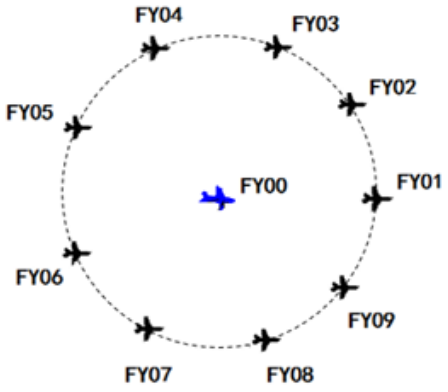



图 1-1 圆形无人机编队示意图

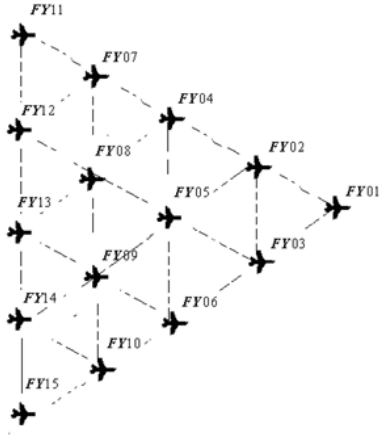



图 1-2 锥形无人机编队示意图

(1)圆形编队：由 10 架无人机构成：编号为 FY00 的无人机位于圆心，其余 9 架无人机在理想状态下均匀分布在某圆周上，全保持在同一个高度上飞行。（见图 1-1）。

(2)锥形编队：由 15 架无人机构成，任意三个无人机均组成等边三角形，即任意相邻两架无人机之间的间距相等. （见图 1-2）。

问题一： 无人机采用圆形编队，位于圆心的无人机（FY00）和编队中另 2 架无人机发射信号，发射信号的无人机位置无偏差且编号已知时，被动接收信号的无人机的位置略有偏差，建立被动接收信号无人机的定位模型。

问题二：无人机采用圆形编队， FY00 和 FY01 以及编队中若干编号未知的无人机发射信号，假设发射信号的无人机位置无偏差，接收信号的无人机位置略有偏差，除 FY00 和 FY01 外，还需要几架无人机发射信号，才能实现被动接收信号无人机的有效定位？

问题三：无人机采用圆形编队，圆周半径为 100 m。 当初始时刻无人机的位置略有偏差时，仅根据接收到的方向信息来调整无人机的位置，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，使得 9 架无人机最终均匀分布在某个圆周上。

问题四：无人机采用锥形编队，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 对问题一的分析

在发射信号的无人机位置无偏差，且编号已知的条件下，要求建立被动接收信号无人机的定位模型。

根据题意，采用纯方位无源定位的方法调整无人机的位置，因此方向信息即夹角信息相当重要，自然地，我们需要找到夹角信息的适当表示方法来完成无人机的纯方位无源定位。

具体而言，以 FY00 为坐标原点，FY00-FY01 方向为 x 轴正方向，将圆周上无人机的理想位置表示为三角函数形式。接收无人机的实际位置在其理想位置基础上添加小偏差$(\Delta x, \Delta y)$。通过余弦定理导出三个夹角的余弦表达式，形成关于 $\Delta x$ 和 $\Delta y$ 的方程组。由于偏差微小（$\Delta x^2$和 $\Delta y^2$可忽略），方程组可线性化近似求解，从而确定接收无人机的精确位置。

## 2.2 对问题二的分析

相比于上一问本题的难点在于发射信号的无人机编号未知，但编队结构固定,呈圆形分布。针对无人机纯方位无源定位问题，本文基于圆周角定理的几何定位原理，建立了多圆交点定位模型。通过证明每个方向角测量值唯一对应一个外接圆轨迹，提出仅需三架发射无人机（含 FY00、FY01 及另一未知编号无人机）即可实现被动接收机的有效定位。通过求解三个外接圆方程组的实数交点确定无人机位置。

## 2.3 对问题三的分析

本小题设定了一个圆形编队场景：1 架中心无人机（FY00）位于编队圆心，其余 9 架无人机（FY01 FY09）应精确且均匀地分布在以 FY00 为中心的同一圆周上。然而在实际飞行过程中，无人机群不可避免地会存在初始位置偏差。因此，我们的任务是设计出一个调整方案，使得所有无人机能够仅依赖自身获取的位置信息，自主调整到一个全局最优且均匀的理想圆形编队。

由于编队中所有无人机的位置是高度耦合的，对某一架无人机进行局部调整，很可能引发连锁反应，破坏其他无人机的位置，进而导致整个队形的振荡、恶化。因此，我们要寻求一种能够同时优化所有 9 架无 人机位置的全局优化方法，在 18 维向量空间（9 架无人机，每架 2 个坐标）中，找到能使整体队形最优的全局解。

## 2.4 对问题四的分析

与问题 1 第三小问的圆形编队相比，锥形编队的几何结构更为复杂，内部连接关系多样，无人机之间的位置耦合性更强，因此我们通过线性变换建立非直角坐标系（基向量夹角 60°），解决了等边三角形约束，推导广义余弦公式适配等边三角形、锥形结构，统一了问题框架。但我们遇到的挑战是如何从一个混乱的初始布局出发，为全部 15 架无人机同时找到一组能够完美构成目标锥形的坐标。我们考虑过 A*算法，但是简单地让每架无人机飞向最近的理想点会导致分配冲突。因此，要寻找一个全局最优位置，我们采用粒子群迭代优化算法。

# 三、模型假设

## 3.1 假设无人机之间互不干扰

由于无人机接收信号为被动接收，且无人机之间尽可能保持电磁静默，少向外发射电磁波信号，故无人机之间接收到的信息互不干扰，例如无人机 i 不会受到无人机 j 所接收到信息的干扰。

### 3.2 假设无人机在圆形编队上的角度偏差不超过±0.5°

从无人机编队实际应用和技术可行性来看，角度偏差假设通常会基于传感器精度、控制算法能力等确定。一般工程实践里，为保证编队稳定性与任务精度，**±0.5°** 这类小偏差更贴合实际。

### 3.3 假设每架无人机均知道自己的编号

假定发射信号的无人机在空间中所处位置精准无偏移，若其用于区分识别的编号是已知、明确的，则便于后续每架无人机做出调整。

### 3.4 假设无人机每次调整的时间忽略不计

结合实际情况，我们近似认为无人机每次调整的时间可忽略不计

## 四、符号说明

| 符号 | 说明 |
|---|---|
| $\boldsymbol{\alpha_1}$ | 接收无人机 P 为顶点，由前两架发射机形成的夹角 |
| $\boldsymbol{\alpha_2}$ | 接收无人机 P 为顶点，由后两架发射机形成的夹角 |
| $\boldsymbol{\alpha_3}$ | 接收无人机 P 为顶点，由第一、第三架发射机形成的夹角 |
| $\boldsymbol{pbest_i}$ | 第 i 个粒子从开始到现在所经历过的最低适应度值的位置 |
| $\boldsymbol{gbest}$ | 是整个粒子群从开始到现在所找到的全局最佳位置 |
| $\boldsymbol{c_1}$ | 认知系数 |
| $\boldsymbol{c_2}$ | 社会系数 |
| **w** | 惯性权重 |

## 五、模型的建立与求解

### 5.1 问题一：被动接收信号无人机的位置调整模型

根据题意将此问题分为三个步骤，首先确定出平面直角坐标系，表示出无人机的位置坐标，接下来通过向量的形式将接收无人机位置参数化，结合余弦定理，求解出小偏差 Δx 和 Δy，进而建立被动接收信号无人机的定位调整模型。

## Step 1 平面直角坐标系的确定

在已知位置的三架无人机位置无偏差且无人机知道自己的编号的条件下，根据无人机的相对位置关系，除 FY00 和 FY01 发射信号外，不妨设另一架发射信号的无人机为 FY0（i+1）。以 FY00 为圆心，$\overrightarrow{FY00FY01}$为 x 轴正方向的单位向量，建立平面直角坐标系 xoy，如图 2 所示，

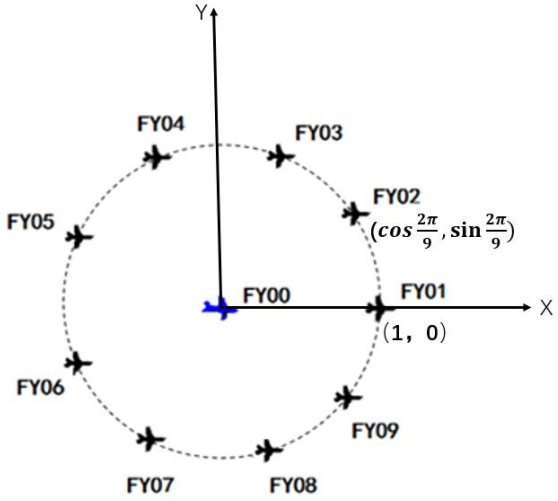



**图 2　平面直角坐标系下无人机的位置坐标**

$$FY00 = (0,0) \quad FY01 = (1,0)$$

$$FY0(\mathrm{i}+1) = (\cos\frac{2i\pi}{9}, sin\frac{2i\pi}{9})(\mathrm{i} = 1,2,\dots,9)$$

设需要确定位置的无人机为 FY0(k+1)，由于其位置“略有偏差”，故其坐标可设为

$$FY0(k+1) = (\cos\frac{2k\pi}{9} + \Delta x, sin\frac{2k\pi}{9} + \Delta y)$$

例如：FY02 对应$\theta_2$=40°$=\frac{2}{9}\pi$，位置为$\left(\cos\frac{2\pi}{9},\ sin\frac{2\pi}{9}\right)$

## Step 2 接收无人机位置参数化

根据上述给定的点坐标，可以得到若干向量的坐标表示，方向从接收无人机指向发射无人机的向量。

$$\overrightarrow{FY0(k+1)FY00} = \left\{-\cos\frac{2k\pi}{9} - \Delta x, -sin\frac{2k\pi}{9} - \Delta y\right\}$$

$$\overrightarrow{FY0(k+1)FY01} = \left\{1 - \cos\frac{2k\pi}{9} - \Delta x, -sin\frac{2k\pi}{9} - \Delta y\right\}$$

$$\overrightarrow{FY0(k+1)FY0(\iota+1)} = \left\{\cos\frac{2i\pi}{9} - \cos\frac{2k\pi}{9} - \Delta x, sin\frac{2i\pi}{9} - sin\frac{2k\pi}{9} - \Delta y\right\}$$

分别设三个向量为$\boldsymbol{\alpha}$，$\boldsymbol{\beta}$，$\boldsymbol{\gamma}$，根据题意可得，无人机 FY0(k+1)可以接收到$\boldsymbol{\alpha}$，$\boldsymbol{\beta}$，$\boldsymbol{\gamma}$两两位置之间夹角的信息。

## Step 3 基于余弦定理建立方程组

由于无人机测量了夹角，故这些余弦值已知，则根据欧氏空间中两向量夹角的定义$\cos\theta = \frac{\vec{\mu}\vec{v}}{|\vec{\mu}||\vec{v}|}$
代入向量得三个方程：

$$cos<\boldsymbol{\alpha}，\boldsymbol{\beta}>$$

$$=\frac{\left(-cos\frac{2k\pi}{9}-\Delta x\right)\left(1-cos\frac{2k\pi}{9}-\Delta x\right)+(-sin\frac{2k\pi}{9}-\Delta y)(-sin\frac{2k\pi}{9}-\Delta y)}{\sqrt{\left(-cos\frac{2k\pi}{9}-\Delta x\right)^2+(-sin\frac{2k\pi}{9}-\Delta y)^2}\sqrt{\left(1-cos\frac{2k\pi}{9}-\Delta x\right)^2+(-sin\frac{2k\pi}{9}-\Delta y)^2}}$$

$$\cos<\boldsymbol{\beta}，\boldsymbol{\gamma}>$$

$$=\frac{\left(1-\cos\frac{2k\pi}{9}-\Delta x\right)\left(\cos\frac{2i\pi}{9}-\cos\frac{2k\pi}{9}-\Delta x\right)+(-sin\frac{2k\pi}{9}-\Delta y)(sin\frac{2i\pi}{9}-\cos\frac{2k\pi}{9}-\Delta y)}{\sqrt{(1-\cos\frac{2k\pi}{9}-\Delta x)^2+(sin\frac{2k\pi}{9}+\Delta y)^2}\sqrt{(\cos\frac{2i\pi}{9}-\cos\frac{2k\pi}{9}-\Delta x)^2+(sin\frac{2i\pi}{9}-\cos\frac{2k\pi}{9}-\Delta y)^2}}$$

$$\cos<\boldsymbol{\alpha}，\boldsymbol{\gamma}>$$

$$=\frac{\left(-\cos\frac{2k\pi}{9}-\Delta x\right)\left(\cos\frac{2i\pi}{9}-\cos\frac{2k\pi}{9}-\Delta x\right)+(-sin\frac{2k\pi}{9}-\Delta y)(sin\frac{2i\pi}{9}-\cos\frac{2k\pi}{9}-\Delta y)}{\sqrt{(\cos\frac{2k\pi}{9}-\Delta x)^2+(-sin\frac{2k\pi}{9}-\Delta y)^2}\sqrt{(\cos\frac{2i\pi}{9}-\cos\frac{2k\pi}{9}-\Delta x)^2+(sin\frac{2i\pi}{9}-\cos\frac{2k\pi}{9}-\Delta y)^2}}$$

## Step 4 简化与求解方程组

由于 Δx, Δy 极小“略有偏差”，忽略二阶小量$\Delta x^2, \Delta y^2$, 分母中的模长近似为理想距离,上述三式为关于$\Delta x$，$\Delta y$的二元方程组，解得$\Delta x$，$\Delta y$，求出其精确值，从而可以使接收信号无人机位置确定下来。

## 5.2.问题二：发出信号无人机数目的确定

要确定在 FY00、FY01 固定发射信号时，额外所需的最少发射无人机数量以实现定位。我们通过几何定位，根据圆周角定理的推论可得，同弧所对的圆周角相等，每个夹角测量值对应一个圆弧轨迹，被动接收机所测方向角$\alpha_i$对应其位于以两发射机为弦、圆周角为$\alpha_i$的外接圆优（劣）弧上, 多组轨迹的交点

即为被动接收机位置，即探究最少需添加几组轨迹（即额外几架发射机）可使交点唯一。

### 5.2.1 定理

定理：由圆周角定理的推论可得同弧所对应的圆周角相等，即$\alpha = \beta$。证明见附录一

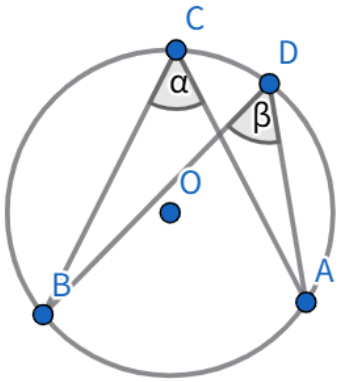



**图 3　同弧所对应的圆周角**

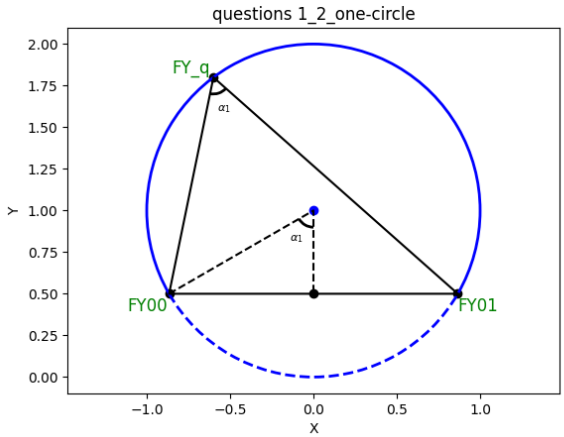
### 5.2.2 未知编号无人机数目的确定

**Step 1 位置坐标的确立**

建立如问题一的笛卡尔坐标系，设除编号为 FY00、FY01 的无人机发射信号外，另一发射信号的无人机编号为 FY0(i+1)，需要定位的无人机编号为 FY0（k+1）

**Step 2 基础圆定义（对于每对发射无人机）**

由公理可得，同弧所对应的圆周角相等，顶角为$\alpha_i$的点恰在上述圆的优弧或劣弧$\widehat{AB}$上，若$\alpha_1$，$\alpha_2$，$\alpha_3$所形成的圆弧轨迹相交于一点，那么即可确定FY0(k+1)的位置



$\odot O_1$：由 FY00-FY01 的夹角$\alpha_1$确定，设 FY00 与 FY01 之间的距离$d_1=1$，由该两点与待定位的无人机所构成的三角形的外接圆半径为$R_1$，由正弦定理可得$R_1=\frac{1}{2\sin\alpha_1}$，圆心在 FY00-FY01 的垂直平分线上（横坐标$x_{O_1}$=$\frac{1}{2}$）

$\odot O_2$，$\odot O_3$：由 FY00 与 FY0(i+1)的夹角$\alpha_2$、FY01 与 FY0(i+1)之间的夹角$\alpha_3$确定，半径：

$$R_2=\frac{1}{2\sin\alpha_2} \qquad R_3=\frac{d_3}{2\sin\alpha_2}$$

**Step 3 讨论圆弧的位置关系**

首先确定三个圆的圆心位置，圆心$O_1$必在线段 FY00FY01 的垂直平分线上，易得其横坐标为$\frac{1}{2}$，故设其坐标为$(\frac{1}{2},y)$，可得到方程

$$(\frac{1}{2})^2+y^2={R_1}^2$$

$\Rightarrow$ 解得：$y=\pm\sqrt{{R_1}^2-\frac{1}{4}}$

根据条件接收信号的点在圆上舍去一个不符合题意的结果

同理，圆心$O_2$在线段 FY00FY0(i+1)的垂直平分线上

$$\begin{cases} y-\dfrac{sin\frac{2i\pi}{9}}{2}=-\dfrac{1}{\tan\frac{2i\pi}{9}}(x-\dfrac{\cos\frac{2i\pi}{9}}{2}) \\ x^2+y^2={R_2}^2 \end{cases}$$

解得$O_2$的坐标为（$a_2,b_2$）

同理，圆心$O_3$在线段 FY01FY0(i+1)的垂直平分线上

$$\begin{cases} y-\dfrac{sin\frac{2i\pi}{9}}{2}=-\dfrac{\cos\frac{2i\pi}{9}-1}{sin\frac{2i\pi}{9}}(x-\dfrac{\cos\frac{2i\pi}{9}+1}{2}) \\ (x-1)^2+y^2={R_3}^2 \end{cases}$$

解得$O_3$的坐标为$(a_3,b_3)$

得到三个圆圆心位置及半径后进一步讨论其位置关系，设

$$\odot O_1：(x-a_1)^2+(y-b_1)^2={R_1}^2$$

$$\odot O_2：(x-a_2)^2+(y-b_2)^2={R_2}^2$$

$$\odot O_3：(x-a_3)^2+(y-b_3)^2={R_3}^2$$

解方程组

$$\begin{cases}(x-a_1)^2+(y-b_1)^2={R_1}^2\\(x-a_2)^2+(y-b_2)^2={R_2}^2\end{cases}$$

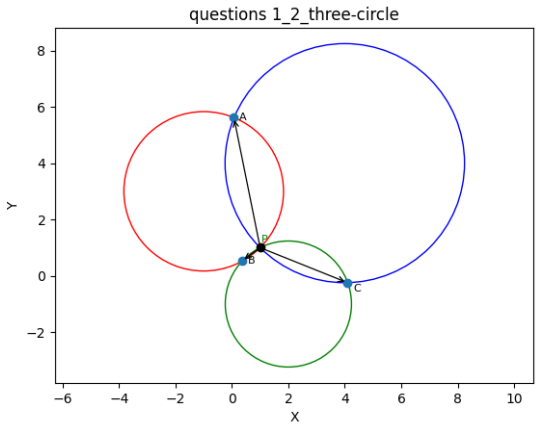



**图 5　三组轨迹所确定的被动接收机的位置**

最终结果如图 5 所示

**Step 4　唯一性证明与最少发射机数**

- 充分性：三圆交于一点（被动接收机真实位置），其余交点因测量角一致性被排除。
- 必要性：若仅有两架发射机（如 FY00、FY01），则 P*P* 位于$\odot O_1$的整个圆弧上，位置不唯一。

结论：除 FY00、FY01 外，仅需 **1 架**额外发射无人机（共 3 架发射机）即可实现有效定位。

## 5.3 问题三：无人机自主定位调整方案的确定

### 5.3.1 指标的确立

我们设定了半径一致性和角度均匀性两个量化指标，用于衡量无人机位置的优劣：

- **半径一致性**：所有 9 架外围无人机到编队中心 FY00 的欧氏距离应相等。
- **角度均匀性**：从编队中心看去，任意两架相邻无人机之间的夹角应严格等于理论值 $360° / 9 = 40°$。

基于这两个指标，我们构建了一个**适应度函数**，将一个编队构型映射为一个实数，其值越小，代表编队质量越高，越接近理想状态，和深度学习里的损失函数概念相似。PSO 算法里的一个粒子，也就是一个潜在的解，是由 9 架无人机的$9 \times 2 = 18$个坐标值构成的向量定义的，P = $[x_1, y_1, x_2 y_2, \ldots, x_9, y_9]$。

### 5.3.2 适应度函数

- **半径偏差成本**：所有无人机半径$r_1$与平均半径$\bar{r}$的方差之和来度量，之所以使用方差而不是标准差,是因为方差能更严厉地惩罚极端值，更为强力地引导无人机向平均半径收敛。

$$Cost_R = \sum_{i=1}^{9} (r_i - \bar{r})^2$$

其中，$r_i = \sqrt{{x_i}^2 + {y_i}^2}$，$\bar{r}$为 9 架无人机半径的算术平均值。

- **角度间隔成本**：所有无人机之间的实际间隔角度$\Delta\theta_i$与理想角度 40°的方差之和。

$$Cost_{angle} = \sum_{i=1}^{8} (\Delta\theta_i - 40°)^2 + (\Delta\theta_{9,1} - 40°)^2$$

其中，$\Delta\theta_i$是对所有无人机按极坐标角度排序后，第 i 架与第 i+1 架无人机之间的角度差。$\Delta\theta_{9,1}$是第 9 架与第 1 架之间的角度差，这是为了保证圆周的闭环。

- **适应度**：将上述两个成本进行加权求和。我们设置权重$\omega_1$和$\omega_2$均为 1.0，表示同等重视半径的一致 性和角度的均匀性。若某特定任务对其中一项有更高要求，则可调整该权重。

$$Fitness = \omega_1 \cdot Cost_R + \omega_2 Cost_{angle}$$

### 5.3.3 粒子群优化算法（PSO）[1]

我们选择的粒子群优化算法是一种有效且计算成本较低的全局优化算法。它通过模拟鸟群觅食，让每个粒子（潜在解）同时借鉴自身的“历史经验”和群体的“集体智慧”，从而在复杂的解空间中高效地寻找全局最优解。在粒子群算法的每一次迭代中，每个粒子都要更新自己的速度和位置。

**更新公式：**

- **速度更新公式 (Velocity Update):**

$$\boldsymbol{v_i(t+1) = \omega \cdot v_i(t) + c_1 \cdot r_1\left(pbest_i - x_i(t)\right) + c_2 \cdot r_2\left(gbest_i - x_i(t)\right)}$$

- **位置更新公式 (Position Update):**

$$\boldsymbol{x_i(t+1) = x_i(t) + v_i(t+1)}$$

速度更新是 PSO 的精髓，它决定了粒子下一步“飞”是往哪个方向，以及“飞”多快。PSO 中的惯性、认知、社会分别对应速度更新公式中的 $w \cdot v_i(t)$ 、$c_1 \cdot r_1 \cdot (pbest_i - x_i(t))$、$c_2 \cdot r_2 \cdot (gbest - x_i(t))$

**其中，**

- **惯性部分$\boldsymbol{w \cdot v_i(t)}$**:
  - 含义: 代表粒子保持其当前运动状态的“惯性”或“记忆”。
  - $v_i(t)$:是粒子在当前时刻 t 的速度。
  - $w$ (惯性权重): 我们设置为 0.5。这个参数用于平衡算法的全局探索和局部开发能力。
    - w 值增大会使粒子倾向于保持当前方向。
    - w 值减小会减弱原始速度的影响，使粒子更容易被“认知”和“社会”部分吸引，从而在已知最优解的附近进行局部搜索。

- **认知部分$\boldsymbol{c_1 \cdot r_1 \cdot (pbest_i - x_i(t))}$**:
  - 含义:代表粒子的“个体经验”，将粒子拉向其自身历史上找过的最佳位置
  - $pbest_i$: 是第 i 个粒子从开始到现在所经历过的最低适应度值的位置。
  - $(pbest_i - x_i(t))$: 是一个向量，指向从粒子当前位置到其自身历史最佳位置的方向。
  - $c_1$（认知系数）：在代码中被设置为 0.8，是一个学习因子，决定粒子飞向自身最佳位置的 “加速度”。

- $r_1$：一个在 (0, 1) 区间内的随机数。它的存在为搜索过程增加了随机性，使得粒子的行为不是完全确定的，有助于跳出局部最优。

- **社会部分$c_2 \cdot r_2 \cdot (gbest - x_i(t))$**:
  - 含义: 代表粒子向群体学习的“集体智慧”,将粒子拉向整个种群目前为止发现的最佳位置。
  - $gbest$: 是整个粒子群从开始到现在所找到的全局最佳位置。
  - $(gbest - x_i(t))$: 是一个向量，指向从粒子当前位置到全局最佳位置的方向。
  - $c_2$(社会系数): 在代码中被设置为 0.9，也是一个学习因子，决定粒子飞向种群全局最佳位置的“加速度”。
  - $r_2$：另一个在 (0, 1) 区间内的随机数。

位置更新用来模拟例子在物理世界中的运动，非常直观，下一时刻的位置 $x_i(t+1)$ 就是当前所在的位置 $x_i(t)$ 加上新速度的位移 $v_i(t+1)$ 。我们设置时间步长 Δt 为 1。

### 5.3.4 算法流程

在预设的搜索空间内，随机生成一群粒子。每个粒子被赋予一个随机的位置（一套完整的无人机坐标）和速度。然后通过适应度函数，计算每个粒子的当前适应度值。

在每一次迭代中，每个粒子都会更新其速度和位置。其更新方向是三个向量的合成：自身的惯性、飞向自身历史最优位置（pbest，个体经验）的趋势、以及飞向整个种群至今为止发现的全局最优位置（gbest，社会知识）的趋势。

经过足够多的迭代后，整个粒子群将大概率收敛到适应度函数值最低的区域。此时全局最优位置 gbest 所对应的坐标，即为我们寻求的最优编队方案。

### 5.3.5 参数设置

以下是 PSO 算法的超参数设置，其选择是在兼顾收敛速度与求解质量后确定的：

- **粒子数量**（n_particles）: 500。我们设置了较大的种群规模，使之能更好地搜索 18 维空间，降低陷入局部最优解的风险。
- **迭代次数**（n_iterations）: 1000。尽管 PSO 能快速收敛，但我们仍然设置了 1000 轮次的迭代，让它更精确地收敛到全局最优。
- **惯性权重**（w）: 0.5。我们设置为 0.5，相当折中，使粒子能在前期广泛搜索，在后期精细调整。
- **认知系数**（c1）: 0.8。它决定粒子多大程度上相信并返回自己曾经找到过的最优位置。
- **社会系数**（c2）: 0.9。它决定了粒子多大程度上跟随整个群体发现的最优位置。我们把 c2 设置得略大于 c1 有助于加快种群的收敛速度。

**搜索空间**: 基于初始位置，在每个坐标维度上设定+/- 50 米的动态边界。

### 5.3.6 优化效果对比

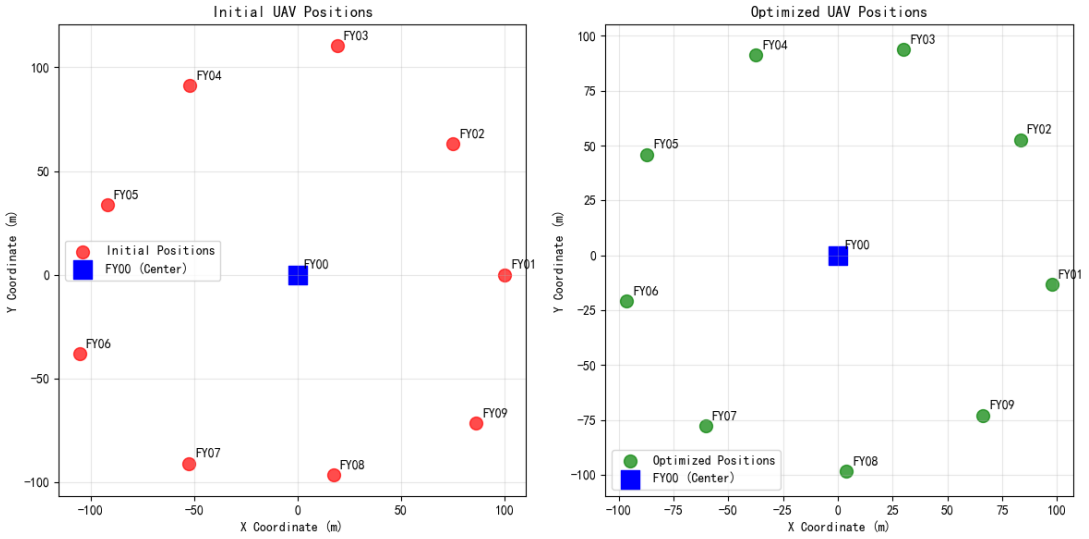



**图 6 初始位置（左）优化后位置（右）的效果对比图**

左图（初始位置）：无人机（红色圆点）较为散乱地分布在中心点（蓝色方块）周围。肉眼可直接看出，各无人机的半径和角度间隔均不一致，队形松散。

右图（优化后位置）：所有无人机（绿色圆点）如规则地排列在一个的圆周上，间距均匀，达到了理想的效果。

表 1 列出了优化前后半径和角度的对比数据

**表 1**

| 编号 | 初始半径（m） | 优化后半径(m) | 初始角度(°) | 优化后角度(°) |
|---|---|---|---|---|
| **FY01** | 100 | 98.559534 | 0 | -7.714802 |
| **FY02** | 98 | 98.560885 | 40.10 | 32.290028 |
| **FY03** | 112 | 98.558919 | 80.21 | 72.293749 |
| **FY04** | 105 | 98.560312 | 119.75 | 112.275301 |
| **FY05** | 98 | 98.563620 | 152.249220 | 152.249220 |
| **FY06** | 112 | 98.571438 | 159.86 | -167.771245 |
| **FY07** | 105 | 98.544763 | 240.07 | -127.784741 |
| **FY08** | 98 | 98.574347 | 280.17 | -87.775436 |
| **FY09** | 112 | 98.562958 | 320.28 | -47.737670 |

**表 2**

| 指标 | 初始编队 | 优化后编队 | 提升效果 |
|---|---|---|---|
| **半径方差（m)** | **31.429** | **0.0001** | **显著降低，半径一致** |
| **角度间隔方差（°)** | **0.04** | **0.0004** | **显著降低，均匀间隔** |

表 2 根据数据分析，可以发现优化效果极为显著。初始编队中较大的半径和角度方差，反映了其混乱无序的状态。经过 PSO 迭代优化后，这两项核心指标均降至几乎为零，这不仅仅是数值上的提升，更代表了编队从一个无序状态

到高度有序的、精确的几何结构的质的飞跃，表明算法成功找到了满足我们所有约束条件的完美解。

## 5.4 问题四：锥形编队的具体调整方案

本方案的解决思路是通过数学建模将“完美的锥形”这一概念量化；然后利用 PSO 求解该数学模型，找到最优解。

### 5.4.1 定义坐标系与变换模型[3]

定义基准坐标系：以圆心 FY00 为原点 O，$\vec{\imath}=(1,0)$ $\vec{\jmath}=(0,1)$为$R^2$的基向量，定义线性变换$\boldsymbol{A}$，使得 $\boldsymbol{A}(\overrightarrow{\imath})=\vec{\imath}$

$$\boldsymbol{A}(\overrightarrow{\jmath})=\frac{1}{2}\vec{\imath}+\frac{\sqrt{3}}{2}\vec{\jmath}$$

那么 $$\boldsymbol{A}(\vec{\imath},\vec{\jmath})=(\vec{\imath},\frac{1}{2}\vec{\imath}+\frac{\sqrt{3}}{2}\vec{\jmath})=(\vec{\imath},\vec{\jmath})\begin{pmatrix}1 & \frac{1}{2}\\ 0 & \frac{\sqrt{3}}{2}\end{pmatrix}$$

变换矩阵为： $$\mathrm{A}=\begin{pmatrix}1 & \frac{1}{2}\\ 0 & \frac{\sqrt{3}}{2}\end{pmatrix}$$

该矩阵将原坐标系中的向量映射到新基$(\vec{\imath},\frac{1}{2}\vec{\imath}+\frac{\sqrt{3}}{2}\vec{\jmath})$下，用于描述方向信息与坐标的转换关系。

设$\boldsymbol{\alpha}=x_1\vec{\imath}+y_1\vec{\jmath}$ $\boldsymbol{\beta}=x_2\vec{\imath}+y_2\vec{\jmath}$

则$\boldsymbol{\alpha}$的坐标为$(x_1,y_1)$，$\boldsymbol{\beta}$的坐标为$(x_2,y_2)$

$$\boldsymbol{A\alpha}=\begin{pmatrix}1 & \frac{1}{2}\\ 0 & \frac{\sqrt{3}}{2}\end{pmatrix}\begin{pmatrix}x_1\\ y_1\end{pmatrix}=\begin{pmatrix}x_1+\frac{1}{2}y_1\\ \frac{\sqrt{3}}{2}y_1\end{pmatrix}$$

$$\boldsymbol{A\beta}=\begin{pmatrix}x_2+\frac{1}{2}y_2\\ \frac{\sqrt{3}}{2}y_2\end{pmatrix}$$

这说明在基为$\vec{\imath}$，$\vec{\jmath}$下坐标为(x,y)的向量经过线性变换映射到基为$\vec{\imath}$，$\frac{1}{2}\vec{\imath}+\frac{\sqrt{3}}{2}\vec{\jmath}$下坐标为(x,y)的向量，这个向量在基$\vec{\imath}$，$\vec{\jmath}$下的坐标为(x+$\frac{1}{2}y$, $\frac{\sqrt{3}}{2}y$)

$$\text{那么}\cos<\boldsymbol{\alpha},\boldsymbol{\beta}>=\frac{(x_1+\frac{1}{2}y_1)(x_2+\frac{1}{2}y_2)+\frac{\sqrt{3}}{2}y_1\frac{\sqrt{3}}{2}y_2}{\sqrt{(x_1+\frac{1}{2}y_1)^2+(\frac{\sqrt{3}}{2}y_1)^2}\sqrt{(x_2+\frac{1}{2}y_2)^2+(\frac{\sqrt{3}}{2}y_2)^2}}$$

例如经过线性变换$\boldsymbol{A}$，图 7-1 的中点(1,1)映射到图 7-2 中点(1,1)，而图 7-2 中点(1,1)在图 7-1 中的坐标表示为$(\frac{3}{2},\frac{\sqrt{3}}{2})$

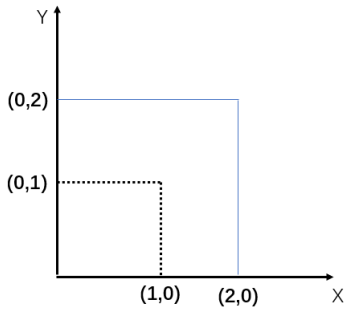



图 7-1　变换前（左）的坐标系

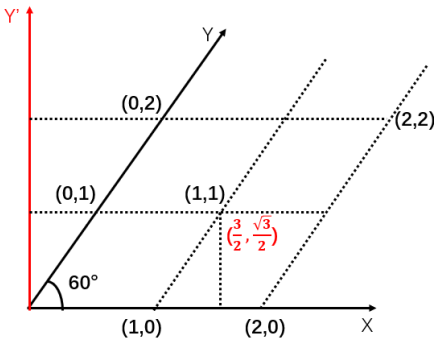



图 7-2　变换后（右）的坐标系

### 5.4.2 建立角度定位模型

在无人机群中建立如图 8 所示的坐标系

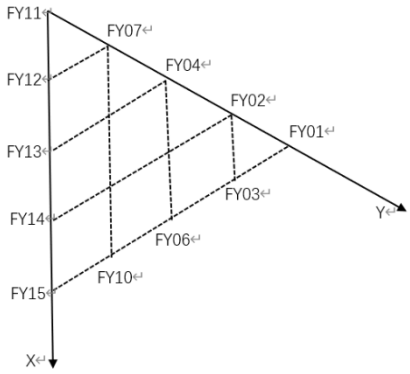



图 8 通过线性变换制得的非直角坐标系

FY11 的坐标为(0,0)　FY15 的坐标为(4,0)

FY01 的坐标为(0,4)　FY 的坐标为 12(1,0)

则$\overrightarrow{FY12FY11}=\{-1,0\}$　$\overrightarrow{FY12FY01}=\{-1,4\}$　$\overrightarrow{FY12FY13}=\{3,0\}$

根据如第一小问中的余弦公式可以解得：

$$\cos\alpha_1=-\frac{1}{\sqrt{13}}\qquad\cos\alpha_2=-1\qquad\cos\alpha_3=\frac{1}{\sqrt{13}}$$

由此可以确定 FY12 的位置。

### 5.4.3 基于粒子群算法的全局优化调整方案

**Step1 适应度函数**

我们通过 get_ideal_pairs() 函数，定义了理想锥形编队中所有应相距 50m 的无人机对。这个“连接对” 列表是锥形几何拓扑的数学表达，是算法判断队形优劣的唯一依据。

**指标**：为了量化一个“候选队形”与理想构型的差距，我们设计了 calculate_cone_fitness 函数。该函数接收一个“粒子”（也就是一个完整的 15 机编队方案）作为输入，然后执行以下操作：

- 遍历 get_ideal_pairs 列表中的每一对无人机。
- 计算这对无人机在当前粒子方案中的实际距离。
- 计算该实际距离与理想距离的误差平方。
- 将所有无人机对的误差平方累加，得到一个总误差值，即该粒子的适应度。适应度值越低，代表队形越接近完美的锥形。

**解空间与原点：**

- 解空间：是一个 28 维的空间（14 架可动无人机 $\times 2$ 个坐标）。
- 原点：我们将 FY11 无人机视为坐标原点(0,0)，不参与优化。所有其他无人机的最优位置都是相对于这个原点计算的

**Step2 粒子群优化算法**

粒子群优化算法在问题一的第三小问已经介绍，这里不再赘述。接下来我们论述在本问题中的粒子群的实现：

- 粒子定义：每个“粒子”代表一个由 14 架可动无人机坐标构成的完整编队方案。
- 优化流程：算法初始化 500 个“候选方案”（粒子），然后在 100 次迭代中，每个粒子依据其“个体历史 最佳”和“群体全局最佳”的经验来更新自己的飞行方向和速度。通过这种个体学习和社会学习的结合，整个粒子群逐渐向适应度函数值最低的区域收敛，最终找到全局最优的编队方案 gbest。

**Step3 参数设置**

- 粒子数量（n_particles）: 100。确保了足够的种群多样性，以探索复杂的 28 维解空间。
- 迭代次数（n_iterations）: 100。这里没有设为第三小问的 1000，而是设为 100，原因之一是本问中维度变大，迭代次数过多会大幅增加时间复杂度；二是 PSO 本身收敛快，100 次迭代足以达到 很好的效果。
- 惯性权重（w）: 0.5。平衡算法的全局探索和局部精细搜索能力。
- 认知系数（c1）: 0.8。粒子从自身历史经验中学习的比重。
- 社会系数（c2）: 0.9。粒子从群体智慧中学习的比重。

这些参数的设定是经过实践检验的。

### 5.4.4 优化效果对比

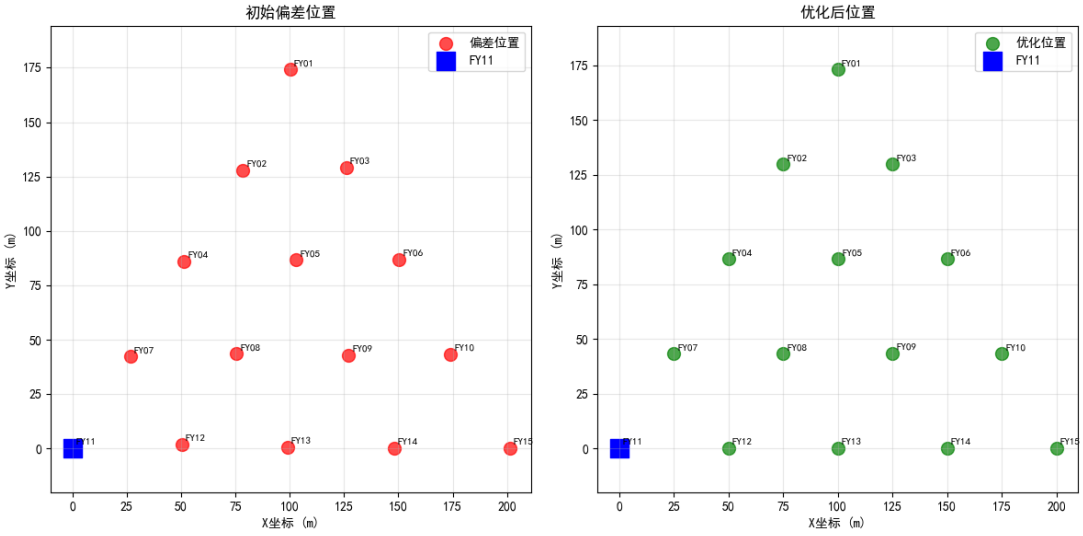



**图 9　初始布局（左） 优化后布局（右）的效果对比图**

左图是初始布局，无人机的位置里理想位置有所偏差；右图是 PSO 优化后的无人机布局，无人机几乎到达了理想位置。

表 2

| 测量点 | 初始偏差（m） | 优化后偏差（m） |
|---|---|---|
| **FY01** | 1.03 | 0.05 |
| **FY02** | 3.99 | 0.03 |
| **FY03** | 1.27 | 0.04 |
| **FY04** | 1.62 | 0.01 |
| **FY05** | 3.00 | 0.11 |
| **FY06** | 0.38 | 0.03 |
| **FY07** | 1.92 | 0.10 |
| **FY08** | 0.56 | 0.00 |
| **FY09** | 2.16 | 0.15 |
| **FY10** | 1.00 | 0.01 |
| **FY12** | 1.65 | 0.04 |
| **FY13** | 1.12 | 0.05 |
| **FY14** | 2.00 | 0.04 |
| **FY15** | 1.22 | 0.0 |
| **平均偏差** | 1.64 | 0.05 |
| **最大偏差** | 3.99 | 0.15 |

精度改进: 96.9%

表 2 通过对比，可以一目了然地看出本方案将一个有偏差的布局优化为一个高度精确、贴合理想的布局，这证明本方案具有实际可行性

# 六、模型的优缺点

## 6.1 模型的优点

本文所建立的模型理论基础且简洁，普适性强，可用于大多数在现实情况中无人机的定位，只需保证三个无人机位置信息的正确性即可由其发射信号，

通过纯方位无源定位的方法调整整个无人机群的位置，同时尽可能保持电磁静默，避免外界干扰。

在锥形编队调整方案的求解中，本文通过线性变换导出在非直角坐标系下的余弦求解公式，建立两个夹角为 60°的单位向量下的坐标系，完美适用于锥形编队，使得输入的坐标数据足够简洁方便，同样优化了余弦求解过程，能与前面问题讨论所得出的结论达到和谐的统一。

从算法角度来考虑，本模型使用粒子群优化算法，逐轮迭代寻找全局最优解，设立了足够多的"粒子"和足够多的轮次来避免无人机陷入局部最优。粒子群优化本身也是一个速度高效的算法，相对于"老牌"遗传算法而言，它收敛更快，计算参数更少，贴切无人机实时调位的应用场景。

## 6.2 模型的缺点

在若干无人机发射且编号未知的条件下，本文忽略了其对该编号自身编号的确定，而是考虑采用穷举法得到所有可能的情况，给问题的求解带来不确定的因素

本文对其余无人机位置的确定完全基于三台已知编号且位置完全正确的无人机，这使得在实际操作过程中需要严格操纵该三台无人机，给操纵者带来一定的挑战。

粒子群优化算法的效果不可避免地依赖于超参数的设置。而寻找最优超参数的过程是一个漫长的实践过程，为此我们提出了改进措施：可以使用网格搜索法，或者贝叶斯优化，来寻找最优超参组合。不过对于本体而言，无人机本身的偏离值就不大，超参数的好坏对粒子群优化效果的影响并不太极端。

# 七、参考文献

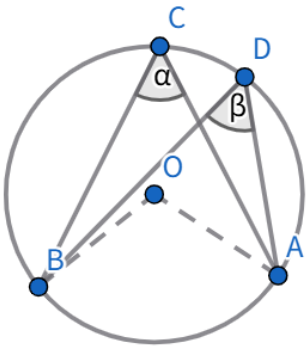
[1]. 王孟雅,杨佳杰,黄彬铭,等.基于多目标纯方位无源定位的无人机协作策略[J]. 电脑编程技巧与维护,2025,(05):94-97

[2]. 司守奎，孙玺菁.数学建模算法与应用 Python [M]. 北京：国防工业出版社，2022.

[3]. 丘维声.高等代数（第 2 版）[M]. 北京：清华大学出版社，2019.

# 八、附录

## 附录一：圆周角定理推论：同弧所对圆周角相等的证明




连接 OA、OB，则$\angle AOB$是弧$\overset{\frown}{AB}$所对的圆心角。应用圆周角定理，

对于圆周角$\angle ACB$，$\angle ACB=\frac{1}{2}\angle AOB$

对于圆周角$\angle ADB$，$\angle ADB=\frac{1}{2}\angle AOB$

等量代换可得，$\angle ACB=\angle ADB$

## 附录二：问题一代码

```
import numpy as np;

import matplotlib.pyplot as plt

import sympy as sp

class node:

    #已知无人机的直角坐标

    x_0 = 0

    y_0 = 0

    def __init__(self, i, j, a1, a2, a3, q):

        self.x_1 = np.cos(2 * (i-1) * np.pi / 9)
```

```
            self.y_1 = np.sin(2 * (i-1) * np.pi / 9)
            self.x_2 = np.cos(2 * (j-1) * np.pi / 9)
            self.y_2 = np.sin(2 * (j-1) * np.pi / 9)
            self.delx = sp.symbols('delta_x')
            self.dely = sp.symbols('delta_y')
            self.x_q = sp.cos(2 * (q-1) * sp.pi / 9) + self.delx
            self.y_q = sp.sin(2 * (q-1) * sp.pi / 9) + self.dely
            self.a1 = a1
            self.a2 = a2
            self.a3 = a3
def to_vector(startp, endp):
    start = np.array(startp)
    end = np.array(endp)
    return end - start
def solve(n):
    #三个向量  0i 0j ij
    v1 = to_vector([n.x_q, n.y_q], [n.x_0, n.y_0])
    v2 = to_vector([n.x_q, n.y_q], [n.x_1, n.y_1])
    v3 = to_vector([n.x_q, n.y_q], [n.x_2, n.y_2])
    m_v1 = sp.sqrt(v1[0]**2 + v1[1]**2)
```

```
        m_v2 = sp.sqrt(v2[0]**2 + v2[1]**2)
        m_v3 = sp.sqrt(v3[0]**2 + v3[1]**2)
        cos_a1 = sp.cos(sp.rad(float(n.a1)))
        cos_a2 = sp.cos(sp.rad(float(n.a2)))
        cos_a3 = sp.cos(sp.rad(float(n.a3)))

        eq1 = sp.Eq((v1[0]*v2[0] + v1[1]*v2[1]) / (m_v1 * m_v2), cos_a1)
        eq2 = sp.Eq((v1[0]*v3[0] + v1[1]*v3[1]) / (m_v1 * m_v3), cos_a2)
        eq3 = sp.Eq((v2[0]*v3[0] + v2[1]*v3[1]) / (m_v2 * m_v3), cos_a3)
        solution = sp.nsolve((eq1, eq2, eq3), (n.delx, n.dely), (0, 0),
verify = False)
        print("偏移量求解结果[Δx, Δy]是:" ,solution)
        x_q_real = np.cos(2 * (q - 1) * np.pi / 9) + float(solution[0])
        y_q_real = np.sin(2 * (q - 1) * np.pi / 9) + float(solution[1])
        print(f"实际坐标: ({x_q_real:.6f}, {y_q_real:.6f})")
        with open("q1_1_result.csv", "a") as f:
            f.write(f"{q},{x_q_real:.6f},{y_q_real:.6f}\n")
        return
if __name__ == "__main__":
        with open("q1_1_result.csv", "w") as f:
            f.write("接收信号无人机编号,x 坐标,y 坐标\n")
```

```
    n1, n2 = map(int, input("请输入 2 个发射信号的无人机编号，用空格分隔:").split())
    #a1 为 q 0 和 q n1 对应的夹角，a2 为 q 0 和 q n2 对应的夹角，a3 为 q n1 和 q n2 对应的夹角
    for i in range(1,8):
        q = int(input("请输入当前接收信号的无人机编号:"))
        a1, a2, a3 = map(float, input("请输入已知角度，用空格分隔:").split())
        n = node(n1, n2, a1, a2, a3, q)
        solve(n)
    '''
```

测试数据:

input:

1 2

3

49.5 69.5 20

4

29 49 20

7

28.5 8.5 20

9

```
67.5 47.5 20
'''
```

## 附录三：问题二代码

```
#本代码部分借鉴了教材'司守奎，孙玺菁 数学建模算法与应用 Python'
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Arc
from matplotlib.patches import Circle
fig, ax = plt.subplots()
def draw_angle(ax, p_center, p1, p2, r=0.4, color='red', label=None,
fontsize=12):
    v1 = np.array([p1[0] - p_center[0], p1[1] - p_center[1]])
    v2 = np.array([p2[0] - p_center[0], p2[1] - p_center[1]])
    def angle(v):
        return np.degrees(np.arctan2(v[1], v[0]))
    angle1 = angle(v1)
    angle2 = angle(v2)
    if angle2 < angle1:
        angle1, angle2 = angle2, angle1

    arc = Arc(p_center, width=r*2, height=r*2, theta1=angle1,
```

```
theta2=angle2, edgecolor=color, lw=2)
        ax.add_patch(arc)
        if label:
            mid_angle = np.radians((angle1 + angle2) / 2)
            label_x = p_center[0] + (r + 0.1) * np.cos(mid_angle)
            label_y = p_center[1] + (r + 0.1) * np.sin(mid_angle)
            ax.text(label_x, label_y, label, fontsize=fontsize,
color=color,
                    horizontalalignment='center',
verticalalignment='center')
def draw_single_circle(C = (0,1), R = 1, p1 = None, p2 = None, p3 =
None):
        p_half = ((p1[0] + p2[0])/2, (p1[1] + p2[1])/2)
        #circle = plt.Circle(C, R, color = 'blue', fill = False)
        #ax.add_patch(circle)
        ax.set_aspect('equal')
        ax.scatter(p1[0], p1[1], color = 'black')
        ax.scatter(p2[0], p2[1], color = 'black')
        ax.scatter(p3[0], p3[1], color = 'black')
        ax.scatter(C[0], C[1], color = 'blue')
        ax.scatter(p_half[0], p_half[1], color = 'black')
```

```
        ax.text(p1[0], p1[1] - 0.13, p1[2], fontsize = 12, color = 'green',
verticalalignment = 'bottom')
        ax.text(p2[0]-0.25, p2[1] - 0.13, p2[2], fontsize = 12, color =
'green', verticalalignment = 'bottom')
        ax.text(p3[0]-0.25, p3[1], 'FY_q', fontsize = 12, color = 'green',
verticalalignment = 'bottom')
        x_line = [p1[0], p2[0], p3[0], p1[0]]
        y_line = [p1[1], p2[1], p3[1], p1[1]]
        ax.plot(x_line, y_line, color = 'black')
        ax.plot([p2[0], C[0], p_half[0]], [p2[1], C[1], p_half[1]], color =
'black', linestyle = '--')
        draw_angle(ax, p3, p1, p2, r = 0.1, color = 'black', label =
r'$\alpha_1$', fontsize = 8)
        draw_angle(ax, C, p2, p_half, r = 0.1, color = 'black', label =
r'$\alpha_1$', fontsize = 8)
        v1 = np.array([p1[0] - C[0], p1[1] - C[1]])
        v2 = np.array([p2[0] - C[0], p2[1] - C[1]])

        angle = lambda v: np.degrees(np.arctan2(v[1], v[0]))
        angle1 = angle(v1)
        angle2 = angle(v2)
```

```
        if angle2 < angle1:
            angle1, angle2 = angle2, angle1
        if angle2 - angle1 > 180:
            angle1 += 360
        arc_dash = Arc(C, width=2*R, height=2*R, theta1=angle1,
theta2=angle2, edgecolor='blue', linestyle='--', lw=2)
        ax.add_patch(arc_dash)
        arc_solid1 = Arc(C, width=2*R, height=2*R, theta1=angle2,
theta2=angle1 + 360, edgecolor='blue', linestyle='-', lw=2)
        ax.add_patch(arc_solid1)
def distance(a, b):
        return ((a[0] - b[0])**2 + (a[1] - b[1])**2)**0.5
def draw_3c(ax, P, C1, C2, C3, color = ('red', 'blue', 'green'))
        R1 = distance(C1, P)
        R2 = distance(C2, P)
        R3 = distance(C3, P)
        circle1 = Circle(C1, R1, color = color[0], fill = False)
        circle2 = Circle(C2, R2, color = color[1], fill = False)
        circle3 = Circle(C3, R3, color = color[2], fill = False)
        ax.add_patch(circle1)
        ax.add_patch(circle2)
```

```
        ax.add_patch(circle3)
        ax.scatter(P[0], P[1], color = 'black', zorder = 5)
        ax.text(P[0] + 0.03, P[1] + 0.2, 'P', fontsize = 8, color = 'green')
        jiao1, jiao2, jiao3 = (0.07, 5.62), (0.36, 0.53), (4.09, -0.24)
        ax.scatter([jiao1[0], jiao2[0], jiao3[0]], [jiao1[1], jiao2[1],
jiao3[1]])
        ax.text(jiao1[0] + 0.2, jiao1[1], 'A', fontsize = 8, va = 'center')
        ax.text(jiao2[0] + 0.2, jiao2[1], 'B', fontsize = 8, va = 'center')
        ax.text(jiao3[0] + 0.2, jiao3[1] - 0.2, 'C', fontsize = 8, va = 'center')
        arrowprops = dict(arrowstyle="->", color='black', lw=1.5)
        ax.annotate("", xy=jiao1, xytext=P,
arrowprops=dict(arrowstyle="->", color='black', lw=0.9))
        ax.annotate("", xy=jiao2, xytext=P,
arrowprops=dict(arrowstyle="->", color='black', lw=0.9))
        ax.annotate("", xy=jiao3, xytext=P,
arrowprops=dict(arrowstyle="->", color='black', lw=0.9))
        '''
        ax.scatter([C1[0], C2[0], C3[0]], [C1[1], C2[1], C3[1]], c=color,
s=50)
        ax.text(C1[0] + 0.1, C1[1], 'C1', fontsize=8, va='center')
        ax.text(C2[0] + 0.1, C2[1], 'C2', fontsize=8, va='center')
```

```
        ax.text(C3[0] + 0.1, C3[1], 'C3', fontsize=8, va='center')
        '''
if __name__ == "__main__":
        draw_single_circle(C = (0,1), R = 1, p1 = (0.8660254037844386,
0.5, 'FY01'), p2 = (-0.8660254037844386, 0.5, 'FY00'), p3 = (-0.6, 1.8))
        #draw_3c(ax, (1,1), (-1,3), (4,4), (2,-1))
        plt.title("questions 1_2_one-circle")
        plt.xlabel("X")
        plt.ylabel("Y")
        plt.grid(False)
        plt.axis('equal')
        plt.show()
        #print((1-0.8**2)**0.5)
```

## 附录四：问题三代码

```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
def calculate_fitness(particle):
        positions = particle.reshape((9, 2))
        # 半径偏差
```

```
        radii = np.sqrt(np.sum(positions**2, axis=1))
        avg_radius = np.mean(radii)
        radius_cost = np.sum((radii - avg_radius)**2)
            # 角度间隔
        angles = np.arctan2(positions[:, 1], positions[:, 0]) * 180 / np.pi
            sorted_angles = np.sort(angles)
        # 相邻角度差
        diffs = np.diff(sorted_angles)
        last_diff = (sorted_angles[0] + 360) - sorted_angles[-1]
        all_diffs = np.append(diffs, last_diff)
        ideal_angle = 40.0
        angle_cost = np.sum((all_diffs - ideal_angle)**2)
            w1 = 1.0   # 半径成本的权重
        w2 = 1.0   # 角度成本的权重
            total_fitness = w1 * radius_cost + w2 * angle_cost
          return total_fitness
def pso_optimization(initial_positions, n_particles=500,
n_iterations=1000):
        dimensions = 18          # 维度
        w = 0.5                  # 惯性
        c1 = 0.8                 # 认知
```

```
    c2 = 0.9                    # 社会
      # 搜索范围
    initial_flat = np.array(initial_positions[1:]).flatten()  # 排除
FY00
    search_space_min = initial_flat - 50
    search_space_max = initial_flat + 50
       # 粒子的初始位置和速度
    particles_pos = np.random.uniform(search_space_min,
search_space_max, (n_particles, dimensions))
    particles_vel = np.zeros((n_particles, dimensions))
    pbest_pos = particles_pos.copy()
    pbest_fitness = np.array([calculate_fitness(p) for p in
particles_pos])
    gbest_pos = pbest_pos[np.argmin(pbest_fitness)]
    gbest_fitness = np.min(pbest_fitness)


    print(f"初始最佳适应度: {gbest_fitness:.6f}")
    for i in range(n_iterations):
        for j in range(n_particles):
            r1 = np.random.rand(dimensions)
            r2 = np.random.rand(dimensions)
```

```
                cognitive_vel = c1 * r1 * (pbest_pos[j] -
particles_pos[j])
                social_vel = c2 * r2 * (gbest_pos - particles_pos[j])
                particles_vel[j] = w * particles_vel[j] + cognitive_vel +
social_vel
                particles_pos[j] += particles_vel[j]
                particles_pos[j] = np.clip(particles_pos[j],
search_space_min, search_space_max)
                # 个体最优
                current_fitness = calculate_fitness(particles_pos[j])
                if current_fitness < pbest_fitness[j]:
                    pbest_fitness[j] = current_fitness
                    pbest_pos[j] = particles_pos[j].copy()
            # 全局最优
            if np.min(pbest_fitness) < gbest_fitness:
                gbest_fitness = np.min(pbest_fitness)
                gbest_pos = pbest_pos[np.argmin(pbest_fitness)]
            if i % 200 == 0:
                print(f"迭代次数 {i}: 最佳适应度 = {gbest_fitness:.6f}")
        print(f"最终最佳适应度: {gbest_fitness:.6f}")
        optimal_coords = gbest_pos.reshape((9, 2))
```

```
    return optimal_coords

def calculate_adjustment_plan(initial_positions,
optimal_positions):
    adjustment_steps = []
    for i in range(1, 10):
        initial_pos = initial_positions[i]
        target_pos = optimal_positions[i-1]
        displacement = np.array(target_pos) - np.array(initial_pos)
        distance = np.linalg.norm(displacement)
        if distance > 0.1:
            angle = np.arctan2(displacement[1], displacement[0])
* 180 / np.pi
            step = {
                'drone_id': f'FY{i:02d}',
                'from': initial_pos,
                'to': target_pos,
                'distance': distance,
                'direction': angle
            }
            adjustment_steps.append(step)
    return adjustment_steps
```

```
def visualize_positions(initial_positions, optimal_positions):
    """
    可视化初始位置和优化后的位置
    """
    plt.rcParams['font.sans-serif'] = ['SimHei', 'DejaVu Sans']
    plt.rcParams['axes.unicode_minus'] = False
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    initial_x = [pos[0] for pos in initial_positions]
    initial_y = [pos[1] for pos in initial_positions]
    plt.scatter(initial_x, initial_y, c='red', s=100, alpha=0.7,
label='Initial Positions')
    plt.scatter(0, 0, c='blue', s=200, marker='s', label='FY00
(Center)')
    for i, (x, y) in enumerate(initial_positions):
        plt.annotate(f'FY{i:02d}', (x, y), xytext=(5, 5),
textcoords='offset points')
    plt.title('Initial UAV Positions')
    plt.xlabel('X Coordinate (m)')
    plt.ylabel('Y Coordinate (m)')
    plt.grid(True, alpha=0.3)
```

```
        plt.legend()
        plt.axis('equal')
        plt.subplot(1, 2, 2)
        opt_x = [pos[0] for pos in optimal_positions]
        opt_y = [pos[1] for pos in optimal_positions]
        plt.scatter(opt_x, opt_y, c='green', s=100, alpha=0.7,
label='Optimized Positions')
        plt.scatter(0, 0, c='blue', s=200, marker='s', label='FY00
(Center)')
        for i, (x, y) in enumerate(optimal_positions):
            plt.annotate(f'FY{i:02d}', (x, y), xytext=(5, 5),
textcoords='offset points')
        plt.title('Optimized UAV Positions')
        plt.xlabel('X Coordinate (m)')
        plt.ylabel('Y Coordinate (m)')
        plt.grid(True, alpha=0.3)
        plt.legend()
        plt.axis('equal')
        plt.tight_layout()
        plt.show()
def analyze_formation_quality(positions):
```

```
radii = [np.sqrt(x**2 + y**2) for x, y in positions[1:]]
avg_radius = np.mean(radii)
radius_std = np.std(radii)
angles = [np.arctan2(y, x) * 180 / np.pi for x, y in positions[1:]]
sorted_angles = sorted(angles)
angle_diffs = []
for i in range(len(sorted_angles)):
    if i == len(sorted_angles) - 1:
        diff = (sorted_angles[0] + 360) - sorted_angles[i]
    else:
        diff = sorted_angles[i+1] - sorted_angles[i]
    angle_diffs.append(diff)
ideal_angle = 40.0
angle_std = np.std(angle_diffs)
print(f"\n 编队质量分析:")
print(f"平均半径: {avg_radius:.2f} m")
print(f"半径标准差: {radius_std:.2f} m")
print(f"理想角度间隔: {ideal_angle:.1f}°")
print(f"实际角度间隔: {[f'{diff:.1f}°' for diff in angle_diffs]}")
print(f"角度间隔标准差: {angle_std:.2f}°")
return {
```

```
            'avg_radius': avg_radius,
            'radius_std': radius_std,
            'angle_diffs': angle_diffs,
            'angle_std': angle_std
        }
def to_polar_zb(zj_positions):
        polar_positions = []
        for x, y in zj_positions:
            r = np.sqrt(x**2 + y**2)
            theta = np.arctan2(y, x)
            polar_positions.append((r, np.degrees(theta)))
        return polar_positions
if __name__ == "__main__":
        dir = os.path.dirname(os.path.abspath(__file__))
        file_path = os.path.join(dir, 'q1_3_jizb.csv')
        data = pd.read_csv(file_path)
        lie = []
        for index, row in data.iterrows():
            id = int(row['无人机编号'])
            jzb = row['极坐标 m']
            angle = np.radians(row['极坐标°'])
```

```
        lie.append((jzb * np.cos(angle), jzb * np.sin(angle)))
print("初始无人机位置（直角坐标):")
for i, pos in enumerate(lie):
        print(f"FY{i:02d}: ({pos[0]:.2f}, {pos[1]:.2f})")
# PSO
optimal_positions = pso_optimization(lie)
print("\n 优化后的 FY01-FY09 位置坐标（x, y):")
for i, pos in enumerate(optimal_positions):
        print(f"FY{i+1:02d}: ({pos[0]:.2f}, {pos[1]:.2f})")
polar_op_positions = to_polar_zb(optimal_positions)
print("\n 优化后的 FY01-FY09 的极坐标:")
for i, (r, theta) in enumerate(polar_op_positions):
        print(f"FY{i+1:02d}: 半径={r:.6f}m，角度={theta:.6f}°")
# 可视化
all_optimal = [(0, 0)] + [tuple(pos) for pos in optimal_positions]
visualize_positions(lie, all_optimal)
# 分析编队质量
print("\n 初始编队质量:")
initial_quality = analyze_formation_quality(lie)
print("\n 优化后编队质量:")
optimal_quality = analyze_formation_quality(all_optimal)
```

```
    results_df = pd.DataFrame({
        'UAV_ID': [f'FY{i+1:02d}' for i in range(9)],
        'Initial_X': [pos[0] for pos in lie[1:]],
        'Initial_Y': [pos[1] for pos in lie[1:]],
        'Optimal_X': [pos[0] for pos in optimal_positions],
        'Optimal_Y': [pos[1] for pos in optimal_positions],
        'Movement_Distance':
[np.linalg.norm(np.array(optimal_positions[i]) - np.array(lie[i+1]))
                                for i in range(9)]
    })

    output_file = os.path.join(dir, 'q1_3_optimization_results.csv')
    results_df.to_csv(output_file, index=False, encoding='utf-8-
sig')
```

## 附录五：问题四代码

```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
def load_ideal_coordinates():
    dir =os.path.dirname(os.path.abspath(__file__))
```

```
    original_file =os.path.join(dir,'coordinate_original.csv')
    data = pd.read_csv(original_file)
    ideal_coords = {}
    uav_to_index = {
        'FY11': 0, 'FY01': 1, 'FY02': 2, 'FY03': 3, 'FY04': 4, 'FY05': 5,
        'FY06': 6, 'FY07': 7, 'FY08': 8, 'FY09': 9, 'FY10': 10,
        'FY12': 11, 'FY13': 12, 'FY14': 13, 'FY15': 14
    }

    for _, row in data.iterrows():
        uav_id = row['无人机编号']
        if uav_id in uav_to_index:
            index = uav_to_index[uav_id]
            ideal_coords[index] = [row['x 坐标'], row['y 坐标']]
    return ideal_coords
def calculate_fitness(particle):
    movable_positions = particle.reshape((14, 2))
    positions = np.zeros((15, 2))
    positions[0] = [0, 0]
    positions[1:] = movable_positions
    ideal_coords = load_ideal_coordinates()
```

```
    position_cost = 0.0
    for i in range(15):
        ideal_pos = np.array(ideal_coords[i])
        actual_pos = positions[i]
        position_cost += np.linalg.norm(actual_pos - ideal_pos) **
2
        key_distance_pairs = [
        (1, 2), (1, 3),
        (2, 4), (2, 5), (3, 5), (3, 6),
        (4, 7), (4, 8), (5, 8), (5, 9), (6, 9), (6, 10),
        (7, 11), (8, 12), (9, 13), (10, 14),
        (2, 3), (4, 5), (5, 6), (7, 8), (8, 9), (9, 10),
        (11, 12), (12, 13), (13, 14)
    ]
        distance_cost = 0.0
    ideal_distance = 50.0
    for i, j in key_distance_pairs:
        actual_distance = np.linalg.norm(positions[i] - positions[j])
        distance_cost += (actual_distance - ideal_distance) ** 2
    w1 = 10.0     # 位置收敛权重(主)
    w2 = 0.1      # 距离约束权重(辅)
```

```
    total_fitness = w1 * position_cost + w2 * distance_cost
    return total_fitness
def pso_optimization(initial_positions, n_particles=500,
n_iterations=100):
    """
    目标导向的 PSO 优化
    """
    dimensions = 28
    w = 0.5
    c1 = 0.8
    c2 = 0.9
    ideal_coords_dict = load_ideal_coordinates()
    ideal_coords = np.array([
        ideal_coords_dict[1],    # FY01
        ideal_coords_dict[2],    # FY02
        ideal_coords_dict[3],    # FY03
        ideal_coords_dict[4],    # FY04
        ideal_coords_dict[5],    # FY05
        ideal_coords_dict[6],    # FY06
        ideal_coords_dict[7],    # FY07
        ideal_coords_dict[8],    # FY08
```

```
            ideal_coords_dict[9],     # FY09
            ideal_coords_dict[10],    # FY10
            ideal_coords_dict[11],    # FY12
            ideal_coords_dict[12],    # FY13
            ideal_coords_dict[13],    # FY14
            ideal_coords_dict[14]     # FY15
        ]).flatten()

        search_space_min = ideal_coords - 30
        search_space_max = ideal_coords + 30
        particles_pos = np.zeros((n_particles, dimensions))
        current_flat = np.array(initial_positions[1:]).flatten()
        for i in range(n_particles // 2):
            particles_pos[i] = current_flat + np.random.normal(0, 10,
dimensions)
        for i in range(n_particles // 2, n_particles):
            particles_pos[i] = ideal_coords + np.random.normal(0, 5,
dimensions)
        particles_pos = np.clip(particles_pos, search_space_min,
search_space_max)
        particles_vel = np.zeros((n_particles, dimensions))
```

```
    pbest_pos = particles_pos.copy()
    pbest_fitness = np.array([calculate_fitness(p) for p in
particles_pos])
    gbest_pos = pbest_pos[np.argmin(pbest_fitness)]
    gbest_fitness = np.min(pbest_fitness)
    print(f"初始最佳适应度: {gbest_fitness:.2f}")
    for i in range(n_iterations):
        for j in range(n_particles):
            r1 = np.random.rand(dimensions)
            r2 = np.random.rand(dimensions)
            cognitive_vel = c1 * r1 * (pbest_pos[j] -
particles_pos[j])
            social_vel = c2 * r2 * (gbest_pos - particles_pos[j])
            particles_vel[j] = w * particles_vel[j] + cognitive_vel +
social_vel
            particles_pos[j] += particles_vel[j]
            particles_pos[j] = np.clip(particles_pos[j],
search_space_min, search_space_max)
            current_fitness = calculate_fitness(particles_pos[j])
            if current_fitness < pbest_fitness[j]:
                pbest_fitness[j] = current_fitness
```

```
                pbest_pos[j] = particles_pos[j].copy()
        if np.min(pbest_fitness) < gbest_fitness:
            gbest_fitness = np.min(pbest_fitness)
            gbest_pos = pbest_pos[np.argmin(pbest_fitness)]
        if i % 50 == 0:
            print(f"迭代 {i}: 适应度 = {gbest_fitness:.2f}")
    print(f"最终适应度: {gbest_fitness:.2f}")
    return gbest_pos.reshape((14, 2))
def analyze_formation_accuracy(initial_positions,
optimal_positions):
    """
    分析编队与理想坐标的接近程度
    """
    print("\n=== 编队精度分析 ===")
      ideal_coords = load_ideal_coordinates()
       mapping = [
        (1, 'FY01'), (2, 'FY02'), (3, 'FY03'), (4, 'FY04'), (5, 'FY05'),
        (6, 'FY06'), (7, 'FY07'), (8, 'FY08'), (9, 'FY09'), (10, 'FY10'),
        (11, 'FY12'), (12, 'FY13'), (13, 'FY14'), (14, 'FY15')
    ]
    print("初始位置与理想位置偏差:")
```

```
initial_errors = []
for i, (initial_idx, drone_id) in enumerate(mapping):
    initial_pos = np.array(initial_positions[initial_idx])
    ideal_pos = np.array(ideal_coords[initial_idx])
    error = np.linalg.norm(initial_pos - ideal_pos)
    initial_errors.append(error)
    print(f"    {drone_id}: {error:.2f}m")

print(f"\n 初始平均偏差: {np.mean(initial_errors):.2f}m")
print(f"初始最大偏差: {max(initial_errors):.2f}m")
print("\n 优化后位置与理想位置偏差:")
optimal_errors = []
all_optimal = [(0, 0)] + [tuple(pos) for pos in optimal_positions]
for i, (initial_idx, drone_id) in enumerate(mapping):
    optimal_pos = np.array(all_optimal[initial_idx])
    ideal_pos = np.array(ideal_coords[initial_idx])
    error = np.linalg.norm(optimal_pos - ideal_pos)
    optimal_errors.append(error)
    print(f"    {drone_id}: {error:.2f}m")
print(f"\n 优化后平均偏差: {np.mean(optimal_errors):.2f}m")
print(f"优化后最大偏差: {max(optimal_errors):.2f}m")
```

```
    improvement = ((np.mean(initial_errors) -
np.mean(optimal_errors)) / np.mean(initial_errors)) * 100
    print(f"\n 精度改进: {improvement:.1f}%")
    return {
        'initial_avg_error': np.mean(initial_errors),
        'optimal_avg_error': np.mean(optimal_errors),
        'improvement': improvement
    }
def visualize_comparison(initial_positions, optimal_positions):
    plt.rcParams['font.sans-serif'] = ['SimHei', 'DejaVu Sans']
    plt.rcParams['axes.unicode_minus'] = False
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 6))
    ideal_coords_dict = load_ideal_coordinates()
    ideal_coords = [
        ideal_coords_dict[0],     #同上
        ideal_coords_dict[1],
        ideal_coords_dict[2],
        ideal_coords_dict[3],
        ideal_coords_dict[4],
        ideal_coords_dict[5],
        ideal_coords_dict[6],
```

```
            ideal_coords_dict[7],
            ideal_coords_dict[8],
            ideal_coords_dict[9],
            ideal_coords_dict[10],
            ideal_coords_dict[11],
            ideal_coords_dict[12],
            ideal_coords_dict[13],
            ideal_coords_dict[14]
        ]
        uav_labels = ['FY11', 'FY01', 'FY02', 'FY03', 'FY04', 'FY05', 'FY06',
'FY07', 'FY08', 'FY09', 'FY10', 'FY12', 'FY13', 'FY14', 'FY15']
        ideal_x = [pos[0] for pos in ideal_coords]
        ideal_y = [pos[1] for pos in ideal_coords]
        ax1.scatter(ideal_x, ideal_y, c='blue', s=100, alpha=0.8, label='
理想位置')
        ax1.scatter(0, 0, c='red', s=200, marker='s', label='FY11')
        for i, (x, y) in enumerate(ideal_coords):
            ax1.annotate(uav_labels[i], (x, y), xytext=(3, 3),
textcoords='offset points', fontsize=8)
        ax1.set_title('理想锥形编队')
        ax1.set_xlabel('X 坐标 (m)')
```

```
        ax1.set_ylabel('Y 坐标 (m)')
        ax1.grid(True, alpha=0.3)
        ax1.legend()
        ax1.axis('equal')
        initial_x = [pos[0] for pos in initial_positions]
        initial_y = [pos[1] for pos in initial_positions]
        ax2.scatter(initial_x, initial_y, c='red', s=100, alpha=0.7, label='
偏差位置')
        ax2.scatter(0, 0, c='blue', s=200, marker='s', label='FY11')
        for i, (x, y) in enumerate(initial_positions):
            ax2.annotate(uav_labels[i], (x, y), xytext=(3, 3),
textcoords='offset points', fontsize=8)
        ax2.set_title('初始偏差位置')
        ax2.set_xlabel('X 坐标 (m)')
        ax2.set_ylabel('Y 坐标 (m)')
        ax2.grid(True, alpha=0.3)
        ax2.legend()
        ax2.axis('equal')
        all_optimal = [(0, 0)] + [tuple(pos) for pos in optimal_positions]
        opt_x = [pos[0] for pos in all_optimal]
        opt_y = [pos[1] for pos in all_optimal]
```

```
        ax3.scatter(opt_x, opt_y, c='green', s=100, alpha=0.7, label='优
化位置')
        ax3.scatter(0, 0, c='blue', s=200, marker='s', label='FY11')
        for i, (x, y) in enumerate(all_optimal):
            ax3.annotate(uav_labels[i], (x, y), xytext=(3, 3),
textcoords='offset points', fontsize=8)
        ax3.set_title('优化后位置')
        ax3.set_xlabel('X 坐标  (m)')
        ax3.set_ylabel('Y 坐标  (m)')
        ax3.grid(True, alpha=0.3)
        ax3.legend()
        ax3.axis('equal')
        plt.tight_layout()
        plt.show()
if __name__ == "__main__":
        dir = os.path.dirname(os.path.abspath(__file__))
        file_path = os.path.join(dir, 'coordinate_mess.csv')
        data = pd.read_csv(file_path)
            lie = []
        for index, row in data.iterrows():
            lie.append((row['x 坐标'], row['y 坐标']))
```

```
    print("===  目标导向锥形编队优化  ===")
    print("策略：以理想坐标为目标进行收敛优化")
    print("\n 开始 PSO 优化...")
    optimal_positions = pso_optimization(lie)
      # 精度分析
    accuracy_results = analyze_formation_accuracy(lie,
optimal_positions)
       # 移动方案
    movements = []
    mapping = [
        (1, 'FY01'), (2, 'FY02'), (3, 'FY03'), (4, 'FY04'), (5, 'FY05'),
        (6, 'FY06'), (7, 'FY07'), (8, 'FY08'), (9, 'FY09'), (10, 'FY10'),
        (11, 'FY12'), (12, 'FY13'), (13, 'FY14'), (14, 'FY15')
    ]
        for i, (initial_idx, drone_id) in enumerate(mapping):
        initial_pos = lie[initial_idx]
        target_pos = optimal_positions[i]
        displacement = np.array(target_pos) - np.array(initial_pos)
        distance = np.linalg.norm(displacement)
        if distance > 0.1:
            angle = np.arctan2(displacement[1], displacement[0])
```

```
* 180 / np.pi
                movements.append({
                    'drone_id': drone_id,
                    'distance': distance,
                    'direction': angle
                })
            print(f"\n 调整方案 (共{len(movements)}架无人机):")
        for move in movements:
            print(f"    {move['drone_id']}: 移动
{move['distance']:.2f}m, 方向{move['direction']:.1f}°")
        # 可视化
        visualize_comparison(lie, optimal_positions)
        results_data = []
        for i, (initial_idx, drone_id) in enumerate(mapping):
            results_data.append({
                'UAV_ID': drone_id,
                'Initial_X': lie[initial_idx][0],
                'Initial_Y': lie[initial_idx][1],
                'Optimal_X': optimal_positions[i][0],
                'Optimal_Y': optimal_positions[i][1],
                'Movement_Distance':
```

```
np.linalg.norm(np.array(optimal_positions[i]) -
np.array(lie[initial_idx]))
            })
    results_df = pd.DataFrame(results_data)
    output_file = os.path.join(dir, 'q2_target_guided_results.csv')
    results_df.to_csv(output_file, index=False, encoding='utf-8-
sig')
```