Homework 3

要求

> 8 架四旋翼无人机执行两阶段任务，分别为编队跟踪阶段和围捕阶段。编队跟踪阶段，无人机需要以 扇形编队 追踪一量 未知机动 的无人车，只能得到该车的 位置信息 ，这一阶段需要避免 机体间碰撞 以及 机体与障碍物碰撞 ，安全距离 `0.5m` ；接近无人车时，无人机进入围捕阶段，编队扇形要 收缩 ，同时要求小组内的无人机尽可能 同时 打击无人车，保持时间一致性，最先和最后进入目标 `1m` 范围内的时间间隔 `<= 1s`

问题分析

`lab2` 实现了时变编队跟踪控制，本题目希望基于 `lab2` 得到答案，因此主要分析如何在 `lab2` 基础上实现，对 `lab3` 的两阶段逐一分析

编队跟踪阶段

1. **领导者**：`lab3` 中的领导者即为 `ugv` 。`lab2` 中已知领导者的时变位置信息，并可以求导得到速度信息，但在 `lab3` 中只能通过订阅话题得到领导者的当前位置信息，若要使用 `lab2` 的算法，则领导者的速度信息可以通过离散法得到，即记录上一时刻的位置，与当前位置作差再除以时间，即为粗略的速度估计；
2. **编队**：扇形编队的设计关键在于 `h` 如何设计，参考 `lab2` ，对于 3 台无人机，可以修改 `h` 中的总角度，并设计 `ka,ki,kn` 分别为组别中该无人机在总角度中大小，哪个位置，以及该组别的无人机总数。这里的 `h` 在实验中并不要求旋转，但是为了方便 控制补偿输入 的设计，沿用 `lab2` 的 ω。同时，为了保持在初始阶段的扇形编队长度较大， `r` 一开始设计的比较大，然后随程序运行时间匀速递减：$r = max(1, 30 - rd * elapsed_time)$，`rd` 即为递减系数；
3. **避障**：障碍物的位置信息已知，无人机的位置信息已经存储，所以无需订阅。一旦有潜在发生碰撞的风险，即：与圆柱体碰撞物的距离小于 `d_thresh_s` 或者与无人机的距离小于 `d_thresh_u` 就进行控制干预，具体的手段是在计算 δ_x 时加上远离障碍物的距离（δ_x 和 δ_y 按照与障碍物中心的夹角按比例增加），并给定该项一个权重系数 `kc` 调节避障的权重，同时，为了确保距离障碍物的距离越近则远离的拉力越大，需要给该项除以整体距离，因此，初始的 `kc` 相对较大

阶段切换

1. **切换条件**：切换两种无人机控制方式的条件是小组内任意一台无人机距离无人车 `<switch` ，`switch` 为距离阈值

围捕阶段

1. **收缩包围**：通过计算，5 台无人机包围无人车的情况下，若要满足无人机均在 `1m` 圆环内，同时互相保持 `0.5m` 的安全距离，则 `均匀包围` 是最简单的设计方式。在 `均匀包围` 的情况下，每两个相邻的无人机距离为 `0.618m`，满足要求，具体的设计方式与 `lab2` 的题目一致，最终得到 5 台无人机包围无人车，这个问题可以进一步简化，在 `lab3` 虽然没有要求旋转包围，但是为了简化设计 `补偿控制输入`，在 `lab3` 中仍然保持旋转包围

数学推导

本章节主要对 `问题分析` 部分提出的设计给出相应的数学表达形式，显然，需要进行计算的只有时变的 `r`，以包围半径以 `5m/s` 速度缩小，初始包围半径 `30m` 为例，给出 `r` 表达式如下：

$$r_t = \max(30 - 5t, 1) \tag{1}$$

代码实现

此章节包含将示例代码修改为满足题目要求的解题代码所需要的关键修改步骤：

1. **确定每台无人机的包围位置**：

```
self.data = {
    "uav1": {"ka":0.0, "kn": 3, "ki": 0.5},
    "uav2": {"ka":0.0, "kn": 3, "ki": 1},
    "uav3": {"ka":0.0, "kn": 3, "ki": 2},
    "uav4": {"ka":0.0, "kn": 5, "ki": 0},
    "uav5": {"ka":0.0, "kn": 5, "ki": 1},
    "uav6": {"ka":0.0, "kn": 5, "ki": 2.5},
    "uav7": {"ka":0.0, "kn": 5, "ki": 3},
    "uav8": {"ka":0.0, "kn": 5, "ki": 3.5}
}
```

2. **确定无人车的位置和速度**

```
self.ugv_data = {
    "ugv1": {"pose1": None, "pose2": None, "velocity": None},
    "ugv2": {"pose1": None, "pose2": None, "velocity": None}
}

for i, ugv in enumerate(self.ugv_list):
    # 订阅位姿和速度话题
    rospy.Subscriber(f'/{ugv}/pose', PoseStamped,
    lambda msg, idx=i: self.ugv_pose_callback(msg, idx))

def ugv_pose_callback(self, msg, idx):
    now = rospy.Time.now()
    ugv_name = self.ugv_list[idx]

    # 更新位置信息
    self.ugv_data[ugv_name]["pose1"] = self.ugv_data[ugv_name]["pose2"]
    self.ugv_data[ugv_name]["pose2"] = msg
    p1 = self.ugv_data[ugv_name]["pose1"]
    p2 = self.ugv_data[ugv_name]["pose2"]

    # 计算速度
    if p1 == None or p2 == None or self.ugv_last_time == None:
        self.ugv_last_time = now
        return
    p1 = p1.pose.position
    p2 = p2.pose.position
    self.ugv_data[ugv_name]["velocity"] = {
        "vx":(p2.x - p1.x)/(now - self.ugv_last_time).to_sec(),
        "vy":(p2.y - p1.y)/(now - self.ugv_last_time).to_sec()
    }
    self.ugv_last_time = now
```

3. 最终参数：

```
# 包围圈半径缩小函数
rd = 4
r = max(1, 30 - rd * elapsed_time)
w = 0.05
# 潜在碰撞距离阈值
d_thresh_u = 1
d_thresh_s = 3
# 避障控制权重
kc_min_u = 4
kc_min_s = 45
# 阶段切换距离阈值
switch = 3
# 编队跟踪
g1_ka = 80 / 180
g2_ka = 80 / 180
# 控制参数
c = 4.0
b = 2.0
sigma = 0.01
```

4. 计算状态向量：

```
self.data[uav]["hpx"] =
kr * math.cos(w * elapsed_time + ka * math.pi * ki / kn)
self.data[uav]["hpy"] =
kr * math.sin(w * elapsed_time + ka * math.pi * ki / kn)
self.data[uav]["hvx"] =
-w * kr * math.sin(w * elapsed_time + ka * math.pi * ki / kn)
self.data[uav]["hvy"] =
w * kr * math.cos(w * elapsed_time + ka * math.pi * ki / kn)
```

5. 切换控制阶段：

```
""" 是否切换为打击阶段 """
d2c = math.sqrt((pos_uav.x - pos_ugv.x) ** 2 + (pos_uav.y - pos_ugv.y) ** 2)
if d2c <= switch:
    kr = 1
    ka = 2
    tag = True
```

6. 无人机组内碰撞检测：

```
dx = pos_uav.x - pos_other.x
dy = pos_uav.y - pos_other.y
d = math.sqrt(dx ** 2 + dy ** 2)
if  d <= d_thresh_u:
    d = d_thresh_u - d
    theta = math.atan2(abs(dy), abs(dx))
    abs_x = d * math.cos(theta)
    abs_y = d * math.sin(theta)
    kc_u = kc_min_u / d
    delta_x += kc_u * abs_x if dx > 0 else -abs_x * kc_u
    delta_y += kc_u * abs_y if dy > 0 else -abs_y * kc_u
```

7. 圆柱体碰撞检测：

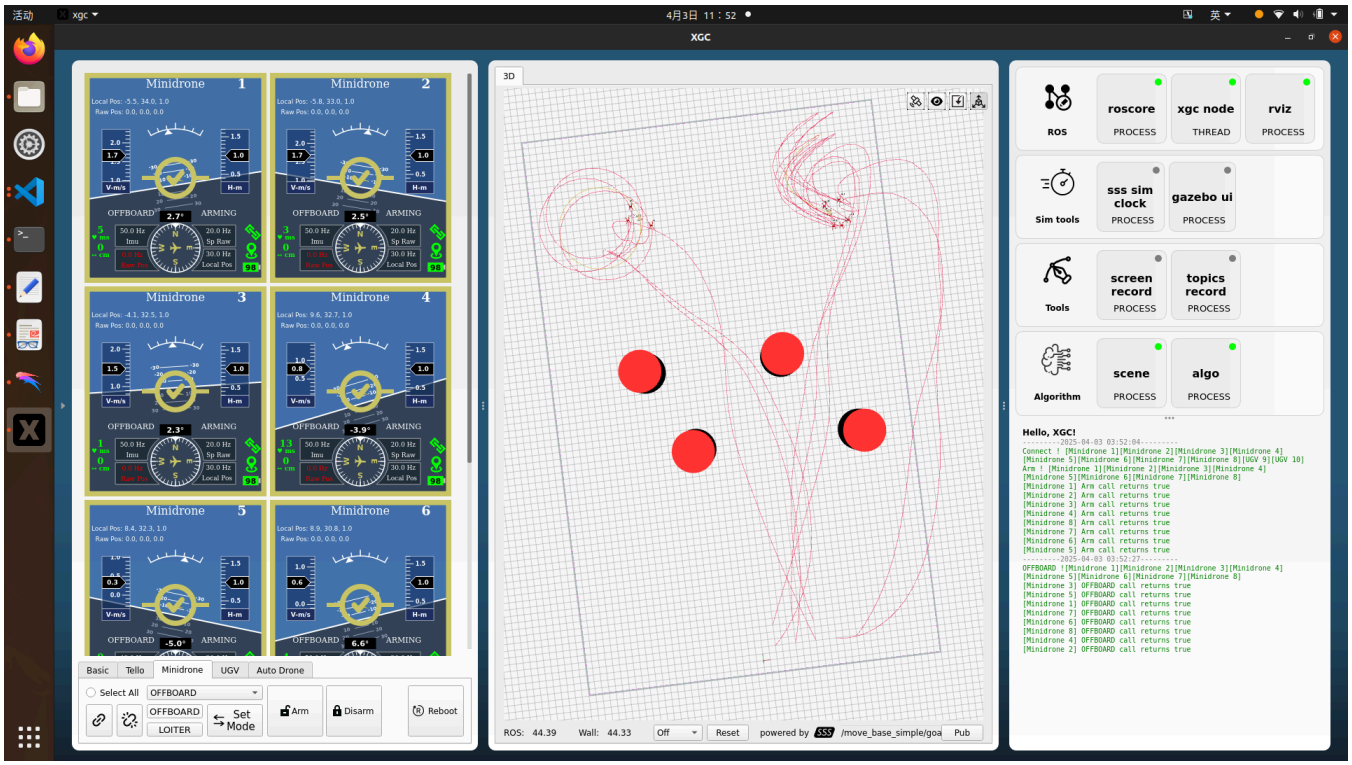
```
for o in self.obstacles:
    dx = pos_uav.x - o['x']
    dy = pos_uav.y - o['y']
    d = math.sqrt(dx ** 2 + dy ** 2)
    d_thresh_o = d_thresh_s + o['radius']
    if  d <= d_thresh_o:
        d = d_thresh_o - d
        theta = math.atan2(abs(dy), abs(dx))
        abs_x = d * math.cos(theta)
        abs_y = d * math.sin(theta)
        kc_s = kc_min_s / d
        delta_x += kc_s * abs_x if dx > 0 else -abs_x * kc_s
        delta_y += kc_s * abs_y if dy > 0 else -abs_y * kc_s
```

8. 补偿控制输入：

```
acc_x += w**2 * kr * math.cos(w * elapsed_time + ka * math.pi * ki / kn)
acc_y += w**2 * kr * math.sin(w * elapsed_time + ka * math.pi * ki / kn)
```

结果展示

> 此处仅展示截图，详见视频： lab3_OA_group.webm

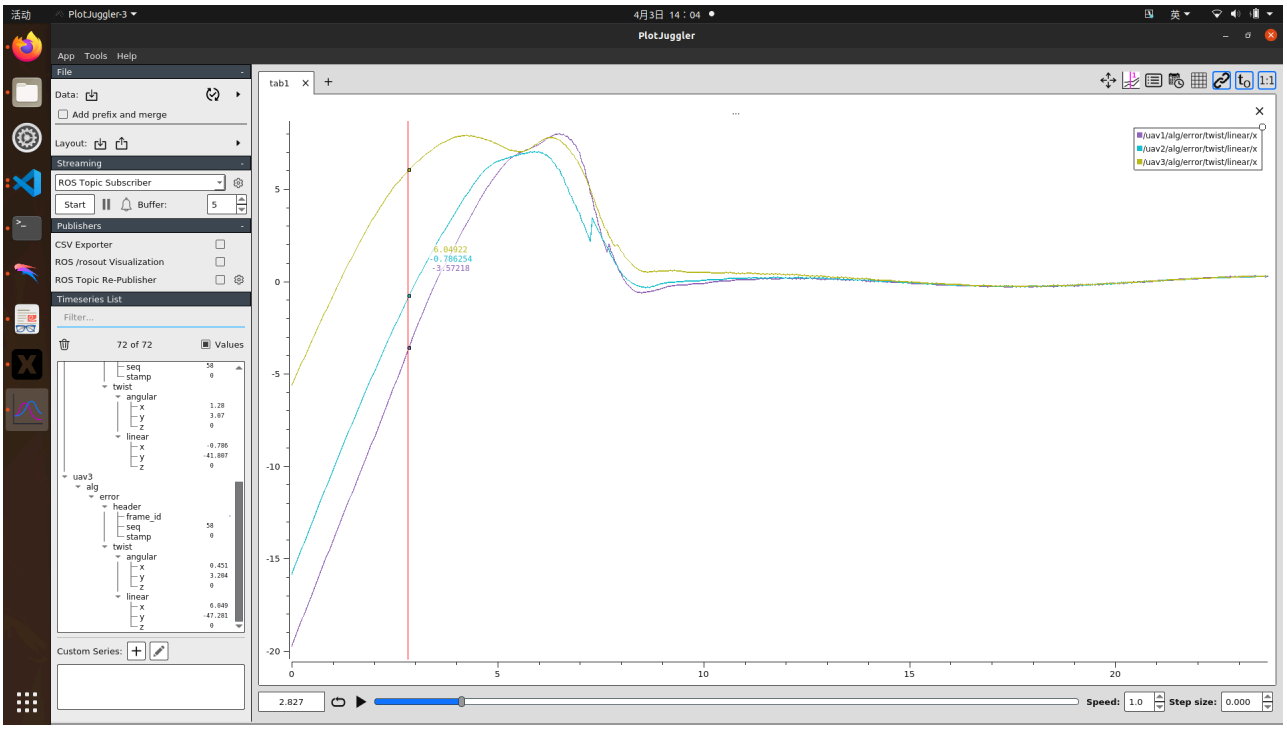


误差分析

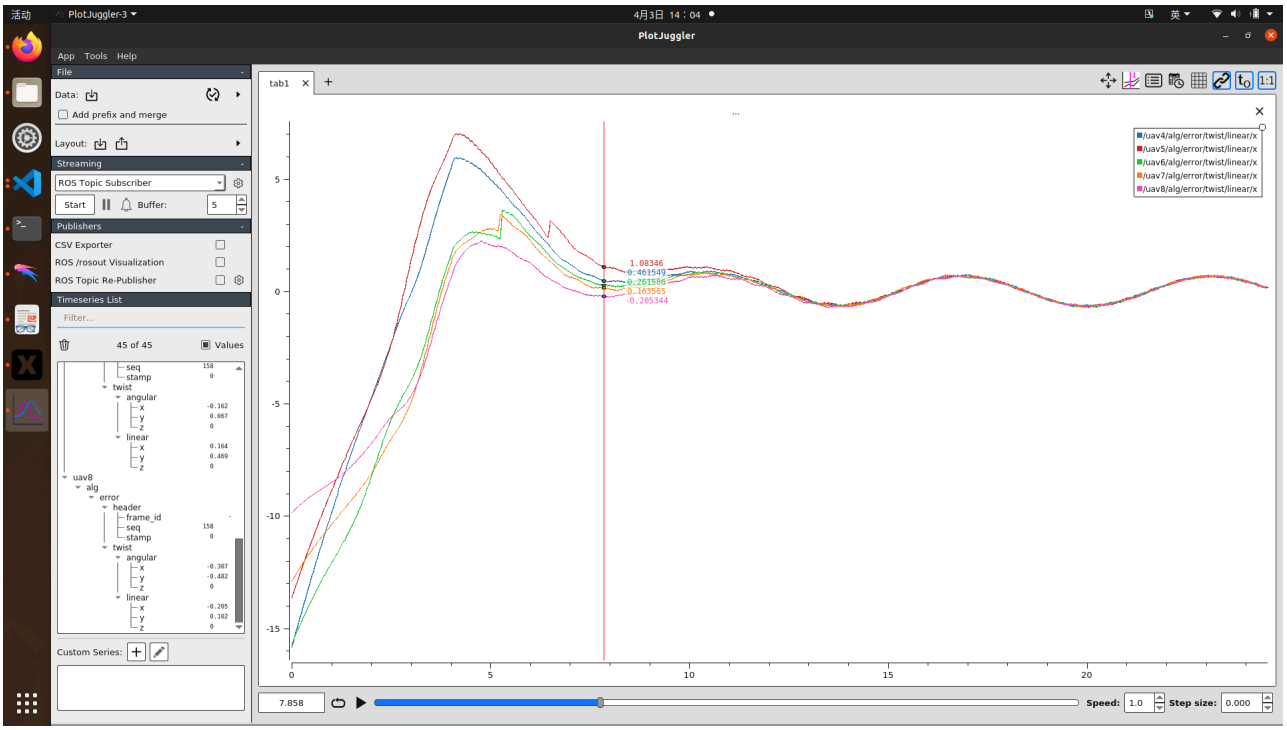
> 此处仅展示截图，详见视频： lab3_analyze_group1.webm 和 lab3_analyze_group2.webm

（见下页）

- Group1



- Group2



1. **现象说明**：`group1` 的误差曲线最终区域收敛，`group2` 的误差曲线最终趋于发散
2. **现象分析**：`group2` 有 `5` 台无人机，组内无人机之间的避障比较难控制，正如`问题分析`阶段所言，若要满足无人机均在 `1m` 圆环内，同时互相保持 `0.5m` 的安全距离，则`均匀包围`是最简单的设计方式。在`均匀包围`的情况下，每两个相邻的无人机距离为 `0.618m`，恰好满足要求，十分接近规定的安全距离，因此无人机在围捕阶段也会是频繁受

到避障的影响，难以消除旋转误差。同时，为了进一步观察避障对误差的影响，我设计了两个对比实验：一个是取消了围捕阶段的避障 ，详见：lab3_non_OA_group.webm 和增大围捕阶段的避障权重，详见：lab3_OA_over.webm