**2019代码简要说明**

**git branch**

master:最基础的代码（很久没动过了）

infantry:步兵(NUC)

infantryTX2:步兵(TX2)

hero:英雄

sentry:哨兵

UAV:无人机

pinwheel:大风车（识别算法已完成，未写电控切换模式以及转动方向）

kalman:测试运动预测时的分支，已停用

pid:研发PID时的测试分支，已merge到其他车

**代码总体架构：**

在2016年官方开源代码上进行修改（明年可考虑研究大连交通2018开源代码）

**可执行文件：**RM （主程序） CameraTest（相机曝光度调节及标定照片拍摄） Calibrate（相机标定）

**使用方法：**

**RM** 无参数：无画面输出，用于开机自启；加参数 -t ：有画面输出，用于调试

使用前先确定摄像头的file descriptor路径（默认为/dev/video0)，并在param\_config.xml中修改（可参考西安电子科技大学2018哨兵开源的方法绑定摄像头）

若需要和STM32通信，则需要确定串口路径并在main.cpp中修改（可以放到xml里面，参考摄像头的，只是我懒得改了2333）（建议使用ch340转接，路径默认为/dev/ttyUSB0）

**CameraTest**  **拍照前一定要确保已切换到画面的窗口！！！**

q：切换到720p60帧模式

e：切换到480p120帧模式

w：增加曝光

s：减少曝光

空格：拍照（默认保存至 项目文件夹/Calibration/pic，文件名为0.jpg 1.jpg ...）**（如果没有这个文件夹，就创建一个）**

ESC：退出

**Calibrate** **如果Calibration文件夹里面没有result文件夹，则创建一个**

照片路径在imageList.xml里面，照着改就行

标定设置在calibrationParams.xml

BoardSize\_Width和BoardSize\_Height是宽和高方向有多少个角（例如我们用的图片是10x7个角

Square\_Size是正方形边长，单位mm

Input是定义照片路径的xml文件（就是上面提到的那个）

Calibrate\_NrOfFrameToUse是标定使用多少张照片

**标定方法：**

1. 用CameraTest拍照，注意标定板尽量在每个象限都出现，而且数量要和上面那行提到的数量一致

2.运行Calibrate，标定过程会逐张显示照片，自动跳完之后可以按ESC退出，文件会保存为Calibration/result/cam.xml

3. 修改标定数据文件名，并在param\_config.xml作相应的修改

**代码流程：**

**线程1：**ImageProducer 获取图像

从相机获取图像，将图像和帧序号放入data[]

**线程2：**ImageConsumer 处理图像（重点）

设置云台与相机的位置关系、初始化一堆类

主循环：判断模式、获取装甲对应的矩形（ArmorDetector）、计算需要转动的角度（相对值）、运动预测、PID计算、数据发送，判断目标丢失情况、输出效果图

**线程3：**paraReceiver 接收并处理来自STM32的数据

**调试步骤：**

**param\_config.xml**

把2个轴的PID参数都改为Kp=1, Ki=0, Kd=0

更改目标颜色enemy\_color

确认摄像头路径device

设置标定数据文件位置intrinsic\_file\_480和intrinsic\_file\_720

设置子弹速度bullet\_speed

将scale\_z和scale\_z\_480改为1.0（原来的可以备份）

设置一个合适的曝光度（可进入CameraTest来观察）

**main.cpp**

确认串口路径fd2car

**ImageConProd.cpp**（主要调ImageConsumer里面的）

设置云台与相机的位置关系barrel\_ptz\_offset\_y、ptz\_camera\_y、ptz\_camera\_z（具体定义参考官方技术文档）

TX2和妙算需要在ArmorDetector.cpp里面将#define USE\_NEON取消注释

运行主程序RM，观察识别情况，适当调节param\_config.xml里面的装甲识别参数（具体定义参考官方文档）

将目标装甲板放到一个确定的距离，识别，观察输出的距离值，根据比例调整param\_config.xml里面的scale\_z（另外一个基本不用调）。同一位置、不同位置都多次重复这个步骤，直到输出的距离值比较符合实际

根据跟踪情况和电控协商调pid**（可以视觉调，也可以电控调，注意电控调pid的话要给视觉单独调一套pid，不能和其他共用！！！）**(PID的原理自学，或者听电控相应培训）调试的原则是先将P从0逐渐增大，在不抖动的情况下尽量给大；如果出现跟踪稳定落后于目标，则适当增加I；如果调完前面2个还有部分比较高频的抖动，则增加D

然后差不多就可以了（通信协议见下面）

**通信协议：**

波特率：115200， 数据位8位，停止位1位，无奇偶校验，无硬件流控制

**视觉传给电控：（serial.cpp）**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 字节序号 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 内容 | 0xFF | yaw轴转的角度 | | pitch轴转的角度 | | 自定义数据 | | 0xFE |

**角度说明：**数据值为角度值乘100（将double的数据强转为short（2字节）

**高低位说明：**未定义NETWORK的情况下，第1位是yaw角度的低8位，第2位是yaw轴角度的高8位，其他同理（详情自行了解字节序）（如果电控获取的角度数据出现高低位颠倒（反正数据很奇怪就对了），可以考虑将serial.cpp的#define NETWORK取消注释，或者让电控来处理

**自定义数据：**留了16位，可用于实现各种辅助功能（比如运行状态（无目标，发现目标，适合射击），目标距离等），详见ImageConsumer主循环发送数据部分send\_data[2]，待深入开发

**电控给视觉：（RemoteController.cpp）**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字节序号 | 0 | 1 | 2 | 3 |
| 内容 | 0xFF | cmd1 | cmd2 | 0xFE |

cmd1和cmd2是2个8位的数据，可实现各种辅助功能（比如识别颜色，运行模式（装甲，能量机关等），能量机关转动方向等），待深入开发

**我们代码和官方代码的不同之处**

将和项目相关的路径改为相对路径

将经常调的参数放到xml

增加相机测试和标定程序

可选将部分数据导出成csv格式

增加pid计算功能

部分哨兵，无人机，英雄利用了自定义数据

将装甲识别代码用SSE2指令集来写

串口字节序可调

新版能量机关识别算法（半成品，粗暴对位版算法）

删除了已经过时的部分代码

这篇文档只能说是提供了一个学习的方向，具体的实现过程需要自己仔细阅读每一行代码，了解各个函数和数据之间的关系（大概知道也好）。