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检测结果方差的最小阈值
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::KNearest> forest;
cv::Ptr<cv::ml::SVM> svm;
std::vector<cv::Point2f> armorFigureSrd;// 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

指定双线程

log工具模式选择,
0: 纯输出 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

runPlus & run

之前
是否已找到
目标

是

获取放大的
latestShootBRect
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

返回resultid

resultId >= 0

是

detect_armor.shoot
Strategy()通过判断
yaw值得出是否击
打1是打, 0是不打

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

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bBoxes
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
两灯条角点之差
两灯条长度之差
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_ts图像容器

使用
computeHOGs(img_
ts(, gradient_ts(,
false)函数得img_ts
的梯度值放入
gradient_ts(数阵中

将
convert_to_mlp(grad
ient_ts(, data)函数
将gradient_ts(矩阵
内的结果转化为机
器学习可读的格式
data

用forest->predict(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

latestShootTarget
是否英雄

是

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

clearTrack()
重置跟踪

否

getArmorLocation(target)

getArmorLocation(t
arget)获得
个Target的位置

开始
getArmorLocation(t
arget)函数

设定装甲板四点和
中心的世界坐标

调用
solvePnP(getPointArr
mor,
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(Rv, Rv)将
旋转向量Rv变换为
旋转矩阵Rv

centerInCameraCoo
rdinate = Rv *
centerHero + Tv
用旋转向量解出
centerInCameraCoo
rdinate, 即装甲板
中心在相机坐标中
的位置

获得装甲板中心在
像口坐标中的位置
cv::Mat
centerInShotterCoo
rdinate =
g_in_rota_matrix *
(centerInCameraCo
ordinate -
g_trans_matrix)

计算触物线补偿

使用curve_fit()获得
系数对x和y进行补
偿

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

是

找到当前图片里的
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

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

看看目标欧氏距离<999999

否

latestShootTarget.c
ount += 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

这帧找到上一帧的目标
return 6

<=0

这帧没找到, 但还是去
瞄着历史位置
return 0