bubing22

Main.cpp

两灯最大角度差
无灯两边最大角度差
两灯长度最大比值
中心距与灯长度大比值
两灯横向最小距离
筛选区域最小亮度
mser检测结果方重若重叠阈值
欧式距离与旋转向量变动值的线性组合权重
装甲片移动装甲向量变动阈值
装甲片移动空间欧式距离阈值
装甲片移动相对车辆的 yaw阈值
当前要传给上位机的那个装甲片目标的id
targetsHistory
maxHistory
latestShootTarget
latestShootTarget.count = -1;
frameCounter
cv::HOGDescriptor hog;
cv::Ptr<cv::MSER> mser;
cv::Ptr<cv::ml::RTrees> forest;
cv::Ptr<cv::ml::SVM> svm;
std::vector<cv::Point2f> armorFigure5d;// 装甲板实际比例缩放后取的图片尺寸

初始化 DetectArmor 类的对象 detect_armor

yaw = 0.0
pitch = 0.0
cv::VideoCapture cap

摄像头参数初始化 cameraInit()
读入相机参数

相机设置
1. 分辨率设置
2. 曝光设置

开始

通讯初始化

指定取线程

log工具模式选择，0: 终端输出 1: 窗口带最终结果 2: 窗口详细输出

初始化

打开相机

是否成功打开

否

是

cap.read(frame)
读入摄像头图像到 cv::Mat frame

运行探测装甲主程序 detect_armor.runPlus(frame, pitch, yaw); 获得Track flag和pitch yaw

返回resultId

resultId >= 0

是

通过detect_armor.shootStrategy()
判断是否Target最近两次都出现而且yaw值在0.5以下
得出是否击打，返回1是打，0是不打

communicator.send(++timestamp, -yaw, pitch, shootFlag, 0)
将timestamp, -yaw, pitch, shootFlag串口通信发送给电控

runPlus & run

之前是否已找到目标

是

获取放大的latestShootRect的ROI

在ROI中运行run函数寻找目标

未找到目标

开始在原图中运行run函数寻找目标

找到目标

detect(frame, totalTargets)获得totalLights

detectFrame, totalTargets)函数寻找所有可能的目标结果放入totalTargets容器

开始detectFrame, totalTargets)函数探测灯条

转化HSV和灰度

初始化
std::vector<std::vector<cv::Point>> regions; //区域点集容器
std::vector<cv::Rect> bboxes; //矩形容器

MSER探测灰度图中的bbowes
mser->detectRegions(gray, regions, bboxes)

去除太小的轮廓和位于gray边缘的轮廓

去除重复的轮廓

去除不符合颜色要求的轮廓

去除不符合长宽比的轮廓

detect(frame, totalTargets)配对totalLights

将结果放入std::vector<Light> totalLights容器

开始detectFrame, totalTargets)函数匹配灯条

以灯条中心的x坐标从左到右排序灯条

获取
两灯条角
两灯条质点之差
两灯条长度之比
两灯条中心点距离
灯条中心距离与两灯条长度均值的比值

筛选灯角度差，两灯条长度比值，灯条中心距离与两灯条长度均值的比值，两灯条横向间距，左灯条倾斜角，灯条顶线的斜率

将筛选结果灯条两两匹配，获取灯条匹配矩形的四个顶点，将顶点放入target类，将将target放入totalTargets容器

classify(totalTargets, finalTargets)

通过分类器classify(totalTargets, finalTargets)函数结果放入finalTargets容器
返回classifyflag

开始classify(totalTargets, finalTargets)函数

遍历totalTargets
将截图crop放入std::vector<cv::Mat> img_bz 图像容器

使用computeHOG(img_bz, gradient_bz, false)函数将img_bz的梯度值放入gradient_bz矩阵中

将convert_to_mlp(gradient_bz, data)函数将gradient_bz矩阵的结果转化为机器学习可读的格式data

用forest->getVotes(data, result, cv::ml::DTrees::PREDICT_MAX_VOTE)获取每个Target的随机森林分类投票结果

分类器检测结果得分

小于40

舍弃

大于40

识别为装甲板

两灯条中心距离与两灯条长度均值的比值

>=1.1

< 1.1

识别为英雄
Targets.type 赋值为1

识别为一般目标
Targets.type 赋值为0

放入heroTargets

放入finalTargets

是否发现英雄

否

将英雄帧数计数器清零
发现常规单位
Return classifyFlag = 0

是

是否连续3帧发现英雄

否

将英雄帧数计数器++
将heroTargets加到finalTargets后面
发现常规单位
Return classifyFlag = 0

是

直接用heroTargets覆盖finalTargets
发现英雄 return classifyFlag = 1

latestShootTarget是否英雄

是

clearTrack()
重置跟踪

getArmorLocation(target)

getArmorLocation(Target)个Target的位置

开始getArmorLocation()argets)函数

设定装甲板四点和中心的世界坐标

调用solvePnP(gpointArmor, targets.pixelPts2f, g_cam_matrix, g_dist_coeffs, Ex, Tx) 获得旋转向量和平移向量

罗德里格斯变换Rodrigues(Tx, Rx)将旋转向量tx变换为旋转矩阵Rx

centerInCameraCoordinate = Rx * centerhero + Tx
用旋转向量解出centerInCameraCoordinate，即装甲板中心在相机坐标中的位置

获得装甲板中心在焦口坐标中的位置
cv::Mat centerInShotterCoordinate = g_in_rota_matrix * (centerInCameraCoordinate - g_trans_matrix)

计算陀螺线补偿

使用curve_fit()获得系数对x和y进行补偿

多项式（二次）最小二乘拟合预测yaw

使用angles = cal_degree(location)计算得到pitch和yaw

track(finalTargets, pitch, yaw)

通过跟踪算法track(finalTargets, pitch, yaw)获取track flag

开始track(finalTargets, pitch, yaw)函数

旧记录自增，删除6次之前记录

是否已选定打击对象

否

找yaw最近的

把target的yaw值从小到大排序，并找到yaw最小的closestTarget

将closestTarget赋值给latestShootTarget并放入targetsHistory容器

是

找到当前图片里的打击对象

比较latestShootTarget和当前目标yaw的欧式距离和轮廓相似度

阈值以外

舍弃

阈值以内

若距目标欧氏距离<999999

否

latestShootTarget.count += 1 这帧没找到

是

将当前目标放入targetsHistory容器

targetsHistory容器

为空

没找到，也没的去瞄
return -1

不为空

latestShootTarget.count

!=0

这帧没找到，但还是去瞄着历史位置
return 0

==0

这帧找到上一帧的目标
return 6