基于 K-means 聚类的角度自适应无人机编队调整模型

摘要

在基于 $K-means$ 聚类的无人机编队角度自适应调整模型中，无人机能够仅仅通过接收到的方向信息调整与其他飞机间的夹角，从而减少无人机编队保持和调整过程中发射电磁波的次数和数量，避免外界干扰。

对于问题一 (1)，根据圆的几何性质，建立定位无人机极坐标变量 (ρ,θ) 与无人机方向信息 α_1,α_2 的非线性方程组，分圆心无人机在定位无人机**左侧，中间，右侧**三种情况建立被动接收信号无人机的定位模型。

对于问题一 (2)，**第一步**，建立增加一架无人机后，定位无人机直角坐标 (x,y) 与夹角 α_1,α_2 的非线性方程组模型，发现定位无人机的坐标含有多个解，当无人机偏离角度较小时，可以进行定位。**第二步**，建立增加两架无人机后定位无人机直角坐标 (x,y) 与夹角 $\alpha_1,\alpha_2,\alpha_3$ 的非线性方程组模型，此时有唯一解。**因此，定位无人机在偏离角度较小时增加一架无人机即可以有效定位，偏离较大时增加两架无人机可以有效定位**。

对于问题一 (3)，**第一步**，分析 **10** 架无人机任意两架之间所有可能出现的夹角，发现夹角只可能出现 $[10,20,30,40,50,60,70,80]$ 的情况，如果无人机偏离的夹角在 $(-5\%,5\%)$ 的范围内，可以使用 $K-means$ 聚类的方法调整定位无人机与发射信号无人机的夹角，从而调整无人机的位置。**第二步**，结合问题一 **(1)** 的模型，采用圆心上和圆周上共三架无人机发射信号，使用可能出现的夹角集合建立 $K-means$ 聚类调整模型，建立原始角度到正确角度的映射关系，结合问题一 **(2)** 的方法，可以进一步得出通过角度调节后无人机极坐标。**第三步**，选取编号为 $0,4,7$ 的无人机发射信号，通过第二步建立的模型，求解需要调整的角度进行调整，得出第一次调整后的坐标；继续选取编号为 $0,1,2$ 的无人机发射信号，求解需要调整的角度进行调整，得出第二次调整后的坐标；**经过检验，第二次调整之后，无人机已经均匀分布在半径为 $105.77m$ 的圆周上**。

对于问题二，**第一步**，以 $FY13$ 为坐标原点建立直角坐标系，编号为 $FY05$，$FY08$，$FY09$ 的无人机为发射信号无人机，坐标为真实值，其他无人机均带有随机生成的噪声，噪声的范围为 $(-2.5\%,2.5\%)$。第二步，对无人机的夹角进行分析，发现只有 $[0,19.11,30]$ 三种情况，故建立基于 $K-means$ 聚类的无人机编队角度自适应调整模型并进行计算机仿真，结果显示无人机仅需要调整一次即可调整到正确位置上。

最后，使用锥形的无人机数据对模型进行灵敏度分析，总结模型的优缺点，并提出模型可改进的方向。

关键字：　K-means 聚类　自适应　无源定位　编队保持

一、 问题重述

1.1 问题背景

无人机编队保持是无人机研究的一个重要内容，无人机编队为了保持整体的队形，需要向其他无人机发射电磁波信号，为了避免外界的干扰，无人机应减少发射电磁波信号的次数和发射电磁波的无人机数量。在纯方位无源定位的问题背景下，获取信号的无人机仅仅只能获取发射信号飞机之间的夹角，并调整自身的位置。

1.2 问题要求

- **问题一 (1)** 利用位于圆心的无人机（$FY00$）和编队中另 2 架无人机的编号信息和接收信号无人机获取的夹角信息，建立数学模型，在直径已知的情况下，求解接收信号无人机的极坐标。
- **问题一 (2)** 仅利用接收信号无人机得到的 $FY00$ 和 $FY01$ 和编队中其他若干架无人机夹角信息，建立数学模型，说明除 $FY00$ 和 $FY01$ 外，还需要几架无人机发射信号，才能求出无人机的直角坐标。
- **问题一 (3)** 根据 (1)(2) 的分析和模型，选取圆周和圆周上最多 3 架飞机发射信号，建立数学模型，求解接收信号无人机根据获取的夹角信息调整与发射信号无人机的夹角的值，使无人机最终均匀的分布在某一个圆周上。利用建立的数学模型进行计算机仿真，给出每次调整的角度和调整后无人机的坐标。
- **问题二** 结合问题一，建立直角坐标系并写出无人机正确的坐标值，确定发射信号的无人机编号，对接收信号的无人机坐标加噪声，生成有偏差的无人机锥形梯队坐标数据。使用问题一建立的模型进行计算机仿真，给出每次调整的角度和调整后无人机的坐标。

二、 问题分析

2.1 问题一 (1) 的分析

发射信号无人机的编号已知，根据圆的性质，可以得出圆心角的信息，可以假设需要定位的无人机的极坐标为 (ρ, θ)，获取到的夹角为 α_1, α_2，通过几何特征，建立方程组，从而求解接收信号无人机的极坐标，进而建立被动接收信号无人机的定位模型。

2.2 问题一 (2) 的分析

可以假设需要定位的无人机的极坐标为 (ρ,θ)，先在圆周上增加一架发射信号的无人机，并设获取到的夹角为 α_1,α_2，通过几何特征，建立方程组，看接收信号无人机的极坐标是否有解和唯一解，如果无法求解，再增加一架发射信号的无人机，并设获取到的夹角为 $\alpha_1,\alpha_2,\alpha_3$, 添加约束条件看方程是否有解和唯一解，依次类推，直到方程有唯一解。

2.3 问题一 (3) 的分析

首先，分析圆周上的任意两个角可能出现的所有情况，对出现的情况进行分析，然后建立角度调整模型，无人机可以根据接收到的有偏差的角度，调整到实际位置的真实角度。然后使用计算机，结合第二问对模型进行验证，求出每次调整的角度和调整后的坐标，直到无人机均匀分布在某个圆周上。

2.4 问题二的分析

首先需要建立直角坐标系，然后给出一组有误差的数据，根据问题一的模型和方法，建立锥形无人机的角度自适应调整模型，然后使用计算机对模型进行验证，求出每次调整的角度和调整后的坐标，直到无人机均匀分布在某个锥形上。

三、 模型假设

- 问题一中假设圆的半径已知;
- 无人机初始状态的相对位置不变，不会发生错位，越位;
- 接收信号无人机与其他无人机的夹角与真实值的偏离在合理范围内，在 $(-5\%,5\%)$ 之间;
- 无人机可以调整与其他飞机之间形成的角度。

四、 符号说明

符号	说明	单位
x_m	圆的半径长度	m
(ρ,θ)	某个点的极坐标	(m, 弧度)
$\alpha_1,\alpha_2,\alpha_3,\beta_1,beta_2$	角	弧度
$n_0,n_1,\ldots,n_9$	对应 FY00——FY09 的无人机	-
P_{n_j}	编号为 n_j 的无人机	-
$\angle P_{n_j}P_{n_2}P_{n_0}$	P_{n_j},P_{n_2},P_{n_0} 无人机形成的角	弧度
$((x_{n_j},y_{n_j}))$	编号为 n_j 无人机的直角坐标	(m,m)

五、 模型的建立和求解

5.1 问题一 (1) 模型的建立和求解

假设无人机编号为

$$N=\{0,1,2,3,4,5,6,7,8,9\},$$

对应的无人机编号为 $FY00-FY09$, 而发射信号的无人机编号为

$$N_1=\{n_0,n_1,n_2\},$$

其中 $n_0=0,n_1<n_2$ 且 $N_1\in N$，圆的直径为 X_m，接收信号的无人机为 P_{n_j}, 其极坐标为 (ρ,θ)，设 $\angle P_{n_0}P_{n_j}P_{n_1}=\alpha_1$,$\angle P_{n_0}P_{n_j}P_{n_2}=\alpha_2$,$\angle P_{n_1}P_{n_j}P_{n_2}=\alpha_3$.

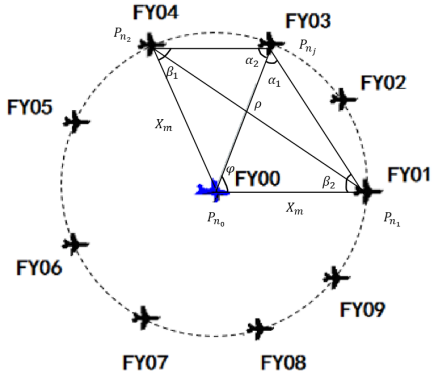

图 1　圆心飞机在中间的情况

①当 $\alpha_3>\alpha_1$ 且 $\alpha_3>\alpha_2$ 时, 如图1:设 $\angle P_{n_j}P_{n_2}P_{n_0}=\beta_1$,$\angle P_{n_0}P_{n_1}P_{n_j}=\beta_2$,$\angle P_{n_1}P_{n_0}Pn_j=\psi$, 根据正弦定理有

$$\frac{X_m}{sm\alpha_2}=\frac{\rho}{sm\beta_1},$$

$$\frac{X_m}{sm\alpha_1} = \frac{\rho}{sm\beta_2},$$

角的关系有

$$\beta_1 + \beta_2 + \alpha_1 + \alpha_2 + f(\theta) = 2\pi,$$

其中 $f(\theta)$ 为圆心角，计算公式为

$$f(\theta) = \begin{cases} \frac{\pi n}{180}(n_2 - n_1), n_1 < n_j < n_2 \\ 2\pi - \frac{\pi n}{180}(n_2 - n_1), n_j > n_2 \end{cases}$$

上述四个方程中有三个未知数，可以解出 ρ，β_1，β_2. 在此基础上很容易求得 θ 的值

$$\alpha_1 + \beta_2 + \psi = \pi$$

$$\theta = \frac{2}{9}\pi \times n_1 + \psi$$

由以上方程即可解出 ρ，θ 的值。

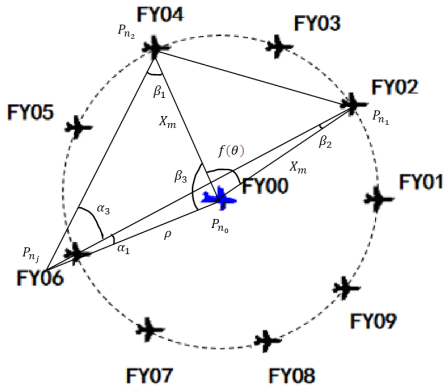

图 2　圆心无人机在右侧的情况

②当 $\alpha_2 > \alpha_1$ 且 $\alpha_2 > \alpha_3$ 时, 如图2: 设 $\angle P_{n_j}P_{n_2}P_{n_0} = \beta_1, \angle P_{n_0}P_{n_1}P_{n_j} = \beta_2, \angle P_{n_1}P_{n_0}P_{n_j} = \psi$, 根据正弦定理有

$$\frac{X_m}{sm(\alpha_1 + \alpha_3)} = \frac{\rho}{sm\beta_2}$$

$$\frac{X_m}{sm\alpha_1} = \frac{\rho}{sm\beta_1}$$

角的关系有

$$\beta_1 + \beta_3 + \alpha_1 + \alpha_3 = \pi$$

$$\alpha_1 + \beta_3 + f(\theta) + \beta_2 = \pi$$

其中 $f(\theta)$ 为圆心角，计算公式为

$$f(\theta) = \begin{cases} \frac{\pi n}{180}(n_2 - n_1), n_1 < n_j < n_2 \\ 2\pi - \frac{\pi n}{180}(n_2 - n_1), n_j > n_2 \end{cases}$$

上述五个方程中有三个未知数，可以解出 ρ，β_1，β_2. 在此基础上很容易求得 θ 的值

$$\alpha_1 + \beta_2 + \psi = \pi$$

$$\theta = \frac{2}{9}\pi \times n_1 + \psi$$

由以上方程即可解出 ρ，θ 的值。

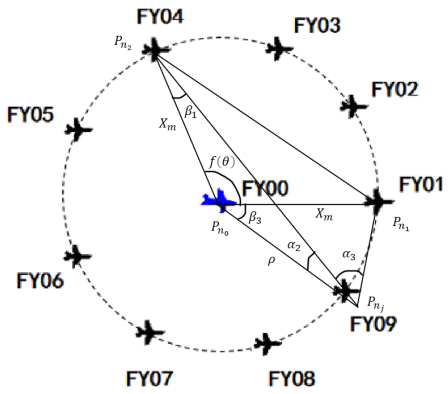

图 3　圆心无人机在左侧的情况

③当 $\alpha_1 > \alpha_2$ 且 $\alpha_1 > \alpha_3$ 时, 如图3: 设 $\angle P_{n_j}P_{n_2}P_{n_0} = \beta_1, \angle P_{n_0}P_{n_1}P_{n_j} = \beta_2, \angle P_{n_1}P_{n_0}P_{n_j} = \beta_3$, 根据正弦定理有

$$\frac{X_m}{sm(\alpha_2 + \alpha_3)} = \frac{\rho}{sm\beta_2}$$

$$\frac{X_m}{sm\alpha_2} = \frac{\rho}{sm\beta_1}$$

角的关系有

$$\beta_3 + \beta_1 + f(\theta) + \alpha_2 = \pi$$

$$\beta_2 + \beta_3 + \alpha_1 + \alpha_3 = \pi$$

$$f(\theta) = \begin{cases} \frac{\pi n}{180}(n_2 - n_1), n_1 < n_j < n_2 \\ 2\pi - \frac{\pi n}{180}(n_2 - n_1), n_j > n_2 \end{cases}$$

上述五个方程中有四个未知数，可以解出 ρ，β_1，β_2，β_3. 在此基础上很容易求得 θ 的值

$$\alpha_1 + \alpha_3 + \beta_2 + \beta_3 = \psi$$

$$\theta = \frac{2}{9}\pi \times n_1 - \psi$$

由以上方程即可解出 ρ，θ 的值。

5.2 问题一 (2) 模型的建立和求解

若发射信号的无人机编号为

$$\{n_0, n_1, n_2\},$$

其中接收信号的无人机为 P_{n_j}, 假设 α_1, α_2 是三架飞机调整后夹角较小的两个角, 现假设 $P_{n_0}, P_{n_1}, P_{n_j}$ 的坐标已知,P_{n_j} 坐标为 (x_{n_j}, y_{n_j})

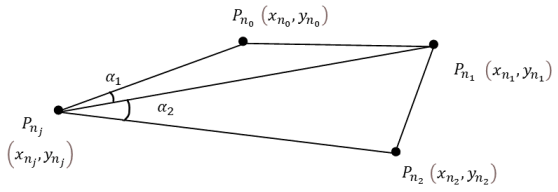

图 4　两个夹角求一点

对于 α_1 而言,

$$\cos\alpha_1 = \frac{(x_{n_j}-x_{n_0})^2+(x_{n_j}-x_{n_1})^2+(y_{n_j}-y_{n_0})^2+(y_{n_j}-y_{n_1})^2-(x_{n_0}-x_{n_1})^2-(y_{n_0}-y_{n_1})^2}{2\sqrt{(x_{n_j}-x_{n_0})^2+(y_{n_j}-y_{n_0})^2}\times\sqrt{(x_{n_j}-x_{n_1})^2+(y_{n_j}-y_{n_1})^2}}$$

同样地，对于 α_2 而言,

$$\cos\alpha = \frac{(x_{n_j}-x_{n_1})^2+(x_{n_j}-x_{n_2})^2+(y_{n_j}-y_{n_1})^2+(y_{n_j}-y_{n_2})^2-(x_{n_1}-x_{n_2})^2-(y_{n_1}-y_{n_2})^2}{2\sqrt{(x_{n_j}-x_{n_1})^2+(y_{n_j}-y_{n_1})^2}\times\sqrt{(x_{n_j}-x_{n_2})^2+(y_{n_j}-y_{n_2})^2}}$$

此时含有两个方程和两个未知数, 存在解, 但等式右端含有平方项, 故存在多解。当无人机偏离较小时,3 架无人机可有效定位。

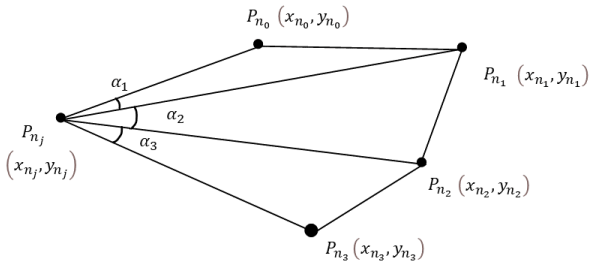

图 5　三个夹角求一点

未得出唯一解，在三架无人机的基础上, 再加一架无人机 n_k, 设四架无人机较小的

3 个角为 $\alpha_1, \alpha_2, \alpha_3$, 则

$$\cos\alpha_3 = \frac{(x_{n_j}-x_{n_2})^2+(x_{n_j}-x_{n_k})^2+(y_{n_j}-y_{n_2})^2+(y_{n_j}-y_{n_k})^2-(x_{n_1}-x_{n_2})^2-(y_{n_1}-y_{n_2})^2}{2\sqrt{(x_{n_j}-x_{n_2})^2+(x_{n_j}-y_{n_2})^2}\times\sqrt{(x_{n_j}-x_{n_k})^2+(y_{n_j}-y_{n_k})^2}}$$

由上面的 3 个式子可得 (x_j, y_j) 的唯一解, 因此除了 $FY00, FY01$ 至少还需要两架无人机才能实现接收信号无人机的有效定位.

5.3 问题一 (3) 模型的建立和求解

5.3.1 角度分析

圆具有很好的几何性质，现假设所有无人机的位置都没有偏差，首先对圆心无人机和圆上一架无人机发射信号，其余无人机接收信号的角度进行分析。

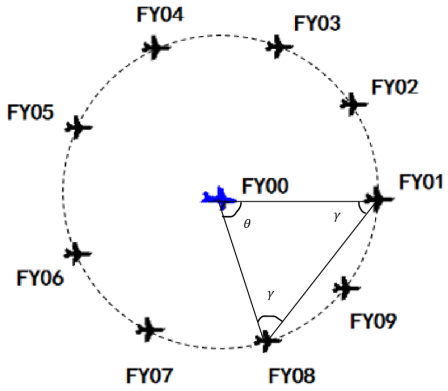

图 6　圆上一架无人机的情形

假设发射信号的无人机编号为 $N_1 = \{n_0, n_1, n_2\}$, 其中 $n_0 = 0, n_1 < n_2$，圆外两架无人机的夹角为 θ, 此时

$$\alpha = \frac{\pi n}{180}(n_2 - n_1),$$

其中 n 代表圆周上的无人机数量，根据等腰三角形的性质，得

$$\gamma = \frac{\pi - \theta}{2},$$

由此可以得夹角的集合为

$$\left[10,\quad 30,\quad 50,\quad 70\right]$$

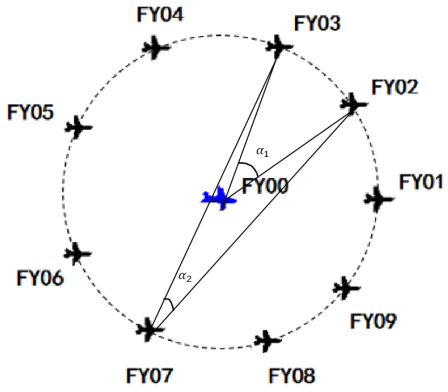

图 7　圆上两架无人机的情形

然后分析圆上有两架无人机的情形，圆上有两架无人机时，包含了刚才讨论的情形，因此只需要讨论圆上任意三架无人机的角度即可，发射信号的无人机编号为 $N_1 = \{n_0, n_1, n_2\}$, 其中 $n_0 = 0, n_1 < n_2$，圆外两架无人机的夹角为 α_1, 此时

$$\alpha_1 = \frac{\pi n}{180},$$

由于同一段弧长对应的圆周角是圆心角的一半，因此

$$\alpha_2 = \frac{\alpha_1}{2},$$

由此可以得 α_2 的集合为

$$\left[10,\ \ 20,\ \ 30,\ \ 40,\ \ 50,\ \ 60,\ \ 70,\ \ 80\right]$$

易知，当发射信号的无人机更多时，和圆上有两架无人机的情形是一样的。通过分析发现，接收信号的无人机与圆上任意两架无人机的夹角是固定的，因此，当实际夹角的误差在 $(-5\%, 5\%)$ 之间时，可以使用 k-means 聚类方法调整无人机的角度。

5.3.2 基于 K-means 聚类的无人机编队角度自适应调整模型

k-means 算法属于聚类算法，是一种典型的无监督学习算法。“k”表示聚类的个数，“means”意味着算法是以平均值来寻找聚类的中心的。

k-means 算法的主要思想和原理为，首先从数据中随机找到 k 个中心，按照点到中心的距离大小，将其归为其中一类，接着每一次聚类完成后重新计算各个类的几何中心，然后重新归类，如此循环，直到最终所有类的中心不再改变为止。此时所有的数据都成功聚类。

建立了 K-mean 聚类方法之后，就可以调整接收信号的无人机的角度，建立模型之后使用计算机仿真，仿真过程中需要使用问题一 (2) 中求解调整后坐标的模型，已经证明了三个角可以确定唯一的解，但是由于误差的累积和计算机表示数值的误差，三架无人机时计算机难于求解，为了验证模型，我们采用两架无人机的模型求解。

5.3.3 模型结果与检验

通过计算机仿真的结果如下:

	第一次调整						第二次调整						
编号	初始极坐标	角度1	调整后角度1	角度2	调整后角度2	调整后极坐标	角度1	调整后角度1	角度2	调整后角度2	调整后极坐标	均值	误差百分比
0	[0,0]	*	*	*	*	[0,0]	*	*	*	*	[0,0]	*	*
1	[100,0]	30.94	30	30.84	30	[105.51,359.91]	*	*	*	*	[105.51,359.91]	105.8	-0.25%
2	[98.0, 40.1]	52.54	50	10.33	10	[105.43,40.03]	*	*	*	*	[105.43,40.03]	105.8	-0.06%
3	[112.0,80.21]	55.36	60	9.74	10	[105.23,80.09]	50	50	20.1	20	[105.51, 79.91]	105.8	0.16%
4	105,119.75]	*	*	*	*	[105.0, 119.75]	30.16	30	20.1	20	[105.71, 119.85]	105.8	0.30%
5	[98.0,159.86]	75.34	80	52.24	50	[96.55,154.85]	13.09	10	21.1	20	[105.94, 159.87]	105.8	0.30%
6	[112,199.96]	47.7	50	64.9	60	[115.49,192.34]	5.93	10	13.2	10	[106.09, 199.97]	105.8	0.16%
7	[105,240.7]	*	*	*	*	[105.0, 240.07]	20.12	20	10	10	[106.09, 240.09]	105.8	-0.06%
8	[98,280.17]	10.13	10	65.22	70	[91.4,281.06]	22.63	20	32.9	30	[105.94, 280.19]	105.8	-0.25%
9	[112,320.28]	9.93	10	47.7	50	[105.43,319.79]	20.12	20	49.9	50	[105.71, 320.21]	105.8	-0.32%

图 8　调整过程

从结果可以看出,仅需要两次调整即可,调整后的无人机均匀分布在半径为 $105.77m$ 的圆周上，并且最大误差为 -0.32%, 平均误差为 4.476×10^{-17}, 可以认为无人机均匀分布在圆周两侧。

5.4 问题二模型的建立和求解

首先建立以编号为 $FY13$ 的无人机的直角坐标系，无人机之间的距离设置为 $50m$, 求出每一架无人机的坐标，经过分析，当选取锥形内部三架无人机发射信号时，飞机间的角度只有三种情况

$$\left[0,\quad 19.11,\quad 30\right],$$

因此假设编号为 $FY05, FY08, FY09$ 的飞机位置是准确无误的，然后对其它接收信号的无人机加以 5% 的噪声，建立基于 K-means 聚类的无人机编队角度自适应调整模型，计算机的仿真结果如下:

编号	飞机编号	初始直角坐标	初始极坐标	调整前角度1	调整后角度1	调整前角度2
0	fy15	[1.9 -97.8]	[97.82, 271.11]	11	19.11	11.27
1	fy14	[-2.5 -52.2]	[52.26, 267.26]	28.62	30	0.34
2	fy13	[-2.2 1.4]	[2.61, 147.53]	28.32	30	29.22
3	fy12	[-1.6 49.4]	[49.43, 91.86]	0.73	0	29.64
4	fy11	[-0.4 101.1]	[101.1, 90.23]	10.75	19.11	10.85
5	fy10	[43.1 -76.9]	[88.16, 299.27]	0.11	0	29.28
6	fy9	[43.3 -25]	*	*	*	*
7	fy8	[43.3 25]	*	*	*	*
8	fy7	[44.6 76.4]	[88.47, 59.72]	0.71	0	29.53
9	fy6	[86.4 -47.9]	[98.79, 331.0]	31.42	30	30.83
10	fy5	[86.6 0]	*	*	*	*
11	fy4	[87.8 50.]	[101.04, 29.66]	29.99	30	29.31
12	fy3	[128.7 -26.7]	[131.44, 348.28]	30.05	30	1.19
13	fy2	[128.3 24.5]	[130.62, 10.81]	30.55	30	0.22
14	fy1	[172.31 -2.]	[172.32, 359.33]	10.48	19.11	11.45

图 9　调整过程

通过模型易知，仅调整一次即可将接收信号的无人机调整到正确位置上。

5.4.1 灵敏性分析

问题一 (2) 中已经证明了圆上三架无人机定位时可以确定唯一的解，但是由于误差的累积和计算机表示数值的误差，圆上三架无人机定位时计算机难于求解，因此圆上三架无人机定位时不受误差的影响，而圆上两架无人机定位时对飞机偏离的程度敏感，因此我们分析锥形编队队形下对误差的敏感度，使用计算机得出以下结果：

噪音范围	误差范围	调整次数
[-0.5, 0.5]	[-1%，1%]	1
[-1, 1]	[-2%，2%]	1
[-1.5, 1.5]	[-3%，3%]	1
[-2.0, 2.0]	[-4%，4%]	1
[-2.5, 2.5]	[-5%，5%]	1
[-3, 3]	[-6%，6%]	计算机无解
[-3.5, 3.5]	[-7%，7%]	1
[-4.0, 4.0]	[-8%，8%]	1
[-4.5, 4.5]	[-9%，9%]	计算机无解
[-5, 5]	[-10%，10%]	计算机无解
[-5.5, 5.5]	[-11%，11%]	计算机无解
[-6, 6]	[-12%，12%]	计算机无解

图 10　模型对误差的敏感度

从结果可以看出，当误差小于 5%, 模型具有良好的性能，当误差超过 5% 时，模型鲁棒性下降，超过 10%, 时模型性能大幅下降。

六、 模型的评价

6.1 模型的优点

- 模型简单，无人机调整次数少;
- 无人机仅仅只需要角度就可以调整自身位置;
- 发射信号的无人机数量少。

6.2 模型的缺点

- 模型依赖于初值的影响较大;
- 使用圆上两架无人机定位时要求飞机的误差在 $(-5\%, 5\%)$;

6.3 模型的改进与推广

- 改进三架无人机时计算机的求解方法，消除对无人机偏离造成的误差影响；
- 模型开始时随机选定无人机发射信号。减少模型开始时指定发射信号无人机的人为干扰。

参考文献

[1] 杨盛庆. 基于几何力学与最优控制的无人机编队方法研究 [D]. 北京理工大学,2014.

附录 A　文件列表

表 1　Add caption

文件名	文件描述
圆形编队求解.py	圆形编队求解模型代码
锥形编队求解.py	锥形编队求解模型代码
圆形编队求解数据.xlsx	圆形编队求解数据
锥形编队求解数据.xlsx	锥形编队求解数据
锥形模型敏感度分析结果.xlsx	锥形模型敏感度分析结果

附录 B　代码

圆形编队求解.py

```python
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt

#极坐标->直角坐标
def to_xy(l,a):
    a = a * math.pi /180
    return (l * math.cos(a),l * math.sin(a))

#直角坐标->极坐标
def to_pa(pj):
    pt,at= 0,0

    if(pj[0] >= 0 and pj[1] >= 0):
        pt,at= (pj[0] ** 2 + pj[1] ** 2)**(1/2),math.atan(pj[1]/pj[0])/math.pi
    *180.0

    if(pj[0] < 0 and pj[1] > 0):
        pt,at= (pj[0] ** 2 + pj[1] ** 2)**(1/2), math.atan(pj[1]/pj[0])/math.pi
    *180.0 + 180

    if(pj[0] <= 0 and pj[1] <= 0):
        pt,at= (pj[0] ** 2 + pj[1] ** 2)**(1/2), math.atan(pj[1]/pj[0])/math.pi
    *180.0 + 180

    if(pj[0] > 0 and pj[1] < 0):
```

```
        pt,at= (pj[0] ** 2 + pj[1] ** 2)**(1/2), math.atan(pj[1]/pj[0])/math.pi*180.0 + 360
    return pt,at

#用直角坐标系求两点的距离
def distance(p1,p2):
    return np.sqrt((p1[0]-p2[0])**2+(p1[1]-p2[1])**2)

#用直角坐标系求夹角
def angle(p1, p2, p3):
    a=math.sqrt((p2[0]-p3[0])*(p2[0]-p3[0])+(p2[1]-p3[1])*(p2[1] - p3[1]))
    b=math.sqrt((p1[0]-p3[0])*(p1[0]-p3[0])+(p1[1]-p3[1])*(p1[1] - p3[1]))
    c=math.sqrt((p1[0]-p2[0])*(p1[0]-p2[0])+(p1[1]-p2[1])*(p1[1]-p2[1]))
    A=math.degrees(math.acos((a*a-b*b-c*c)/(-2*b*c)))
    return A

#是否在一个圆周上
def is_circle():
    return (location[:,0] == 100).all() and (location[:,1] % 40 ==0).all()

#原始的极坐标
location_angle =  np.array([[0,0],[100,0],[98,40.10],[112,80.21],[105,119.75],[98,159.86],[112,199.96],[105,240.07],[98

real_angle  =     np.array([[0,0],[100,0],[100,40.],[100,80],[100,120.],[100,160],[100,200],[100,240.0],[100,280.],[100

#装换为直角坐标系
location =  np.array([[0,0],[100,0],[98,40.10],[112,80.21],[105,119.75],[98,159.86],[112,199.96],[105,240.07],[98

real  =     np.array([[0,0],[100,0],[100,40.],[100,80],[100,120.],[100,160],[100,200],[100,240.0],[100,280.],[100



for i in range(len(location)):
    location[i] = to_xy(location[i,0],location[i,1])


for i in range(len(real)):
    real[i] = to_xy(real[i,0],real[i,1])


def scatter():
    theta = []
```

```
    for i in [0,40,80,120,160,200,240,280,320,360]:
        theta.append(math.radians(i))
    r = [100, 100, 100, 100, 100, 100, 100, 100, 100, 100]
    plt.figure(figsize=(20,20),dpi=80)

    ax = plt.subplot(111,projection='polar')      #极坐标

    loc_l = location_angle[:,0]
    loc_a = location_angle[:,1]

    theta2 = []
    for i in loc_a:
        theta2.append(math.radians(i))

    ax.plot(theta, r, '+')
    ax.scatter(theta2,loc_l,color='orange')
    plt.show()



#求1架无人机发射信号的所有可能的角度
angle_1 = set()
for i in range(len(real)):
    for j in range(len(real)):
        if(i==j or i==0 or j==0):
            continue
        angle_1.add(round(angle(real[j],real[i],real[0]),0))

#求2架无人机发射信号的所有可能的角度
angle_2 = set()
for i in range(len(real)):
    for j in range(len(real)):
        for k in range(len(real)):
            if(i==j or i==k or j==k or i==0 or j==0 or k==0):
                continue
            '''
            angle_2.add(round(angle(real[k],real[j],real[i]),2))
            angle_2.add(round(angle(real[k],real[j],real[0]),2))
            angle_2.add(round(angle(real[k],real[i],real[0]),2))
            '''
            angle_2.add(min(round(angle(real[k],real[j],real[i]),2),round(angle(
    real[k],real[j],real[0]),2)))
            angle_2.add(min(round(angle(real[k],real[i],real[0]),2),round(angle(
    real[k],real[j],real[0]),2)))


#求3架无人机发射信号的所有可能的角度
```

```
angle_3 = set()
for i in range(len(real)):
    for j in range(len(real)):
        for k in range(len(real)):
            for l in range(len(real)):
                if(i==j or i==k or i==l or j==k or j==l or k==l or i==0 or j==0 or
    k==0 or l==0):
                    continue
                a=round(angle(real[l],real[k],real[i]),2)
                b=round(angle(real[l],real[k],real[j]),2)
                c=round(angle(real[l],real[k],real[0]),2)
                d=round(angle(real[l],real[j],real[i]),2)
                e=round(angle(real[l],real[j],real[0]),2)
                f=round(angle(real[l],real[i],real[0]),2)
                l = sorted([a,b,c,d,e])



from sympy import symbols,nsolve
x, y = symbols('x, y')
lst = []

def adjust(pj,p1,p2,p3):
    p,a = to_pa(pj)
    b1 = angle(pj,p1,p2)
    b2 = angle(pj,p1,p3)
    b3 = angle(pj,p2,p3)

    if(max(b1,b2,b3) == b3 ):
        a1 = b1
        a1_opposite = distance(p1,p2)
        a2 = b2
        a2_opposite = distance(p1,p3)
        p = distance(pj,p1)
        l_j1 = ((x-p1[0])**2+(y-p1[1])**2)**(1/2)
        l_j2 = ((x-p2[0])**2+(y-p2[1])**2)**(1/2)
        l_j3 = ((x-p3[0])**2+(y-p3[1])**2)**(1/2)


    if(max(b1,b2,b3) == b2 ):
        a1 = b1
        a1_opposite = distance(p1,p2)
        a2 = b3
        a2_opposite = distance(p2,p3)
        p = distance(pj,p2)
        l_j1 = ((x-p2[0])**2+(y-p2[1])**2)**(1/2)
        l_j2 = ((x-p1[0])**2+(y-p1[1])**2)**(1/2)
```

```
        l_j3 = ((x-p3[0])**2+(y-p3[1])**2)**(1/2)

    if(max(b1,b2,b3) == b1 ):
        a1 = b3
        a1_opposite = distance(p3,p2)
        a2 = b2
        a2_opposite = distance(p1,p3)
        p = distance(pj,p3)
        l_j1 = ((x-p3[0])**2+(y-p3[1])**2)**(1/2)
        l_j2 = ((x-p2[0])**2+(y-p2[1])**2)**(1/2)
        l_j3 = ((x-p1[0])**2+(y-p1[1])**2)**(1/2)

    if(a1 % 10 > 5):
        diff_a1 =  (a1 % 10 ) - 10
        a1_just = a1 -  diff_a1
    else:
        diff_a1 =  (a1 % 10 )
        a1_just = a1 -  diff_a1

    if(a2 % 10 > 5):
        diff_a2 = (a2 % 10 ) - 10
        a2_just = a2 -  diff_a2
    else:
        diff_a2 =  (a2 % 10 )
        a2_just = a2 -  diff_a2

    cos_b1 = math.cos(a1_just/180*math.pi)
    cos_b2 = math.cos(a2_just/180*math.pi)

    initial = 104
    aa = nsolve([((l_j2**2+l_j1**2-a1_opposite**2)/(2*l_j2*l_j1))-cos_b1,
                 ((l_j1**2+l_j3**2-a2_opposite**2)/(2*l_j1*l_j3)-cos_b2)],
                [x, y],[initial * math.cos(a /180 * math.pi) ,initial * math.sin(a
    /180 * math.pi)])

    print("初始直角坐标: ",tuple(pj.round(2)),"初始极坐标: ",tuple([round(i,2) for
    i in to_pa(pj)]),
        "调整前的角度1: ",round(a1,2),"需要调整的度数: ",round(diff_a1,2),"调整后的
    角度1: ",round(a1_just,2),
        "调整前的角度2: ",round(a2,2),"需要调整的度数: ",round(diff_a2,2),"调整后的
    角度2: ",round(a2_just,2),
        "调整后的极坐标: ",tuple([round(i,2) for i in to_pa(aa)]))
    lst.append([pj.round(2),[round(i,2) for i in to_pa(pj)],round(a1,2),round(
    diff_a1,2),
        round(a1_just,2),round(a2,2),round(diff_a2,2),round(a2_just,2),[round(i,2)
    for i in to_pa(aa)]])
    return aa
```

```
195
196
197 print("第一次调整")
198 for i in range(len(location)):
199     if(i==0 or i==4 or i==7):
200         continue
201     print("编号",i)
202     location[i,0],location[i,1]=adjust(location[i],location[0], location[4],
    location[7])
203 lst = []
204
205 '''
206 for i in [5,6,8]:
207     for j in [1,2,3,4,7,9]:
208         for k in [1,2,3,4,7,9]:
209             if(i==j or i==k or k==j or i==0):
210                 continue
211             print(i,j,k)
212             _=adjust(location[i],location[0], location[j], location[k])
213 '''
214 print()
215 print("第二次调整")
216 for i in range(len(location)):
217     if(i==0 or i==1 or i==2):
218         continue
219     print("编号",i)
220     location[i,0],location[i,1]=adjust(location[i],location[0], location[1],
    location[2])
221
222 df = pd.DataFrame(lst)
223 df.columns=['初始直角坐标','初始极坐标','调整前的角度1','需要调整的度数',
224 '调整后的角度1','调整前的角度2','需要调整的度数','调整后的角度2','调整后的极坐标']
225 print(df)
```

锥形编队求解.py

```
1  import numpy as np
2  import pandas as pd
3  import math
4  import matplotlib.pyplot as plt
5
6  #极坐标->直角坐标
7  def to_xy(l,a):
8      a = a * math.pi /180
9      return (l * math.cos(a),l * math.sin(a))
10
11 #直角坐标->极坐标
12 def to_pa(pj):
```

```
    pt,at= 0,0

    if(pj[0]==0 and pj[1]==0):
        pa,at=0,0

    if(pj[0] > 0 and pj[1] > 0):
        pt,at= (pj[0] ** 2 + pj[1] ** 2)**(1/2),math.atan(pj[1]/pj[0])/math.pi
    *180.0

    if(pj[0] < 0 and pj[1] > 0):
        pt,at= (pj[0] ** 2 + pj[1] ** 2)**(1/2), math.atan(pj[1]/pj[0])/math.pi
    *180.0 + 180

    if(pj[0] < 0 and pj[1] < 0):
        pt,at= (pj[0] ** 2 + pj[1] ** 2)**(1/2), math.atan(pj[1]/pj[0])/math.pi
    *180.0 + 180

    if(pj[0] > 0 and pj[1] < 0):
        pt,at= (pj[0] ** 2 + pj[1] ** 2)**(1/2), math.atan(pj[1]/pj[0])/math.pi
    *180.0 + 360
    return pt,at

#用直角坐标系求两点的距离
def distance(p1,p2):
    return np.sqrt((p1[0]-p2[0])**2+(p1[1]-p2[1])**2)

#用直角坐标系求夹角
def angle(p1, p2, p3):
    a=math.sqrt((p2[0]-p3[0])*(p2[0]-p3[0])+(p2[1]-p3[1])*(p2[1] - p3[1]))
    b=math.sqrt((p1[0]-p3[0])*(p1[0]-p3[0])+(p1[1]-p3[1])*(p1[1] - p3[1]))
    c=math.sqrt((p1[0]-p2[0])*(p1[0]-p2[0])+(p1[1]-p2[1])*(p1[1]-p2[1]))
    v=math.acos((a*a-b*b-c*c)/(-2*b*c))
    A=math.degrees(v)
    return A



#直角坐标系
q=math.sqrt(3)
location =   np.array([[0,-100],[0,-50],[0,0],[0,50],[0,100],[25*q,-75],[25*q
    ,-25],[25*q,25],[25*q,75],[50*q,-50],[50*q,0],[50*q,50],[75*q,-25],[75*q
    ,25],[100*q,0]])
real  =     np.array([[0,-100],[0,-50],[0,0],[0,50],[0,100],[25*q,-75],[25*q
    ,-25],[25*q,25],[25*q,75],[50*q,-50],[50*q,0],[50*q,50],[75*q,-25],[75*q
    ,25],[100*q,0]])
real_zhui_angle = real
np.random.seed(0)
```

```
zaoyin =  np.random.randint(low= -60 ,high= 60,size=(15,2)) * 0.1
location = location + zaoyin
location[[6,7,10]]=real[[6,7,10]]
print(location)


from sympy import symbols,nsolve
x, y = symbols('x, y')
lst=[]

def adjust(pj,p1,p2,p3):
    p,a = to_pa(pj)
    b1 = angle(pj,p1,p2)
    b2 = angle(pj,p1,p3)
    b3 = angle(pj,p2,p3)

    if(max(b1,b2,b3) == b3 ):
        a1 = b1
        a1_opposite = distance(p1,p2)
        a2 = b2
        a2_opposite = distance(p1,p3)
        p = distance(pj,p1)
        l_j1 = ((x-p1[0])**2+(y-p1[1])**2)**(1/2)
        l_j2 = ((x-p2[0])**2+(y-p2[1])**2)**(1/2)
        l_j3 = ((x-p3[0])**2+(y-p3[1])**2)**(1/2)


    if(max(b1,b2,b3) == b2 ):
        a1 = b1
        a1_opposite = distance(p1,p2)
        a2 = b3
        a2_opposite = distance(p2,p3)
        p = distance(pj,p2)
        l_j1 = ((x-p2[0])**2+(y-p2[1])**2)**(1/2)
        l_j2 = ((x-p1[0])**2+(y-p1[1])**2)**(1/2)
        l_j3 = ((x-p3[0])**2+(y-p3[1])**2)**(1/2)

    if(max(b1,b2,b3) == b1 ):
        a1 = b3
        a1_opposite = distance(p3,p2)
        a2 = b2
        a2_opposite = distance(p1,p3)
        p = distance(pj,p3)
        l_j1 = ((x-p3[0])**2+(y-p3[1])**2)**(1/2)
        l_j2 = ((x-p2[0])**2+(y-p2[1])**2)**(1/2)
        l_j3 = ((x-p1[0])**2+(y-p1[1])**2)**(1/2)

```

```
if(a1< 20):

    if(a1 < 10):
        a1_just = 0
        diff_a1 = a1
    else:
        a1_just = 19.10660535086906
        diff_a1 =19.10660535086906- a1
if(a2< 20):
    if(a2 < 10):
        a2_just = 0
        diff_a2 = a2
    else:
        a2_just = 19.10660535086906
        diff_a2 =19.10660535086906 - a2

if(a1>=20):
    if(a1 % 10 > 5):
        diff_a1 =  (a1 % 10 ) - 10
        a1_just = a1 -  diff_a1
    else:
        diff_a1 =  (a1 % 10 )
        a1_just = a1 -  diff_a1
if(a2>=20):
    if(a2 % 10 > 5):
        diff_a2 = (a2 % 10 ) - 10
        a2_just = a2 -  diff_a2
    else:
        diff_a2 =  (a2 % 10 )
        a2_just = a2 -  diff_a2

cos_b1 = math.cos(a1_just/180*math.pi)
cos_b2 = math.cos(a2_just/180*math.pi)

initial = 104
aa = nsolve([((l_j2**2+l_j1**2-a1_opposite**2)/(2*l_j2*l_j1))-cos_b1,
            ((l_j1**2+l_j3**2-a2_opposite**2)/(2*l_j1*l_j3)-cos_b2)],
           [x, y],(pj[0] ,pj[1]))

print("初始直角坐标：",tuple(pj.round(2)),"初始极坐标：",tuple([round(i,2) for
i in to_pa(pj)]),
    "调整前的角度1：",round(a1,2),"需要调整的度数：",round(diff_a1,2),"调整后的
角度1：",round(a1_just,2),
    "调整前的角度2：",round(a2,2),"需要调整的度数：",round(diff_a2,2),"调整后的
角度2：",round(a2_just,2),
    "调整后的极坐标：",tuple([round(i,2) for i in to_pa(aa)]))

```

```
    lst.append([pj.round(2),[round(i,2) for i in to_pa(pj)],round(a1,2),round(diff_a1,2),
        round(a1_just,2),round(a2,2),round(diff_a2,2),round(a2_just,2),[round(i,2) for i in to_pa(aa)]])
    return aa

print(angle(real[6],real[7],real[4]))

for i in range(len(location)):
    print(i)
    if(i==6 or i==7 or i==10):
        print(location[i,0],location[i,1])
    else:
        location[i,0],location[i,1]=adjust(location[i],location[6], location[7], location[10])

'''
for i in [5,6,8]:
    for j in [1,2,3,4,7,9]:
        for k in [1,2,3,4,7,9]:
            if(i==j or i==k or k==j or i==0):
                continue
            print(i,j,k)
            _=adjust(location[i],location[0], location[j], location[k])


for i in range(len(location)):
    if(i==0 or i==1 or i==2):
        continue
    print(i)
    location[i,0],location[i,1]=adjust(location[i],location[0], location[1], location[2])
'''


df = pd.DataFrame(lst)
df.columns=['初始直角坐标','初始极坐标','调整前的角度1','需要调整的度数',
'调整后的角度1','调整前的角度2','需要调整的度数','调整后的角度2','调整后的极坐标']
print(df)
```