总控软件软件设计与实现方案

目录

[1 方案设计 3](#_Toc17793)

[1.1 组成 3](#_Toc13015)

[1.2 架构设计 4](#_Toc26293)

[2 详细方案设计 5](#_Toc6318)

[2.1 主界面 6](#_Toc5749)

[2.1.1 软件组成 6](#_Toc11449)

[2.2 连接界面 7](#_Toc6933)

[2.2.1 软件组成 7](#_Toc23731)

[2.3 调试终端 7](#_Toc20791)

[2.3.1 软件组成 7](#_Toc26948)

[2.3.2 软件实现方案 8](#_Toc6975)

[2.4 控制界面 8](#_Toc17945)

[2.4.1 软件组成 8](#_Toc13725)

[2.4.2 软件实现方案 9](#_Toc16266)

[2.5 波形界面 10](#_Toc24137)

[2.5.1 软件组成 10](#_Toc15018)

[2.5.2 软件实现方案 11](#_Toc28954)

[2.6 协议数据 11](#_Toc19135)

[2.6.1 软件组成 12](#_Toc26596)

[2.6.2 软件实现方案 12](#_Toc28517)

[2.7 数据传输 12](#_Toc25733)

[2.7.1 软件组成 12](#_Toc10480)

[2.7.2 软件实现方案 12](#_Toc23615)

[2.8 接收线程 13](#_Toc15566)

[2.8.1 软件组成 13](#_Toc30491)

[2.8.2 软件实现方案 13](#_Toc4593)

[2.9 底层数据 13](#_Toc2856)

[2.9.1 软件组成 13](#_Toc3465)

[2.9.2 软件实现方案 13](#_Toc6856)

[3 通信协议说明 14](#_Toc23192)

[4 仿真系统 15](#_Toc13941)

[4.1 模型制作 15](#_Toc6951)

[4.1.1 SolidWorks 15](#_Toc26018)

[4.1.2 Blender 17](#_Toc14376)

[4.2 源码说明 21](#_Toc6050)

[5 软件预研 22](#_Toc27968)

[5.1 编程环境 22](#_Toc25782)

[5.2 运行界面 22](#_Toc22544)

[6 软件使用说明 25](#_Toc26811)

[6.1 远程桌面 25](#_Toc20162)

[6.2 CAN连接 26](#_Toc3080)

[6.3 使能关节 26](#_Toc8150)

[6.4 查看记录 27](#_Toc29820)

[6.5 数据导入导出 29](#_Toc11656)

[6.6 控制四足 31](#_Toc26581)

[7 开发注意事项 32](#_Toc10574)

[7.1 GIT管理 32](#_Toc29112)

[7.2 环境配置 33](#_Toc6191)

[7.2.1 Windows 33](#_Toc29467)

[7.2.2 Linux 33](#_Toc20239)

[7.3 CAN异常处理 34](#_Toc11767)

[7.4 异常退出Debug 34](#_Toc20005)

[7.4.1 Windows平台 34](#_Toc14172)

[7.4.2 Linux平台 36](#_Toc23847)

图表目录

[图 1 主控软件流程图 4](#_Toc17305)

[图 2 UML 类图 5](#_Toc11089)

[图 3 总控软件架构图 6](#_Toc8817)

[图 4 总控软件界面 6](#_Toc20463)

[图 5 软件连接界面 7](#_Toc25661)

[图 6 调试终端组成 8](#_Toc4914)

[图 7 波形界面显示图 11](#_Toc20085)

[图 8 插入新零件 16](#_Toc19446)

[图 9 连接重组 17](#_Toc379)

[图 10 3D模型表面映射 18](#_Toc17740)

[图 11 OBJ文件格式参考 19](#_Toc1407)

[图 12 照明模式说明 20](#_Toc21114)

[图 13 仿真软件UML类图 21](#_Toc7840)

[图 14 3D模型最终显示场景 22](#_Toc25667)

[图 15 总控软件界面 23](#_Toc11347)

[图 16 软件连接界面 23](#_Toc872)

[图 17 调试终端组成 23](#_Toc6153)

[图 18 波形界面显示图 24](#_Toc21614)

[图 19 关节在线控制 25](#_Toc5900)

[图 20 关节离线控制 25](#_Toc2210)

[图 21 Windows远程桌面连接 26](#_Toc17322)

[图 22 CAN连接 26](#_Toc24667)

[图 23 初始化所有电机 27](#_Toc20886)

[图 24 初始化所有电机2 27](#_Toc8831)

[图 25 初始化单个电机 27](#_Toc24971)

[图 26 接收异常窗口 28](#_Toc11592)

[图 27 终端调试界面 28](#_Toc14386)

[图 28 关节波形显示界面 29](#_Toc1096)

[图 29 Csv文件头 29](#_Toc23622)

[图 30 导入离线文件失败 30](#_Toc32683)

[图 31 导入离线数据成功 30](#_Toc24566)

[图 32 导出文件成功 30](#_Toc189)

[图 33 顺序/逆序执行 31](#_Toc22923)

[图 34 关节在线控制 32](#_Toc20396)

[图 35 软件工程管理 33](#_Toc20298)

# 方案设计

## 组成

采用可跨平台C++图形用户界面开发的Qt软件框架进行设计，与底层四足采用CAN进行通信，用于四足的运动控制、信息交互等。主要功能包括：

1. 可以收送CAN命令接收返回信息的调试终端；
2. 可以将CAN实时接收到的四足关节数据进行波形展示的窗口，关节数据包括速度、位置、电流；
3. 能够进行单关节绝对位置运动、相对位置运动，速度点击运动的控制窗口；
4. 能够从CSV文件读取运动规划轨迹数据，并自动解析数据匹配命令发送给多关节完成各种四足步态运动。

工作流程如下：

1. 打开软件，进入连接界面，选择CAN通信的波特率，点击连接，如果CAN连接正常，则连接窗口自动关闭，进入主界面；
2. 总控系统获取四足位置、电流和速度数据，将数据解析后存入底层的全局变量中；
3. 如果有打开波形显示界面，则可以实时显示四足的姿态数据波形，该波形界面可以暂停刷新，查看历史波形。
4. 从菜单栏Tool可以打开接收异常检测界面，能够检测接收到的CAN数据是否丢帧和异常。
5. 从菜单栏Tool可以打开终端CAN收发界面，可以给指定的节点重复/单次发送CAN信息，接受CAN帧直接显示。
6. 在菜单栏的Control部分可以打开控制窗口，包括单关节控制、多关节控制、离线控制、遥控控制；

软件流程图如下，其中柔性臂和轮子控制部分尚未实现。



图 1主控软件流程图

## 架构设计

软件采用分层架构，依次分为数据库层、持久层、服务层、业务层和表现层。软件设计分层就是为了在不同的层次上应对这些稳定性不同的需求，在上层设计中响应不稳定的需求，而在下层设计中实现稳定的需求，而在分层后的设计中使得上层依赖下层，而不允许下层依赖上层，则可以使得应对大部分需求变化时对系统的修改最少。其中数据库层GlobalData类用于保存全局相关的数据，持久层DataTransmision类提供访问和修改数据的接口，服务层Package类为不同的业务逻辑提供一些通用的接口，业务层ReceiveWorkerThread定时从CAN接收数据，并依据协议进行解析，将解析后的角度、速度、电流等信息保存至全局变量中，最后是表现层，主要是用户界面，负责视觉、用户互动以及界面逻辑，包括主界面、各种菜单字界面，比如控制界面、波形界面、调试界面等。

UML类图如下：

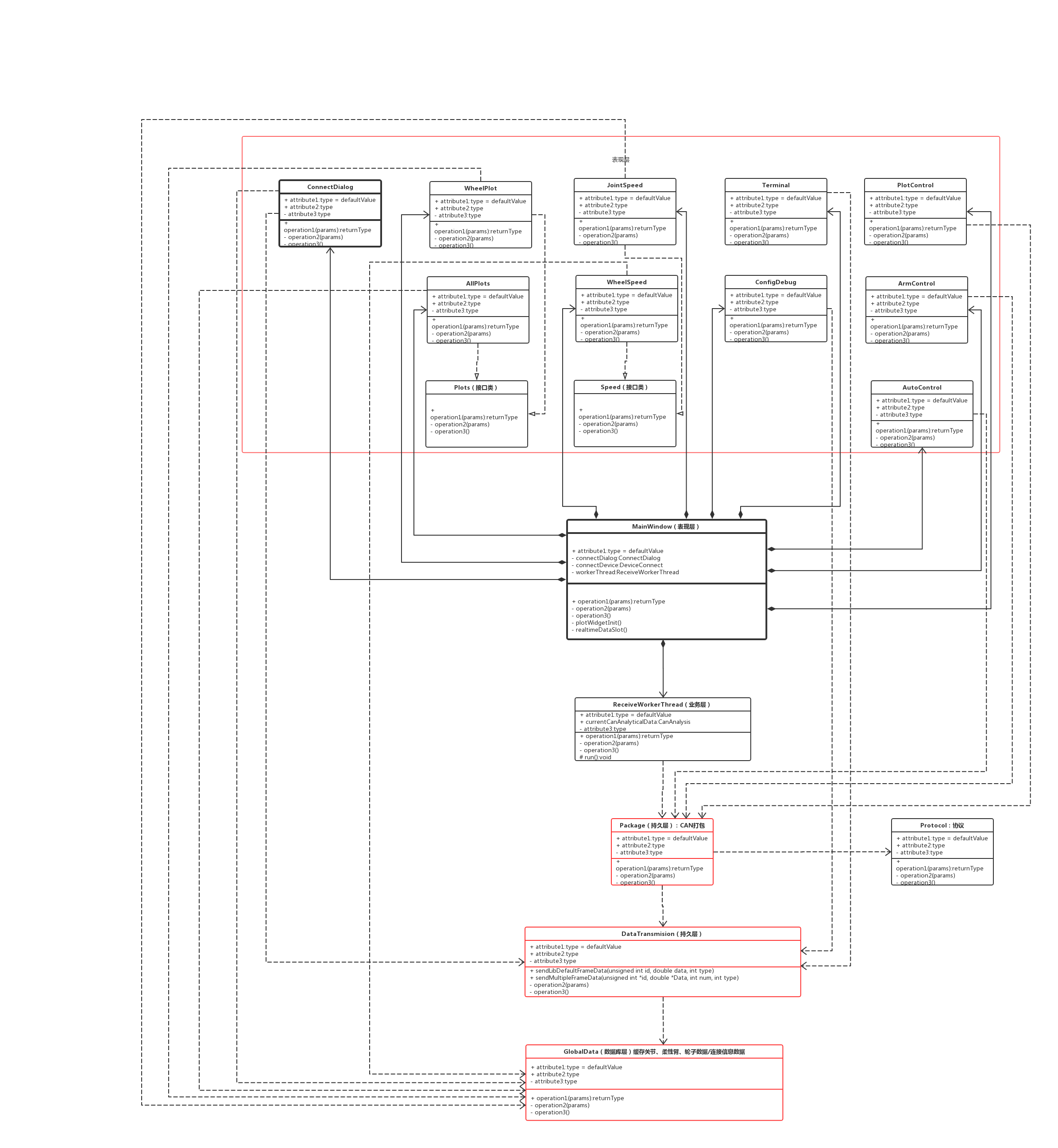


图 2UML 类图

# 详细方案设计

根据UML对整个软件进行详细分类设计，将不同的功能划分到一个独立的类中进行封装，相同功能的部分作为基类处理，使得后续无论是增加新功能还是修改已有的功能都能最小程度的修改原有类，减少无可预知错误的发生，当某个模块发生错误时，能够屏蔽一些可疑的模块让软件正常运行，具体架构涉及到的类如下图 3总控软件架构图。



图 3总控软件架构图

## 主界面

### 软件组成

主界面有菜单栏和背景图片组成，需要使用的功能需要自己从菜单栏进行选择，打开对应的功能窗口，如图 4总控软件界面所示。

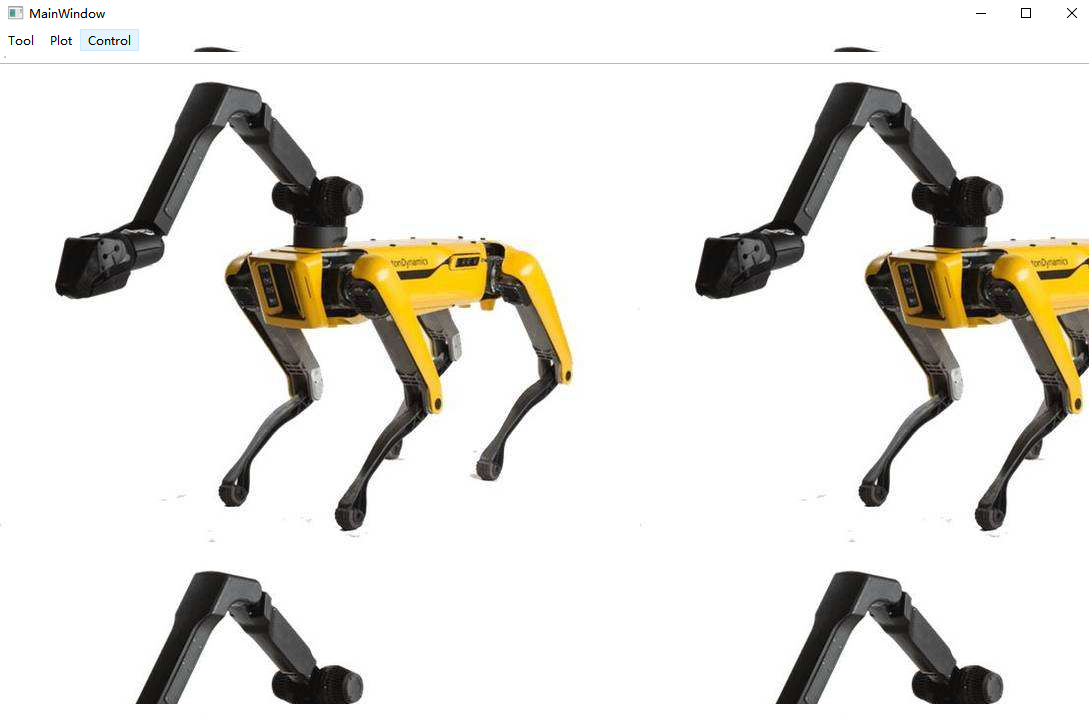


图 4总控软件界面

## 连接界面

### 软件组成

可以对CAN的波特率进行选择，点击OK后会自动进行CAN连接，连接成功后窗口会消失，如图 5软件连接界面所示。

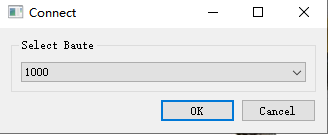


图 5软件连接界面

## 调试终端

### 软件组成

CAN收发调试终端的主要功能是设置CAN发送的数据格式，给指定的CAN节点发送单帧或者重复发送单帧数据，如果勾选了Repeat Send则会进行重复发送，发送的周期根据Period设置的时间决定，如果勾选了Receive Msg，则当收到CAN节点传回来的数据时，会将数据在Received CAN messages进行显示。界面如图 6调试终端组成所示。

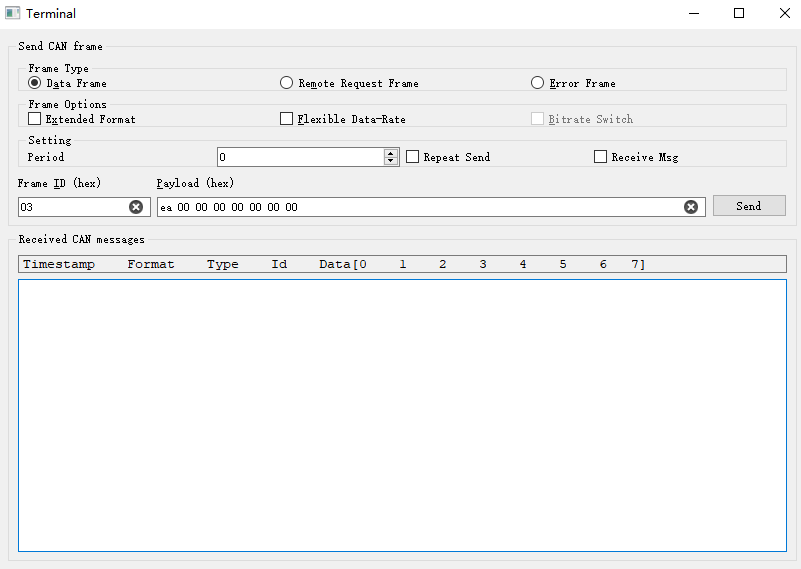


图 6调试终端组成

### 软件实现方案

Termial的UI由Send CAN frame和Received CAN messages两部分组成，其中Send CAN frame使用了ReceiveFrameBox类，Received CAN messages使用了SendFrameBox类，收发模块由两个单独的类去实现。

接收模块ReceiveFrameBox类启动了一个QTimer定时器，定时读取底层接收到的CAN数据，将其在接收窗口显示出来。发送模块SendFrameBox类解读界面输入的数据，调用底层CAN发送接口，将数据一次性或定时发送出去。

## 控制界面

### 软件组成

控制界面主要有单关节控制、多关节控制、离线轨迹控制和遥控控制界面组成，可以从菜单栏点击Control进行选择。如果只是对单个关节（比如1-12节点）进行初始化、急停、位置、点动等控制，选择单关节SingleJointControl即可，可以用来测试关节功能是否正常。多关节控制界面就不需要选择不同的节点，所有节点均在同一个界面显示。离线轨迹控制用来读取仿真的离线数据，将数据导入表格中，再依次周期性的打包成CAN控制命令，发送给四足，使得四足能够按照设定的步态运行。

### 软件实现方案

关节控制模块继承于Controls基类，该基类用于实现控制界面的基本操作，包括界面的object初始化绑定、信号与槽的初始化、各种事件处理函数的实现，由于这些为公共接口，无需在子类中再次实现。主要的成员函数包括：

1. initUiObject：判断对象数组是否有赋值，如果没有则返回false。
2. controlWidgetInit：界面滑条、方框设定最大最小值等。
3. initConnection：信号与槽的初始化
4. forwardReversalPushbtnClicked：正转反转函数处理
5. setPosButtonClicked/setSpeedButtonClicked/setSpeedButtonReleased：设置按钮处理函数，速度为点动模式，按下按钮才会动，松开运动停止

JointControl：继承类，继承于Controls，只需要完成界面对象的赋值，该值从UI中获取，然后调用基类的一系列初始化方法就能够完成界面的搭建，比如initConnection、controlWidgetInit，依照设计方法里面的模板方法，将需要子类更改的方法使用虚函数提供接口出来，基类提供固定的初始化框架即可，可以防止子类使用过程中出现初始化步骤上的错误。

离线轨迹控制窗口采用了TableView的形式载入CSV文件的数据，该TableView继承自QTableView，将其封装成了一个独立的类ControlTableView，功能包括表的初始化、表新增行、更改表行属性、导出表数据至CSV、从CSV导入数据至表中、表中按钮处理函数等等，在类OfflineControl里面调用ControlTableView，实际对其进行控制。

ControlTableView类的重点函数为：

1. tableClickButton，表中widget对应的事件函数，包括上移、下移、增加、删除、运行事件，其中运行事件runFunc会读取当前的命令类型，如位置运动、速度运动等，然后将表格该行的数据打包通过CAN发送给四足。
2. execSeqEvent，顺序或逆序执行表格列表的命令，也可以定时执行。运行前会进行数据异常检查，如果表格内数据和机器人当前的位置相差POS\_LIMIT\_VALUE，那么顺序或逆序执行将会终止。整个顺序或逆序执行的过程由定时器taskTimer进行控制，中途可以执行暂停操作。
3. exportToCsv/importCsv，将表格数据导出为csv文件和将csv文件的数据导入到表格界面中，导入文件是需要判断是否为空文件，以及格式文件的检查，另外，导入后会生成一些按钮，会阻塞线程，需要有防止页面假死的措施，导出csv文件时，将数据按照指定的格式导出，后期如果修改csv文件格式，需要对该函数进行更改。

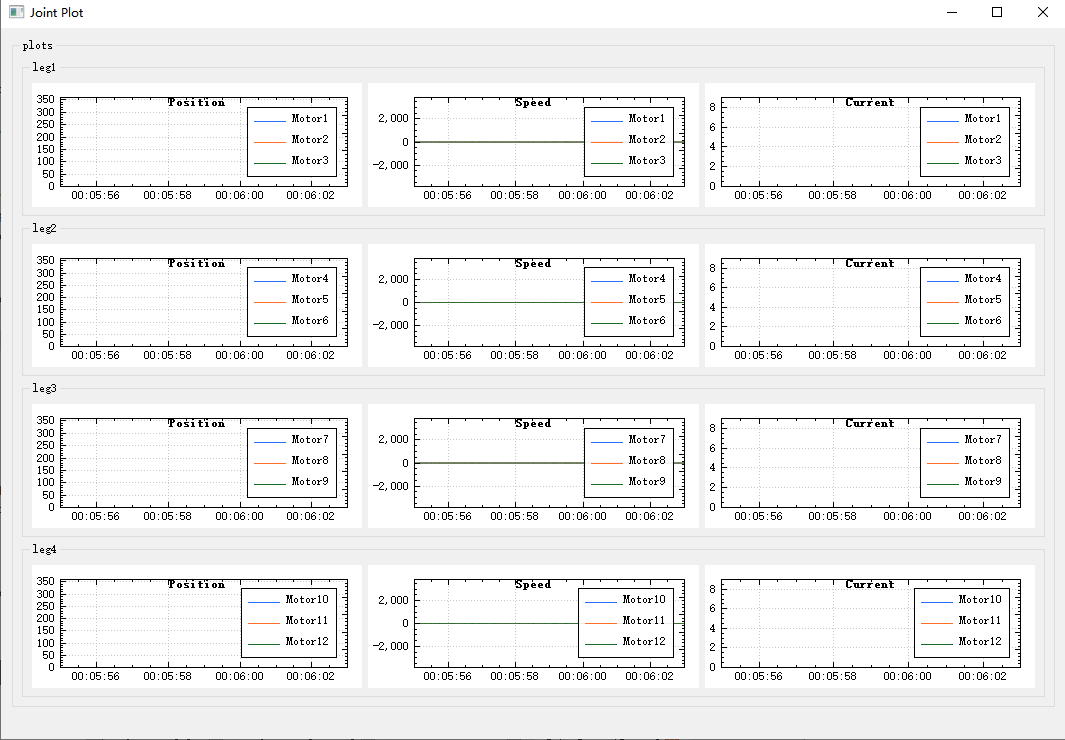
另外还有一个自动控制的类AutoControl，目前还处于测试阶段，使用流程为：

1. 实例化类。
2. 调用moveBodySet或者moveLegSet函数设置好目标位置、关节Id、运动速度，或者调用autoRunSet启动自动运行任务。其中moveBodySet或者moveLegSet函数一般用来进行遥控控制，autoRunSet用来自动控制。
3. 调用start启动运行。
4. 调用stop停止运行。

## 波形界面

### 软件组成

该部分主要由两个界面组成，包括关节数据波形显示部分、关节速度测量显示部分。关节数据波形会将CAN接收到的各个关节的数据以波形的形式展示出来，该波形界面可以暂停、回看以及查看波形某个位置的具体数据等。关节速度测量可以选择相对应的节点，实时获取该节点的速度信息，并将该速度转换为rpm的单位以数码管的形式展示，方便做大屏的显示。具体界面效果如图 7波形界面显示图：



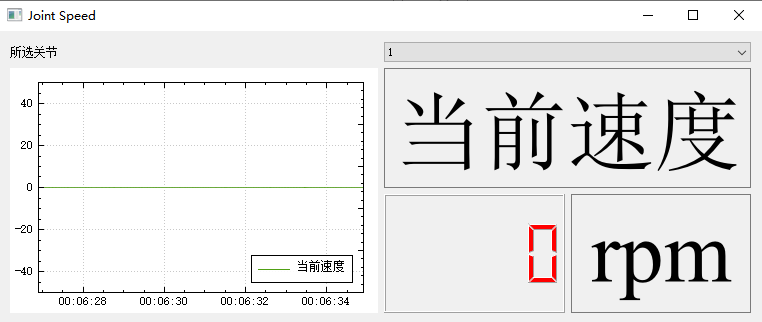


图 7波形界面显示图

### 软件实现方案

两个界面的波形显示部分均继承自Plots类，这是波形显示的模板类，具体实现得其继承的子类去实现，提供了相关函数接口以及虚拟接口。相关的函数接口包括：

1. plotPartWidget：绘制波形框架，包括x、y坐标轴范围设置等。
2. setPlotPen：设置波形线的名称和颜色。
3. realtimeDataPlot：定时绘制函数，绘制周期由定时器plotDataTimer设定，该函数不对外界展示，会调用setLinesReplot虚函数接口，该虚函数接口具体在子类进行实现。
4. addDataToUi：刷新界面数据，调用widgetAddData函数。
5. setLinesPausePlot：波形暂停，调用setLinePausePlot函数。
6. setLinesReplot：波形重新开始，调用setLineReplot函数。

JointPlot类：显示所有关节的波形，继承于Plots类，只需要实现initWidget，addDataToUi，setLinesReplot，setLinesPausePlot函数即可。初始化界面控件时调用基类的plotPartWidget、setPlotPen接口，将控件指针传递会基类去初始化控件。

JointSpeedMeas类：采集当前关节速度，继承于Plots类，只需要实现initWidget，addDataToUi，setLinesReplot，setLinesPausePlot函数即可。初始化界面控件时调用基类的plotPartWidget、setPlotPen接口，将控件指针传递会基类去初始化控件。

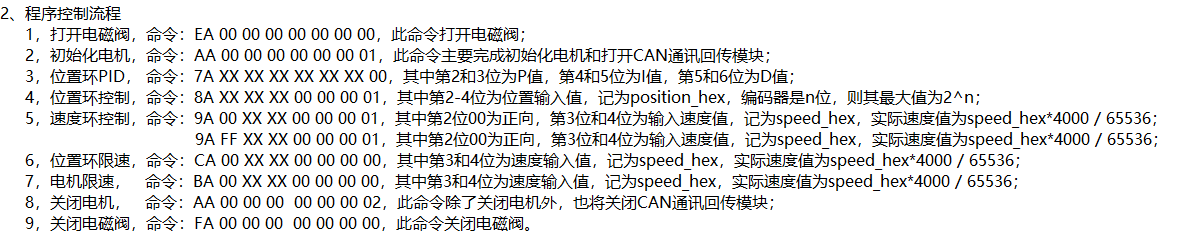
## 协议数据

### 软件组成

协议数据部分的主要作用为将从四足接收到的CAN数据解析成相对应的位置、电流、速度等信息，同时，能够将目标位置、速度等信息封装成CAN数据包，这是一个与CAN数据进行解封装的功能。

### 软件实现方案

具体实现包括两个类，Protocol协议接口类，供调用协议用来做具体事务的类使用，后续如果需要更改协议，只需要改动这里即可，有关命令如下，



具体的接口函数有，getRawData：根据ID将某个关节的数据放到一维数组的某个位置，parsePos：将原始数据解析成位置信息，packPos：将位置信息封装成字节数组，用于CAN信息封装。

另外一个类是Package类，这个类会调用Protocol类的方法。比较重要的几个接口函数为：unpackOperate：解读从CAN卡读到的数据，并更加电机协议换算成电机电流、速度、角度信息。packOperate：以速度、角度、位置信息将数据进行打包，然后再调用底层发送接口传递给CAN。packOperateMulti：可以同时打包多帧数据，直接调用底层发送多帧的CAN接口，减少各帧数据发送的延时。

## 数据传输

### 软件组成

数据传输层，实现底层CAN数据的连接、收发功能，提供相对应的传输接口供上层调用，上层无需关注该传输层的实现细节，比如使用的是何种类型的CAN卡硬件，各个CAN卡的数据格式都是统一成了一样的接口，按照了面向接口编程的设计思想，让上层调用统一的接口即可实现对传输层的控制。

### 软件实现方案

DataTransmision类作为持久层，用于提供发送/接收CAN数据、读取CAN数据、读写串口数据接口等，主要函数包括：

CANTransmit：发送CAN数据函数接口，区分不同CAN卡，其他类可以直接调用，不用管里边是怎么实现的。CANTransmitMulti：同时发送多帧数据的底层CAN数据接口，在这里对不同的CAN卡做区分，上层无需理会这里是怎么实现的。Receive：接收CAN数据函数接口，自动区分不同的CAN卡，读取到的数据以统一的接口开放给外部使用。

## 接收线程

### 软件组成

接收线程指的是为接收CAN各个节点的数据而开辟的一个线程，与主线程并行运行，保证接收CAN节点数据的过程持续发生，能够实时采集到最新的四足底层CAN节点数据。

### 软件实现方案

在Qt软件中，采用了继承自QThread类的方法来实现线程，在子类中对run虚函数进行重写，在run函数中实现循环语句，调用Package类的unpackOperate方法，持续地解读从CAN卡读到的数据（通过DataTransmission类的CANReceive方法读取），并根据电机协议换算成电机电流、速度、角度信息（通过Protocol类的parsePos、parseSpeed、parseCurrent等方法换算），将这些解读后的数据存入底层全局变量global中供其余模块调用。

实例化该线程对象并调用start方法即可开启线程运行，该步骤在MainWindow类中实现。

在unpackOperate函数中，还对节点的状态值nodeStatus进行了存储，在ReceiveError类中会调用该状态值，判断是否有错误信息，并将错误信息更新到界面显示。

## 底层数据

### 软件组成

底层数据主要包括一些共享的数据，如当前的CAN原始数据、CAN解读换算出来的数据，CAN节点状态值以及一些宏定义配置变量。由于这些数据会经常被上层模块访问，所以写在一个公共的模块中。

### 软件实现方案

实现方案参考GlobalData类，采用单例模式，可以实现私有的构造函数，并在构造函数中进行一些初始化，由于采用的是单例模式里面的懒汉模式，所以需要注意多线程问题，新建实例前需要双重判断互斥锁，程序退出时还需要释放内存空间，采用静态局部变量自动析构的方式进行，最后将实例化后的指针定义为一个宏定义，#define global GlobalData::getInstance()，方便外部进行引用。

# 通信协议说明

主从命令一般由子系统发出，主要用于传输关节运动数据，实时控制四足关节运动。以速度环控制的协议进行举例，数据位总共占据8个字节，每个字节依次说明如表 1末端速度控制模式主从命令格式。

表 1末端速度控制模式主从命令格式

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 参数 | MOD | DIR | D1 | D2 | D3 | D4 | D5 | D6 |
| 字节数 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

* MOD：模式标示，占1个字节，用代码9A表示。
* DIR：方向位，占1个字节，用代码00/FF表示，00代表正向，FF代表方向。
* D1-D4：第3位和4位为输入速度值，记为speed\_hex，，实际速度值为speed\_hex\*4000 / 65536。

其余控制协议类似，大概说明如下：

1. 打开电磁阀，命令：EA 00 00 00 00 00 00 00，此命令打开电磁阀；
2. 初始化电机，命令：AA 00 00 00 00 00 00 01，此命令主要完成初始化电机和打开CAN通讯回传模块；
3. 位置环PID， 命令：7A XX XX XX XX XX XX 00，其中第2和3位为P值，第4和5位为I值，第5和6位为D值；
4. 位置环控制，命令：8A XX XX XX 00 00 00 01，其中第2-4位为位置输入值，记为position\_hex，编码器是n位，则其最大值为2^n；
5. 位置环限速，命令：CA 00 XX XX 00 00 00 00，其中第3和4位为速度输入值，记为speed\_hex，实际速度值为speed\_hex\*4000 / 65536；
6. 电机限速，命令：BA 00 XX XX 00 00 00 00，其中第3和4位为速度输入值，记为speed\_hex，实际速度值为speed\_hex\*4000 / 65536；
7. 编码器校正，命令：DA 00 00 00 00 00 00 01，此命令实现增量式编码器的重新校正，即重新复位电角度；
8. 清除状态，命令：6A 00 00 00 00 00 00 01， 此命令实现状态位报错后，恢复软件控制状态机为初始状态；
9. 关闭电机，命令：AA 00 00 00 00 00 00 02，此命令除了关闭电机外，也将关闭CAN通讯回传模块；
10. 关闭电磁阀，命令：FA 00 00 00 00 00 00 00，此命令关闭电磁阀；
11. 返回值定义：00 00 00 00 00 00 00 00，第1字节：电机工作状态位；第2-4字节：电机当前绝对位置，从高位到低位；第5-6字节：电机实时电流值，从高位到低位；第7-8位：电机实时速度值，从高位到低位。

# 仿真系统

总控软件作为姿态数据TCP服务器，向客户端定时发送姿态数据，这里的客户端指的是仿真系统软件，该软件接收TCP网络数据后解析出各个关节的角度，然后驱动整个四足3D模型在场景里面的运动。

连接时需要注意的是服务器IP为192.168.137.134，端口为2019。可以在源码netclient.cpp中修改ADDRESS、PORT宏定义。

由于缺乏简化版的模型文件，目前仿真系统的3D模型尚未导入，后续需要自己导入，并修改源码才能正确驱动仿真系统。

## 模型制作

由于基于Qt的3D模型仿真软件需要读取obj/mtl类型的模型文件，而Solidworks给出的文件为装配体，需要先使用SolidWoks软件将其转化为单独的stl零件。然后采用Blender软件将这些stl零件制作成obj/mtl模型文件。

### SolidWorks

软件介绍：SolidWorks公司开发的运行在微软Windows平台下的三维机械CAD软件。由于软件涉及到了STL文件，简单说明一下，它是由3D Systems软件公司创立、原本用于立体光刻计算机辅助设计软件的文件格式。STL文件使用三维笛卡尔坐标系统，通过单位法线和三角形的顶点（由右手定则排序）描述原始的非结构化三角形表面。

使用Solidworks需要注意的一个操作，怎么将装配体合并为一个STL零件？

1. 在SolidWorks中创建好一个装配体零件，装配体零件中可以包含镜像零部件
2. 面板中点击插入零部件——新零件，或从插入菜单中选择插入——零部件——新零件。



图 8 插入新零件

1. 此时光标会变成指针侧下方有个绿色箭头，此时不要乱点，对准装配体的一个面点击，然后新零件会处于草图编辑状态，此时不要绘制任何草图，直接点击右上方图标退出草图，新零件处于特征编辑状态。
2. 不要退出特征编辑状态，选择菜单中的插入——特征——连接重组，并选择需要组合的装配体，点击确定，最后退出特征编辑状态。



图 9 连接重组

1. 打开新零件，可以观察到装配体变成了一个整体零件了。

### Blender

软件介绍：Blender是一套专业的，自由及开放源代码的三维计算机图形软件。它主要是用来制作obj/mtl文件，可以从以下几点了解obj/mtl文件。

1. OBJ文件格式是一种简单的开放格式。它在CAD软件中具有广泛的输出和导入支持。
2. OBJ文件格式是一种文本文件格式，可以在文本编辑器中编辑OBJ文件
3. OBJ文件也将更加轻巧，体积小
4. 应用多色3D打印更为流行，因为标准3D打印格式STL不支持颜色和纹理信息。OBJ文件格式允许您以称为材料模板库（MTL）格式的伴随文件格式存储颜色和纹理信息，将这两个文件放在一起可以渲染多色纹理模型。
5. MTL可以定义材料属性，如环境颜色，漫反射颜色，镜面反射颜色，透明度等。MTL格式也支持纹理贴图，这是一种指定颜色和纹理的更方便的方法。

在纹理映射中，3D模型表面（或多边形网格）中的每个点都映射到二维图像

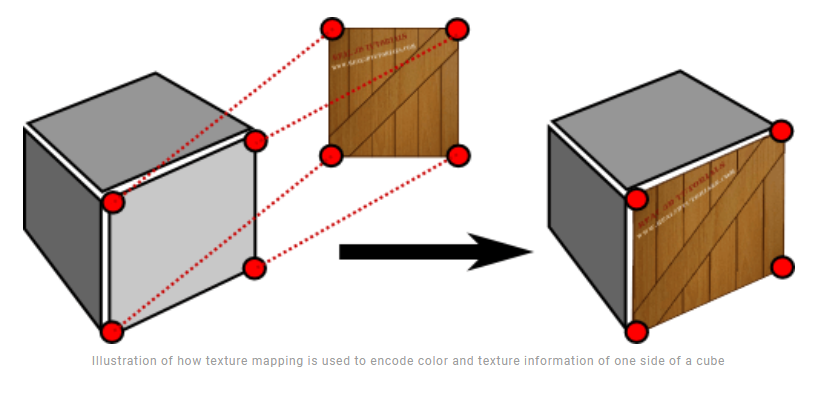


图 10 3D模型表面映射

Obj/mtl文件有它的文本格式，通过不同的关键字进行解读。

1. 顶点v x y z：v命令通过其三个笛卡尔坐标x，y和z指定顶点。顶点会自动分配一个名称，具体取决于在文件中找到它的顺序。文件中的第一个顶点名为'1'，第二个顶点为'2'，第三个顶点为'3'，依此类推。
2. 顶点法线vn x y z：vn是指定表面的法向量，x，y和z是法向量的分量。此法线向量尚未与任何顶点相关联，将使用另一个名为f命令的命令将其与顶点相关联。在许多文件中省略了顶点法线命令，因为我们将使用f命令将顶点分组为多边形面，可以根据顶点坐标和顶点出现的顺序自动确定法线向量。
3. 顶点纹理vt u v [w]：顶点纹理命令指定纹理贴图中的一个点。U和V是纹理贴图中的X和Y坐标。
4. 面f v1[/vt1][/vn1] v2[/vt2][/vn2] v3[/vt3][/vn3]：face命令可能是最重要的命令。它指定由此行之前的顶点构成的多边形面。要引用顶点，只需遵循顶点的隐式编号系统，这个编号称为索引。

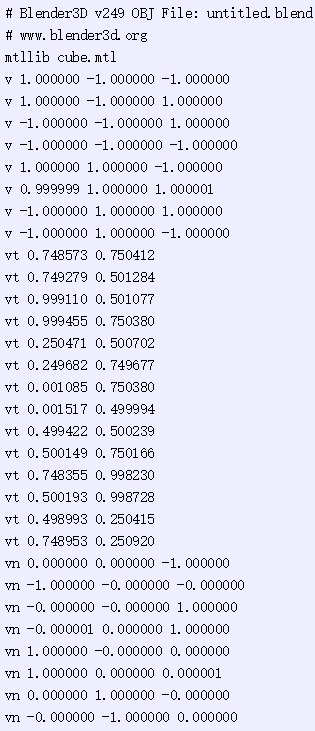


图 11 OBJ文件格式参考

例如f 8/11/7 7/12/7 6/10/7：8/11/7描述了三角形的第一个顶点，7/12/7描述了三角形的第二个顶点，6/10/7描述了三角形的第三个顶点。

对于第一个顶点，8指向要用的顶点。此例中是-1.000000 1.000000 -1.000000（索引从1开始，和C++中从0开始不同）。11指向要用的纹理坐标。此例中是0.748355 0.998230。7指向要用的法线。此例中是0.000000 1.000000 -0.000000。

若几个顶点共用同一个坐标，索引就显得很方便，文件中只需保存一个”V”，可以多次引用，节省了存储空间。

1. 使用材质命令usemtl name：指定一个材质来使用，接下来的所有f命令都采用该材质，除非重新使用该命令。材质命名为newmtl name。
2. 指定材质文件mtllib name：由于材料是在MTL文件中指定的，因此mtlib命令必须位于usemtl命令之前。
3. Ka R G B：此命令定义材质的环境颜色ambient color。R，G和B是红色，绿色和蓝色通道。每个都可以取0到1之间的值。
4. Kd R G B：Kd是漫反射颜色diffuse color。
5. d value：此命令指定材质的透明度。值可以是0（完全透明）到1（完全不透明）
6. illum value：此命令指定照明模式。

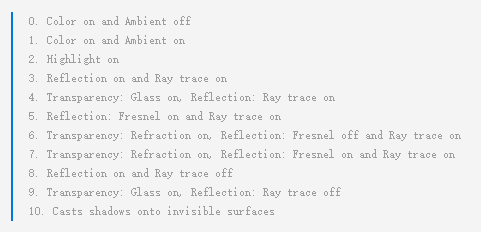


图 12 照明模式说明

使用Blender制作obj/mtl步骤总结如下

1. 导入通过SolidWorks软件做好的stl文件
2. 设置尺寸大小，建议统一放大缩小相同倍数
3. 设置颜色，跟实物的颜色类似即可
4. 设置坐标原点为零件的正中心
5. 导出obj/mtl文件

## 源码说明

工程名字为：RobotSimSystem，整个工程架构参考图 13仿真软件UML类图。

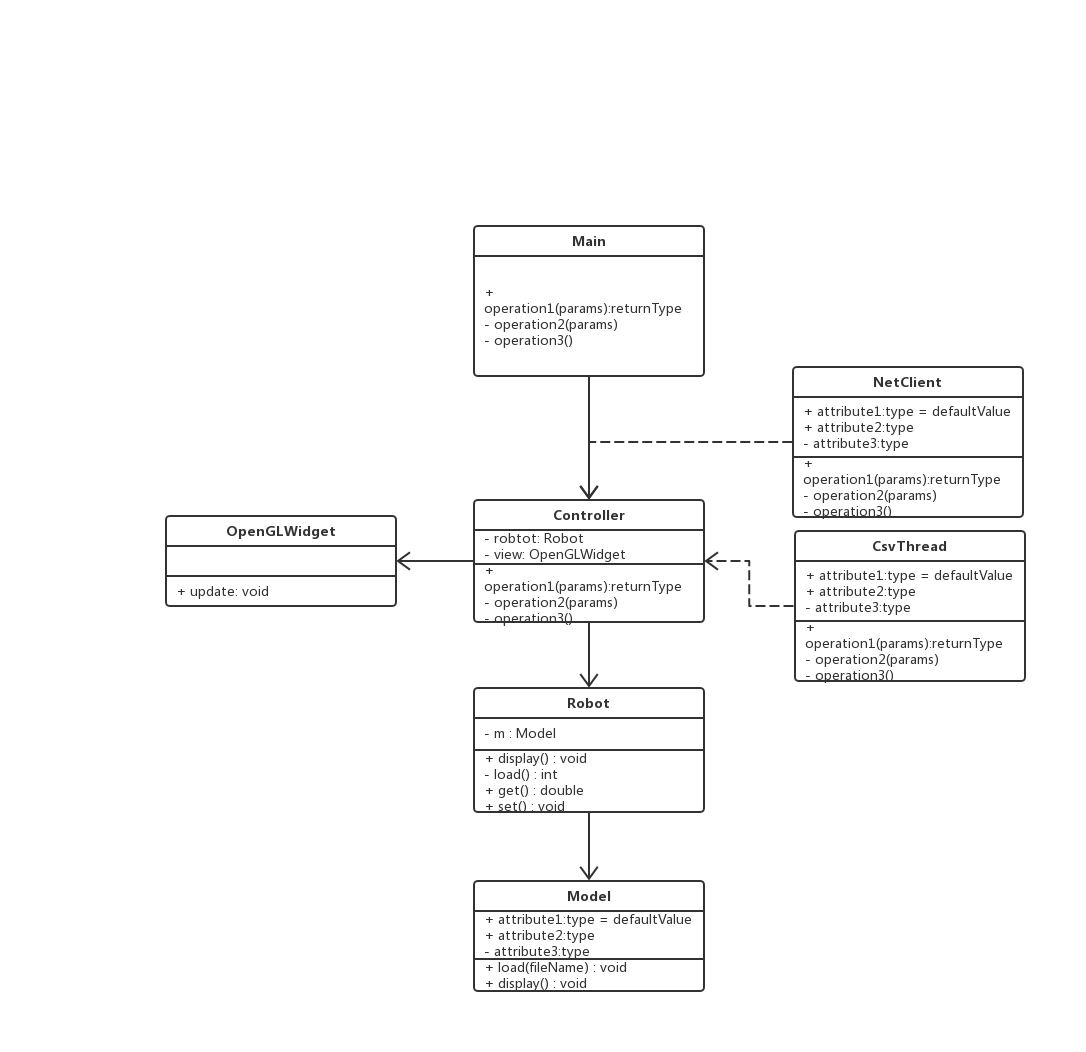


图 13仿真软件UML类图

以上模型文件读取的文件解析过程可以参考Model类方法load、load\_materials，模型显示参考类方法display。

如果有修改好四足模型文件，需要打开robot.h里面的DOG宏，这时会自动从可执行目录下查找roof.obj, djoint1.obj, djoint2.obj, djoint3.obj以及对应的mtl文件，其中roof为底座，joint名字的代表关节，如果能够找到则会正确加载并显示。

这个导入实现过程参考robot.cpp文件的load方法，显示过程参考robot.cpp文件的display方法。由于导入后都是独立的零件，所以需要在display方法里面进行重新拼装，拼装的过程就是改变glTranslatef/glRotatef函数的值，直到拼装出完整的四足为止。

另外，网络驱动的过程参考netclient.cpp文件的socketRead方法，解析出关节数据后再去驱动3D模型各个关节的转动，TCP字节流协议以start开始，end结束，中间12个关节的位置数据。

最终3D模型及场景类似效果显示如下图：

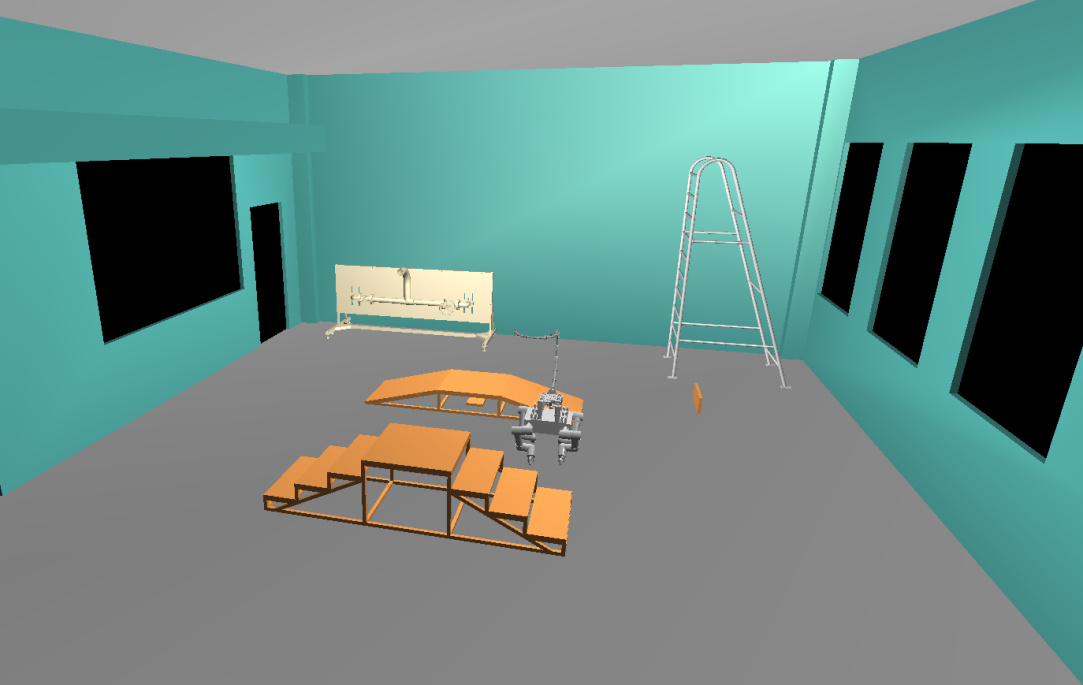


图 14 3D模型最终显示场景

# 软件预研

## 编程环境

1. Qt：5.12.0
2. 编译器：MinGw 64-bit
3. 操作系统：Ubuntu16.04/Windows10
4. 服务器IP：192.168.137.134，Port：2019
5. 波形插件：QCustomPlot 2.0.1，下载地址[https://www.qcustomplot.com/index.php/downloadhttps://www.qcustomplot.com/index.php/download](https://www.qcustomplot.com/index.php/download)

## 运行界面

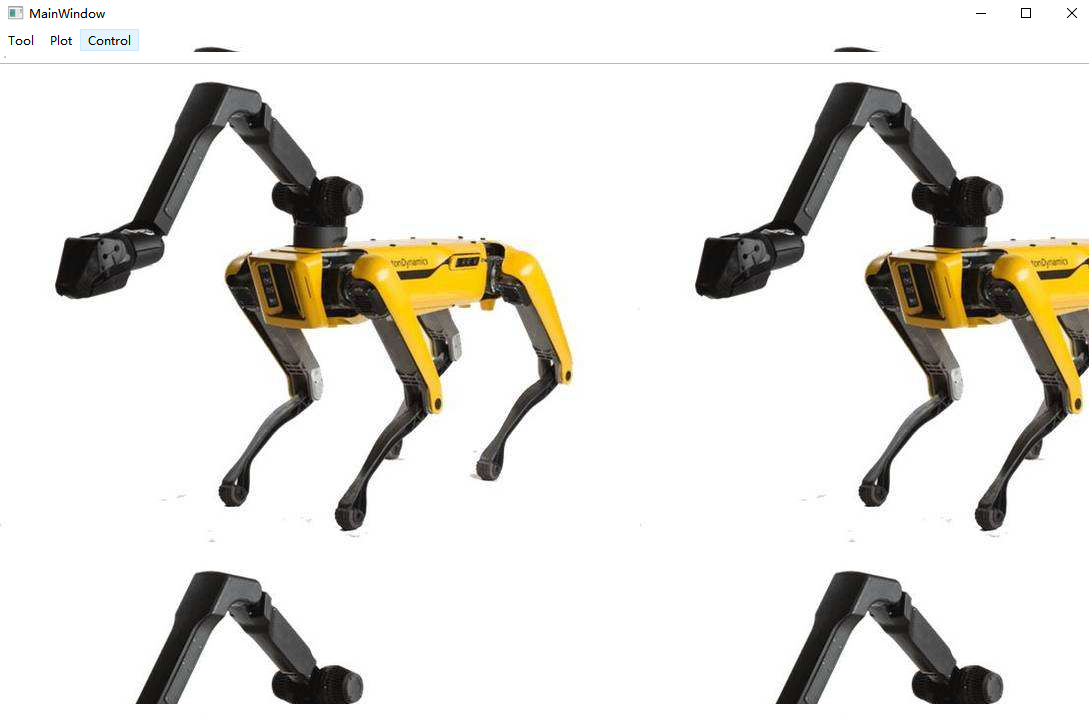


图 15总控软件界面

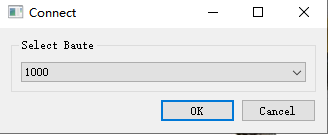


图 16软件连接界面

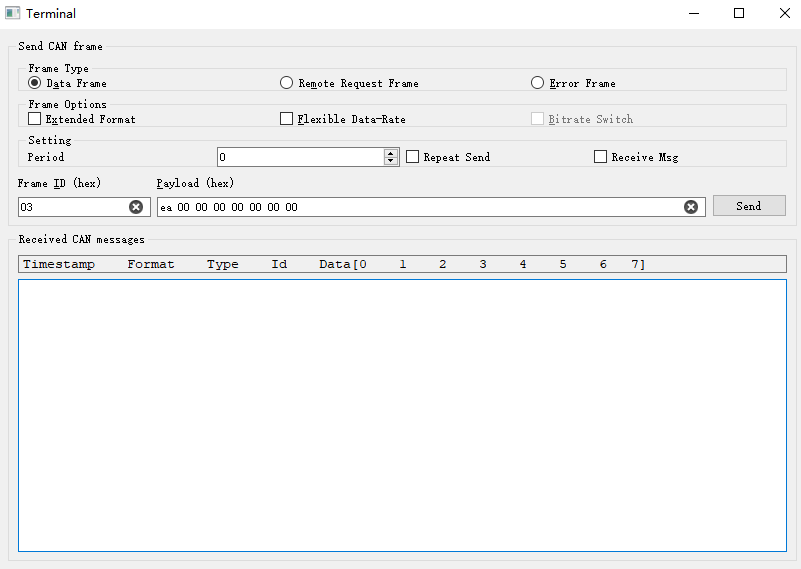
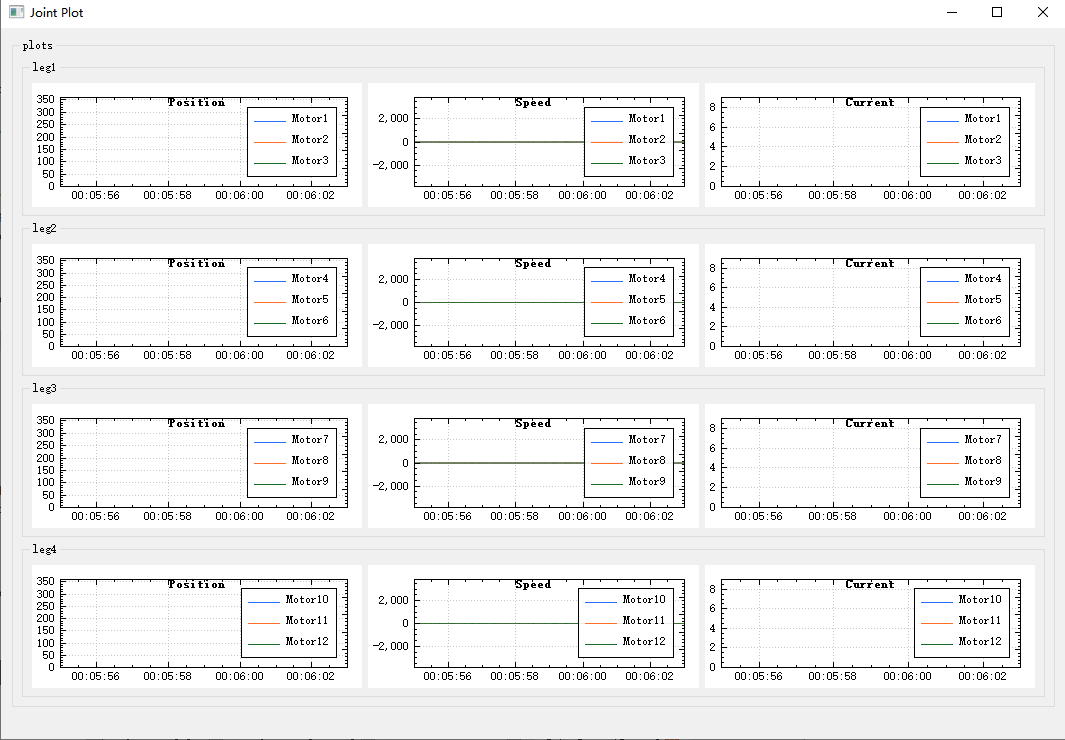


图 17调试终端组成



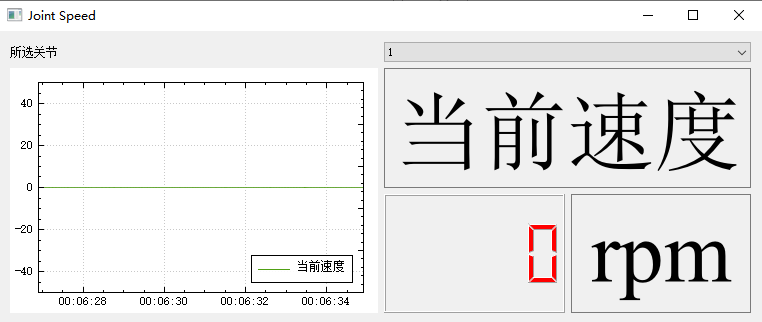


图 18波形界面显示图

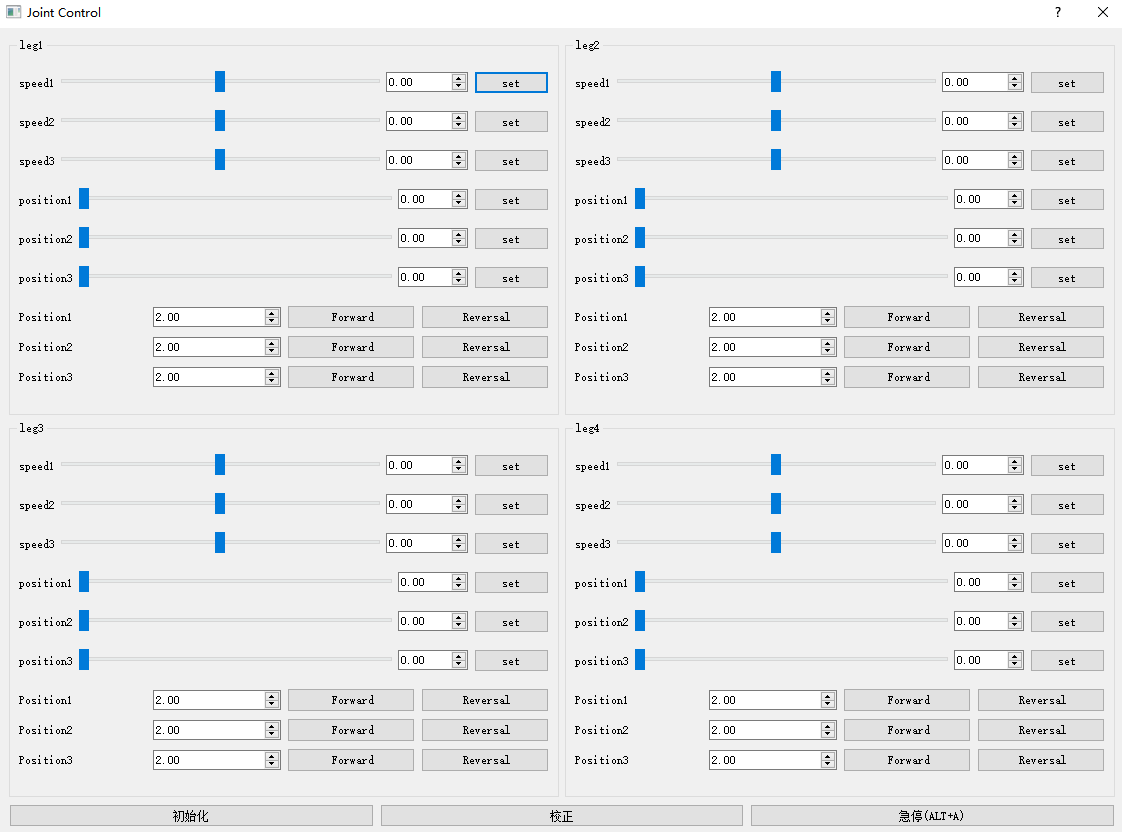


图 19关节在线控制

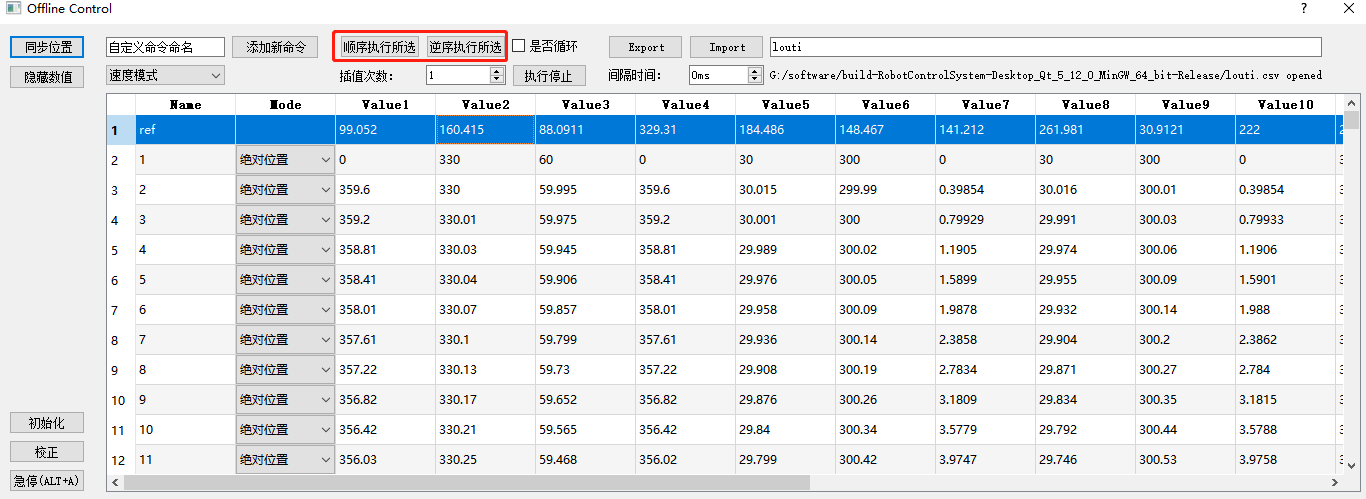


图 20关节离线控制

# 软件使用说明

## 远程桌面

该步骤在调试阶段可以省略，在后续无线控制中才开启使用。远程桌面指的是通过Windows操作系统开启远程桌面连接NUC的桌面，进而实现通过Windows来控制NUC打开总控软件来控制四足的运行。连接的IP地址为192.168.137.134，用户名为robot，密码为robot2019，注意需要保证NUC和Windows电脑保持在同一个网段，如图 20Windows远程桌面连接。

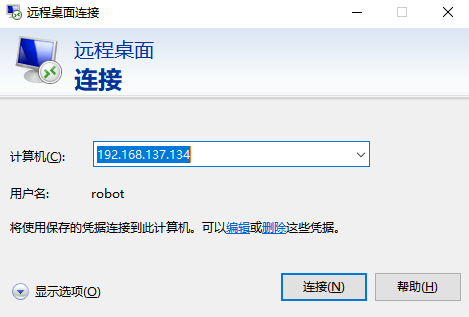


图 21Windows远程桌面连接

## CAN连接

打开软件后会弹出CAN连接窗口，波特率设置为1000kb/s，电脑插上USB-CAN卡，装好对应的驱动后，点击OK按钮即可正常连接，如图 8CAN连接，连接成功后该连接窗口会自动关闭，否则点击OK会没有反应。

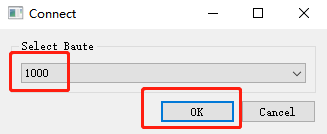


图 22CAN连接

如有无法正常连接的情况，需要检查CAN卡是否正常插入，并安装好驱动，同时可执行文件目录下含有相对应的动态库文件，可以将工程目录libs里面对应的dll文件复制过去。

## 使能关节

使能关节指的是给四足给个关节发送使能信号，让关节开始正常工作，同时往上位机发送电流、位置、速度等信息。上位机只有接收到这些数据才能继续后续的步骤，初始化成功后可以听到抱闸打开的声音，初始化一次可能不行，建议隔几秒再点击一下初始化，直到抱闸完全打开。

可以初始化单个电机，也可以一次性初始化所有电机，具体操作方法如下所示。

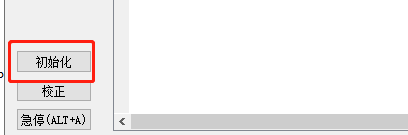
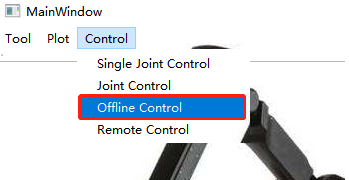


图 23初始化所有电机

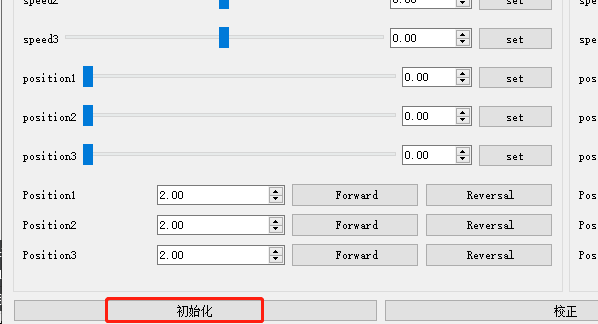
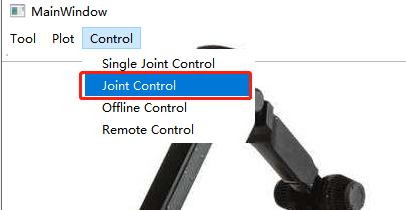


图 24初始化所有电机2

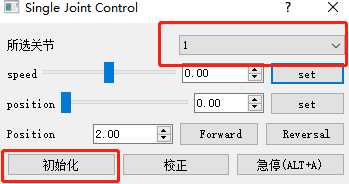
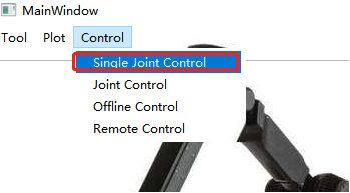
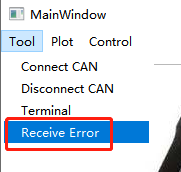


图 25初始化单个电机

## 查看记录

初始化完成后，可以打开接收异常窗口查看各个关节的数据是否有丢失的情况，如有丢失，丢失的最长时间是多少，还可以查看节点接收状态信息，如状态位出现异常，显示的数字会标红，括号内的数字会加一处理。如图 25接收异常窗口，即为正常的接收情况，正常的状态值为06。



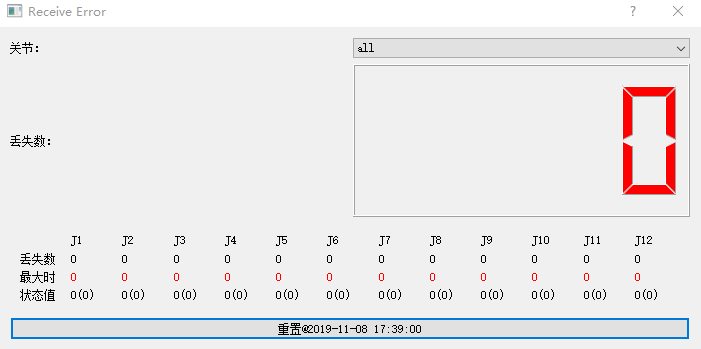
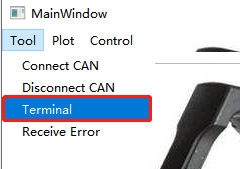


图 26接收异常窗口

另外，可以打开终端调试窗口，查看实际的CAN数据接收结果，从Tool菜单栏点击Terminal可以打开该窗口，勾选Receive Msg即可接收到实时的CAN信息。界面如图 26终端调试界面



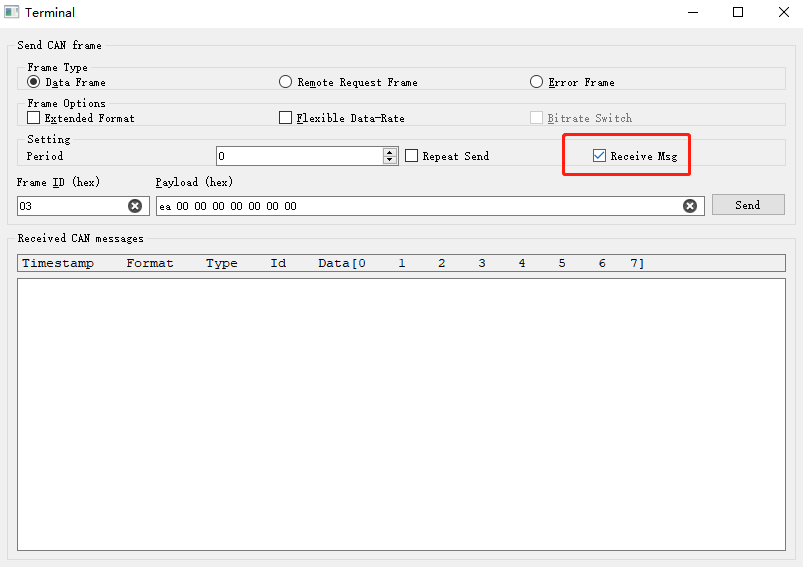


图 27终端调试界面

还可以打开波形界面查看各个关节的数据随时间的变化波形，能够更加方便的对历史数据进行查看，双击波形可以暂停移动，点击波形上的某一个位置可以显示具体的值。如图 27关节波形显示界面

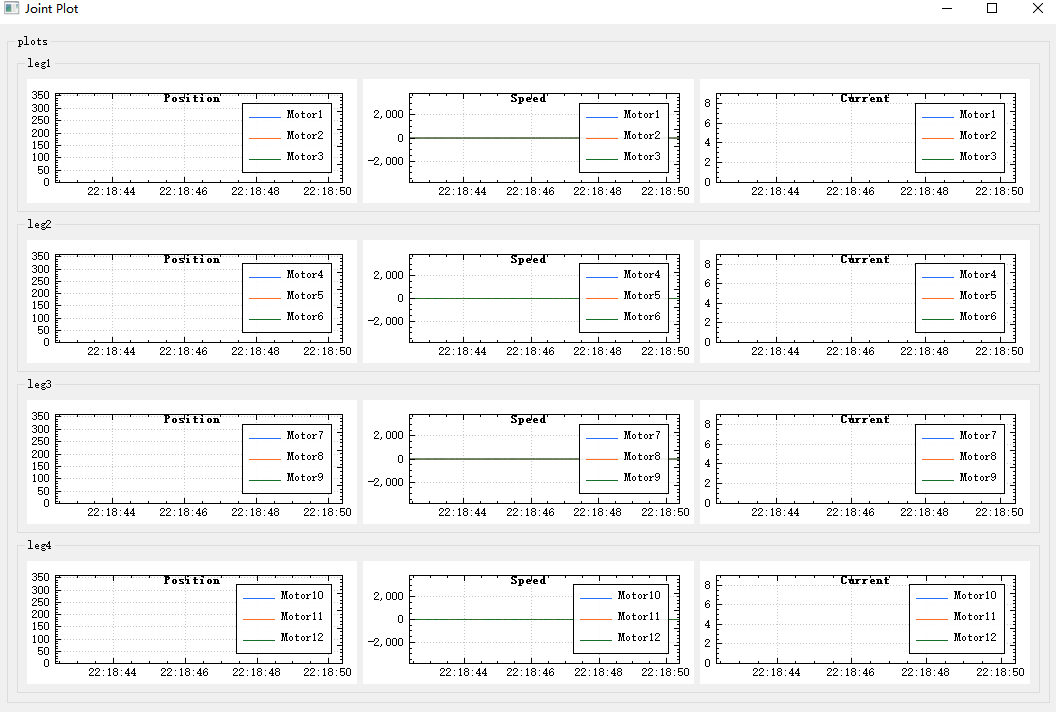


图 28关节波形显示界面

## 数据导入导出

这里的数据指的是四足步态运动的离线轨迹数据，由步态算法仿真后给出，将这些数据按总控软件指定的格式生成csv文件，文件头如图 28Csv文件头所示，第一行为表头名称，第二行代表各个关节的参考位置，通过将四足各个关节转到需要设定的零点后读取此时的关节角得到，后续的行数字代表各个关节相对于ref参考位置的变化量，总控软件会读取此时的变化量和参考量来计算目标位置，目标位置范围为0~359度。

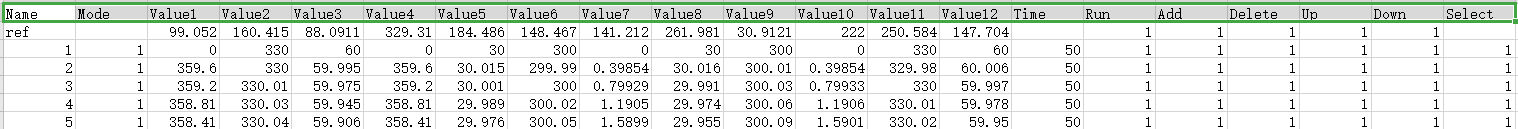


图 29Csv文件头

从Offline Control窗口输入文件名，再点击Import即可导入离线数据，如未找到导入文件，则会提示导入失败，需要将文件置于提示的目录下，如图 29导入离线文件失败，Linux环境下csv文件需要置于/home/csv文件夹下。

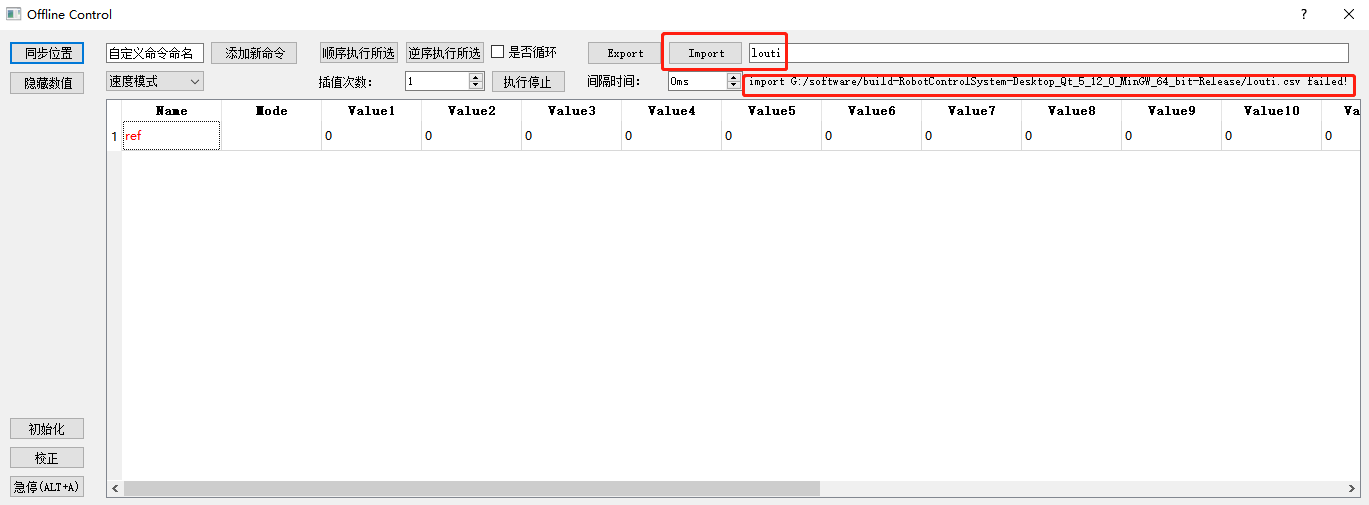


图 30导入离线文件失败

导入数据成功后，会显示opened，表格内容会自动填充，Name列代表的是该行数据的名称，可以随意更改，方便区分，默认按照数字编排，运动模式有绝对运动、相对运动和速度运动三种，一般步态运动的离线控制均采用绝对位置模式，导入成功界面如图 30导入离线数据成功。

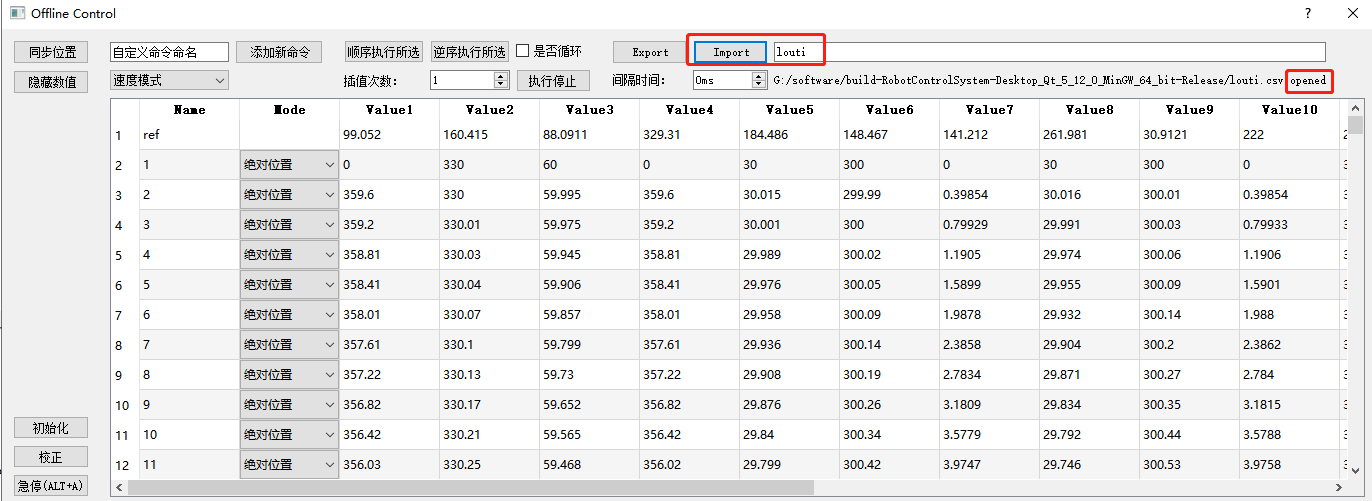


图 31导入离线数据成功

如果需要再次对导入的文件进行修改，比如修改每一行命令的名称，或者单行的数据进行微调整，修改完成后，可以输入需要另存为的文件名，然后点击Export，文件就会自动保存在提示的目录下，如图 31导出文件成功。

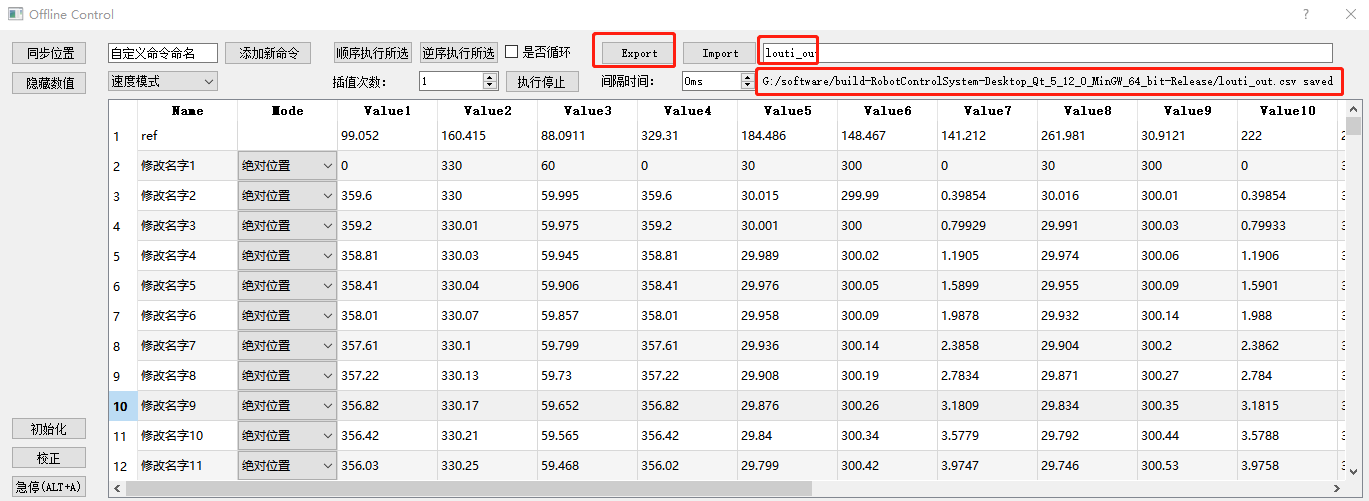


图 32导出文件成功

## 控制四足

导入离线数据后，就可以控制四足完成对应的步态动作了，点击顺序执行所选，就会从第一条数据开始执行，每隔间隔时间执行下一条数据，直到执行到最后一条后停止。顺序运行完成后，可以点击逆序运行所选，程序会从最后一条数据开始执行，直到执行到第一条后停止。该功能可以用于完成某个步态动作后，让动作逆着执行回来，使得四足恢复到原来的位置。

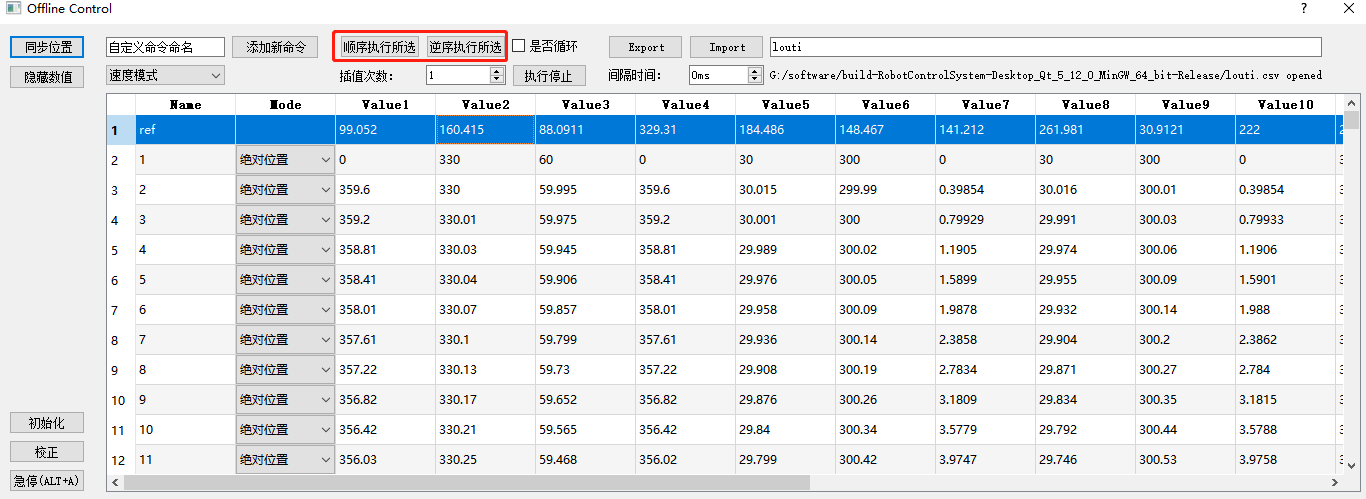
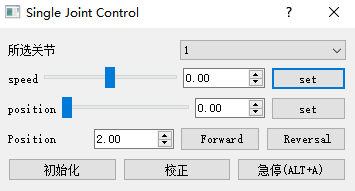


图 33顺序/逆序执行

除了离线控制外，也可以在线控制四足，包括速度点动（松开按钮自动停止）、绝对位置、相对位置运动。可以从Control菜单栏Single Joint Control、Joint Control打开。



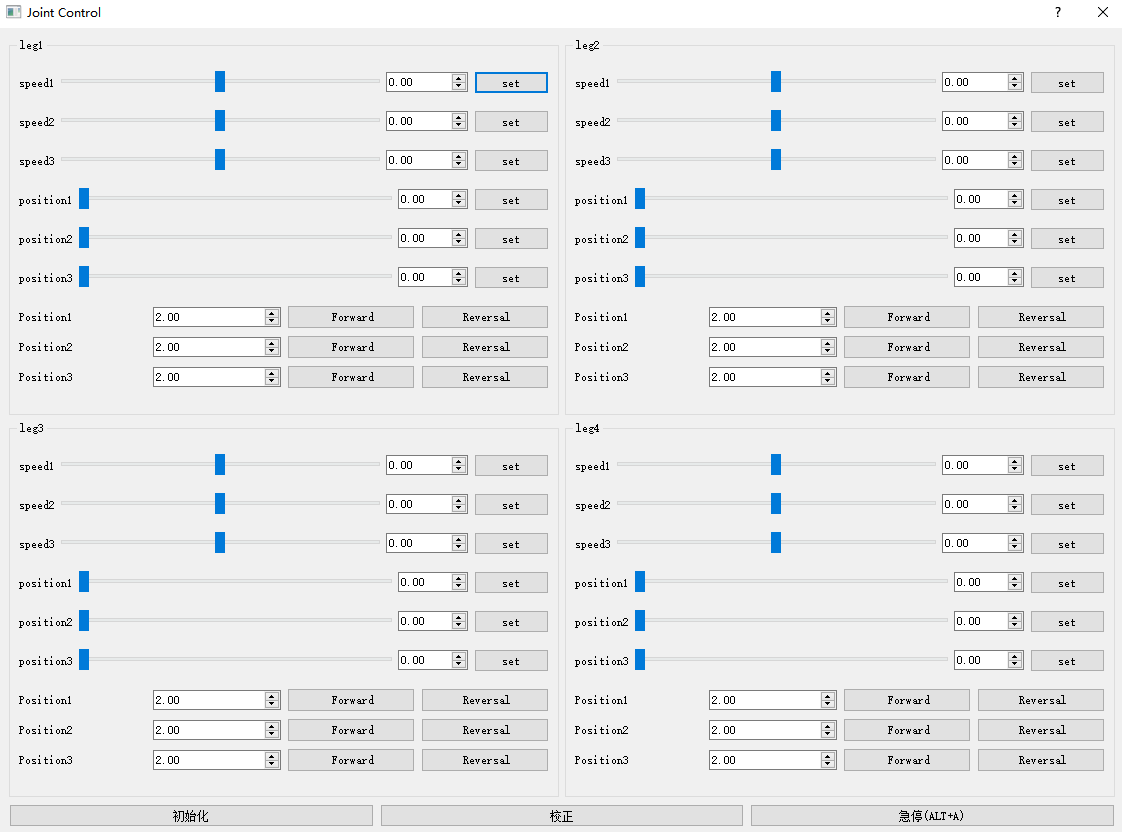


图 34关节在线控制

# 开发注意事项

## GIT管理

整个软件的版本管理采用GIT进行，通过Git log可以查看历史的提交记录，当完成一个新的版本时可以按照git add .，git commit -m “备注信息”，完成版本记录，其余命令可以参考git的使用手册。

可以使用VS Code软件打开整个工程，利用自带的git插件打开版本管理工具，如图 34软件工程管理所示，VS Code建议安装C/C++ Intellisense、Doxygen Document、VIM、vscode-fileheader、C/C++ GNU Global、Auto Comment Blocks、Better Comments等插件，方便编辑查看代码。

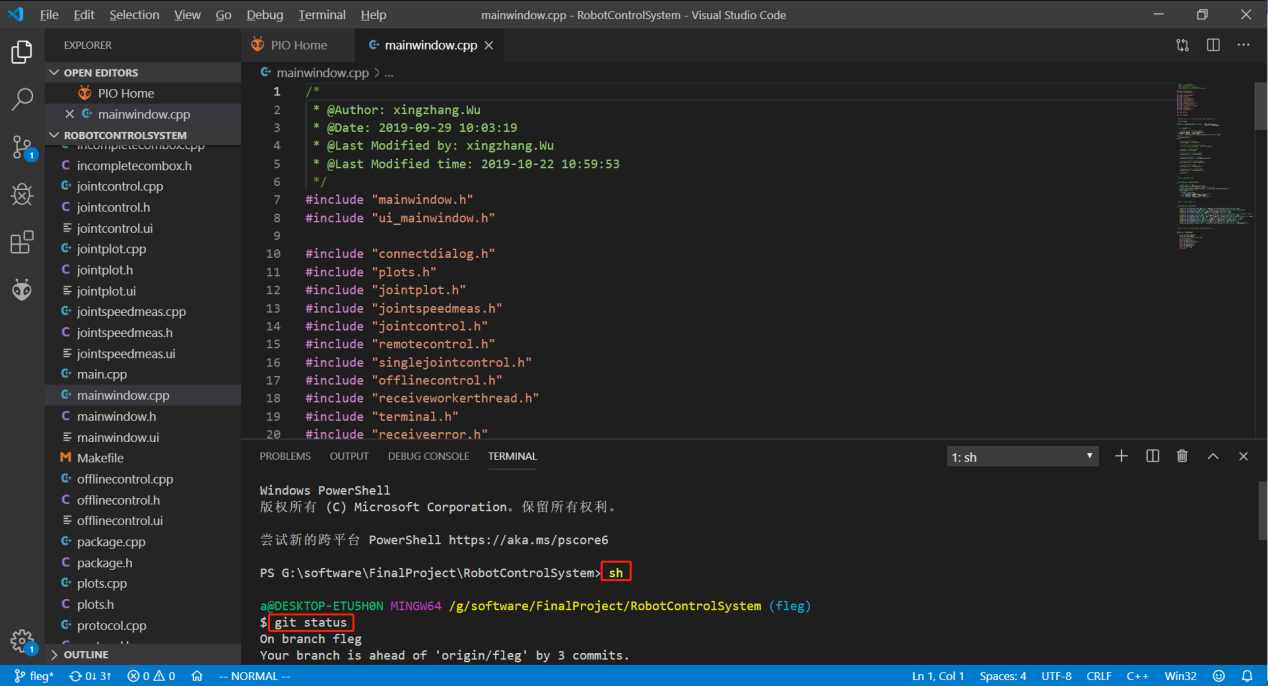


图 35软件工程管理

## 环境配置

### Windows

需要设置环境变量path，增加D:\Qt\Qt5.12.0\5.12.0\mingw73\_64\bin，即你所安装qt目录下的bin路径，安装微软常用运行库合集。

### Linux

1. USBCAN-II新版驱动基于libusb实现，请确保运行环境中有libusb-1.0的库。可连网在线安装，命令如apt-get install libusb-1.0-0。
2. 将工程目录下的libs/zlg/libusbcan.so文件拷到/lib目录，如果源码globaldata.h里面的宏USE\_ZLG关闭了的话，是将libs/cx/libusbcan.so文件拷到/lib目录。
3. 永久赋予普通用户操作USBCAN设备的权限，需要修改udev配置，增加文件：/etc/udev/rules.d/50-usbcan.rules，内容如下：

SUBSYSTEMS=="usb", ATTRS{idVendor}=="0471", ATTRS{idProduct}=="1200", GROUP="users", MODE="0666"

重新加载udev规则后插拔设备即可应用新权限：# udevadm control --reload

## CAN异常处理

之前出现过总控软件接收四足CAN数据时丢帧问题，有的关节丢失数据长达3分钟以上，将波特率降低也无效果，四足数据发送周期为10ms，总共有12个关节，每个关节有11个字节，所以1s中数据量为12100Byte，而波特率有1M/s，故理论上不会有堵塞的问题，由于下面关节发送CAN数据时并不是隔开发的，有可能多个关节同时发送CAN数据，所以可能会有仲裁的问题，而CAN底层是由自动重发机制的，哪怕仲裁失败后续还会由重发的机会，后面排除底层STM固件打开自动重发功能解决该问题。

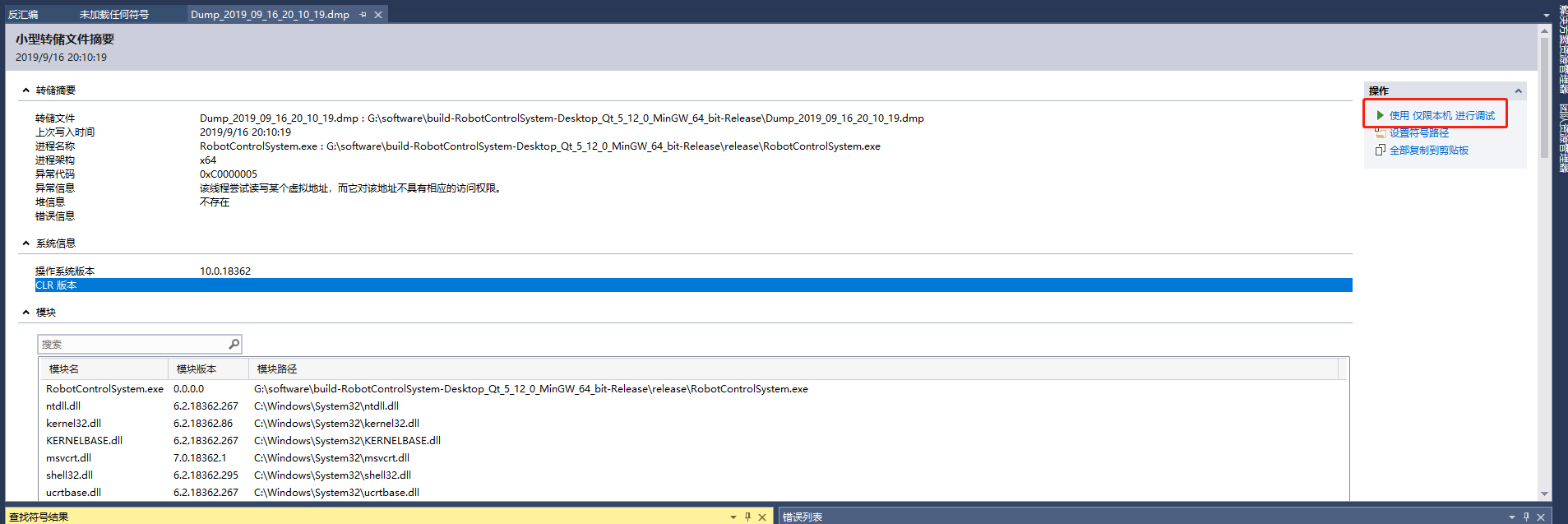
## 异常退出Debug

目的是能够在应用程序出现异常退出的情况下生成dump文件（windows平台）或者core文件（linux平台）。在windows平台可以根据dump文件找到出错的地址，然后更加汇编代码找到出错的代码位置。在linux平台中，当程序异常时，根据生成的core文件进行gdb，找到出错的代码文件即位置，以此来快速找出问题的位置并进行修改。

### Windows平台

第一种方法：在可执行文件目录下打开cmd，输入命令objdump -S RobotControlSystem.exe > RobotControlSystem.asm，根据dmp文件查看具体的出错地址，再在asm文件中寻找相对应的函数名字。

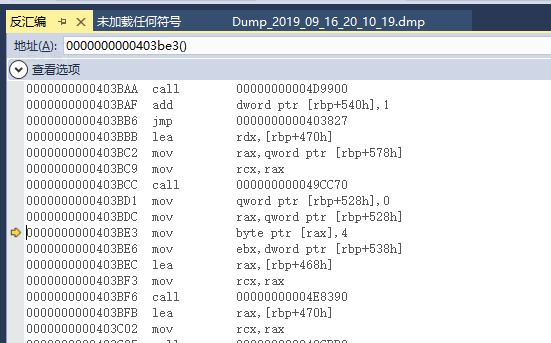
需要现在程序中编写异常回调行数，异常退出时可以生成dump文件，出现问题后，使用vs打开dump文件，直接双击就能自动打开（如果安装了vs的话）。



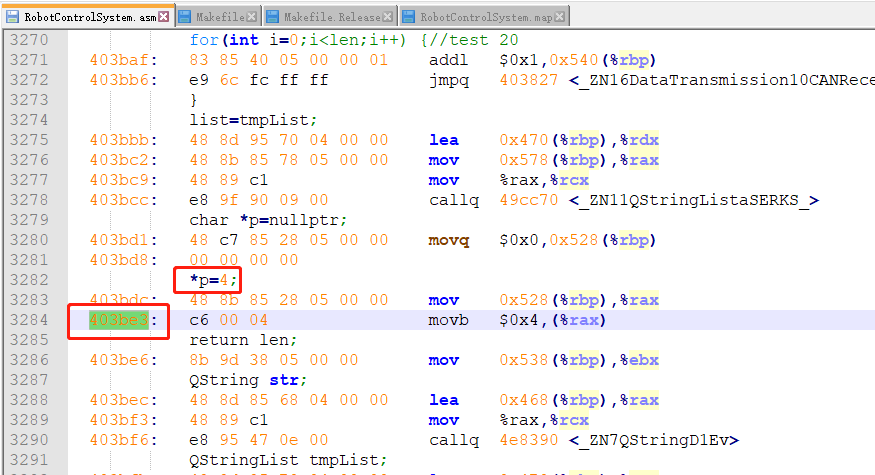
使用本机进行调试，然后点击查看反汇编，即可找到出现该问题是的汇编代码对应的位置。



找到该汇编代码对应的地址。

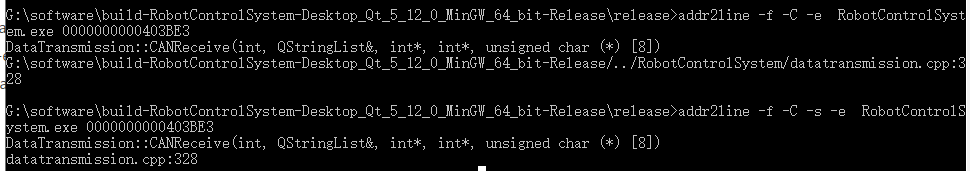


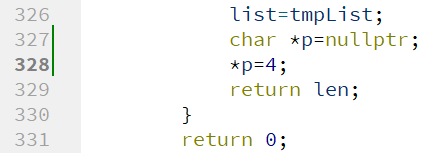
在生成的RobotControlSystem.asm文件中查找该地址，可以看到对应的代码的位置，问题为给野指针赋值导致的奔溃。



第二种方法：cmd命令行输入addr2line.exe -f -C -s -e RobotControlSystem.exe 0000000000403BE3。其中，-f --functions：在显示文件名、行号输出信息的同时显示函数名信息；-C --demangle[=style]：将低级别的符号名解码为用户级别的名字；-s --basenames：仅仅显示每个文件名的基址（即不显示文件的具体路径，只显示文件名）；-e --exe=<executable>：指定需要转换地址的可执行文件名。

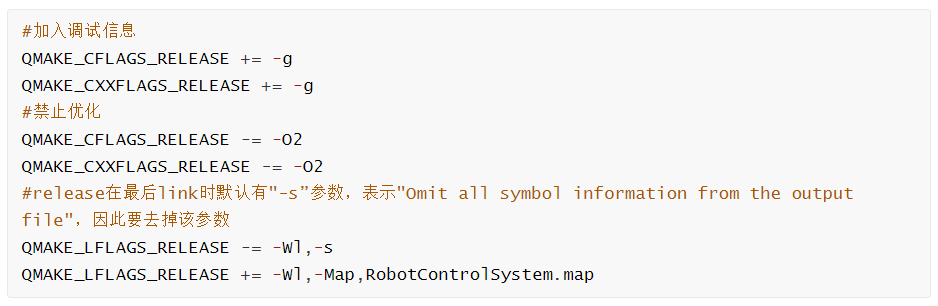
从方法一找出出现问题的代码的汇编地址，然后使用addr2line查找文件中对应的行数。



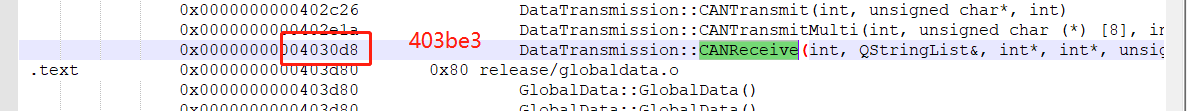


同样能够找到出现问题的位置为328行，给空指针赋值导致的异常。

第三种方法：生成map文件。需要修改pro文件：



根据方法1找到出错的地址，0000000000403BDC，找到map文件中最靠近的一个函数。



### Linux平台

linux环境下，在终端输入ulimit -c unlimited，在运行应用程序，出现异常后，会在该目录生成core文件，最后使用命令：gdb 应用程序名字 core，输入bt，找到出错的位置。

需要配置好LD\_LIBRARY\_PATH。

