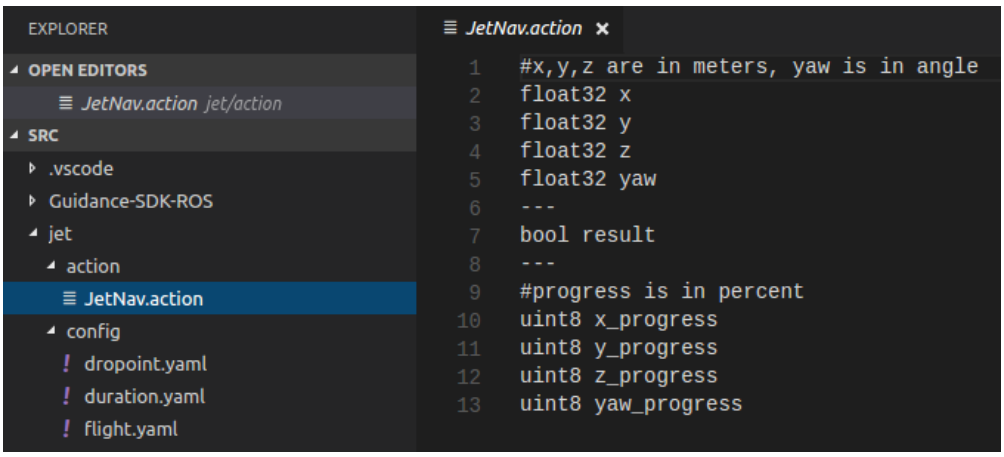
# 飞机控制逻辑说明文档

## 1.1 项目结构

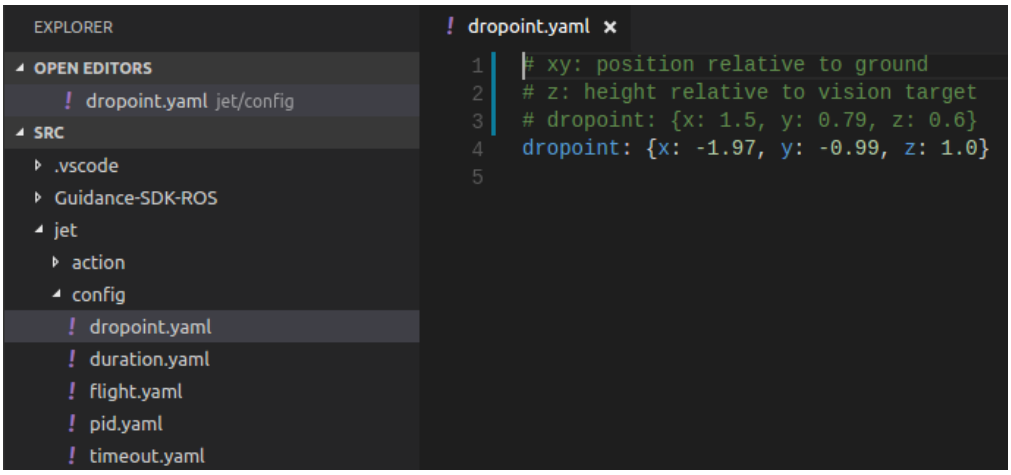
### 1.1.1 项目总体结构


### 1.1.2 飞机控制部分项目结构

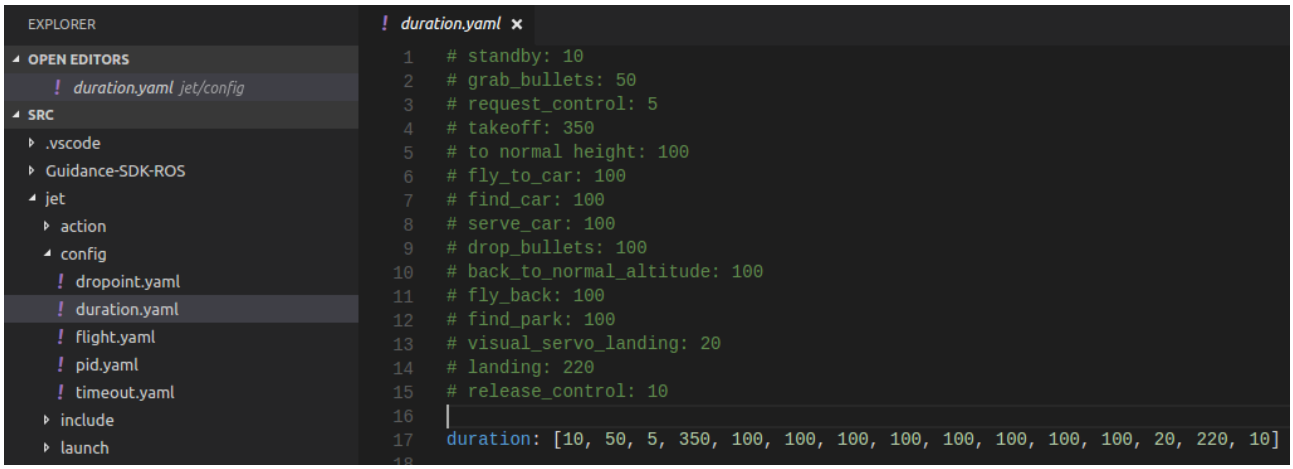

jet/action/JetNav.action: actionlib 导航



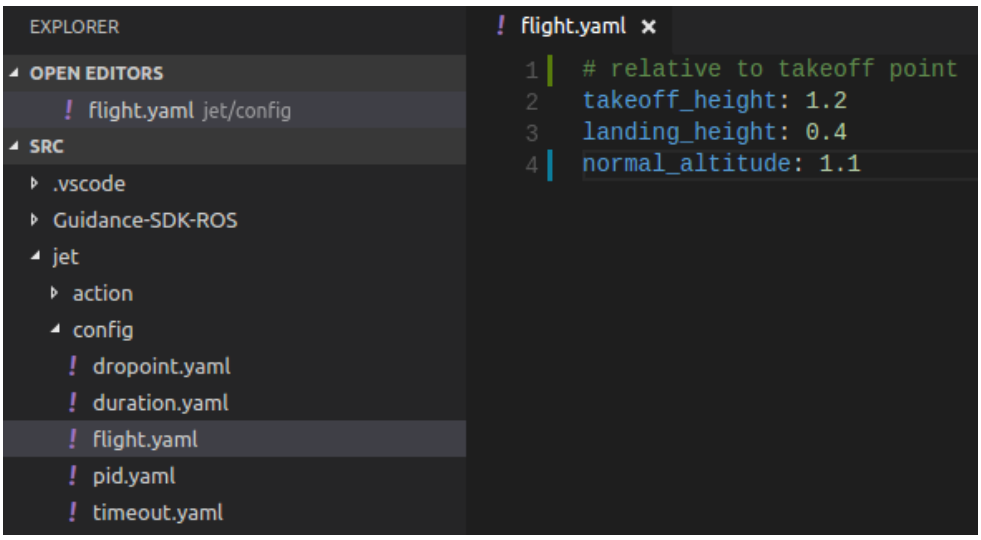

jet/config/dropoint.yaml: 空投点配置



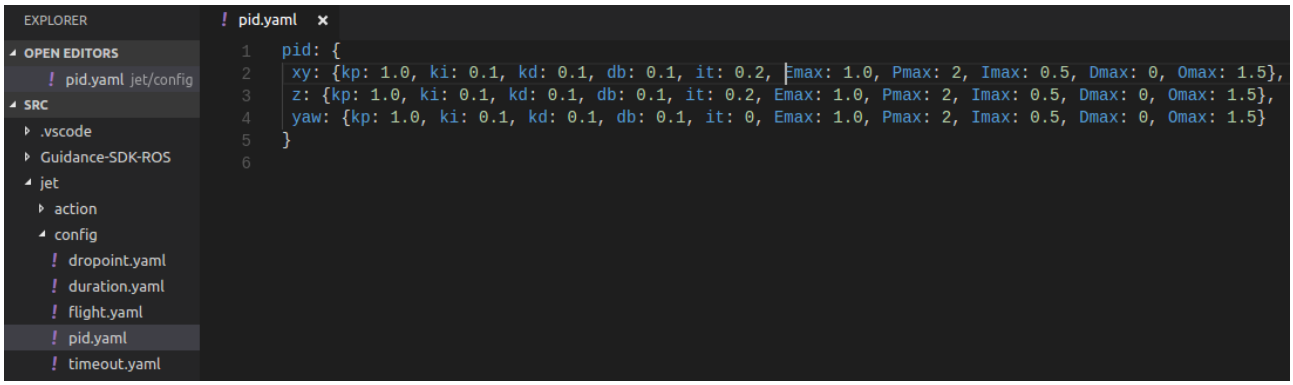

jet/config/duration.yaml: 过渡时间配置



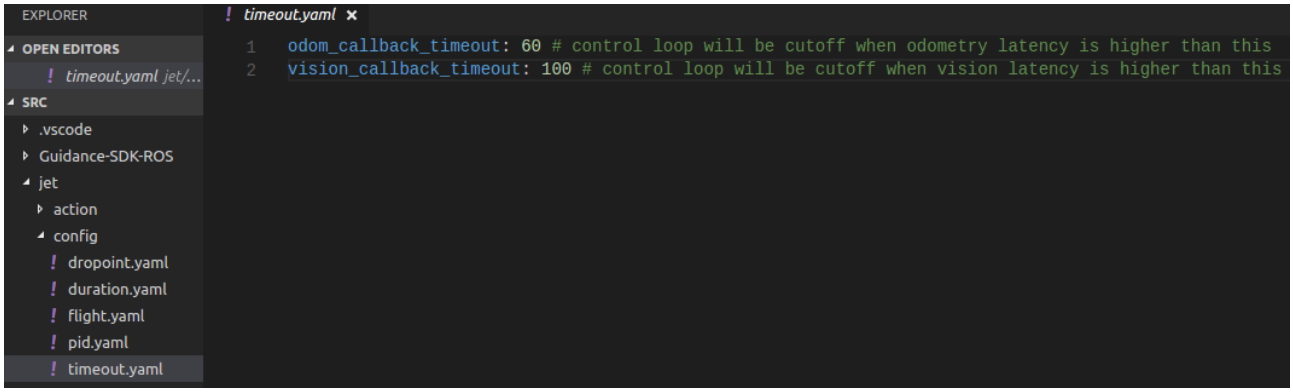

jet/config/flight.yaml: 飞行参数配置



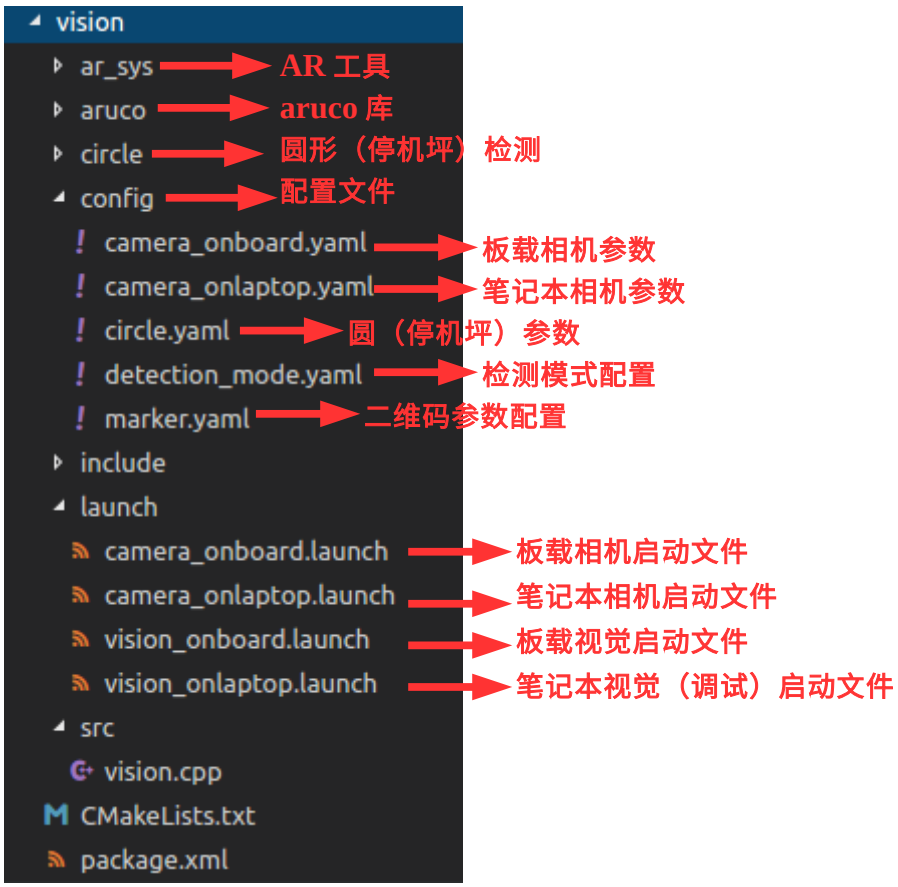

jet/config/pid.yaml: PID 闭环控制参数配置



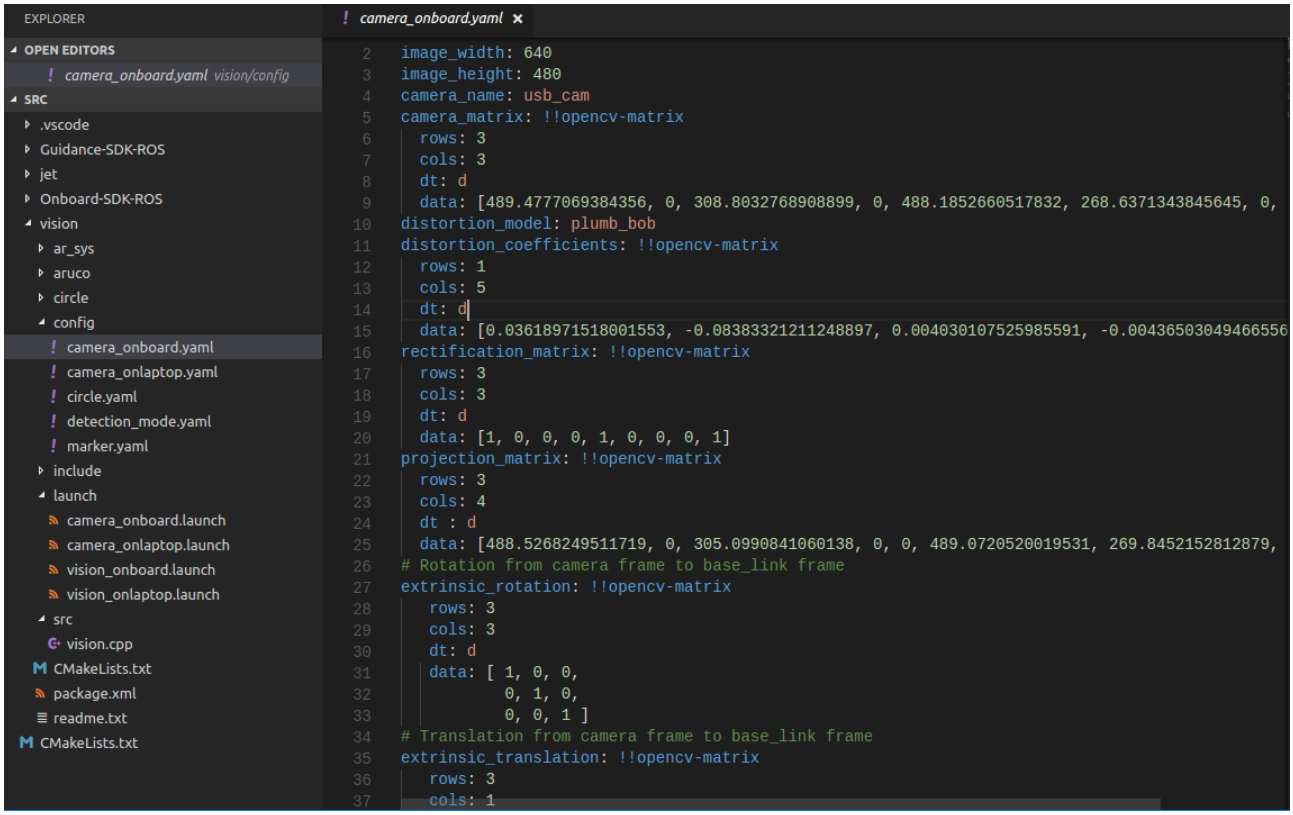

jet/config/timeout.yaml: 超时配置



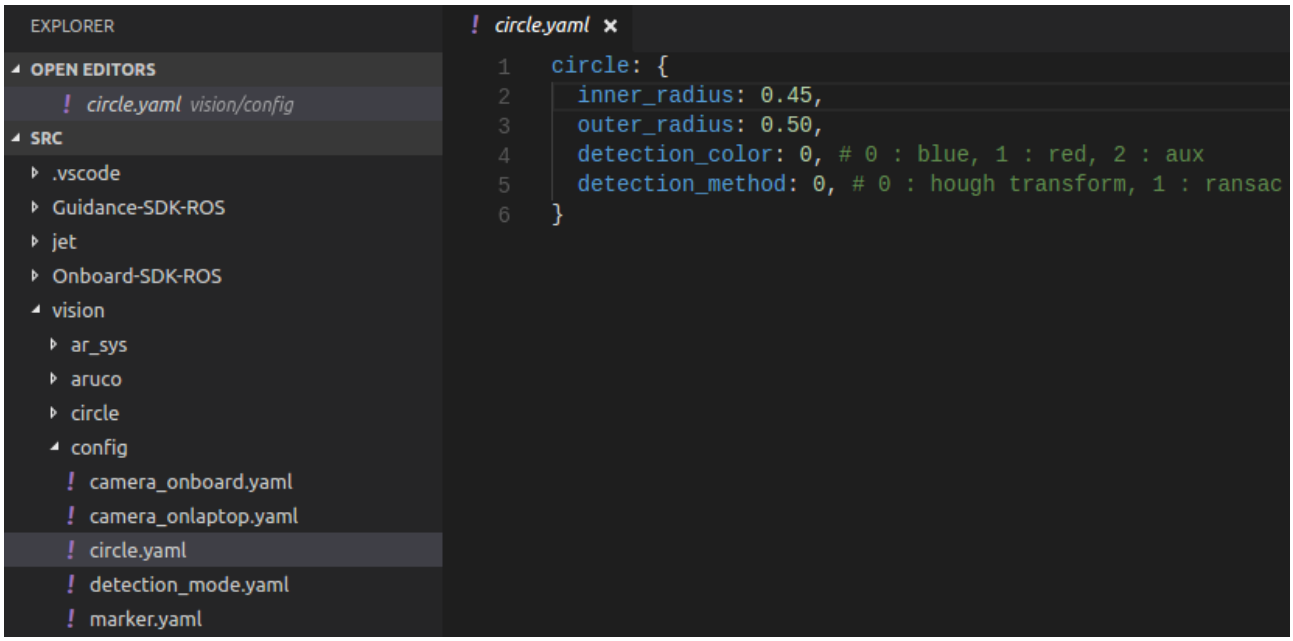

### 1.1.3 视觉部分项目结构



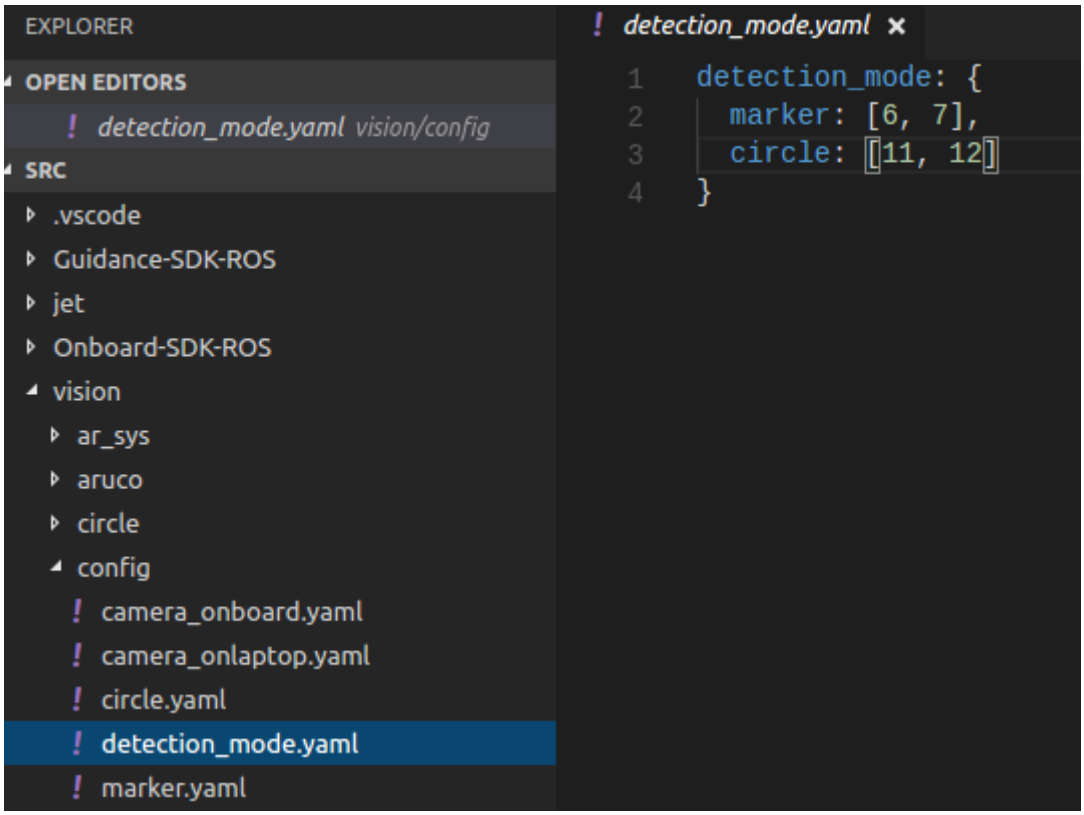

vsion/config/camera_*.yaml: 相机参数配置



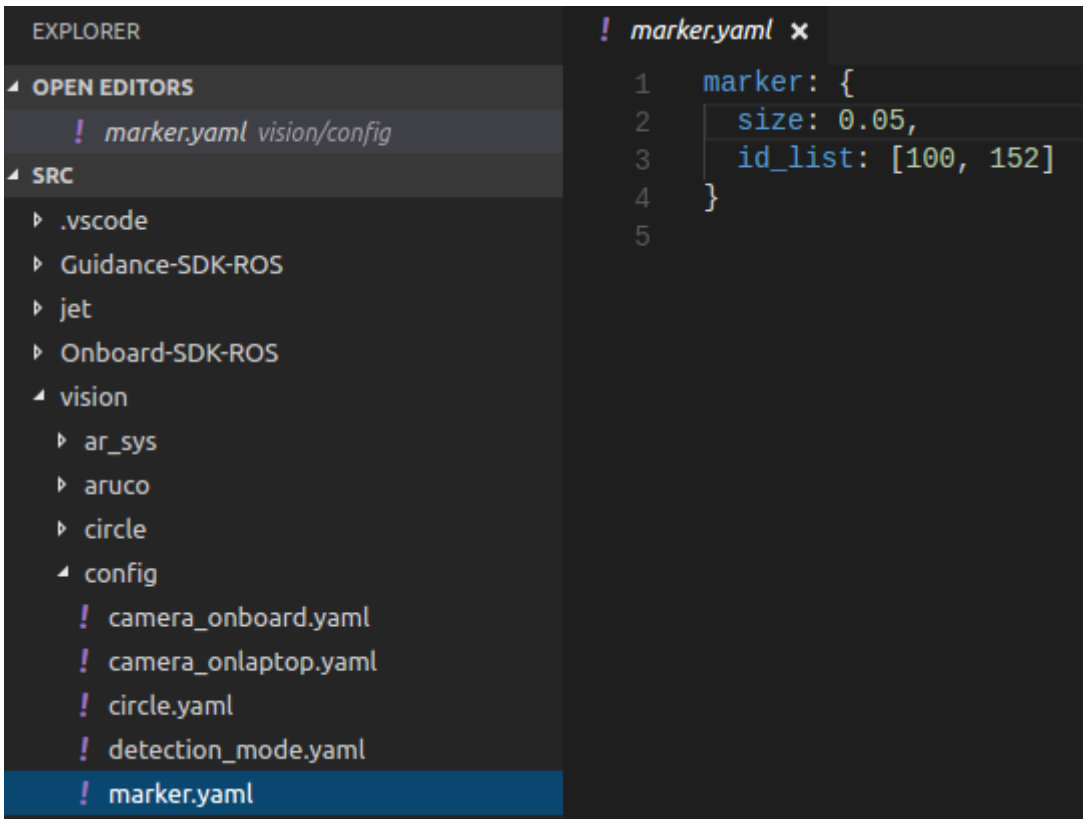
vision/config/circle.yaml: 圆（停机坪）参数配置




vision/config/detection_mode.yaml: 检测模式配置



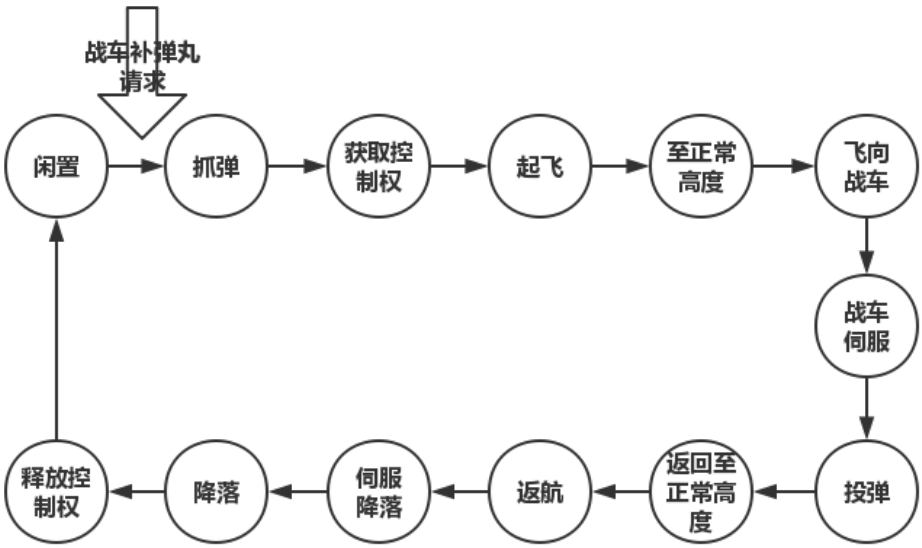

vision/config/marker.yaml: 二维码参数配置




> 以上所有配置文件均可在运行时动态加载（调用相关的 service），调参十分方便。

## 1.2 状态机

### 1.2.1 主逻辑



### 1.2.2 状态/命令类型枚举

```
typedef enum
{
  STAND_BY,
  GRAB_BULLETS,
  REQUEST_CONTROL,
  TAKE_OFF,
  TO_NORMAL_ALTITUDE,
  FLY_TO_CAR,
  FIND_CAR,
  SERVE_CAR,
  DROP_BULLETS,
  BACK_TO_NORMAL_ALTITUDE,
  FLY_BACK,
  FIND_PARK,
  VISUAL_SERVO_LANDING,
  LANDING,
  RELEASE_CONTROL,
} JetCmd_e;
```

### 1.2.3 状态机执行流程

```
void Jet::stateMachine()
{
    static bool success = false;
    static uint32_t tick = 0;
    switch (jet_state)
    {
        case STAND_BY:
        if (!success)
        {
            success = doStandby();
            std::cout << "stateMachine: " << "Standby" << std::endl;
        }
        else if (tick < duration[STAND_BY])
        {
            tick++;
            std::cout << "stateMachine: " << "Standby@Tick: " << tick <<
std::endl;
        }
        else
        {
            tick = 0;
            success = false;
            jet_state = GRAB_BULLETS;
            std::cout << "stateMachine: " << "Standby->Grab Bullets" << tick <<
std::endl;
        }
        break;

        case GRAB_BULLETS:
        if (!success)
        {
            success = doGrabBullets();
            std::cout << "stateMachine: " << "Grab Bullets" << std::endl;
```

```
        }
        else if (tick < duration[GRAB_BULLETS])
        {
            tick++;
            std::cout << "stateMachine: " << "Grab Bullets@Tick: " << tick <<
std::endl;
        }
        else
        {
            tick = 0;
            success = false;
            jet_state = REQUEST_CONTROL;
            std::cout << "stateMachine: " << "Grab Bullets->Request Control" <<
tick << std::endl;
        }
        break;

        case REQUEST_CONTROL:
        if (!success)
        {
            success = doRequestControl();
            std::cout << "stateMachine: " << "Request Control" << std::endl;
        }
        else if (tick < duration[REQUEST_CONTROL])
        {
            tick++;
            std::cout << "stateMachine: " << "Request Control@Tick: " << tick
<< std::endl;
        }
        else
        {
            tick = 0;
            success = false;
            jet_state = TAKE_OFF;
            std::cout << "stateMachine: " << "Request Control->Takeoff" << tick
<< std::endl;
        }
        break;

        case TAKE_OFF:
        if (!success)
        {
            success = doTakeoff();
            std::cout << "stateMachine: " << "Takeoff" << std::endl;
        }
        else if (tick < duration[TAKE_OFF])
        {
            tick++;
            std::cout << "stateMachine: " << "Takeoff@Tick: " << tick <<
std::endl;
        }
        else
        {
            tick = 0;
            success = false;
            jet_state = TO_NORMAL_ALTITUDE;
            std::cout << "stateMachine: " << "Takeoff->To Normal Altitude" <<
tick << std::endl;
        }
        break;

        case TO_NORMAL_ALTITUDE:
        if (!success)
```

```
        {
            success = doToNormalAltitude();
            std::cout << "stateMachine: " << "To Normal Altitude" << std::endl;
        }
        else if (tick < duration[TO_NORMAL_ALTITUDE])
        {
            tick++;
            std::cout << "stateMachine: " << "To Normal Altitude@Tick: " <<
tick << std::endl;
        }
        else
        {
            tick = 0;
            success = false;
            jet_state = FLY_TO_CAR;
            std::cout << "stateMachine: " << "To Normal Altitude->Fly to Car"
<< tick << std::endl;
        }
        break;

        case FLY_TO_CAR:
        if (!success)
        {
            success = doFlyToCar();
            std::cout << "stateMachine: " << "Fly to Car" << std::endl;
        }
        else if (tick < duration[FLY_TO_CAR])
        {
            tick++;
            std::cout << "stateMachine: " << "Fly to Car@Tick: " << tick <<
std::endl;
        }
        else
        {
            tick = 0;
            success = false;
            jet_state = FIND_CAR;
            std::cout << "stateMachine: " << "Fly to Car->Find Car" << tick <<
std::endl;
        }
        break;

        case FIND_CAR:
        if (!success)
        {
            success = doFindCar();
            std::cout << "stateMachine: " << "Find Car" << std::endl;
        }
        else if (tick < duration[FIND_CAR])
        {
            tick++;
            std::cout << "stateMachine: " << "Find Car@Tick: " << tick <<
std::endl;
        }
        else
        {
            tick = 0;
            success = false;
            jet_state = SERVE_CAR;
            std::cout << "stateMachine: " << "Find Car->Serve Car" << tick <<
std::endl;
        }
        break;
```

```
        case SERVE_CAR:
        if (!success)
        {
            success = doServeCar();
            std::cout << "stateMachine: " << "Serve Car" << std::endl;
        }
        else if (tick < duration[SERVE_CAR])
        {
            tick++;
            std::cout << "stateMachine: " << "Serve Car@Tick: " << tick <<
std::endl;
        }
        else
        {
            tick = 0;
            success = false;
            jet_state = DROP_BULLETS;
            std::cout << "stateMachine: " << "Serve Car->Drop Bullets" << tick
<< std::endl;
        }
        break;

        case DROP_BULLETS:
        if (!success)
        {
            success = doDropBullets();
            std::cout << "stateMachine: " << "Drop Bullets" << std::endl;
        }
        else if (tick < duration[DROP_BULLETS])
        {
            tick++;
            std::cout << "stateMachine: " << "Drop Bullets@Tick: " << tick <<
std::endl;
        }
        else
        {
            tick = 0;
            success = false;
            jet_state = BACK_TO_NORMAL_ALTITUDE;
            std::cout << "stateMachine: " << "Drop Bullets->Back to Normal
Altitude" << tick << std::endl;
        }
        break;

        case BACK_TO_NORMAL_ALTITUDE:
        if (!success)
        {
            success = doBackToNormalAltitude();
            std::cout << "stateMachine: " << "Back to Normal Altitude" <<
std::endl;
        }
        else if (tick < duration[BACK_TO_NORMAL_ALTITUDE])
        {
            tick++;
            std::cout << "stateMachine: " << "Back to Normal Altitude@Tick: "
<< tick << std::endl;
        }
        else
        {
            tick = 0;
            success = false;
            jet_state = FLY_BACK;
```

```
            std::cout << "stateMachine: " << "Back to Normal Altitude->Fly
Back" << tick << std::endl;
        }
        break;

        case FLY_BACK:
        if (!success)
        {
            success = doFlyBack();
            std::cout << "stateMachine: " << "Fly Back" << std::endl;
        }
        else if (tick < duration[FLY_BACK])
        {
            tick++;
            std::cout << "stateMachine: " << "Fly Back@Tick: " << tick <<
std::endl;
        }
        else
        {
            tick = 0;
            success = false;
            jet_state = FIND_PARK;
            std::cout << "stateMachine: " << "Fly Back->Find Park" << tick <<
std::endl;
        }
        break;

        case FIND_PARK:
        if (!success)
        {
            success = doFindPark();
            std::cout << "stateMachine: " << "Find Park" << std::endl;
        }
        else if (tick < duration[FIND_PARK])
        {
            tick++;
            std::cout << "stateMachine: " << "Find Park@Tick: " << tick <<
std::endl;
        }
        else
        {
            tick = 0;
            success = false;
            jet_state = VISUAL_SERVO_LANDING;
            std::cout << "stateMachine: " << "Find Park->Visual Servo Landing"
<< tick << std::endl;
        }
        break;

        case VISUAL_SERVO_LANDING:
        if (!success)
        {
            success = doVisualServoLanding();
            std::cout << "stateMachine: " << "Visual Servo Landing" <<
std::endl;
        }
        else if (tick < duration[VISUAL_SERVO_LANDING])
        {
            tick++;
            std::cout << "stateMachine: " << "Visual Servo Landing@Tick: " <<
tick << std::endl;
        }
        else
```

```
        {
            tick = 0;
            success = false;
            jet_state = LANDING;
            std::cout << "stateMachine: " << "Visual Servo Landing->Landing" <<
tick << std::endl;
        }
        break;

        case LANDING:
        if (!success)
        {
            success = doLanding();
            std::cout << "stateMachine: " << "Landing" << std::endl;
        }
        else if (tick < duration[LANDING])
        {
            tick++;
            std::cout << "stateMachine: " << "Landing@Tick: " << tick <<
std::endl;
        }
        else
        {
            tick = 0;
            success = false;
            jet_state = RELEASE_CONTROL;
            std::cout << "stateMachine: " << "Landing->Standby" << tick <<
std::endl;

            calied = false; // re-calibrate odom
        }
        break;

        case RELEASE_CONTROL:
        if (!success)
        {
            success = doReleaseControl();
            std::cout << "stateMachine: " << "Release Control" << std::endl;
        }
        else if (tick < duration[RELEASE_CONTROL])
        {
            tick++;
            std::cout << "stateMachine: " << "Release Control@Tick: " << tick
<< std::endl;
        }
        else
        {
            tick = 0;
            success = false;
            jet_state = STAND_BY;
            std::cout << "stateMachine: " << "Release Control->Standby" << tick
<< std::endl;
            if (freestyle)
            {
                freestyle = false; // clear freestyle flag
                std::cout << "+---------------------Jetbang Free Style
Done----------------------+" << std::endl;
                help();
            }
        }
        break;

        default:
```

```
        jet_state = STAND_BY;
        break;
    }
}
```

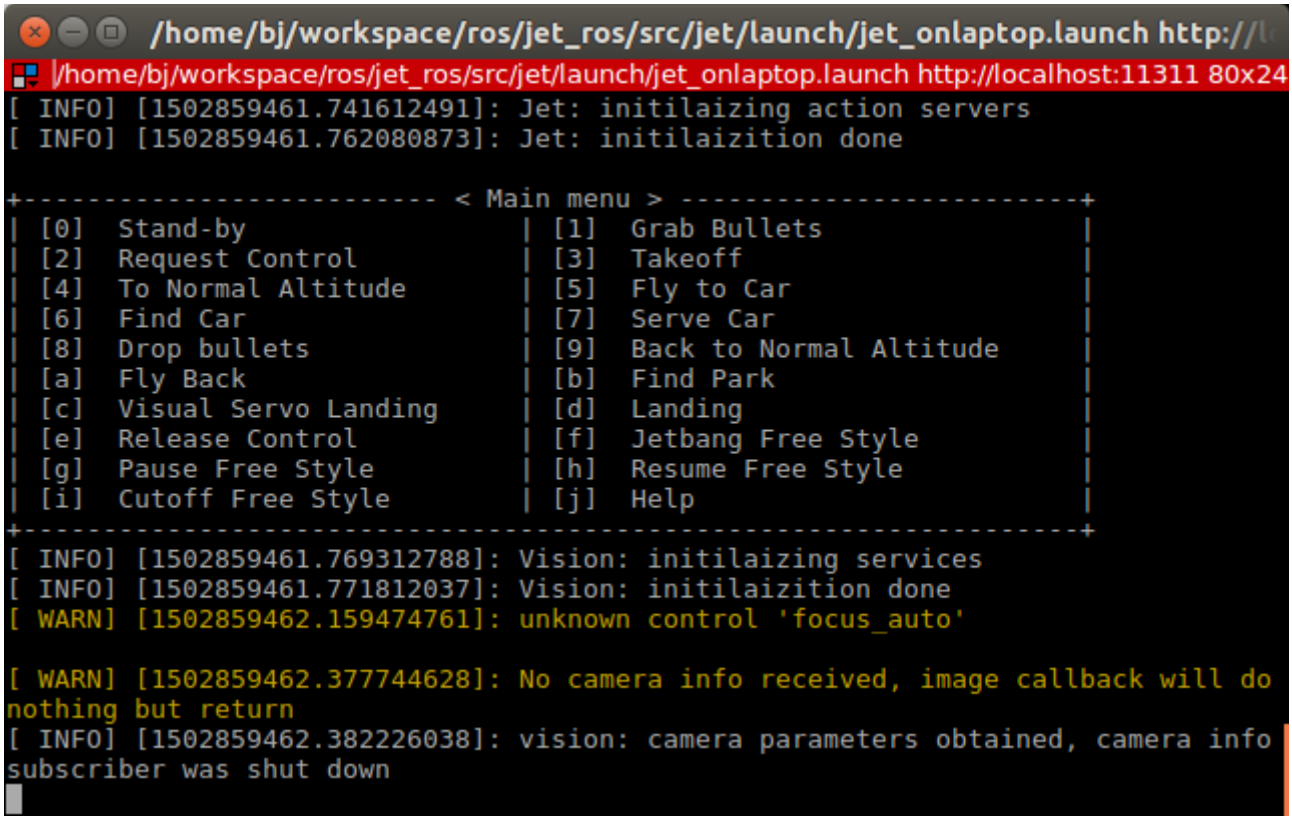
## 1.4 启动

### 1.4.1 仿真

1）首先准备两台电脑，一台 windows，一台 ubuntu，windows 用来跑 DJI Simulator，ubuntu 跑控制程序；

2）用 mini-USB 线连接 windows 电脑和飞控，用 USB 转串口线连接 ubuntu 电脑和飞机 UART-CAN2，windows 电脑上打开 DJI Assistant，启用 API 控制，打开模拟器；下面在 ubuntu 电脑上操作：

3）cd 到 jet_ros 文件夹，在此文件夹下打开两个 terminal，分别 source devel/setup.bash 之后，在一个 terminal 运行 roslaunch dji_sdk sdk_linux.launch，在另一个 terminal 运行 roslaunch jet jet_onlaptop.launch；在运行后者的窗口出现以下界面，可进行单元测试，也可运行全自动测试（Jetbang Free Style)，如想从某一项目开始测试，可先进入该项目后选择[h] Resume Free Style. 自动运行过程中可中断（[g] Pause Free Style），也可停止（[i] Cutoff Free Style）。

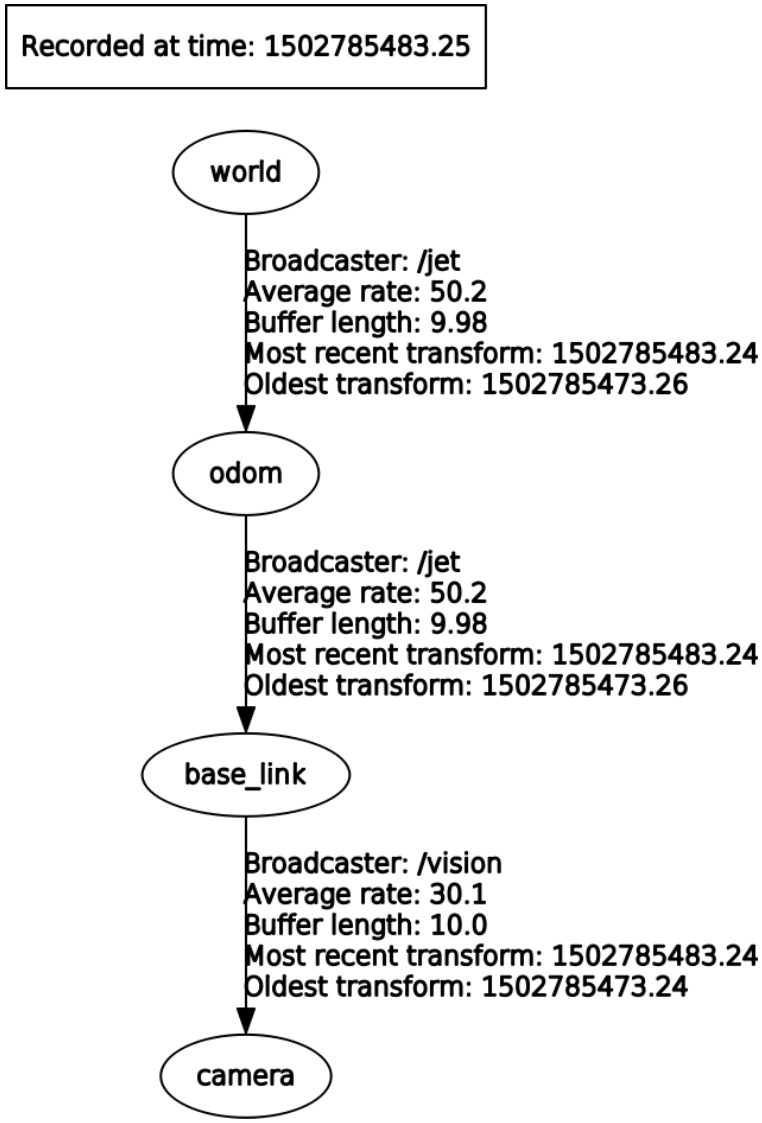


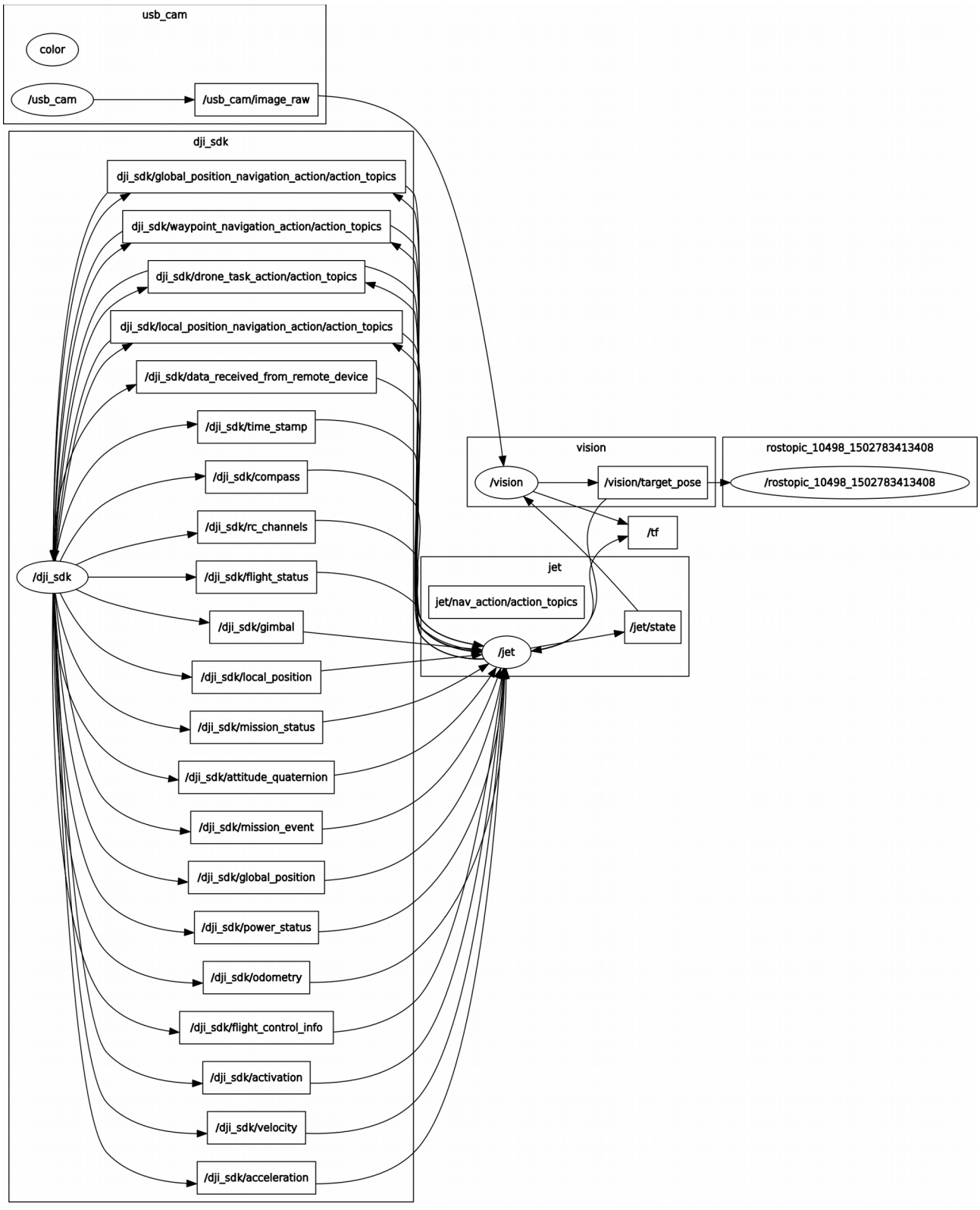
### 1.4.2 真机运行

launch 相应的启动文件即可（roslaunch jet jet_onboard.launch)

PS: 仿真环境下里程计用的是飞控的模拟数据，坐标为北东天，程序里面的飞行控制也用的是北东天，而 guidance 为北东地，实际飞行的时候 jet_onboard.launch 里面的 use_guidance 属性一定要为 true，否则坐标系不对，会炸机！！！

### 1.4.3 节点关系图与 tf 树






## 1.4.4 Topics 与 Services

rostopic list:

/jet/nav_action/cancel

/jet/nav_action/feedback

/jet/nav_action/goal

/jet/nav_action/result

/jet/nav_action/status

/jet/pose_calied

/jet/state

/tf

/vision/detection_mode

/vision/result

/vision/target_pose

rosservice list:

/jet/charge

/jet/grabber/cmd

/jet/grabber/stat

/jet/reload_dropoint_param

/jet/reload_duration_param

/jet/reload_flight_param

/jet/reload_pid_param

/usb_cam/set_camera_info

/usb_cam/start_capture

/usb_cam/stop_capture

/vision/reload_camera_param

/vision/reload_circle_param

/vision/reload_detmod_param

/vision/reload_marker_param