基于YOLO视觉检测的无人机自主跟随

1. 实验目的

本实验面向 RflySim 视觉仿真场景，完成一个“数据采集、目标检测模型训练、无人机视觉跟随控制”的完整闭环。通过本实验，学习者应能够：

1. 理解 RflySim3D、VisionCaptureApi、UE4CtrlAPI、PX4MavCtrl 与 Python 程序之间的协同关系。
2. 掌握在仿真环境中自动生成无人机目标图像与 YOLO 标注数据的方法。
3. 掌握将采集数据划分为训练集和验证集，并生成 YOLO 数据集配置的方法。
4. 掌握基于 ultralytics YOLO 模型训练无人机目标检测权重的基本流程。
5. 掌握将 YOLO 检测结果转换为飞行控制量，实现追踪机对目标无人机的自主跟随。
6. 理解视觉伺服控制中的目标中心偏差、目标面积占比、偏航控制、高度控制和前后距离控制策略。

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\2.AdvExps\e17_YoloDroneFollow

文件目录结构说明：

```
├── Config.json                 # RflySim 视觉传感器配置文件
├── DroneFollow.py              # 基于 YOLO 检测结果的无人机自主跟随程序
├── get_drone_dataset.py        # 仿真图像采集与 YOLO 标签自动生成程序
├── Python38Run.bat             # RflySim Python 运行环境启动脚本
├── ShootBall3SITL.bat          # 启动跟随实验所需 SITL 仿真环境脚本
├── split_dataset.py            # 数据集训练集与验证集划分脚本
├── StartRflySim3D.bat          # 启动 RflySim3D 场景脚本
└── train_yolo.py               # YOLO 无人机检测模型训练脚本
```

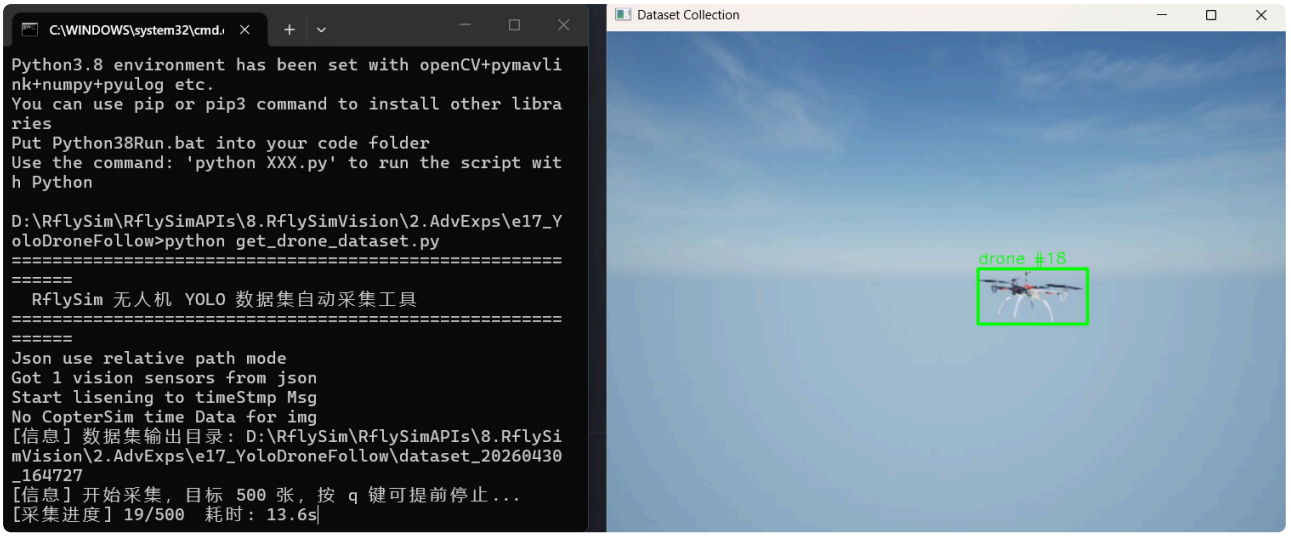
4. 实验内容或步骤

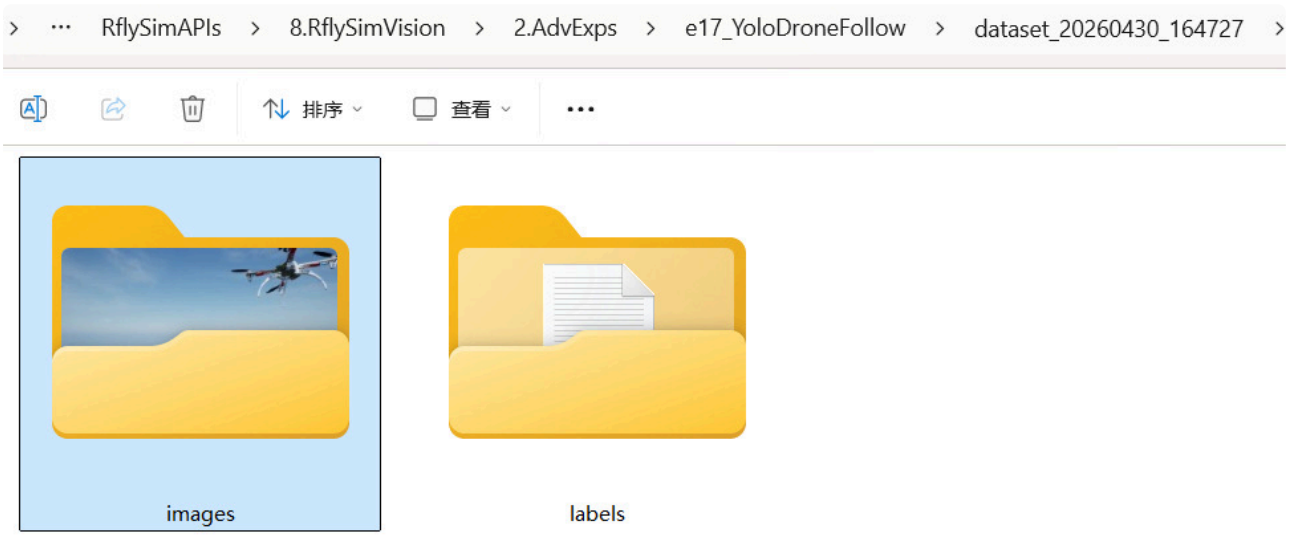
此处编写实验步骤，并添加相关图片。以及每一步设置结束后，具体的变化等等

4.1 数据采集与训练

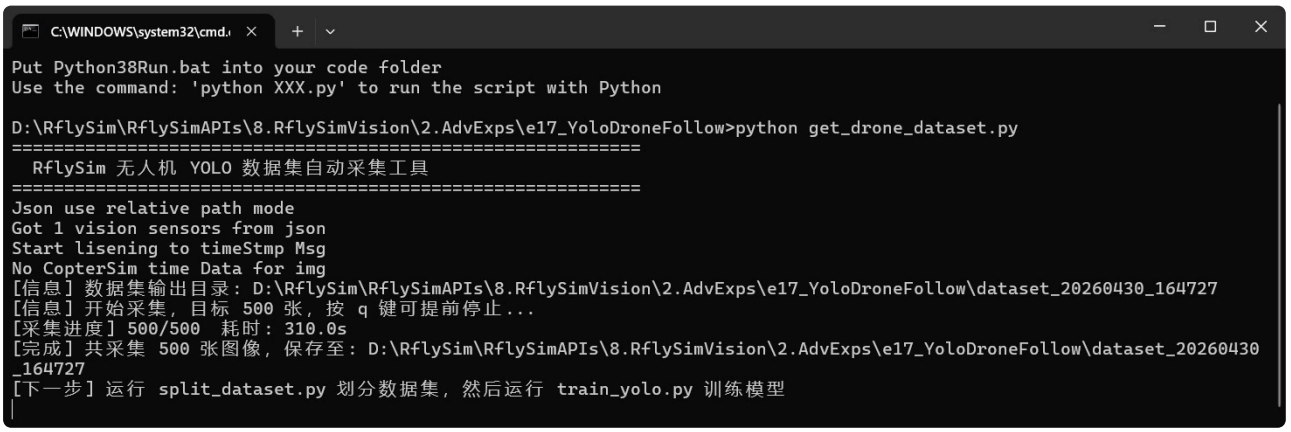
双击运行`StartRflySim3D.bat`文件，等待仿真环境初始化完成。脚本将会启动 RflySim3D 软件。

双击运行`Python38Run.bat`脚本，在弹出的对话框中输入：`python get_drone_dataset.py`，该程序将采集500张图像（约5分钟）。



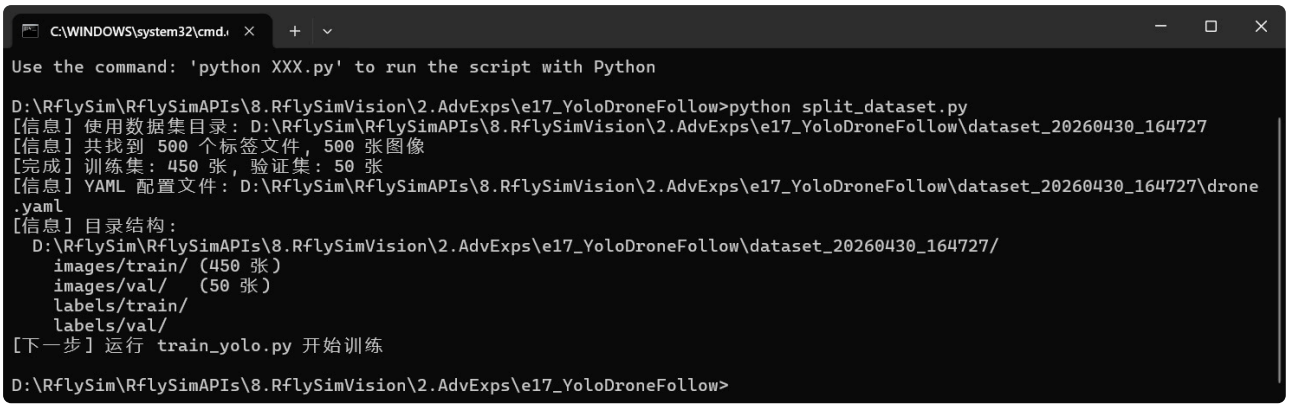


等待运行完成后，关闭窗口。

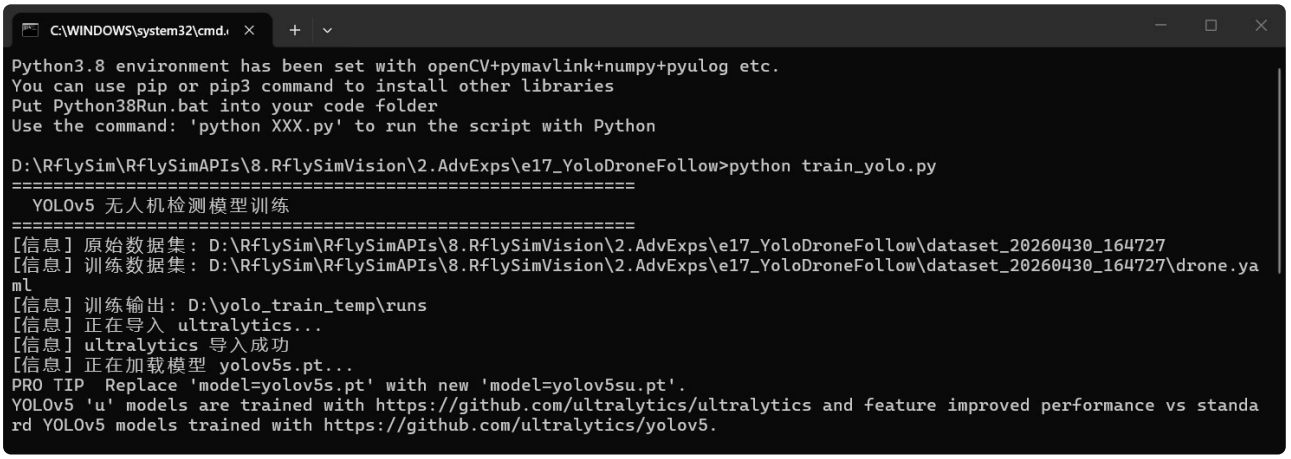


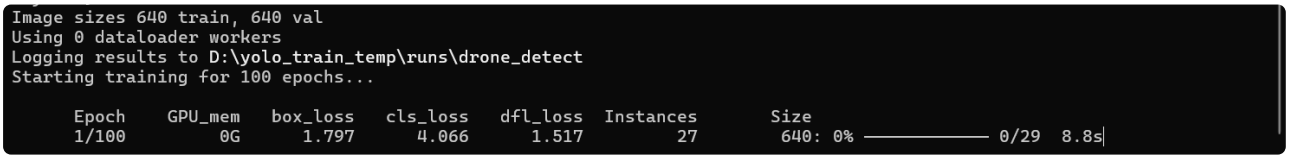
双击运行`Python38Run.bat`脚本，在弹出的对话框中输

入：`python split_dataset.py`。划分数据集。



等待运行完成后，关闭窗口。双击运行`Python38Run.bat`脚本，在弹出的对话框中输入：`python train_yolo.py`，Yolo训练。





训练完成后，会生成一个名为`best.pt`的模型文件，该文件为训练好的模型。

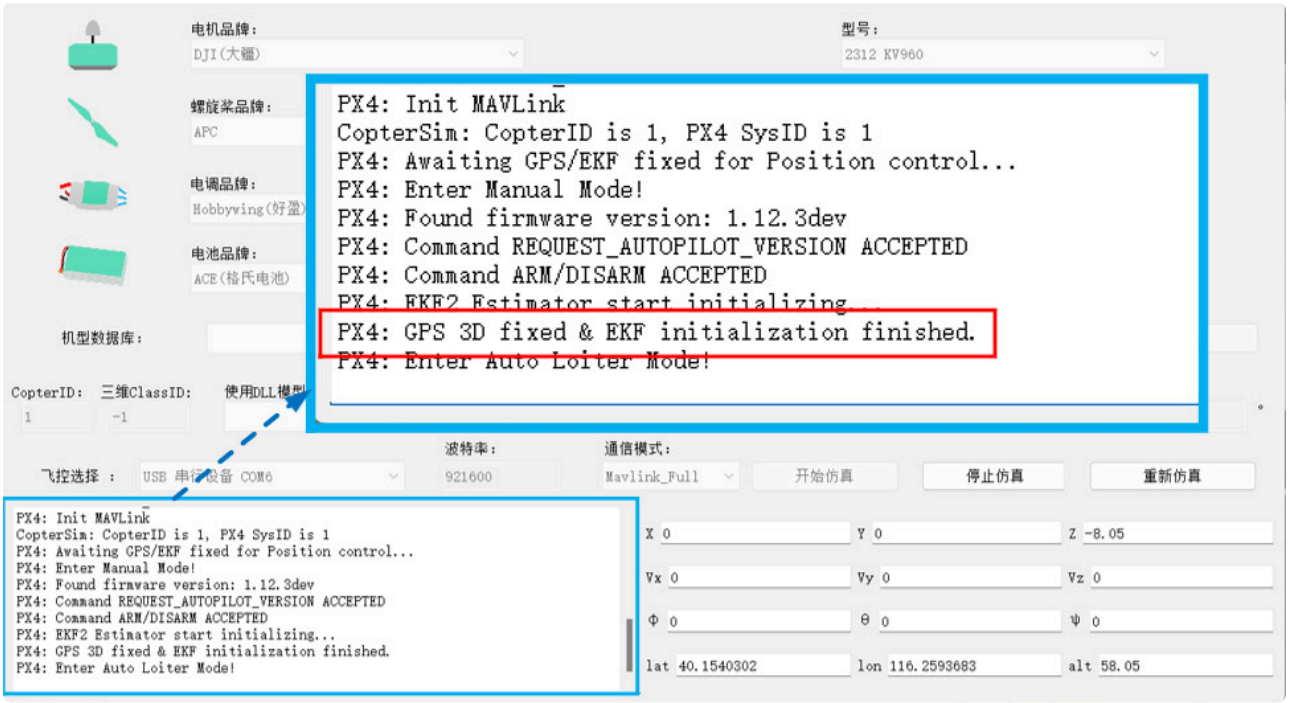
> **注意：** 程序运行过程中将自动下载YOLO官方模型文件。若无法下载可通过链接：https://pan.baidu.com/s/1F3GFML6BBLLVYIhYr1kCbA?pwd=4iuc 提取码: 4iuc 下载。 其中`best.pt`为本实验训练的模型，`yolo11n.pt`、`yolov5su.pt`为YOLO官方模型文件。

> 默认情况下，`train_yolo.py`使用 CPU 进行 YOLO 训练，即训练参数中设置了`device='cpu'`。如果需要调用 GPU 进行 CUDA 训练，请先确认电脑已安装 NVIDIA 显卡驱动、CUDA 版本对应的 PyTorch，以及`ultralytics`依赖；然后打开`train_yolo.py`，在`model.train(...)`参数中将`device='cpu'`修改为

> `device=0` 或 `device='cuda:0'`，即可使用第 0 块 NVIDIA GPU 进行训练。若有多块 GPU，可按需改为 `device=1`、`device='cuda:1'` 等。

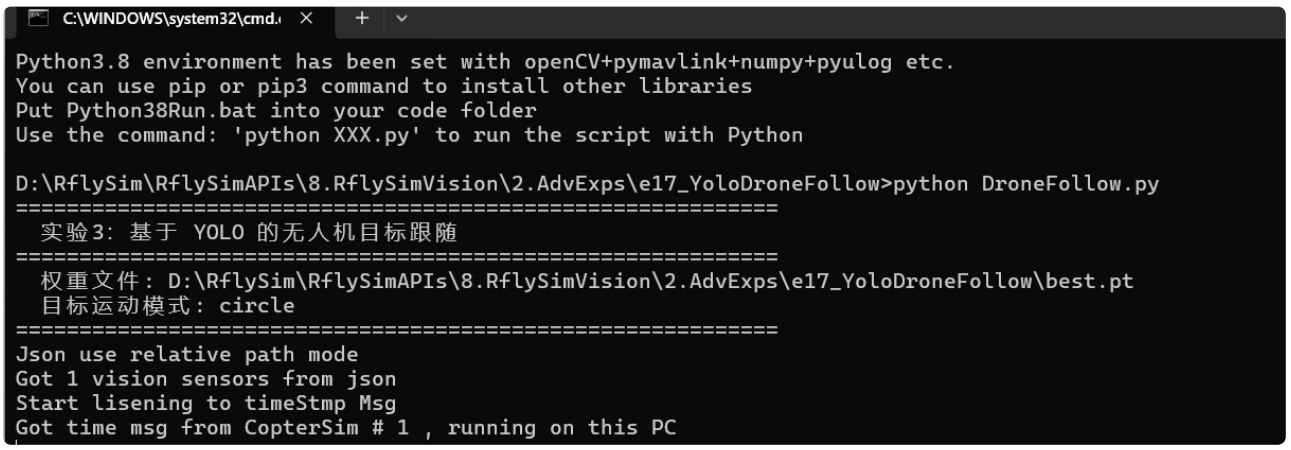
4.2 仿真初始化

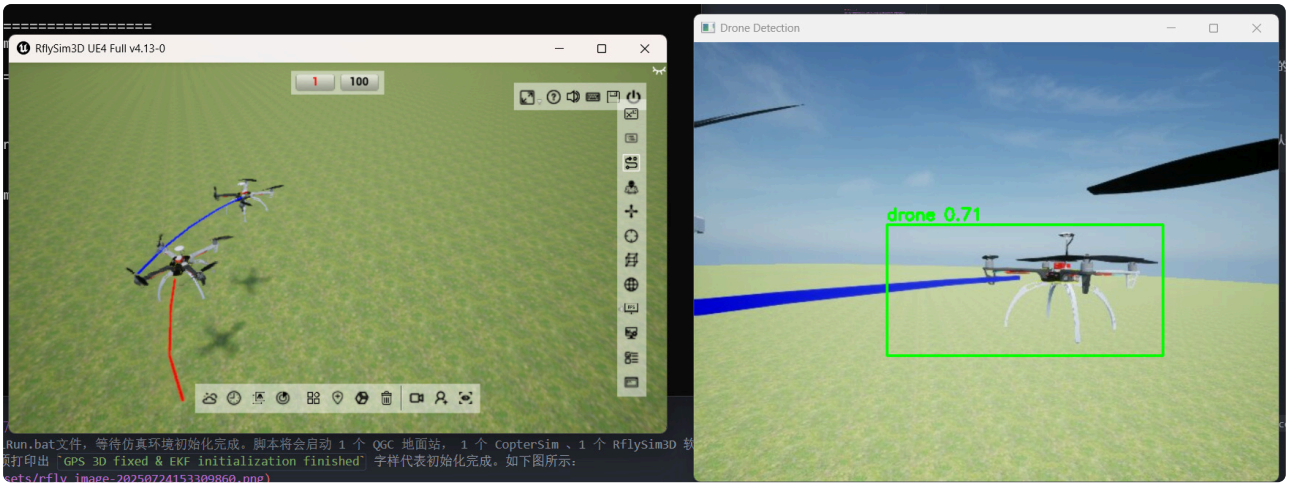
等待训练完成后，双击运行`ShootBall3SITL.bat`文件，等待仿真环境初始化完成。脚本将会启动 1 个 QGC 地面站， 1 个 CopterSim 、1 个 RflySim3D 软件， 等待CopterSim软件下侧日志栏必须打印出 `GPS 3D fixed & EKF initialization finished` 字样代表初始化完成。如下图所示：



4.3 无人机跟随

双击运行`Python38Run.bat`脚本，在弹出的对话框中输入：`python DroneFollow.py`。





跟随控制策略

- **保持安全距离**：根据目标在画面中的面积比例控制前进/后退
- **面积 < 1.5%** → 太远，快速接近 (vx=1.0)
- **面积 1.5~10%** → 适中，平滑跟随 (vx=0.5)
- **面积 > 10%** → 太近，后退 (vx=-0.3)

5. 关键知识点

5.1 实验整体思路

本实验的核心思路是先在 RflySim3D 中构造可控的无人机视觉场景，再通过 Python 接口自动采集图像和标签，训练出用于识别仿真无人机的 YOLO 模型，最后在实时仿真中用检测框位置和大小生成追踪机的速度控制指令。

整体流程如下：

1. 数据采集阶段：启动 RflySim3D 场景，程序随机改变目标无人机的位置和姿态，并根据目标三维几何点投影计算二维检测框，保存图像和 YOLO 标签。
2. 数据整理阶段：查找最新采集的数据集，将图像和标签按比例划分为训练集与验证集，并生成 YOLO 训练配置。
3. 模型训练阶段：加载 YOLO 预训练权重，使用采集数据进行无人机目标检测训练，训练完成后复制最优权重。
4. 自主跟随阶段：加载训练好的检测权重，实时获取视觉传感器图像，检测目标无人机位置，再通过 PX4MavCtrl 下发速度控制指令。

5.2 仿真图像采集与自动标注

数据采集程序首先设置采集数量、采样间隔、相机内参、观测平台位置和目标无人机初始位姿。目标无人机在画面中随机变化位置与姿态，从而生成多样化训练样本。

```
MAP_NAME = "SLAMScene"            # 使用的地图名称
NUM_SAMPLES = 500                 # 总采集数量
SAMPLE_INTERVAL = 0.5             # 采集间隔（秒）

pic_w = 640
pic_h = 480
focal = 320
px = pic_w / 2
py = pic_h / 2

PosInit = [0, 0, -8.086 - 4]
angCopterE = [0, 0, 0]

InitTargePos = [0.8, 0, -8.086 - 4]
InitTargeAng = [0, 0, math.pi / 2]
```

其中 `PosInit` 表示观测平台位置，可理解为固定摄像机所在载体；`InitTargePos` 表示被检测目标无人机的初始位置。程序通过 `UE4CtrlAPI` 控制 RflySim3D 中的目标模型，通过 `VisionCaptureApi` 获取图像。

```
ue.sendUE4Cmd(f'RflyChangeMapbyName {MAP_NAME}')
ue.sendUE4Cmd('r.setres 720x405w', 0)
ue.sendUE4Cmd('t.MaxFPS 30', 0)

ue.sendUE4Pos(1, 0, 0, PosInit, angCopterE)

vis.jsonLoad()
isSuss = vis.sendReqToUE4()
if not isSuss:
    print("[错误] 取图请求失败，请确认 RflySim3D 已启动")
    sys.exit(0)
vis.startImgCap(True)

ue.sendUE4Pos(100, DRONE_TYPE_IDS[0], 0, InitTargePos, InitTargeAng)
```

自动标注的关键是把目标无人机的三维空间关键点投影到图像平面。程序先根据目标位置和姿态计算无人机包围盒的 9 个三维关键点，然后投影成图像坐标，最后取所有投影点的最小外接矩形作为检测框。

```python
def getUav9Point(uavPos, centerH, uavAng):
    """获取无人机包围盒的9个空间关键点"""
    center = np.array(uavPos) + [0, 0, -centerH]
    inv_rot = np.linalg.inv(eul2rot(uavAng))
    offsets = [
        [-0.035, -0.035, -0.135 - 0.015],
        [0.245, -0.245, -0.035],
        [0.245,  0.245, -0.035],
        [-0.245, -0.245, -0.035],
        [-0.245,  0.245, -0.035],
        [0.13, -0.13, 0.17],
        [0.13,  0.13, 0.17],
        [-0.13, -0.13, 0.17],
        [-0.13,  0.13, 0.17]
    ]
    return [center + np.transpose(np.dot(inv_rot, np.transpose(np.array(o)))) for o in offsets]
```

YOLO 标签要求使用归一化后的 `中心点 x、中心点 y、宽、高` 格式，因此程序将 `xyxy` 边界框转换为 YOLO 标注格式：

```python
def xyxy2yolo(x1y1, x2y2, img_w, img_h):
    """将 xyxy 格式转为 YOLO 的归一化 xywh 格式"""
    cx = ((x1y1[0] + x2y2[0]) / 2) / img_w
    cy = ((x1y1[1] + x2y2[1]) / 2) / img_h
    w = (x2y2[0] - x1y1[0]) / img_w
    h = (x2y2[1] - x1y1[1]) / img_h
    return round(cx, 8), round(cy, 8), round(w, 8), round(h, 8)
```

当视觉接口中有新图像时，程序保存图像，同时写入类别编号为 `0` 的无人机目标标签。这样无需人工框选即可生成训练数据。

```python
if vis.hasData[0]:
    img = vis.Img[0]

    imwrite_unicode(os.path.join(path_img, f"{cnt}.jpg"), img)

    y1, y2, y3, y4 = xyxy2yolo(x1y1, x2y2, pic_w, pic_h)
    label_path = os.path.join(path_labels, f"{cnt}.txt")
    with open(label_path, 'w') as f:
        f.write(f"0 {y1} {y2} {y3} {y4}")

    cnt += 1
```

5.3 数据集划分与 YOLO 配置生成

数据集划分程序会自动查找最新的采集目录，避免用户手动修改路径。默认训练集比例为 90%，其余 10% 作为验证集。

```
DATASET_DIR = ""  # 留空则自动查找
TRAIN_RATIO = 0.9  # 训练集比例

script_dir = os.path.dirname(os.path.abspath(__file__))

if not DATASET_DIR:
    candidates = sorted(glob.glob(os.path.join(script_dir, "dataset_*")),
reverse=True)
    if not candidates:
        print("[错误] 未找到 dataset_* 目录，请先运行 get_drone_dataset.py 采集数据")
        sys.exit(1)
    DATASET_DIR = candidates[0]
```

随后程序创建 YOLO 常用目录结构，并把图像和标签移动到训练集、验证集对应目录中。

```
train_img_dir = os.path.join(DATASET_DIR, "images", "train")
val_img_dir = os.path.join(DATASET_DIR, "images", "val")
train_lbl_dir = os.path.join(DATASET_DIR, "labels", "train")
val_lbl_dir = os.path.join(DATASET_DIR, "labels", "val")

os.makedirs(train_img_dir, exist_ok=True)
os.makedirs(val_img_dir, exist_ok=True)
os.makedirs(train_lbl_dir, exist_ok=True)
os.makedirs(val_lbl_dir, exist_ok=True)

indices = list(range(num))
random.shuffle(indices)
train_count = int(num * TRAIN_RATIO)
train_indices = set(indices[:train_count])
```

训练配置文件中定义了训练图像目录、验证图像目录、类别数量和类别名称。这里类别只有一种，即 `drone`。

```
yaml_content = f"""# YOLOv5 训练数据集配置文件（目录格式）
# 自动生成

path: {DATASET_DIR}
train: images/train
val: images/val

# 类别数量
nc: 1

# 类别名称
names: ['drone']
"""
```

5.4 YOLO 模型训练流程

训练程序会查找最新的数据集目录，并检查是否已经生成训练配置。为了提高兼容性，程序还会检测路径中是否包含非 ASCII 字符；如果包含，则复制数据集到英文临时目录，减少 ultralytics 在 Windows 中文路径环境下的报错概率。

```
def has_chinese(path):
    """检查路径是否包含非 ASCII 字符"""
    try:
        path.encode('ascii')
        return False
    except UnicodeEncodeError:
        return True
```

实际训练使用 ultralytics 的 `YOLO` 接口加载预训练权重，并指定训练轮数、图像尺寸、批量大小、设备和输出目录。默认使用 CPU，便于在无 NVIDIA GPU 的电脑上运行。

```python
from ultralytics import YOLO

model = YOLO('yolov5s.pt')

results = model.train(
    data=yaml_path,
    epochs=100,
    imgsz=640,
    batch=16,
    device='cpu',
    workers=0,
    name='drone_detect',
    project=project_dir,
    exist_ok=True,
    patience=20,
    save=True,
    plots=True
)
```

训练完成后，程序会在训练输出目录中查找最优权重，并复制为本实验跟随程序可直接加载的模型文件。

```python
best_candidates = glob.glob(os.path.join(project_dir, "**", "best.pt"),
recursive=True)

if best_candidates:
    best_src = best_candidates[0]
    best_dst = os.path.join(script_dir, "best.pt")
    shutil.copy2(best_src, best_dst)
    print(f"[完成] best.pt 已复制到: {best_dst}")
```

5.5 YOLO 检测与无人机跟随控制

自主跟随程序首先加载训练得到的权重。如果没有找到训练权重，则提示用户先完成训练，并尝试使用通用预训练权重作为替代。

```python
WEIGHTS_PATH = os.path.join(sys.path[0], "best.pt")
if not os.path.exists(WEIGHTS_PATH):
    print("[警告] 未找到 best.pt，请先完成实验1的训练，或将权重文件复制到本目录")
    print("[提示] 将使用 yolov5s.pt 作为替代（COCO预训练，对仿真无人机效果有限）")
    WEIGHTS_PATH = os.path.join(sys.path[0], "yolov5s.pt")
```

目标无人机由 RflySim3D 中的质点模型生成，不需要额外启动一个 CopterSim。追踪机则通过 MAVLink 接口进入 Offboard 控制，并接收速度指令。

```
TARGET_ID = 100
TARGET_TYPE = 3
TARGET_INIT_POS = [3, 0, -1.5]

mav = PX4MavCtrl.PX4MavCtrler(1)
mav.InitMavLoop()

ue.sendUE4Pos(TARGET_ID, TARGET_TYPE, 0, TARGET_INIT_POS, [0, 0, 0])
```

目标运动函数提供圆形、8 字形、随机运动三种模式。默认圆形轨迹便于观察追踪机是否能够稳定保持目标在视野中心附近。

```
def update_target_position(t, dt):
    """根据运动模式更新目标无人机位置"""
    global target_pos, random_vel, last_random_change

    if TARGET_MOTION == "circle":
        target_pos[0] = CIRCLE_CENTER[0] + CIRCLE_RADIUS * math.cos(CIRCLE_SPEED *
t)
        target_pos[1] = CIRCLE_CENTER[1] + CIRCLE_RADIUS * math.sin(CIRCLE_SPEED *
t)
        target_pos[2] = CIRCLE_HEIGHT
        yaw = math.atan2(
            CIRCLE_RADIUS * CIRCLE_SPEED * math.cos(CIRCLE_SPEED * t),
            -CIRCLE_RADIUS * CIRCLE_SPEED * math.sin(CIRCLE_SPEED * t)
        )
        target_ang = [0, 0, yaw]

    ue.sendUE4Pos(TARGET_ID, TARGET_TYPE, 0, target_pos, target_ang)
```

检测函数直接对当前图像做 YOLO 推理。若检测到多个候选框，程序选择置信度最高的检测框，并返回目标中心点和框尺寸。

```python
def DetectDrone(img):
    """调用 ultralytics YOLO 检测无人机"""
    results = yolo_model(img, conf=0.4, verbose=False)
    result = results[0]

    if len(result.boxes) == 0:
        cv2.imshow("Drone Detection", img)
        cv2.waitKey(1)
        return [-1, -1, -1], [-1, -1], False

    best_idx = result.boxes.conf.argmax()
    box = result.boxes[best_idx]
    confidence = float(box.conf)
    x1, y1, x2, y2 = [int(v) for v in box.xyxy[0]]

    cx = (x1 + x2) // 2
    cy = (y1 + y2) // 2
    bw = x2 - x1
    bh = y2 - y1
    min_wh = min(bw, bh)

    return [cx, cy, min_wh], [bw, bh], True
```

跟随控制器将检测框中心偏差转换为高度和偏航控制，将检测框面积占比转换为前进、保持或后退速度。其控制优先级是先校正高度，再校正偏航，最后控制距离。

```python
def FollowController(pic_idx):
    """视觉伺服跟随控制器"""
    global max_r, last_ctrl_cmd

    if pic_idx[0] < 0 or pic_idx[1] < 0:
        if max_r > height / 2 - 50:
            return last_ctrl_cmd
        return [0, 0, 0, 0]

    ex = pic_idx[0] - width / 2
    ey = pic_idx[1] - height / 2

    if abs(ey) > 30:
        return [0, 0, K_z * ey, 0]

    if abs(ex) > 30:
        return [0, 0, 0, K_yawrate * ex]

    target_area_ratio = (pic_idx[2] ** 2) / (width * height)

    if target_area_ratio < TOO_FAR_RATIO:
        vx = APPROACH_VX
    elif target_area_ratio > TOO_CLOSE_RATIO:
        vx = RETREAT_VX
    else:
        error = target_area_ratio - DESIRED_AREA_RATIO
        vx = FOLLOW_VX - error * 10
        vx = np.clip(vx, RETREAT_VX, APPROACH_VX)

    vz = K_z * ey
    yawrate = K_yawrate * ex
    last_ctrl_cmd = [vx, 0, vz, yawrate]
    return last_ctrl_cmd
```

主循环每秒约 30 次更新目标位置、读取视觉图像、执行检测和下发控制指令。当短时间丢失目标时，程序沿用上一帧控制量；当丢失时间较长时，追踪机悬停并重新进入搜索逻辑。

```python
if is_detected and not detected:
    if time.time() - lost_time < 3:
        ctrl = copy.deepcopy(ctrlLast)
    else:
        ctrl = [0, 0, 0, 0]
        print(f"\r[丢失] t={t:.1f}s  目标丢失，悬停等待...", end="", flush=True)
        if time.time() - lost_time > 8:
            is_detected = False
            max_r = 0
else:
    ctrl = FollowController(pic_idx)

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

5.6 控制参数含义

- `K_z`：图像纵向偏差到垂直速度的比例系数，目标偏离画面中心越多，高度修正越明显。
- `K_yawrate`：图像横向偏差到偏航角速度的比例系数，用于把目标转回画面中心。
- `TOO_FAR_RATIO`：目标面积过小的阈值，说明目标距离较远，追踪机需要接近。
- `TOO_CLOSE_RATIO`：目标面积过大的阈值，说明目标距离过近，追踪机需要后退。
- `DESIRED_AREA_RATIO`：期望目标面积占比，用作平滑距离控制的参考值。
- `APPROACH_VX`、`FOLLOW_VX`、`RETREAT_VX`：分别表示接近、正常跟随和后退时的前向速度。

6.参考资料

1. RflySim官方文档： https://rflysim.com/doc/zh/
2. RflySim安装与推荐配置说明： https://rflysim.com/doc/zh/HowToInstall.pdf
3. Ultralytics YOLO 官方文档： https://docs.ultralytics.com/
4. OpenCV 官方文档： https://docs.opencv.org/
5. PX4 Offboard 控制说明： https://docs.px4.io/main/zh/flight_modes/offboard.html

7.常见问题

Q1：运行采集或跟随程序时提示“取图请求失败”怎么办?

A1：通常是 RflySim3D 未启动、视觉传感器配置未加载成功，或场景尚未初始化完成。应先运行 RflySim3D 启动脚本，等待场景窗口正常显示后再运行 Python 程序；同时检查 `Config.json` 是否存在，并确认视觉传感器配置与当前实验场景匹配。

Q2：运行训练程序时提示找不到数据集或找不到drone.yaml 怎么办?

A2：需要按顺序先运行数据采集程序，再运行数据集划分程序。采集程序负责生成 `dataset_` 开头的数据集目录，划分程序负责生成训练和验证目录以及 `drone.yaml` 配

置。如果直接运行训练程序，可能因为缺少这些中间结果而退出。

Q3：训练速度很慢或电脑没有 NVIDIA GPU 怎么办？

A3：训练程序默认使用 `device='cpu'`，兼容性较好但速度较慢。若电脑具备 NVIDIA GPU，并已正确安装显卡驱动、CUDA 对应版本的 PyTorch 和 ultralytics 依赖，可将训练参数中的 `device='cpu'` 改为 `device=0` 或 `device='cuda:0'`。如果仍使用 CPU，建议先保持较小数据量完成流程验证。

Q4：跟随程序运行后无人机不动或无法进入跟随状态怎么办？

A4：应确认 SITL 仿真已经完成初始化，CopterSim 日志中出现 `GPS 3D fixed & EKF initialization finished` 后再运行跟随程序。若追踪机未解锁或未进入 Offboard，程序无法稳定下发速度控制。还应确认 QGC、CopterSim、RflySim3D 均已正常启动。

Q5：检测窗口中无法识别目标无人机怎么办？

A5：可能原因包括训练权重缺失、模型训练样本不足、目标不在相机视野内或光照/距离变化超出训练数据覆盖范围。可先确认 `best.pt` 已生成并位于实验目录；若模型仍识别不稳定，可增加采集样本数量、扩大目标位置和姿态随机范围，然后重新划分数据集并训练。

Q6：为什么程序用目标框面积控制前后距离？

A6：单目视觉难以直接获得精确距离，但目标在图像中的面积会随距离变化而变化。目标框面积占比小通常表示目标较远，面积占比大通常表示目标较近。因此程序使用面积阈值判断接近、跟随或后退，实现简化的视觉距离控制。

---

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