6.1 装甲板识别

本着从实际出发的精神，在讲解算法思路前先来分析一下装甲板，图6.1为官方RM2021机器人制作规范手册中的步兵图像。

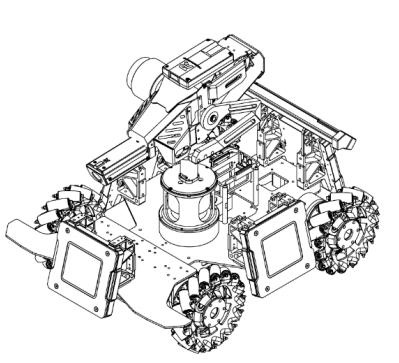


图6.1 步兵制作规范图像

在图中可以观察得出有关于装甲板的信息：

1. 装甲板略带倾斜固定在小车的四周。
2. 同一块装甲板的两灯条平行、灯条长宽确定、两灯条间的水平间距确定。
3. 在步兵以45°对敌时（图示状态），会有两个装甲板出现在摄像头画面中。
4. 两个装甲板的倾斜角度相差不大，容易将中间两个灯条误识别成大装甲板（难点）。

根据以上信息，可以初步构思出识别装甲板的思路。

1. 用findContours函数找出图片中所有的灯条。
2. 根据长宽比、面积大小和倾斜角度来筛选出正确的灯条。
3. 对正确的灯条进行配对，根据装甲板长宽比、灯条长度比、灯条角度差和灯条高度差来筛选出合适的灯条配对作为候选装甲板。
4. 提取候选装甲板中间的图案做数字识别进行筛选和分类。

6.2 数字识别算法

由上文可知，在实际比赛情况下，每个机器人在四个方向上均有一块装甲板，在识别装甲板的过程中，很有可能将两个装甲板之间的区域误认为是一块装甲板，且场地上灯光等因素也会给传统视觉识别方案带来很大干扰，因此需要加入数字识别功能，进一步筛选检测到的所有装甲板。

考虑到在嵌入式平台上的算法运行速度，并未采用深度学习中目标检测方法进行数字识别，使用了基于伽马校正和SVM的方法进行数字区域的识别和筛选。首先，对于上一步骤中获得的所有装甲板区域，需要对其进行伽马校正以增强数字区域亮度，之后进行二值化操作，以获得待放入SVM识别的图像数据。伽马校正的计算公式为：

其中系数A一般取1，和为像素点的灰度值。如图6.2所示，当时，在低灰度值区域内，像素值动态范围变大，对比度和图像整体的灰度值变大。当时，在低灰度值区域内，像素值动态范围变小，对比度和图像整体的灰度值变小。由于装甲板图片在低曝光下进行采集，需要令来增加图像亮度。经过不同光照环境下的大量测试，在本程序中取=0.46。

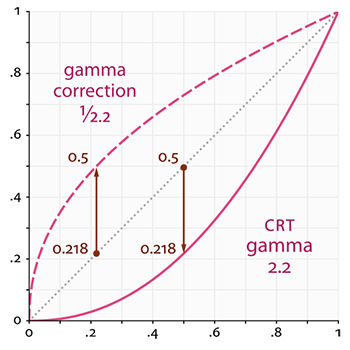


图6.2 伽马校正示意图

此外，由于OpenCV中SVM模块在多分类时，无法获得当前输入图像的置信度，从而导致将非数字区域识别为数字区域。因此，可以通过指针的方式遍历图像中的所有像素点，统计所有白色像素点的个数并除以图像尺寸，即可获取数字区域在图像中的占比，进而设置阈值来判断是否为装甲板区域。图6.3为经过伽马矫正后1-5号装甲板的图像。



图6.3 伽马矫正后装甲板图像

支持向量机（SVM）是一种用来解决二分类问题的监督学习算法，它的基本模型是在特征空间上找到最佳的分离超平面，如图6.4所示，以使得训练集上正负样本间隔最大。在引入了核函数之后，SVM也可以用来解决非线性问题。在将正负样本分离之后，SVM还需尽最大努力使分开的正负样本有最大间隔，使得分隔具有更高的可信度，而且对于未知的新样本有更好的泛化能力。此外，SVM基于强大的数学工具，在处理超高维空间的特征时效果很好。实践表明，当特征空间的维度超过样本数量时，SVM的效果是最好的。SVM占用系统的内存很少，只需要存放支持向量，而最近邻法等算法则需要将全部样本点存放在内存中。

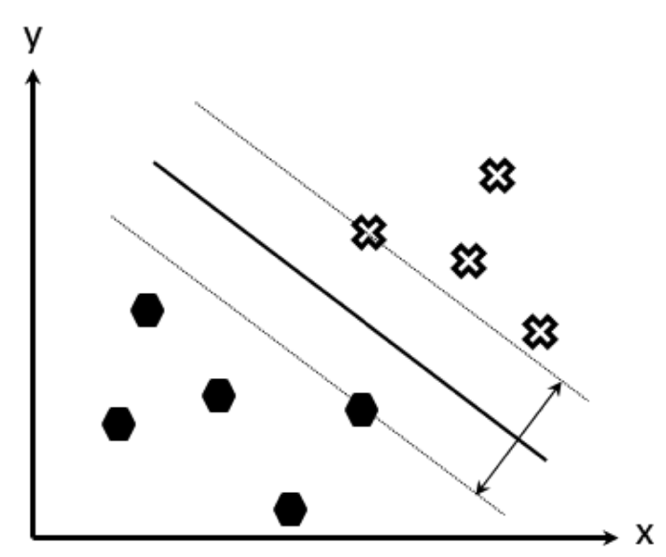


图6.4 SVM的基本模型

方向梯度直方图（Histogram of Oriented Gradients，HOG）是一种表示图像形状和纹理的模型，该模型将图像划分为网格，分别计算不同网格中梯度方向的分布图。构建分类器时，由于HOG可以看作是一个鲁棒的高维描述子，能准确反映一个类别的本质特征，因此将HOG和SVM结合的效果较好。

此处使用OpenCV中的HOGDescriptor类，实现了HOG特征的计算。输入图像尺寸统一缩放至48\*48，采用9箱直方图，区块大小为16\*16，单元格为8\*8，步长为一个单元格，产生的HOG向量大小为900。随后使用OpenCV中的ml::SVM类，构建了SVM分类器。选择SVM的类型为SVM::C\_SVC以处理分类问题，核函数选择线性内核SVM::LINEAR，线性分类在原始特征空间内完成，以提高处理速度；设置训练条件为训练10000次或误差小于1e-6时结束。

6.3 位姿解算

第一步，需要根据大小装甲板的真实尺寸建立世界坐标系，如图6.5所示。

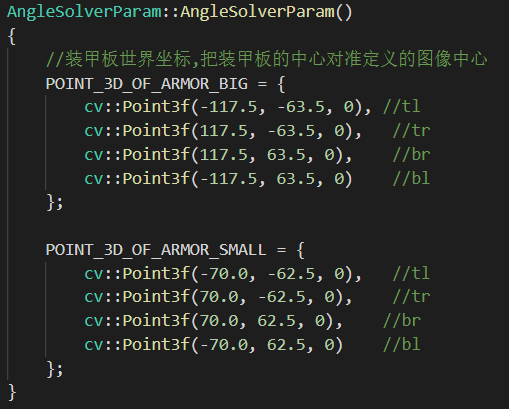


图6.5 真实装甲板坐标

第二步，调用solvePnP函数以获取装甲板坐标。

要使用solvePnP，先要对摄像头进行标定，得到相机的内参矩阵和畸变参数。标定摄像头时应注意：

1. 在标定前对着标定板进行白平衡操作并调整合适的焦距，使棋盘格黑白边界更鲜明。
2. 初次标定后，应挑选出误差较大的图片删去，再次标定。

之后使用如下自封装solvePnP函数：

solvePnP(\_params.POINT\_3D\_OF\_ARMOR\_EMERGY,objectPoints, \_params.CameraIntrinsicMatrix, \_params.DistortionCoefficient, \_rVec, \_tVec,false);

参数说明：

objectPoints：特征点的世界坐标，坐标值需为float型，不能为double型，可以为mat类型，也可以直接输入vector。

imagePoints：特征点在图像中的像素坐标，可以输入mat类型，也可以直接输入vector，注意输入点的顺序要与前面的特征点的世界坐标一一对应。

cameraMatrix：相机内参矩阵。

distCoeffs：相机的畸变参数，属性为Mat\_(5, 1)。

rvec：输出的旋转向量。

tvec：输出的平移矩阵。

第三步：计算yaw轴偏差、pitch偏差以及距离：

因为目的是让枪管瞄准装甲板的中央，而通过solvePnP函数得到平移向量和旋转向量是以摄像头中心为原点的，要得到正确的转角需要进行**平移**，将原点平移到**云台的轴**上。

一般而言x坐标可以不用动，因为摄像头一般放在枪管的上方这样x方向就没有偏差。y方向和z方向需要修正。修正后即可通过反三角函数得到角度值，算法实现如图6.6所示。

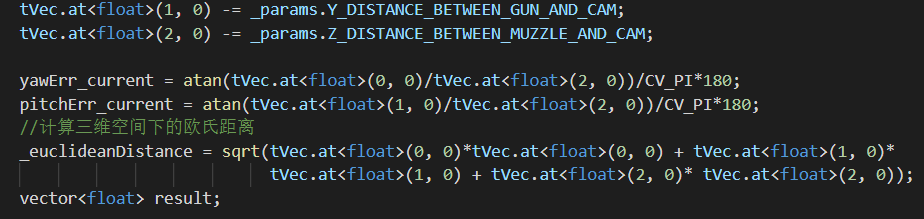


图6.6 获取偏转角度及距离算法实现

6.4 代码设计

哨兵视觉自瞄上位机程序基于Manifold 2-G，Qt 5.15.2以及Qt Creator IDE构建。上位机整体结构框图如图6.7所示，采用了Qt中的QThread多线程框架进行构建，所用到的外设有Manifold上的CAN0接口，工业相机MV-SUA33GC-T。

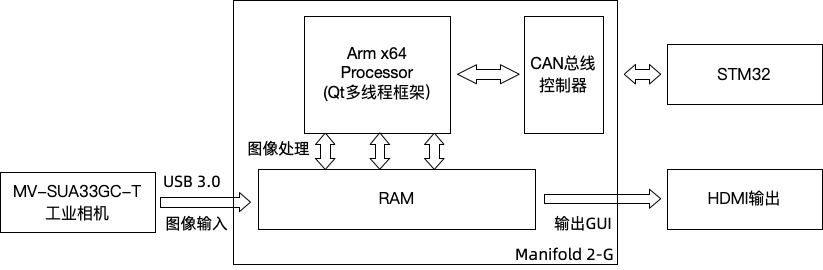


图6.7 上位机整体结构框图

本程序分为主线程，图像采集线程以及处理线程。主线程负责对图像采集和处理时序的控制，调试时GUI界面的刷新以及CAN总线的收发操作。如图6.8所示，基于Qt中信号与槽的设计理念，使用互斥锁和connect方法实现各线程中数据的异步传输。

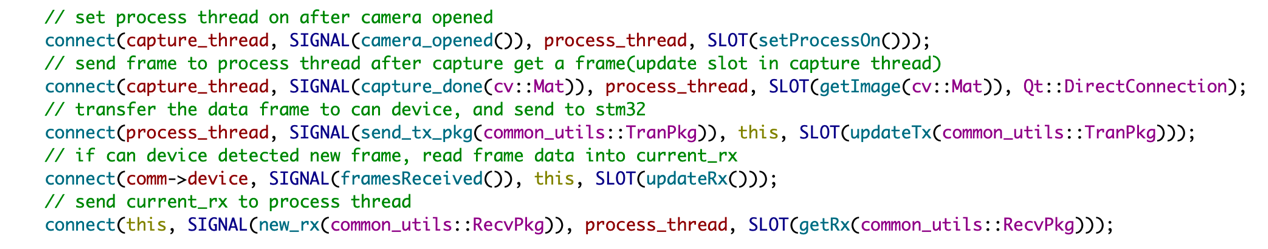


图6.8 多线程中数据传输方式

在程序主线程中通过对QObject对象TaskManager的相关操作，实现了对采集以及处理线程工作时序的控制，以及CAN总线上数据的更新逻辑。在TaskManager中包含QObject对象CommProtocol以及QWidget对象ImageShow，以及QThread对象CaptureThread、ProcessThread。采集线程与处理线程为QThread类的继承，通过重写其run虚函数，即可在主线程中通过调用start方法开启子线程循环。程序工作流程如图6.9所示。

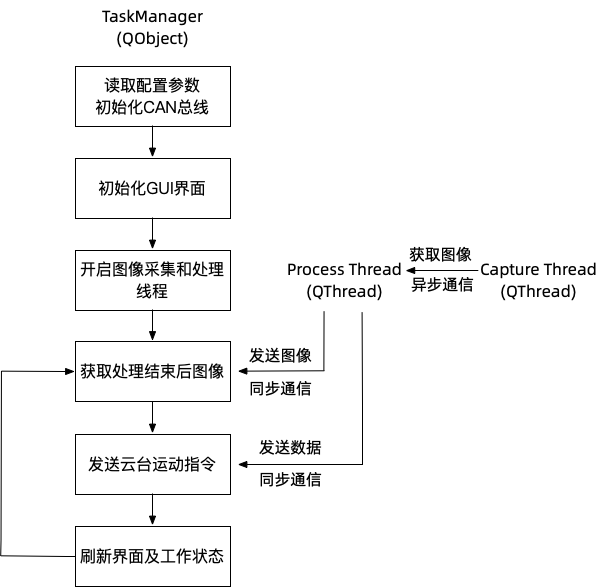


图6.9 程序工作流程

采集线程以一定的帧率采集当前场景的图像，与软件中断的思想类似，采集的图像将以信号的形式将采集到的图像发送至处理线程的槽函数中，实现图像的更新，并置图像更新标志位为true。处理线程在线程循环中不断检测图像更新标志位，若检测到图像发生了更新，则进入后续自瞄程序；在图像处理完成之后，将会筛选出此时的首要打击目标点，经位姿结算后将当前需传给电控端的偏转角度发送给主线程相应的CAN通信槽函数中，实现对哨兵云台的实时控制。此外，在调试模式下，主线程将CAN总线上的收发数据以及处理后的图像实时显示在GUI界面中，程序运行时显示的GUI界面如图6.10所示。



图6.10 程序运行界面

此外，为了提高代码的可读性以及调节参数时的便利性，使用了Qt中关于json文件读写的接口实现了算法中各个参数的统一保存与读取；使用qInstallMessageHandler实现了分级全局日志管理功能。

6.4 通信方式

本程序基于Linux SocketCan以及Qt中QCanBus模块实现了Manifold与下位机之间的数据通信。使用iproute工具设置波特率为1Mbps，使用QCanBusDevice::Filter设置CAN总线上的帧过滤，使得发送帧的ID为0x101，接收帧的ID为0x102，开启fd扩展帧和berr-reporting选项。

CAN总线的通信收发格式分别如图6.11和图6.12所示，数据帧长度为8个字节。

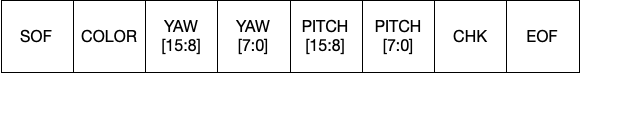


图6.11 接收数据协议



图6.12 发送数据协议

对于从STM32端接收的数据，约定其最高位为帧开始位0x55；第6位COLOR为从裁判系统读取的当前我方颜色，红方时为0x00，蓝方时为0x01；第5位和第4位分别代表当前云台的yaw偏转角度的高八位和低八位，精度为0.01度；第3位和第2位分别代表当前云台的pitch偏转角度的高八位和低八位，精度为0.01度；第1位CHK为校验和；第0位为帧结束位0x00。

对于向STM32端发送的数据，约定其最高位为帧开始位0x00；第6位DIS为经由位姿解算出的当前距离打击目标的距离，精度为0.1m；第5位和第4位分别代表当前云台在yaw方向上所要偏转角度的高八位和第八位，精度为0.01度；第3位和第2位分别代表当前云台在pitch方向上所要偏转角度的高八位和第八位，精度为0.01度；第1位CHK为校验和；第0位为帧结束位0x00。

8.1 Qt以及Qt Creator的编译安装

在多次的代码调试修改过程中，我们发现使用C++原生多线程框架时，可能由于Manifold自身处理速度等问题，导致程序异常崩溃，且使用串口进行通信时，数据传输速度较慢且稳定性较差。因此，我们尝试使用Qt开发框架，使用Qt中的多线程框架以及内置的QCanBus模块代替串口通信。

在arm架构下使用apt安装的qt版本为5.5，且长期缺乏官方维护，出于稳定性考虑，采用了编译qt源码的方式进行安装，使用的版本为5.15.2 LTS。

8.1.1 安装相应的软件依赖

sudo apt-get install libxcb-xinerama0-dev libsocketcan-dev

sudo apt-get install '^libxcb.\*-dev' libx11-xcb-dev libglu1-mesa-dev libxrender-dev libxi-dev libxkbcommon-dev libxkbcommon-x11-dev

使用apt安装上述编译qt时所需的库即可。

8.1.2 配置相应的编译选项

cd /home/dji/data/qt-everywhere-src-5.15.2

./configure -opensource -nomake examples -nomake tests -skip qtwebengine

由于移植系统时，只会复制emmc中的内容，硬盘中的内容不会复制，此处选择的安装路径为默认的/usr/local，编译路径在/home/dji/data，即128G固态硬盘挂载的目录下，以防止emmc内存不足。注意不要加入-developer-build选项。若不跳过webengine模块，编译会遇到较多bug，且源码大小为2.1G，话费时间极长。

8.1.3 编译安装

make -j6

sudo make install

cd /usr/local/Qt-5.15.2/bin

./qmake -v

编译安装后的结果如图8.1所示，可见输出的qmake和qt的版本号均正确。

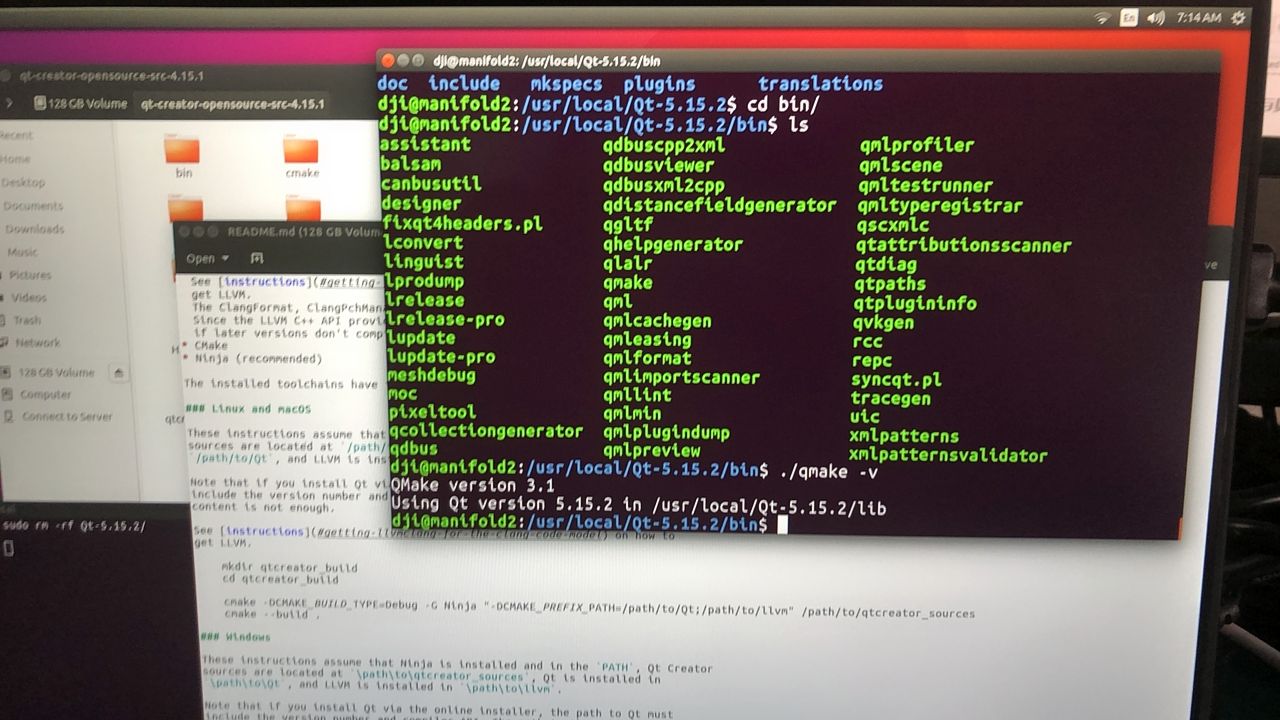


图8.1 编译安装qt

8.1.4 安装Qt creator

此处使用上文中安装的qmake对Qt creator 4.15.1版本源码进行编译安装。安装完毕后运行结果如图8.2所示。

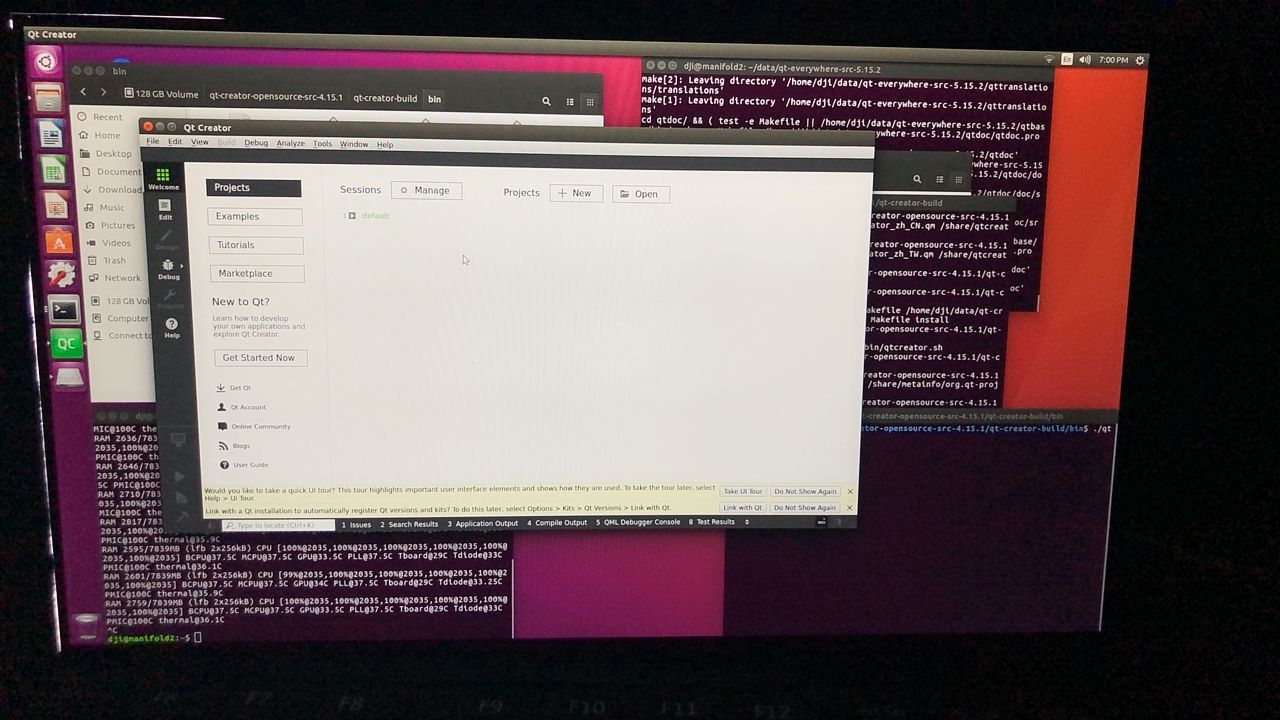


图8.2 安装Qt creator