1.代码文档简要说明

当前文档仅涉及到飞机部分，小车的代码说明不在这个文档里面说明。飞机的程序主要包含，定位模块，避障规划模块，决策模块，以及目标检测模块。其中定位模块，避障规划模块都是用开源算法，目标检测使用的是yolov5，决策模块是基于行为树库上做的二次开发代码，因此对于开源算法库模块，文档里仅提供如何应用，以及简单的算法参数配置，至于算法内更深层的原理以及参数调优，需要读者查阅算法对应的论文以及阅读开源算法源码。因为决策模块是我们内部开发，因此文档会对决策模块使用

2. 程序编译，运行以及环境配置

2.1程序编译

```
# 拷贝代码，如果是真机拷贝至NX内，仿真有专门的文档这里忽略
# 如代码拷贝至 /home/nvidia/27com_ws/src 里面
# 如果代码第一次编译，赋予执行脚本权限
sudo chmod +x /home/nvidia/27com_ws/src/livox_ros_driver2/build.sh
# 执行编译命令
/home/nvidia/27com_ws/src/livox_ros_driver2/build.sh ROS1

##############################################################################################

# 如果代码之前编译过，这次仅增量编译
catkin_make
```

如果程序编译过程需要问题，参考 2.3环境配置及问题解决

2.2 程序运行说明

程序运行前，先检查ROS运行环境是是否收到干扰，打开/home/nvidia/.bashrc文件，如果环境配置不对，就按照下面的方法更改

```
sudo gedit ~/.bashrc
....

#注释除ros内的环境以外的环境变量，避免其他环境的冲突干扰
#如:
#source /home/nvidia/realsense_ws/devel/setup.bash
#source /home/nvidia/catkin_ws/devel/setup.bash
```

启动传感器shell脚本(sensor.sh)说明

```
#启动d435i相机驱动程序
gnome-terminal -x bash -c "source $HOME/realsense_ws/devel/setup.bash;roslaunch
realsense2_camera rs_aligned_depth.launch; exec bash"
sleep 3s

#启动激光雷达驱动程序
gnome-terminal -x bash -c "source $HOME/livox_ws/devel/setup.bash;roslaunch
livox_ros_driver2 msg_MID360.launch; exec bash"
sleep 3s

#启动mavros与飞控通信节点
gnome-terminal  -x bash -c "roslaunch mavros px4.launch; exec bash"
```

算法模块启动脚本(func_planner.sh)

```
#启动定位算法模块
gnome-terminal --tab  -e 'bash -c "source $HOME/27com_ws/devel/setup.bash; roslaunch
faster_lio mapping_mid360.launch rviz:=false; exec bash;"'
sleep 3s

#启动路径规划可视化节点
gnome-terminal --tab  -e 'bash -c "source $HOME/27com_ws/devel/setup.bash; roslaunch
ego_planner rviz.launch; exec bash"'
sleep 5s

#启动目标检测模块
gnome-terminal --tab  -e 'bash -c "source $HOME/27com_ws/devel/setup.bash; python3
$HOME/27com_ws/src/object_det/scripts/det.py; exec bash"'
sleep 5s

#启动路径规划节点
gnome-terminal --tab  -e 'bash -c "source $HOME/27com_ws/devel/setup.bash; roslaunch
ego_planner run_in_exp_310_with_vins.launch; exec bash"'
sleep 3s

#启动决策模块
gnome-terminal --window  -e 'bash -c "source $HOME/27com_ws/devel/setup.bash;roslaunch
mission_pkg test_planner.launch; exec bash;"'

#以下是终端调试窗口
gnome-terminal --window  -e 'bash -c "rostopic echo /mavros/setpoint_raw/local; exec bash;"'
gnome-terminal --window  -e 'bash -c "rostopic echo /planning/pos_cmd; exec bash;"'
gnome-terminal --window  -e 'bash -c "rostopic echo /mavros/local_position/pose; exec
bash;"'
gnome-terminal --window  -e 'bash -c "rostopic echo /move_base_simple/goal; exec bash;"'
```

启动方式很简单就是运行两个shell脚本

2.3 环境配置及问题解决

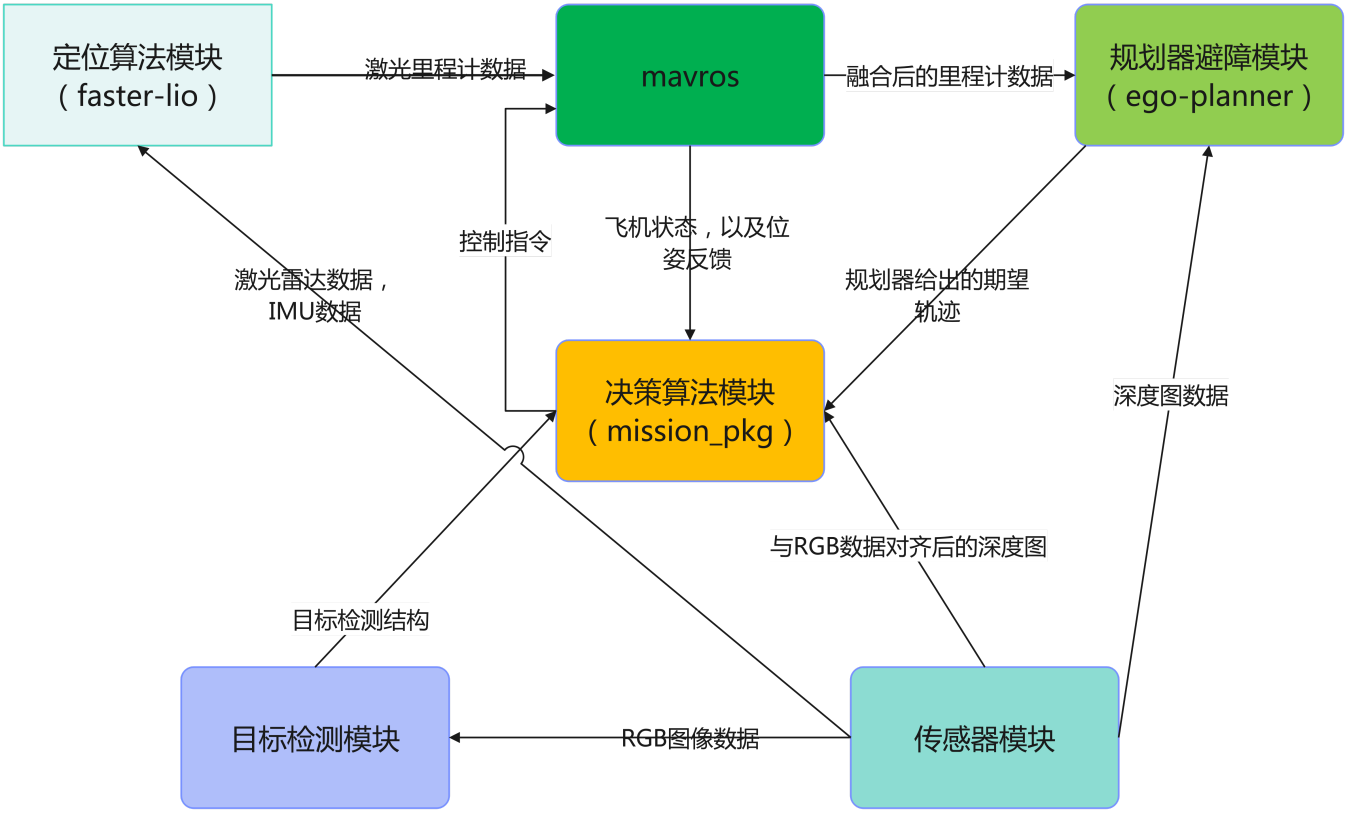
310飞机硬件上的系统适配不在这个文件里说，参考另外的文档或者联系卓翼技术支持远程。

主要解决的问题是（1. rviz可视化黑屏问题，2.解决启动传感器报错问题）

3.程序框架说明

3.1 程序框架结构与简要说明

下面给出程序的整体架构图以及及各个模块之间的数据流向



可以看到决策模块作为流入的数据最多。

3.2传感器配置说明

3.2.1仿真传感器配置与使用

仿真传感器配置通过加载一个config.json文件去实现，对于比赛，我们需要一个激光雷达传感器，一个前视相机，一个深度相机，有这么几点需要注意的地方，仿真里面可以改变相机的视场角，但是真机的相机的视场角往往比较下，如果：RGB相机的市场角只有50度，深度相机的稍大一点，为了保证算法从仿真迁移到真机后，减少真机上的调试工作，应该把视场角配置的与真机一致，但是比赛的时候，仿真赛可以不与真机一致。

仿真的传感器配置文件在src/sensor_pkg/Config.json，内容如下：

```
{
    "VisionSensors":[
        {
            "SeqID":0,
```

```
            "TypeID":23,
            "TargetCopter":1,
            "TargetMountType":0,
            "DataWidth":64,
            "DataHeight":272,
            "DataCheckFreq":10,
            "SendProtocol":[1,127,0,0,1,9999,0,0],
            "CameraFOV":0,
            "SensorPosXYZ":[0,0,-0.1],
            "SensorAngEular":[0,0,0],
            "otherParams":[100,0,0,0,0,0,0,0]
        },
        {
            "SeqID":1,
            "TypeID":1,
            "TargetCopter":1,
            "TargetMountType":0,
            "DataWidth":640,
            "DataHeight":480,
            "DataCheckFreq":30,
            "SendProtocol":[1,127,0,0,1,9998,0,0],
            "CameraFOV":90,
            "SensorPosXYZ":[0.2,0,0],
            "SensorAngEular":[0,0,0],
            "otherParams":[0,0,0,0,0,0,0,0]
        },
        {
            "SeqID":2,
            "TypeID":2,
            "TargetCopter":1,
            "TargetMountType":0,
            "DataWidth":640,
            "DataHeight":480,
            "DataCheckFreq":30,
            "SendProtocol":[1,127,0,0,1,9997,0,0],
            "CameraFOV":90,
            "SensorPosXYZ":[0.2,0,0],
            "SensorAngEular":[0,0,0],
            "otherParams":[0.1,12,0.001,0,0,0,0,0]
        }
    ]
}
```


sensor:0是激光雷达，1，2分别代码RGB图像与深度图像，为了保证数据对其，把RGB图与深度图的相关参数配置成一样，如视场角，分辨率，安装位姿。为了方便后续算法使用。因为在结算气球坐标的时候用到了这个数据对气候后的深度图。

关于RflySim平台传感器更详细的配置，参考https://rflysim.com/doc/zh/或者从安装平台的目录下How to use.pdf文件进入

3.2.2 真机传感器配置与使用

真机传感器已经是固定了，一个激光雷达，一个相机，一个d435i，激光雷达与下视相机是固定的，而d435i我们可以拿某一部分数据，如RGB，和深度图。其他的数据可以屏蔽。

由于激光雷达，与下视相机驱动上没有什么可改的，正常启动就行

```
#激光雷达启动命令
gnome-terminal -x bash -c "source $HOME/livox_ws/devel/setup.bash;roslaunch
livox_ros_driver2 msg_MID360.launch; exec bash"
```

由于launch文件内的配置以及配置好了，这里不在重复描述。

下视相机启动（假设你使用系统已经做过更新）

```
roslaunch usb_cam usb_cam-test.launch
```

d435i相机启动

```
gnome-terminal -x bash -c "source $HOME/realsense_ws/devel/setup.bash;roslaunch
realsense2_camera rs_aligned_depth.launch; exec bash"
```

上面启动的这个脚本不包含，相机IMU,红外灰度图像。

3.3 定位模块说明

3.3.1 参数配置

定位算法模块可配置的参数文件有两个一个是启动的launch， 另一个就是算内部参数配置文件。

启动文件位置在src/faster-lio-main/faster-lio-main/launch文件目录下，算法内部参数配置文件在src/faster-lio-main/faster-lio-main/config 目录下。

launch启动文件，以真机使用launch文件（src/faster-lio-main/faster-lio-main/launch/mapping_mid360.launch）为例:

```
<launch>
<!-- Launch file for Livox MID360 LiDAR -->

    <arg name="rviz" default="true" />

    <!--包含算法内部的参数配置文件-->
    <rosparam command="load" file="$(find faster_lio)/config/mid360.yaml" />


    <param name="feature_extract_enable" type="bool" value="0"/>
    <param name="point_filter_num" type="int" value="3"/>
    <param name="max_iteration" type="int" value="3" />
    <param name="filter_size_surf" type="double" value="0.5" />
    <param name="filter_size_map" type="double" value="0.5" />
    <param name="cube_side_length" type="double" value="1000" />
    <param name="runtime_pos_log_enable" type="bool" value="0" />
```

```
    <node pkg="faster_lio" type="run_mapping_online" name="laserMapping" output="screen">
       <remap from="/Odometry" to="/mavros/odometry/out" />  <!-- 构建话题映射，将激光lio话题转变成mavros接受的话题-->
    </node>

    <group if="$(arg rviz)">
    <node launch-prefix="nice" pkg="rviz" type="rviz" name="rviz" args="-d $(find faster_lio)/rviz_cfg/loam_livox.rviz" />
    </group>

    <!-- 下面是构建一个完整的tf树，为了方便将激光里程计发个飞控-->
    <node pkg="tf" type="static_transform_publisher" name="laser_base_link" args="0 0 0 0 0 0 base_link lidar 100" />
    <node pkg="tf" type="static_transform_publisher" name="base_link_body" args="0 0 0 0 0 0 body base_link 100" />
    <node pkg="tf" type="static_transform_publisher" name="slam_mavros" args="0 0 0 0 0 0 odom camera_init  100" />
    <node pkg="tf" type="static_transform_publisher" name="odom_map" args="0 0 0 0 0 0  map odom 1000" />

</launch>
```

上面文件内容除了注释部分的代码，其他的来自原始开源算法内部的launch文件，这里是仅仅是将faster-lio算法应用到无人机上做了一个适配工作

算法内部参数配置文件，以真机为例（src/faster-lio-main/faster-lio-main/config/mid360.yaml）

```
common:
    lid_topic:  "/livox/lidar" #配置激光雷达数据
    imu_topic:  "/livox/imu" #配置雷达imu数据
    time_sync_en: false         # ONLY turn on when external time synchronization is really not possible

preprocess:
    lidar_type: 1                # 1 for Livox serials LiDAR, 2 for Velodyne LiDAR, 3 for ouster LiDAR,
    scan_line: 4
    blind: 0.5 #盲区值，即点云最小有效距离(可以过滤扫描在载体上的点)

mapping:
    acc_cov: 0.1
....
```

3.3.1.1 仿真参数配置

对于定位算法模块而言，仿真与真机的launch启动文件几乎相同，唯独不同的是加载的算法内部配置文件参数，打开文件src/faster-lio-main/faster-lio-main/launch/rflysim_sim.launch

里面改动地方就是传递算法内参数配置文件路径，如：

```
<rosparam command="load" file="$(find faster_lio)/config/rflysim_mid360.yaml" />
```

而算法内参数配置文件内容也仅仅改动了话题名称，和点云数据类型(lidar_type),打开文件(src/faster-lio-main/faster-lio-main/config/rflysim_mid360.yaml)

```
common:
    lid_topic:  "/rflysim/sensor0/mid360_lidar"
    imu_topic:  "/rflysim/imu"
    time_sync_en: false         # ONLY turn on when external time synchronization is really
not possible

preprocess:
    lidar_type: 3                # 1 for Livox serials LiDAR, 2 for Velodyne LiDAR, 3 for
ouster LiDAR,
    scan_line: 4
    blind: 0.5
    time_scale: 1e-3
```

3.3.1.2 真机参数配置

真机定位模块参数配置看前面的内容，这里就不赘述了。

3.3.2 注意事项

仿真虽然使用了真机里面雷达数据格式，但是仿真仅仅是填充了x,y,z信息，不包含强度等其他信息，不过不影响定位算法模块使用，因faster-lio里面也没有使用出点的x,y,z信息以外的属性。

3.4 目标检测模块说明

3.4.1 目标检测参数配置

今年的比赛目标为一个框与一个气球，气球为红色，贴出目标检测的关键代码

```
class_name =["frame","balloon"]
# ------------------------------参数--------------------------------
# yolo模型路径
Model_path = script_dir + '/Model/frame_balloon.pt'

...

def roi_track(self,img,left:tuple,right:tuple,threshold =200):
        roi = img[left[1] :right[1],left[0]:right[0]] #img[行, 列]
        red_channel = roi[:,:,2]
        red_mask = red_channel > threshold
        red_pixel_coords = np.column_stack(np.where(red_mask))
        # 计算红色像素的中心位置（相对于ROI）
        if len(red_pixel_coords) > 0:
            center_roi = np.mean(red_pixel_coords, axis=0).astype(int)
```

```
            # 将中心位置转换为原始图像坐标系
            center_original = (center_roi[1] + left[0], center_roi[0] + left[1])
            # print(f"红色像素的中心位置（原始图像坐标系）: {center_original}")
            red_pixels_visualization = np.zeros_like(roi)
            red_pixels_visualization[red_mask] = [0, 0, 255]  # 将红色像素点标记为红色

            # 将红色像素点可视化结果覆盖到原始图像的ROI区域
            img[left[1] :right[1], left[0]:right[0]][red_mask] = [0, 0, 255]

            # 绘制感兴趣区域（ROI）的矩形框
            # cv2.rectangle(img, (left), right, (0, 255, 0), 2)
            self.show_img = True
            #cv2.imshow("segment",img)
            #cv2.waitKey(1)

        else:
            print("没有找到红色像素")
            center_original = None
        if center_original is not None:
            cv2.circle(img, center_original, 5, (0, 255, 0), -1)  # 在中心位置画一个绿色圆点
        return center_original
    ...
cnt =  self.roi_track(img_yolo,xy[0:2],xy[2:4],60)
```

与检测框不同，对于气球的检测需要计算中心位置，有时候检测框的中心不一定是目标中心位置，因此，在代码里面需要使用目标分割，当然可以使用yolo里面高级的功能分割目标，但是由于目标的特殊性（红色），我们可以直接通过颜色通道过滤，这样避免了计算资源的浪费。

3.4.1.1 仿真参数配置

仿真目标检测有出入，因为很多人电脑性能的原因，不能全开性能，如果开启全功能模式（就是上面代码使用yolo进行目标检测），那么不可避免要设置红色通道滤值参数cnt = self.roi_track(img_yolo,xy[0:2],xy[2:4],60)，除此之外还需配置传感器话题数据

```
topic1 = "/rflysim/sensor1/image_rgb" #前视相机的图像
```

3.4.1.2真机参数配置

真机对应话题名称为

```
topic1 = "/camera/color/image_raw" #前视相机的图像
```

3.4.2 注意事项

目标检测后数据频率会降低，达不到30hz，但是如果需要使用目标检测的结果和深度图做位置解算，需要使用对齐的数据，这个在决策模块里面有使用，后续再讲。

3.5 避障模块说明

3.5.1 避障模块参数配置

真机节点启动launch文件在src/ego-planner/src/planner/plan_manage/launch/run_in_exp_310_with_vins.launch里面，接下里看看里面关键的几个参数

```
<launch>
  <!-- 配置地图大小,单位米 -->
  <arg name="map_size_x" value="40.0"/>
  <arg name="map_size_y" value="40.0"/>
  <arg name="map_size_z" value=" 3.0"/>

  <!-- 输入里程计数据-->
  <arg name="odom_topic" value="/mavros/local_position/odom" />

  <!--算法的主要参数-->
  <include file="$(find ego_planner)/launch/advanced_param_droneyee.xml">

    <arg name="map_size_x_" value="$(arg map_size_x)"/>
    <arg name="map_size_y_" value="$(arg map_size_y)"/>
    <arg name="map_size_z_" value="$(arg map_size_z)"/>
    <arg name="odometry_topic" value="$(arg odom_topic)"/>

     <!-- 配置深度图数据 -->
    <arg name="depth_topic" value="/camera/depth/image_rect_raw"/>


    <!-- 深度相机的内参 -->
    <arg name="cx" value="321.04638671875"/>
    <arg name="cy" value="243.44969177246094"/>
    <arg name="fx" value="387.229248046875"/>
    <arg name="fy" value="387.229248046875"/>

    <!-- 轨迹上的速度和加速度最大值设置-->
    <arg name="max_vel" value="0.3" />
    <arg name="max_acc" value="0.5" />
  </include>

  <!-- trajectory server -->
  <node pkg="ego_planner" name="traj_server" type="traj_server" output="screen">
    <remap from="/position_cmd" to="planning/pos_cmd"/>

    <remap from="/odom_world" to="$(arg odom_topic)"/>
    <param name="traj_server/time_forward" value="1.0" type="double"/>
  </node>
```

```
  <node pkg="waypoint_generator" name="waypoint_generator" type="waypoint_generator"
output="screen">
    <remap from="~odom" to="$(arg odom_topic)"/>
    <remap from="~goal" to="/move_base_simple/goal"/>
    <remap from="~traj_start_trigger" to="/traj_start_trigger" />
    <param name="waypoint_type" value="manual-lonely-waypoint"/>
  </node>

</launch>
```

避障算法主要参数文件在src/ego-planner/src/planner/plan_manage/launch/advanced_param_droneyee.xml 里面，因为参数繁多这里也仅仅提到几个常用的参数

```
<param name="grid_map/obstacles_inflation"     value="0.23" />  <!-- 障碍物膨胀系数-->
<param name="grid_map/ground_height"        value="-0.01"/> <!-- 滤除地面点云，这里需要参考输入的
里程计数据是坐标系 -->
<!-- 设置可视化调试的“天花板”高度等 -->
    <param name="grid_map/virtual_ceil_height"   value="2.5"/>
    <param name="grid_map/visualization_truncate_height"   value="2.4"/>
  <param name="grid_map/depth_filter_mindist"   value="0.5"/>  <!-- 过滤掉浆保带来的点-->
```

一般情况，使用相机内参没什么特别达到问题，如果需要相机内参配置，需要查看话题/camera/depth/camera_info，在启动前视相机驱动程序后，使用命令 rostopic echo /camera/depth/camera_info 查看相机内参。输出内容包含如：K: [381.4123840332031, 0.0, 318.9013671875, 0.0, 381.4123840332031, 234.53665161]， 那么对应的fx: 381.4123840332031, fy: 381.4123840332031 ; cx: 318.9013671875; cy: 234.53665161

3.5.1.1 仿真参数配置

为了区分，仿真使用了另外的launch文件和算法发配置文件，但是只是小有改动如文件src/ego-planner/src/planner/plan_manage/launch/RFlySim.luanch

```
<!-- 配置一下TF树 -->
<node pkg="tf" type="static_transform_publisher" name="map_world" args="0 0 0 0 0 0  world
map 1000" />
  <node pkg="tf" type="static_transform_publisher" name="depth_base_link" args="0 0 0 0 0 0
base_link depth 1000" />
 <arg name="odom_topic" value="/mavros/odometry/out" />

 <include file="$(find ego_planner)/launch/RflySimParam0419.xml"> <!-- 使用了不一样的配置文件 --
>
```

而算法参数配置文件的内容一样。

3.5.1.2 真机参数配置

可以参考前面的内容，这里不在赘述

3.5.2 注意事项

在这里，仿真里面，使用的里程计数据应该直接来自定位算法模块的，因为传感器数据都已经走过时间戳对齐了，包括图像，雷达，imu等，但是使用融合后mavros 发出来里程计数据没有与外部传感器做时间对齐，因此在仿真里面使用避障算法使用的传感器应该使用定位算法发出的里程计数据，也许使用飞控发出的里程计数据并没有造成什么不好的后果，但是从数据分析的角度来讲，应该使用定位算法给出的里程计数据

3.6 决策模块说明

3.6.1 行为树简介

行为树除了对有限状态机结构优化，还支持动态加载的功能，因此非常适合使用大模型做决策，在载体与大模型直接的搭建一个便捷的桥梁，因此行为树是具身智能最好的工具。关于行为树的具体使用方法，参考官方文档https://www.behaviortree.dev/docs/Intro，在这文档里面主要，介绍节点的功能，以及如果组合使用完整一个复杂的调度决策模块

3.6.2 行为树各节点功能说明以及配置

节点名称	功能作用	参数说明
Takeoff	起飞	is_rc:是否使用遥控器控制切换模式和解锁； goal:飞机起飞的目标点
Land	降落	speed_z：使用Z方向的速度降落，use_speed：是否使用速度控制降落,还是直接使用Auto.Land模式
Hover	悬停	is_time_ctrl：是否控制悬停时间（-1：表示永久悬停，大于0表示悬停的时间）;hover_is_end 与 stop_hover 参数未在行为树里面使用
PlanNode	路径规划	enabel_yaw_rate:是否使用轨迹的角速度控制；enabel_yaw：是否使用轨迹的角度控制；planner_ctrl_type:控制方式（0：位置控制，1：速度控制）；enabel_planner:使能节点；goal_src: 目标点来源（0：通过xml文件设置，1：通过程序内给出）；goal_position:如果goal_src为0，那么这个值就是设置目标点参数
Hit	撞击	needle_ang_adapt：用来设置刺针角度一个偏移量；goal_tolerance：给规划器的目标不能障碍物上，因此需要有一个偏移量；hit_dist：小于该值时，不适用规划器避障，而直接刺破； method：刺破气球使用的方法(1:发现目标直接去刺，2：按照比赛规则，发现目标不直接测，而是等目标走完一圈再刺)；object_name: 目标检测里面需要刺破的目标名称；
CrossFrame	穿框	ctrl_speed:设置穿过框的速度，ctrl_type:控制飞机穿过框的方式（0：使用视觉伺服控制，1：使用路径规划的方式刺破），如果传感器视场角大，推荐使用视觉伺服控制，否则使用路径规划的方式，对于d435i相机，推荐使用路径规划的方式

节点名称	功能作用	参数说明
GoalYaw	位置+角度控制	is_set_point: 是否设置目标点控制角度，如果false,表示原地旋转；point：给定目标点；goal_yaw：设置飞机旋转的偏航角度（相对于local坐标系）

注意上面未提到的参数就是预留参数，未使用。代码里面除了上面已经列出的，还有其他节点，但是未使用，因此这里也不过多介绍了。

3.6.3 行为树节点组合方法和说明

因为比赛是一个流程化的方式进行，起飞->穿过框->绕过障碍物柱->刺破。 那么主线上就是只有takeoff, cross_frame, Hit， 但是单靠这个三个节点不足以完成整个功能，比如穿框之前得识别框，但是识别过程中，飞机需要保持悬停，再比如刺球的同时需要，避障，同时可能目标丢失后，保持悬停等，因此需要多个过渡和辅助节点。

deom里面使用test_planner.xml文件组合使用行为节点

```
<?xml version="1.0" encoding="UTF-8"?>
    <root BTCPP_format="4" main_tree_to_execute="MainTree">
    <include path="takeoff.xml"/>
    <include path="planner_node.xml" />
    <include path="arrive_goal.xml" />
    <include path="land.xml" />
    <include path="hover.xml" />
    <include path="hit.xml" />
    <BehaviorTree ID="MainTree">
    <Sequence>
        <!-- <SetBlackboard name="goal" value="0.0;0.0;2.5" output_key="goal"/> -->
        <SetBlackboard name="is_rc" value="1" output_key="is_rc"/>
        <Script code=" P1_goal:='0.0;1.8;0.8' "/>  <!--P1点位置 -->
        <Script code=" P2_goal:='4.2;2.1;1.0' "/>  <!--P2点位置 -->
        <Script code=" P3_goal:='8;0.3;1.0' "/>  <!--P3点位置 -->
        <Script code=" P4_goal:='8;0.3;0.6' "/>  <!--P4点位置 -->
        <Script code=" goal_ori:='0;0.0;0' "/>  <!--P2目标点，飞机朝向 -->
        <SetBlackboard name="enabel_yaw" value="false" output_key="enabel_yaw" />
        <SetBlackboard name="enabel_yawrate" value="false" output_key="enabel_yawrate" />
        <SetBlackboard name="ctrl_type" value="0" output_key="ctrl_type" />
        <SetBlackboard name="hover_is_end" value="false" output_key="hover_is_end" />
        <SetBlackboard name="stop_hover" value="false" output_key="stop_hover" />
        <SetBlackboard name="is_time_ctrl" value="3.0" output_key="is_time_ctrl" />


        <Script code=" takeoff_goal:='0.0;0.0;0.8' "/>
        <SubTree ID="takeoff" is_rc="{is_rc}" goal="{takeoff_goal}"/>
        <SubTree ID="hover" hover_is_end="{hover_is_end}" stop_hover="{stop_hover}"
is_time_ctrl="{is_time_ctrl}" />
```

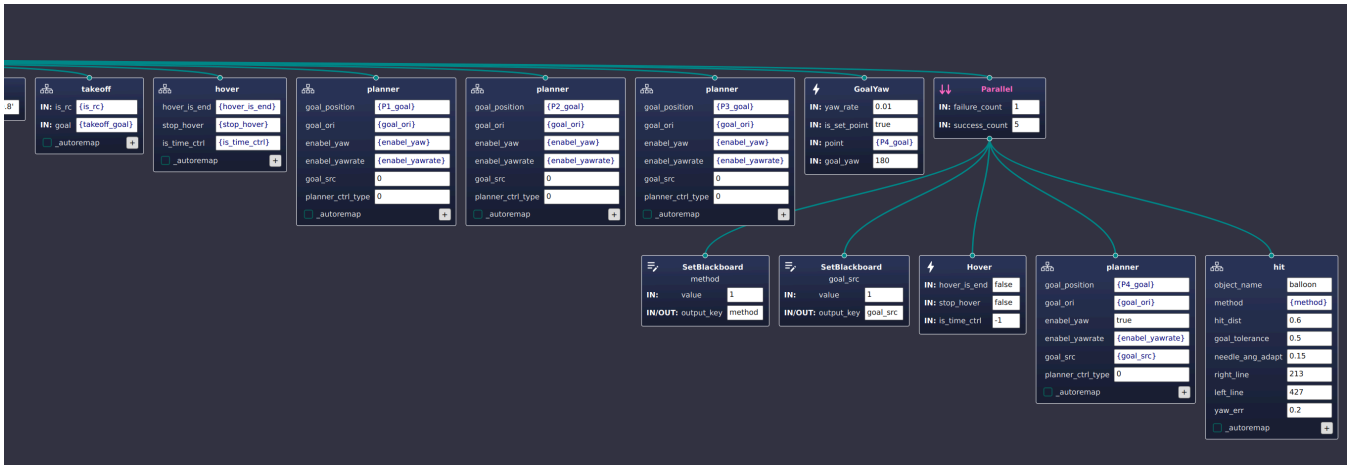
```
        <SubTree ID="planner" goal_position="{P1_goal}" goal_ori="{goal_ori}" enabel_yaw="
{enabel_yaw}" enabel_yawrate="{enabel_yawrate}" goal_src="0" planner_ctrl_type ="0" />
        <SubTree ID="planner" goal_position="{P2_goal}" goal_ori="{goal_ori}" enabel_yaw="
{enabel_yaw}" enabel_yawrate="{enabel_yawrate}" goal_src="0" planner_ctrl_type ="0" />
        <SubTree ID="planner" goal_position="{P3_goal}" goal_ori="{goal_ori}" enabel_yaw="
{enabel_yaw}" enabel_yawrate="{enabel_yawrate}"  goal_src="0" planner_ctrl_type ="0" />
        <!-- <SubTree ID="hover" hover_is_end="{hover_is_end}" stop_hover="{stop_hover}"
is_time_ctrl="{is_time_ctrl}" />  -->
        <!-- <SubTree ID="land" />       -->
        <GoalYaw goal_yaw="180" is_set_point="true" point="{P4_goal}" yaw_rate="0.01"/>
        <Hover  hover_is_end="false" stop_hover="false" is_time_ctrl="-1"/>

        <!-- hit balloon-->
        <Parallel success_count="6" failure_count="1">
        <SetBlackboard name="method" value="2" output_key="method" />
        <SetBlackboard name="goal_src" value="1" output_key="goal_src" />
         <Hover  hover_is_end="false" stop_hover="false" is_time_ctrl="-1"/>
         <DetectObj class_name="balloon" stop_detect="false"/>
         <SubTree ID="planner" goal_position="{P3_goal}" goal_ori="{goal_ori}"
enabel_yaw="true" enabel_yawrate="{enabel_yawrate}" goal_src="{goal_src}" planner_ctrl_type
="1" />
        <SubTree ID="hit" object_name="balloon" method="{method}" hit_dist="0.6"
goal_tolerance="0.5" needle_ang_adapt="0.15" right_line="213" left_line="427"
 yaw_err="0.2"/>
      </Parallel>


    </Sequence>
    </BehaviorTree>

  <TreeNodesModel>
    <SubTree ID="takeoff">
      <input_port name="is_rc" type="int">is remote controller operator</input_port>
      <input_port name="goal" type="BT::Position3D">Takeoff position ... </input_port>
    </SubTree>
    <SubTree ID="planner" />
    <SubTree ID="PointGoal"/>
    <SubTree ID="land"/>
    <Action ID="GoalYaw">
      <input_port name="yaw_rate" type="BT::AnyTypeAllowed">use we want
yaw_rate</input_port>
      <input_port name="is_set_point" type="BT::AnyTypeAllowed">if we want set
point</input_port>
      <input_port name="point" type="BT::AnyTypeAllowed">set we need point,but the
is_set_point is true</input_port>
      <input_port name="goal_yaw" type="BT::AnyTypeAllowed">we need to control the fcu
yaw</input_port>
    </Action>
  </TreeNodesModel>
```

对应在Groot2里面部分图为。



除了行为树本身，外部还是用ROS相关功能，看看launch文件内容，如在src/mission_pkg/launch/test_planner.launch文件里面，文件内容如下：

```
<launch>
    <param name="kx" value="0.005" type="double"/>
    <param name="ky" value="0.005" type="double"/>
    <param name="kz" value="0.005" type="double"/>
    <param name="v_max" value="0.4" type="double"/>
    <param name="img_w" value="640" type="int"/>
    <param name="img_h" value="480" type="int"/>
    <param name="fx" value="607.0685424804688" type="double"/>
    <param name="fy" value="606.6980590820312" type="double"/>
    <param name="cx" value="315.6202087402344" type="double"/>
    <param name="cy" value="248.33175659179688" type="double"/>

    <param name="rgb_ppy" value="240" type="int"/>
    <param name="rgb_ppx" value="320" type="int"/>
    <param name="rgb_fov_h" value="42.0" type="double"/>
    <param name="rgb_fov_v" value="42.0" type="double"/>
    <param name="min_score" value="0.7" type="double"/>
    <param name="planner_cmd_topic" value="/planning/pos_cmd" type="str"/>
    <param name="is_pause_planner" value="false" type="bool"/>
    <param name="depth_topic" value="/camera/aligned_depth_to_color/image_raw" type="str"/>
    <!-- 以下带port的参数名是给BT节点用的 -->
    <param name="stop_hover_port" value="false" type="bool"/>
    <param name="stop_detect_port" value="false" type="bool"/>
    <param name="hover_is_end_port" value="false" type="bool"/>

    <rosparam param="cam2body_R">[0,0,1,-1,0,0,0,-1,0]</rosparam>
    <rosparam param="cam2body_T">[0.2,0,0]</rosparam>

    <!-- 起飞后发一个目标点给飞控 -->
    <node pkg="mission_pkg" type="bt_ros" name="teset_planner" output="screen">
      <param name="config_path" value="$(find mission_pkg)/config/test_planner.xml"
type="str"/>
    </node>
</launch>
```

上述参数里面，kx,ky,kz v_max,为视觉伺服控制增益或衰减参数。 同样这里面也需要配置相机内参fx,fy,cx,cy ， 不过这个参数的看考话题/camera/aligned_depth_to_color/camera_info，因为Hit节点里面用到的是与RGB数据对齐后的深度图。因此，需要使用对齐后的相机内参。操作方法，启动相机传感器，用命令rostopic echo /camera/aligned_depth_to_color/camera_info ,查看，内参K: [609.431884765625, 0.0, 320.63995361328125, 0.0, 608.4337158203125, 246.36697387695312, 0.0, 0.0, 1.0] ，那么fx:609.431884765625; fy: 608.4337158203125, cx: 320.63995361328125; cy: 246.36697387695312， 其他的参数预留。

3.7 决策模块其他注意事项

提供的demo并未使用CrossFrame节点，并不是这个节点不能用，只是因为比赛框，只有上下移动，前后左右不动，且框的两侧都是封闭的（不会规划从一侧穿过的轨迹），这样比赛时只要给一个估计的框中心高度即可。之所以不使用CrossFrame，因为识别框的pt模型鲁棒性要根据检测情况调整，这个pt模型里面气球对接检测可以不用那么准确，因为求中心点还有一道颜色通道值滤除工作，因此可以在不微调模型的情况下完成比赛，省了很多工作。

其次还有个问题，在Hit节点里面并没有实现method为2的代码，因为这个需要接口小车速度来协同来实现，飞机并不知道小车是否已经走完一圈了，唯一判断的标准是，飞机发现小车从P1点往P2点行走，认为小车已经走完一圈了，但是这个判断需要记录一段时间的目标轨迹，如果小车速度太慢，判断就会出现错误，代码里提供了记录小车一段时间的轨迹，也给出了如何判断小车的行径方向，但是没在onRuning函数里面调用，在代码里面有注释，需要读者自行结合小车完善。

4.程序里面用到话题名称及发布着和订阅者

传感器话题这里不区分真机和仿真了，如需了解参考前面的篇章

话题名	发布者	订阅者	说明
/camera/color/image_raw	d435i相机驱动	目标检测节点	输入前视RGB图像
/camera/depth/image_rect_raw	d435i相机驱动	避障路径规划节点	原始深度图
/camera/aligned_depth_to_color/image_raw	d435i相机驱动	决策模块里面Hit节点	与RGB进行对齐后的深度图
/livox/lidar	mid360雷达驱动	定位算法模块	激光雷达点云数据
/livox/imu	mid360雷达驱动	定位算法模块	激光雷达IMU数据
/mavros/local_position/odom	mavros节点	ego-planner模块	飞机里程计数据
/mavros/odometry/out	定位算法模块	mavros节点	定位算法发出里程计数据

话题名	发布者	订阅者	说明
/mavros/setpiont_raw/local	决策模块里面多个节点,(Takeoff,PlanNode,Hit,Hover等)	mavros节点	控制飞机的话题
/mavros/state	mavros节点	决策模块里面多个节点	飞控状态话题
/objects	目标检测节点	决策模块里面多个节点	目标检测发出的目标
/planning/pos_cmd	ego-planner模块	决策模块里面PlanNode节点	规划器发出的轨迹数据
/move_base/sample/goal	决策模块里面Hit,PlanNode节点	ego-planner模块	给规划器的目标点

以上是模块通信的主要话题，当然还有其他调试用的话题，这里不在一一列举了。

5. 源码内文件夹功能说明

文件夹名称与分类基本与模块划分一一对应,文件夹列表如下

behaviortree_cpp： 这个行为数据的库

common_msgs: 里面包含了目标检测里面用到的目标数据结构封装，当然还有其他与本次比赛无关消息文件

```
#Objects.msg文件内容
std_msgs/Header header
uint8 source_id #1: front , 2:down
common_msgs/Obj[] objects
==========================================
#Obj.msg文件内容
std_msgs/Header header

string class_name
uint32 left_top_x
uint32 left_top_y
uint32 right_bottom_x
uint32 right_bottom_y
```

```
uint32 center_x
uint32 center_y
float32 score
```

ego-planner： 规划避障模块；

faster-lio-main: 定位算法模块；

livox_ros_driver2: 激光雷达驱动，注意在这工作空里面仅仅使用了它的数据结构，并未真正用它去连接雷达；

mission_pkg: 决策模块

object_det: 目标检测模块

sensor_pkg: 仿真里面使用，真机不适用它

那么传感器模块在哪呢？ 因为传感器模块是通用的，不管哪届的比赛，什么视觉算法都需要使用，不可能建立一个工作空间就拷贝一个传感器的驱动功能包到对应的工作空间，因此对于真机上的传感器模块分散在系统里面，如：激光雷达模块在 livox_ws 里面， d435i相机在Realsense-master里面， 下视相机驱动在系统内部/opt/ros/noetic/share/usb_cam 里面。

版本更新说明文档

版本号5.27（以更新日期 命名）

1.功能更新说明

-- 解决了飞机飞到P1点后掉头结束，轨迹规划后，突然向左转向的问题；

-- 解决飞机飞到P1点轨迹规划出来了，但是飞机不走的问题；

-- 增加视觉伺服刺球动作末端逻辑代码，后面会左阐述

2.更新主要代码以及参数说明

如果刺球动作在末端，目标检测已经不能正常识别到气球，这里有两个控制量，1.怎么判断已经到末端； 2. 增加一个到末端的标志。但是这样做只有一次刺破球的机会，与使用位置刺不同，位置控制次，可以发现目标就刺，可连续多次刺球。但是需要一个较远的距离（只有这样速度才够快，能够跟上小车，但是这样会带来另个问题，距离较远不适用规划避障，可能会装上障碍物），具体如何使用，自己权衡；

```
bool Hit::getVel(double &vx, double &vy, double &vz,double &yaw_rate)
{ //is_vis_servo_end 就是用来记录已经到刺破末端的标志
  if(is_vis_servo_end) //以及在刺破的末端
  { // 直接飞过去
    vx = v_max;
    vy = 0;
    vz = 0;
    yaw_rate = 0;
    return true;
```

```
  }
   ......

    if(ex < 20 && ey < 20 )
    { //这里的20 表示一个瞄准重点的范围值，小于该值，表示已经正对靶心
      vz = 0;
      vy = 0;
      vx = v_max;
      if(area_rate > 0.1)
      { // 目标面积占图像面积的比例大于这个值，同时也正对目标了
        is_vis_servo_end = true; //这是时候已经在刺破的末端
        ROS_INFO("the object area is too large,area_rate: %f",area_rate);
      }
    }
```

其他在onRunning里面的控制两都需要根据视觉伺服控制末端表示（is_vis_servo_end）来，因为如果这个表示false，表示一切按照正常逻辑执行，即发现目标，规划到目标近处的轨迹，另外里面用到的参数需要根据调试设置，如：目标占比图像的面积小于0.1认为已经在视觉伺服控制末端了，但是还有另外一个参数，就是当目标距离小于多少时从避障规划切换成视觉伺服控制，这个参数在xml里面(hit_dist)

Hit位置控制刺球的代码在onRunning后半部分，这个源码里没有注释，但是不响应，用户可自行更改

```
//位置控制此批，需要把hit_dist参数设置的大一点，拉大刺破距离，这速度才够快
//关键代码如下
 //如果气球位置离得近，且正对飞机刺针，直接飞过去刺破就行
    ROS_INFO("obj_center(%d,%d)",obj.center_x,obj.center_y);
    ROS_INFO("to object distance: %f",dist);
    // if(obj.center_x < left_line || obj.center_x > right_line)
    bool plan_flag = false;
    nh->getParam("is_pause_planner",plan_flag);
    if(!plan_flag)
    {
      nh->setParam("is_pause_planner", true);  //暂停planner node 去控制
      return NodeStatus::RUNNING;  //当次不控制
    }
    else{
      ROS_WARN("use point control in Hit Node,but Planner Node state not switch");
    }
    cmd.header.frame_id = "hit_cmd";
    //此时的目标坐标是气球表面的，因该往里刺
    // cmd.position.x = goal.pose.position.x;
    // cmd.position.y = goal.pose.position.y;
    // cmd.position.z = goal.pose.position.z;
    //为了保证刺破气球，需要尽可能的确保戳点切面与针垂直
    //计算到目标点yaw角。
    // 计算飞机当前飞机的yaw角度

    double dx = goal.pose.position.x - fcu_pose_ptr->pose.position.x;
    double dy = goal.pose.position.y - fcu_pose_ptr->pose.position.y;
    double dz = goal.pose.position.z - fcu_pose_ptr->pose.position.z;
    double length = sqrt(dx * dx + dy * dy + dz * dz);

    double unit_x = dx / length;
```

```
    double unit_y = dy / length;
    double unit_z = dz / length;

    // double cx, cy, cz;
   //把目标点往气球内处设置
    cmd.position.x = goal.pose.position.x + unit_x * goal_tolerance;
    cmd.position.y = goal.pose.position.y + unit_y * goal_tolerance;
    cmd.position.z = goal.pose.position.z + unit_z * goal_tolerance;



    // 使用atan2计算yaw角度（弧度）
    double tgt_yaw = atan2(dy, dx); //这是飞机与目标连线的yaw角度，如果这个角度与目标行径方向的角度相差大，就有可能出现飞机刺空的现象，
    /*优化建议
      保持飞机刺球的方向与气球的行径方向在同一直线上(保证刺针正对球心刺)，即可保证刺破，但是要保证避障的前提下，做到这个点并不容易(有可能在调整飞机姿态的时候就发生碰撞) */
      // double tol_yaw  = 0.2; //允许的角度偏差
      // if(abs(obj_yaw) < M_PI && fabs(tgt_yaw - obj_yaw) > tol_yaw)
      // { //这个可以根据目标速度预测目标位置，直接朝预测目标位置刺破，
      //   //这种情况需要计算目标移动速度，同时也需要计算飞机位置控制的速度，因为这时候飞机离目标已经很近了，使用别的方法可能效率更高
          //因此最简单的方法就是通过减小两个角度差值，旋转飞机角度，控制在（obj_yaw 与 tgt_yaw）的差值很小时，等目标行驶到obj_yaw，直接刺
           //保持飞机位置不变，仅仅控制飞机的yaw角度
      // }
    // if(obj_yaw > 0 && hit_method != Hit::Method::direct )
    { //这个时候目标可能还没走完比赛规则里的一个来回

    }
    // if(abs(obj_yaw) < M_PI && fabs(tgt_yaw - obj_yaw) > tol_yaw )
    // {

    // }


    // ROS_INFO("fcu_y  aw:%f, tgt_yaw:%f",fcu_yaw,tgt_yaw);
    cmd.yaw = tgt_yaw;

    if(cmd.position.z < needle_adapt)
    {
      cmd.position.z = needle_adapt;
    }
    tgt_pose_pub_ptr->publish(cmd); //得到目标点后，直接发给飞控，而不是使用发给规划器去做避障
  }
```

3.关于仿真

这个代码同样适用于仿真，只要替换相应的代码即可。这个代码里面也有一个小部分测试脚本的，sensor_rflysim.sh func_sim.sh 。这两个脚本仅仅加载了部分功能。即后半部分刺球动作的功能，如果需要用它在仿真里面跑完整的功能，还需补充test_sim.xml文件（src/missiong_pkg/config/test_sim.xml）；

4. 优化建议

全程使用速度控制，另外结合小车调试合适的参数

5.注意事项

代码内目标检测目标检测模块里面使用的模型默认是仿真的，需要切回仿真的模型打开目录 src/object_det/scripts/Model， 有这么三个重要的模型文件frame_balloon.pt，frame_balloon_real.pt，frame_balloon_sim.pt 。 frame_balloon.pt 为正在使用的，frame_balloon_real为真机的，frame_balloon_sim.pt 为仿真的。 假设需要真机的，把原有的frame_ballon.pt删除掉，拷贝一份frame_balloon_real.pt， 然后名字改为 frame_balloon.pt即可