1.实验名称及目的

1.1实验名称

蚂蚁算法多无人机路径规划实验（仅限完整版及以上版本）

1.2实验目的

通过应用并优化蚂蚁算法规划出一条可行且较优的路径，这条路径需要符合避障以及避碰的要求。

1.3关键知识点

蚂蚁算法是一种模拟大自然中蚂蚁觅食行为演化出来的智能算法。单个蚂蚁的觅食行为看起来是无序的，但一个蚂蚁种群团结协作就会表现出规律有序的觅食行为。在蚂蚁进行食物搜索的时候，会在行走过的路径上释放一种名为“信息素”的化学物质。当其他蚂蚁经过这条路径时，蚂蚁之间可以通过路径上残留的信息素进行信息交流。单个蚂蚁信息素含量相近，且信息素会随着时间而不断挥发，因此在完全随机的情况下，到达食物源长度越短的路径上遗留信息素的浓度会更高，也会吸引更多的蚂蚁沿着较短路径前进，从而使得较短路径上的信息素浓度不断提高，最终形成一条从巢穴通往食物源的较优路径。在较长路径上，蚂蚁遗留信息素的浓度会较低，也导致更少的蚂蚁选择较长路径，最后导致较长路径上的信息素浓度不断降低。这就是蚂蚁觅食行为中的正反馈的机制。如图1直观的表述了蚂蚁觅食行为的信息素协同机制。

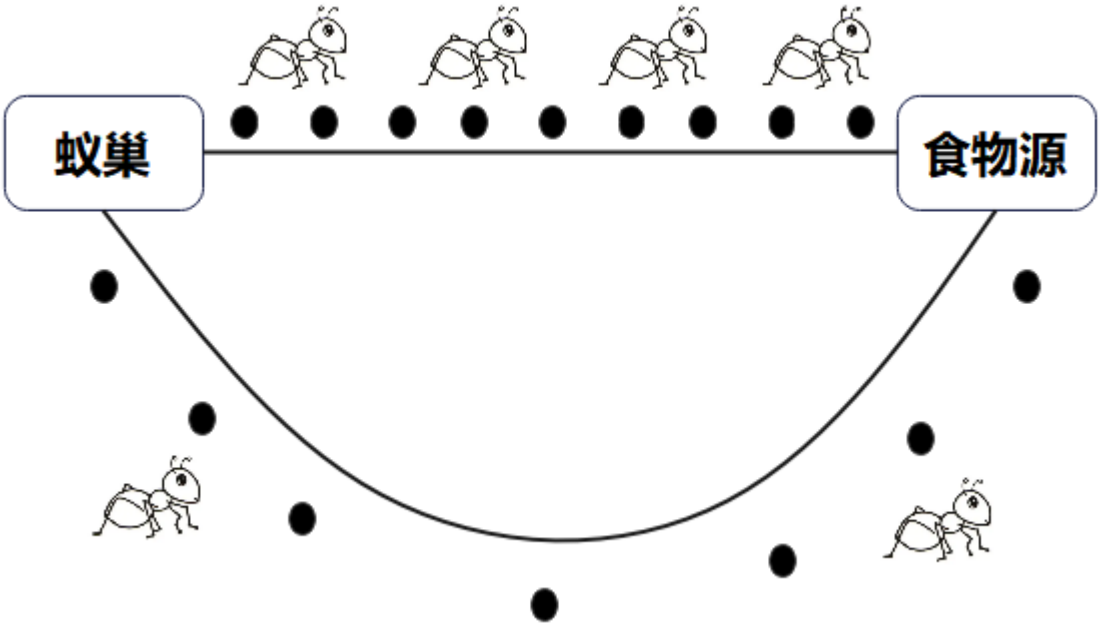

图1 信息素协同机制

但并不是所有蚂蚁都一定会沿着信息素浓度高的路径前进，会有少部分蚂蚁去探索信息素浓度偏低的路径，期待着找出比原路径更优的路径。在蚂蚁算法中，利用“信息素”机制，蚂蚁能够记忆已经找到过的较优路径，使得算法后续能够找到适应度较高的解。部分选择信息素浓度低的路径进行探索的蚂蚁，在算法中则表现为解的随机性，使得算法摆脱局部最优，尽可能找到全局最优解。

传统的蚂蚁算法，会对每个蚂蚁走过的路径都进行信息素更新，这样可能会使得一些劣质蚂蚁引导种群发展方向偏离全局最优，增加算法收敛时间。本节对传统蚂蚁算法进行了一些改进：

1、只更新种群中最优蚂蚁走过路径上的信息素，极大的降低了劣质蚂蚁对种群朝着全局最优方向更新的影响，防止蚂蚁算法陷入局部最优。如图2所示，假设目前种群中有三只蚂蚁，黑色方格为障碍物。在种群一次迭代中，每只蚂蚁找出路径如图所示，从图中可以很明显的看出蚂蚁1找出的路径是本次迭代最优的，所以改进蚂蚁算法只会更新蚂蚁1找出的路径上的信息素，防止误导下一轮迭代的蚂蚁朝蚂蚁2或蚂蚁3的方向前进。

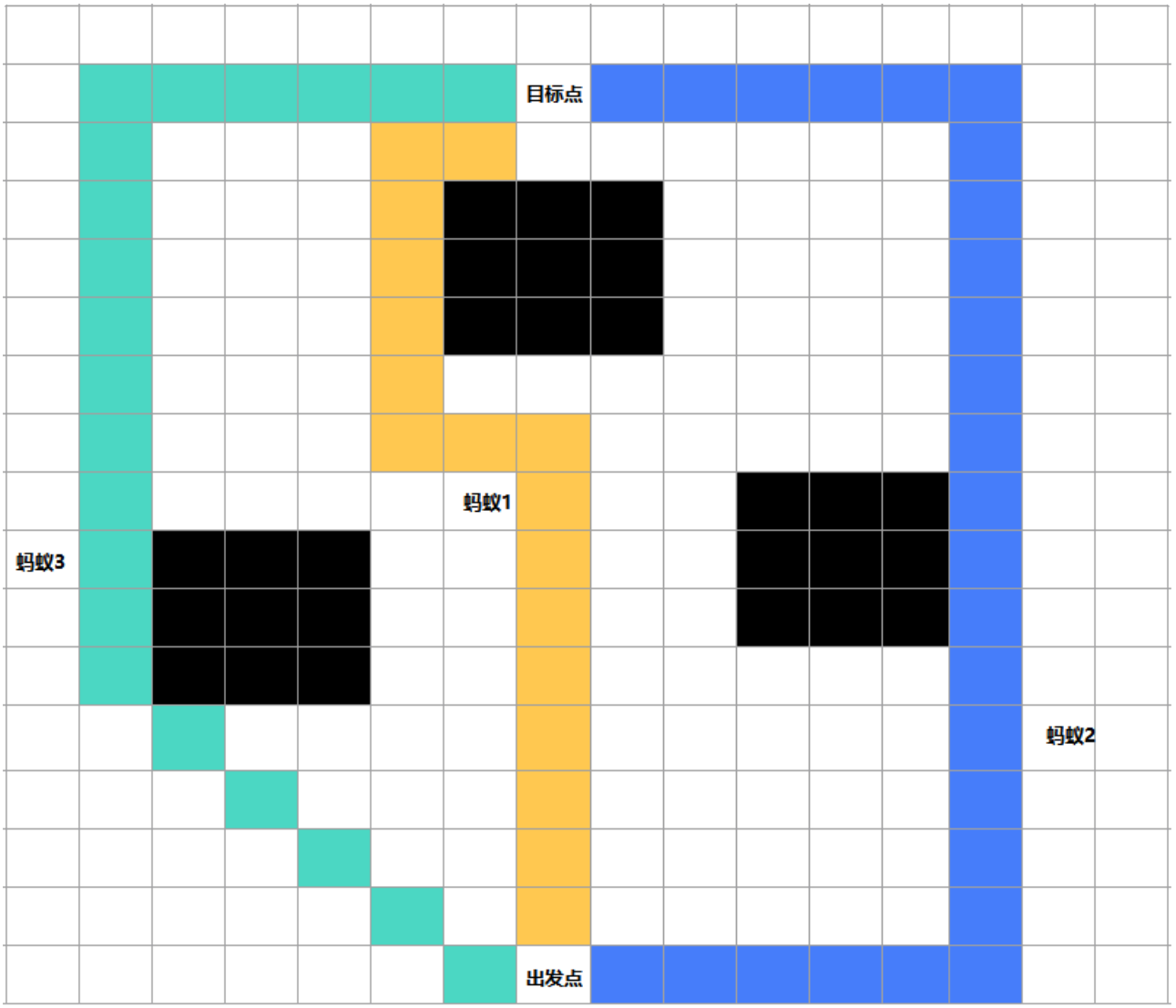

图2 信息素优化示例

2、由于传统蚂蚁算法对目标点没有导向性，只会避开障碍物。如公式1，其中τ_{ij}表示方格i到j路径上信息素浓度，η_{ij}表示从方格i到j转移的期望程度，$allow_k$为可转移方格集合，P_{ij}为从i转移到j的转移概率。由基本蚂蚁算法的转移函数的构成可知，其对目标点导向性不强。

$$P_{ij}^k = \begin{cases} \frac{[\tau_{ij}]^\alpha \bullet [\eta_{ij}]^\beta}{\sum_{s \in allow_k} [\tau_{is}]^\alpha \bullet [\eta_{is}]^\beta}, j \in allow_k \\ 0 \qquad\qquad\qquad\qquad , j \notin allow_k \end{cases} \quad \#\,(\text{公式1})$$

本节在传统目标转移函数中加入蚂蚁当前位置到目标点距离的倒数，引导蚂蚁朝向目标点的方向前进，提高算法收敛的速度。如公式2所示，其中n为可转移方格数量，P_i为转移到相对于当前方格的第i个方格的概率，$d_{b,i}$为相对于当前方格的第i个方格与障碍物的最近距离，$\frac{1}{d_{aim,i}}$为相对于当前方格的第i个方格与目标点的距离的倒数，τ_i为相对于当前方格的第i个方格的信息素含量。其中α，β，γ表示三者的重要程度。

$$P_i = \frac{(d_{b,i})^{\alpha} * \left(\frac{1}{d_{aim,i}}\right)^{\beta} (\tau_i)^{\gamma}}{\sum_{j=0}^{n} (d_{b,j})^{\alpha} * \left(\frac{1}{d_{aim,j}}\right)^{\beta} (\tau_j)^{\gamma}} \#(公式2)$$

在多无人机航迹规划中，还需考虑多无人机的协同约束。为了使得本节描述不会过于复杂，本节仅考虑空间约束，即多个无人机在进行航迹规划的过程中，彼此之间必须保持安全距离，避免碰撞。机体之间的距离D_{ij}需满足：

$$D_{ij} \geq D_{safe}$$

在本节中，为了简单起见，我们采用蚂蚁算法，以多个无人机的总体航程最短作为优化目标进行多无人机轨迹规划。读者可在后续工作中，加入更多的约束条件实现轨迹规划。

主要程序文件解析

main.py - 主程序入口

main.py是整个实验的主程序入口文件，负责初始化仿真环境、创建无人机实例、生成障碍物、启动蚂蚁算法并控制无人机执行规划路径。

```python
import time
import Barrier
import PX4MavCtrlV4 as PX4MavCtrl
from Detect import Detect
import UE4CtrlAPI


if __name__ == '__main__':
    # 创建实例
    ue = UE4CtrlAPI.UE4CtrlAPI()
    mav = PX4MavCtrl.PX4MavCtrler(1)
    mav1 = PX4MavCtrl.PX4MavCtrler(2)
    mav2 = PX4MavCtrl.PX4MavCtrler(3)
    mav_list = [mav, mav1, mav2]
    # 生成障碍物
    barrier = Barrier.Barrier(mav)

    for i in range(len(mav_list)):
        mav_list[i].InitMavLoop()
    print("Simulation Start.")
    print("Enter Offboard mode.")
    time.sleep(5)
    for i in range(len(mav_list)):
        mav_list[i].initOffboard()
    time.sleep(2)
    ue.sendUE4Cmd('RflyChangeViewKeyCmd P', 0)  # 开启碰撞引擎
    ue.sendUE4Cmd('RflyChangeViewKeyCmd T', 0)  # 开启轨迹显示
    time.sleep(1)
    TargetPos = [140, 0, -8]
    detect = Detect(barrier, mav_list, TargetPos)
    detect.run_path()
```

Barrier.py - 障碍物生成模块

Barrier类负责在仿真环境中随机生成圆柱形障碍物。

```
# 障碍物类
class Barrier:
    # 参数为通信实例
    def __init__(self, mav):
        # 障碍物设置区域
        self.barrier_pos = []
        # 设置障碍物个数，每square*square方格放一个
        self.square = 6
        # 记录每一行有多少个障碍物
        self.line_barrier = [0]
        ue = UE4CtrlAPI.UE4CtrlAPI()

        Y_max = 33
        Y_min = -33
        X_max = 100
        X_min = -20
        # 一行最多可放障碍物个数
        self.line_num = int((Y_max - Y_min) / self.square)
        # sum = (Y_max - Y_min)*(X_max - X_min)/pow(square, 2)
        count = 0
        x_min = X_min
        x_max = X_min
        y_min = Y_min
        # 编号从5开始，防止和无人机编号冲突
        num = 5
        # 障碍物中心点最小间距
        distance = 12
        # 发送设置障碍物命令
        serial_num = 120

        while x_max < X_max:
            x_max = x_min + self.square
            for i in range(self.line_num):
                y_max = y_min + self.square
                x = np.random.uniform(x_min, x_max, 1)
                y = np.random.uniform(y_min, y_max, 1)
                # 防止障碍物距离太近
                # 是否在当前点创建障碍物
                flag = True
                if count == 0:
                    ue.sendUE4Pos(num, serial_num, 0, [x[0], y[0], -0.04], [0, 0, 0])
                    self.barrier_pos.append([x[0], y[0]])
                    num = num + 1
                    count = count + 1
                else:
                    if count < 15:
                        for j in range(0, len(self.barrier_pos), 1):
                            if math.sqrt(pow(self.barrier_pos[j][0] - x[0], 2) + pow(

                                flag = False
                            else:
                                continue
```

```
                        if flag:
                            ue.sendUE4Pos(num, serial_num, 0, [x[0], y[0], -0.04], [0
                            self.barrier_pos.append([x[0], y[0]])
                            num = num + 1
                            count = count + 1
                    else:
                        for j in range(count - 15, len(self.barrier_pos), 1):
                            if math.sqrt(pow(self.barrier_pos[j][0] - x[0], 2) + pow(
                                                                                    2))
                                flag = False
                            else:
                                continue
                        if flag:
                            ue.sendUE4Pos(num, serial_num, 0, [x[0], y[0], -0.04], [0
                            self.barrier_pos.append([x[0], y[0]])
                            num = num + 1
                            count = count + 1
                y_min = y_max
            x_min = x_max
            y_min = Y_min
            self.line_barrier.append(num - 5)
        print(self.barrier_pos)
        print(self.line_barrier)
```

Detect.py - 蚂蚁算法核心实现

Detect类是蚂蚁算法路径规划的核心实现模块，包含了完整的蚁群算法实现。

```
class Detect:
    def __init__(self, barrier, mav, aim_position):
        # 角度和弧度记得转化
        # 无人机起始位置点
        self.start_position = [[-30, -20, 0], [-30, 0, 0], [-30, 20, 0]]
        # 障碍物设置初始和末尾横坐标
        self.x_start_barrier = -20
        self.x_end_barrier = 100
        # 场景信息
        self.barrier = barrier
        # 飞机信息
        self.mav = mav
        # 设置安全半径距离
        self.safe_distance = 4.5
        # 目标位置
        self.aim_position = aim_position
        # 方格大小
        self.rect = 1
        # 地图信息素信息,值为0说明此处有障碍物
        self.map = [[1 for i in range(int(80 / self.rect))] for j in range(int(140 /
        # 保存路径长度和路径
        self.path = [[], [], []]
        # 概率转移函数启发因子
        # 与障碍物的距离
        self.alpha = 1
        # 与目标点距离
        self.beta = 3
        # 信息素强度
        self.gamma = 1
        # 信息素衰减
        self.damp = 0.6
        # 蚂蚁种群
        self.population_count = 20
        # 迭代次数
        self.iter = 120
        # 一只蚂蚁的信息素的量
        self.pheromone = 200
        # 当前最优路径
        self.best_result = [[10000, []], [10000, []], [10000, []]]
        # 行走路径
        self.aim_list = [[], [], []]
        # 开始方格
        self.start_square = [[0, 19], [0, 39], [0, 59]]
        # 速度
        self.vel = 2
        # 展示
        self.show = []

    # 蚂蚁算法,获取一条安全路径
    def ant(self, serial):
        # 开始方格
        start_square = self.start_square[serial]
```

```
            # 结束方格
            end_square = self.getgrid([self.aim_position[0]+self.start_position[serial][0
            # 当前方格
            pre_square = start_square
            # 行走路径距离
            path_dist = 0
            # 路径点
            path = []
            # 当前方格已经穿越障碍物区域
            while pre_square[0] <= (self.x_end_barrier - self.start_position[serial][0])
                # 从当前方格只能选择正前，左前，右前，左，右的五个方格前进
                x = pre_square[0]
                y = pre_square[1]
                pre_allow = [[x, y - 1], [x, y + 1], [x + 1, y - 1], [x + 1, y], [x + 1,
                # 去掉不合格点
                num = 0
                for i in range(len(pre_allow)):
                    if pre_allow[i-num][1] < 0 or pre_allow[i-num][1] > 75:
                        pre_allow.pop(i-num)
                        num = num + 1
                pre_allow_possibility = []
                # 计算转移概率
                for i in range(len(pre_allow)):
                    # 预测点的位置
                    pre_pos = self.getpos(pre_allow[i])
                    # 判断当前方格在第几组障碍物旁
                    if pre_pos[0] <= self.x_start_barrier:
                        start_serial = 1
                    else:
                        start_serial = math.ceil((pre_pos[0] - self.x_start_barrier) / se
                        if start_serial > len(self.barrier.line_barrier) - 2:
                            start_serial = len(self.barrier.line_barrier) - 2
                    # 与障碍物最小距离
                    dist_barrier = 100
                    for j in range(self.barrier.line_barrier[start_serial - 1], self.barr
                        d = math.sqrt(
                            pow(pre_pos[0] - self.barrier.barrier_pos[j][0], 2) + pow(
                                pre_pos[1] - self.barrier.barrier_pos[j][1], 2))
                        if d < dist_barrier:
                            dist_barrier = d
                    # 与临时目标点的距离
                    dist_aim = math.sqrt(
                        pow(pre_pos[0] - self.aim_position[0] - self.start_position[seria
                    pre_allow_possibility.append(pow(dist_barrier, self.alpha) * pow(1 /
                        self.map[pre_allow[i][0]][pre_allow[i][1]], self.gamma))
                # 确定转移方格
                pre_allow_possibility_sum = sum(pre_allow_possibility)
                if pre_allow_possibility_sum == 0:
                    return 0
                pre_allow_possibility = [i / pre_allow_possibility_sum for i in pre_allow
                pre_allow_possibility = np.cumsum(pre_allow_possibility)
                pre_allow_possibility = pre_allow_possibility.tolist()
                r = np.random.rand()
```

```
                for i in range(len(pre_allow_possibility)):
                    if r < pre_allow_possibility[i]:
                        pre_square = pre_allow[i]
                        break
                path.append(pre_square)
                path_dist = path_dist + self.getdist(pre_square)
                if pre_square == end_square:
                    return [path_dist, path]

        # 确定迭代下的最优路径
        def Ant_Path(self, serial):
            # 蚂蚁迭代iter次
            better_result = [10000, []]
            self.detect()
            for i in range(self.iter):
                for j in range(self.population_count):
                    result = self.ant(serial)
                    if result != 0:
                        # 选择种群中最好的蚂蚁
                        if result[0] < better_result[0]:
                            better_result = result
                    else:
                        print("蚂蚁死亡")
                # 修改全局信息素
                for j in range(int(140 / self.rect)):
                    for k in range(int(80 / self.rect)):
                        if [j, k] in better_result[1]:
                            self.map[j][k] = self.map[j][k]*self.damp + self.pheromone/be
                        else:
                            self.map[j][k] = self.map[j][k]*self.damp
                print("当前迭代次数为", i+1, "最佳路径长度为", better_result[0])

        def run_path(self):
            target_aim = [[0, 0], [0, 0], [0, 0]]
            show_list = [[], [], []]
            for i in range(3):
                self.aim_list[i], show_list[i] = self.Ant_Path(i)
            # 画图
            for i in range(3):
                for k in show_list[i]:
                    self.show[k[0]][k[1]] = 0.5
            p = multiprocessing.Process(target=show_ant, args=(self.show,))
            p.start()
            # 准备起飞
            for i in range(3):
                self.mav[i].SendMavArm(True)  # Arm the drone
            time.sleep(1)
            for i in range(3):
                self.mav[i].SendPosNED(0, 0, -8, 0)
            print("起飞了")
            time.sleep(6)
            timeInterval = 1 / 1000
            lastTime = time.time()
```

```
        while len(self.aim_list) > 0:
            for i in range(3):
                if len(self.aim_list[i]) > 0:
                    # 每0.001s执行一次以下程序
                    sleepTime = time.time() - lastTime
                    if sleepTime < timeInterval:
                        # time.sleep(timeInterval - sleepTime)
                        # 不断调整
                        # 看当前无人机位置可否直接连接规划的点
                        if math.sqrt(pow(self.aim_list[i][0][0] - self.mav[i].uavPosN
                                self.aim_list[i][0][1] - self.mav[i].uavPosNED[1], 2)
                            if len(self.aim_list[i]) > 6:
                                le = 6
                            else:
                                le = len(self.aim_list[i])-1
                            for j in range(le, 0, -1):
                                if self.Is_safe_line(self.mav[i].uavPosNED, self.aim_
                                    for k in range(0, j, 1):
                                        self.aim_list[i].pop(0)
                                    break
                                if time.time() - lastTime > timeInterval:
                                    break
                            sleepTime = time.time() - lastTime
                            if sleepTime < timeInterval:
                                time.sleep(timeInterval - sleepTime)
                        else:
                            time.sleep(timeInterval - sleepTime)
                    lastTime = time.time()
                    yaw = math.atan2(self.aim_list[i][0][1] - self.mav[i].uavPosNED[1
                                     self.aim_list[i][0][0] - self.mav[i].uavPosNED[0
                    if math.sqrt(pow(self.aim_list[i][0][0] - self.mav[i].uavPosNED[0
                            self.aim_list[i][0][1] - self.mav[i].uavPosNED[1], 2)) <
                            self.mav[i].uavPosNED[0] < 0.05:
                        self.aim_list[i].pop(0)
                    if len(self.aim_list[i]) > 0:
                        target_aim[i][0] = self.mav[i].uavPosNED[0]
                        target_aim[i][1] = self.mav[i].uavPosNED[1]
                        if target_aim[i][0] < self.aim_list[i][0][0]:
                            target_aim[i][0] = target_aim[i][0] + (self.vel/1000+self
                            target_aim[i][1] = target_aim[i][1] + (self.vel/1000+self
                        # self.mav.SendPosNED(target_aim[i][0], target_aim[i][1], sel
                        self.mav[i].SendVelNED(self.vel * math.cos(yaw), self.vel * m
                    else:
                        self.mav[i].endOffboard()
                        time.sleep(0.1)
                        self.mav[i].stopRun()
```

2.实验效果

本实验通过蚂蚁算法规划出的路径如下：

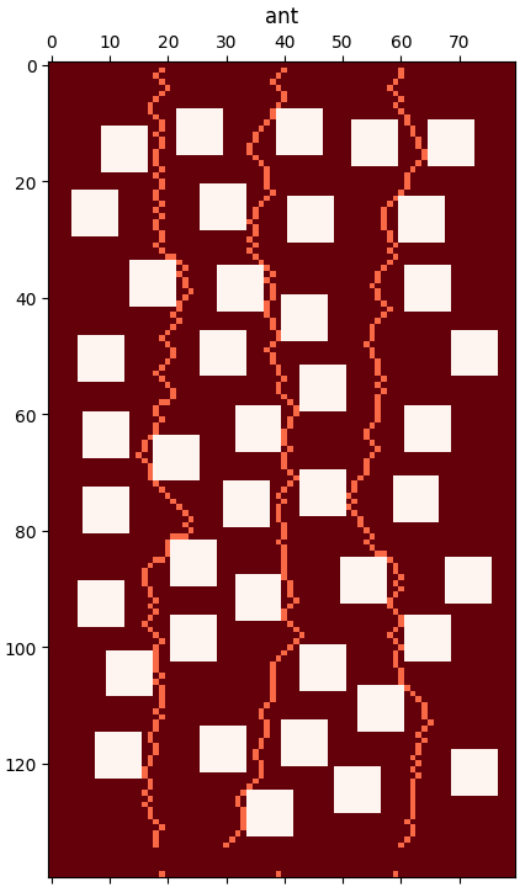

图3 实验效果

3.文件目录

例程目录：
[安装目录]\RflySimAPIs\10.RflySimSwarm\3.CustExps\e3.AISwarmCtrlExp\1.AntAlgorithmMutUAVPathPlan
1.AntAlgorithmMutUAVPathPlan\

表 1 文件目录

文件夹/文件名称	说明
Cylinder&Cylinder.xml	RflySim3D场景障碍物模型文件
Barrier.py	用于随机生成障碍物
CopySceToRflySim3D.bat	场景文件复制脚本
Detect.py	蚂蚁算法的具体实现
main.py	用于启动飞机以及运行蚂蚁算法
PathPlanningSITL.bat	RflySim平台一键启动软件在环仿真脚本
Python38Run.bat	Python环境启动脚本
Readme.pdf	用户指南

4.运行环境

表 2 运行环境

4.1 软件要求

Win 10/Win11系统；RflySim工具链。

> ①：若使用Pixhawk 6X飞控，平台安装时的编译命令为：px4_fmu-v6x_default，推荐PX4固件版本为：1.12.3。其他配套飞控及编译命令请见：https://rflysim.com/doc/zh/1/Hardware.html

4.2 硬件要求

笔记本/台式电脑① 1台。

> ①：推荐配置请见：https://rflysim.com/

1. **：**推荐配置请见：https://doc.rflysim.com/1.1InstallMethod.html

5.实验步骤

5.1 必做实验

Step 1: 障碍物配置

配置障碍物文件，双击运行CopySceToRflySim3D.bat文件，该文件运行过程中会将文件夹Cylinder和文件Cylinder.xml复制到RflySim3D对应的路径下。

图4 障碍物配置

注：本步骤只需在RflySim平台首次运行本例程时进行，后续运行可跳过本步骤。本步骤是将文件夹Cylinder放在..\PX4PSP\RflySim3D\RflySim3D\Plugins\Rfly3DSimPlugin\Content路径下；将Cylinder.xml文件放在..\PX4PSP\RflySim3D\RflySim3D\Plugins\Rfly3DSimPlugin\Content\XML路径下。也可手动进行复制。

Step 2: 仿真初始化

双击运行PathPlanningSITL.bat脚本，观察RflySim3D左上角出现“CopterSim/PX4 EKF 3DFixed：3/3”即表示初始化完成，在RflySim3D中会显示三架飞机。

图5 仿真初始化

Step 3: 启动仿真

在文件夹下，双击Python38Run.bat，打开集成好的python环境，在该环境下运行main.py文件，输入
python
main.py，接着按回车即可启动仿真，等待程序进行迭代，Python38Run弹出“起飞了”，即表示算法迭代完成，同时给出由蚂蚁算法规划出的最佳路径，如下图中的Figure1所示。仿真开始后，即可在RflySim3D中看到三架无人机起飞，并开始避开障碍飞行。

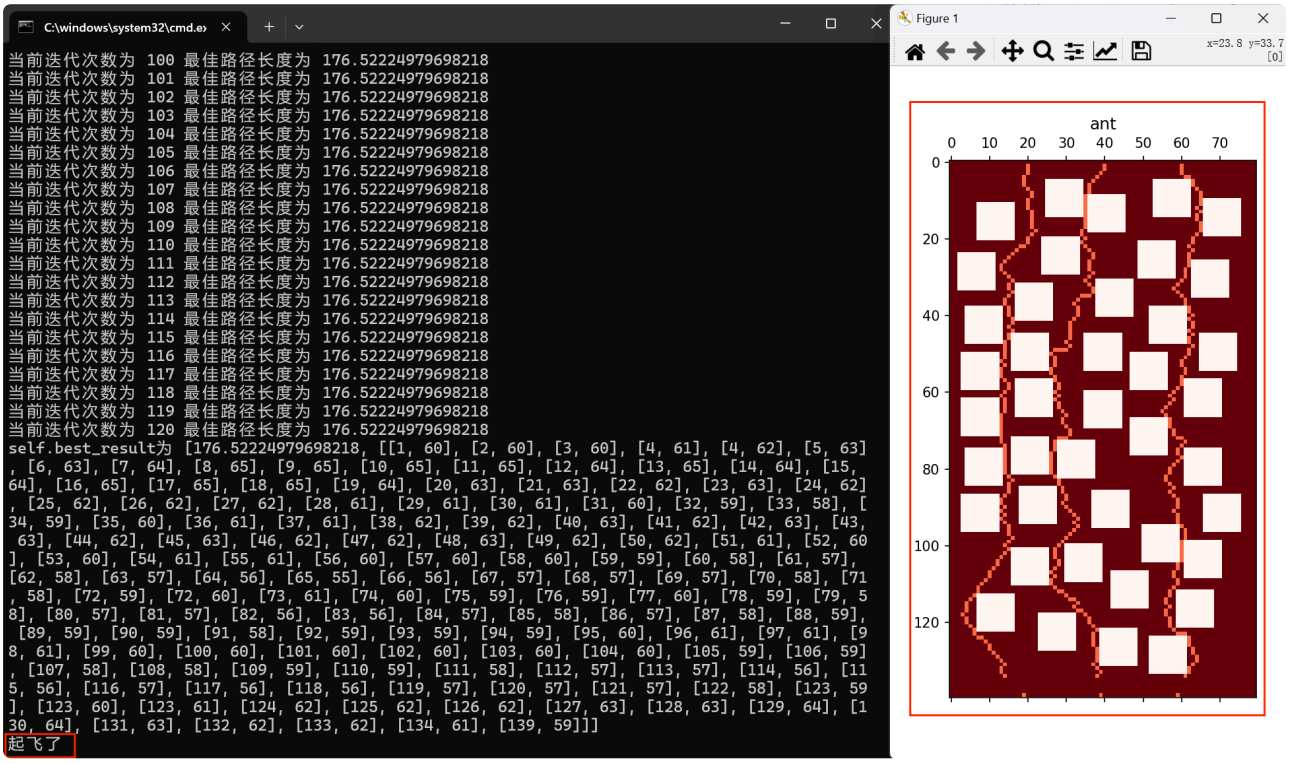

图6启动仿真

Step 4: 实验效果

进入RflySim3D窗口，实验仿真开始后，即可看到RflySim3D中有三架无人机起飞并开始安装蚂蚁算法规划出的最佳路径进行避障飞行。

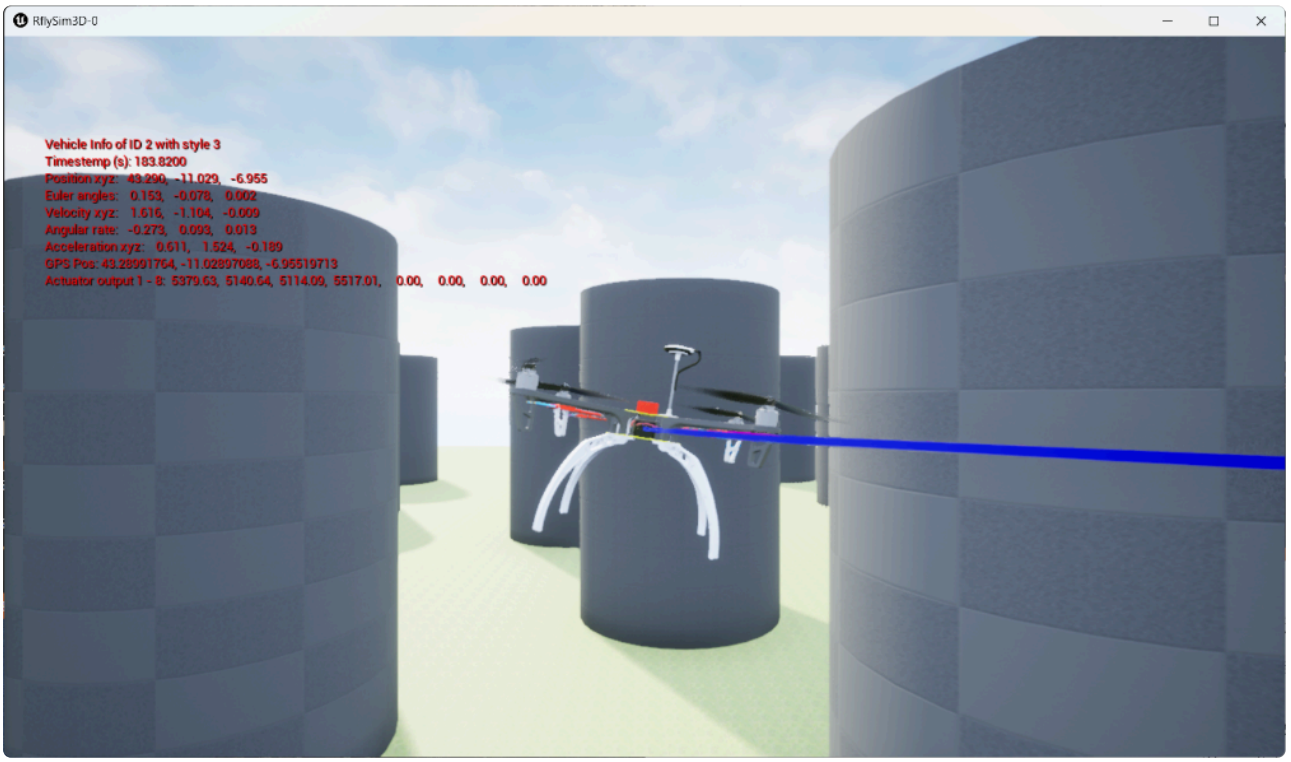

图7 实验效果①

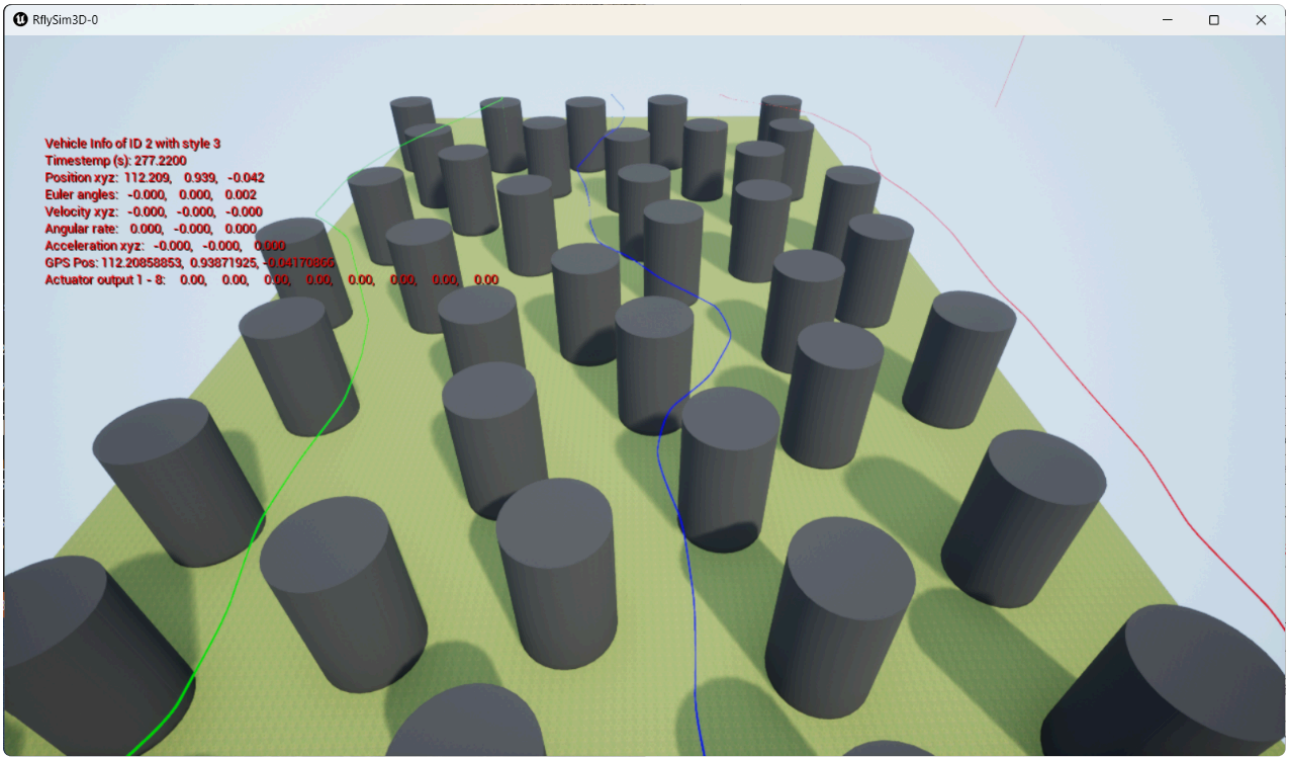

图8 实验效果②

5.2 选做实验（VS Code调试运行）

Step 1: 准备工作

先确保已经按
[RflySim安装目录]/RflySimAPIs/1.RflySimIntro/2.AdvExps/e3.PythonConfig/Readme.pdf
步骤，正确配置VS
Code环境。或者配置了自己的Pycharm等自定义Python环境。

Step 2: VS code调试运行

其他步骤与上文相同，在Step
3启动仿真时，用VScode打开到本实验路径文件夹，运行程序main.py，等待程序进行迭代，VS
code弹出“起飞了”，即表示算法迭代完成，同时给出由蚂蚁算法规划出的最佳路径，如下图中的Figure1所示。仿真开始后，即可在RflySim3D中看到三架无人机起飞，并开始避开障碍飞行。

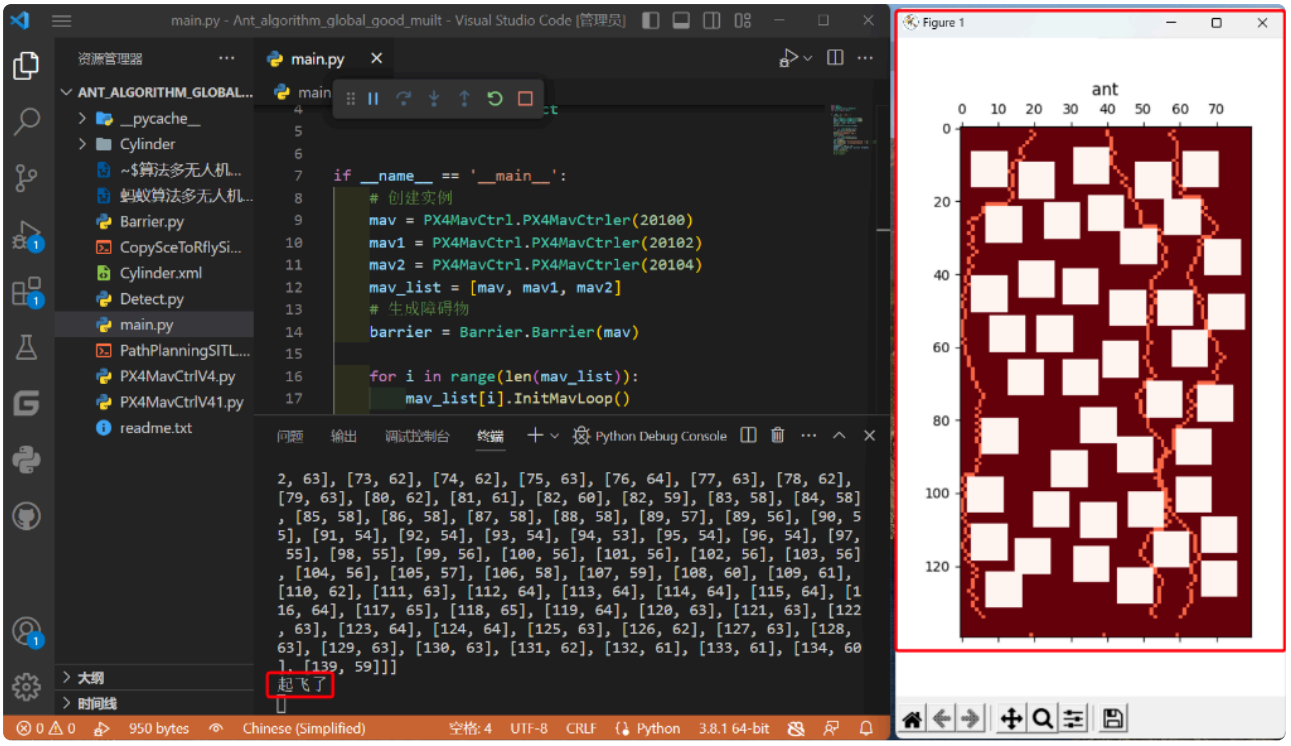

图9 VS code运行示例

6.参考资料

7. 杨爽.群智能优化算法在路径规划中的应用[J].电子技术与软件工程,2019(23):244-245.
8. L. Huan, Z. Ning and L. Qiang, "UAV Path Planning Based on an Improved Ant Colony Algorithm," 2021 4th International Conference on Intelligent Autonomous Systems (ICoIAS), 2021, pp. 357-360, doi: 10.1109/ICoIAS53694.2021.00070.
9. 苏梅梅,程咏梅,胡劲文,赵春晖,贾彩娟,徐钊,张剑锋.基于改进蚁群算法的无人机集群任务分配和路径规划联合优化[J].无人系统技术,2021,4(04):40-50.DOI:10.19942/j.issn.2096-5915.2021.4.035.
10. 刘永建.
基于改进蚁群算法的室内机器人路径规划研究[D].上海工程技术大学,2020.DOI:10.27715/d.cnki.gshgj.2020.000585.

7.常见问题

Q1：路径规划完成，无人机还没有起飞

A1：可能需要断网，进行实验，避免局域网干扰