TJU RM 2018 Advanced 算法源码文档

1.概述

RM2018 Advanced 源码使用 C++编写，实现了以下三个功能：自动射击、大小能量机关打击、自动跟踪拦截。源码包括 20 个 hpp/cpp 文件，1~3 个 xml 文件（保存摄像头标定参数），1 个 ini 文件（可调参数文件），一个 model 文件夹（包括小能量、大能量、装甲类别识别的模型文件）。

2.架构

主程序使用三线程并行运行，分别为：

① 摄像头图像获取线程 ImageCollectThread，该线程不停获取最新图像。

② 图像处理线程 ImageProcessThread，该线程不停处理最新图像。

③ 图像显示线程 ImageDisplayThread，调试状态下，该线程显示所有调试图像。

2.1 运行流程

主程序首先加载可调参数配置文件（configuration.ini），由配置文件先设置摄像头参数（分辨率、曝光时间、自动曝光），随后检查配置文件中程序入口是否为标定程序，是则进入摄像头标定程序，否则开启上面三个线程。

摄像头图像获取线程：首先打开串口，随后进入图像读取循环，注意第一帧会强制重设一遍摄像头参数，这是为了保证摄像头参数被设置。

图像处理线程：首先初始化各模块：加载摄像头参数、装甲检测器、装甲跟踪器、装甲打击模块、轨迹预测器、深度学习管理器、神符识别、神符打击模块、自动跟踪模块。（注意：功能模块的初始化需要串口指针，所以初始化功能模块之前需要确保串口初始化完毕，也就是串口指针不为空）随后进入主循环，循环仅当图像获取线程得到最新图像才会继续后

面的操作，即选择合适的处理函数执行。

图像显示线程：没有初始化过程，进入循环等待需要显示的图像被传递，同时该线程当不处于调试模式的时候被屏蔽。

2.2 细节设计

程序的实现较于上面流程说明更为复杂，架构的时候遇到很多坑，这里介绍一些实现上的细节，免得你们重蹈覆辙。

线程锁 在图像处理线程和获取线程都要访问 frameData 变量，这是不可避免的。而 Mat 类变量在被同时访问时会触发一个异常，这个异常如果处理线程速度比获取线程快的话，是不会触发的（**自己想想为什么**），因此这个 bug 也是在我们将代码从 TX 移植至妙算上才开始频繁触发（妙算的算力不够）。解决方案是给 frameData 加一个线程锁，让他在同时只能被一个线程访问。

DebugDisplayManager 该类负责管理显示图像显示，为静态类。图像显示操作流程为：处理线程将一副想显示的图像提交至该管理器；显示线程显示最新被提交的图像（同名的图像将覆盖）。注意这里的实现每次显示完并不会将列表 clear 而是将已经显示的图像设置一个 visible 为 false。这是因为如果通过 clear 来处理，两个线程可能出现一边提交一边清除的情况，这时候提交的线程里列表的指针指向的位置是被释放的，从而导致 Segmentation Fault。

调试按键 调试时不免用到键盘，对于这个多线程架构，我们设计了专门的按键处理机制。按键在 Display 线程触发，保存在 waitedKey 变量中。但是因为线程不同步可能导致一个按键还没被响应就被覆盖，在 Display 线程中，只保存有效按键给 waitedKey，而在 Process 线程的主循环末尾将 waitedKey 清除，这样可以保证 waitedKey 在被抛弃之前起码被 Process 线程的处理函数访问一次。同样需要注意 Process 线程的处理函数中，应该把对 waitedKey 的处理写在图像处理代码的后面（**自己想想为什么？**）.

调试暂停 程序架构中已经实现了按 P 键可以暂停程序，暂停图像处理线程，为保证图像和处理程序同时暂停（**自己想想为什么要保证这个？**），将暂停代码全部写在 Process 线程中，但是你应该会发现响应不灵问题（结合上面一条，**自己想想应该怎么修改？**）。

2.3 模块化设计

架构使用了模块化设计，充分预留了接口使新的算法而不需要对原来的源码直接修改，而只需要在 main 中创建一个新的对象指针，并指定功能模块使用该指针进行操作即可。该设计保证程序的兼容性，方便回档，不会破坏源码，向上兼容能力强。

功能模块:功能模块是指能被下位机调用的模块。功能模块从 ModuleBase 类(util.hpp) 继承 ,在调用 ModuleBase 的构造函数时应该指定模块的 ID(ID 从 2 的指数中选取 1 2 4 8 … 64，128 表示不调用模块)。功能模块在 SerialManager 中被管理调用，添加新的功能模块只需要编写子类然后在 main 中注册到 SerialManager 里面，然后下位机通过对应 ID 可以调用该功能模块。

自动射击的子模块：自动射击的实现中使用了三个子模块：

① ArmorBaseDetector 装甲检测器基类，可以检测图像中所有装甲。
② ArmorTrackerBase 装甲追踪器基类，自动选择然后一直跟踪其中一个目标，并计算射击角度，算法速率通常比检测器高。
③ Predictor 预测器的基类，根据历史数据预测敌车的若干时间后的位置。
④ ArmorHiter 本身也是可继承的，实现不同的射击策略。

神符打击子模块：DSFHiter 可继承，实现不同的打击策略。

自动跟踪子模块：自动跟踪正前方物体，暂时没有继承必要。

考虑到不同车有不同的射击策略不同的设备，你可以为其专门设计算法，并在 main 中组合使用即可。

2.4 串口通信协议

不论是上向下通信还是下向上通信，本通讯协议都遵守一个基本格式。一条指令代码和对应指令参数构成一段指令，规定指令代码一定大于 200，指令参数一定小于 200。每一次发送数据包含若干个指令段，并且在指令段的最后是一个停止字节，0xC8H（200D）。例如，假设指令 201 有三个参数、202 有 2 个参数、203 没有参数，最后发送这三段指令的数据应该为：

201 102 10 3 202 63 53 203 200

下面定义指令和指令参数。

上位机→下位机通讯：

指令代码	参数个数	描述
201	**4**	设置云台的目标偏转角，用两个字节记录一个角度，前两个字节为 Yaw 轴角度 后两个字节为 Pitch 轴角度。 Yaw 轴角度范围 -60~60；Pitch 轴角度范围 -30~30
202	**1**	给出推荐射速，参数为射速单位是 m/s，例如参数给出 20 则表示推荐使用 20m/s 射速射击。
203	**1**	射击指令，收到该指令应该在电控约束条件下发射弹丸，参数表示摩擦轮开启信息，是否射击： 1：不开启摩擦轮，不射击 2：开启摩擦轮，不射击 4：开启摩擦轮且射击
204	**0**	大神符打击完成指令
205	**0**	程序启动，请求获取裁判系统数据（敌方颜色） 2 为红色 ；0 为蓝色
206	**4**	设置云台目标绝对位置，记参数为 a,b,c,d，角度范围： 对哨兵：Yaw -500~500 Pitch -60,20 对步兵：Yaw -80~80 Pitch -50~50
207	**6**	设置底盘控制量，六个参数分别代表底盘 x 轴平移（朝右为正）、y 轴平移（向前为正）、Yaw 轴旋转（顺时针为正） 其中每一个控制量值 0~40000 均映射到-1~1，1 表示最大速度 哨兵中，仅 x 轴控制量有效

下位机→上位机通讯：

指令代码	参数个数	描述
201	**1**	调用当前上位机模块，参数为模块 ID，具体如下： 大神符打击模块 ID 为 1； 装甲追踪模块 ID 为 2； 补给站自动引导模块 ID 为 4； 自动拦截模块 ID 为 8； 关闭当前模块使用 128；
202	**1**	给出裁判系统信息，第一个字节为敌方颜色 红色：2 蓝色：0
203	**1**	返回当前的射速，该信息应该要经常更新（大约 100ms）
204	**4**	返回云台当前速度（快速更新，大约 20ms），参数信息如下 假设参数为 a,b,c,d，云台速度： Yaw -2pi~2pi Pitch -2pi~2pi
205	**0**	（指令废弃）大神符完成一次打击，请求下一次打击
206	**4**	返回云台当前角度（快速更新），参数信息如下， 假设参数为 a,b,c,d，云台当前角度表示为（朝右为正，朝上为正）

		哨兵：Yaw -500~500　Pitch -60~20 步兵：Yaw -80~80　Pitch -50~50
207	**4**	返回当前位置的估计值，分别为 x,y，参数值 0~40000 分别映射到-30~30、-30~30。

3.算法

算法分为图像处理部分和深度学习部分，分别应用在了装甲检测和神符识别上。

3.1 装甲打击

检测的基本原理：

1) 图像二值化，提取高亮区域。

2) 轮廓提取并近似成多边形，对多边形包裹的区域，检查对应队伍颜色的分量，达到阈值的加入待选灯柱集合，同时记录队伍颜色多少。

3) 依据队伍颜色分量之和排序，取最大的若干个。

4) 对待选灯柱两两配对，计算其匹配程度，并将配对方式和计算结果加入待选结果集合。

5) 舍弃所有匹配程度低于阈值的配对方式，剩余结果按匹配程度排序。

6) 从大到小处理配对方式，若当前配对中的两个灯柱都未被使用过，加入检测结果集合，并标记这两个灯柱为使用过。

匹配得分计算：

包括几个部分：灯柱相对位置，多边形匹配程度，装甲中心是否存在贴纸等因素。具体请参考源码（armorDetector.hpp line 299）。

装甲跟踪原理：

1) 若上一帧没检测到或者已经跟踪了很多帧了，重新检测一帧全图。检测结果若有，则选择一个目标（根据兵种和位置，参考代码 armorTracker.hpp），否则查看上一帧是否有目标。若上一帧有目标，则用跟踪算法跟踪上一帧的目标。
2) 若上一帧成功检测且刚刚开始跟踪，则先对上一帧计算出装甲的世界坐标，重投影到屏幕上（**自己想想为什么这么做？**），在屏幕上该点附近选取一个搜索框，对搜索框中的图像用检测算法检测，若检测成功，使用检测结果；否则尝试使用跟踪算法（MedianFlow）跟踪上一帧目标.

装甲打击原理：

1) 对跟踪的装甲目标，计算世界坐标。
2) 对历史坐标预测一定时间之后的世界位置（对抗时滞）。
3) 将预测位置重新计算云台角度。
4) 对该角度叠加上抵消重力的角度。
5) 计算偏转角，并带入 PID 控制器（**想想为什么要用 PID 控制器**）。
6) 输出加上当前角度得到绝对角度，通过串口发送。
7) 如果偏转角小于 1°，那么发射弹丸。

3.2 能量机关打击

图像处理部分自己看代码吧，懒得写了。

注意神经网络识别九宫格之后，要做一个最大匹配，计算出总概率最大的一个排列，由

此大大提高识别正确率。

3.3 神经网络

兵种识别、大小能量机关识别都通过神经网络实现。实际使用时用 Darknet 框架，模型训练时用Tensorflow得到pb模型,然后通过工具转为Darknet需要的.cfg和.weight文件，再投入实际使用。

兵种识别：通过识别装甲上贴纸的数字实现。这是一个分类模型，输出分为 6 类，分别表示未知和数字 1~5。网络仅使用全连接层实现，输入层为 1x784(28x28)的行向量，一个 100 个神经元的隐含层，输出层 6 个神经元，后接 softmax 导出概率。隐含层激活函数使用 LeakyRelu。数据集是自己采集、标注的数据集。

小能量机关识别：使用 MNIST 手写数字数据集，用卷积神经网络识别。

① 网络架构输入层，大小为 28x28x1；
② 卷积[5,5,1,32]，输出尺寸不变，步长 1，Relu 激活；
③ 最大池化 2x2，步长为 2；
④ 卷积[5,5,32,64]，输出尺寸不变，步长 1，Relu 激活；
⑤ 最大池化 2x2，步长为 2；
⑥ Reshape 到 1x3136
⑦ 全连接层 1024 个神经元，Relu 激活；
⑧ 全连接 10 个神经元；
⑨ Softmax 得到概率；

大能量机关识别：模型和兵种识别的模型差不多，输入变成了 32*32*3=3072 而已。

4.注意事项

程序在部署到一台新设备或者新车上时，需要修改文件 custom_definations.hpp，主要是几个宏定义和文件路径需要配置。编译完成之后，需要为新摄像头重新标定，修改 mainEntry 为 10 可以进入标定程序。程序可调参数对每辆车都有可能不一样。

注意:分辨率类型中,如果选择 640x480 分辨率,程序从摄像头获取的分辨率为 640*480，但是会手动放大至 720P，这是为了保证识别范围。

注意：在妙算或者 TX 上编译源程序时，经常会出现“在未来修改文件”的错误，导致程序不能正确编译，这个时候可以输入下列指令：

```
find . -type f -exc touch {} \;
make clean
make
```

一般就可以编译成功了。

GOOD LUCK