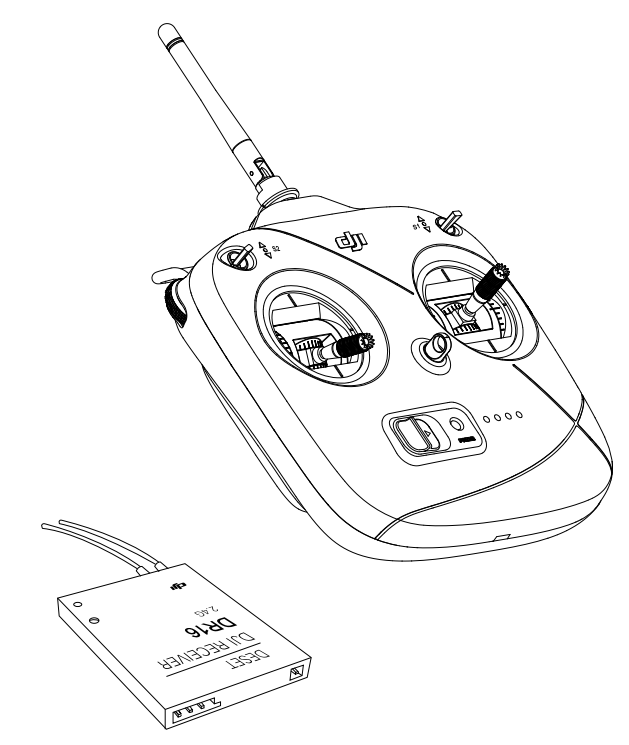
**胡文轩 2021280479 第七组**

**首先我们的目标是要能实现通过视觉的自稳功能，以及目标追踪的功能**

# 暴力方案（仅尝试）

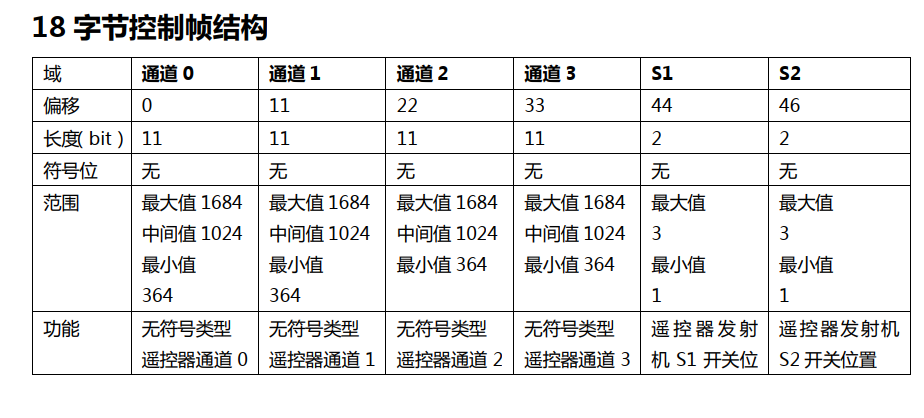
本次实验使用的飞机使用的遥控器与接收机分别为大疆的DT7 & DR16 2.4 GHz 遥控接收系统。DT7 遥控器是一款工作于 2.4 GHz 频段的无线电通信设备，该遥控器仅能与 DR16 接收机配合使用。DR16 接收机是一款工作频率为 2.4 GHz 的 16 通道接收机，可配合 DT7 遥控器使用。



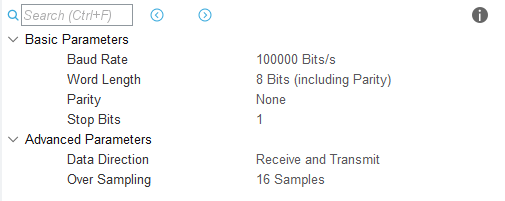
一般来说，DR16接收机输出的信号为标准的DBUS协议数据,当遥控器与接收机建立连接

后，接收机每隔14ms通过DBUS发送一帧18字节数据.在不刷特殊固件的情况下，因此使用的是DBUS协议。

DBUS协议是DJI在SBUS协议上进行二次开发的协议，将原来部分通道值放入了键盘鼠标数据，具体协议如下图所示，，因此我们需要完成的第一件事是解析DBUS数据



接下来我们配置单片机以控制飞机，使用stm32单片机模拟SBUS输入输出来控制飞机，



根据DBUS协议，我们将串口配置为下列配置



随后我们书写DBUS协议处理代码

这里为了提高输出SBUS协议的实时性，我们使用DMA来完成遥控器信号的发送以及直接转换。STM32系列中的DMA（直接存储器访问）是一种重要的外设，用于在系统内部的不同模块之间实现数据传输，而无需CPU的直接干预。DMA允许数据在外设和内存之间直接传输，提高了系统性能和效率。使用DMA可以提高系统性能，减轻CPU负担，使得系统更加高效。下图为单片机上的DMA配置，将USART2（即接收机的输出端引脚）接收到的数据分配给DMA1的Stream 5.

图表

描述已自动生成

SBUS协议：全称serial-bus，是一种[串口通信协议](https://so.csdn.net/so/search?q=%E4%B8%B2%E5%8F%A3%E9%80%9A%E4%BF%A1%E5%8D%8F%E8%AE%AE&spm=1001.2101.3001.7020)，仅用一根信号线就能传输多达16通道的数据，比多路PWM捕获高效且省资源。最初由Futaba（双叶遥控器制造商）开发用于无线遥控器和模型飞行器之间的通信。SBUS协议以串行数据的形式传输多个通道的遥控信号，提供了高分辨率、低延迟和较低的噪声敏感性。SBUS为PX4飞控提供遥控器输出的各个通道的数值，因此我们通过模拟通道值输出来控制飞机的姿态与速度。

为了将USART协议转换为SBUS协议，必须加入硬件反相器，因为SBUS的信号是采用的负逻辑，也就是电平相反，只能在硬件上进行反相，在软件上无法操作，因此需要优先完成以下反相器电路

图示, 示意图

描述已自动生成

为此我们使用洞洞板与SS8050搭建了一个简易的反相器，用于对STM32单片机输出的USART的TX端的信号进行反向，由于串口协议存在开始位、停止位以及奇偶校验位，因此软件反向仅仅只能反转数据段的电平信号，因此需要硬件的反相器。我们本次实验中实际搭建的硬件反相器如下图所示：

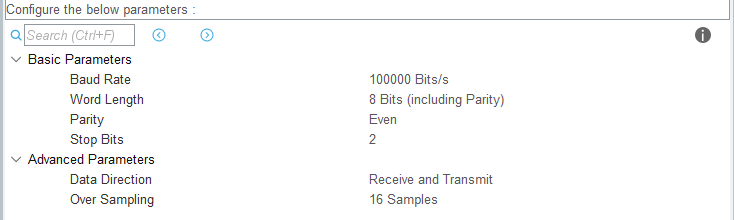


随后我们配置单片机以及书写单片机的代码，本次实验使用到的单片机由RobotPilots实验室设计以及制造，具有多路串口，接收DBUS，并且可以使用3S作为电源供电，配合我们之前完成的硬件反相器，符合本次实验所需要的硬件需求协议如下图所示：解析完SBUS协议之后再输出SBUS协议便可使用单片机直接控制飞机。

卡通人物

低可信度描述已自动生成

接下来我们配置单片机以控制飞机，使用stm32单片机模拟SBUS输入输出来控制飞机，



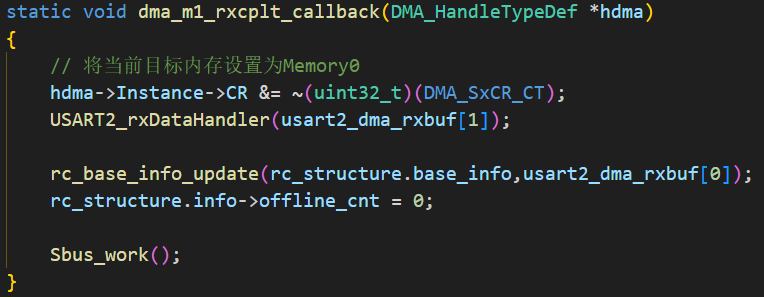
将串口参数配置为如下：100k波特率（100 000 bits/s），8位数据位，2位停止位，偶校验（EVEN)，无控流，25个字节。

为了输出SBUS协议，我们需要了解其协议格式：数据共25个字节

* 开始字节：startbyte=0x0f
* 通道数据字节：[data1][data2]…[data22] data1~22: LSB（低位在前），对应16个通道（ch1-ch16），每个通道11bit（22 × 8=16 × 11)；
* 标志位字节：[flags] flag位标志遥控器的通讯状态，可以通过查询flag位来采取失控保护(例如乐迪R9Ds的flag位为0x00时标志存在连接)。
* 结束字节：endbyte=0x00

由于本身的接收机使用的SBUS的低速模式（14ms发送一次数据），因此在单片机端我们的控制频率需要控制在14ms以下，这样才能保证有一个稳定的控制效果，随后我们书写控制部分的算法，为了保证安全，除了遥控器摇杆处于特殊模式时（即拨杆S1与拨杆S2都处于最上方），才会进行模拟SBUS信号的修改输出，在其他情况下我们都使用DMA将信号直接搬运。

此处我们放在DMA1（接收机接收中断）的中断中进行，即只有当接收到数据的时候，才会发送数据，才会进行数据解析，起到一个保护作用，保证不会出现飞机无法控制的情况，保证遥控器的直接输出。



完成了模拟SBUS输出后，我们接下来进入到我们的控制部分，我们设想将STM32单片机作为一个信息交换的枢纽，单片机一方面接收接收机信号，输出SBUS协议控制飞控，另一方面需要完成与边缘计算设备通讯，作为边缘计算设备的一个API。

文本

描述已自动生成

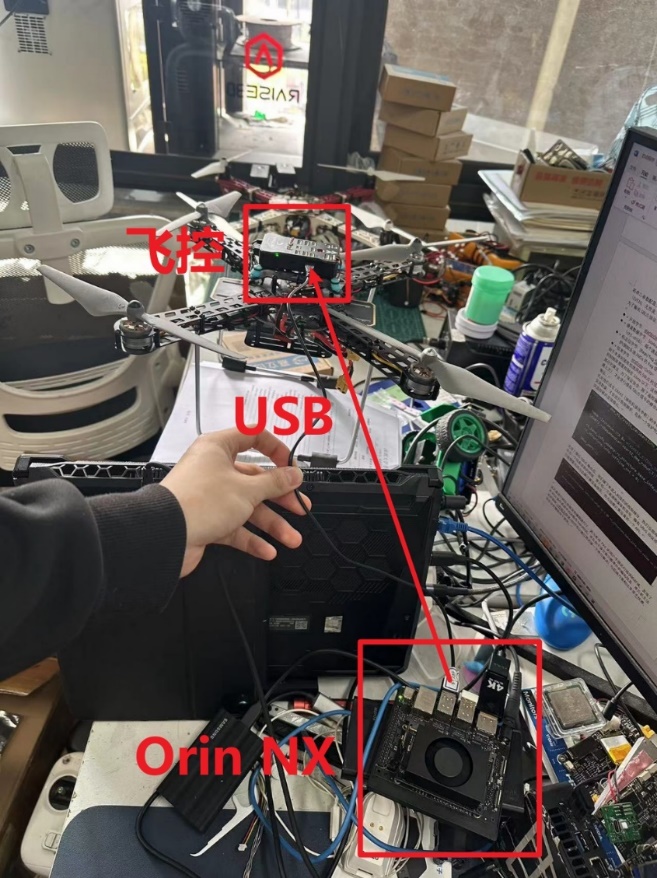
该方案进行到此处就没有继续进行了，因为在对PX4的用户手册进行阅读的时候，发现了在ROS操作系统中的功能包MAVROS。并且由于该方案只能单向的输出信息，单片机无法实时获取飞控相关信息（也许可以通过再另接一个串口解决），与后续方案相比非常不优雅，因此没有继续向下深究。

# ROS开发（最终实现）

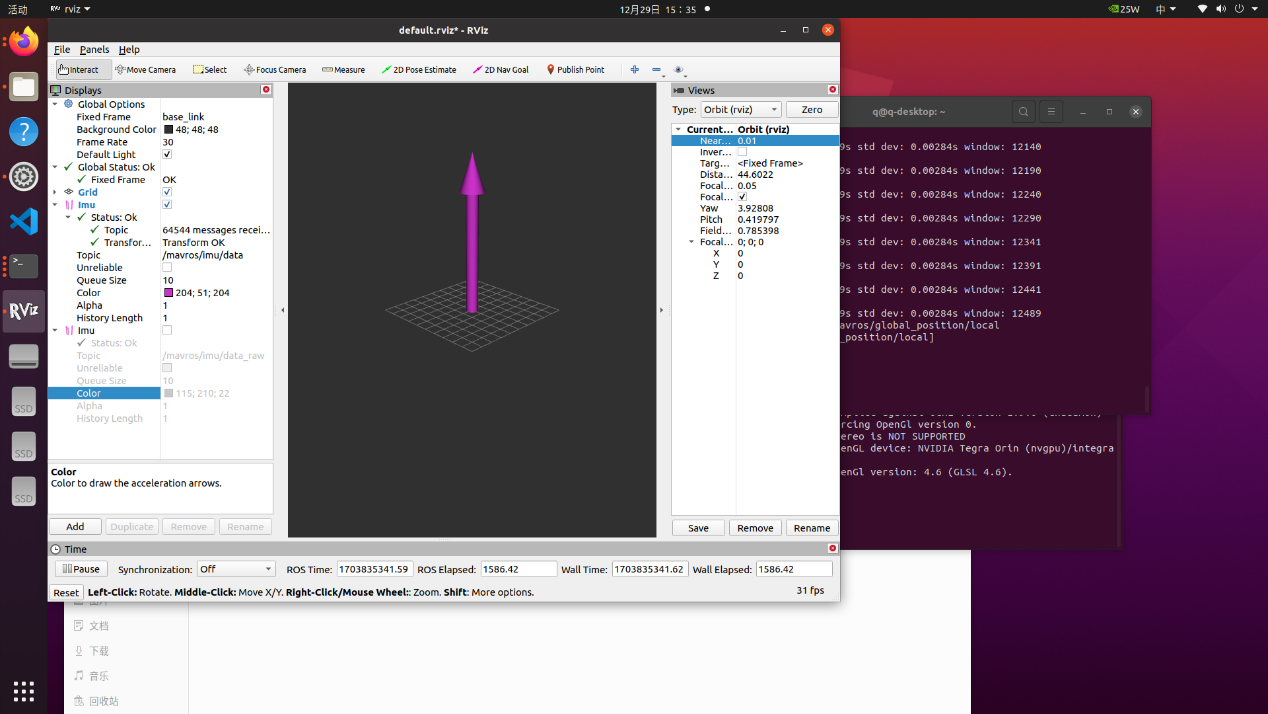
在研读PX4的开发手册时发现，我们可以通过ROS发布当前的位姿，此时MAVROS会订阅该数据，并且使用此数据作为位姿来进行自稳，通过该方式便可以实现视觉的自稳

MAVROS是一个ROS（Robot Operating System）的软件包，用于与MAVLink协议兼容的无人机系统进行通信。MAVLink是一种轻量级的通信协议，广泛用于飞行控制系统和地面站之间的通信。MAVROS允许ROS系统与MAVLink兼容的自动驾驶仪、飞行控制器等硬件进行连接和交互，为ROS中的机器人控制提供了强大的工具。因此我们能直接通过mavlink协议使得边缘计算设备与px4飞控连接。。

此处为了后续更高的帧率，我们此处在边缘设备选择上选择了性能更高的Jetson Orin NX 16GB 模组。同时Orin NX的输入电压是9V~35V，满足我们无人机上的3s电池输出。我们由于 Jetson Orin NX 的 JetPack 5.1.2 对应的是 Ubuntu 20.04，因此我们选择安装 noetic 版本的 ROS 以及其对应的 mavros 功能包。环境配置过程就省略了（虽然在此处是本次实验花费时间最多的地方）最终我们实现了ROS与PX4飞控的连接，我们可以通过订阅/mavros下面的话题来获取飞控相关的姿态信息，这一点是相对于初始方案的优点，通过该方式连接，在Orin NX上我们不仅能获取当前姿态，也能通过MavLink控制无人机。



由于 Jetson Orin NX 与飞控（飞行控制器）一般有两种通信连接方式，一种是通过 USB 连接，另一种是通过串口连接，本次实验主要使用USB方式使得 Jetson Orin NX 通过 Mavros 实现与飞控的通信。由于本次课程提供的飞控为Pixhawk 2.4.8 经过查询官方文档，发现时可以支持该版本使用的，因此本次实验使用USB的连接方式。



图中为MAVROS与PX4连接效果，其中我们订阅了来自px4飞控的IMU数据，可以看到其在rviz中的可视化效果（同一文件夹中有视频）

同时我们可以发布/mavros/setpoint\_velocity/cmd\_vel\_unstamped (geometry\_msgs/Twist)话题，用于设置无人机的目标速度；发布/mavros/setpoint\_attitude/attitude (geometry\_msgs/PoseStamped)话题，用于设置无人机目标位姿。

ofboard模式是PX4飞控系统中的一种工作模式，允许外部设备通过MAVLink协议与飞控进行通信控制。在ofboard模式下，飞控将不再遭循预设的飞行计划，而是接受外部指令进行控制。因此我们可以将飞行模式设置为Offboard模式，从而使得飞机可以遵循Mavlink发布的指令：

电脑萤幕的截图

描述已自动生成

实际的飞行模式设置如上图所示，将控制是否进入Offboard模式放在遥控器的好处是可以随时切换为姿态模式，调试过程中遇到问题的时候可以及时避免更大的意外。同时为了避免意外的发生，我还设置了Emergency Kill switch，当意外发生的时候能迅速关闭电机动力输出（例如卡在网上的时候，事实证明非常有用）

同时我们需要配置一下px4参数树，这样PX4飞控才能获取并使用我们发布的位姿数据。我们使用QGC修改参数树

参数配置如下

MAV\_1\_CONFIG= TELEM 2

MAV\_1\_MODE = Onboard

EKF2\_AID\_MASK =24

EKF2\_HGT\_MODE 设置为 Vision

MAV\_ODOM\_LP设置为1

这段配置的意义是为了让PX4飞控能够接收并使用我们在Orin NX上发布的姿态信息，并且依据此信息进行定位与自稳定。并且通过此方式来配置的好处就是，姿态是在飞控底层进行融合的，而非通过外部再增加控制器的方式来对飞机进行控制，后者不仅需要长时间的调参，还忽略了姿态与运动状态在物理层面上的耦合关系，相比之下前者能达到更好的姿态控制效果。

设置完成之后我们在Orin NX继续开发，在视觉定位方面，我们使用Intel的realsense系列产品，分别是D435i与T265。选择D435i的原因是希望它既能够充当一个获取深度图的设备，又能作为获取图像的设备，这样可以在Orin NX跑VINS来实时获取当前的位置信息，对于图像又可以使用YOLOv5模型来获取需要跟踪的目标信息。而使用T265是因为在算力不够的情况下，其自带的计算芯片能提高帧数，同时获取的姿态更加稳定，也能节省一些开发时间，省去配置环境的麻烦。

T265 是一款跟踪相机，它可以利用惯性测量单元（IMU）和视觉里程计（VIO）技术，实现高精度的六自由度（6DoF）位姿估计1。它还可以实现重定位功能，即在丢失跟踪后能够重新识别之前访问过的场景2。T265 的内部有一个 Intel® Movidius™ Myriad™ 2 视觉处理单元（VPU），它可以将 IMU 数据和两个鱼眼摄像头的视频数据融合起来，从而减轻主机的计算负担1。T265 的两个鱼眼摄像头的分辨率为 848X800，帧率为 30FPS2。

D435i 是一款深度相机，它可以提供高质量的深度图像和彩色图像3。它和 D435 的区别在于，可以为一些需要相机移动的应用提供更好的深度感知，例如 SLAM、跟踪、点云对齐等3。D435i 的 IMU 传感器和 T265 的一样，都是 Bosch BMI055 6 轴惯性传感器1。D435i 的深度传感器的分辨率为 1280 × 720，帧率为 90FPS4。

接下来就是配置VINS与D435i实现视觉里程计的过程（也是非常折磨人）

配置完realsense-ros的驱动后，编译vins-mono的源码，最严峻的问题是由于vins的源码使用的是opencv3进行编写的，而我安装的opencv4.6.0属于opencv4，因此需要更改源代码的很大一部分地方。除此之外，因为ros自带的cv\_brige与vins所需的cv\_brige的版本不同，也需要重新配置cv\_brgie的版本。实验进行到这里其实也没有花费太长时间，在实际运行vins-mono的时候发现实际帧数特别低，并不能保证作为姿态传感器的实时性。随后仔细分析问题，发现是由于在安装opencv的时候，没有针对安装CUDA版本的，导致无法发挥Orin NX的超高性能。

随后使用T265+USB相机的方案继续进行。T265在运行其驱动后便可直接获取其姿态信息，因此我们只需要将位姿信息转换到无人机上便可以实现我们的最终目的。在获取了当前姿态后，我们需要做的就是将T265的姿态转换为PX4飞控所需的姿态，同时发布到PX4订阅的节点下，这样就不需要我们重新部署控制器，也不需要过多的调参

一开始我们打算自己写t265转换到mavros的坐标转换，因为其实只是简单的tf变换，后来在学习的过程中借鉴了部分网上开源的代码。

到这里已经实现了视觉的自稳，具体效果可以查看压缩包中提交的视频。

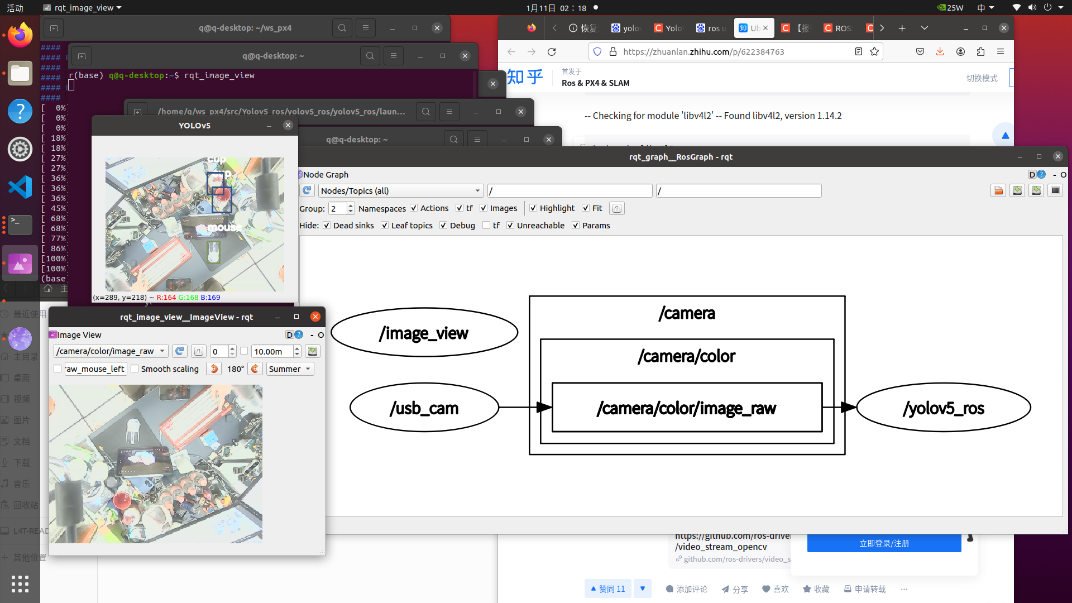
图片包含 室内, 桌子, 电脑, 游戏机

描述已自动生成

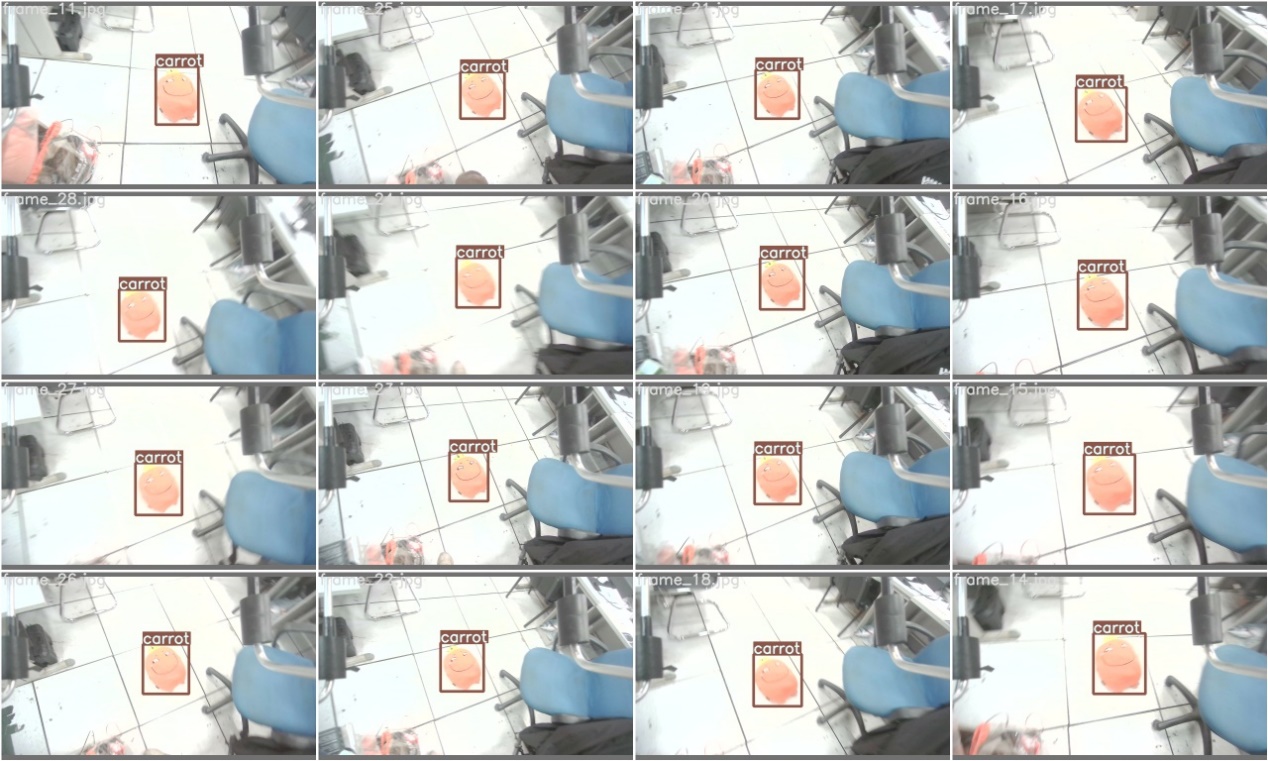
USB相机使用

为了实现目标跟踪的功能我们在飞机上增加了USB相机用于实时识别目标，并且我们部署了Yolov5模型实现目标检测的功能

为了实现目标的实时监测，我们使用了先前使用过的yolov5模型，不同的是使用了yolov5对应的ros功能包来实现目标检测与输出（此处参考教程<https://blog.csdn.net/Chris121345/article/details/122563536>）使用到了yolov5\_ros功能包，该功能包能够直接实现订阅USB相机的图像，并且使用我们预训练好的.pt文件来直接识别目标。其中实现了类似视频流的功能，因此基于ROS的节点订阅与发布，还依赖了USB\_CAM这个功能包，该功能包能直接将 /dev/video0的视频流数据发布到ROS话题中，具体效果如下图所示：

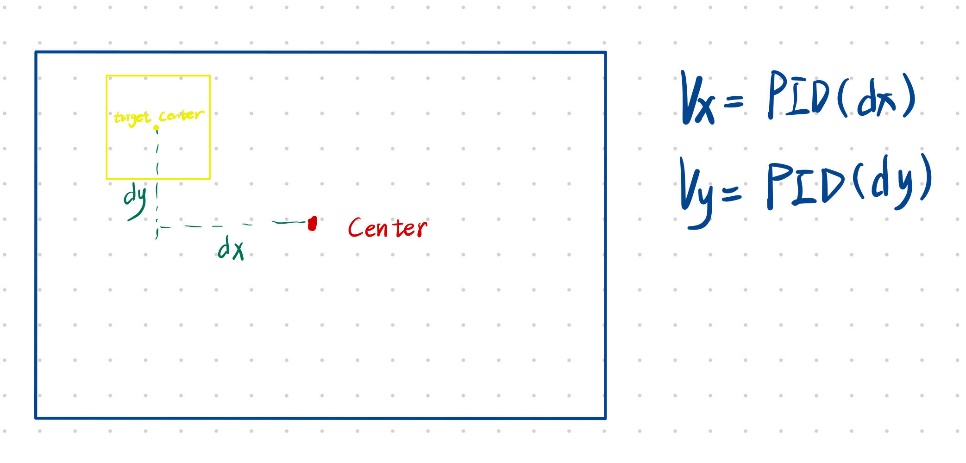


YOLOv5训练的过程利用了本学期实验十一中学习到的技能，使用labelimg软件来进行图像标注，最终实现的效果如图，此处由于USB相机拍摄不便，我们使用AMcap软件作为USB相机的驱动，以avi格式将图像保存下来，后续使用我们自己书写的python程序将avi图像中的每五帧保存成jpg图片以方便我们进行后续训练。

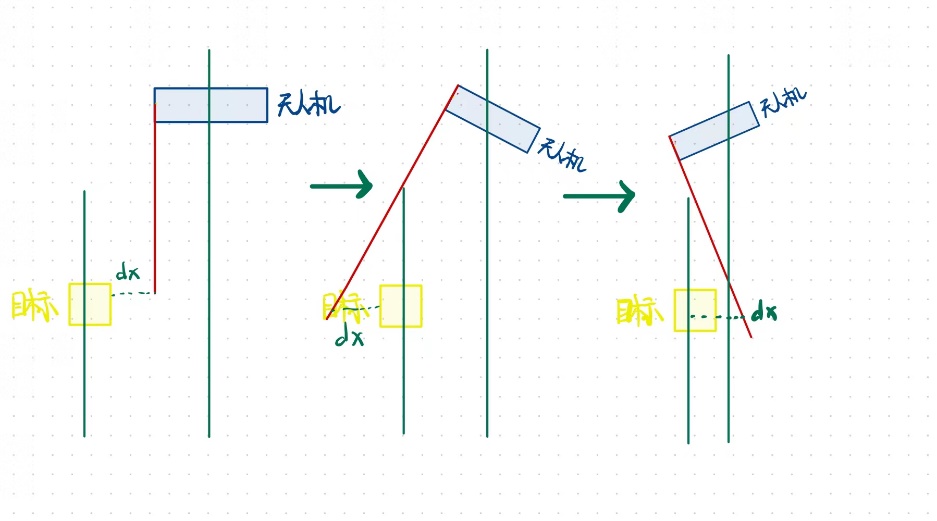


其中USB相机用于实现车辆跟随的功能，由于我们的安装方式缺少云台，因此不可避免的遇到观测与控制耦合的情况，即为了实现某个运动姿态会改变相机的观测姿态，这会最终结果的震荡以及超调。因此我们对这个问题进行了理论的分析，最终通过对自身姿态的解算，实现观测与控制的解耦。

理想状态控制框图如下图，高度使用先前提到的内容实现自稳，控制无人机的x与y方向的速度实现维持yolov5输出的坐标在摄像头的中心

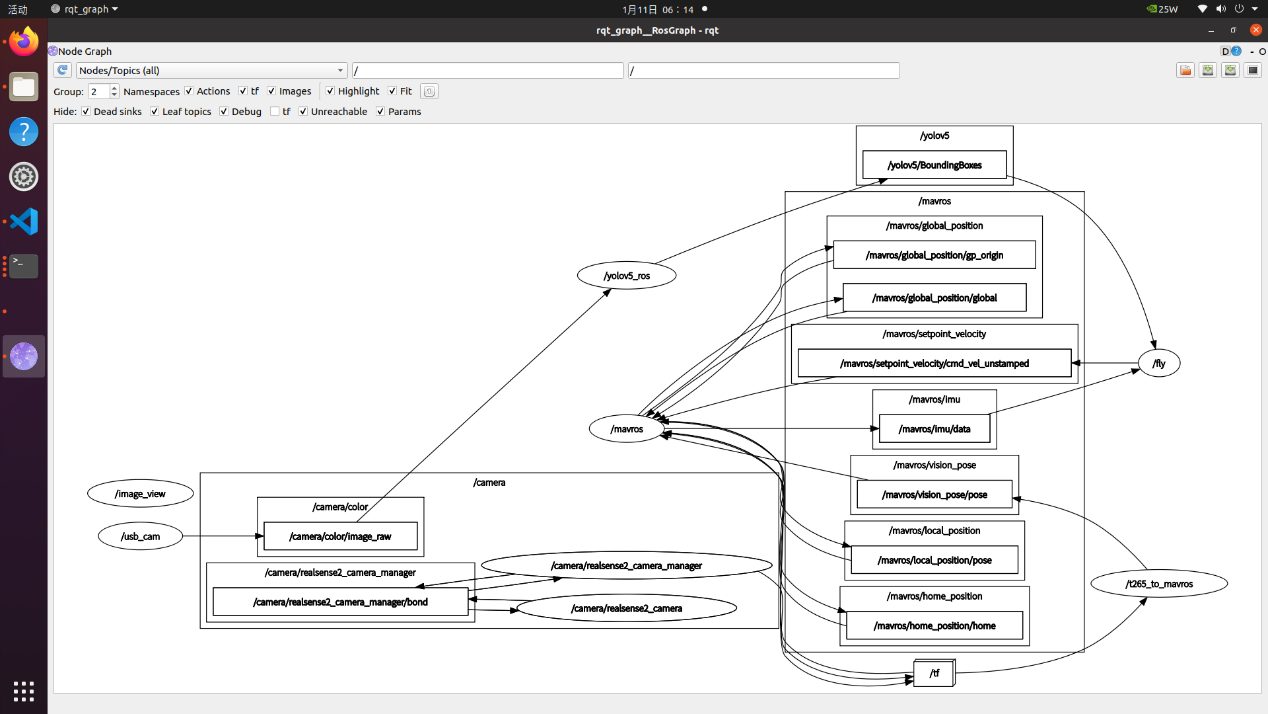


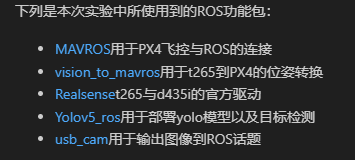
如下图所示，当追求跟踪的速度时，无人机的姿态可能会产生震荡，当然我们可以在控制中假如约束条件，即保证无人机的仰角和俯角不超过某个值，这样也能也能抵消部分震荡，但无法从根源解决这种问题，为此计算出实际的dx我们需要计算出目标的三维坐标以及自身的实际坐标，为此可能使用D435i是非常完美且正确的选择，因为配合D435i的深度图，可以获取下图中红线的距离（深度相机检测到距离物体的距离），但是本次实验使用的USB相机可能无法实现该功能，



在实际的使用，由于测距不稳定，以及紧张的期末周复习进度，因此最终我们并没有使用这套解耦过程。而是将飞行过程中的姿态控制在pitch轴与roll轴不超过5°，同时限制无人机的速度，避免失控。

最终节点示意图如下图所示：





遇到的问题：

* Orin NX每次上电，需要重新插拔USB设备才能识别到USB设备，也就是说每次重新上电都需要插拔一次T265，PX4飞控以及USB相机的线，这样才能正常使用；该问题直到报告提交时仍未解决，但不影响最终效果

# Appendix

## DBUS解码代码

#include "remote.h"

#include <string.h>

#include <stdlib.h>

#include "drv\_tim.h"

extern rc\_t rc\_structure;

/\*\*

\* @brief 按键长按时间设置

\*/

void key\_board\_cnt\_max\_set(rc\_base\_info\_t \*info)

{

info->mouse\_btn\_l.cnt\_max = MOUSE\_BTN\_L\_CNT\_MAX;

info->mouse\_btn\_r.cnt\_max = MOUSE\_BTN\_R\_CNT\_MAX;

info->Q.cnt\_max = KEY\_Q\_CNT\_MAX;

info->W.cnt\_max = KEY\_W\_CNT\_MAX;

info->E.cnt\_max = KEY\_E\_CNT\_MAX;

info->R.cnt\_max = KEY\_R\_CNT\_MAX;

info->A.cnt\_max = KEY\_A\_CNT\_MAX;

info->S.cnt\_max = KEY\_S\_CNT\_MAX;

info->D.cnt\_max = KEY\_D\_CNT\_MAX;

info->F.cnt\_max = KEY\_F\_CNT\_MAX;

info->G.cnt\_max = KEY\_G\_CNT\_MAX;

info->Z.cnt\_max = KEY\_Z\_CNT\_MAX;

info->X.cnt\_max = KEY\_X\_CNT\_MAX;

info->C.cnt\_max = KEY\_C\_CNT\_MAX;

info->V.cnt\_max = KEY\_V\_CNT\_MAX;

info->B.cnt\_max = KEY\_B\_CNT\_MAX;

info->Shift.cnt\_max = KEY\_SHIFT\_CNT\_MAX;

info->Ctrl.cnt\_max = KEY\_CTRL\_CNT\_MAX;

}

/\*\*

\* @brief 遥控信息更新

\*/

void rc\_base\_info\_update(rc\_base\_info\_t \*info, uint8\_t \*rxBuf)

{

info->ch0 = (rxBuf[0] | rxBuf[1] << 8 ) & 0x07FF;

info->ch0 -= 1024;

info->ch1 = (rxBuf[1] >> 3 | rxBuf[2] << 5 ) & 0x07FF;

info->ch1 -= 1024;

info->ch2 = (rxBuf[2] >> 6 | rxBuf[3] << 2 | rxBuf[4] << 10) & 0x07FF;

info->ch2 -= 1024;

info->ch3 = (rxBuf[4] >> 1 | rxBuf[5] << 7 ) & 0x07FF;

info->ch3 -= 1024;

info->s1.value = ((rxBuf[5] >> 4) & 0x000C) >> 2;

info->s2.value = ( rxBuf[5] >> 4) & 0x0003;

info->mouse\_vx = rxBuf[6] | (rxBuf[7 ] << 8);

info->mouse\_vy = rxBuf[8] | (rxBuf[9 ] << 8);

info->mouse\_vz = rxBuf[10] | (rxBuf[11] << 8);

info->mouse\_btn\_l.value = rxBuf[12] & 0x01;

info->mouse\_btn\_r.value = rxBuf[13] & 0x01;

info->W.value = rxBuf[14] & 0x01;

info->S.value = ( rxBuf[14] >> 1 ) & 0x01;

info->A.value = ( rxBuf[14] >> 2 ) & 0x01;

info->D.value = ( rxBuf[14] >> 3 ) & 0x01;

info->Shift.value = ( rxBuf[14] >> 4 ) & 0x01;

info->Ctrl.value = ( rxBuf[14] >> 5 ) & 0x01;

info->Q.value = ( rxBuf[14] >> 6 ) & 0x01 ;

info->E.value = ( rxBuf[14] >> 7 ) & 0x01 ;

info->R.value = ( rxBuf[15] >> 0 ) & 0x01 ;

info->F.value = ( rxBuf[15] >> 1 ) & 0x01 ;

info->G.value = ( rxBuf[15] >> 2 ) & 0x01 ;

info->Z.value = ( rxBuf[15] >> 3 ) & 0x01 ;

info->X.value = ( rxBuf[15] >> 4 ) & 0x01 ;

info->C.value = ( rxBuf[15] >> 5 ) & 0x01 ;

info->V.value = ( rxBuf[15] >> 6 ) & 0x01 ;

info->B.value = ( rxBuf[15] >> 7 ) & 0x01 ;

info->thumbwheel.value = ((int16\_t)rxBuf[16] | ((int16\_t)rxBuf[17] << 8)) & 0x07ff;

info->thumbwheel.value -= 1024;

if(rc\_structure.info->offline\_cnt > REMOTE\_OFFLINE\_CNT\_MAX)

{

rc\_structure.info->status = REMOTE\_OFFLINE;

}

else

{

rc\_structure.info->status = REMOTE\_ONLINE;

}

}

void rc\_init(rc\_t \*rc, rc\_info\_t \*info, rc\_base\_info\_t \*base\_info)

{

rc->base\_info = base\_info;

rc->info = info;

/\* 基本信息置零 \*/

memset(base\_info,0,sizeof(rc\_base\_info\_t));

/\* 拨杆旋钮状态初始化 \*/

base\_info->s1.status = keep\_R;

base\_info->s2.status = keep\_R;

base\_info->thumbwheel.status = keep\_R;

/\* 设置失联计数 \*/

info->offline\_cnt = REMOTE\_OFFLINE\_CNT\_MAX;

}

void rc\_switch\_status\_interrupt\_update(rc\_base\_info\_t \*info)

{

/\* 左拨杆判断 \*/

if(info->s1.value != info->s1.value\_last)

{

switch(info->s1.value)

{

case 1:

info->s1.status = up\_R;

break;

case 3:

info->s1.status = mid\_R;

break;

case 2:

info->s1.status = down\_R;

break;

default:

break;

}

info->s1.value\_last = info->s1.value;

}

else

{

info->s1.status = keep\_R;

}

/\* 右拨杆判断 \*/

if(info->s2.value != info->s2.value\_last)

{

switch(info->s2.value)

{

case 1:

info->s2.status = up\_R;

break;

case 3:

info->s2.status = mid\_R;

break;

case 2:

info->s2.status = down\_R;

break;

default:

break;

}

info->s2.value\_last = info->s2.value;

}

else

{

info->s2.status = keep\_R;

}

}

/\*\*

\* @brief 遥控器旋钮状态跳变判断并更新

\*/

void rc\_wheel\_status\_interrupt\_update(rc\_base\_info\_t \*info)

{

if(abs(info->thumbwheel.value\_last) < WHEEL\_JUMP\_VALUE)

{

if(info->thumbwheel.value > WHEEL\_JUMP\_VALUE)

{

info->thumbwheel.status = up\_R;

}

else if(info->thumbwheel.value < -WHEEL\_JUMP\_VALUE)

{

info->thumbwheel.status = down\_R;

}

else

{

info->thumbwheel.status = keep\_R;

}

}

else

{

info->thumbwheel.status = keep\_R;

}

info->thumbwheel.value\_last = info->thumbwheel.value;

}

//键盘控制

/\*\*

\* @brief 键盘按键状态判断并更新

\*/

void key\_board\_status\_update(key\_board\_info\_t \*key)

{

switch(key->status)

{

case down\_K:

key->status = short\_press\_K;

key->cnt++;

break;

case up\_K:

key->status = relax\_K;

key->cnt = 0;

break;

case short\_press\_K:

key->cnt++;

if(key->cnt >= key->cnt\_max)

{

key->value = long\_press\_K;

key->cnt = key->cnt\_max;

}

break;

default:

break;

}

}

/\*\*

\* @brief 遥控器接收产生中断时键盘按键状态判断并更新

\*/

void key\_board\_status\_interrupt\_update(key\_board\_info\_t \*key)

{

switch(key->status)

{

case relax\_K:

if(key->value == 1)

{

key->status = down\_K;

key->cnt = 0;

}

break;

case short\_press\_K:

if(key->value == 0)

{

key->status = up\_K;

key->cnt = 0;

}

else if(key->value == 1)

{

key->cnt++;

if(key->cnt >= key->cnt\_max)

{

key->status = long\_press\_K;

key->cnt = key->cnt\_max;

}

}

break;

case long\_press\_K:

if(key->value == 0)

{

key->status = up\_K;

key->cnt = 0;

}

break;

default:

break;

}

}

/\*\*

\* @brief 遥控器接收产生中断时所有键盘按键状态判断并更新

\*/

void all\_key\_board\_status\_interrupt\_update(rc\_base\_info\_t \*info)

{

key\_board\_status\_interrupt\_update(&info->mouse\_btn\_l);

key\_board\_status\_interrupt\_update(&info->mouse\_btn\_r);

key\_board\_status\_interrupt\_update(&info->Q);

key\_board\_status\_interrupt\_update(&info->W);

key\_board\_status\_interrupt\_update(&info->E);

key\_board\_status\_interrupt\_update(&info->R);

key\_board\_status\_interrupt\_update(&info->A);

key\_board\_status\_interrupt\_update(&info->S);

key\_board\_status\_interrupt\_update(&info->D);

key\_board\_status\_interrupt\_update(&info->F);

key\_board\_status\_interrupt\_update(&info->G);

key\_board\_status\_interrupt\_update(&info->Z);

key\_board\_status\_interrupt\_update(&info->X);

key\_board\_status\_interrupt\_update(&info->C);

key\_board\_status\_interrupt\_update(&info->V);

key\_board\_status\_interrupt\_update(&info->B);

key\_board\_status\_interrupt\_update(&info->Shift);

key\_board\_status\_interrupt\_update(&info->Ctrl);

}

/\*\*

\* @brief 所有键盘按键状态判断并更新

\*/

void all\_key\_board\_status\_update(rc\_base\_info\_t \*info)

{

key\_board\_status\_update(&info->mouse\_btn\_l);

key\_board\_status\_update(&info->mouse\_btn\_r);

key\_board\_status\_update(&info->Q);

key\_board\_status\_update(&info->W);

key\_board\_status\_update(&info->E);

key\_board\_status\_update(&info->R);

key\_board\_status\_update(&info->A);

key\_board\_status\_update(&info->S);

key\_board\_status\_update(&info->D);

key\_board\_status\_update(&info->F);

key\_board\_status\_update(&info->G);

key\_board\_status\_update(&info->Z);

key\_board\_status\_update(&info->X);

key\_board\_status\_update(&info->C);

key\_board\_status\_update(&info->V);

key\_board\_status\_update(&info->B);

key\_board\_status\_update(&info->Shift);

key\_board\_status\_update(&info->Ctrl);

}

uint32\_t rc\_offline\_cnt;

void RC\_HEART(rc\_t \*rc)

{

rc->info->offline\_cnt ++;

if(rc->info->offline\_cnt >= REMOTE\_OFFLINE\_CNT\_MAX)

{

rc->info->status = REMOTE\_OFFLINE;

rc->info->offline\_cnt = REMOTE\_OFFLINE\_CNT\_MAX;

}

else

{

rc->info->status = REMOTE\_ONLINE;

}

}

## SBUS编码代码

/\*

\* @Author: hwwx 93569047+hwwx@users.noreply.github.com

\* @Date: 2023-12-27 10:22:18

\* @LastEditors: hwwx 93569047+hwwx@users.noreply.github.com

\* @LastEditTime: 2024-01-02 09:02:46

\* @FilePath: \Master(GS)\Application\Remote\Sbus.c

\* @Description: 这是默认设置,请设置`customMade`, 打开koroFileHeader查看配置 进行设置: https://github.com/OBKoro1/koro1FileHeader/wiki/%E9%85%8D%E7%BD%AE

\*/

#include "Sbus.h"

extern UART\_HandleTypeDef huart3;

uint8\_t packet[25];

uint16\_t channels[16] = {1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024};

extern rc\_t rc\_structure;

void move\_control()

{

}

void Sbus\_motifily()

{

if(rc\_structure.base\_info->s1.value == 1 && rc\_structure.base\_info->s2.value == 1)

{

move\_control();

}

}

void Sbus\_encore()

{

channels[0] = rc\_structure.base\_info->ch0 + 1024; // roll

channels[1] = rc\_structure.base\_info->ch1 + 1024; // pitch

channels[2] = rc\_structure.base\_info->ch2 + 1024; // yaw

channels[3] = rc\_structure.base\_info->ch3 + 1024; // 油门

channels[4] = rc\_structure.base\_info->s1.value;

channels[5] = rc\_structure.base\_info->s2.value;

channels[6] = rc\_structure.base\_info->thumbwheel.value + 1024;

// SBUS header

packet[0] = 0x0F;

// 16 channels of 11 bit data

packet[1] = (unsigned char) ((channels[0] & 0x07FF));

packet[2] = (unsigned char) ((channels[0] & 0x07FF)>>8 | (channels[1] & 0x07FF)<<3);

packet[3] = (unsigned char) ((channels[1] & 0x07FF)>>5 | (channels[2] & 0x07FF)<<6);

packet[4] = (unsigned char) ((channels[2] & 0x07FF)>>2);

packet[5] = (unsigned char) ((channels[2] & 0x07FF)>>10 | (channels[3] & 0x07FF)<<1);

packet[6] = (unsigned char) ((channels[3] & 0x07FF)>>7 | (channels[4] & 0x07FF)<<4);

packet[7] = (unsigned char) ((channels[4] & 0x07FF)>>4 | (channels[5] & 0x07FF)<<7);

packet[8] = (unsigned char) ((channels[5] & 0x07FF)>>1);

packet[9] = (unsigned char) ((channels[5] & 0x07FF)>>9 | (channels[6] & 0x07FF)<<2);

packet[10] = (unsigned char) ((channels[6] & 0x07FF)>>6 | (channels[7] & 0x07FF)<<5);

packet[11] = (unsigned char) ((channels[7] & 0x07FF)>>3);

packet[12] = (unsigned char) ((channels[8] & 0x07FF));

packet[13] = (unsigned char) ((channels[8] & 0x07FF)>>8 | (channels[9] & 0x07FF)<<3);

packet[14] = (unsigned char) ((channels[9] & 0x07FF)>>5 | (channels[10] & 0x07FF)<<6);

packet[15] = (unsigned char) ((channels[10] & 0x07FF)>>2);

packet[16] = (unsigned char) ((channels[10] & 0x07FF)>>10 | (channels[11] & 0x07FF)<<1);

packet[17] = (unsigned char) ((channels[11] & 0x07FF)>>7 | (channels[12] & 0x07FF)<<4);

packet[18] = (unsigned char) ((channels[12] & 0x07FF)>>4 | (channels[13] & 0x07FF)<<7);

packet[19] = (unsigned char) ((channels[13] & 0x07FF)>>1);

packet[20] = (unsigned char) ((channels[13] & 0x07FF)>>9 | (channels[14] & 0x07FF)<<2);

packet[21] = (unsigned char) ((channels[14] & 0x07FF)>>6 | (channels[15] & 0x07FF)<<5);

packet[22] = (unsigned char) ((channels[15] & 0x07FF)>>3);

// flags

packet[23] = 0x00;

// footer

packet[24] = 0X00;

}

void Sbus\_transimit()

{

/\*

a.高速模式：每4ms发送一次

b.低速模式：每14ms发送一次

\*/

if(rc\_structure.info->status == REMOTE\_ONLINE)

{

HAL\_UART\_Transmit\_DMA(&huart3,packet,sizeof(packet));

}

else

{

}

}

void Sbus\_work()

{

Sbus\_motifily();

Sbus\_encore();

Sbus\_transimit();

}

## 跟踪节点代码

#include <cmath>

// 定义 PID 控制器的参数

double Kp = 0.1; // 比例系数

double Ki = 0.01; // 积分系数

double Kd = 0.05; // 微分系数

// 定义目标位置（画面中心）

int target\_x = 640 / 2; // 假设画面宽度为640

int target\_y = 480 / 2; // 假设画面高度为480

// 定义 PID 控制器的变量

double integral\_x = 0;

double integral\_y = 0;

double prev\_error\_x = 0;

double prev\_error\_y = 0;

void bb\_cb(const yolov5\_ros\_msgs::BoundingBoxes::ConstPtr& msg)

{

BoundingBox = \*msg;

// 获取YOLO输出的第一个边界框的位置

int box\_x = BoundingBox.bounding\_boxes[0].xmin + (BoundingBox.bounding\_boxes[0].xmax - BoundingBox.bounding\_boxes[0].xmin) / 2;

int box\_y = BoundingBox.bounding\_boxes[0].ymin + (BoundingBox.bounding\_boxes[0].ymax - BoundingBox.bounding\_boxes[0].ymin) / 2;

// 计算误差

int error\_x = target\_x - box\_x;

int error\_y = target\_y - box\_y;

// 计算积分

integral\_x += error\_x;

integral\_y += error\_y;

// 计算微分

double derivative\_x = error\_x - prev\_error\_x;

double derivative\_y = error\_y - prev\_error\_y;

// 计算控制输出

double control\_output\_x = Kp \* error\_x + Ki \* integral\_x + Kd \* derivative\_x;

double control\_output\_y = Kp \* error\_y + Ki \* integral\_y + Kd \* derivative\_y;

// 更新上一次的误差

prev\_error\_x = error\_x;

prev\_error\_y = error\_y;

// 在这里使用控制输出进行相应的操作，例如更新飞行器的速度

ROS\_INFO("Control Output X: %f, Y: %f", control\_output\_x, control\_output\_y);

}