# PaperPass[免费版]查重报告

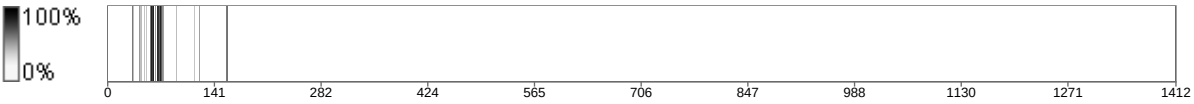
## 简明打印版

**查重结果(相似度):**

总　体：　7%
本地库：　7%　(本地库包含期刊库、学位库、会议库、联合库)
• 期刊库：　6%　(期刊库相似度是指论文与学术期刊库的比对结果)
• 学位库：　1%　(学位库相似度是指论文与学位论文库的比对结果)
• 会议库：　0%　(会议库相似度是指论文与会议论文库的比对结果)
• 联合库：　1%　(联合库相似度是指论文与大学生联合比对库的比对结果)
• 图书库：(免费版不检测图书库)
• 专利库：(免费版不检测专利库)
• 报纸库：(免费版不检测报纸库)
• 外文库：(免费版不检测外文库)
互联网：(免费版不检测互联网资源)

检测版本：免费版(仅检测中文)
报告编号：667900E0372E6X2B8
论文题目：洪晨辉_3220101111_无人机寻物研究
论文作者：佚名
论文字数：24151
段落个数：1052
句子个数：1412
提交时间：2024-6-24 13:15:12
比对范围：期刊库、硕博学位库、会议库、大学生联合比对库
查询真伪：https://www.paperpass.com/check

**句子相似度分布图:**




**本地库相似资源列表(期刊库、硕博学位库、会议库、大学生联合比对库):**

1.相似度：5.3% 篇名：《基于信息素决策的无人机集群协同搜索算法》
　来源：学术期刊 2021年
2.相似度：0.6%
　来源：大学生联合比对库

**互联网相似资源列表:**

免费版不检测互联网资源库

浙江大学


# 本科实验报告

姓名： 洪晨辉

学院： 控制科学与工程学院

系： 控制科学与工程学系

专业： 自动化（控制）

学号： 3220101111

指导教师： 林峰

2024 年 5 月 28 日

# 浙江大学实验报告

课程名称： 控制与决策系统仿真 实验类型： 探究型

实验项目名称： 控制与决策系统在集群无人机寻物方面的应用

学生姓名：洪晨辉 专业：自动化（控制）学号：3220101111

指导老师： 林峰

装订线

## 一、实验目的和要求

1. 探究集群无人机系统路径规划方法
2. 学习 Matlab 底层使用，培养基础代码能力
3. 玩玩仿真

## 二、实验内容和原理

### 1. 背景

近几年来，无人机技术取得了显著进步。无人机因其体积小、成本低、机动性强和操作灵活等特点，广泛应用于军事、农业、物流、灾害救援、环境监测等多个领域。特别是在搜索与救援、勘探和巡逻等任务中，无人机展示出了极高的效率和优势。为了进一步提升无人机在这些任务中的表现，无人机轨迹的研究已经成为了一个重要的方向。

在军事应用中，无人机被广泛用于侦察和情报收集，通过优化无人机的飞行轨迹，可以提高任务的成功率和安全性。在农业领域，无人机可以进行精准的农田监测和农药喷洒，优化的飞行轨迹可以减少资源浪费并提高作业效率。在物流方面，无人机可以实现快速的包裹投递，合理的飞行路径规划能够降低成本并缩短运输时间。此外，在灾害救援和环境监测中，无人机能够快速覆盖广泛区域，通过优化轨迹，能更高效地搜集重要数据和进行救援行动。

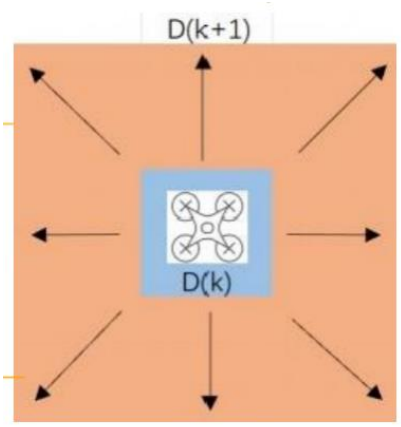
尽管无人机体积小、成本低、安全高效，但受限于体积、负载等原因，单架无人机完成复杂任务较为困难。因此，一般任务需要多架无人机进行协同区域搜索。协同区域搜索是指在满足环境和性能等多个约束条件下，为多架无人机规划搜索路径，并协调无人机之间的关系，确保无人机可以有效地执行区域搜索任务。本实验报告力求在力所能

及的范围内，模拟一种较为有效的协同区域搜索算法。

2. 内容

1．设定两种情境。第一种情况下，目标物的数量与位置均为未知，要求在规定的任务时间内，发现目标物的数量尽可能多。第二种情况下，目标物的位置和权重均为已知，要求在规定的任务时间内，发现目标物的权重之和尽可能大。

2．规定无人机移动规则。限于能力，这里不考虑无人机的动力学约束，假定无人机可以在一个由方格组成的空间内，每过一单位时间移动到周边 3*3-1 方格内的任一格。

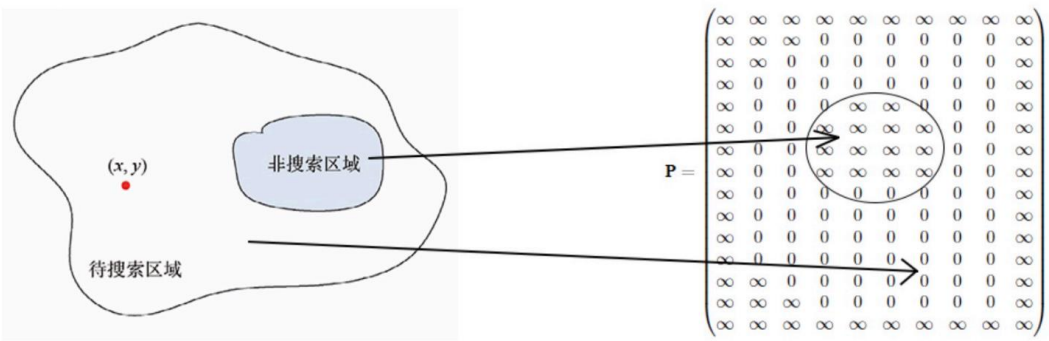



3．寻找集群无人机算法。这里采取较为简单的蚁群算法。蚁群在觅食过程中，可以通过分泌信息素并利用信息素的传播、挥发等特性来引导整个蚁群向有实物的区域快速移动。信息素是定义在栅格中，具有生物信息素的分泌、扩散、挥发等物理特性，并对周围栅格无人机具有吸引和排斥能力的数字量。同样，模仿自然界中蚁群觅食的机理，定义无人机集群的信息素地图。无人机可以在信息素地图上写入新的信息，同时也可以从地图上读取现有的信息。

4．编写代码，实现仿真。

3. 原理

1)环境地图向信息素地图的映射与初始化，如下图所示。



2)无人机集群对地静目标搜索机制

针对地面静止目标，在一定时间内无人机集群搜索的覆盖面积越大则发现目标的可能性越大。因此，用一定时间内的搜索面积覆盖率来表征搜索效率的大小。在不考虑回访、重点搜索等复杂任务场景下，针对搜索地面静止目标的任务场景，定义信息素只有排斥信息素一种，且排斥信息素只具备分泌功能，不具备挥发和扩散功能。因此，网格（i，j）的信息素定义为

$$\begin{cases} p_{i,j} = f_{combine}(p^r_{i,j}, p^a_{i,j}) = p^r_{i,j} \\ p_{i,j}(t+1) = p_{i,j}(t) + g_{deploy}(\mathbf{r}_{pos}(t)) \end{cases}$$

无人机集群搜索地面静止目标时的搜索机制分为 2 个阶段，即单机搜索阶 段、多机通信阶段。在单机搜索阶段，每个无人机是一个独立的个体，相互之间没有通信，各自按照设计的搜索规则和决策规则进行不间断搜索，并按照更新规则更新本机的信息素地图。在多机通信阶段， 无人机集群进行协同，实现信息素地图的融合和更新。通信完成后（但在本问题中，我们不考虑通信所花时间），各无人机按照更新后的信息素地图继续开始搜索。

3）单机搜索阶段无人机集群的信息素地图更新规则

在搜索过程中无人机保持信息素地图的更新， 无人机每搜索一步，信息素地图更新一次。更新规则为：

$$\begin{cases} p_{i,j}(t+1) = p_{i,j}(t) + g_{deploy}(\mathbf{r}_{pos}(t)) \\ g_{deploy} = \begin{cases} K, \mathbf{r}_{pos}(t) \in (i,j) \\ 0, \mathbf{r}_{pos}(t) \notin (i,j) \end{cases} \end{cases}$$

同时为每个无人机定义轨迹矩阵 $M_{trace} = (m_{i,j})(m \times n)$。矩阵共 m ×n 元素，对应信息素矩阵 P 中的 m×n 栅格，其随无人机位置变化同步更新，更新规则如下：

$$\begin{cases} m_{i,j}(t+1) = \begin{cases} 1, \mathbf{r}_{pos}(t) \in (i,j) \\ 0, \mathbf{r}_{pos}(t) \notin (i,j) \end{cases} \\ m_{i,j}(0) = 0; \end{cases}$$

4）决策机制：

无人机依据本机当前时刻的运动状态和信息素地图来做出实时决策，指导无人机下一时刻的运动。无人机根据信息素地图所做出的决策行为用 $f_{behavior}$ 表示，即

$$f_{behavior}:\mathbf{P}(t)\longrightarrow Position(t+1)$$

$$Position(t)=(i,j)$$

$$Position(t+1)=\{(i+a,j+b)|a,b\in\{-1,0,1\}\}$$

采取一种离散化的控制指令，即无人机需要遍历一系列坐标点，而不是遵循连续的速度指令控制，以此降低问题的规模和模型求解的复杂度。提出 2 种决策方式，并将两者混合：一种是局部决策，另一种是全局决策。

局部决策是指无人机仅根据周围栅格的信息素浓度做出决策。在决策前，应先从本机信息素地图 P(t) 中提取局部信息素地图 $P_{local}(t)$，$P_{local}(t)$ 为一个 3 × 3 的矩阵。

$$\mathbf{P}_{local}=\begin{pmatrix}p_{i-1,j-1} & p_{i-1,j} & p_{i-1,j+1}\\ p_{i,j-1} & p_{i,j} & p_{i,j+1}\\ p_{i+1,j-1} & p_{i+1,j} & p_{i+1,j+1}\end{pmatrix}$$

显然，$P_{local}$ 代表栅格 (i, j) 及周围栅格的信息素信息。$P_{local}$ 作为信息素矩阵 P(t) 的子矩阵，代表将当前时刻无人机所在栅格以及其周围 8 个栅格的信息。

为了增大搜索面积覆盖率，下个步长内无人机应该向信息素浓度最小的栅格移动。

$$f_{behavior}(\mathbf{P}_{local}(t))=Position^{*}(t+1)$$

$$Position^{*}(t+1)=min\{p_{i+a,j+b}|a,b\in\{-1,0,1\}\}$$

全局信息决策则要利用整张信息素地图的信息，而为了利用全局信息，同样要定义一个 3 × 3 的矩阵 $P_{global}$。为了得到 $P_{global}$，需要对信息素矩阵 P 进行压缩，意味着 $P_{global}$ 需要包含 P 中所有的信息，得到 $P_{global}$ 的计算步骤如下所示。

输入：$\mathbf{P}, Position(t) \in (i,j)$。
输出：$\mathbf{P}_{global}$。
步骤 1　　定义 $\mathbf{P}' = (p'_{i,j})_{m\times n}$。

$$p'_{i,j} = \begin{cases} 0 & p_{i,j} = +\infty \\ p_{i,j} & p_{i,j} \neq +\infty \end{cases}$$

步骤 2　　把 $\mathbf{P}'$ 压缩成 $3\times 3$ 矩阵 $\mathbf{P}'_{\mathbf{global}}$。

$$\mathbf{P}'_{\mathbf{global}} = \begin{pmatrix} \frac{\sum_{a=1,b=j}^{a=i,b=n} p'_{a,b}}{i(n-j+1)} & \frac{\sum_{a=1,b=j}^{a=m,b=n} p'_{a,b}}{m(n-j+1)} & \frac{\sum_{a=i,b=j}^{a=m,b=n} p'_{a,b}}{(m-i+1)(n-j+1)} \\ \frac{\sum_{a=1,b=1}^{a=i,b=n} p'_{a,b}}{in} & +\infty & \frac{\sum_{a=i,b=1}^{a=m,b=n} p'_{a,b}}{(m-i+1)n} \\ \frac{\sum_{a=1,b=1}^{a=i,b=j} p'_{a,b}}{ij} & \frac{\sum_{a=1,b=1}^{a=i,b=j} p'_{a,b}}{mj} & \frac{\sum_{a=i,b=1}^{a=m,b=n} p'_{a,b}}{(m-i+1)j} \end{pmatrix}$$

步骤 3　　$\mathbf{P}'_{local} \leftarrow \mathbf{P}_{local}$。把 $\mathbf{P}'_{local}$ 中不等于 $+\infty$ 的元素用 0 代替。
步骤 4　　得到 $\mathbf{P}_{global}$。

$$\mathbf{P}_{global} = \mathbf{P}'_{local} + \mathbf{P}'_{global}$$

查重 56%
在得到 2 个 3×3 的矩阵 Pglobal 和 Pglobal 之后，根据这 2 个矩阵的值来生成无人机的控制指令。移动规则为：无人机将优先搜索 Pglobal 中信息素浓度最小的栅格，如果栅格（i，j）周围的 8 个栅格中有不少于 2 个从未被搜索过的栅格，
查重 51%
即 Pglobal 中至少含有 2 个 0 元素，那么无人机将在下一时刻**随机**选择其中一个栅格并搜索。

# 附录A　程序代码

xinxisu.m　（解决问题 1）

## 三、主要仪器设备

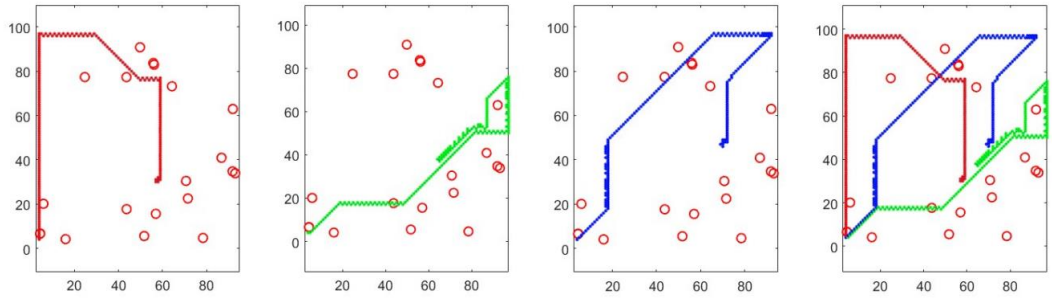
Matlab R2023a, Macbook Pro.

## 四、实验过程与实验结果

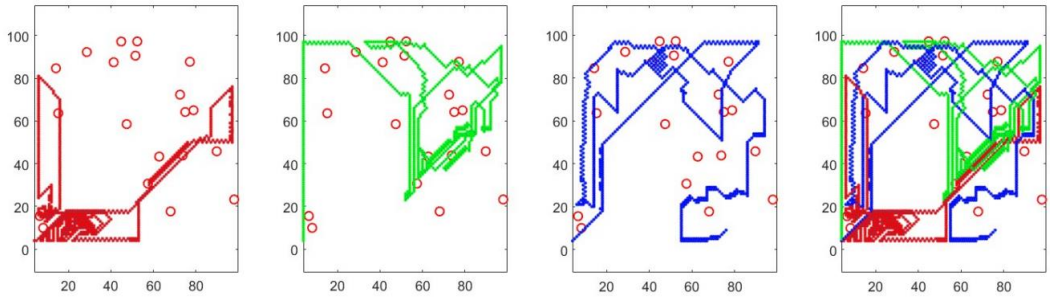
### 1. 目标物的数量与位置均未知

初始化一个 m=100,n=100,包含 20 个物体的空间，并规定其边界两层的 $P_{net}=+\infty$。取障碍为矩形，包络于[(20,20),(20,50),(50,20),(50,50)]点集之中。在原点释放三架搜索半径为 2 的无人机。其轨迹如图所示。

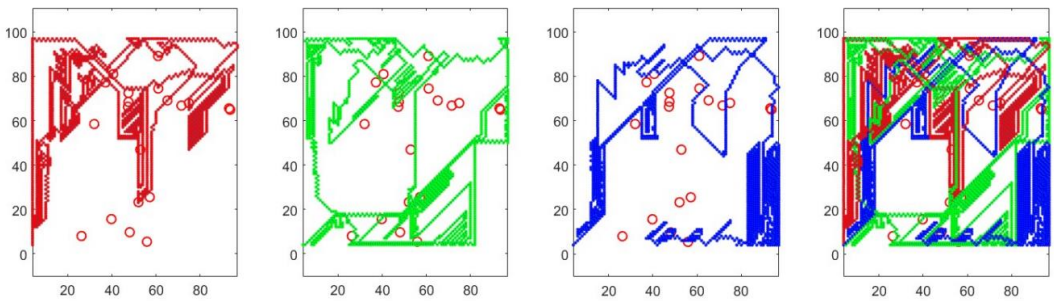
t = 200



t = 700



t = 1500

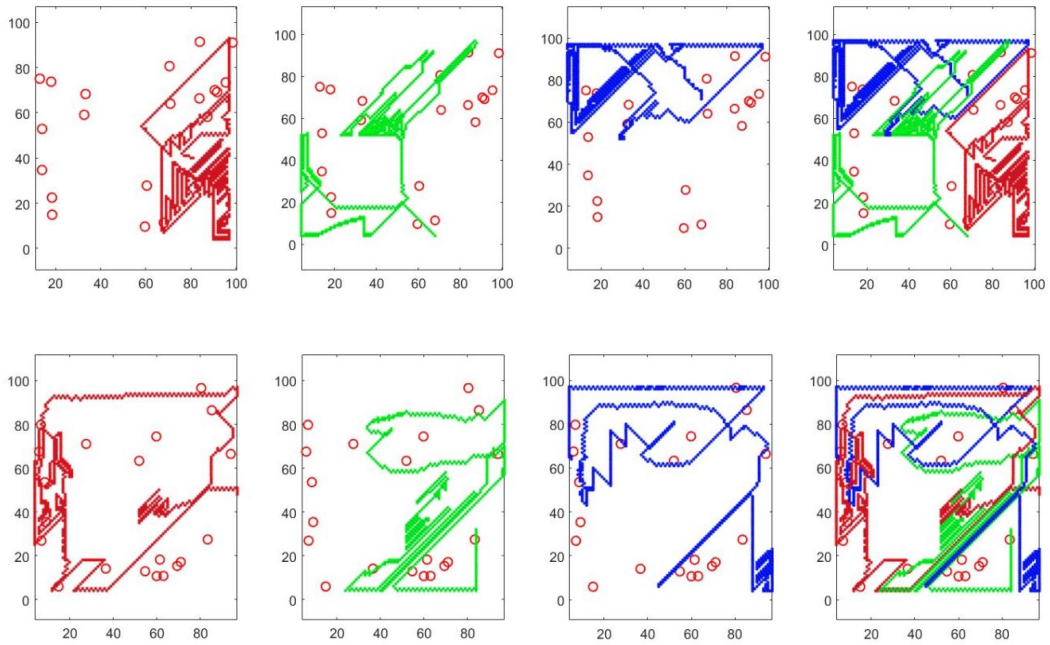


# 附录A　程序代码

xinxisu.m　（解决问题 1）

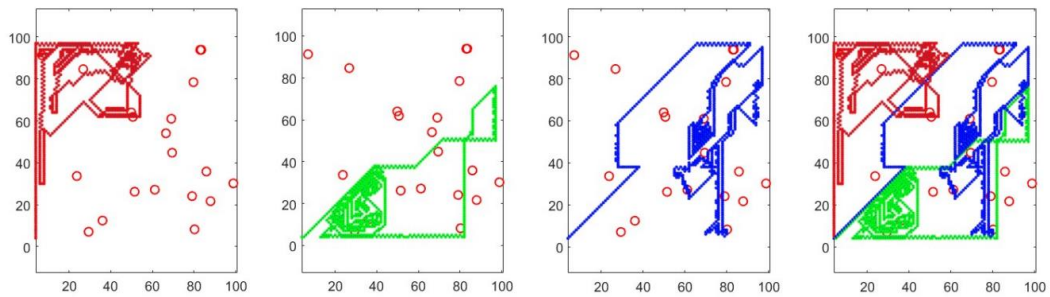
对比三个图像，可看到无人机集群有效探索、遍历了空间中的所有空位，并在 t=1500 时成功寻找到了所有的物体。

为探寻无人机释放位置对该算法效果是否有影响，取极端情况(修改基本假设),保持对照组条件不变，随机各无人机初始位置，让 T=700。效果如下：



经过同样时间，其释放位置基本不影响集群对空间的搜索率。可基本推定，该算法对无人机释放位置具有鲁棒性。

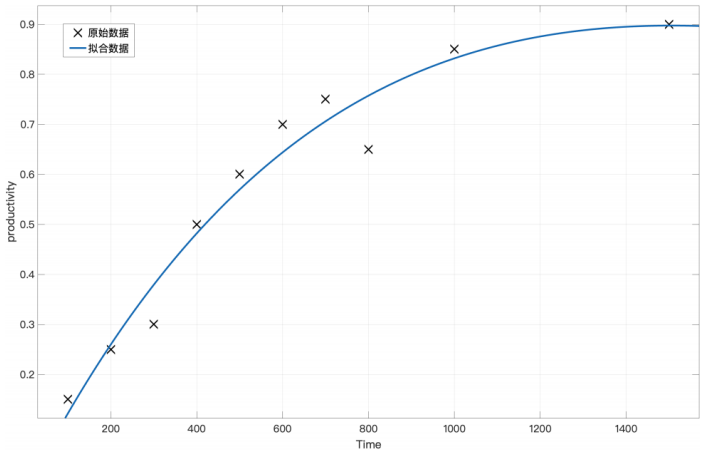
改变障碍物大小同理：



对第一组中的无人机集群对物体收率进行指数曲线拟合，发现其收率稳定上升，其变化速度随时间逐渐放缓。该算法有效达成遍历探索空间的目标，并能在合理的时间区间内寻找到所有物体。

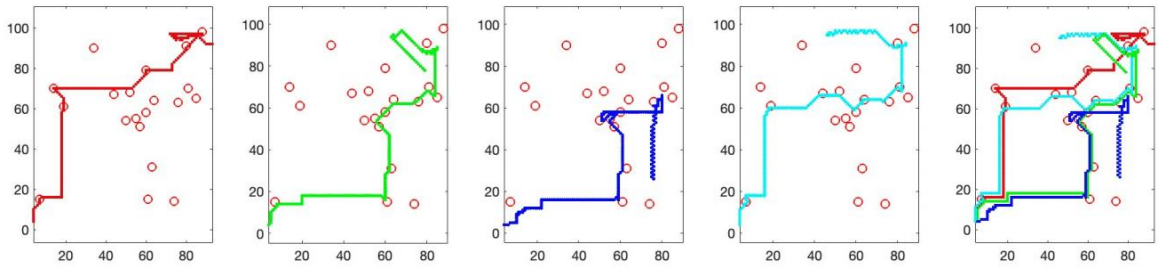
# 附录A　程序代码

xinxisu.m　（解决问题 1）




2. 目标物的数量与位置均已知

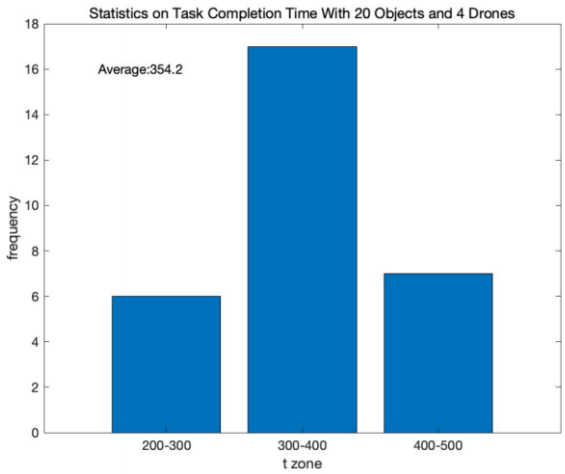
初始化一个 m=100,n=100,包含 20 个带有不同权重物体的空间，并 规定其边界两层的 $P_{net}=+\infty$。物体的权重总和为 1。取障碍为矩形，包络于 [(20,20),(20,50),(50,20),(50,50)]点集之中。在原点释放四架 d=2　　的无人机。其中一次仿真的轨迹如图所示。



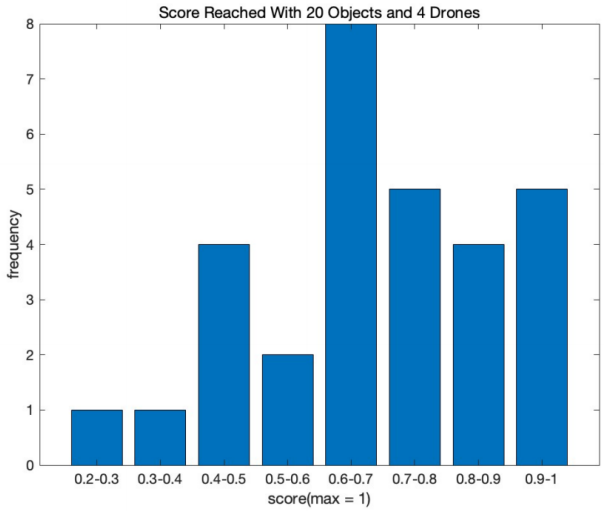

对该初始条件进行三十次随机试验(每一次物体位置、权重都不同)。其完成时间分布统计如下：

# 附录A　程序代码

xinxisu.m　（解决问题 1）




如图所示，平均完成时间为 352.4。
将时间限制为 200 秒，其搜索到的物体的权重之和如图所示：

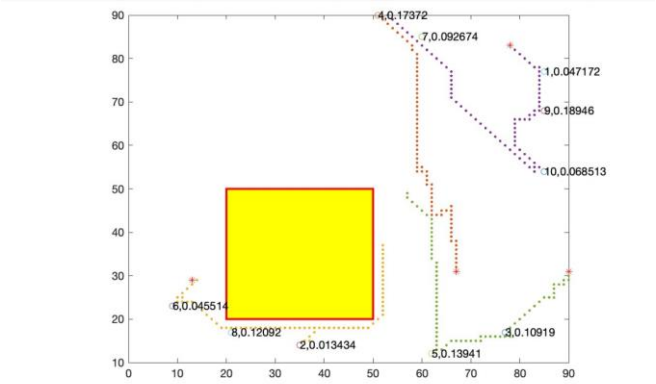



平均值为 0.69,标准差 0.19。于任务平均完成时间对比，可认为该算法有效地在给定时间内得分良好。

如果每个无人机的位置均随机，还可以达到极好的效果：

# 附录A　程序代码

xinxisu.m　（解决问题 1）




```
found No.6 object on (9) (23) at 7s
found No.9 object on (85) (68) at 15s
found No.8 object on (21) (17) at 17s
found No.3 object on (77) (17) at 18s
found No.10 object on (85) (54) at 34s
found No.5 object on (62) (12) at 36s
found No.2 object on (35) (14) at 39s
found No.7 object on (60) (85) at 66s
found No.4 object on (51) (90) at 72s
With a total dispense of 72s
```

## 五、实验体会

我所编写的模型具有许多优点。通过矩阵初始化和全局变量设置，模型结构清晰，易于理解和修改。这种简洁性不仅降低了模型的复杂度，还为进一步优化提供了基础。此外，我的模型支持多无人机的协同工作。通过共享的信息素矩阵和各自的状态矩阵，无人机集群可以实现高效的目标检测和搜索任务。这种协同机制使得多无人机系统在复杂环境中表现出更高的效率和可靠性。多无人机协同工作能够覆盖更大的搜索区域，提高了目标检测的成功率，且在面对障碍物时，协同机制能够有效减少碰撞风险。

模型还具备灵活的地形自适应能力，具体而言，表现在非常好的障碍设置能力。**通过简单调整矩阵值，可以模拟无人机集群在不同搜索环境中的表现。**不同边界形状和障碍物区域的设定增加了模拟环境的多样性，使模型更接近实际应用场景。这种灵活性不仅有助于测试无人机在多种环境中的适应性，还能为实际应用中的环境建模提供借鉴。此外，利用信息素进行全局搜索，使模型具有很强的全局搜索能力。信息素的分布和动态更新确保了所有区域都能被覆盖，减少了遗漏目标的可能性。这种全局搜索能力在广域搜索任务中尤为重要，能

# 附录A　程序代码

xinxisu.m　（解决问题 1）

够显著提高任务的完成度和精度。

但是这个模型的缺点也很明显。首先，对于网格边界的处理较为简单，只考虑了矩形边界，并直接设置为无限大（inf）。这种简单处理在实际应用中可能不够精细，可能无法准确反映复杂地形边界的真实情况。比如，**如果要把无人机空间拓展到三维，体现地形高度，这一模型就完全不够用了。**

其次，当前模型的无人机移动决策主要基于局部和全局信息素的结合，**缺乏高级的智能决策机制。**没有采用机器学习或强化学习等先进算法，导致无人机有时会出现不利于搜索的统一游走行为，降低了搜索效率。

模型将地形空间简化为笛卡尔坐标系下的矩阵空间，采用正方形格点分割。然而，这种方式会导致无人机在斜对角移动时速度与水平、竖直移动速度不一致，违背了速度约束。这种简化处理虽然在理论上可行，但在实际应用中可能会影响无人机的运动表现。

最后，模型忽略了实际场景中的动态变化。目标和障碍物的位置在初始化后是固定的，但实际的搜寻任务很少只涉及不动的物体，通常还涉及位置不断变化的人。对于这样的目标，定态模型显然不够。

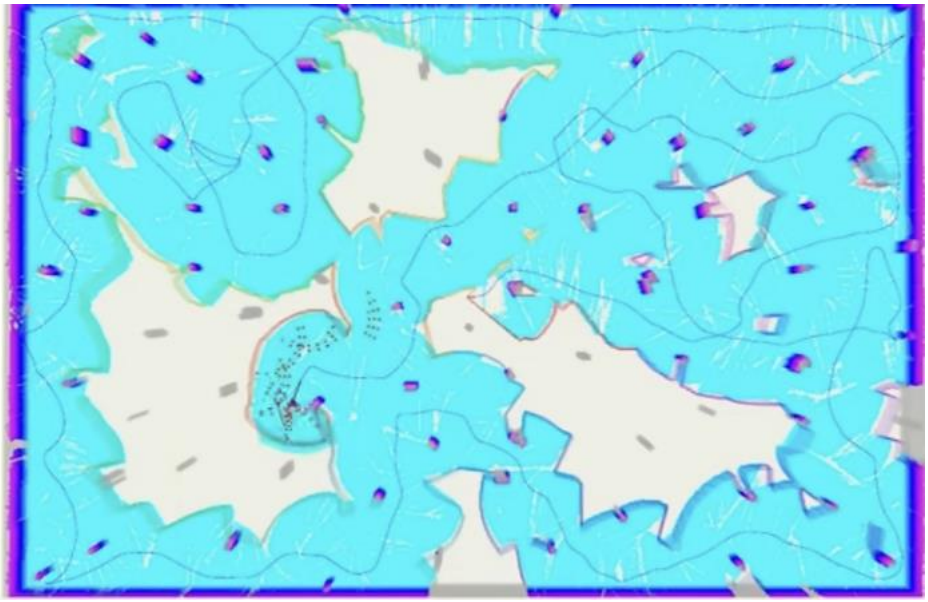
有没有考虑动态约束，并存在智能决策机制的模型？有。如下为一些可以用来结合改进该任务的方案。限于时间、平台和能力限制，这里不再复现代码，仅作展示。

FUEL：https://github.com/HKUST-Aerial-Robotics/FUEL

香港科技大学开发。引入了 NLopt 开源库的优化方法来求解系统中的约束问题，Lin-Kernighan 启发法解决路径规划问题（其认为路径规划为典型旅行商问题，如大多数路径规划算法会做的那样——而信息素方法巧妙地绕开了这一问题）。缺陷是，这仍然不是一个三维模型，但作为搜索遍历方法已经足够优秀。

# 附录A　程序代码

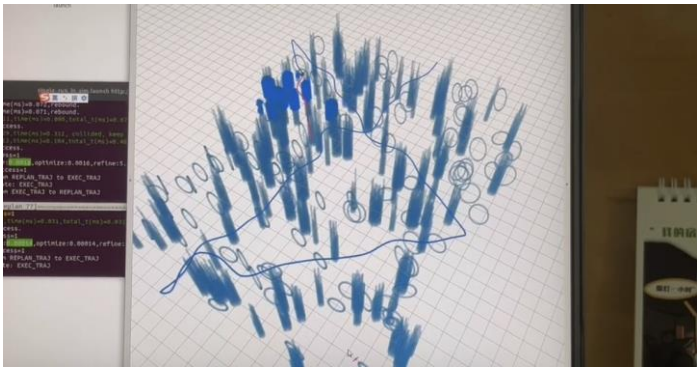
xinxisu.m　（解决问题 1）



其中一次仿真

EGOswarm　　：https://github.com/ZJU-FAST-Lab/ego-planner-swarm

本院高飞开发，其威力不再赘述。该算法可以大幅度改进无人机的避障和集群协作能力，且最大的优势在于为完全的三维模型，使得地形适应程度再上一层楼。（下为我的实机仿真）



六、实验收获

该次实验收获颇丰。通过该次实验，我初步了解了控制决策系统在无人机导航以及其派生的寻物任务中的应用，探讨了一种基于信息素动态更新的多无人机协同搜索模型，分析其优缺点，并提出了未来的改进方向。通过对模型的详细探讨，我对该模型的优点、缺点以及改进方向有了一些新的思考，并根据这些缺点寻找到了目前世界上最为前沿的算法，并挑选了其中一种（EGOplanner）上手，体会到了

# 附录A　程序代码

xinxisu.m　（解决问题 1）

其魅力。通过这些改进，可以进一步提升多无人机协同搜索模型的性能，使其在实际应用中更加高效和可靠。

同时，我也通过 MATLAB 代码的编写收获了一个非常粗糙，但是可塑性强、具有一定鲁棒性并且非常好玩的模型。我认识到了该问题的本质有如下数个方向：无人机状态方程获取（本模型没有引入动力学约束：我不会）、约束方程的建立与方向优化问题求解（本模型采用离散的信息素）以及无人机的路径规划（本模型和方向优化问题混同）。同时，我也提高了我的代码编写能力和调试能力。这对于以后独自完成更为复杂任务，建立更为准确的决策仿真模型大有裨益。

装订线

## 七、代码

在附录中提供。

# 附录A 程序代码

xinxisu.m （解决问题 1）

```
clear;
clc;

%length=100;
%width=100;
%sqrt(2)*d
global m n K M N T d;
%m = floor(width/(2*d))+2;%length;
%n = floor(length/(2*d))+2;; %width
m=100;
n=100;

T = 700;
K= 1;
M = 20;
N = 3;
d = 2;

%this section is just for plotting has nothing to do with the
   actual algorithm

Color array = ['r','g','b','c','m','y'];

%ends

P_net matrix = zeros (m,n);
%P_next matrix=zeros (m,n);
%P_matrix=zeros (m,n);
%%M matrix = zeros (m,n);
%row = 2;
%col = 2;%initiatematr
t=T; %initiate time
P_matrix next t_array=zeros (m,n,N);
P_matrix array=zeros (m,n,N);
ifdeploy_array=zeros (1,N);

M_matrix next t_array=zeros (m,n,N);
M_matrix array=zeros (m,n,N);
delta_P_matrix array=zeros (m,n,N);
delta_P_matrix new array=zeros (m,n,N);

%draw the boundary if there is one
   %contemporary one
for i=1:m
   P_net matrix(i,1,:)=inf;
   P_matrix next t_array(i,1,:)=inf;
   P_matrix array(i,1,:)=inf;
   M_matrix array(i,1,:)=inf;
   M_matrix next t_array(i,1,:)=Inf;
   P_net matrix(i,n)=inf;
```

```
   P_matrix next t_array(i,n,:)=inf;
   P_matrix array(i,n,:)=inf;
   M_matrix array(i,n,:)=inf;
   M_matrix next t_array(i,n,:)=Inf;
end
for i=1:n
   P_net matrix(1,i)=inf;
   P_matrix next t_array(1,i,:)=inf;
   P_matrix array(1,i,:)=inf;
   M_matrix array(1,i,:)=inf;
    M_matrix next t_array(1,i,:)=Inf;
   P_net matrix(m,i)=inf;
   P_matrix next t_array(m,i,:)=inf;
   P_matrix array(m,i,:)=inf;
   M_matrix array(m,i,:)=inf;
   M_matrix next t_array(m,i,:)=Inf;
end

%more layer!!

for i=2:m-1
   P_net matrix(i,2,:)=inf;
   P_matrix next t_array(i,2,:)=inf;
   P_matrix array(i,2,:)=inf;
   M_matrix array(i,2,:)=inf;
   M_matrix next t_array(i,2,:)=Inf;
   P_net matrix(i,n-1)=inf;
   P_matrix next t_array(i,n-1,:)=inf;
   P_matrix array(i,n-1,:)=inf;
   M_matrix array(i,n-1,:)=inf;
   M_matrix next t_array(i,n-1,:)=Inf;
end
for i=2:n-1
   P_net matrix(2,i)=inf;
   P_matrix next t_array(2,i,:)=inf;
   P_matrix array(2,i,:)=inf;
   M_matrix array(2,i,:)=inf;
    M_matrix next t_array(2,i,:)=Inf;
   P_net matrix(m-1,i)=inf;
   P_matrix next t_array(m-1,i,:)=inf;
   P_matrix array(m-1,i,:)=inf;
   M_matrix array(m-1,i,:)=inf;
   M_matrix next t_array(m-1,i,:)=Inf;
end

   %customized one
for i = 20:80
   for j = 20:70
      P_net matrix(i,j)=inf;
      P_matrix next t_array(i,j,:)=inf;
      P_matrix array(i,j,:)=inf;
      M_matrix array(i,j,:)=inf;
   end
end
```

```
%drone position initialization
%exapmle:
   row array=zeros (1,N);
   col_array=zeros (1,N);
   for i=1:N

      row_array(:,i)=3;
     col_array(:,i)=3;
   end

%row_array=floor((rand(1,N) *  (m-2) + 2));
%col_array=floor((rand(1,N) *  (n-2) + 2));


%object position initialization
flag_for_this_while = true;
while flag_for_this_while
   has_in_bound =false;

   object_x = rand(1,M) *  (m-4) + 4;
   object_y = rand(1,M) *  (n-4) + 4;

   for i = 1:M
      if  (
         P_net_matrix(floor(object_x(i)),floor(object_y(i)))
         == inf )
         has_in_bound = true;
      end
   end

   if  ( ~has_in_bound )
      flag_for_this_while = false;
   end
end

whether_found =  zeros (1,M);

complete = 0; %stupid sign on judging whether all the objects
   are on the sight


%UAV={P_matrix_next_t,P_matrix,M_matrix,row,col};
  %main function t++;
  %create UAV series
 % U=cell(N,5);
 % for i=1:N
  %    U(i,:)=UAV;
 % end

while(t)

   ifdeploy_array(:)=g_deploy(row_array(1),col_array(1));
      %initialize xinxisu
```

```
%find the objects at the beginning of each cycle
if(complete~=1)
   whether found = JudgeObject(whether_found,...
   row_array,col_array,object_x,object_y,t);
end

if( sum (whether_found) == M && ~complete )
   disp(['With a total dispense of ', num2str(T-t), 's']);
   complete = 1;
end

for i=1:N
 %ifdeploy_array(i)=g_deploy(row_array(i),col_array(i));
     %create xinxisu

 %%calculate the P global for decision of each UAV
  P_local=create P_local(row_array(i),...
  col_array(i),P_matrix_array(:,:,i));
  P_global_new =
     ZipP(P_matrix_array(:,:,i),row_array(i),col_array(i));
 % P_local trans = create P local_trans(P_local);
  P_local trans = P local;
  P global_total =
     P_global_create(P_local_trans,P_global_new);
 %...and their corresponding movement
  [hor,ver] = new_locate(P_global_total);
  row_array(i)=row_array(i) + hor;
  col_array(i)=col_array(i) + ver;

  ifdeploy_array(i)=g_deploy(row_array(i),col_array(i));
     %generate xinxisu after moving

 P_matrix array(:,:,i) =
    update_P_matrix(P_matrix_array(:,:,i),...
 ifdeploy_array(i),row_array(i),col_array(i)); %update the
    corresponding UAV's P matrix
 M_matrix array(:,:,i) =
    update_M_matrix(M_matrix_array(:,:,i),...
 t,row_array(i),col_array(i)); %update the corresponding
    UAV's trajectory

 delta_P_matrix new array(:,:,i)
    =update_delta_matrix(row_array(i),col_array(i)...
 ,ifdeploy_array(i));%delta P calculation store current
    location
 %plot each figure;

 figure(1)
 f = figure(1);
 u = f.Units;
 f.Units = 'inches';
 f.Position = [3,3,3*(N+1),3];
 subplot(1,N+1,i);
```

```
     [old_dot_x,old_dot_y]=find(delta_P_matrix_array(:,:,i));
     [new_dot_x,new_dot_y]=...
     find(delta_P_matrix_new_array(:,:,i));
     plot(old_dot_x,old_dot_y,'.',new_dot_x,new_dot_y,'.'...
     ,'Color',Color_array(i));
     line([old_dot_x,new_dot_x],[old_dot_y,new_dot_y]);


     if t == T
       for k = 1:M
          drawcircle(d,object_x(k),object_y(k));
       end
     end

     hold on

     subplot(1,N+1,N+1)
     [old_dot_x,old_dot_y]=find(delta_P_matrix_array(:,:,i));
     [new_dot_x,new_dot_y]=...
     find(delta_P_matrix_new_array(:,:,i));
     plot(old_dot_x,old_dot_y,'.',new_dot_x,new_dot_y,'.',...
     'Color',Color_array(i));
     line([old_dot_x,new_dot_x],[old_dot_y,new_dot_y]);

     if t == T
       for k = 1:M
          drawcircle(d,object_x(k),object_y(k));
       end
     end

    % plot();

     %plot([old_dot_x,old_dot_y],[new_dot_x,new_dot_y]);
     hold on

     delta_P_matrix_array(:,:,i)=...
     delta_P_matrix_new_array(:,:,i);
     %update delta_P and to store the last location;
     P_net_matrix =
        update_P_net(P_net_matrix,delta_P_matrix_array(:,:,i));
      %finish P net update

      %update all P_matrix of the UAVs
      for j = 1:N
       P_matrix_array(:,:,j)=P_net_matrix;
      end

   end
   t=t-1;
end

if(~complete)
   disp(['Unfortunately, only ',num2str(sum(whether_found)),'
      object(s) was found. Better luck next time!'])
```

```
end


%%plot the result
for i = 1:N
    [dotx,doty] = find(M_matrix_array(:,:,i) == 1);

   figure(3)
   plot(dotx,doty,'.')
   hold on
end
for i = 1:M
      plot(object_x,object_y,'-o');
end
%plot the object


function ifdeploy=g_deploy(row,col)
global m n K;
if  (row <= m && col<=n )
   ifdeploy = K;
end
end

function
   P_matrix next_t=update_P_matrix(P_matrix,ifdeploy,row,col)
P_matrix next t=P matrix;
P_matrix next t(row,col) = ifdeploy;
end

function M_matrix next_t = update_M_matrix(M_matrix,t,row,col)
global m n;
M_matrix next t=M matrix;
if t~=0
  if row <= m&& col<=n
      M_matrix next t(row,col) = 1;
  end
end
end
function delta_P_matrix =update_delta_matrix(row,col,ifdeploy)
global m n;
delta_P_matrix = zeros (m,n);
delta_P_matrix(row,col)=ifdeploy;
end

function P_net next t_=
   update_P_net(P_net_matrix,delta_P_matrix)
   P_net next t=P net matrix+delta P matrix;
end


%upload P_net_matrix
function P_local=create P_local(row,col,P_matrix)
P_local=zeros (3,3); %2<=row<=m-1 2<=col<=n-1
```

```
P_local(1,1)=1 * P_matrix(row-1,col-1) + 0.5 *
   P_matrix(row-2,col-2);
P_local(1,2)=1 * P_matrix(row-1,col) + 0.5 *
   sum (P_matrix(row-2,col-1:col+1)) / 3;
P local(1,3)=1 * P_matrix(row-1,col+1) + 0.5 *
   P_matrix(row-2,col+2);
P local(2,1)=1 * P_matrix(row,col-1) + 0.5 *
   sum (sum (P_matrix(row-1:row+1,col-2))) / 3;
P local(2,2)=P_matrix(row,col);
P local(2,3)=1 * P_matrix(row,col+1) + 0.5 *
   sum (sum (P_matrix(row-1:row+1,col+2))) / 3;
P local(3,1)=1 * P_matrix(row+1,col-1) + 0.5 *
   P_matrix(row+2,col-2);
P local(3,2)=1 * P_matrix(row+1,col) + 0.5 *
   sum (P_matrix(row+2,col-1:col+1)) / 3;
P_local(3,3)=1 * P_matrix(row+1,col+1) + 0.5 *
   P_matrix(row+2,col+2);
end

function P_local_trans = create_P_local_trans(P_local)
P_local_trans=P_local;
for i=1:3
  for j=1:3
     if(P_local_trans(i,j)~=inf)
         P_local_trans(i,j)=0;
      end
   end
end
end



function P_global_new = ZipP(P,row,col)
%Cram P into P_global
%  Nothing special
global m n;
P new = P;

i = row;
j = col;

%dunno why but whyyyyyyy?
indices =  (P_new == inf);
P new(indices) = 0;

P_global_new = zeros (3,3);

%calculate the P_global_new matrix
P global_new(1,3) = sum (sum (P_new(1:i,j:n)))/(i*(n - j + 1));
P global_new(2,3) = sum (sum (P_new(1:m,j:n)))/(m*(n - j + 1));
P_global_new(3,3) = sum (sum (P_new(i:m,j:n)))/((m - i + 1)*(n
   - j + 1));
```

```
P_global_new(1,2) = sum(sum(P_new(1:m,j:n)))/(m*(n - j + 1));
P global_new(2,2) = inf;
P_global_new(3,2) = sum(sum(P_new(1:m,1:n)))/((m - i + 1)*n);
P global_new(1,1) = sum(sum(P_new(1:i,1:j)))/(i*j);
P_global_new(2,1) = sum(sum(P_new(1:m,1:j)))/(m*j);
P global_new(3,1) = sum(sum(P_new(i:m,1:n)))/((m - i + 1)*j);

end

function P_global_total =
   P_global_create(P_local_trans,P_global_new)
P global_total = P_local_trans+P_global_new;
end

function  [hor,ver] = new_locate(P_global_total)
    minimum = min(min(P_global_total));
     [hor,ver]=find(P_global_total==minimum);%the grid index
        with the smallest value

    index = floor(rand*(length(hor))+1);
    hor = hor(index)-2;
    ver = ver(index)-2;

end

function whether_found = JudgeObject(whether_found,...
row_array,col_array,object_x,object_y,t)

global T M N d;%fuck MATLAB

   for i = 1:N
      for j = 1:M
         distance_square =  (row_array(i) -
            object_x(j))^2+(col_array(i)-object_y(j))^2;
         if( distance_square < d^2 &&whether_found(j)==0)
            disp(['found No.',num2str(j),' object on
                (',num2str(object_x(j)),')
                (',num2str(object_y(j)),') at
                ',num2str(T-t),'s']);
            whether_found(j) = 1;
         end
      end
   end
end

function drawcircle(r,x0,y0)
alpha=0:pi/50:2*pi;

x=r*cos(alpha)+x0;
y=r*sin(alpha)+y0;
plot(x,y,'r-','linewidth',1);
hold on;

axis equal;
```

```
   end
```

xinxisu2.m （解决问题 2）

```
clear;
clc;

%length=100;
%width=100;
%sqrt(2)*d
global m n K M N T d score;
%m = floor(width/(2*d))+2;%length;
%n = floor(length/(2*d))+2;; %width
m=100;
n=100;

T = 1500;
K= 1;
M = 20;
N = 4;
d = 2;

score = 0;

%this section is just for plotting has nothing to do with the
   actual algorithm

Color_array = ['r','g','b','c','m','y'];

%ends

P_net matrix = zeros(m,n);

%P_next matrix=zeros(m,n);
%P matrix=zeros(m,n);
%%M_matrix = zeros(m,n);
%row = 2;
%col = 2;%initiatematr
t=T; %initiate time
P_matrix next t_array=zeros(m,n,N);
P_matrix array=zeros(m,n,N);
ifdeploy_array=zeros(1,N);

M_matrix next t_array=zeros(m,n,N);
M_matrix array=zeros(m,n,N);
delta_P_matrix array=zeros(m,n,N);
delta_P_matrix new array=zeros(m,n,N);

%draw the boundary if there is one
   %contemporary one
for i=1:m
   P_net matrix(i,1,:)=inf;
   P_matrix next t_array(i,1,:)=inf;
   P_matrix array(i,1,:)=inf;
```

```
   M_matrix_array(i,1,:)=inf;
   M_matrix_next_t_array(i,1,:)=Inf;
    P_net_matrix(i,n)=inf;
    P_matrix_next_t_array(i,n,:)=inf;
    P_matrix_array(i,n,:)=inf;
   M_matrix_array(i,n,:)=inf;
   M_matrix_next_t_array(i,n,:)=Inf;
end
for i=1:n
   P_net_matrix(1,i)=inf;
   P_matrix_next_t_array(1,i,:)=inf;
   P_matrix_array(1,i,:)=inf;
   M_matrix_array(1,i,:)=inf;
    M_matrix_next_t_array(1,i,:)=Inf;
   P_net_matrix(m,i)=inf;
   P_matrix_next_t_array(m,i,:)=inf;
   P_matrix_array(m,i,:)=inf;
   M_matrix_array(m,i,:)=inf;
   M_matrix_next_t_array(m,i,:)=Inf;
end

%more layer!!

for i=2:m-1
   P_net_matrix(i,2,:)=inf;
   P_matrix_next_t_array(i,2,:)=inf;
   P_matrix_array(i,2,:)=inf;
   M_matrix_array(i,2,:)=inf;
   M_matrix_next_t_array(i,2,:)=Inf;
   P_net_matrix(i,n-1)=inf;
   P_matrix_next_t_array(i,n-1,:)=inf;
   P_matrix_array(i,n-1,:)=inf;
   M_matrix_array(i,n-1,:)=inf;
   M_matrix_next_t_array(i,n-1,:)=Inf;
end
for i=2:n-1
   P_net_matrix(2,i)=inf;
   P_matrix_next_t_array(2,i,:)=inf;
   P_matrix_array(2,i,:)=inf;
   M_matrix_array(2,i,:)=inf;
    M_matrix_next_t_array(2,i,:)=Inf;
   P_net_matrix(m-1,i)=inf;
   P_matrix_next_t_array(m-1,i,:)=inf;
   P_matrix_array(m-1,i,:)=inf;
   M_matrix_array(m-1,i,:)=inf;
   M_matrix_next_t_array(m-1,i,:)=Inf;
end

   %customized one
for i = 20:50
   for j = 20:50
      P_net_matrix(i,j)=inf;
      P_matrix_next_t_array(i,j,:)=inf;
      P_matrix_array(i,j,:)=inf;
```

```
      M_matrix array(i,j,:)=inf;
   end
end

row_array=zeros (1,N);
col_array=zeros (1,N);

%drone position initialization for problem1.
%exapmle:

   row array=zeros (1,N);
   col array=zeros (1,N);
   for i=1:N
      row array(:,i)=3;
      col_array(:,i)=3;
   end

%{
row_array=floor((rand(1,N) *  (m-4) + 4));
col_array=floor((rand(1,N) *  (n-4) + 4));
%}

figure(3)
plot(col_array,row_array,'*','Color','r')
hold on

%object position initialization
flag_for_this_while = true;
while flag_for_this_while
   has_in_bound =false;

   object_x = floor(rand(1,M) *  (n-4) + 4);
   object_y = floor(rand(1,M) *  (m-4) + 4);

   for i = 1:M
      if  ( P_net_matrix(object_y(i),object_x(i)) == inf )
         has_in_bound = true;
      end
   end

   if  ( ~has_in_bound )
      flag_for_this_while =  false;
   end
end

flag_outputgraph = 0;

whether_found =  zeros (1,M);

complete = 0; %stupid sign on judging whether all the objects
   are on the sight

%weight
object_weight =  zeros (1,M);
```

```
for i = 1:M
   object_weight(i) = rand;
end

constant = 1 / sum(object_weight);
object_weight = constant * object_weight;
%calculate the initial position in this problem

%{
for i = 1:N
   col array(i) = floor(sum(object_weight .* object_x) );
   row array(i) = floor(sum(object_weight .* object_y) );
end
%}

%for this problem, initialize the object_affected P_net
for i = 1:M
   %P_net matrix(object_x(i)-1:object_x(i)+1,...
   object_y(i)-1:object_y(i)+1) = P_net_matrix(...
   object_x(i)-1:object_x(i)+1,object_y(i)-1:object_y(i)+1) - ...
   object_weight(i) * 2 * ones(3,3);
   length = 15; %only odd numbers are permitted!

   Generated = GenerateMatrix(length);

   for j = 1:length
      for k = 1:length

         center =  (length + 1)/2;

         true_x = object_x(i) +  (j - center);
         true_y = object_y(i) +  (k - center);

         if( true_x >= 1 && true_y >=1 && true_x <= n && true_y
            <= m )
            P_net matrix(true_y,true_x) =
               P_net matrix(true_y,true_x) - 5 *
               object_weight(i)*Generated(j,k);
         end
      end
   end
end


%UAV={P_matrix_next_t,P_matrix,M_matrix,row,col};
  %main function t++;
  %create UAV series
 % U=cell(N,5);
 % for i=1:N
  %    U(i,:)=UAV;
 % end

while(t)
```

```
ifdeploy_array(:)=g_deploy(row_array(1),col_array(1));
   %initialize xinxisu

%find the objects at the beginning of each cycle
if(complete~=1)
    [whether_found,P_net_matrix] = ...
   JudgeObject(M_matrix_array,P_net_matrix,length,whether_found,...
   row_array,col_array,object_x,object_y,t,object_weight);
end

if( sum(whether_found) == M && ~complete )
   disp(['With a total dispense of ', num2str(T-t), 's']);
   complete = 1;
   flag_outputgraph = 1;
end

if( flag_outputgraph == 1 )
   for i = 1:N
       [doty,dotx] = find(M_matrix_array(:,:,i) == 1);

      figure(3)
      plot(dotx,doty,'.')
      hold on
   end
   rectangle('Position', [20, 20, 30, 30], 'EdgeColor', 'r',
       'FaceColor','y' , 'LineWidth', 2);

%plot the object
   for i = 1:M
      %plot(object_x(i),object_y(i),'o');
      dy = 0.05;
      text(object_x(i),object_y(i) +
         dy,[num2str(i),',',num2str(object_weight(i))]);
      hold on
   end

   for k = 1:M
      drawcircle(d,object_x(k),object_y(k));
   end

   flag_outputgraph = 2;
end

for i=1:N
  %ifdeploy_array(i)=g_deploy(row_array(i),col_array(i));
      %create xinxisu

  %%calculate the P_global for decision of each UAV
   P_local=create P_local(row_array(i),col_array(i),...
   P_matrix array(:,:,i));
   P global_new =
      ZipP(P_matrix_array(:,:,i),row_array(i),col_array(i));
  % P_local trans = create P local_trans(P_local);
   P_local_trans = P_local;
```

```
 P global_total =
    P global_create(P_local_trans,P_global_new);
%...and their corresponding movement
 [hor,ver] = new_locate(P_global_total);
 row_array(i)=row_array(i) + hor;
 col_array(i)=col_array(i) + ver;

 ifdeploy_array(i)=g_deploy(row_array(i),col_array(i));
    %generate xinxisu after moving

P_matrix array(:,:,i) = update_P_matrix(...
P_matrix array(:,:,i),ifdeploy_array(i),row_array(i)...
,col_array(i)); %update the corresponding UAV's P matrix M
matrix array(:,:,i) = update_M_matrix...
 (M_matrix_array(:,:,i),t,row_array(i),col_array(i));
  %update the corresponding UAV's trajectory

delta_P_matrix_new_array(:,:,i) =update_delta_matrix...
 (row_array(i),col_array(i),ifdeploy_array(i));
%delta P calculation store current location
%plot each figure;
%{
figure(1)
f = figure(1);
u = f.Units;
f.Units = 'inches';
f.Position = [3,3,3*(N+1),3];
subplot(1,N+1,i);
 [old_dot_y,old_dot_x]=find(delta_P_matrix_array(:,:,i));
 [new_dot_y,new_dot_x]=find(delta_P_matrix_new_array(:,:,i));
plot(old_dot_x,old_dot_y,'.',new_dot_x,new_dot_y,'.'...
,'Color',Color_array(i));
line([old_dot_x,new_dot_x],[old_dot_y,new_dot_y]);


if t == T
  for k = 1:M
     drawcircle(d,object_x(k),object_y(k));
  end
end

hold on

subplot(1,N+1,N+1)
 [old_dot_y,old_dot_x]=find(delta_P_matrix_array(:,:,i));
 [new_dot_y,new_dot_x]=find(delta_P_matrix_new_array(:,:,i));
plot(old_dot_x,old_dot_y,'.',new_dot_x,new_dot_y,'.'...
,'Color',Color_array(i));
line([old_dot_x,new_dot_x],[old_dot_y,new_dot_y]);

if t == T
  for k = 1:M
     drawcircle(d,object_x(k),object_y(k));
  end
```

```
     end

    % plot();

     %plot([old_dot_x,old_dot_y],[new_dot_x,new_dot_y]);
     hold on
     %}
     delta_P matrix_array(:,:,i)=delta_P_matrix_new_array(:,:,i);
        %update delta_P and to store the last location;
     P_net matrix =
        update_P_net(P_net_matrix,delta_P_matrix_array(:,:,i));
        %finish P_net update

      %update all P_matrix of the UAVs
      for j = 1:N
       P_matrix array(:,:,j)=P_net_matrix;
      end

   end
   t=t-1;
end

if(~complete)
   disp(['Unfortunately, only ',num2str(sum(whether_found)),'
      object(s) was found. Better luck next time!'])
end


%%plot the result if havent been plotted before
if( flag_outputgraph == 0 )
for i = 1:N
    [doty,dotx] = find(M_matrix_array(:,:,i) == 1);

   figure(3)
   plot(dotx,doty,'.')
   hold on
end
%plot the object
for i = 1:M
      plot(object_x,object_y,'-o');
      dy = 0.05;
      text(object_x,object_y +
         dy,num2str(i),num2str(object_weight(i)));
      hold on
end
end

function ifdeploy=g_deploy(row,col)
global m n K;
if  (row <= m && col<=n )
   ifdeploy = K;
end
end
```

```
function
   P_matrix next_t=update_P_matrix(P_matrix,ifdeploy,row,col)
P_matrix next_t=P matrix;
P_matrix_next_t(row,col) = ifdeploy;
end

function M_matrix_next_t = update_M_matrix(M_matrix,t,row,col)
global m n;
M_matrix next_t=M matrix;
if t~=0
  if row <= m&& col<=n
      M_matrix next_t(row,col) = 1;
  end
end

end
function delta_P_matrix =update_delta_matrix(row,col,ifdeploy)
global m n;
delta_P_matrix = zeros (m,n);
delta_P_matrix(row,col)=ifdeploy;
end

function P_net next_t = update_P_net(P_net_matrix,delta_P_matrix)
   P_net next_t=P net matrix+delta P matrix;
end


%upload P_net_matrix
function P_local=create P_local(row,col,P_matrix)
P_local=zeros (3,3); %2<=row<=m-1 2<=col<=n-1


P_local(1,1)=1 * P_matrix(row-1,col-1) + 0.5 *
   P matrix(row-2,col-2);
P_local(1,2)=1 * P_matrix(row-1,col) + 0.5 *
   sum (P_matrix(row-2,col-1:col+1)) / 3;
P local(1,3)=1 * P_matrix(row-1,col+1) + 0.5 *
   P_matrix(row-2,col+2);
P local(2,1)=1 * P_matrix(row,col-1) + 0.5 *
   sum (sum (P_matrix(row-1:row+1,col-2))) / 3;
P local(2,2)=P_matrix(row,col);
P local(2,3)=1 * P_matrix(row,col+1) + 0.5 *
   sum (sum (P_matrix(row-1:row+1,col+2))) / 3;
P local(3,1)=1 * P_matrix(row+1,col-1) + 0.5 *
   P_matrix(row+2,col-2);
P local(3,2)=1 * P_matrix(row+1,col) + 0.5 *
   sum (P_matrix(row+2,col-1:col+1)) / 3;
P_local(3,3)=1 * P_matrix(row+1,col+1) + 0.5 *
   P matrix(row+2,col+2);
end
%{
function P_local trans = create P local_trans(P_local)
P_local_trans=P_local;
for i=1:3
```

```
   for j=1:3
      if(P_local_trans(i,j)~=inf)
          P_local_trans(i,j)=0;
       end
    end
end
end

%}

function P_global_new = ZipP(P,row,col)
%Cram P into P global
%  Nothing special
global m n;
P_new = P;

i = row;
j = col;

%dunno why but whyyyyyyy?
indices =  (P_new == inf);
P new(indices) = 0;

P_global_new = zeros(3,3);

%calculate the P global_new matrix
%{
P_global_new(1,3) = sum(sum(P_new(1:i,j:n)))/(i*(n - j + 1));
P_global_new(2,3) = sum(sum(P_new(1:m,j:n)))/(m*(n - j + 1));
P_global_new(3,3) = sum(sum(P_new(i:m,j:n)))/((m - i + 1)*(n - j
   + 1));
P global_new(1,2) = sum(sum(P_new(1:i,1:n)))/(i*n);
P_global_new(2,2) = inf;
P_global_new(3,2) = sum(sum(P_new(i:m,1:n)))/((m - i + 1)*n); P
global_new(1,1) = sum(sum(P_new(1:i,1:j)))/(i*j);
P global_new(2,1) = sum(sum(P_new(1:m,1:j)))/(m*j);
P global_new(3,1) = sum(sum(P_new(i:m,1:n)))/((m - i + 1)*j);
%}

P_global_new(1,3) = sum(sum(P_new(1:i,j:n)));
P global_new(2,3) = sum(sum(P_new(1:m,j:n)))/2;
P global_new(3,3) = sum(sum(P_new(i:m,j:n)));
P_global_new(1,2) = sum(sum(P_new(1:i,1:n)));
P global_new(2,2) = inf;
P_global_new(3,2) = sum(sum(P_new(i:m,1:n)));
P global_new(1,1) = sum(sum(P_new(1:i,1:j)));
P global_new(2,1) = sum(sum(P_new(1:m,1:j)))/2;
P_global_new(3,1) = sum(sum(P_new(i:m,1:j)));
%}
end

function P_global_total =
   P_global_create(P_local_trans,P_global_new)
P global_total = 2 * P_local_trans+ P_global_new / 3000 ;
```

```
end

function  [hor,ver] = new_locate(P_global_total)
    minimum = min(min(P_global_total));
     [hor,ver]=find(P_global_total==minimum);%the grid index with
        the  smallest value

    index = floor(rand*(length(hor))+1);
    hor = hor(index)-2;
    ver = ver (index)-2;

end

function  [whether_found,P_net_matrix] = JudgeObject(...
M_ma,P_net_matrix,length,whether_found,row_array,...
col_array,object_x,object_y,t,object_weight)

global T M N m n d K score;%fuck MATLAB

   for i = 1:N
      for j = 1:M
         distance_square =  (row_array(i) -
            object_y(j))^2+(col_array(i)-object_x(j))^2;
         if( distance_square < d^2 &&whether_found(j)==0)
            score = score + object_weight(j);
            disp(['found No.',num2str(j),' object on
                (',num2str(object_x(j)),')
                (',num2str(object_y(j)),') at  ',num2str(T-t),'s,
               score',num2str(round(score,3))]);

            whether_found(j) = 1;

            Generated = GenerateMatrix(length);

            for k = 1:length
               for l = 1:length

                  center =  (length + 1)/2;

                  true_x = object_x(j) +  (k - center);
                  true_y = object_y(j) +  (l - center);

               if( true_x >= 1  && true_y >=1  && true_x <= n  &&
                  true_y <= m )
                  P_net matrix(true_y,true_x) =
                     P_net matrix(true_y,true_x) + 5 *
                     object_weight(j)*Generated(k,l);
               end
               end
            end

            for a = 1:m
               for b = 1:n
                  if(P_net_matrix(a,b) ~= inf &&
```

```
                    mod(P_net_matrix(a,b),K) == 0 )
                    P_net_matrix(a,b) = 0;
                end
            end
        end

    end
  end
end

function drawcircle(r,x0,y0)
alpha=0:pi/50:2*pi;

x=r*cos (alpha)+x0;
y=r*sin(alpha)+y0;
plot(x,y,'r-','linewidth',1);
hold on;

axis equal;
end

function Generated = GenerateMatrix(length)
%remember to set on odd number!
   layer =  (length+1)/2;
   Generated = zeros (length,length);

   for i = 1:layer
      Generated(i:length+1-i,i:length+1-i) =
         Generated(i:length+1-i,i:length+1-i) + 0.3 *
         ones(2*(layer - i)+1,2*(layer - i)+1);
   end
end
```