运行python程序

1. 实验目的

通过一个具体案例，带学习者深入探索python的基础语法，数据结构，控制流，函数，库使用等核心知识

2. 实验要求

- 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\1.RflySimIntro\1.BasicExps\e7_RunPythonProject

- ./ShootBall3.py：主要控制程序，实现了视觉追踪红色球体的完整流程，包括图像处理、目标识别和无人机控制。
- ./Python38Run.bat：Python 3.8运行环境批处理文件，用于启动预配置的Python环境。
- ./ShootBall3SITL.bat：软件在环(SITL)仿真启动脚本，启动仿真环境。
- ./ShootBall3HITL.bat：硬件在环(HITL)仿真启动脚本，用于连接实际飞控硬件。

4. 实验内容或步骤

无人机跟踪小球实验

4.1 步骤1：开启仿真

双击运行"ShootBall3SITL.bat"文件开启软件在环仿真系统。也可插入飞控，并运行硬件在环仿真脚本"ShootBall3HITL.bat"，输入串口号来开启HITL仿真。

4.2 步骤2：设置搜索路径

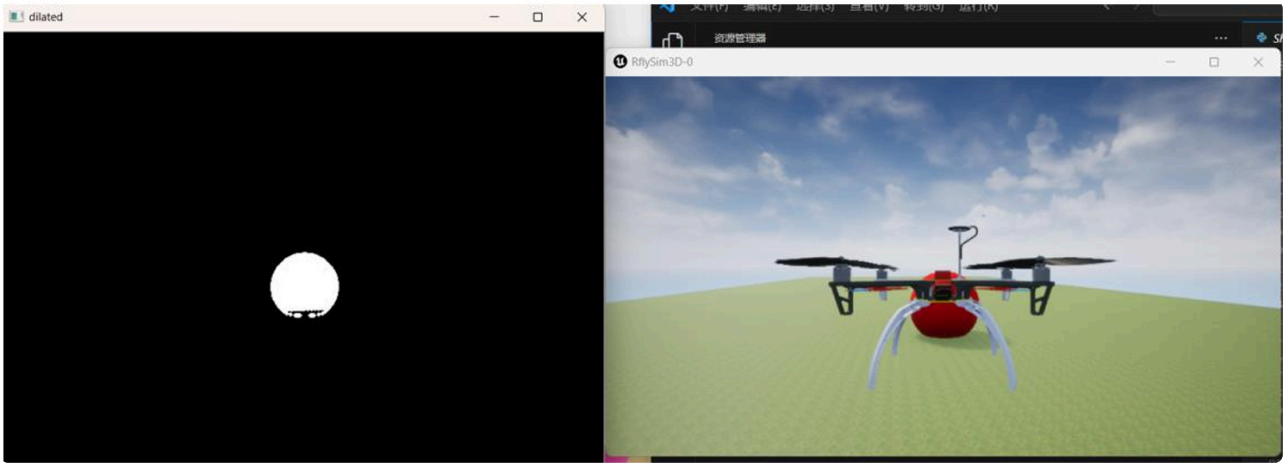
运行 `PX4PSPRflySimAPIs\RflySimSDK` 目录下的 `ReLabPath.py` 文件。

4.3 步骤3：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 ShootBall3.py

4.4 步骤4：观察结果

可以看到场景切换到草地场景，生成了一个多旋翼飞机，以及一个红球。飞机飞到靠左后方一段距离，并开启视觉跟踪，飞到小球面前停止。



4.5 步骤5：结束仿真

在下图"ShootBall3SITL.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. 关键知识点

关键知识点1：库与 API 导入

导入基础工具库（时间、数值计算、系统操作）和计算机视觉库`cv2`，用于后续图像处理。

```
import time

import numpy as np

import cv2  # 用于图像处理

import sys
```

导入 RflySim 平台的专用 API，分别用于控制无人机（PX4MavCtrlv4）、获取仿真环境图像（VisionCaptureApi）和操作 UE4 仿真窗口（UE4CtrlAPI）。

```
#     RflySim相关API（无人机控制、视觉捕获、UE4仿真环境控制）

import PX4MavCtrlV4 as PX4MavCtrl  # Pixhawk飞控控制

import VisionCaptureApi  # 视觉图像捕获

import UE4CtrlAPI  # Unreal Engine 4仿真环境控制
```

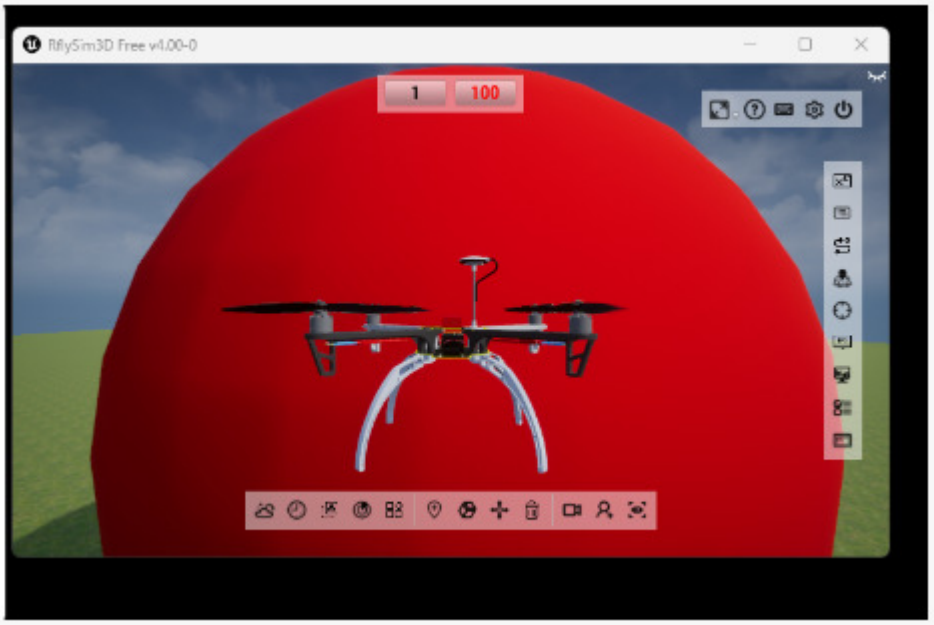
关键知识点2：初始化配置

初始化UE4控制接口和视觉捕获接口，确保能正常获取图像和发送控制指令。

```
#      初始化UE4控制接口

ue = UE4CtrlAPI.UE4CtrlAPI()

#      初始化视觉捕获接口

vis = VisionCaptureApi.VisionCaptureApi()

vis.jsonLoad()  # 加载传感器配置文件（定义取图分辨率等参数）

isSuss = vis.sendReqToUE4()  # 向RflySim发送取图请求，验证通信

if not isSuss:

            sys.exit(0)  # 取图请求失败则退出

vis.startImgCap()  # 开启共享内存取图（实时获取仿真环境图像）
```

配置 UE4 窗口性能（降低分辨率和帧率）以减少资源消耗。

```
#      配置UE4窗口参数（仅影响显示，不影响取图分辨率）

ue.sendUE4Cmd('r.setres 720x405w',0)  # 设置窗口分辨率（降低资源占用）

ue.sendUE4Cmd('t.MaxFPS 30',0)  # 设置最大帧率（30Hz，与取图频率一致）

time.sleep(2)  # 等待配置生效
```



初始化无人机控制接口，同时创建追踪目标（红色球体）。

```python
# 初始化无人机控制接口

mav = PX4MavCtrl.PX4MavCtrler(1)  # 实例化PX4飞控控制器

mav.InitMavLoop()  # 初始化MAVLink数据接收循环（与飞控通信）

# 在UE4中创建一个红色球体（默认颜色），设置位置和姿态

ue.sendUE4Pos(100,152,0,$$3,0,-2],$$0,0,0])  # 参数：ID、位置、姿态

time.sleep(0.5)
```

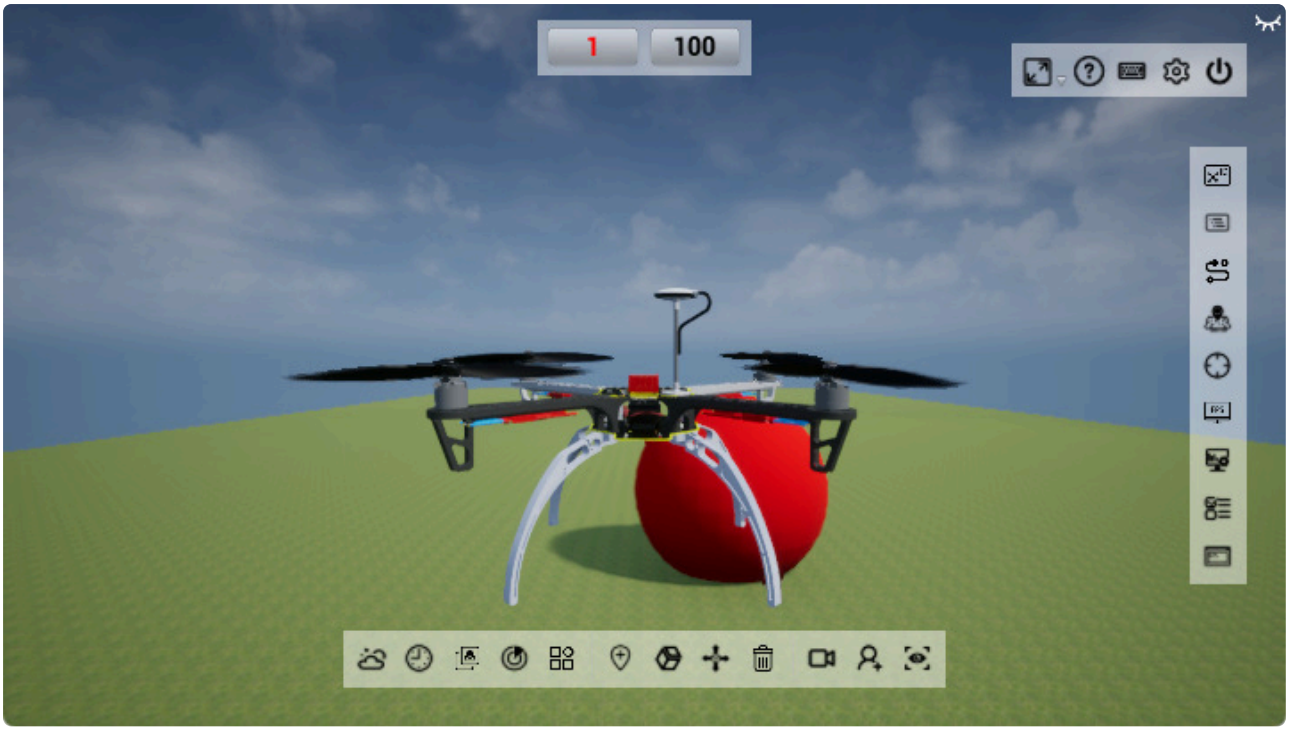

```
Json use relative path mode
jsonPath= C:\PX4PSP\RflySimAPIs\8.RflySimVision\1.BasicExps\1-VisionCtrlDemos\e3_ShootBall/Config.json
Got 1 vision sensors from json
Start lisening to timeStmp Msg
Got time msg from CopterSim # 1 , not on this PC
Got time msg from CopterSim # 1 , running on this PC
```

关键知识点3：核心函数定义

3.1红色球体识别与定位（`calc_centroid`）

通过颜色阈值提取图像中的红色区域

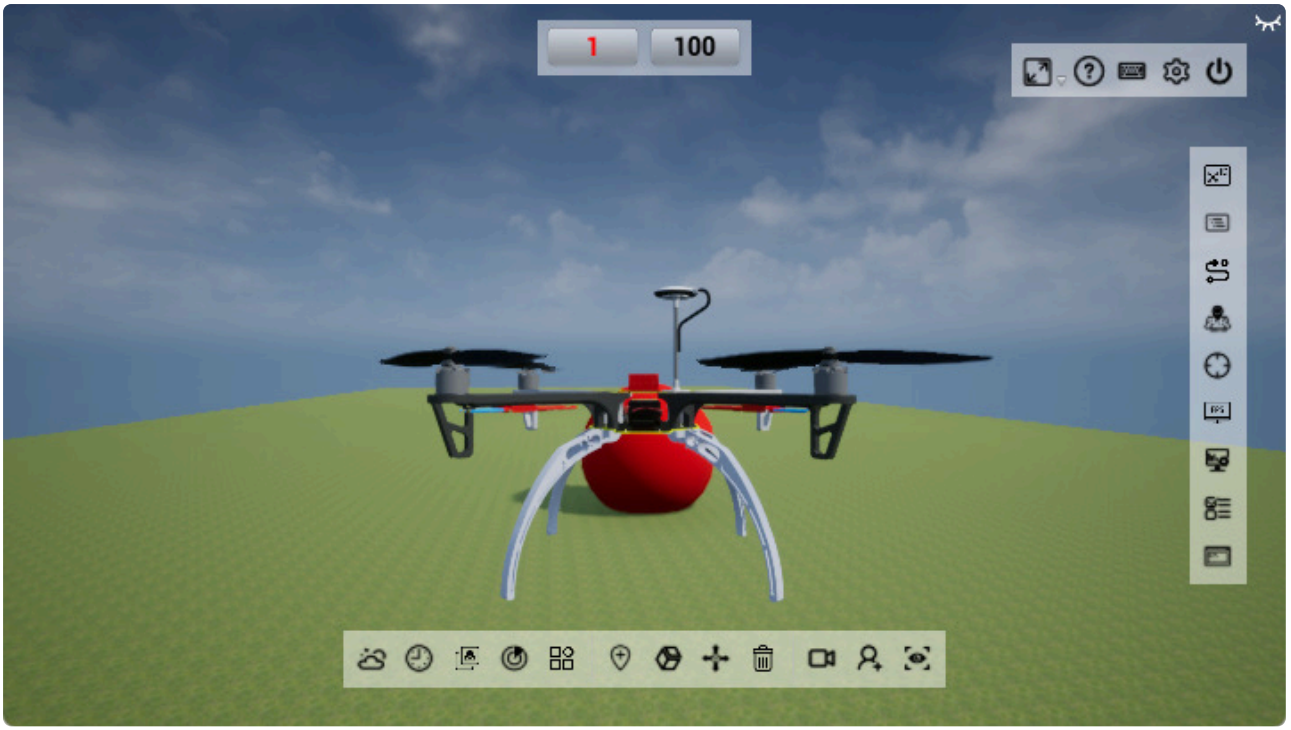
```python
def calc_centroid(img):

            """获取图像中红色物体的质心（中心坐标）和面积"""

            #      定义红色在BGR色彩空间中的阈值范围（用于提取红色区域）

            low_range = np.array([0,0,80])

            high_range = np.array([100,100,255])

            th = cv2.inRange(img, low_range, high_range)  # 二值化：红色区域为白色，其他为黑色

            #      膨胀操作：增强红色区域的连通性（减少噪声影响）

        dilated = cv2.dilate(th, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3)), iterations=2)

        cv2.imshow("dilated", dilated)  # 显示处理后的二值化图像

        cv2.waitKey(1)

        #      计算图像矩（用于求质心）

        M = cv2.moments(dilated, binaryImage=True)

        #      若红色区域面积超过最小阈值（避免噪声），则计算质心

        if M["m00"] >= min_prop*width*height:

            cx = int(M["m10"] / M["m00"])  # x方向质心

            cy = int(M["m01"] / M["m00"])  # y方向质心

            return [cx, cy, M["m00"]]  # 返回质心坐标和面积

        else:

            return [-1, -1, -1]  # 未检测到有效红色区域
```



3.2速度控制指令计算（`controller`）

检测红色球体的图像位置

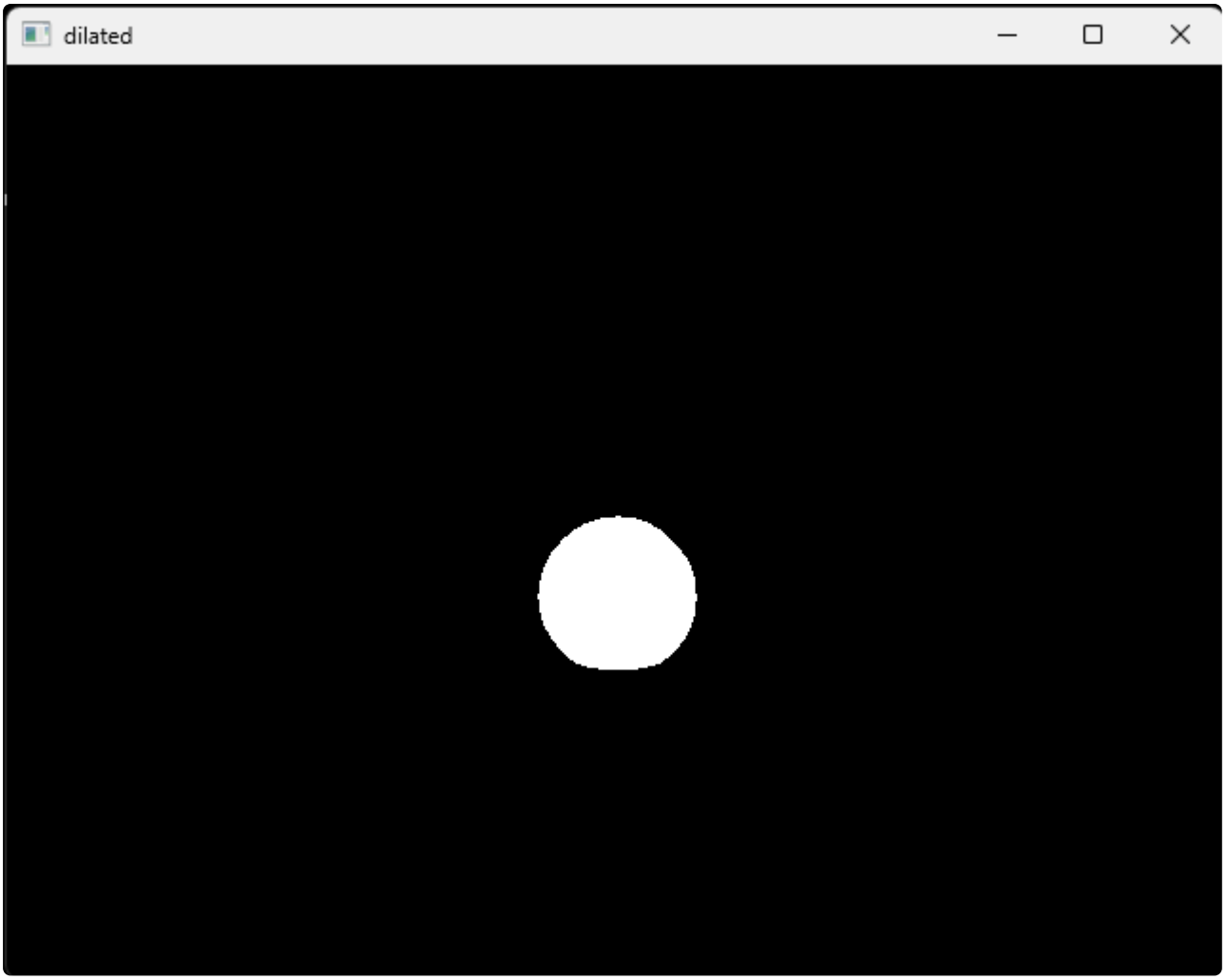
```python
def controller(p_i):

            """根据红色球体的图像位置，计算无人机速度指令"""

            #      若未检测到球体（质心坐标为负），则顺时针旋转搜索

            if p_i[0] < 0 or p_i[1] < 0:

                return [0, 0, 0, 1]  # [前向，右向，下向，偏航角速度]：仅顺时针偏航

            #      检测到球体：计算位置偏差（质心与图像中心的差距）

            ex = p_i[0] - width / 2  # x方向偏差（水平）

            ey = p_i[1] - height / 2  # y方向偏差（垂直）

        # 前向速度：若球体面积小于最大阈值（距离过远），则向前移动（速度2）；否则停止

        vx = 2 if p_i[2] < max_prop*width*height else 0

        vy = 0  # 右向速度初始化为0

        vz = K_z * ey  # 下向速度（与垂直偏差成比例，K_z为比例系数）

        yawrate = K_yawrate * ex  # 偏航角速度（与水平偏差成比例，K_yawrate为比例系数）

        return [vx, vy, vz, yawrate]  # 返回速度控制指令
```



3.3图像处理与控制指令生成（`procssImage`）

检查是否有可用图像，调用球体识别和控制指令计算函数，生成最终控制指令。

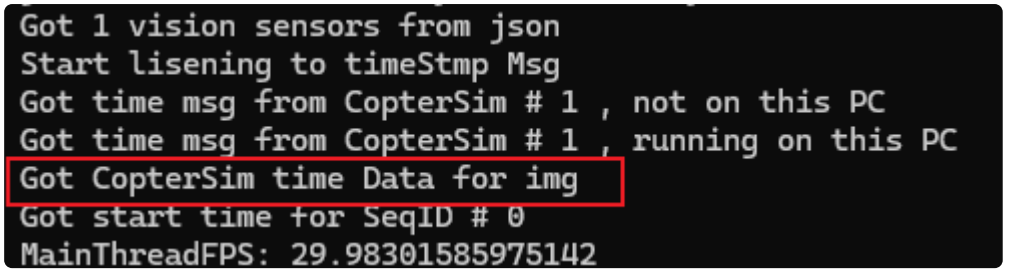
```
def procssImage():

            """处理图像并生成无人机速度控制指令"""

            global ctrl_last

            if vis.hasData[0]:  # 若有图像数据

                img_bgr = vis.Img[0]  # 获取BGR格式图像

                p_i = calc_centroid(img_bgr)  # 识别红色球体位置

                ctrl = controller(p_i)  # 计算控制指令

            else:

                ctrl = [0,0,0,0]  # 无图像数据时停止控制

            return ctrl
```



3.4速度限制函数（`sat`）

```python
def sat(inPwm, thres=1):

            """限制控制指令的最大值（饱和处理，防止速度过大）"""

            outPwm = inPwm

            for i in range(len(inPwm)):

                if inPwm[i] > thres:

                    outPwm[i] = thres

                elif inPwm[i] < -thres:

                    outPwm[i] = -thres

            return outPwm
```

- 功能：限制控制指令的幅值，避免无人机速度过大导致不稳定。



关键知识点4：主循环（30Hz 控制逻辑）

```
lastTime = time.time()

startTime = time.time()

timeInterval = 1/30.0  # 控制频率30Hz（每0.033秒执行一次）

flag = 0  # 状态标志（用于分阶段执行任务）
```

图像处理与控制参数

```
#     图像处理与控制参数

width = 640  # 图像宽度

height = 480  # 图像高度

min_prop = 0.000001  # 红色区域最小面积比例（过滤噪声）

max_prop = 0.3  # 红色区域最大面积比例（判断是否过近）

K_z = 0.003 * 640 / height  # 垂直方向比例系数（适配图像高度）

K_yawrate = 0.005 * 480 / width  # 偏航方向比例系数（适配图像宽度）
```

主循环以 30Hz 频率运行，按时间分阶段控制无人机：

```
    主循环（30Hz频率执行）

    ctrlLast = [0,0,0,0]  # 上一时刻控制指令

    while True:
              #     控制循环时间间隔（确保30Hz执行）

              lastTime += timeInterval

              sleepTime = lastTime - time.time()

              if sleepTime > 0:

                  time.sleep(sleepTime)

              else:

                  lastTime = time.time()

              #     每100次循环打印一次帧率（监控实时性）

              num += 1

              if num%100 == 0:

                  tiem = time.time()

                  print('MainThreadFPS: '+str(100/(tiem-lastClock)))

                  lastClock = tiem
```

5 秒后：解锁无人机，切换到外部控制模式，起飞到 5 米高度。

```
          # 阶段1：5秒后解锁无人机并起飞到5米高度

          if time.time() - startTime > 5 and flag == 0:

              print("5s, Arm the drone")

              mav.initOffboard()  # 初始化offboard模式（外部控制模式）

              flag = 1

              mav.SendMavArm(True)  # 解锁无人机

              print("Arm the drone!, and fly to NED 0,0,-5")

              mav.SendPosNED(0, 0, -5, 0)  # 发送位置指令：NED坐标系（北0, 东0, 下-5，即高度5米）
```

```
5s, Arm the drone
Arm the drone!, and fly to NED 0,0,-5
MainThreadFPS: 23.517588187647554
MainThreadFPS: 29.987084467866136
MainThreadFPS: 29.901951663155902
```

15 秒后：飞行到指定位置（NED 坐标 (-30,-5,-5)）。

```
            # 阶段2：15秒后飞到指定位置

            if time.time() - startTime > 15 and flag == 1:

                flag = 2

                mav.SendPosNED(-30,-5, -5, 0)  # 飞到NED坐标(-30,-5,-5)

                print("15s, fly to pos: -30,-5, -5")
```

```
15s, fly to pos: -30,-5, -5
MainThreadFPS: 30.066253506668897
MainThreadFPS: 29.993232346750982
MainThreadFPS: 30.007130648402516
```

25 秒后：开始通过视觉识别红色球体，计算速度控制指令并发送给无人机，实现追踪功能。

```
            # 阶段3：25秒后开始追踪红色球体

            if time.time() - startTime > 25 and flag == 2:

                flag = 3

                #     显示原始图像（调试用）

                if vis.hasData[0]:

                    img_bgr = vis.Img[0]

                    cv2.imshow("dilated", img_bgr)

                    cv2.waitKey(1)

                print("25s, start to shoot the ball.")
```

```
25s, start to shoot the ball.
MainThreadFPS: 29.35197783631419
MainThreadFPS: 29.897008919126826
MainThreadFPS: 30.045992682318104
MainThreadFPS: 30.05330815362963
MainThreadFPS: 29.96541652875168
MainThreadFPS: 30.019116654535356
```

控制指令经过平滑处理（限制变化率），避免无人机动作突变，提高稳定性。

```
            # 阶段4：持续追踪球体（25秒后）

            if time.time() - startTime > 25 and flag == 3:

                ctrlNow = procssImage()  # 获取当前控制指令

                ctrl = sat(ctrlNow, 5)  # 限制最大速度为5

                #     平滑控制指令（限制速度变化率，避免突变）

                if ctrl[0]-ctrlLast[0] > 0.5:

                    ctrl[0] = ctrlLast[0] + 0.05

                elif ctrl[0]-ctrlLast[0] < -0.5:

                    ctrl[0] = ctrlLast[0] - 0.05

                #      （右向速度同理处理）

                if ctrl[1]-ctrlLast[1] > 0.5:

                    ctrl[1] = ctrlLast[1] + 0.05

                elif ctrl[1]-ctrlLast[1] < -0.5:

                    ctrl[1] = ctrlLast[1] - 0.05

                ctrlLast = ctrl  # 更新上一时刻指令

                #     发送速度指令给无人机（FRD坐标系：前、右、下、偏航）

                mav.SendVelFRD(ctrl[0], ctrl[1], ctrl[2], ctrl[3])
```

总结

这段代码实现了一个完整的 "无人机视觉追踪红色球体" 的仿真控制流程：通过 RflySim 获取仿真环境图像，用 OpenCV 识别红色球体位置，根据位置偏差计算速度控制指令，最终

通过 MAVLink 协议发送给无人机，实现自动追踪。整个过程分阶段执行（起飞→移动→追踪），并通过频率控制、指令平滑等机制保证稳定性。

6. 参考资料

1. RflySim官方文档
2. Python编程基础教程

7. 常见问题

Q1：在运行程序时遇到图像处理错误或无法识别红色球体，该如何解决?

A1：检查仿真环境中是否正确生成了红色球体，并确认图像捕获接口是否正常工作。如果问题仍然存在，检查摄像头参数和图像处理函数中的颜色阈值范围是否合适。

Q2：如何调整无人机的追踪灵敏度和速度?

A2：可以通过修改代码中的[K_z](file:///F:/git/RflySimAPIs/1.RflySimIntro/1.BasicExps/e7_RunPythonProject/ShootBall3.py#L38-L38)和[K_yawrate](file:///F:/git/RflySimAPIs/1.RflySimIntro/1.BasicExps/e7_RunPythonProject/ShootBall3.py#L39-L39)参数来调整追踪灵敏度，还可以通过修改速度限制函数[sat](file:///F:/git/RflySimAPIs/1.RflySimIntro/1.BasicExps/e7_RunPythonProject/ShootBall3.py#L115-L125)中的阈值来控制最大速度。

Q3：如何确保仿真环境的性能稳定?

A3：可以通过调节UE4窗口的分辨率和帧率参数来优化性能，例如使用命令`ue.sendUE4Cmd('r.setres 720x405w',0)`设置较低的分辨率，使用`ue.sendUE4Cmd('t.MaxFPS 30',0)`限制最大帧率。

---

1. https://rflysim.com/ ↩

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