无人机视觉穿环控制实验

1. 实验目的

本实验旨在帮助学员掌握以下核心技能和知识：

1. **视觉传感器接口使用**：学习使用RflySim平台提供的`VisionCaptureApi`接口从RflySim3D仿真环境中获取RGB图像数据，理解共享内存取图机制和相机参数配置方法。
2. **目标识别算法**：掌握基于OpenCV的两种经典目标检测方法——霍夫圆检测（HoughCircles）用于识别圆环目标，以及HSV颜色空间分割结合多边形逼近（approxPolyDP）用于识别矩形方框目标。
3. **视觉伺服控制**：理解基于图像误差的视觉伺服（Visual Servoing）控制思想，通过将目标在图像中的偏移量映射为飞行速度指令，实现无人机的闭环自主飞行。
4. **多机分布式协同**：了解多机仿真系统的搭建方法，学习如何为每架飞机配置独立的视觉传感器和控制程序，实现分布式视觉穿环任务。

本实验通过RflySim平台的视觉接口，实现无人机基于视觉的自主穿环飞行控制。利用`VisionCaptureApi`接口获取RflySim3D的RGB图像，结合OpenCV进行目标识别（圆环、方框），并通过视觉伺服算法控制无人机依次穿越场景中的彩色圆环和方框。本实验包含三个递进实验：

- **单机穿环**：单架无人机独立完成视觉穿环任务。
- **双机分布式穿环**：两架无人机各自独立运行视觉穿环程序，依次穿环。（仅限完整版及以上版本）
- **三机分布式穿环**：三架无人机各自独立运行视觉穿环程序，依次穿环。（仅限完整版及以上版本）

2. 实验要求

此编写实验目的

- 软件要求：Windows 10及以上版本；RflySim工具链[1]。
- 硬件要求：笔记本/台式电脑1台[2]。

3.实验地址

例程目录：
[安装目录]\RflySimAPIs\8.RflySimVision\1.BasicExps\1-VisionCtrlDemos\e4_CrossRing

文件目录结构说明：

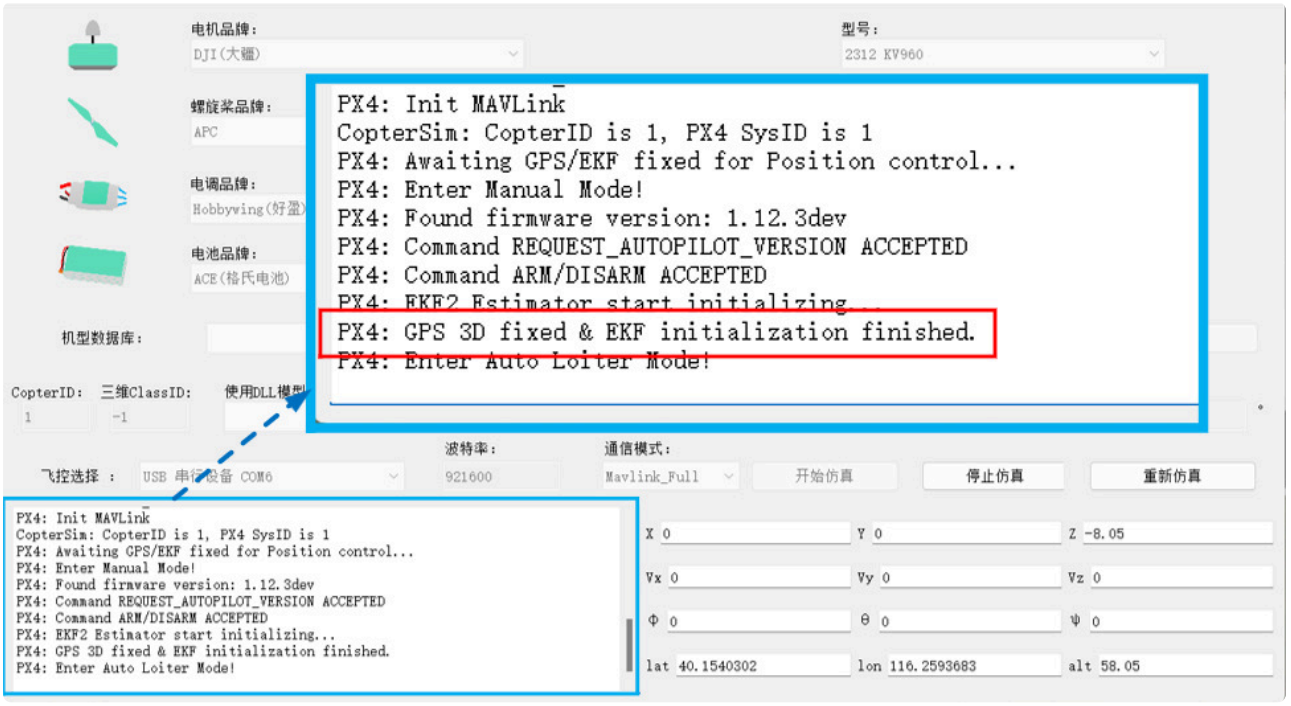
```
├── Config.json                    # 单机视觉传感器配置文件
├── CrossRing3.py                  # 单机视觉穿环主控制程序
├── CrossRing3HITL.bat             # 单机硬件在环仿真启动脚本
├── CrossRing3SITL.bat             # 单机软件在环仿真启动脚本
├── Python38Run.bat                # RflySim Python运行脚本
├── WinWSL.bat                     # Windows WSL启动脚本
├── WslGUI.bat                     # WSL GUI启动脚本
├── TwoUAVDemo/                    # 双机分布式穿环实验文件夹
│   ├── Config1.json               # 1号飞机视觉传感器配置
│   ├── Config2.json               # 2号飞机视觉传感器配置
│   ├── CrossRing2HITL.bat         # 双机硬件在环仿真启动脚本
│   ├── CrossRing2SITL.bat         # 双机软件在环仿真启动脚本
│   ├── CrossRing3_vehicle1.py     # 1号飞机视觉穿环控制程序
│   ├── CrossRing3_vehicle2.py     # 2号飞机视觉穿环控制程序
│   └── Python38Run.bat            # RflySim Python运行脚本
├── ThreeUAVDemo/                  # 三机分布式穿环实验文件夹
│   ├── Config1.json               # 1号飞机视觉传感器配置
│   ├── Config2.json               # 2号飞机视觉传感器配置
│   ├── Config3.json               # 3号飞机视觉传感器配置
│   ├── CrossRing3HITL.bat         # 三机硬件在环仿真启动脚本
│   ├── CrossRing3SITL.bat         # 三机软件在环仿真启动脚本
│   ├── CrossRing3_vehicle1.py     # 1号飞机视觉穿环控制程序
│   ├── CrossRing3_vehicle2.py     # 2号飞机视觉穿环控制程序
│   ├── CrossRing3_vehicle3.py     # 3号飞机视觉穿环控制程序
│   └── Python38Run.bat            # RflySim Python运行脚本
```

4. 实验内容或步骤

5.1.必做实验：单机视觉穿环

Step 1：开启仿真

双击运行"CrossRing3SITL.bat"文件开启软件在环仿真系统。脚本将会启动1个QGC地面站、1个CopterSim、1个RflySim3D软件，等待CopterSim软件下侧日志栏打印出`GPS 3D fixed & EKF initialization finished`字样代表初始化完成。如下图所示：



Step 2：运行控制程序

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 `python CrossRing3.py` 。

Step 3：观察结果

可以看到飞机起飞后，通过视觉识别依次穿越前方的绿色圆环、蓝色圆环和红色方框，完成穿环任务。

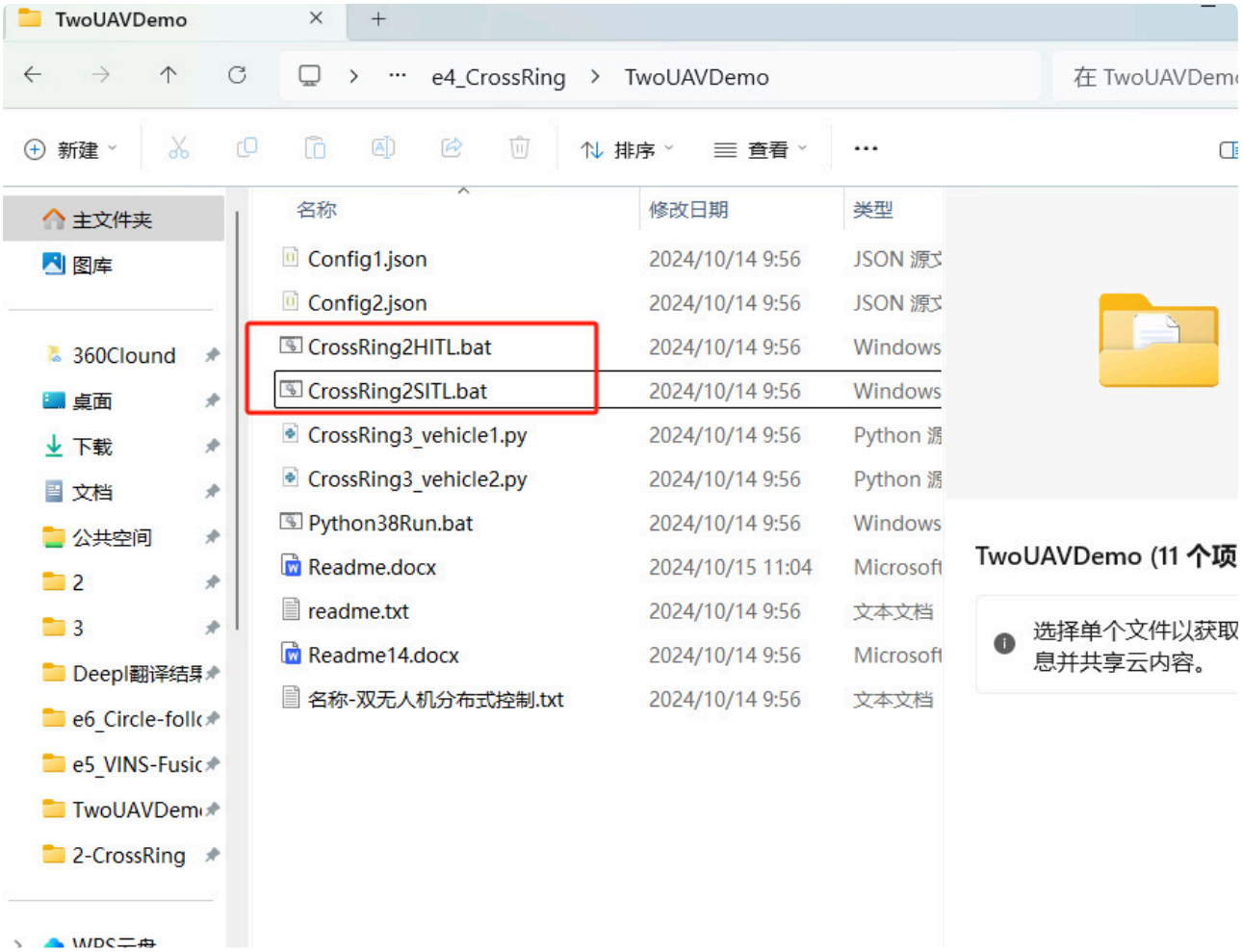
Step 4：结束仿真

在"CrossRing3SITL.bat"脚本开启的命令提示符CMD窗口中，按下回车键（任意键）就能快速关闭CopterSim、QGC、RflySim3D等所有程序。

5.2.选做实验A：双无人机分布式穿环（仅限完整版及以上版本）

Step 1：开启仿真

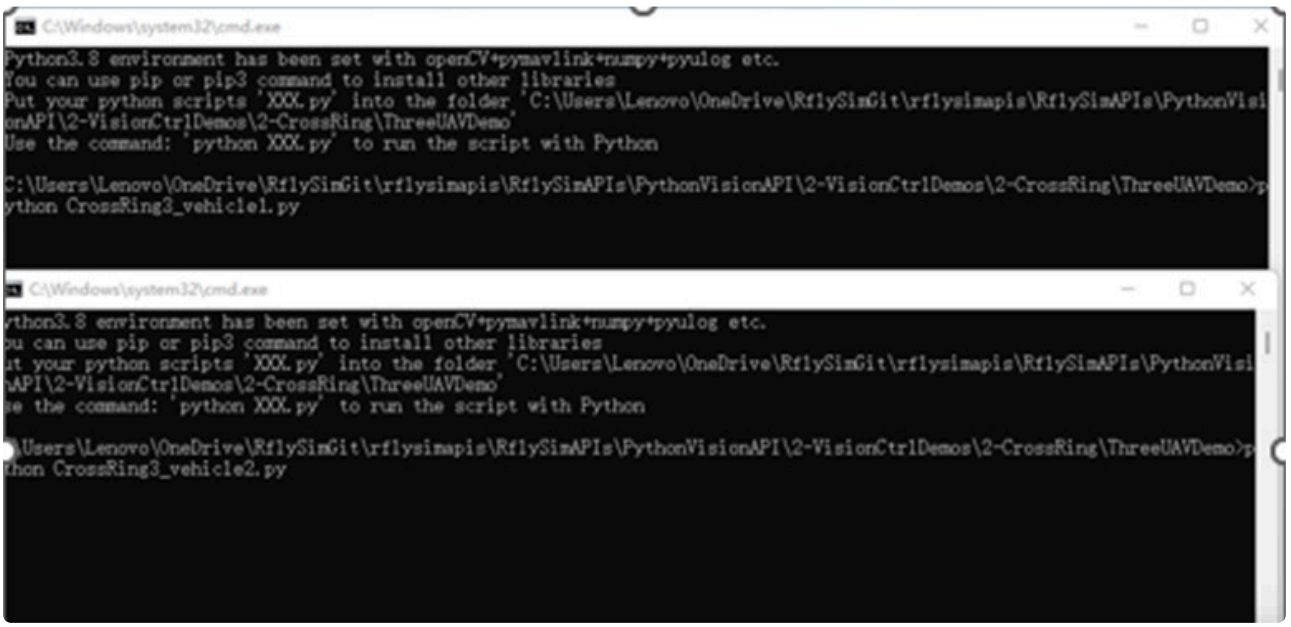
进入 TwoUAVDemo 文件夹，双击运行"CrossRing2SITL.bat"文件开启双机软件在环仿真系统。脚本将会启动1个QGC地面站、2个CopterSim、1个RflySim3D软件，等待所有CopterSim初始化完成。



Step 2：启动控制程序

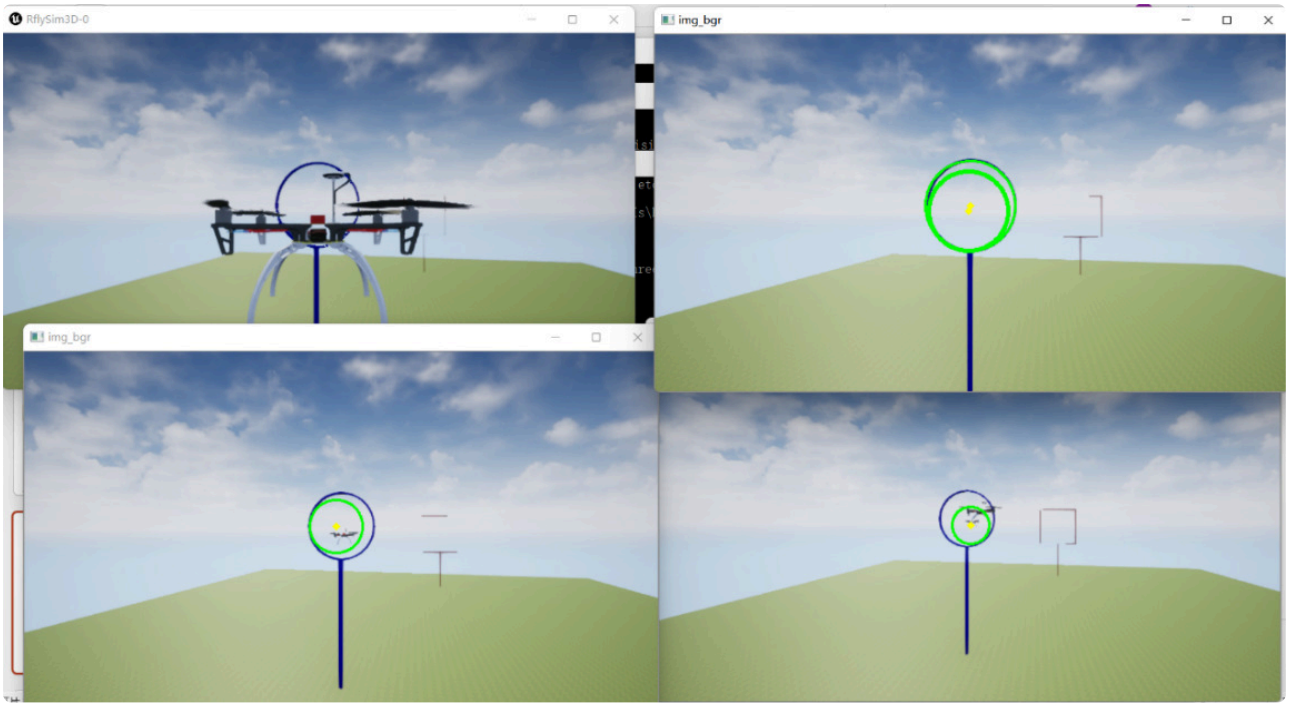
等待两架飞机初始化完毕后（RflySim3D会提示），双击"Python38Run.bat"两次，打开两个Python环境窗口。分别在两个窗口中输入以下命令（先不按回车）：

- 第一个窗口：`python CrossRing3_vehicle1.py`
- 第二个窗口：`python CrossRing3_vehicle2.py`



Step 3：依次运行

回到第一个Python窗口，按下回车键运行1号飞机的视觉穿环程序；间隔几秒钟后，切换到第二个窗口按下回车键。可看到两架飞机依次起飞并穿环。



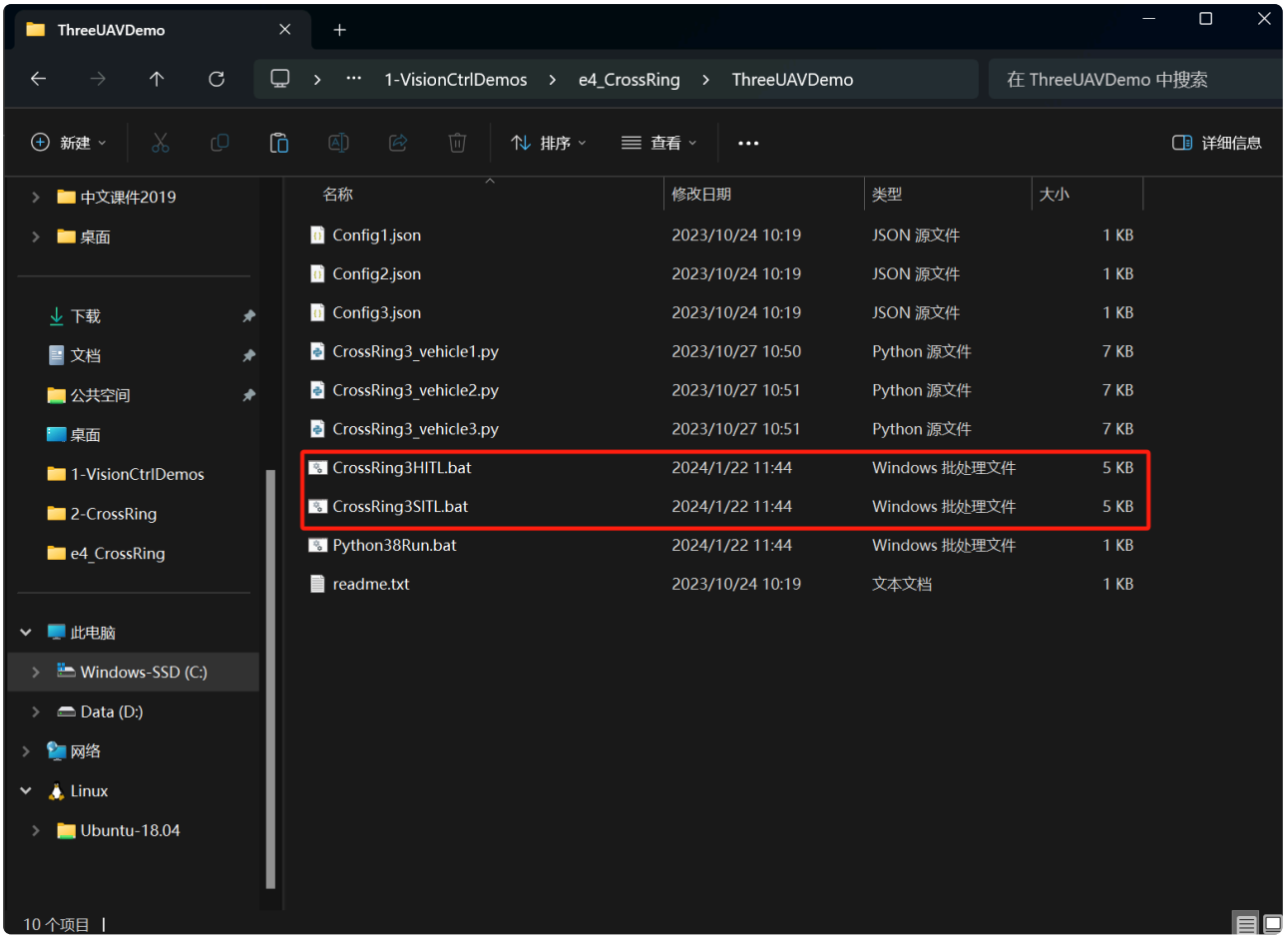
Step 4：结束仿真

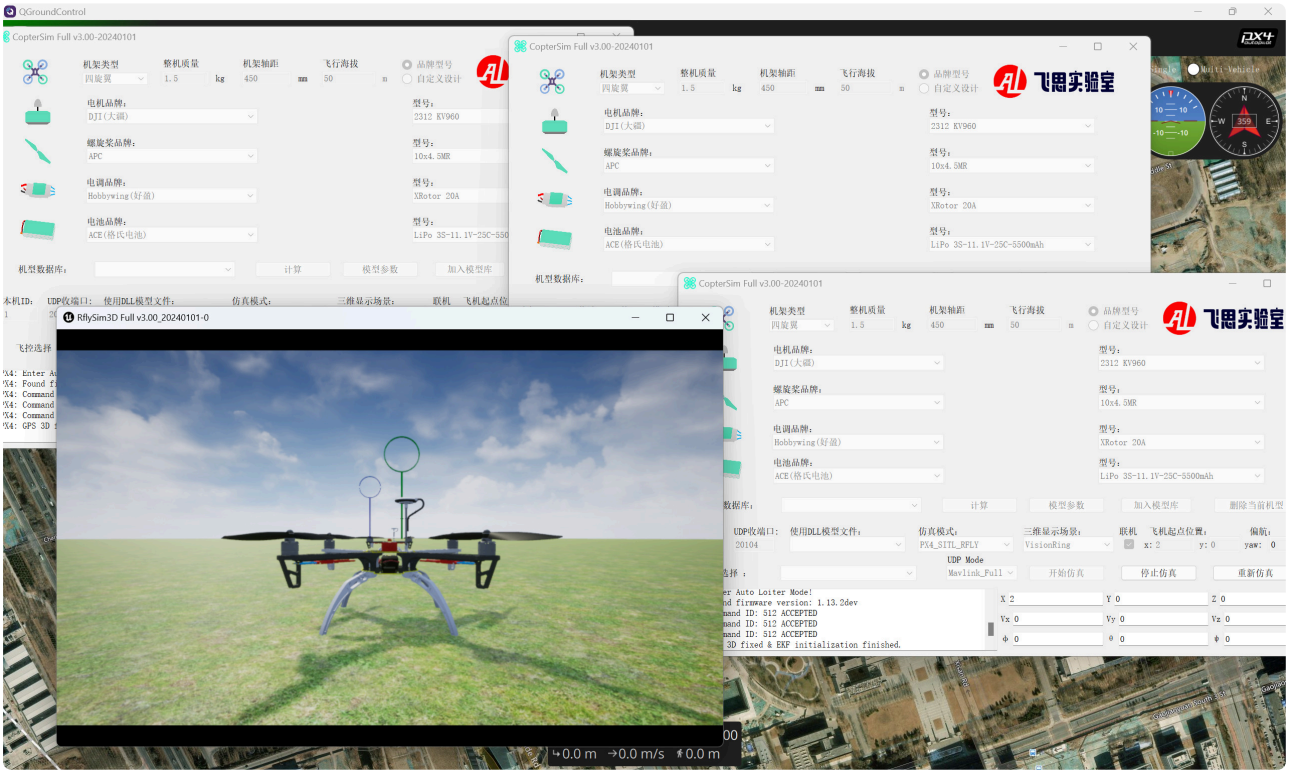
在"CrossRing2SITL.bat"脚本开启的命令提示符CMD窗口中，按下回车键（任意键）就能快速关闭所有程序。

5.3.选做实验B：三无人机分布式穿环（仅限完整版及以上版本）

Step 1：开启仿真

进入 ThreeUAVDemo 文件夹，双击运行"CrossRing3SITL.bat"文件开启三机软件在环仿真系统。脚本将会启动1个QGC地面站、3个CopterSim、1个RflySim3D软件，等待所有CopterSim初始化完成。

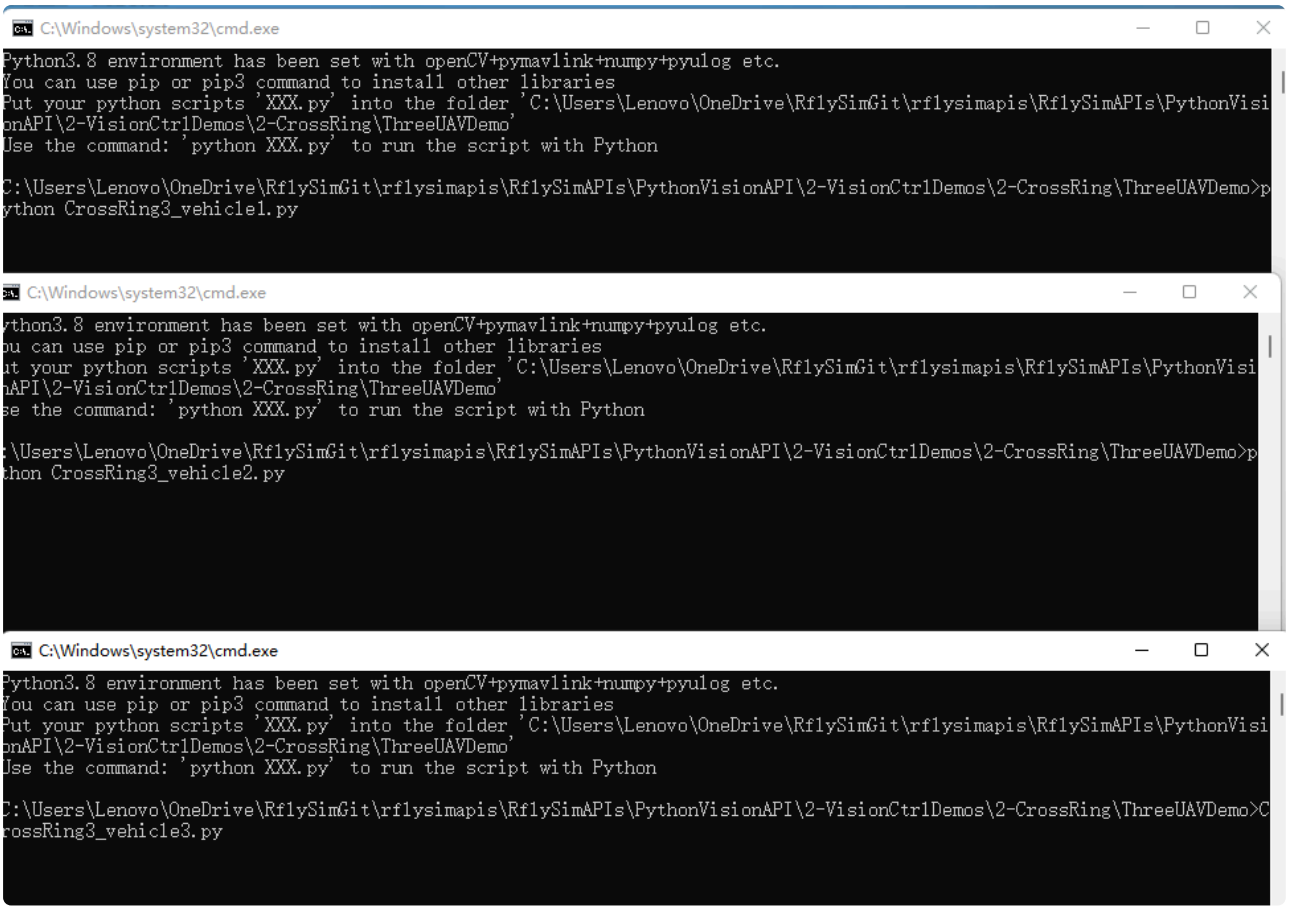




Step 2：启动控制程序

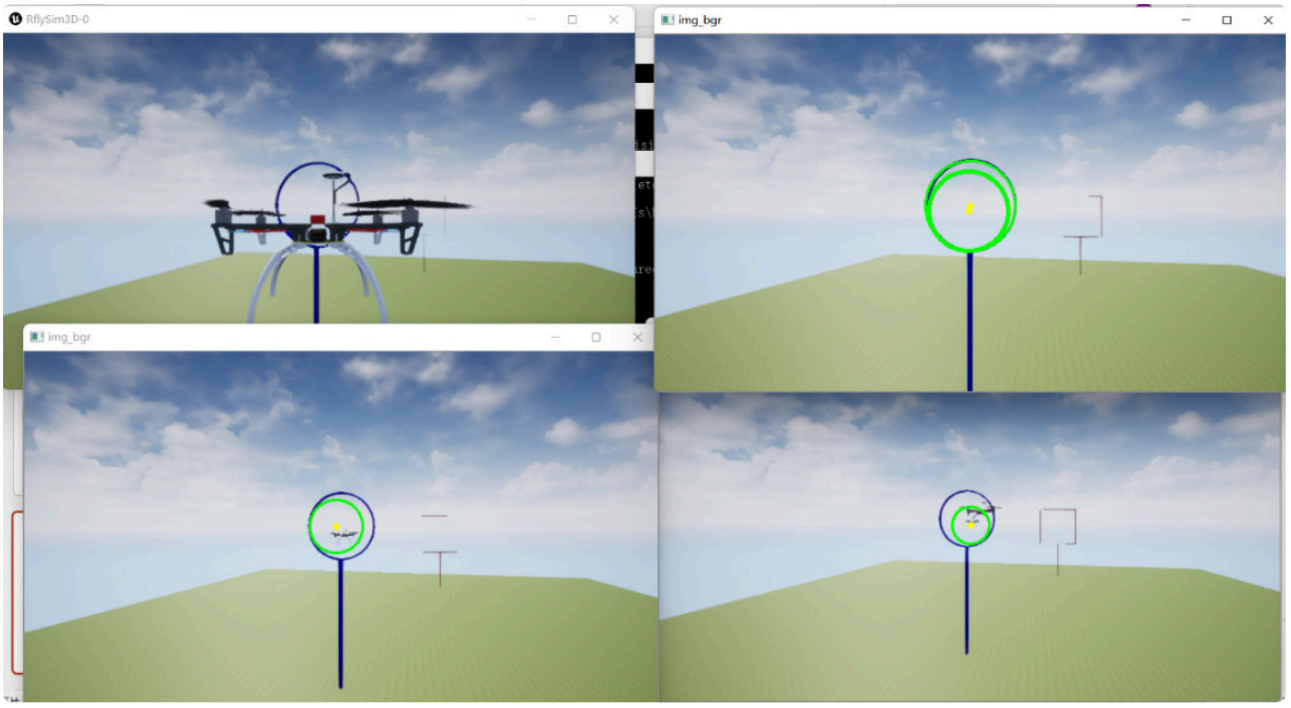
等待三架飞机初始化完毕后（RflySim3D会提示），双击"Python38Run.bat"三次，打开三个Python环境窗口。分别在三个窗口中输入以下命令（先不按回车）：

- 第一个窗口：`python CrossRing3_vehicle1.py`
- 第二个窗口：`python CrossRing3_vehicle2.py`
- 第三个窗口：`python CrossRing3_vehicle3.py`



Step 3：依次运行

回到第一个Python窗口，按下回车键运行1号飞机的视觉穿环程序；间隔几秒钟后，切换到第二个窗口按下回车键；再间隔几秒钟后，切换到第三个窗口按下回车键。可看到三架飞机依次起飞并穿环。



> **注意**：由于前面飞机会遮挡圆环，这种模式下的圆环识别可能存在波动，会降低控制效果。建议启动各飞机时适当延长间隔（5~10秒）。

Step 4：结束仿真

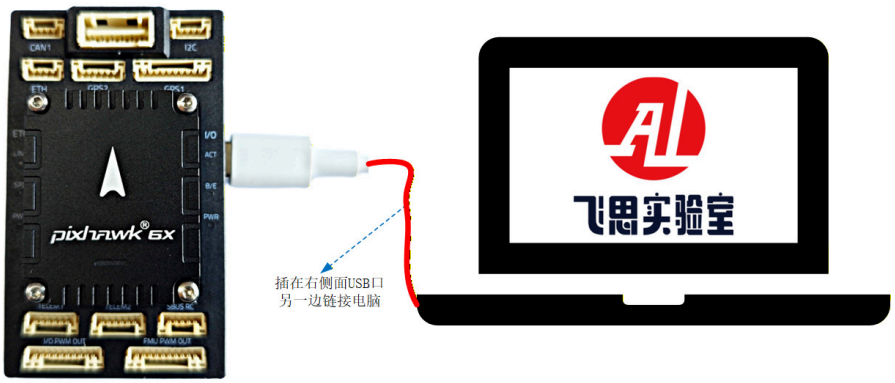
在"CrossRing3SITL.bat"脚本开启的命令提示符CMD窗口中，按下回车键（任意键）就能快速关闭所有程序。

5.4.选做实验：硬件在环仿真

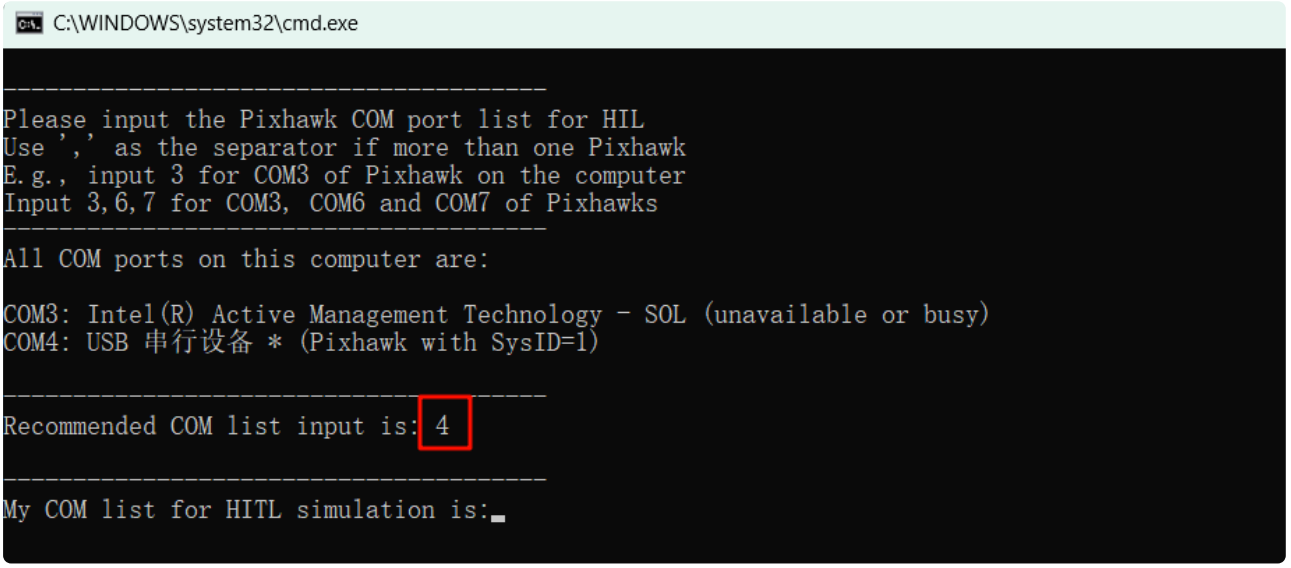
以上所有实验均支持硬件在环仿真，仅需将对应的SITL脚本替换为HITL脚本即可：

- 单机：将"CrossRing3SITL.bat"替换为"CrossRing3HITL.bat"
- 双机：将"CrossRing2SITL.bat"替换为"CrossRing2HITL.bat"
- 三机：将"CrossRing3SITL.bat"替换为"CrossRing3HITL.bat"

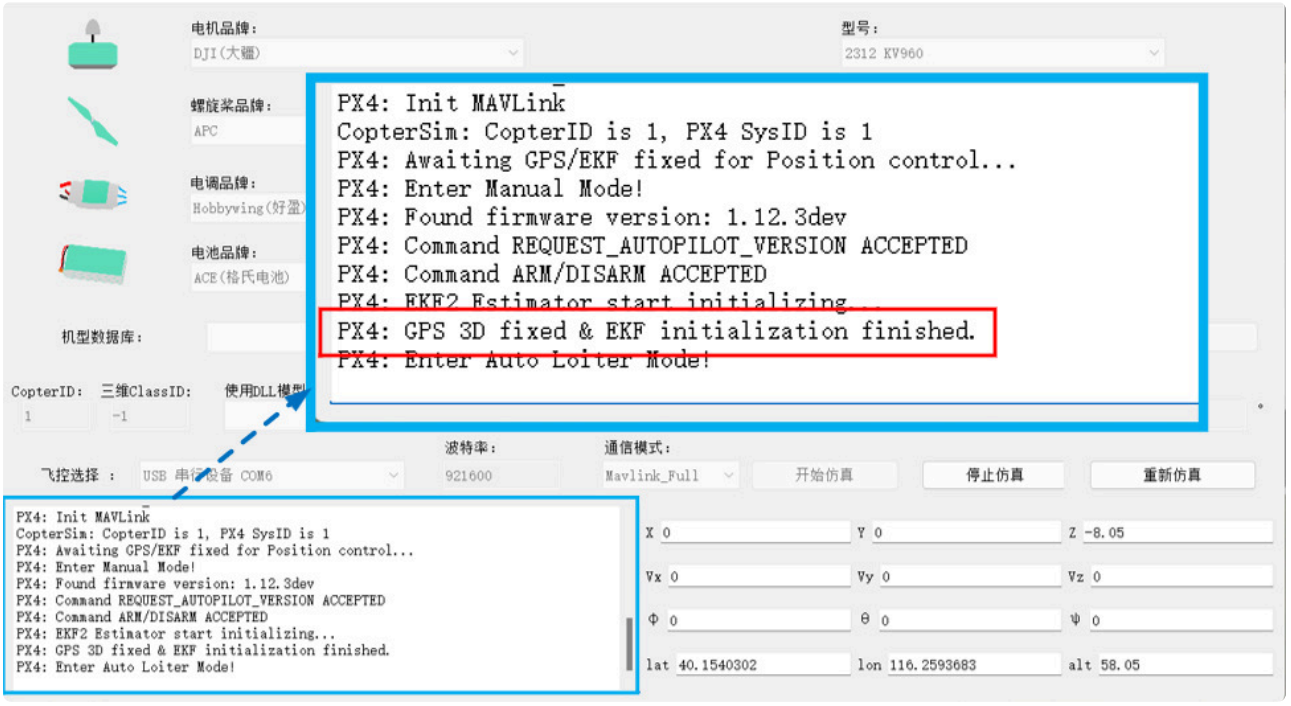
本实验在进行硬件在环仿真之前需要将飞控内部固件进行还原并需确保可正常进行硬件在环仿真，具体还原方式可见例程：`[RflySim安装路径]\RflySimAPIs\1.RflySimIntro\2.AdvExps\e2.FCUIntro`。按照如下图所示，进行硬件连接并接入电脑。



双击运行HITL脚本文件，在弹出的对话框中输入飞控COM号（如：4）。



等待仿真环境初始化完成，CopterSim软件下侧日志栏打印出 `GPS 3D fixed & EKF initialization finished` 字样代表初始化完成。如下图所示：



其余步骤与软件在环仿真步骤一致。

5.5.选做实验：VS Code调试运行

准备工作：

- 先确保已经按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。
- 其他步骤与上文相同，在运行Python脚本时，可使用VS Code（或Pycharm等工具）来打开对应的.py文件，并阅读代码，修改代码，调试执行等。

扩展实验：

- 请自行使用VS Code阅读CrossRing3.py源码，通过程序跳转，了解每条代码的执行原理；再通过调试工具，验证每条指令的执行效果。
- 请尝试修改代码中的控制增益参数（K_z、K_yawrate），观察对穿环控制精度的影响。
- 请尝试修改`taskChange`函数中的距离阈值，观察任务切换时机对穿环效果的影响。

5. 关键知识点

整体思路与框架

本实验的核心思路是**「感知—决策—控制」闭环**：

1. **感知层**：通过`VisionCaptureApi`从RflySim3D获取前置摄像头的RGB图像。
2. **决策层**：根据无人机当前NED坐标的X分量（前向位置），通过`taskChange()`函数将飞行过程划分为多个阶段（range1→range2→range3→land→finish），每个阶段对应不同的目标类型和检测算法。
3. **控制层**：通过视觉伺服算法，将目标在图像中的偏移量（像素误差）映射为机体坐标系下的速度指令（前向速度vx、垂直速度vz、偏航角速度yawrate），发送给PX4飞控实现自主飞行。

程序的整体执行流程如下：

```
初始化视觉传感器和飞控连接
    ↓
起飞并飞向初始位置 (5, 0, -5)
    ↓
进入穿环主循环（30Hz）
    ├── 获取图像 → 目标检测 → 计算图像误差
    ├── 读取飞机位置 → 判断任务阶段
    └── 计算控制指令 → 发送速度指令
    ↓
到达降落区域 → 降落 → 结束
```

关键代码解析

以下基于单机穿环主程序 `CrossRing3.py` 进行详细解析。

1）初始化与视觉接口使用

程序首先导入所需库并初始化三个核心模块：UE4控制、视觉传感器、飞机控制。

```python
import PX4MavCtrlV4 as PX4MavCtrl
import VisionCaptureApi
import UE4CtrlAPI

# 创建UE4控制实例，用于设置仿真环境参数
ue = UE4CtrlAPI.UE4CtrlAPI()

# 创建视觉传感器实例
vis = VisionCaptureApi.VisionCaptureApi()
vis.jsonLoad()  # 加载Config.json中的传感器配置文件
isSuss = vis.sendReqToUE4()  # 向RflySim3D发送取图请求，并验证
if not isSuss:  # 如果请求取图失败，则退出
    sys.exit(0)
vis.startImgCap(True)  # 开启取图，并启用共享内存图像转发
```

关键属性说明：

- `vis.hasData[0]`：布尔值，标志位表示索引0的传感器是否有新的图像数据更新。
- `vis.Img[0]`：NumPy数组，存储索引0传感器的最新一帧BGR图像数据。

UE4环境设置：

```python
ue.sendUE4Cmd('r.setres 720x405w',0)  # 设置UE4窗口分辨率（仅影响显示窗口，不影响取图）
ue.sendUE4Cmd('t.MaxFPS 30',0)  # 设置UE4最大刷新频率为30Hz，即取图频率上限
```

2）相机参数配置（Config.json）

视觉传感器的参数由`Config.json`配置文件决定，主要字段含义如下：

```
{
    "VisionSensors":[{
        "SeqID": 0,                             // 传感器编号
        "TypeID": 1,                            // 传感器类型，1=RGB彩色图像
        "TargetCopter": 1,                      // 绑定到1号飞机
        "DataWidth": 720, "DataHeight": 405,    // 图像分辨率 720×405
        "DataCheckFreq": 30,                    // 数据检查频率 30Hz
        "SendProtocol": [0,127,0,0,1,9999,0,0], // 传输协议，首位0表示共享内存模式
        "CameraFOV": 90,                        // 相机视场角 90度
        "SensorPosXYZ": [0.3, 0, 0],            // 相机安装位置（机体坐标系，前方0.3m）
        "SensorAngEular": [0, 0, 0]             // 相机安装角度（欧拉角，正前方）
    }]
}
```

注意：多机实验中每架飞机使用独立的Config文件（Config1.json、Config2.json、Config3.json），其中`TargetCopter`字段需分别设置为1、2、3以对应各自的飞机编号。多机程序中加载方式也有所不同：

```
vis.jsonLoad(0, 'Config1.json')  # 使用共享内存方式，加载1号飞机配置
```

3）任务阶段划分

`taskChange()`函数根据无人机在NED坐标系中的X坐标（前向位置）将飞行过程划分为5个阶段：

```
def taskChange(pos_x):
    if pos_x < 40:
        task = "range1"    # 阶段1：识别并穿越绿色圆环
    elif pos_x < 70:
        task = "range2"    # 阶段2：识别并穿越蓝色圆环
    elif pos_x < 130:
        task = "range3"    # 阶段3：识别并穿越红色方框
    elif pos_x < 140:
        task = "land"      # 阶段4：减速降落
    else:
        task = "finish"    # 阶段5：任务完成
    return task
```

该设计使得不同阶段使用不同的检测算法，既提高了识别精度，又减少了无关目标的干扰。

4）目标识别算法

`objectDetect()` 函数是目标检测的调度器，根据当前任务阶段选择对应的检测策略：

```
def objectDetect(task):
    if vis.hasData[0]:
        img_bgr = vis.Img[0]   # 获取最新一帧图像
    else:
        return -1, -1, -1      # 无新数据返回无效值

    if task == "range1":
        b, g, r = cv2.split(img_bgr)
        return circleDetect(img_bgr, g)    # 提取绿色通道，检测绿色圆环
    elif task == "range2":
        b, g, r = cv2.split(img_bgr)
        return circleDetect(img_bgr, b)    # 提取蓝色通道，检测蓝色圆环
    else:
        # range3阶段：使用HSV颜色空间检测红色方框
        hsv = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2HSV)
        # 红色在HSV中跨越0度，需要两个范围
        lower_red1 = np.array([0, 25, 25])
        upper_red1 = np.array([10, 255, 255])
        lower_red2 = np.array([170, 25, 25])
        upper_red2 = np.array([180, 255, 255])
        mask = cv2.inRange(hsv, lower_red1, upper_red1) + cv2.inRange(hsv,
lower_red2, upper_red2)
        # 形态学膨胀，将细线框变粗以便稳定检测
        kernel = np.ones((5, 5), np.uint8)
        mask_dilated = cv2.dilate(mask, kernel, iterations=2)
        return squareDetect(img_bgr, mask_dilated)
```

圆环检测（霍夫圆检测）：

```python
def circleDetect(img_bgr, img_b):
    """使用霍夫变换检测圆形目标"""
    circles = cv2.HoughCircles(
        img_b, cv2.HOUGH_GRADIENT, 1, 20,
        param1=100, param2=30,      # Canny边缘检测阈值和累加器阈值
        minRadius=0, maxRadius=0    # 不限制半径范围
    )
    if circles is not None:
        circles = np.uint16(np.around(circles))
        obj = circles[0, 0]  # 取第一个检测到的圆
        # 在图像上绘制检测结果（绿色圆环和黄色圆心）
        cv2.circle(img_bgr, (obj[0], obj[1]), obj[2], (0, 255, 0), 2)
        cv2.circle(img_bgr, (obj[0], obj[1]), 2, (0, 255, 255), 3)
        height, width, channel = img_bgr.shape
        # 返回目标中心相对于图像中心的偏移量(ex, ey)和半径r
        return obj[0] - width/2, obj[1] - height/2, obj[2]
    else:
        return -1, -1, -1
```

方框检测（轮廓检测 + 多边形逼近）：

```python
def squareDetect(img_bgr, img_edge):
    """使用多边形逼近和对角线校验检测方形目标"""
    squares = []
    cnts, hierarchy = cv2.findContours(img_edge, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)
    for cnt in cnts:
        cnt_len = cv2.arcLength(cnt, True)
        cnt = cv2.approxPolyDP(cnt, 0.05 * cnt_len, True)  # 多边形逼近
        # 筛选条件：4个顶点、面积>2000像素、凸多边形
        if len(cnt) == 4 and cv2.contourArea(cnt) > 2000 and
cv2.isContourConvex(cnt):
            cnt = cnt.reshape(-1, 2)
            diag_delta = diagonal_check(cnt)  # 对角线长度差异校验
            if diag_delta < 0.2:  # 两条对角线长度差异小于20%
                squares.append(cnt)
    if squares:
        # 返回方框中心相对图像中心的偏移和对角线长度
        return (squares[0][0][0]+squares[0][2][0])/2 - width/2, \
               (squares[0][0][1]+squares[0][2][1])/2 - height/2, \
               np.sqrt(np.dot(squares[0][0]-squares[0][2], squares[0][0]-squares[0]
[2]))
    else:
        return -1, -1, -1
```

其中`diagonal_check()`辅助函数通过对比四边形两条对角线的长度差异来过滤非正方形的误检：

```python
def diagonal_check(p):
    d1 = np.sqrt(np.dot(p[0]-p[2], p[0]-p[2]))  # 对角线1长度
    d2 = np.sqrt(np.dot(p[1]-p[3], p[1]-p[3]))  # 对角线2长度
    return abs(d1-d2)*1.0/d1  # 返回相对差异
```

5）视觉伺服控制算法

`approachObjective()`是穿环任务的核心控制循环，以30Hz频率运行，实现感知-控制闭环：

```python
def approachObjective():
    # 控制增益参数（根据图像分辨率自适应缩放）
    K_z = 0.004 * 640 / height        # 垂直方向控制增益
    K_yawrate = 0.005 * 480 / width   # 偏航方向控制增益
    task = "range1"
    startAppTime = time.time()
    lastTime = time.time()
    timeInterval = 1/30.0    # 控制周期 33.3ms (30Hz)

    while (task != "finish") & (task != "land"):
        # 定时器逻辑，确保稳定的30Hz控制频率
        lastTime = lastTime + timeInterval
        sleepTime = lastTime - time.time()
        if sleepTime > 0:
            time.sleep(sleepTime)
        else:
            lastTime = time.time()

        # 步骤1：目标检测，获取图像误差 (ex, ey) 和目标尺寸 r
        ex, ey, r = objectDetect(task)
        # 步骤2：获取无人机当前NED位置
        pos_x = mav.uavPosNED[0]
        # 步骤3：根据位置更新任务阶段
        task = taskChange(pos_x)
        # 步骤4：若检测到目标，计算并发送控制指令
        if ex != -1:
            vx = sat(time.time() - startAppTime, 3)    # 前向速度，带饱和限幅，最大3m/s
            vy = 0                                     # 侧向不控制
            vz = K_z * ey                              # 垂直速度 = 增益 × 垂直像素误差
            yawrate = K_yawrate * ex                   # 偏航角速度 = 增益 × 水平像素误差
            mav.SendVelFRD(vx, vy, vz, yawrate)        # 发送FRD机体系速度指令
```

控制逻辑核心思想：

- **前向速度vx**：使用`sat()`饱和函数，随时间线性增加但不超过3m/s，确保飞机匀速前进。

- **垂直速度vz**：与目标在图像中的纵向偏移成正比，目标偏上则下降，偏下则上升。
- **偏航角速度yawrate**：与目标在图像中的横向偏移成正比，目标偏左则左转，偏右则右转。
- 当目标未被检测到（ex=-1）时，不发送新指令，飞机保持上一次的运动状态。

其中`sat()`饱和限幅函数定义如下：

```
def sat(inPwm, thres=1):
    outPwm = inPwm
    if inPwm > thres:
        outPwm = thres
    elif inPwm < -thres:
        outPwm = -thres
    return outPwm
```

6）主程序流程

主函数orchestrates整个飞行任务：

```
if __name__ == '__main__':
    mav = PX4MavCtrl.PX4MavCtrler(1)  # 创建飞机控制实例（连接1号飞机）
    mav.InitMavLoop()                 # 初始化Mavlink监听循环

    print("Enter Offboard mode.")
    time.sleep(5)                      # 等待系统就绪
    mav.initOffboard()                # 进入Offboard控制模式
    time.sleep(0.5)
    mav.SendMavArm(True)              # 解锁飞控电机
    mav.SendPosNED(5, 0, -5, 0)       # 飞到起始位置 (NED: x=5m, y=0, z=-5m即高度5m)

    time.sleep(5)                      # 等待飞机到达起始位置
    approachObjective()               # 开始穿环任务
```

7）多机扩展说明

双机和三机实验的Python程序（如`CrossRing3_vehicle1.py`、`CrossRing3_vehicle2.py`等）与单机程序结构完全一致，主要区别在于：

1. **配置文件加载**：使用独立的配置文件（Config1.json、Config2.json等），并通过指定文件名加载：

```
vis.jsonLoad(0, 'Config1.json')  # 加载1号飞机的传感器配置
mav = PX4MavCtrl.PX4MavCtrler(1) # 连接1号飞机的Mavlink端口
```

2. **飞机编号对应**：每个程序中的`PX4MavCtrler(n)`和Config文件中的`TargetCopter:n`必须一致，确保视觉传感器和控制指令对应到正确的飞机。
3. **独立运行**：各飞机的程序在独立的Python进程中运行，互不干扰，实现分布式控制架构。

6.参考资料

1. RflySim官方文档： https://rflysim.com/doc/zh/
2. OpenCV官方文档 - 霍夫圆检测：
   https://docs.opencv.org/4.x/da/d53/tutorial_py_houghcircles.html
3. OpenCV官方文档 - 轮廓检测与多边形逼近：
   https://docs.opencv.org/4.x/d4/d73/tutorial_py_contours_begin.html
4. OpenCV官方文档 - HSV颜色空间：
   https://docs.opencv.org/4.x/df/d9d/tutorial_py_colorspaces.html
5. PX4官方文档 - Offboard模式：
   https://docs.px4.io/main/en/flight_modes/offboard.html

7.常见问题

Q1：运行CrossRing3.py后，OpenCV窗口显示黑屏或无图像输出。

A1：该问题通常由视觉传感器未正确连接到RflySim3D引起。请检查以下几点：

- 确认RflySim3D已正常启动，且CopterSim已输出`GPS 3D fixed & EKF initialization finished`日志。
- 确认Config.json中的`SendProtocol`首位为0（共享内存模式），且`DataWidth`和`DataHeight`与程序中设定的`width`和`height`一致（默认720×405）。
- 确认`vis.sendReqToUE4()`返回True，若返回False说明取图请求失败，需检查RflySim3D是否正常运行。

Q2：飞机起飞后无法正确穿越圆环，出现偏航震荡或飞行轨迹不稳定。

A2：该问题通常与控制增益参数设置不当有关。可尝试以下调整：

- 降低偏航控制增益`K_yawrate`（如从0.005调整为0.003），减小偏航方向的响应灵敏度。
- 降低垂直控制增益`K_z`（如从0.004调整为0.002），使垂直方向运动更加平稳。
- 检查`saturationYawRate()`函数的限幅范围`yr_bound`，适当减小以限制最大偏航角速度。

Q3：多机实验中，某架飞机无法获取图像或控制指令发送到错误的飞机。

A3：多机实验中每架飞机必须使用独立的配置文件和控制端口，请检查：

- 各Config文件（Config1.json、Config2.json、Config3.json）中的`TargetCopter`字段是否分别填写了正确的飞机编号（1、2、3）。
- 各Python程序中`PX4MavCtrler(n)`的参数`n`是否与对应Config文件中的`TargetCopter`一致。
- 各Python程序中`vis.jsonLoad(0, 'ConfigN.json')`是否加载了正确的配置文件。

Q4：在三机穿环实验中，后方飞机穿环失败或识别不稳定。

A4：这是由于前方飞机遮挡圆环导致的视觉识别干扰。建议：

- 增大各飞机启动的时间间隔（建议5~10秒），让前方飞机先穿过圆环后再启动后方飞机。
- 观察OpenCV显示窗口确认当前识别状态，若频繁出现误检测（检测框闪烁不稳定），可适当等待更长时间。

Q5：运行HITL（硬件在环）脚本时，提示COM端口无法连接或飞控无响应。

A5：请按以下步骤排查：

- 确认飞控已通过USB线正确连接到电脑，并在设备管理器中确认COM端口号。
- 在运行HITL脚本弹出的对话框中，输入正确的COM端口号（仅输入数字，如4）。
- 确认飞控固件已按照 `[RflySim安装路径]\RflySimAPIs\1.RflySimIntro\2.AdvExps\e2.FCUIntro` 中的说明还原为默认固件。
- 确保没有其他软件（如QGC）占用该COM端口。

---

1. https://rflysim.com/ ↩
2. 推荐配置请见： https://rflysim.com/doc/zh/HowToInstall.pdf ↩