北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

航空器创新设计与实践系列竞赛

“十万火急”赛道

设计报告

题　　目	：	基于 Faster-rcnn 算法实现无人机飞往指定区域灭火
学院/书院	：	精工书院
姓　　名	：	刘梓越 谭佳明 陈萌　王诗绮 尹嘉祥
学　　号	：	1120220823 1120222762 1120211925 1120211993 1120210261

2023 年 7 月 11 日

摘要

随着科技和社会的发展，无人机所能应用的场景越发广泛，在物流、应急救援、测绘、植物保护等众多领域蓬勃发展。无人机救火的优点主要可以体现在：提高工作效率，有效减少消防人员受到的伤害等。在“十万火急”无人机仿真比赛中，将探究无人机应用于灭火救灾方面的可能性并在比赛环境中进行模拟。首先，拍摄环境中“着火”圆环的数字牌制作数据集，再通过 Faster-rcnn 算法进行训练，最后 实现自主识别比赛环境中的数字；其次，利用 PID 控制算法，实现了无人机的起飞、飞行至目标圆环以及降落。通过仿真结果可以看出，无人机可以快速且精准地完成 比赛要求，实现飞往指定区域灭火，验证了无人机在灭火领域中有着光明的前景。

关键词：无人机 Airsim 仿真 Faster-rcnn

Abstract

With the development of science and technology and society, drones can be applied to a wider and wider range of scenarios, booming in logistics, emergency rescue, surveying and mapping, plant protection and many other fields. The advantages of drone firefighting can be mainly reflected in the improvement of work efficiency and the effective reduction of firefighters' injuries. In the "Hundred Thousand Emergencies" Drone Simulation Competition, we will explore the possibility of applying drones to firefighting and disaster relief and simulate it in the competition environment. Firstly, a data set is created by photographing the number plates of the "fire" circle in the environment, and then the Faster-rcnn algorithm is used for training, and finally the numbers in the competition environment are recognised autonomously; secondly, the PID control algorithm is used to realise the take-off of the UAV, its flight to the target circle, and its landing. The simulation results show that the UAV can quickly and accurately complete the requirements of the competition and fly to the designated area to extinguish the fire, which verifies that the UAV has a bright future in the field of fire extinguishing.

Keywords: Drone Airsim Simulation Faster-rcnn

目录

第 1 章 算法简介

由“十万火急”比赛要求可得，无人机需从起降区起飞，飞行至固定位置的2*2m 的水库取水，经过无人机摄像头识别到标有队伍分配到的数字的圆环，携带水源穿越该“着火”圆环进行“喷水”作业，完成任务后，无人机返回至起降区的白色圆环内即可视为完成比赛。本组根据比赛要求将流程简化为下图：

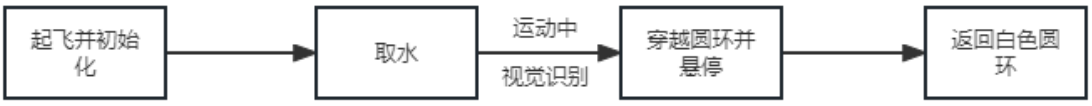

图 1-1 设计流程图

上图算法可归纳为三个部分：起飞取水的飞行控制，运动中的图像识别以及从圆环返回起降区的飞行控制。

1.1 起飞取水

在正式起飞前需要根据比赛信息做准备工作。首先，根据队伍所分配的数字设置目标信息，例如：target_number = '2'。再初始化检测器的所有信息。随后启动无人机飞行至取水区。下面为起飞取水过程中的详细介绍：

1.1.1 连接到 AirSim 模拟器

Import airsim 导入 airsim 库，调用 airsim.MultirotorClient() 连接到 Airsim 模拟器。

1.1.2 获取控制权

调用 client.enableApiControl(True) 获得 Airsim 模拟器中无人机的控制权。

1.1.3 解锁

调用 client.armDisarm(True) 解锁无人机的旋翼，使模拟环境中的无人机能够起飞。

1.1.4 起飞

调用 client.takeoffAsync().join() 使得无人机自主起飞。

1.1.5 中间点和终点位置

import numpy as np 导入 numpy 库，调用 np.array() 将中间点 midpoint 与终点 endpoint 的坐标以列表的形式表示出来。

我们所采取的策略是无人机飞至水库的上方，然后下降取水，因此设置中间点保证无人机能够越过水库箱体不发生运动干涉。

1.1.6 当前目标位置

先将中间点 midpoint 作为目标 target ，保证无人机先飞至中间点位置。

1.1.7 记录位置

声明 x , y, z 三个列表变量，用于存储生成的坐标。

1.1.8 控制循环

采用 while 循环一直执行，直至无人机到达终点跳出循环，保证无人机达到目标取水点前运动与坐标获取的持续性。

1.1.9 获取无人机当前位置

调用 client.getMultirotorState().kinematics_estimated.position 获取当前时刻的坐标值，并储存于 current 变量中。

1.1.10 计算位置误差

将目标坐标 target 与 当前位置 current 作差得到位置误差，并调用 print() 将当前坐标及位置误差信息打印到控制台便于观察调试。

1.1.11 记录位置

调用 append() 函数，将每个采样点时刻的坐标添加至对应列表变量 x, y, z 末尾处。

1.1.12 检查是否到达目标位置

经过调试与比较数据发现，实际取水点与目标取水点之间存在误差，且 y 轴方向上误差较大，因此将 y 轴上位置误差小于 1 设定为阈值。当 error[1] > = 1 时，继续控制循环；当 error[1] > = 1 时，跳出控制循环。

当无人机到达中间点时，将终点坐标 endpoint 赋给 target 即将终点坐标作为目标。因为 target, midpoint 变量为列表，不能直接进行比较判断，因此添加 .all() ，判断各个元素是否相等。

1.1.13 控制无人机运动

调用 moveToPositionAsync() 使得无人机以 target 为目标进行飞行。

1.2 识别并穿越圆环

根据比赛要求，需要识别对应数字并穿越该圆环，而目标识别需要有大量图片的数据集，采用 Faster-rcnn 算法对数据集进行训练，得到对应的模型，实现视觉识别数字的效果

1.2.1 拍摄数字及圆环图片

控制无人机飞行以达到调整无人机的坐标，能够对数字圆环拍摄出不同方位的照片。

调用 client.moveToZAsync(h, v).join()设定飞行速度及所要达到的高度，调用 client.moveToPositionAsync(-20, 45, -16, 1)规定无人机按命令速度飞行至指定坐标点。调用指定摄像头与调整相机位置 client.simSetCameraPose("front_center", airsim.Pose(airsim.Vector3r(0, 0, 0), airsim.to_quaternion(0, 0, 0)))

图 1-2 制作数据集前所拍摄的照片

利用循环拍摄出千张图片，同时进行筛选，去除拍摄不完整、方向雷同的照片最后得到 1080 张包含各个角度，且数字圆环均包括的照片（1080*720）

1.2.2 制作数据集并训练

采用 SpireView-v4.9.2 为照片上的数字及圆环按照“1”、“2”、“3”和“circle”打上标签，并且转化为 coco 格式，最终得到数据集。

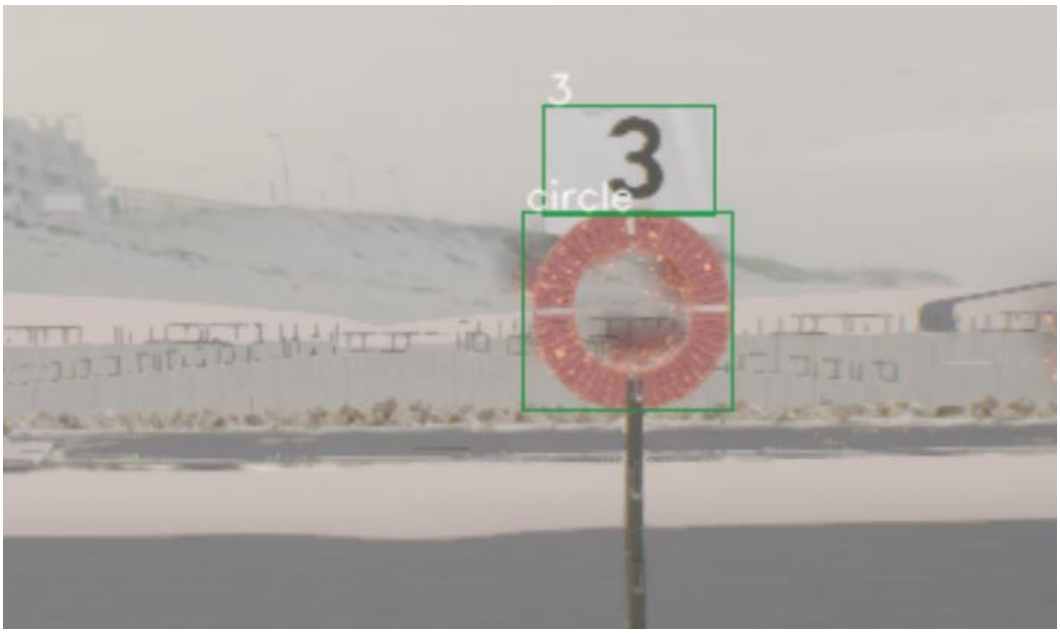

图 1-3 已打过标签的图片

通过 Faster-rcnn 算法制作训练集，首先利用在 Ubuntu 系统下配置好 mmdetection 环境，再对 Faster-rcnn 算法的相关参数进行更改，如下图所示：

```
# CLASSES = ('person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus',
#            'train', 'truck', 'boat', 'traffic light', 'fire hydrant',
#            'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog',
#            'horse', 'sheep', 'cow', 'elephant', 'bear', 'zebra', 'giraffe',
#            'backpack', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee',
#            'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat',
#            'baseball glove', 'skateboard', 'surfboard', 'tennis racket',
#            'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl',
#            'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'carrot',
#            'hot dog', 'pizza', 'donut', 'cake', 'chair', 'couch',
#            'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop',
#            'mouse', 'remote', 'keyboard', 'cell phone', 'microwave',
#            'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock',
#            'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush')
CLASSES = ('1', '2', '3', 'circle')
```

图 1-4 相关参数的修改

更改完参数之后开始训练，如下图：

```
ta: 0:59:40, time: 0.555, data_time: 0.046, memory: 3294, loss_rpn_cls: 0.4021,
loss_rpn_bbox: 0.0370, loss_cls: 0.4085, acc: 88.7383, loss_bbox: 0.1499, loss:
0.9976
2023-07-11 06:00:55,166 - mmdet - INFO - Epoch [1][100/542]     lr: 3.976e-03, e
ta: 0:56:50, time: 0.510, data_time: 0.006, memory: 3294, loss_rpn_cls: 0.0594,
loss_rpn_bbox: 0.0302, loss_cls: 0.3098, acc: 90.3379, loss_bbox: 0.4131, loss:
0.8125
2023-07-11 06:01:20,884 - mmdet - INFO - Epoch [1][150/542]     lr: 5.974e-03, e
ta: 0:55:45, time: 0.514, data_time: 0.007, memory: 3294, loss_rpn_cls: 0.0357,
loss_rpn_bbox: 0.0305, loss_cls: 0.2230, acc: 92.0176, loss_bbox: 0.3276, loss:
0.6168
2023-07-11 06:01:46,698 - mmdet - INFO - Epoch [1][200/542]     lr: 7.972e-03, e
ta: 0:55:02, time: 0.516, data_time: 0.007, memory: 3294, loss_rpn_cls: 0.0235,
loss_rpn_bbox: 0.0319, loss_cls: 0.1702, acc: 93.3711, loss_bbox: 0.3106, loss:
0.5361
2023-07-11 06:02:12,622 - mmdet - INFO - Epoch [1][250/542]     lr: 9.970e-03, e
ta: 0:54:29, time: 0.518, data_time: 0.007, memory: 3294, loss_rpn_cls: 0.0152,
loss_rpn_bbox: 0.0273, loss_cls: 0.1643, acc: 93.4434, loss_bbox: 0.3515, loss:
0.5583
2023-07-11 06:02:38,604 - mmdet - INFO - Epoch [1][300/542]     lr: 1.197e-02, e
ta: 0:54:00, time: 0.520, data_time: 0.007, memory: 3294, loss_rpn_cls: 0.0128,
loss_rpn_bbox: 0.0237, loss_cls: 0.1488, acc: 94.0664, loss_bbox: 0.3094, loss:
0.4947
```

图 1-5 训练后结果及精确度

检查最后的训练结果，map 值足以应用于该仿真环境中，我们进行随机测试，发现视觉识别效果良好

图 1-6 随机测试数字识别结果

至此，我们完成了视觉识别的前期准备。

1.2.3 识别并穿越圆环

在运动过程中进行视觉识别，对准目标数字下的圆环从而穿越。

client.moveToPositionAsync()拍摄正前方的图像；

img_png = np.frombuffer(responses[0].image_data_uint8, dtype=np.uint8) frombuffer 将 data 以流的形式读入转化成 ndarray 对象；

img_bgr = cv2.imdecode(img_png, cv2.IMREAD_COLOR) 从指定的内存缓存中读取数据，并把数据转换(解码)成图像格式；

target_circle = image_test(target_number, [], 'fail', model, img_bgr)[1]识别图像，得到目标圆环左上角和右下角的坐标；

center = int(target_circle[0] + target_circle[2])/2 取圆环中心坐标；

通过循环直至 center >= 630 and center <= 650 无人机对准目标数字继续前行准备穿过圆环；

through_circle(client, target_number, target_circle, target_judge, model)穿越圆环，保证穿过圆环后起飞。

1.3 返回起降区

比赛要求无人机需返回至起降区的白色圆环处。

client.moveByVelocityAsync(0, 0, -3, 3).join()

move_back(client)

此处利用 PID 控制算法控制无人机快速稳定飞到白色圆圈上空，最后降落并释放控制。

第 2 章 结果展示

运行代码在仿真环境中进行模拟，可以完成比赛所有任务，下列为无人机飞行过程中的关键步骤及最终结构：

图 2-1 起飞至取水区时的运动过程

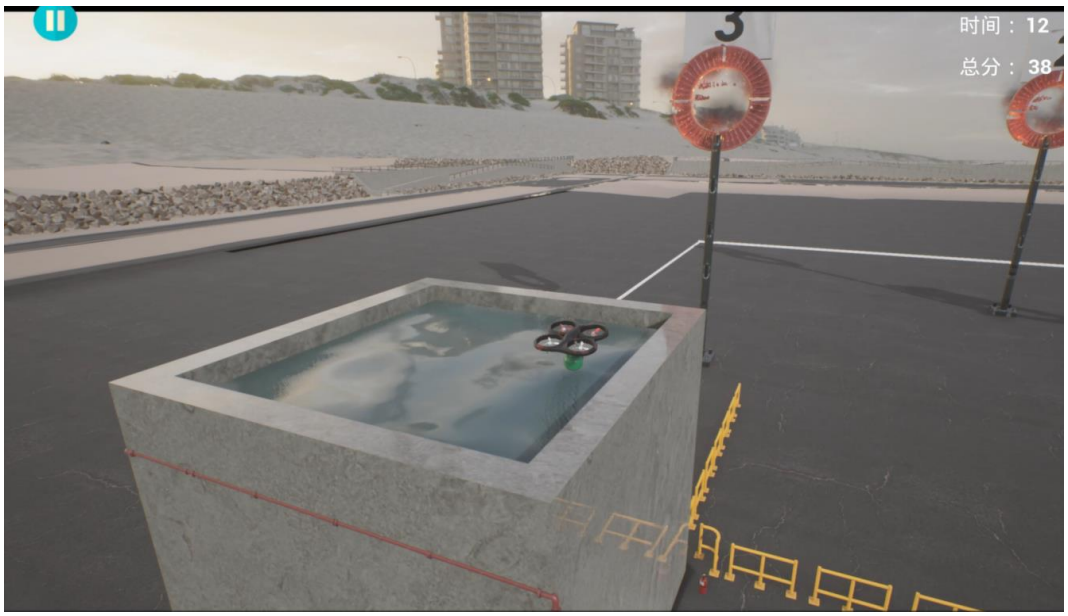

图 2-2 无人机进行取水

图 2-3 无人机穿越指定的标有 2 的“着火”圆环

图 2-4 无人机返回至起降区的白色圆环内

图 2-5 目标数字为 2 的最终成绩

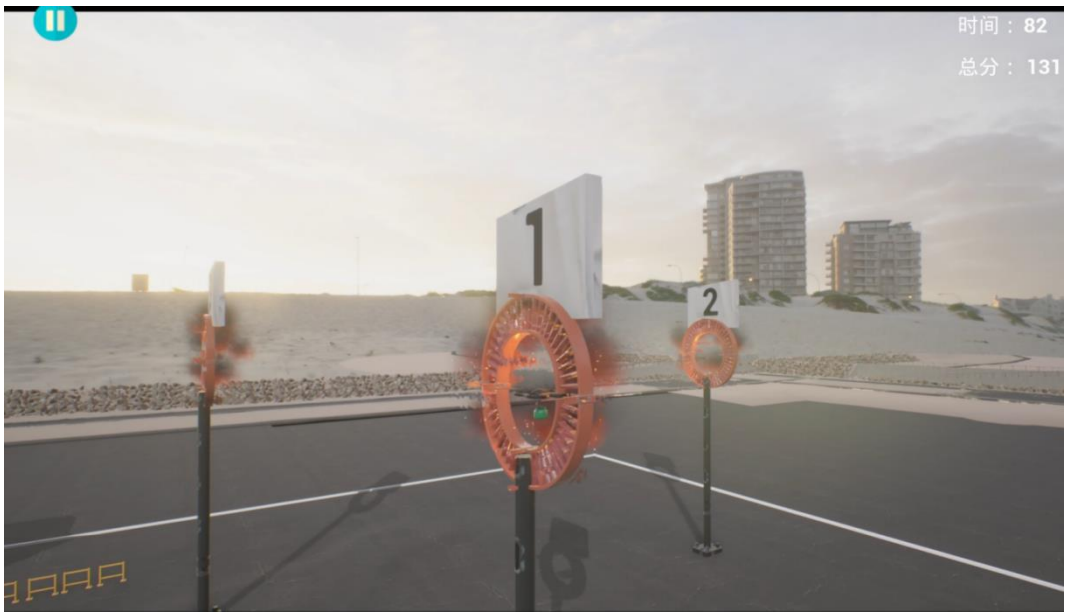

图 2-6 无人机穿越指定的标有 1 的“着火”圆环

图 2-7 目标数字为 1 时的最终成绩

图 2-8 目标数字为 3 时的最终成绩

第 3 章 创新点

3.1 创新点

3.1.1 误差调整

在控制无人机接水的过程中不断调整误差，直到误差小于一定值，使得无人机可以快速且稳定地接水

3.1.2 数据集收集

在视觉识别过程中，首先对数字牌和圆环进行拍照，再对照片进行筛选，力求包含全部数字和不同角度，然后转为 coco 数据集

3.1.3 Ubuntu 和 Windows 系统联合训练

在 Ubuntu 系统下配置好 Mmdetection 后利用 Faster_rcnn 对制作好的 coco 数据集进行训练，得到模型和对应算法后再在 Windows 系统上配置 Mmdetection 后进行视觉识别

3.1.4 同步进行移动与视觉识别

控制无人机移动和视觉识别同时进行，同时借鉴摄像识别方法，无人机只需读取一次训练好的模型，大大减少了识别过程的时间，增强了无人机的稳定性

3.1.5 应用不同移动速度

在无人机移动过程中，针对不同的视觉识别结果应用了不同的移动速度，大大减少了在移动过程中耗费的时间

3.1.6 未识别目标自动返程

同样利用视觉识别方法，使得无人机穿过圆环后，由于识别不到目标圆环，则立刻停止运动并返程，大大减少了耗费时间，同时也避免了一些极限情况的相碰

3.1.7 PID 算法

在返程过程中利用 PID 控制算法，无人机可以快速且稳定地返回到白色圆圈上空，并成功降落，PID 控制算法是指通过比较实际输出与期望输出之间的误差，来计算出一个修正量，然后将这个修正量应用到控制系统中，以达到使输出值更接近期望值的目的。将 PID 控制算法应用于无人机移动中，可以帮助无人机在飞行过程中保持稳定的姿态和位置。

3.2 可完善点

3.2.1 对 PID 控制算法参数的优化设计

3.2.2 对数据集的完善

3.2.3 对视觉识别过程中速度的控制和极限情况的处理

3.2.4 可以通过直接识别计算距离来进行识别并移动

参考文献

[1]王君,张德育,康鑫英.改进 Faster-RCNN 的低空小型无人机检测方法[J].沈阳理工大学学报,2021,40(04):23-28.

[2]吴锡,王梓屹,宋柯等.基于 Faster RCNN 检测器的输电线路无人机自主巡检系统[J].电力信息与通信技术,2020,18(09):8-15.DOI:10.16543/j.2095-641x.electric.power.ict.2020.09.002.

[3]罗小兰. 无人机对地多移动目标的视觉识别跟踪技术研究[D].电子科技大学,2022.DOI:10.27005/d.cnki.gdzku.2022.001750.

[4]成海秀,陈河源,曹惠茹等.无人机目标跟踪系统的设计与实现[J].机电工程技术,2020,49(11):165-17.

[5]邹杨,苗红霞,李成林等.基于四旋翼无人机姿态控制的自耦 PID 算法优化[J].自动化与仪器仪表,2023(03):121-124.DOI:10.14016/j.cnki.1001-9227.2023.03.121.

[6]李玉炫. 多旋翼无人机姿态与容错控制方法研究[D].沈阳大学,2022.DOI:10.27692/d.cnki.gsydx.2022.000229.

Airsim 官方平台：https://github.com/microsoft/AirSim

Airsim 知乎教程：https://www.zhihu.com/column/multiUAV

Mmdetection 官方教程：https://github.com/open-mmlab/mmdetection

附录 A

A.1 运动测试

为追踪无人机在取水过程中的运动轨迹，本组还利用数据绘制出相应的坐标曲线。先安装 matplotlib 库用于画图和 numpy 库用于记录坐标，再调用 plt.plot(), plt.legend(), plt.show() 画出无人机位置随时间变化图，便于进行后续参数调整。下位无人机自主采水时的模拟结果图:

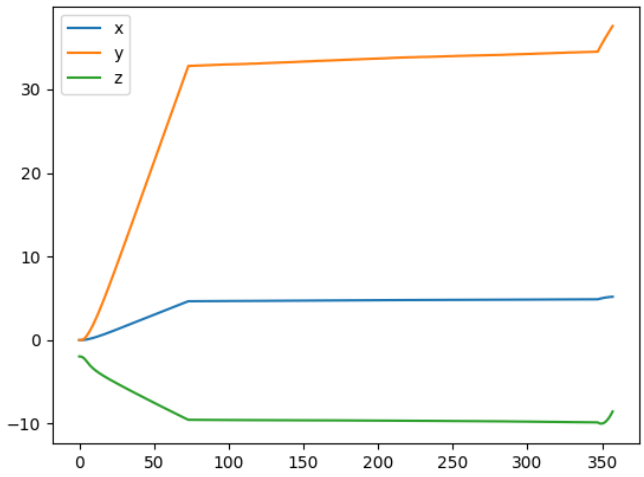

图 A-1 各轴坐标随时间变化曲线

通过模拟结果可以看出，本方案可在保证精度的条件下实现无人机自主采水控制，未发生运动干涉，满足比赛规则要求。

A.2 视觉识别环境

本组所使用的视觉识别环境是在 Cuda=11.3，在 Airsim 虚拟环境中安装了 Pytorch=1.10，下配置好 Mmdetection-2.0 后使用 Faster-rcnn 进行识别。

A.3 源代码

```
import time
import numpy as np
import airsim
from mmdet.apis import init_detector, inference_detector, show_result_pyplot
import cv2

def image_test(target_number, target_circle, target_judge, model, img):
    result = inference_detector(model, img)
    # show_result_pyplot(model, img, result)
        # 获取所有检测到的对象的类别和置信度
        class_names = model.CLASSES
    obj_list = []

    if all(bboxes.size > 0 for bboxes in result):
        max_confidence = max([bbox[-1] for bboxes in result for bbox in bboxes])
    else:
        max_confidence = 0.0  # 当序列为空时，将最大置信度设为 0.0

        # 置信度大于 0.6 时则成功
        for i, bboxes in enumerate(result):
        obj = []
        for bbox in bboxes:
            if bbox[-1] > 0.8 and max_confidence != 0.0:
                obj.append({'label': class_names[i], 'bbox': bbox[:-1], 'confidence': bbox[-1] /
max_confidence})
            elif bbox[-1] > 0.8 and max_confidence == 0.0:
                obj.append({'label': class_names[i], 'bbox': bbox[:-1], 'confidence': 0.0})
        obj_list.append(obj)

        # 初始化数字 1、2、3 和 circle 的列表
       numbers = {1: [], 2: [], 3: []}
    circle = []

        # 遍历检测结果
```

```
    for i, obj in enumerate(obj_list):
        # 判断当前目标框的类别
        if i == 0 or i == 1 or i == 2:
            # 将数字 1、2、3 的目标框添加到对应的数字列表中
            if obj and obj[0].get('label') and obj[0]['label'] in ('1', '2', '3'):
                label = int(obj[0]['label'])  # 获取数字 1、2、3 的标签
                numbers[label] += obj
    elif i == 3:
            # 将 circle 的目标框添加到 circle 列表中
            circle += obj

    # 遍历所有的 circle 目标框，将其和数字 1、2、3 对应起来
    for circle_obj in circle:
        cx1, _, cx2, _ = circle_obj['bbox']
    for label, target_list in numbers.items():
        for target in target_list:
            x1, _, x2, _ = target['bbox']
            if abs(cx1 - x1) < 50 and abs(cx2 - x2) < 50:
                target['circle'] = circle_obj  # 将对应的 circle 目标框添加到数字目标框的
属性中

    # 打印数字 1、2、3 和 circle 的目标框，输出识别的参数和结果
  for label, target_list in numbers.items():
    for i, target in enumerate(target_list):
        circle_label = target['circle']['label'] if 'circle' in target else 'None'
        if target_number == target['label']:
            target_judge = 'success'
            target_circle = target['circle']['bbox'] if 'circle' in target else 'None'
    end = [target_judge, target_circle]
  return end

def get_water(client):
  # 中间点和终点位置
  midpoint = np.array([2, 35.5, -10])
```

```
    endpoint = np.array([2, 38.5, -6])

    # 当前目标位置
    target = midpoint

    # 记录位置
    x = []
    y = []
    z = []

    # 控制循环
    while True:
        # 获取无人机当前位置
        pos = client.getMultirotorState().kinematics_estimated.position
        current = np.array([pos.x_val, pos.y_val, pos.z_val])

        # 计算位置误差
        error = target - current
        # print("Position: ", "x=", pos.x_val, "y=", pos.y_val, "z=", pos.z_val, "error:", error)

            # 记录位置
        x.append(current[0])
        y.append(current[1])
        z.append(current[2])

        # 检查是否到达目标位置
        if error[1] < 1:
            if (target == midpoint).all():
                target = endpoint
            else:
                break
        # 控制无人机运动
        client.moveToPositionAsync(target[0], target[1], target[2], 5, 0.1).join()
```

```
def through_circle(client, target_number, target_circle, target_judge, model):
    d = 1
    s = 0
    # 获取无人机当前位置
    pos = client.getMultirotorState().kinematics_estimated.position
    while True:
        # 拍摄并识别正前方的图像，
        responses = client.simGetImages([airsim.ImageRequest("0",
airsim.ImageType.Scene)])
        img_png = np.frombuffer(responses[0].image_data_uint8, dtype=np.uint8)  #
frombuffer 将 data 以流的形式读入转化成 ndarray 对象
        img_bgr = cv2.imdecode(img_png, cv2.IMREAD_COLOR)  # 从指定的内存缓存
中读取数据，并把数据转换(解码)成图像格式;
        target_judge = image_test(target_number, [], 'fail', model, img_bgr)[0]

        # 判断能否正确识别到目标数字
        if target_judge == 'fail':
            print("已距离圆环足够近，请穿过后就停止")
            # 获取无人机当前位置
            pos = client.getMultirotorState().kinematics_estimated.position
            if target_number == '1':
                s = 9.5
            elif target_number == '2':
                s = 5.5
            elif target_number == '3':
                s = 7.5
            # 由于识别精度不同，前进距离不同
            client.moveToPositionAsync(pos.x_val, pos.y_val+s, -13, 1). join()
                    break
        # 移动
        client.moveToPositionAsync(pos.x_val, pos.y_val+d, -13, 1)
        d += 1
    return
```

```
def move_back(client):
    # PID 参数
    kp = 1.5
    ki = 0.1
    kd = 1.2

    # 定义目标位置
    target_position = np.array([0.0, 0.0, -2.0])

    # 时间间隔
    dt = 0.05

    # 运行时间
    run_time = 12

    # 最大速度和加速度
    max_vel = 10.0
    max_acc = 5.0

    # 获取初始状态
    pos = client.getMultirotorState().kinematics_estimated.position
    current_position = np.array([pos.x_val, pos.y_val, pos.z_val])

    error_sum = 0
    last_error = 0
    control = np.zeros(3)

    while True:
        # 计算误差
        pos = client.getMultirotorState().kinematics_estimated.position
        current_position = np.array([pos.x_val, pos.y_val, pos.z_val])
        error = target_position - current_position
```

```
if pos.z_val > -2.0:
    # 无人机悬停 3 秒钟
    client.hoverAsync().join()
        time.sleep(1)
# 计算 PID 控制量
p_term = kp * error
i_term = ki * (error_sum + error * dt)
d_term = kd * ((error - last_error) / dt)

# 计算总控制量
control = p_term + i_term + d_term

# 限制控制量
control_norm = np.linalg.norm(control)
if control_norm > max_acc:
    control = control / control_norm * max_acc
v = client.getMultirotorState().kinematics_estimated.linear_velocity
v_arr = np.array([v.x_val, v.y_val, v.z_val])
vel_norm = np.linalg.norm(v_arr)
if vel_norm > max_vel:
    control = control - (vel_norm - max_vel) * control / control_norm

# 更新状态
client.moveByVelocityAsync(control[0], control[1], control[2], dt).join()
error_sum += error
last_error = error

# 输出当前位置和控制量
# print(f"Current Position: {current_position}, Control: {control}")

# 模拟飞行器运动延迟
time.sleep(dt)

diatance = np.linalg.norm(current_position - target_position)
```

```
        if diatance < 2:
            break

    # 无人机悬停 3 秒钟
    client.hoverAsync().join()
    time.sleep(1)
    client.moveToPositionAsync(0.0, 0.0, -2.0, 1).join()

    return

# 设置目标信息
target_number = '1'
target_circle = []
target_judge = 'fail'

# 连接无人机并起飞
client = airsim.MultirotorClient()
client.confirmConnection()
client.enableApiControl(True)
client.armDisarm(True)
client.takeoffAsync()

#定义视觉识别文件
config_file = 'checkpoint/faster_rcnn_r50_fpn_2x_coco.py'
checkpoint_file = 'checkpoint/latest.pth'
device = 'cuda:0'
# 初始化检测器
model = init_detector(config_file, checkpoint_file, device=device)

# 飞到水库上方并接水
get_water(client)
print("已成功接水，请穿过目标数字下的圆环处灭火")

# 在运动中视觉识别，对准目标数字下的圆环， 穿越圆环灭火
```

```
x = 0
v = 1
client.moveToPositionAsync(2, 42, -13, v,
drivetrain=airsim.DrivetrainType.MaxDegreeOfFreedom,
                    yaw_mode=airsim.YawMode(True, 0)).join()
while True:
    # 拍摄正前方的图像
    responses = client.simGetImages([airsim.ImageRequest("0", airsim.ImageType.Scene)])
    img_png = np.frombuffer(responses[0].image_data_uint8, dtype=np.uint8) #
frombuffer 将 data 以流的形式读入转化成 ndarray 对象
    img_bgr = cv2.imdecode(img_png, cv2.IMREAD_COLOR) # 从指定的内存缓存中
读取数据，并把数据转换(解码)成图像格式;
    # 识别图像
    target_circle = image_test(target_number, [], 'fail', model, img_bgr)[1]
    # 判断当前图像中是否有目标数字
    if any(target_circle) == True:
        # 根据目标数字下圆环中心点的坐标判断是否对准
        center = int(target_circle[0] + target_circle[2])/2
        if center >= 600 and center <= 680:# 当到达一定范围时加速
            v = 0.5
        if center >= 630 and center <= 650:
            print(f"已对准目标数字 {target_number} 下的圆环， 接下来将向前行驶")
            through_circle(client, target_number, target_circle, target_judge, model)
            break
        if center > 650:
            x += 0.5
        elif center < 630:
            x -= 0.5
    else:
        print(f"当前的图像中没有目标数字 {target_number}， 请控制无人机进行移动后
再识别")
        x += 0.5
    client.moveToPositionAsync(2-x, 42, -13, v,
drivetrain=airsim.DrivetrainType.MaxDegreeOfFreedom,
```

```
                    yaw_mode=airsim.YawMode(True, 0))
print("已成功灭火，请回到起降区白色圆环内")

# 无人机悬停 1 秒钟
client.hoverAsync().join()
time.sleep(1)

# 返回起降区白色圆环内
client.moveByVelocityAsync(0, 0, -3, 3).join()
move_back(client)
print("已处于白色圆圈内，请降落")

# 降落并释放控制
client.landAsync().join()  # land
client.armDisarm(False)  # lock
client.enableApiControl(False)  # release control
```

A.4 小组分工

刘梓越：数据集准备、视觉识别、穿越圆环、返程、设计报告书写

谭佳明：数据集准备、模型训练、视觉识别、设计报告书写

陈萌：取水控制、设计报告书写

王诗绮：数据集准备、视觉识别、设计报告书写

尹家祥：返程控制、设计报告书写

（按照工作量排序）