中国地质大学

# 三维轨迹规划与控制系统实习报告——

# 基于 EtherCAT 总线的三维运动控制系统设计


指导老师：吴涛、郑世祺

姓　　名：刘瑾瑾

班　　级：231202

学　　号：20201000128


二〇二三年六月

目录

# 第一章 实习概述

## 1.1 实习目的

1) 培养学生实践技能，在学好理论知识的基础上，动手编程和动手实践，练就一身真本领，实现对学生的全面培养。

2) 使学生掌握运动控制系统的基本概念和设计方法，通过自身的动手设计，提高学生的动手能力。学习掌握如何配置高级 C 语言开发环境与运动控制库函数的使用。在充分掌握运动控制库函数的编写原理和程序思想好，自己动手编程，提高学生的编程和独立思考的能力。

3) 在掌握基本知识和常规方法后，每个小组成员共同独立设计三位轨迹规划与控制系统，进行创新改进来增加新功能或者改善原始功能，提高学生的独立思考问题，解决问题，分析问题和思维创新的能力。

## 1.2 实习内容

1) 掌握运动控制系统的基本概念和相关算法。充分了解运动控制系统的组成和功能以及相应的设计方法，并且掌握 EtherCAT 技术的基本原理、寻址方式、通讯模式、分布式时钟等相关功能、组成和原理。

2) 掌握三维总线式运动控制实验平台组成和基本结构。了解三维三维总线式立体加工平台的硬件组成和使用方法、数控加工机床坐标系的建立。

3) 掌握运动控制平台的基本操作。通过控制平台连接结构示意图，学会如何连接控制平台线管硬件，完成对控制器、伺服驱动器、西门子 V20 的接线。以及掌握运动控制器、伺服驱动器面板操作、西门子 V20 变频器的相关使用方法和主要参数。

4) 理解运动控制库函数的设计思想和原理，掌握如何配置高级 C 语言开发环境与运动控制库函数。完成新建基于对话框的 MFC 的应用程序、配置 ZMC 链接库。

5) 掌握 EtherCAT 协议总线型控制器使用方法，完成对控制器连接和掌握控制器连接相关函数。在掌握单轴点位控制实验、手动操作实验的实验原理基础上，完成基本轴参数初始化、加减速控制。

6) 掌握变频器的控制方法，理解 MScom 编程实例编程思想，完成对电机的控制。

7) 掌握复杂轨迹控制方法和插补原理，完成添加控件、添加初始化函数、添加响应函数设计。

8) 完成综合性运控系统的开发，通过 C 和 QT 编程实现，完成对控制区窗口分模块、参数配置（IP 地址链接、自定义轴参数）、自动控制（功能部分、显示部分）、手动控制、状态监测（添加头文件，链接 OpenGL 库文件、在 Visual Studio 下用 OpenGL 绘制坐标系、为成员函数和消息响应函数添加代码）等功能。

9) 掌握 ARTCAM 的使用方法和操作步骤，完成打开 ARTCAM 软件,新建项目、将图片导入新建的项目、查看 3D 视图、光顺浮雕设置阳模/阴模、选择刀具路径、设置刀具参数、设置材料厚度、生成刀具路径、加工仿真、生成 G 代码文件。

## 1.3 小组分工

本小组共有 4 名成员：刘瑾瑾、程婷婷、李晨希、吴哲。

分工如下：

刘瑾瑾：队伍负责人，协调安排工作，负责 Qt 上位机密码登录界面和修改界面设计，生成日志与连接数据库代码编写，界面整体功能规划。

程婷婷：负责上位机设计，包括进行 Qt 上位机的界面设计以及编程实现 Qt 上位机的各种控制、显示和辅助功能。

李晨希：对 G 代码进行研究和改进，操作机器进行雕刻，并排除硬件故障，包括变频器、控制器故障等。

吴哲：负责寻找雕刻素材，利用 PS 制作灰度图，并使用 ARTCAM 生成 G 代码，对 G 代码进行研究和改进，并排除程序在运行过程中出现的故障，负责换刀等工作。

# 第二章 系统软硬件介绍

本次实习的三维运动控制系统是一个软硬件兼备的、功能齐全的、结构完善的综合性系统，硬件主要是指运动控制器、伺服驱动器、变频器、伺服电机、主轴电机等组成的三维总线式立体加工平台；软件涉及的内容包括 EtherCAT 总线通信、Modbus 通信、上位机软件设计、Artcam 软件制作 G 代码等。

## 2.1 运动控制系统简介

运动控制通常是指在复杂条件下，将预定的控制方案、规划指令转变成期望的机械运动，实现机械运动的精密控制。随着制造业的快速升级，降低人工成本和缩减制造成本的需求与日俱增，促使着工业运动控制技术不断进步，出现了诸如全闭环交流伺服驱动系统、直线电机驱动技术、可编程计算机控制器、运动控制器、运动控制卡等许多先进的实用技术。

一个典型的运动控制系统主要由上位计算机、运动控制器、驱动器、伺服（步进）电机、执行机构和反馈装置来构成，如下图 2-1 所示。

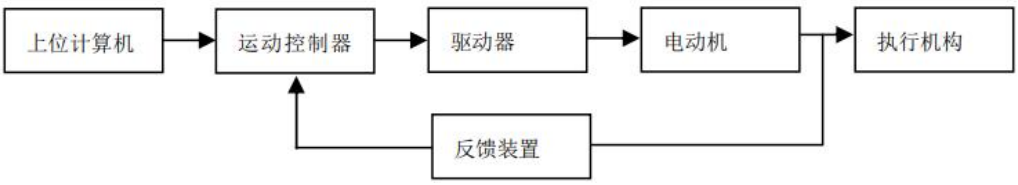


图 2- 1 运动控制系统结构图

运动控制器是以中央逻辑控制单元为核心、以传感器为信号元件，以电机/动力装置和执行单元为控制对象的一种控制装置，主要用于对机械传动装置的位置、速度进行实时的控制管理，使运动部件按照预期的轨迹和规定的运动参数完成相应的动作。与传统的数控装置相比，运动控制器具有以下特点:

1) 技术更新功能更加强大，可以实现多种运动轨迹控制，是传统数控装置的换代产品。

2) 结构形式模块化，可以方便地相互组合，建立适用于不同场合、不同功能需求的控制系统。

3) 操作简单，在 PC 上经简单编程即可实现运动控制，而不一定需要专门的数控软件。

目前，运动控制技术由面向传统的数控加工行业专用运动控制技术而发展为具有开放结构、能结合具体应用要求而快速重组的先进运动控制技术。与此相适

应，运动控制器从以**单片机、微处理器为核心或以专用芯片(ASIC)**为核心处理器的运动控制器，发展到了**基于 PC 总线**的以 DSP 和 FPGA 作为核心处理器的开放式运动控制器。这种开放式运动控制器，充分利用 DSP 的计算能力，进行复杂的运动规划、高速实时多轴插补、误差补偿和运动学、动力学计算，使得运动控制精度更高、速度更快、运动更加平稳；充分利用 DSP 和 FPGA 技术，使系统的结构更加开放，可根据用户的应用要求进行客制化的重组，设计出个性化的运动控制器。基于 PC 总线的开放式运动控制器已成为当今自动化领域应用最广、功能最强的运动控制器，并且在全球范围内得到了广泛的应用。

传统运动控制系统采用脉冲+方向的位置闭环控制，在复杂系统中如机器人驱动系统中，由于动力线和反馈线众多，让走线非常复杂。随着工业 4.0、智能制造、工业互联等概念的深入推广与逐步落地，极大地推动了运动控制产品在工业以太网、模块化及分布式伺服驱动器、深度软件开发等领域的发展。因此，使运动控制器系统网络化并拥有良好的人机交互接口成为新的趋势。总线型伺服控制已成为现在主流发展趋势，特别在工业机器人中应用广泛，大大简化现场布线，也使得故障率大大降低。

## 2.2 EtherCAT 技术原理

EtherCAT（Ethernet for Control Automation Technology）总线技术是由德国的 Beckoff（倍福）自动化公司于 2003 年提出的一种开放式工业以太网现场总线技术，有着以下的优势：第一，系统结构简单、拓扑结构灵活、数据传输高效，长距离传输时信号不易受干扰；第二，EtherCAT 总线技术是在传统商用以太网技术的基础上改造而来的，因此与标准以太网有着很好的兼容性；第三，具有良好的同步性能，同步精度可小于 1us，同步抖动可达到 20ns 以下，这对于自动化控制领域来说有着举足轻重的优势。

传统以太网的通信过程是每个节点接收、处理、转发数据包按次序进行。EtherCAT 改造了这个过程，在特殊硬件 IP 核的帮助下， EtherCAT 可以同时传输和处理以太网数据包。每个从站节点都有现场总线内存管理单元 FMMU（Fieldbus Memory Manage Unit），FMMU 的功能是对经过从站节点的数据包进行地址分析，如果发现该数据包中有属于本节点的数据则会读取该数据并同时转发报文至下一个设备。同样地，在报文经过的时候也可以插入数据。 读取→插入→转发数据的整个过程都由硬件来完成，因此 报文传输的实时性将由具体物理层器件的性能决定。通常报文在每个节点处只有几纳秒 的延迟，这使得通信的循环周期时间大大缩短，系统整体的通信效率不再受到从站所采用的处理器的响应时间影响。EtherCAT 运行原理如图 2-2 所示。

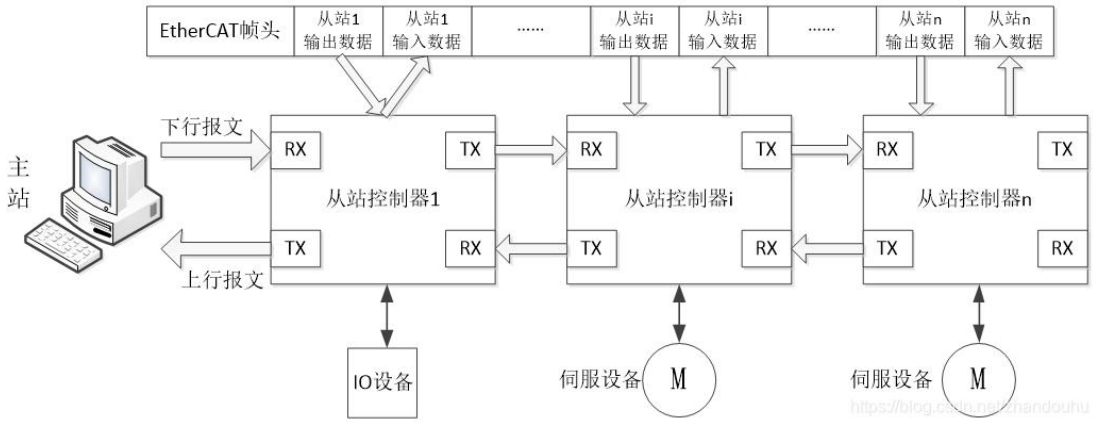


图 2- 2 EtherCAT 运行原理图

EtherCAT 巧妙利用了以太网全双工通信的特点，最终形成了主从式的环形逻辑拓扑结构。实质上，每个 EtherCAT 网段都可以看作一个可以接收并发送以太网报文的独立的以太网设备，只不过它没有类似的以太网控制器和处理器，取而代之的是按照特定拓扑结构级联的 EtherCAT 从站。每个 EtherCAT 从站在接收到的报文中提取或者插入用户数据，然后将处理好的报文发送到下一个 EtherCAT 从站。报文在依次被所有从站处理之后由最后一个 EtherCAT 从站回传，并由第一个从站作为响应报文返回给 EtherCAT 主站控制单元。整个过程基于以太网的全双工模式，通过 Tx 线发送出去的报文会从 Rx 线返回，因此在 EtherCAT 通信系统中，任何物理拓扑结构在逻辑上永远是环形。EtherCAT 的每个从站设备对过程数据的大小几乎没有限制，可以从 1 比特到 60K 字节不等，如果有需要还可以使用多个以太网帧来传输。同一个以太网帧中可以嵌入多个 EtherCAT 命令数据，每个数据对应独立的从站设备或从站的内存区域。为了让数据的处理和传输直接在标准以太网帧中进行， EtherCAT 做了相应的优化，并且将标准以太网帧的 Ethertype 值修改为 0x88A4。

EtherCAT 可以实现非常短的收敛周期，其中一个重要的原因在于过程数据的交换工作不需要处理器来完成，全部交给硬件来执行。EtherCAT 保证了带宽利用率的最大化，每个从站节点的数据不需要使用单独的以太网帧来传输。利用全双工通信的优势，EtherCAT 可以实现超过 100Mbps 的数据速率。一个 EtherCAT 网络最多可以连接 65535 个从站设备，每个 EtherCAT 以太网帧可容纳多达 1486 字节的过程数据，刷新周期可达 300us。此外 EtherCAT 技术是可扩展的，并没有固定在百兆以太网的基础上，未来还会扩展到千兆以太网。

## 2.3 三维总线式立体加工平台

该实验装置由三部分组成：三轴 XYZ+旋转主轴三维雕刻机械、电气部分、控制部分。

图 2- 3 三维总线式运动控制实验台

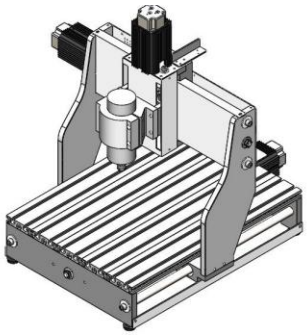

图 2- 4 雕刻机结构示意图

### 2.3.1 机械部分

表 2- 1 结构参数表

| | |
|---|---|
| 工作行程/Working area | 400*300*100mm |
| 最大加工速度/Max working speed | 4000mm/min |
| 最大进料高度/Feeding height | 150mm |
| 加工精度/Working drive | ±0.02~0.05mm |
| 重复定位/Repositioning accuracy | ±0.01~0.02mm |
| 工作电压/Power | AC220V50~60Hz |
| 机架重量/Weight | 约 38KG |
| 台面尺寸/Platform size | 540*400mm |
| 传动单位/Transmission unit | TBI1605 滚珠丝杆，导程 5mm |
| 滑动单位/Gliding unit | XY 轴为 20 直径镀铬加硬光轴,Z 轴为 16 直径镀铬加硬光轴 |
| 主轴功率/Main axle power rate | 800W(循环水冷) |
| 主轴转速/Main axle rotiting speed | 0-3000 RPM/min |
| 刀具安装直径/Engraving toois | ER11 - 1~7mm（默认 3.175 夹头） |
| 主轴尺寸/Spindle size | 158*65mm |
| 电压电流/Voltage and current | AC220V（必须使用变频器输出电压） 5A |

雕刻机由 XYZ 三轴滚珠丝杠+导轨组成，滚珠丝杠把伺服电机的旋转运动变成直线运动。丝杠旋转一圈的运动距离是 5mm（丝杠导程）。滚珠丝杠是工具机械和精密机械上最常使用的传动元件，其主要功能是将旋转运动转换成线性运动，或将扭矩转换成轴向反复作用力，同时兼具高精度、可逆性和高效率的特

点。由于具有很小的摩擦阻力，滚珠丝杠被广泛应用于各种工业设备和精密仪器。

### 2.3.2 电气部分

包含 3 个迈信 EP3E-GL1A8-EC 总线型伺服驱动器，2 个 200W 伺服电机，1 个带抱闸伺服电机。西门子变频器 6SL3210-5BB17-5UV1+800W 高速主轴电机。 急停开关、门限位开关，XYZ 三轴左右限位开关。主要外接输入信号接入正运动控制器 IN 点如下图 2.4 所示。三个伺服电机编码器旋转一周产生 217 个脉冲。

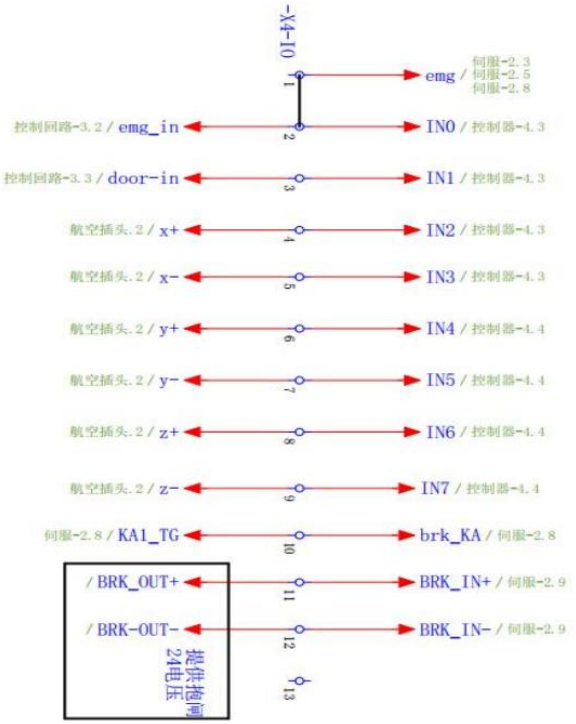


图 2- 5 外部输入信号接线图

### 2.3.3 控制部分

正运动 EtherCAT 运动控制器，支持 60/30 个 EtherCAT/RTEX 数字伺服轴，支持 EtherCAT/RTEX/脉冲 3 种类型轴混合插补。支持 ZBasic 多任务编程，支持直线插补、圆弧插补、螺旋线插补、空间圆弧、支持速度前瞻，电子齿轮、电子凸轮、运动叠加、比较输出、支持多任务，多轴组，多机械手协同运动。

### 2.3.4 数控加工机床坐标系

在数控编程时为了描述机床的运动，简化程序编制的方法及保证记录数据的互换性，数控机床的坐标系和运动方向均已标准化，ISO 和我国都拟定了命名

的标准。机床坐标系（Machine Coordinate System）是以机床原点 O 为坐标系原点并遵循右手笛卡尔直角坐标系建立的由 X、Y、Z 轴组成的直角坐标系。机床坐标系是用来确定工件坐标系的基本坐标系，是机床上固有的坐标系，并设有固定的坐标原点。

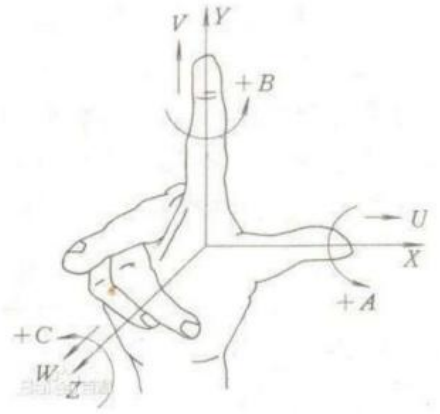


图 2- 6 右手笛卡尔坐标系

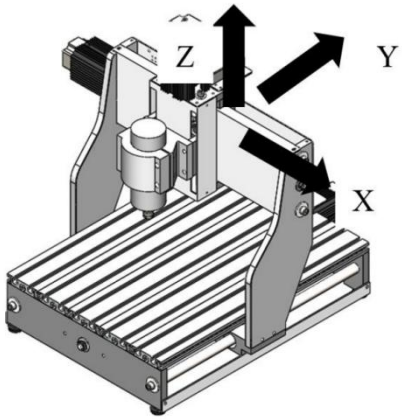


图 2-7 数控雕刻机床坐标系

坐标原则：

1）遵循右手笛卡儿直角坐标系。

2）永远假设工件是静止的，刀具相对于工件运动。

3）刀具远离工件的方向为正方向。

坐标轴

1）先确定 Z 轴。

a）传递主要切削力的主轴为 Z 轴。

b）若没有主轴，则 Z 轴垂直于工件装夹面。

c）若有多个主轴，选择一个垂直于工件装夹面的主轴为 Z 轴。

2）再确定 X 轴。（X 轴始终水平，且平行于工件装夹面）

a）没有回转刀具和工件，X 轴平行于主要切削方向。（牛头刨）

b）有回转工件，X 轴是径向的，且平行于横滑座。（车、磨）

c）有刀具回转的机床，分以下三类：

① Z 轴水平，由刀具主轴向工件看，X 轴水平向右。

② Z 轴垂直，由刀具主轴向立柱看，X 轴水平向右。

③ 龙门机床，由刀具主轴向左侧立柱看，X 轴水平向右。

3）最后确定 Y 轴。按右手笛卡儿直角坐标系确定。

## 2.4 运动控制平台的组成

如图 2-8 所示，运动控制平台由运动控制器、伺服驱动器、变频器和电机四

部分组成。

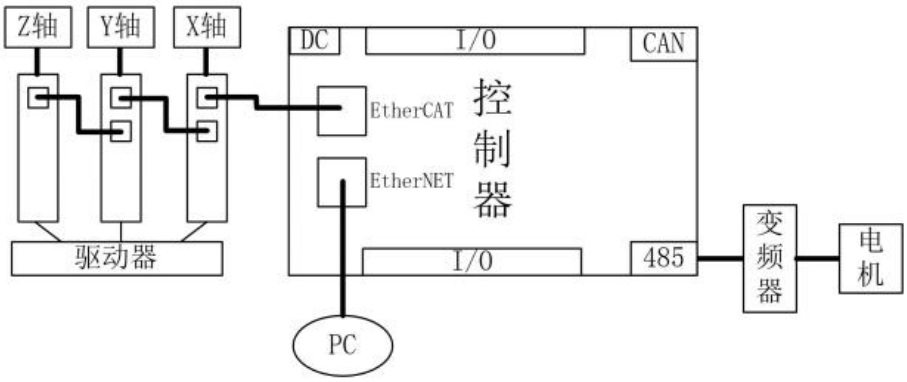


图 2- 8 控制平台连接结构示意图

### 2.4.1 运动控制器

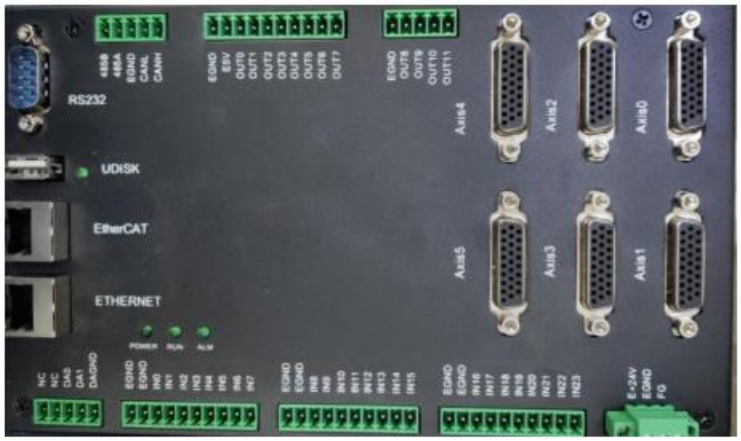


图 2- 9 运控控制器实物图

从控制器上的标注可以非常清楚的看到控制器资源，控制器采用 DC 24V 供电。主要包含：1）两路网口（ETHERNET、ETHERCAT）；2）RS232 串口，连接其他外设；3）485 接口，此运动平台采用 485 接口连接变频器来控制主轴电机；4）CAN 接口，支持通过 ZCAN 协议来连接扩展模块，比如 CAN 温度检测模块；5）多路输入输出，可以作为限位开关、原点开关、操作按钮等信号连接接口；6）DA 信号输出：控制部分特殊电机。

### 2.4.2 伺服驱动器

伺服驱动器又称为“伺服控制器”、“伺服放大器”，是用来控制伺服电机的一种控制器，其作用类似于变频器作用于普通交流马达，属于伺服系统的一部分，主要应用于高精度的定位系统。一般是通过位置、速度和力矩三种方式对伺服电机进行控制，实现高精度的传动系统定位，目前是传动技术的高端产品。


图 2- 10 伺服驱动器示意图

### 2.4.3 西门子 V20 变频器

变频器（Variable-frequency Drive，VFD）是应用变频技术与微电子技术，通过改变电机工作电源频率方式来控制交流电动机的电力控制设备。变频器主要由整流（交流变直流）、滤波、逆变（直流变交流）、制动单元、驱动单元、检测单元微处理单元等组成。变频器靠内部 IGBT 的开断来调整输出电源的电压和频率，根据电机的实际需要来提供其所需要的电源电压，进而达到节能、调速的目的，另外，变频器还有很多的保护功能，如过流、过压、过载保护等。

图 2- 11 变频器实物图

### 2.4.4 主轴电机

主轴电机与控制 XYZ 三个轴运动的伺服电机不同，它的由变频器控制，主要参数如下表所示。

表 2- 2 主轴电机基本参数

| 指标 | 数值 |
| --- | --- |
| 功率 | 800W |
| 额定电压 | 220V |
| 最大运行频率 | 400Hz |
| 极对数 | 1 |
| 冷却方式 | 水冷 |
| 最高转速 | 24000r/min |
| 相数 | 单相电机 |

## 2.5 Artcam 软件生成 G 代码

### 2.5.1 ArtCAM 软件

ArtCAM 软件产品系列是英国Delcam公司出品的独特的CAD造型和CNC、CAM 加工解决方案，是复杂立体三维浮雕设计、珠宝设计和加工的首选CAD/CAM 软件解决方案。ArtCAM 软件可以把如手绘稿、扫描文件、照片、灰度图、CAD 等文件一切平面数据，转化为生动而精致的三维浮雕数字模型，并生成能够驱动数控机床运行的代码。ArtCAM 包括了丰富的模块，这些模块功能齐备，运行快速，性能可靠，极富创造性。

### 2.5.2 G 代码

G 代码（G-code，又称 RS-274），是最为广泛使用的数控（numerical control）编程语言，有多个版本，主要在计算机辅助制造中用于控制自动机床。可以实现快速定位、逆圆插补、顺圆插补、中间点圆弧插补、半径编程、跳转加工等功能。1）使用 ArtCAM 生成的 G 代码文件中的语句主要包括以下内容：

①G 代码：准备功能，控制机床动作（比如 G00 快速移动）

②M 代码：辅助功能，辅助机床动作（比如 M03 主轴正转）

③T 指令：换刀

④S 指令：设定主轴转速(r/min)

⑤F 指令：表示工件被加工时，刀具相对于工件的合成进给速度，F 的单位取决于 G94(每分钟进给量 mm/min)或 G95(主轴每转一转刀具的进给量 mm/r)。使用下式实现<每转进给量>与<每分钟进给量>的转化：

$$fm=fr\times S$$

fm：每分钟的进给量：(mm/min )

fr：每转进给量：(mm/r )

S：主轴转数，(r/min)

当工作在 G01，G02 或 G03 方式下，编程的 F 一直有效，直到被新的 F 值所取代，而工作在 G00 方式下，快速定位的速度是各轴的最高速度，与所编 F 无关。

2）常用的 G 代码指令如下：

表 2- 3 G 代码及功能汇总表

| G 代码 | 功能 | 格式 |
|---|---|---|
| G00 | 快速移动 | G00X_Y_Z_ |
| G01 | 直线插补 | G01X_Y_Z_F_ |
| G02 | 顺圆插补 | G02X_Y_Z_R_<br>G02X_Y_Z_I_J_K_ |
| G03 | 逆圆插补 | G03X_Y_Z_R_<br>G03X_Y_Z_I_J_K_ |

3）常用的 M 代码功能如下：

表 2- 4 M 代码及功能汇总表

| M 代码 | 功能 |
|---|---|
| M00 | 程序停止 |
| M01 | 选择停止 |
| M02 | 程序结束 |
| M03 | 主轴正转 |
| M04 | 主轴反转 |
| M05 | 主轴停止转动 |
| M06 | 换刀指令 |
| M08 | 切削液开 |
| M09 | 切削液关 |
| M19 | 主轴定位 |
| M30 | 程序结束，并返回程序起始 |

# 第三章 上位机软件设计

本部分主要介绍以 Qt 软件开发的三维雕刻机实验平台的可视化上位机控制程序，以控制和获取系统的运动状态。对于本章的叙述，首先为基本的开发环境配置，其次根据 Qt 设计的四个界面及功能分别进行叙述。

## 3.1 环境配置

### 3.1.1 开发环境

本程序基于 Qt 5.14.2 软件进行开发，编译器为 MinGW 32bit，环境为 Windows 10_32 位操作系统：Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.60 GHz 处理器；16GB 内存。使用有线网络接口将 PC 连接到三维雕刻机实验平台所在局域网下，并将 PC 的 IP 地址设置为 192.168.0.5，子网掩码设置为 255.255.0.0。

### 3.1.2 开发文件配置

如果使用的 kits 是 MinGW，则应使用 MinGW 目录下的 zmotion 库，根据使用的 kits 选择 32 位库。

1）将 zmotion.lib、zmotion.h 和 zmotion.h 复制到工程目录下。

a）直接打开 zmotion.h 文件，会发现汉字注释显示乱码：

```
1  /********************************* ZMCε�n�����  ************************************
16
17
18 #ifdef  __cplusplus
19 extern "C" {
20 #endif
21
22
23 #ifndef _ZMOTION_DLL_H
24 #define _ZMOTION_DLL_H
25
26
27 /*******************************************************
28 ********И���
29 *******************************************************/
30
31 typedef unsigned __int64   uint64;
32 typedef __int64   int64;
33
34
35 //#define BYTE           INT8U
36 //#define WORD           INT16U
37 //#define DWORD          INT32U
38 typedef unsigned char  BYTE;
39 typedef unsigned short  WORD;
40 //typedef unsigned int  DWORD;
41 //#define __stdcall
42 typedef unsigned char  uint8;                /* defined for unsigned 8-bits integer variabl
43 typedef signed   char  int8;                 /* defined for signed 8-bits integer variable
```

图 3- 1 zmotion.h 文件乱码示意图

b)将该文件以记事本打开，另存为“UTF-8”编码格式，即可正常显示。

c)由于 C++环境中缺少“__int64”类型，需添加头文件“stdint.h”：

```
#ifndef _ZMOTION_DLL_H
#define _ZMOTION_DLL_H
#include "stdint.h"

/************************************************************
数据类型定义
************************************************************/

typedef unsigned __int64   uint64;
typedef __int64   int64;
```

图 3- 2 添加“stdint.h”头文件

d)添加库文件：

① 选择“外部库”。

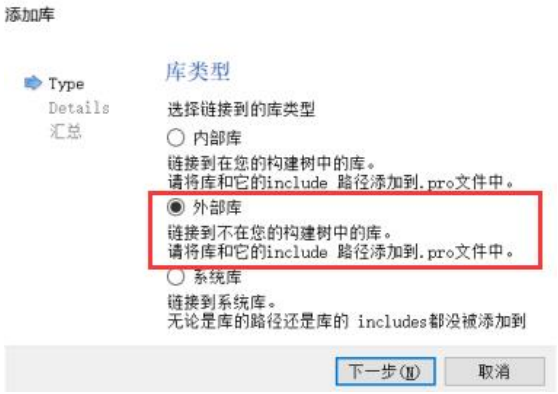


图 3- 3 选择“外部库”

② 添加“库文件 zmotion.lib”,去掉“为 debug 版本添加‘d’作为后缀”勾选。

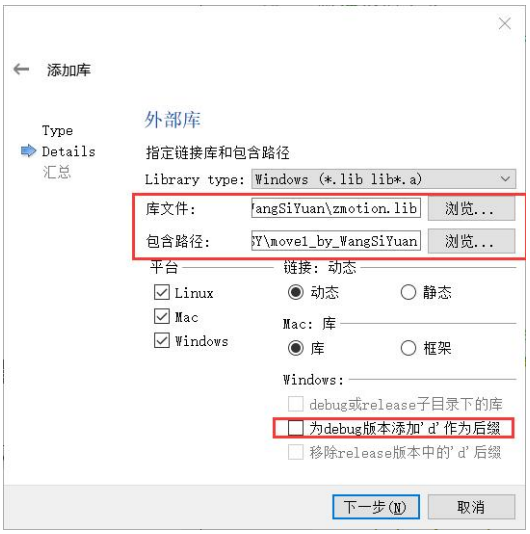


图 3- 4 添加库文件的设置参数

③ 工程文件.pro 中出现下列代码即为设置成功。


图 3- 5 添加库代码设置

2）由于 MinGW 没有 ZAux 辅助函数库，需添加 zaux.cpp 和 zaux.h 两个文件。ZAux 辅助函数库的实质就是向控制器发送一个字符串的 Basic 语言代码，根据参数，组织成 Basic 代码，使用 ZMC_DirectCommand()函数将 Basic 代码发送到控制器。

3）添加绘图库函数：为了实时显示雕刻机末端运动的轨迹，采用 OpenGl 库搭建三维显示窗口，根据末端的移动坐标绘制运动轨迹。

## 3.2 整体设计

上位机程序通过 Qt 信号与槽机制编写，由登录界面，主界面，密码修改界面三个界面组成。

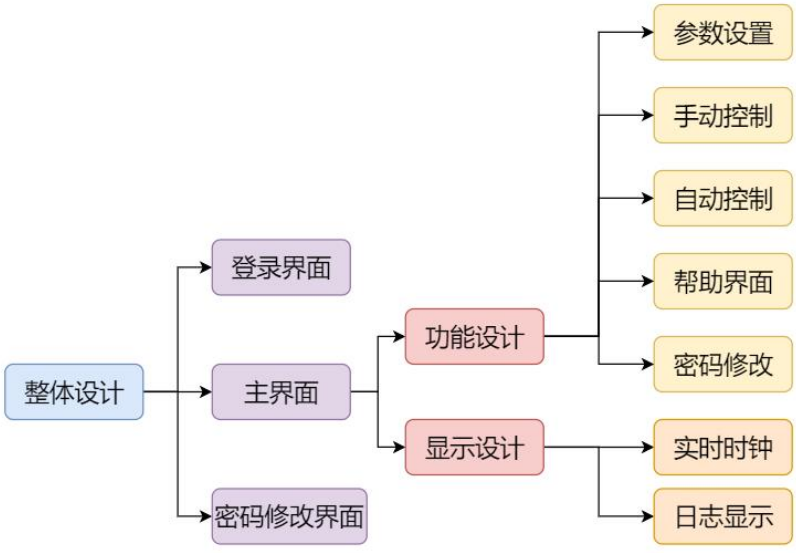


图 3- 6 整体设计结构图

### 3.2.1 登录界面

运行上位机程序后，弹出登录界面，账号为小组四个成员的学号，密码默认为学号后六位，提升了系统的保密性，且设置万能账户“20201000000”，密码为“000000”，以防止忘记密码。账号密码采用数据库存储，管理效率高。登录界面默认账号为“20201000128”，无需每次都输入账号，直接登录即可，方便进行操作。


图 3- 7 登录界面

### 3.2.2 主界面

主界面包括主界面包括参数设置、手动控制、自动控制、帮助界面和密码修改功能，并配有实时时钟与日志显示。


图 3- 8 主界面设计

### 3.2.3 密码修改界面

点击主界面的“密码修改”图标，进入密码修改界面。主要设计功能如下：

1) 输入旧密码、新密码和再次输入新密码新密码
2) 判断旧密码是否输入错误
3) 判断新密码是否一致，不一致弹出警告框，将新密码文本框清空
4) 判断新旧密码是否一致，一致弹出警告框，将新密码文本框清空
5) 若无上述情况，则修改成功


图 3- 9 密码修改界面

## 3.3 参数设置

只有设置好相应的参数，运动控制系统才能正常工作，默认初始化参数如下图所示：

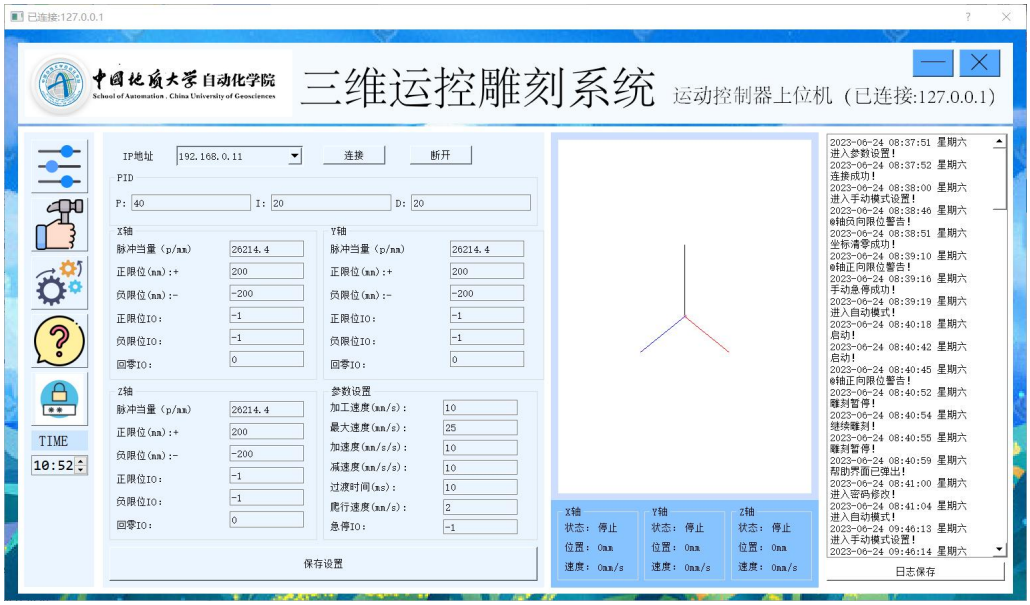


图 3- 10 默认参数显示

### 3.2.1 IP 地址连接

1）设计思想

在对控制器进行操作时必须先调用链接函数链接控制器，当链接成功后方可通过返回的句柄操作对应的控制器。在关闭应用程序时要调用断开链接函数释放链接。 如表 3-11 所示，控制器连接函数如下：

图 3- 11 控制器连接相关函数

| 函数 | 功能 |
| --- | --- |
| ZAux_ OpenCom | 串口链接控制器 |
| ZAux_ SetComDefaultBaud | 串口通讯参数设置 |
| ZAux_ OpenEth | 以太网链接控制器 |
| ZAux_ SearchEthlist | 搜索当前网段下的控制器 IP |
| ZAux_ Close | 关闭控制器链接 |

2）程序实现

```
1. void Dialog::on_lianjie_clicked()
2. {
3.     char*  buffer;
4.     int32 iresult;
5.     if(nullptr != g_handle)
6.     {
7.         ZMC_Close(g_handle);
8.         g_handle = nullptr;
9.         killTimer(m_timerid);//停止计时
10.        killTimer(m_timerid_3D);//停止计时
11.    }
12.    QString currentIP(ui->comboBox->currentText());
13.    //connect(ui->comboBox,SIGNAL(currentIndexChanged(int)),this,
 SLOT(            slotLoadList(int)));
14.    QByteArray ba = currentIP.toUtf8();  //借助 bytearray 将
Qstring 转            char*
15.    buffer=ba.data();
16.    iresult = ZMC_OpenEth(buffer, &g_handle);
17.    if(ERR_SUCCESS != iresult)
18.    {
19.        g_handle = nullptr;
20.        messagebox_warning.setText("连接失败");
21.        QDateTime time = QDateTime::currentDateTime();
22.        QString qstr = time.toString("yyyy-MM-dd hh:mm:ss dddd")
;
23.        text += qstr.toStdString();
24.        text +="\n 连接失败！ \n";
25.        ui->text_log->setText(QString::fromStdString(text));
26.        messagebox_warning.exec();
27.        setWindowTitle("未连接");
28.        ui->labTitle->setText(QString("运动控制器上位机（%1）
").arg("未连                  接"));
```

```
        return;
    }
    m_timerid=startTimer(9);//开始计时
    m_timerid_3D=startTimer(8);//开始计时
    setWindowTitle("已连接:"+currentIP);

    ui->labTitle->setText(QString("运动控制器上位机（%1）").arg("已连
                接:"+currentIP));
    lianjie_ok=1;
    QDateTime time = QDateTime::currentDateTime();
    QString qstr = time.toString("yyyy-MM-dd hh:mm:ss dddd");
    text += qstr.toStdString();
    text +="\n 连接成功！\n";
    ui->text_log->setText(QString::fromStdString(text));
}
```

点击“连接”按钮，首先判断当前是否处于连接状态。若是，则先断开当前连接，在连接新的运动控制器；否则，则直接连接。同理，点击“断开”按钮时，也会判断当前是否处于连接状态，以免重复调用函数出现无法预料的错误。

3）计算机网络设置

检测函数返回值判断函数是否执行成功，返回 0 时则成功，非 0 失败。以太网口链接返回不成功时，检查传递的 IP 是否正确，并且保证电脑 IP 地址和控制器 IP 地址在同一个网段。

PC 与控制器链接时，注意连接 ETHERNET 口。同时在配置 PC 网口，设置→网络 Internet→状态→更改适配器选项，选择所连接的网口，右键属性，如下图所示：

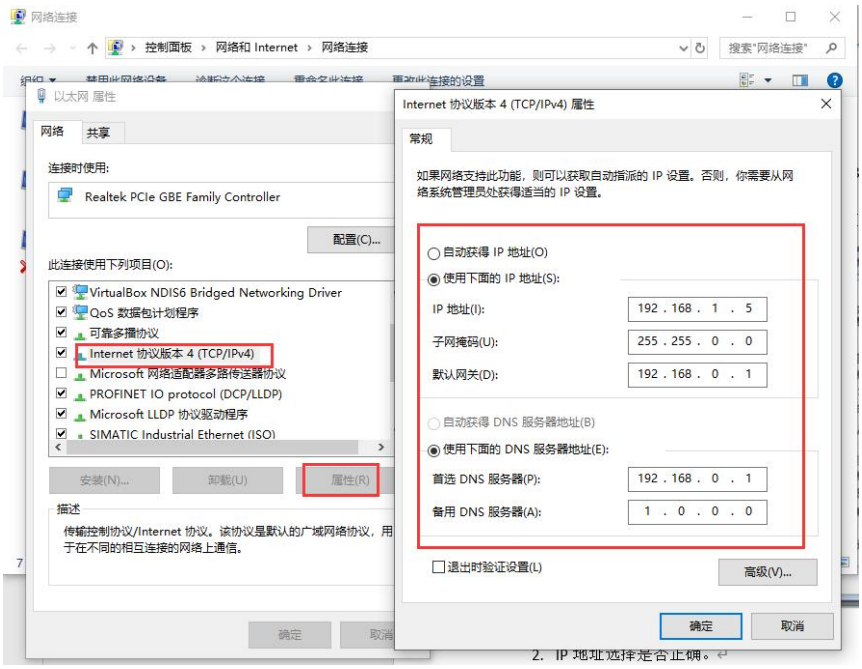


图 3- 12 更改适配器选项

### 3.2.2 轴参数设置

1）设计思想

本次实习的雕刻机为三维，其有三个轴分别为 X 轴、Y 轴、Z 轴。其三个轴对应为三组轴参数，在对轴参数进行配置时，主要用到以下函数：

表 3- 1 轴参数配置函数

| 函数名 | 功能 |
| --- | --- |
| ZAux_ Direct SetAtype | 设置轴类型 |
| ZAux_ _Direct_ SetUnits | 设置轴脉冲当量 |
| ZAux_ Direct_ SetInvertStep | 设置脉冲输出模式 |
| ZAux_ Direct SetSpeed | 设置轴速度 |
| ZAux_ _Direct_ SetAccel | 设置轴加速度 |
| ZAux_ Direct_ SetDecel | 设置轴减速度 |
| ZAux_ Direct SetSramp | 设置轴 S 曲线时间 |

① 轴类型：常用的类型为 1-脉冲输出类型(步进/伺服)，3-编码器输入类型(外部编码器/轴本身的编码器)，根据实际的硬件来配置对应的轴类型。在本章所介绍的虚拟仿真环境中，使用的为虚拟轴(0)，其类型为程序内部模拟运行的轴，通常用于轴叠加与机械手场合。

| Atype 类型 | 描述 |
| --- | --- |
| 0 | 虚拟轴； |
| 1 | 脉冲方向方式的步进或伺服 |
| 2 | 模拟信号控制方式的伺服 |
| 3 | 正交编码器 |
| 4 | 步进+编码器 |
| 6 | 脉冲方向方式的编码器，可用于手轮输入 |
| 7 | 脉冲方向方式步进或伺服+EZ 信号输入 |
| 8 | ZCAN 扩展脉冲方向方式步进或伺服 |
| 9 | ZCAN 扩展正交编码器。 |
| 10 | ZCAN 扩展脉冲方向方式的编码器。 |
| 65 | ECAT 脉冲类型 |
| 66 | ECAT 速度闭环 |
| 67 | ECAT 力矩闭环 |
| 70 | ECAT 自定义操作，只读取编码器. |

图 3- 13 轴类型

② 脉冲当量为矢量轴参数的基本单位，如速度、加速度、位置、等。当 units 设置为 1 时，speed 单位为脉冲/s，当 units 设置为机构移动 1mm 所需脉冲数时， 那么 speed 单位为 mm/s。 必须先修改脉冲当量，然后再修改速度、位移这些参数。否则当 units 变化时会导致速度、加速度、位置的变化。

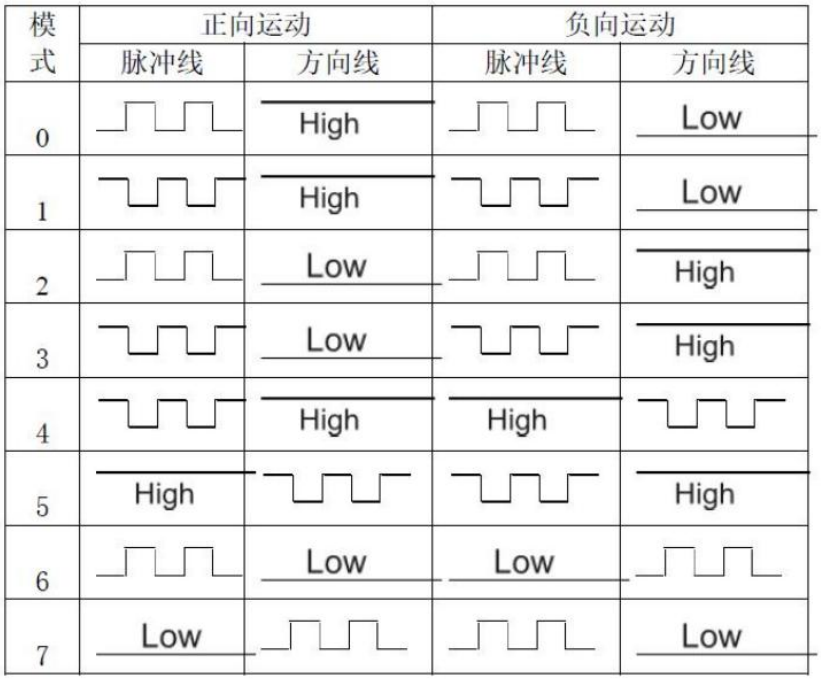


图 3- 14 脉冲输出模式

③ 其中，运动控制系统插补器中很重要的一环则是加、减速控制。常用的加、 减速控制方式有直线加减速(T 曲线加减速)、三角函数加减速、指数加减速、s 曲线加减速等，本次实习采用的是直线加减速(T 曲线加减速)算法。其具体原理如下：

当前指令进给速度 $V_{i+1}$ 大于前一指令进给速度 $V_i$ 时，处于加速阶段。瞬时速度计算如下：

$$V_{i+1}=V_i+aT$$

式中，a 为加速度；T 为插补周期。此时系统以新的瞬时速度 $V_{i+1}$ 进行插补计算，得到该周期的进给量，对各坐标轴进行分配。这是一个迭代过程，该过程一直进行到 $V_i$ 为稳定速度为止。

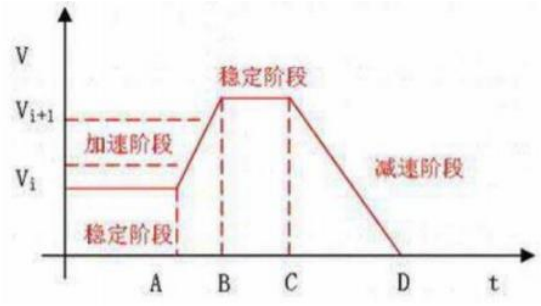


图 3- 15 直线加减速

同理，处于减速阶段时 $V_{i+1}=V_i-aT$。此时系统以新的瞬时速度进行插补计算，这个过程一直进行到新的稳定速度为零为止。

这种算法的优点是算法简单，占用机时少，响应快，效率高。但其缺点也很明显，从图 2-1 中可以看出，在加减速阶段的起点 A、C,终点 B、D 处加速度有突变，运动存在柔性冲击。另外，速度的过渡不够平滑，运动精度低。因此，这种加减速方法一般用于起停、进退刀等辅助运动中。

2）程序实现（部分）

```
void Dialog::setstatus()
{
    float inttemp;
    QString strtemp;
    if(windowTitle()=="未连接")
    {
        messagebox_warning.setText("请先连接控制器");
        messagebox_warning.exec();
        return;
    }
    if(ui->comboBox->currentText()=="127.0.0.1")
    {
        ZAux_Direct_SetAtype(g_handle,0,0);//设置类型为脉冲方向
        ZAux_Direct_SetInvertStep(g_handle,0,1);//设置类型为脉冲方向
        ZAux_Direct_SetAtype(g_handle,1,0);//设置类型为脉冲方向
        ZAux_Direct_SetInvertStep(g_handle,1,1);//设置类型为脉冲方向
        ZAux_Direct_SetAtype(g_handle,2,0);//设置类型为脉冲方向
        ZAux_Direct_SetInvertStep(g_handle,2,1);//设置类型为脉冲方向
    }
    else
        ZAux_Direct_SetAtype(g_handle,0,65);//设置类型为脉冲方向

    /***********参数设置***************/
    for(int i=0;i<3;i++){
    strtemp=ui->lineEdit22->text();
    inttemp=strtemp.toFloat();
    ZAux_Direct_SetAccel(g_handle,i,inttemp);//设置加工速度
    ZAux_Direct_SetLspeed(g_handle,i,0);//初始速度为0

    strtemp=ui->lineEdit24->text();
    inttemp=strtemp.toFloat();
    ZAux_Direct_SetAccel(g_handle,0,inttemp);//设置加速度

    strtemp=ui->lineEdit25->text();
    inttemp=strtemp.toFloat();
    ZAux_Direct_SetDecel(g_handle,i,inttemp);//设置减速度

```

```
    ZAux_Direct_SetSramp(g_handle,i,20);//梯形速度

    strtemp=ui->lineEdit28->text();
    inttemp=strtemp.toInt();
    ZAux_Direct_SetAlmIn(g_handle, 0, inttemp);//急停
    }
    /***********X 轴设置***************/
    strtemp=ui->lineEdit4->text();
    inttemp=strtemp.toFloat();
    ZAux_Direct_SetUnits(g_handle,0,inttemp);//设置脉冲当量

    strtemp=ui->lineEdit5->text();
    inttemp=strtemp.toFloat();
    ZAux_Direct_SetFsLimit(g_handle,0,inttemp);//设置正向软限位

    strtemp=ui->lineEdit6->text();
    inttemp=strtemp.toFloat();
    ZAux_Direct_SetRsLimit(g_handle,0,-inttemp);//设置负向软限位

    ui->save->setEnabled(false);
}
```

### 3.2.3 设置 PID 参数

1）设计思想

PID 参数使用 ZMC_Execute 发送 EtherCat 总线的命令 SDO_WRITE 实现，SDO_WRITE 用于写入总线数据。

SDO_WRITE 的参数分别为

① 槽号，设备 0 号对应 EtherCat 总线

② 轴号，设备 X，Y，Z 轴对应 0，1，2 号轴

③ 数据字典，$2005，$2006，$2007 分别对应 P，I，D 参数

④ 子编号

⑤ 数据类型，这里 3 为 int16，可参照《Zbasic》手册

⑥ 数据，要写入的数据

2）程序实现

```
for(int i=0;i<3;i++){
strtemp=lineedit_pid[i]->text();
inttemp=strtemp.toFloat();
QString sdo;
sdo=QString("SDO_WRITE(0,%1,$%2,0,3,%3)").arg(i).arg(2005+i).arg
(inttemp);
//参数分别为（槽号：EtherCat 总线，轴号，数据字典，子编号，数据类型：
int16，
数据值）
QByteArray ba = sdo.toUtf8();
```

```
    char* pchar;
pchar = ba.data();
ZMC_Execute(g_handle,pchar,10,nullptr,0);
    }
```

## 3.3 手动控制

手动控制界面包括 X、Y、Z 的三个轴的手动控制、变频器的控制、轴参数的获取、轴限位、坐标清零和坐标置零。

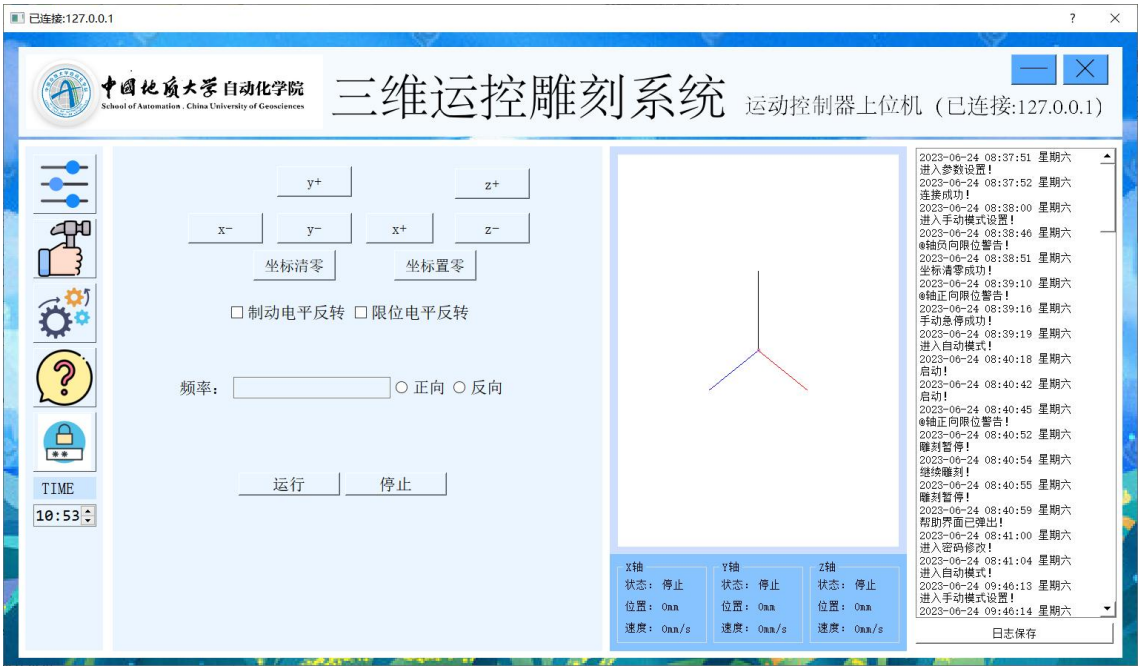


图 3- 16 手动控制界面

### 3.3.1 CNC 手动控制

1）设计思想

通过六个手动控制按键控制 X、Y 和 Z 三个轴的移动，方便进行对刀以及其他移动操作。需要对每个按键写两个槽函数，一个对应于按键按下进行运动，一个对应按键释放停止运动。通过 Qt 信号与槽机制进行按键信号和槽函数的绑定即可。查阅发现下列轴运动函数：

表 3- 2 轴运动函数

| 函数 | 作用 |
| --- | --- |
| ZAux_Direct_Singl_Vmove | 单轴连续运动函数 |
| ZAux_Direct_Singl_Cance | 单轴连续停止函数 |

2）程序实现

对每个按键设置两个 connect（）函数，分别对应按下与抬起，即 pressed（）与 released（）。如下：

```
connect(ui->x_plus, SIGNAL(pressed()), this, SLOT(handcontrol_xplus()));
connect(ui->x_plus, SIGNAL(released()), this, SLOT(handcontrol_xplus_cancel()));
```

对应连接的槽函数如下所示：

```
void Dialog::handcontrol_xplus(){
    ZAux_Direct_Singl_Vmove(g_handle,0,1);
   glwidget->g_mode00or01=0;//手动不画点
}
void Dialog::handcontrol_xplus_cancel();
    ZAux_Direct_Singl_Cancel(g_handle,0,2);
}
```

其中，鼠标左键按下的执行函数调用单轴运动连续函数 ZAux_Direct_Singl_Vmove，鼠标释放时调用 ZAux_Direct_Singl_Cancel 停止运动。

① **ZAux_Direct_Singl_Vmove**

函数功能：单轴指令，连续往一个方向运动。

原型： INT32 __STDCALL ZAUX_DIRECT_SINGL_VMOVE(ZMC_HANDLE HANDLE, INT IAXJS,INT IDIR);

输入参数：

HANDLE 链接标识

IAXIS 轴号

IDIR 方向 -1---负向、1---正向

返回值： 错误码

② **ZAux_Direct_Singl_Cancel**

函数功能：单轴减速停止。

原型：INT32 __STDCALL ZAUX DIRECT_SINGL_CANCEL (ZMC_HANDLE HANDLE, INT IAXIS, INT IMODE);

输入参数： HANDLE

链接标识 IAXIS

轴号 IMODE 模式

0 取消当前运动

1 取消缓冲的运动

2 取消当前运动和缓冲运动

返回值：错误码

### 3.3.2 变频器控制

1）设计思想

驱动控制主轴电动机的变频器使用 Modbus 通讯协议与运动控制器进行通讯。控制电机即是将各个参数传送给下位机，如应该正转还是反转，频率应该是多少，应该运行还是停止。这个操作是通过向运动控制器发送特定的命令来执行的。其正向反向为 radiobutton 控件；其他按键为 pushbutton 控件。其控制使用以下函数：

表 3- 3 主轴控制函数

| 函数 | 作用 |
| --- | --- |
| ZAux_Modbus_Set4x_Long | 设置整形数据到 modbus 字寄存器 |
| ZAux_Modbus_Set0x | 设置 modbus 位寄存器 |

2）程序实现

① 运行模式

按“运行”按钮，则通过 int 类型变量 m_freq 存储设置的频率。bpq_group->isChecked()用来判断正向或反向：如果为 0，则为正向；如果为 1，则为反向，根据设置的方向将频率写入寄存器中。

```
1. void Dialog::on_bpqyunxing_clicked()
2. {
3.   QString pinlv(ui->lineedit_pinlv->text());
4.   int m_freq(pinlv.toInt());
5.   ZAux_Modbus_Set4x_Long(g_handle,3,1,&m_freq);//频率寄存器111
6.   ZMC_Execute(g_handle, "MODBUSM_REGSET(100,1,3)",10,  NULL, 0);
7.   Sleep(150);
8.
9.   int sigChoose = bpq_Group->checkedId();  //选择信号
10.  if(sigChoose==0) {//正向
11.  ZMC_Execute( g_handle, "MODBUSM_REGSET(99,1,0)",10,  NULL, 0);
12.  }
13.  else if(sigChoose==1){//反向
14.  ZMC_Execute( g_handle, "MODBUSM_REGSET(99,1,2)", 10, NULL, 0);
15.  }
16.}
```

a)**ZAux_Modbus_Set4x_Long**

函数功能：设置整形数据到 modbus 字寄存器

原型：int32 __stdcall ZAux_Modbus_Set4x_Long(ZMC_HANDLE handle, uint16 start, uint16 inum, int32 * pidata)

输入参数：

handle 链接标识

start 操作 modbus_ieee 起始编号

inum 写的个数

pidata 数据列表

返回值：错误码

b)**ZAux_Modbus_Set0x**

函数功能：设置 modbus 位寄存器

原型：int32 __stdcall ZAux_Modbus_Set0x(ZMC_HANDLE handle, uint16 start, uint16 inum, uint8* pdata)

输入参数：

handle 链接标识

start 操作 modbus_bit 起始编号

inum 写的个数

pdata 数据列表

返回值：错误码

② 停止模式

按“停止”按钮，则将命令发送至正运动控制器，让变频器停止。

```
1. void Dialog::on_bpqtingzhi_clicked()
2.  {
3.     ZMC_Execute( g_handle, "MODBUSM_REGSET(99,1,1)", 10, NULL, 0);
4. }
```

### 3.3.3 获取轴参数

1）设计思想

要实现实时显示轴的运行状态、速度、位置，需要使用定时器事件。通过定时器时间，更新轴参数，并加入创新功能键盘绑定，具体见 3.6。参数获取使用以下函数：

表 3-4 参数获取函数

| 函数 | 作用 |
|---|---|
| ZAux_Direct_GetIfIdle | 获得轴状态 |
| ZAux_Direct_GetDpos | 获得轴位置 |
| ZAux_Direct_GetMspeed | 获得轴速度 |

2）程序实现

定时器事件函数需要在成员变量中声明一个整型变量，如程序中 m_timerid，m_timerid_3D 这个是定时器标志，用于同时使用多个定时器进行区分，使用

killTimer()和 startTimer()停止和开始定时。程序中，这两个函数放在了连接和断开按钮的槽函数中，只有成功连接后才开启定时器。

```
void Dialog::timerEvent(QTimerEvent *event)
{
    if(m_timerid == event->timerId())
    {
        int idle[3]={1,1,1};
        float fdpos(0);
        float fvspeed(0);
        static bool biaozhi1[3];
        static bool biaozhi2[3];
         for(int i=0;i<3;i++)//x(0) y(1) z(2)
         {
             ZAux_Direct_GetIfIdle(g_handle,i,&idle[i]);//获得轴状态
             if(idle[i])
             {
                 switch (i) {
                 case 0:
                      ui->label_x1->setText("停止");
                     break;
                 case 1:
                     ui->label_y1->setText("停止");
                     break;
                 case 2:
                     ui->label_z1->setText("停止");
                     break;
                 }
             }
             else
             {
                 switch (i) {
                 case 0:
                      ui->label_x1->setText("运行");
                     break;
                 case 1:
                     ui->label_y1->setText("运行");
                     break;
                 case 2:
                     ui->label_z1->setText("运行");
                     break;
                 }
             }
             ZAux_Direct_GetDpos(g_handle,i,&fdpos);//获得轴位置
             switch (i) {
             case 0:
                  ui->label_x2->setText(QString("%1mm").arg(fdpos));
                 break;
             case 1:
                 ui->label_y2->setText(QString("%1mm").arg(fdpos));
```

```
            break;
        case 2:
            ui->label_z2->setText(QString("%1mm").arg(fdpos));
            break;
        }
        ZAux_Direct_GetMspeed(g_handle,i,&fvspeed);
        switch (i) {
        case 0:
         ui->label_x3->setText(QString("%1mm/s").arg(fvspeed));
            break;
        case 1:
         ui->label_y3->setText(QString("%1mm/s").arg(fvspeed));
            break;
        case 2:
         ui->label_z3->setText(QString("%1mm/s").arg(fvspeed));
            break;
        }
}
```

### 3.3.4 轴限位

1）设计思想

使用控件 checkbox 对控件状态进行检测，方便进行制动电平反转与限位电平反转，涉及的运动函数如下：

表 3- 5 轴限位状态相关函数

| 函数 | 作用 |
|---|---|
| ZAux_Direct_SetInvertIn | 设置电平信号翻转 |
| ZAux_Direct_GetIn | 读取电平信号 |

2）程序实现

以制动电平反转问例，if 语句判断 checkbox 的状态。

```
void Dialog::on_xianweifanzhuan_stateChanged(int arg1)
{
    int ionum[6];//需填入 3*6

    ionum[0]=ui->lineEdit7->text().toInt();
    ionum[1]=ui->lineEdit8->text().toInt();
    ionum[2]=ui->lineEdit13->text().toInt();
    ionum[3]=ui->lineEdit14->text().toInt();
    ionum[4]=ui->lineEdit19->text().toInt();
    ionum[5]=ui->lineEdit20->text().toInt();

    if(ui->xianweifanzhuan->isChecked()==1){
        for(int i=0;i<=5;i++){
            ZAux_Direct_SetInvertIn(g_handle,ionum[i],1);
        }
    }
```

```
    else if(ui->xianweifanzhuan->isChecked()==0){
        for(int i=0;i<=5;i++){
            ZAux_Direct_SetInvertIn(g_handle,ionum[i],0);
        }
    }

}
```

### 3.3.5 坐标清零与坐标置零

1）设计思想

坐标清零：将此时存贮坐标位置的变量清零后，通过直线插补函数使轴回归原位。

坐标置零：将当前位置置为新的零点，方便进行对刀操作。直接发送指令，让 MPOS 和 DPOS 置零。

表 3- 6 运动控制函数

| 函数 | 作用 |
| --- | --- |
| ZAux_Direct_MoveAbs | 绝对直线插补运动 |
| ZMC_Execute | 发送参数 |

2）程序实现

① 坐标清零：

```
void Dialog::on_zuobiaoqingling_2_clicked()
{
    float poslist[3]={0,0,0};
    ZAux_Direct_MoveAbs(g_handle,3,poslist);
}
```

② 坐标置零：

```
void Dialog::on_zuobiaozhiling_clicked()
{
    ZMC_Execute(g_handle,"MPOS=0,0,0",10,nullptr,0);
    ZMC_Execute(g_handle,"DPOS=0,0,0",10,nullptr,0);
}
```

## 3.4 自动控制

自动控制界面包含文件导入、启动、坐标清零、对刀点恢复、继续、暂停、对刀点记忆等功能，其功能为读取上位机中的 G 代码文件，控制三维运动平台进行雕刻，并通过 OpenGL 库搭建三维显示窗口进行 3D 坐标显示，绘制运动轨迹。


图 3- 17 自动控制界面设计

### 3.4.1 导入文件

1）设计思想

在进行自动加工工序时，首先需要将待加工 G 代码文件导入到程序中，程序读取 G 代码并将加工过程的中间点存储在数组中，并对 G 代码进行解析控制三维平台。导入文件过程中使用以下函数：

表 3- 7 导入文件函数

| 函数 | 作用 |
|---|---|
| getOpenFileName | 获取整个文件名，打开 tap 文件 |
| QFileInfo | 获取文件信息 |
| fileName | 获取文件名字 |
| suffix | 获取文件后缀 |
| absolutePath | 获取文件绝对路径 |
| isNUll | 检测文件是否打开 |

2）程序实现

获取文件的路径后，逐行将.tap 文件读入缓冲区，并将文件的路径和名称显示出来。

```
1.void Dialog::daoruwenjian()
2.{
3.    ZAux_Direct_Rapidstop(g_handle,0);
```

```
4.//默认值
5.     g_command[0]=0;
6.     g_command[1]=0;//8.3 默认 G00
7.     g_0full=0; g_1full=0; g_mode=90;//默认没有出现 G90/G91,默认坐标移动
       为绝对值
8.
9.     m_aspeed=100.0;
10.     xyz_pos[0]=0.0;//目标点
11.     xyz_pos[1]=0.0;
12.     xyz_pos[2]=0.0;
13.
14.     zhongduan_biaozhi=0;
15.     hangshu=0;
16.     //jindutiao->setValue(0);
17.     ui->labelt->setText(QString::number(0));//已用时间
18.     ui->labelt1->setText(QString::number(0));//剩余时间
19.     ui->label_46->setText(QString::number(0));//显示当前文件行数
20.
21.     ui->textEdit->clear();
22.     QString fileName, filePath,fileSuffix;
23.     QFileInfo fileinfo;
24.     fileFull = QFileDialog::getOpenFileName(this,tr("file"),"/",tr(
       "text(*.tap)"));  //获取整个文件名，打开 tap 文件
25.     //fileFull = E:\QtCode\newExample\myTry\新建文本文档.txt
26.
27.     //获取文件信息
28.     fileinfo = QFileInfo(fileFull);
29.
30.     //获取文件名字
31.     fileName = fileinfo.fileName();
32.
33.     //获取文件后缀
34.
35.     fileSuffix = fileinfo.suffix();
36.
37.     //获取文件绝对路径
38.     filePath = fileinfo.absolutePath();
39.     if(!fileFull.isNull())
40.     {
41.         ui->qidong->setEnabled(true);
42.         ui->zanting->setEnabled(true);
43.         ui->jixu->setEnabled(true);
44.         ui->zuobiaoqingling->setEnabled(true);
45.         ui->huifu->setEnabled(true);
46.         QFile file(fileFull);//通过文件路径，来获取文件
47.         if(!file.open(QFile::ReadOnly ))
48.         {
49.             QMessageBox::warning(this,tr("Error"),tr("read file err
                or:&1").arg(file.errorString()));
50.             return;
```

```
        }
        QTextStream in(&file);

        ui->lineEdit_4->setText(fileFull);

        //逐行读取文件并放入 wen'ben'kuan 文本框
        QTextCodec *codec = QTextCodec::codecForName("GBK");//指定为GBK,因为file.readLine()无法读取中文
        while (!file.atEnd())
        {

            //读取一行文本数据
            QByteArray line = file.readLine();
            //将读取到的行数据转换为Unicode
            QString str = codec->toUnicode(line);       //文件每一行内容
            //qDebug() << str;

            ui->textEdit->insertPlainText(str);//追加放入文本框②

            hangshu++;
        }
        file.close();
        ui->label_46->setText(QString::number(hangshu));//显示当前文件行数

    }
    else
    {
        qDebug()<<"cancel";
        fileFull="No file";
    }
}
```

### 3.4.2 G 代码解译

1）设计思想

G 代码解释器是全软件式数控系统的重要模块。数控机床通常使用 G 代码来描述机床的加工信息，如 走刀轨迹、坐标的选择、冷却液的开启等，将 G 代码解释为数控系统能够识别的数据块是 G 代码解释器的主要功能。G 代码解释器的开放性也是设计和实现中必须要考虑的问题。

译码模块是控制系统中的重要功能模块,将数控加工代码“翻译”成计算机能够识别的语言。该功能模块主要对数控加工代码进行相关的处理。为了提高开发的简便性,采用了通用性比较高的逐字比较法。设计的控制系统 G 代码的译码过程如图 3-18 所示：

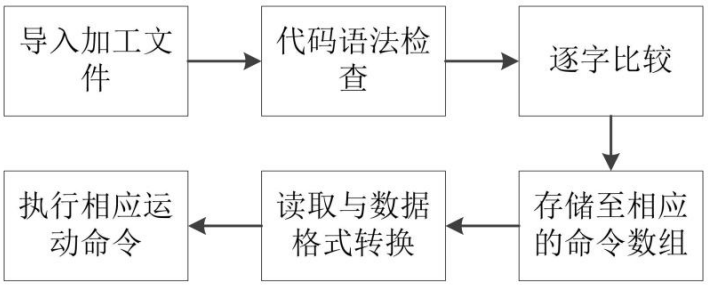


图 3- 18 译码过程流程图

① 将 G 代码读入到相关的程序里，进行代码的检查，检查无误之后，进行逐字比较，生成相应的信息，并将其储存到相应的数组中。

② 将数组中的相关信息读出，进行数据格式转换，完成数据处理后,按照对应的各种命令类型，控制执行器进行加工工作。

③ 每当定时器触发时，若上一条行运动指令执行结束，读取下一行 G 代码，再对该行 G 代码进行解码。其实就是改变对应变量的值，将 G 代码蕴含的信息存入这些变量，之后根据这些变量的值来选择要执行的运动指令。

对于运动指令，先判断该行 G 代码是否包含 G90 或 G91（G90 表示绝对运动，G91 表示相对运动），然后判断是 G00 还是 G01（G00 表示快速移动，G01 表示直线插补，二者都可以用直线插补运动指令），相关函数如下：

| 函数 | 作用 |
|---|---|
| ZAux_Direct_GetIfIdle | 读取当前轴运动状态 |
| ZAux_Direct_SetAtype | 设置轴类型 |
| ZAux_Direct_SetInvertStep | 设定脉冲模式及逻辑方向 |
| ZAux_Direct_SetCreep | 设定爬行速度，用于回零 |
| ZAux_Direct_SetDatumIn | 设定对应轴的原点信号输入口 |
| ZAux_Direct_SetInvertIn | 设置输入信号翻转 |
| ZAux_Direct_Singl_Datum | 单轴回零 |

2）程序实现

只显示代码解析部分，其余部分较长，在此不做说明。

```
1.void Dialog::Readcode_G()  //带未选择文件检测
2.{
3.            for(;*fileat;fileat++)//依次对各个字符进行解析
4.            {
5.                switch(*fileat)
6.                {
7.                case 'G':
8.                    if(g_0full==0)
9.                    {   g_command[0] = atoi(fileat+1);g_0full=1;}
10.                    else
```

```
                    {   g_command[1]= atoi(fileat+1); g_1full=1;}
                    break;
                case 'F':
                    m_aspeed  = atof(fileat+1);
                    break;
                case 'X':
                    xyz_pos[0] = atof(fileat+1);
                    break;
                case 'Y':
                    xyz_pos[1] = atof(fileat+1);
                    break;
                case 'Z':
                    xyz_pos[2] = atof(fileat+1);
                    break;
                default: break;
                }
            }
    }
```

### 3.4.3 三维轨迹图绘制

1）设计思想

为了实时显示雕刻机末端运动的轨迹，采用 OpenGl 库搭建三维显示窗口，根据末端的移动坐标绘制运动轨迹。

绘制步骤如下：

① 初始化画布：背景颜色、大小、坐标轴等设置，使用函数 initializeGL()，resizeGL()，drawCoordinate()

② 画布更新：用恒等矩阵替换当前矩阵、设定变换视点，使用函数 paintGL()，RotateViewPoint()；设置鼠标拖动角度、设置缩放倍数，使用函数 mousePressEvent()，mouseReleaseEvent()，mouseMoveEvent()，wheelEvent()；画点使用函数 DrawShape()

③ 每当定时器触发，开始向指定数组中存入当前位置，随后开始更新画布，每次更新都会画出数组中所有点并连成线，不同运动方式的点有不同的画法。最终，画布上的轨迹线是在直线插补过程中的点炼成的线，在快速移动时只显示当前点，并不画出轨迹线。

表 3- 8 三维轨迹图相关函数

| 函数名称 | 功能 |
| --- | --- |
| killTimer | 停止计时 |
| startTimer(a) | 开始计时，a 为计时器触发间隔时间 |
| QGLWidget | 初始化界面 |
| initializeGL | 初始化背景颜色 |

| drawCoordinate | 画坐标轴 |
|---|---|
| RotateViewPoint | 用鼠标旋转视点 |
| paintGL | 用恒等矩阵替换当前矩阵 |
| mousePressEvent | 鼠标拖动角度（点击时） |
| mouseReleaseEvent | 鼠标拖动角度（释放时） |
| mouseMoveEvent | 计算鼠标拖动角度 |
| wheelEvent | 鼠标滚轮缩放函数 |
| DrawShape | 应该按照笛卡尔坐标系，即先 y 再 z 再 x 画点 |
| RotateViewPoint() | 设定变换视点 |

2）程序实现

① 三维图清空：

```
void Dialog::zuobiaoqingling()
{
    float poslist[3]={0,0,0};
    ZAux_Direct_MoveAbs(g_handle,3,poslist);

    //三维图清空，（之后从当前点开始走）
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);//clear=1
    glwidget->points_x.clear();
    glwidget->points_y.clear();
    glwidget->points_z.clear();
    glwidget->xyz_mode.clear();
}
```

② 手动控制不生成轨迹：

```
void Dialog::handcontrol_xplus(){
    ZAux_Direct_Singl_Vmove(g_handle,0,1);
   glwidget->g_mode00or01=0;//手动不画点
}
```

在使用 OpenGL 画图时，首先要判断程序当前是在快速移动还是直线插补，快速移动时只显示当前点，并不画出轨迹线。该判断是在读取 G 代码时运行的。

定时器刷新：程序调用了 QT 定时器，在连接成功时使用 m_timerid_3D=startTimer(8)启动定时器，断开连接时使用 killTimer(m_timerid_3D)关闭定时器。并将定时器的每次触发事件定为 8ms 和 9ms，当定时器触发时程序自动进入 timerEvent 函数，在 timerEvent 函数中采集 xyz 的坐标、判断当前运动状态并刷新画布。

```
 switch(i)  {
          case 0:glwidget->points_x.push_back(fdpos/1000);break;
          case 1:glwidget->points_y.push_back(fdpos/1000);break;
          case 2:glwidget->points_z.push_back(fdpos/1000);
                //xyz 坐标采集结束，判断其在快速移动还是直线插补
                switch( glwidget->g_mode00or01)
                {
      case 0:glwidget->xyz_mode.push_back(0); qDebug()<<0;break;
```

```
9.
10.         case 1:glwidget->xyz_mode.push_back(1); qDebug()<<1;break;
11.
12.          default:break;
13.         }
14.          break;
15.}
```

通过定时器的启动，程序几乎能够实时更新当前 xyz 轴的坐标，程序之后会将得到的坐标放入 DrawShape 函数中画出，若在直线插补中则将得到的点连成线。因此每次更新都会画出数组中所有点并连成线，不同运动方式的点有不同的画法。最终，画布上的轨迹线是在直线插补过程中的点炼成的线，在快速移动时只显示当前点，并不画出轨迹线。

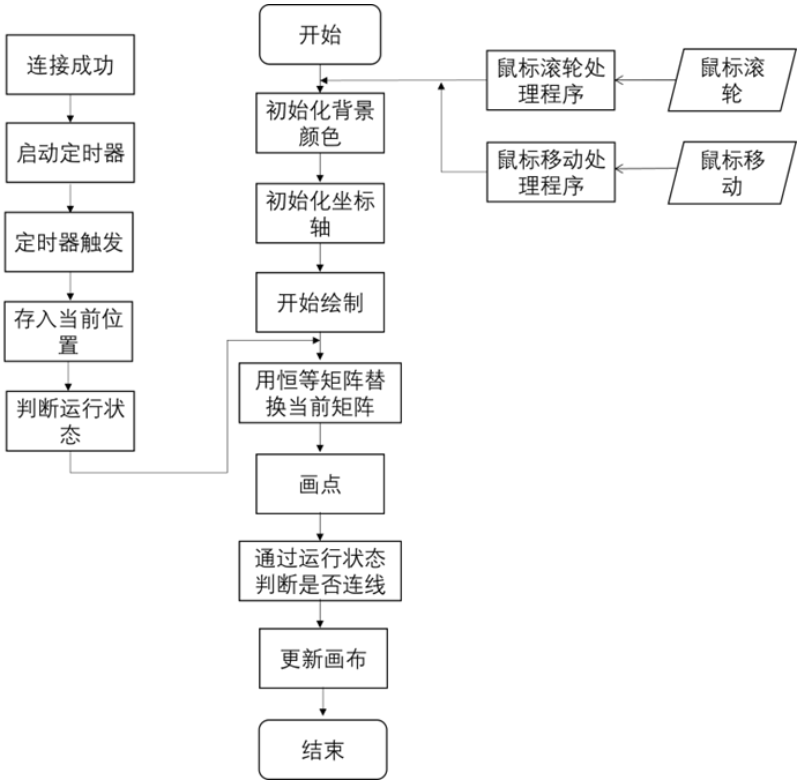


图 3- 19 三维绘画流程图

### 3.4.4 进度显示

1）设计思想

此功能的实现需要定义两个关键控件，分别为显示进度的 progressbar 以及显示代码的 textedit；

Progressbar 的使用非常简单，根据其内置的 setvalue 函数设置 0-100 的数字就可代表其进度，我们在运行时使用当前（运行行数/总行数）×100 即可得出当前进度，再通过定时器刷新到 UI 界面。

Textedit 控件需显示所有代码，并将执行过的代码上色。首先打印所有代码，

其在执行代码的循环中，每执行一行代码，通过 setForeground 函数给本行文本上色即可。

其使用的函数如下：

| 函数 | 作用 |
| --- | --- |
| setvalue | 显示 progressbar 数值 |
| setForeground | Textedit 指定行数上色 |
| setTextColor | Textedit 所有文本上色 |

2）程序实现

```
1. ui->labelt->setText(QString::number(iiiii));//已用时间
2. ui->labelt1->setText(QString::number(hangshu-iiiii));//剩余时间
3. ui->progressBar->setValue((iiiii/hangshu)*100);
```

### 3.4.5 附加功能

表 3- 9 附加功能函数

| 函数 | 作用 |
| --- | --- |
| qidong() | 启动 |
| Pause() | 暂停 |
| Continue() | 继续 |
| zuobiaoqingling() | 坐标清零 |
| duidaodianjiyi() | 对刀记忆 |
| duidaodianhuifu() | 对刀恢复 |

1）启动

使用函数 qidong() ，在 qidong()中，最重要的就是设置定时器中断标志位，当标志位设置为 1 时，开始逐行解析 G 代码，为 0 时，不解析 G 代码。

2）暂停/继续

使用函数 Pause()、Continue() ，在 Pause()、Continue()中，最重要的是调用暂停、继续对应的运动指令，暂停对应指令 ZAux_Direct_MovePause，作用是暂停当前运动（并不是直接取消当前运动），继续对应指令 ZAux_Direct_MoveResume，继续之前暂停的运动。两条指令配套使用。注意在暂停的过程中，会有减速过程，设置合理的加减速度即可忽略这个惯性过程

3）坐标清零+清屏

在 zuobiaoqingling()中，清除了存放坐标点的数组，并且通过绝对直线插补运动到零点。

4）对刀点记忆/恢复

在进行自动加工过程中，可能会由于意外情况的发生需要暂停运行，此时就需要将当前的坐标点记忆，便于后期对刀点恢复重新开始继续加工。通过

duidaodianjiyi()，将当前坐标点记忆到一个数组中；通过 duidaodianhuifu()，在准备恢复对刀点时，系统从数组中读取记忆的坐标点，重新设置主轴和插补运动的参数进行自动加工。

设计的运动控制函数如下：

表 3- 10 运动函数

| 函数 | 作用 |
| --- | --- |
| ZAux_Direct_MoveAbs | 绝对直线插补运动 |
| ZAux_Direct_GetDpos | 获得轴位置 |

程序如下：

```
void Dialog::duidaodianjiyi()
{
    float fdpos(0);
    for(int i=0;i<3;i++)
    {
        ZAux_Direct_GetDpos(g_handle,i,&fdpos);//获得轴位置
        daodianjiyi[i]=fdpos ;
    }
}
void Dialog::duidaodianhuifu()
{
    ZAux_Direct_MoveAbs(g_handle,3,daodianjiyi);
}
```

## 3.5 帮助功能

为了使软件更加的人性化和面向对象开发的性质。设计了参考说明的功能。当人们使用该软件时，若遇到难以解决的问题或者不会操作时，点击参考说明可以需求帮助，在参考说明中我们介绍了软件参数设置功能、自动模式功能、手动模式功能的使用方法。

**三维运控上位机操作说明**

**参数设置:**

(1)连接:①使用以太网连接电脑和 ZMotion 运动控制器，运动控制器默认 IP 为：192.168.0.11

②连接到仿真器时，仿真器默认 IP 为：127.0.0.1

(2) PID 参数: Kp、Ki、Kd 的默认值分别为：40、20、1（建议不动）

(3)X、Y、Z 轴参数设置：可设置三个轴的脉冲当量、正负限位、正负限位 IO、回零 IO;

(4)刀具参数设置，包括：加工速度、最大速度、加速度、减速度、过渡时间、

爬行速度、急停 IO

(5)各参数设置完毕后，点击“保存设置”按钮，保存更改值。

**自动模式:**

使用自动雕刻的步骤：

①设定主轴转速和方向，启动主轴；

②使用手动模式进行对刀，将刀尖对在 ArtCam 中 3D 模型的原点（图形左下角）；

③坐标置零；

④进入自动模式，点击”导入文件“按钮，选择 G 代码文件，等待导入完成，点击”启动“按钮;

⑤加工过程中可点击”暂停“按钮，XYZ 轴即暂停运动，点击”继续“可继续加工。

若要重新雕刻，则点击”暂停“，然后重复步骤②~④。

**手动模式:**

(1)刀具运动控制:

①上位机界面上点击上、下、左、右、前、后，6 个按钮即可实现对应的轴的对应方向运动；

②键盘 W、S、A、D、Q、E 分别对应 Y+、Y-，X+，X-，Z+，Z- ，跟界面中的对应按键实现相同功能;

③”坐标置零“按钮可将当前坐标清零，当前坐标置为坐标原点（0，0，0）；

(2)变频器的启停控制:

①在“频率”框中输入变频器频率，选择”正向“或”反向“，最后点击”运行“，主轴即启动。

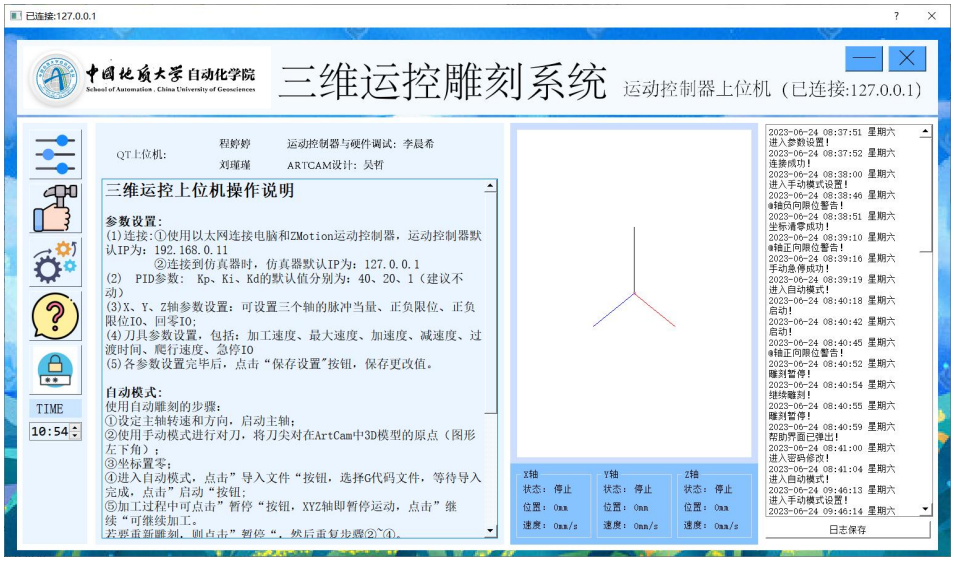


图 3- 20 帮助界面设计

## 3.6 创新功能

本次实践中，针对程序时间计算问题做了针对性改进；为方便查看功能使用情况和方便故障检测加入日志功能；另外，添加了键盘输入、时间显示、登录界面、密码修改等功能。

### 3.6.1 登录

1）设计思想

设计 Qt 登录界面，通过 QsqlDatabase 和 QsqlTableModel 读取数据库存储账号和密码，方便进行账户管理，提高安全性。

2）程序实现

① 使用 Qt 设计师界面类 Dialog，编写登录界面类 page_login。

```
class page_login :public QDialog
{
    Q_OBJECT

public:
    explicit page_login(QWidget *parent = nullptr);
    ~page_login();
private:
        bool myconnect();
private slots:
    void on_btn_login_clicked();
    void on_btn_exit_clicked();
    void paintEvent(QPaintEvent*);

private:
    QString name;
    QString pwd;
    Ui::page_login *ui;
};
```

② 连接数据库函数 myconnect()：

数据库打开成功，返回 1；否则，返回 0。

```
bool page_login:: myconnect1(){

    QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
    db.setDatabaseName("student.db");//数据库名字
    if (!db.open()) {
    QMessageBox::critical(nullptr, QObject::tr("Cannot open data
    base"),
     QObject::tr("Unable to establish a database connection.\n"
          "This example needs SQLite support. Please read "
         "the Qt SQL driver documentation for information how "
                  "to build it.\n\n"
```

```
                "Click Cancel to exit."), QMessageBox::Cancel);
        return false;
    }
   //qDebug()<<"lianjiechengg"<<endl;
    return true;
}
```

③ 登录按钮函数：

首先通过 myconnect 进行数据库的连接，然后读取此时登录界面输入的账号和密码，并存储在全局变量 suser 和 spwd 中，方便后续进行密码修改。

QSqlTableModel 模型不需要书写 SQL 语句，利用封装好的函数可以实现对数据的编辑、插入和删除等操作，并实现数据的排序和过滤，在提交修改数据库内容时会自动进行约束条件的检查。

数据库连接成功后，通过 QsqlTableModel 单表模型的 select()函数进行查询，使用 rowCount（）函数返回查询结果的行数，由于数据库的实体完整性，无需担心重复定义问题，如果存在匹配项，则查询到的表存在一行，则返回 1。

```
void page_login::on_btn_login_clicked()
{
    //数据库查找用户名和密码
    myconnect();
    name=ui->le_user->text();
    pwd=ui->le_password->text();
    Sname(name);
    Spwd(pwd);
    QSqlTableModel model;
     model.setTable("password");
     model.setFilter(QObject::tr("Sno='%1' and pwd='%2'").arg(name).arg(pwd));
     model.select();
     //qDebug()<<model.select()<<endl;
    // qDebug()<<model.rowCount()<<endl;
    if(model.rowCount()==1) //成功进入主界面
            {
                QMessageBox::information(this, tr("提示"), tr("登录成功"));
                accept();
            }
     else   //失败可以重新输入
            {
                QMessageBox::warning(this, tr("warn"), tr("用户名或者密码不正确"));
                ui->le_user->clear();
                ui->le_password->clear();
                ui->le_user->setFocus();
            }
}
```

④ 退出按钮函数

```
void page_login::on_btn_exit_clicked()
{
    exit(0);//退出登录界面
}
```

⑤ 主函数修改

若登录成功，则登录界面会返回 QDialog::Accepted 信号，可跳转到主界面。

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
   page_login l;
    l.show();
    if(l.exec()==QDialog::Accepted)
    {
        QFile styleFile(":/qss/mystyle.qss");
        if(styleFile.open(QIODevice::ReadOnly))
        {
            QString setStyleSheet(styleFile.readAll());
            a.setStyleSheet(setStyleSheet);
            styleFile.close();
        }

        Dialog w;
        w.show();
        return a.exec();
    }
    else
    {
        return 0;
    }

}
```

### 3.6.2 密码修改

1)设计思想

Qt 密码修改、新旧密码规则检索功能，连接数据库，修改数据库内容，规则如下：

① 输入旧密码、新密码和再次输入新密码新密码

② 判断旧密码是否输入错误

③ 判断新密码是否一致，不一致弹出警告框，将新密码文本框清空

④ 判断新旧密码是否一致，一致弹出警告框，将新密码文本框清空

⑤ 若无上述情况，则修改成功

2）程序实现

① password 类设计

```
class password : public QDialog
{
    Q_OBJECT

public:
    explicit password(QWidget *parent = nullptr);
    ~password();

private slots:
    void on_pushButton_clicked();
    void paintEvent(QPaintEvent*);
    void on_pushButton_2_clicked();

private:
    Ui::password *ui;
};
```

② “保存”按钮函数

```
void password::on_pushButton_clicked()
{
    QString pwd=ui->lineEdit_1->text();
    QString npwd1=ui->lineEdit_2->text();
    QString npwd2=ui->lineEdit_3->text();
    if(pwd!=spwd){
        QMessageBox::warning(this, tr("warn"), tr("旧密码输入错误"));
        ui->lineEdit_1->clear();
        ui->lineEdit_1->setFocus();
    }
    else if(npwd1!=npwd2){
        QMessageBox::warning(this, tr("warn"), tr("新密码不一致"));
        ui->lineEdit_3->clear();
        ui->lineEdit_3->setFocus();
    }
    else if(npwd1==spwd){
  QMessageBox::warning(this, tr("warn"), tr("新旧密码不能完全一致"));
        ui->lineEdit_2->clear();
        ui->lineEdit_3->clear();
        ui->lineEdit_2->setFocus();
    }
    else {
        QMessageBox::information(this, tr("提示"), tr("修改成功"));
        Spwd(npwd1);
        qDebug()<<spwd;
        this->hide();
    }
}
```

③ 密码修改函数

通过 QSqlQuery 使用 SQL 语言 update 对数据库中账号的密码进行更改。

```
void Dialog::on_btn_changepassword_clicked()
{
    password p;
    p.show();
    p.exec();
    QSqlQuery qr;
    //qDebug()<<spwd;
    qr.prepare("update password set pwd=? where Sno=?");
    qr.addBindValue(spwd);//可以多个
     qr.addBindValue(suser);
     qr.exec();
}
```

3）功能展示

在上位机修改登录密码，如图 3-21 所示：


图 3- 21 密码修改

数据库中修改前后结果如下：

| | Sno | pwd |
|---|---|---|
| | 过滤 | 过滤 |
| 1 | 20201003740 | 003740 |
| 2 | 20201003544 | 003544 |
| 3 | 20201003596 | 003596 |
| 4 | 20201000000 | 000000 |
| 5 | 20201000128 | 000128 |

| | Sno | pwd |
|---|---|---|
| | 过滤 | 过滤 |
| 1 | 20201003740 | 003740 |
| 2 | 20201003544 | 003544 |
| 3 | 20201003596 | 003596 |
| 4 | 20201000000 | 000000 |
| 5 | 20201000128 | 000000 |

图 3- 22 数据库密码修改

### 3.6.3 日志设计

1）设计思想

使用控件记录各个功能使用情况，方便查看功能使用情况并进行故障检测。

声明一个 string 类型全局变量，用于储存日志。

2）程序实现

首先获取当前时间，记录当前时间后，增加所需记录的功能，最后在“text_log”中显示出。

① 功能记录（下面为主要实现代码）

```
QDateTime time = QDateTime::currentDateTime();
QString qstr = time.toString("yyyy-MM-dd hh:mm:ss dddd");
text += qstr.toStdString();
text +="\n 进入参数设置！\n";
ui->text_log->setText(QString::fromStdString(text));
```

② 可将日志保存至 txt 文件：若日志文件存在，则直接写入；若日志文件不存在，则自动生成后写入。

```
void Dialog::on_savelog_clicked()
{
    QFile file;
    file.setFileName("D:/桌面/日志.txt");
        if(file.open(QIODevice::WriteOnly )){
            const char* res3 = text.c_str();//char*
            file.write(res3);
            file.close();
        }
}
```

3）功能展示

① 上位机界面


图 3- 23 日志界面

② “日志.txt”文件(部分，全部见附录)


图 3- 24 日志.txt 文件

### 3.6.4 时间计算

1）存在问题

例程在计算代码运行时间及剩余时间时存在许多问题：

① 其实际为行数计算，并没有计算时间

② 原算法写在执行 G 代码中，刷新率较低

2）设计思想

首先定义两个全局变量，存放当前时间与初始时间，时间的获取采用 QDtatetime 对象的 currentDateTime 函数获取，其内部包含日期、时刻等多种数据，通过计算当前已执行时间推算剩余时间，按照时间同代码成一定比例的前提下，通过代码的总量求得全部要用的时间，这样通过已完成的代码不断求解已用时间，可以进行实时的更新，设计 timerUpdata 更新时间。

定义一个新的定时器，设定为每 1s 触发一次时间更新的函数，通过计算已执行的时间，推算剩余行数计算所需的时间。在面对暂停、继续的问题时，需设置一个标志位，在暂停使其变化时将时间也记录下来，在后续的计算中利用数学方法进行剔除，防止暂停时间对于整体计算时间的影响。

3）程序实现

```
void Dialog::timerUpdata(){
        //获取系统时间
        QDateTime sysTime3 = QDateTime::currentDateTime();

        //获取时分秒以“:”号拆分赋予 list 数组
     QStringList list = sysTime3.toString("hh:mm:ss").split(':');
        //将时分秒绑定控件
     ui->timeEdit->setTime(QTime(list[0].toInt(),list[1].toInt(),
     list[2].toInt()));

if(biaozhi == 1){

        QString str2;
        QDateTime Time = QDateTime::currentDateTime();

        if(str2 != Time.toString("ss")){
        str2 = Time.toString("ss");
        if(syytime == 0){syytime = str2.toInt();}
        else{
  if(str2.toInt() < syytime){sytime += 60-syytime+str2.toInt();}
            else{sytime += abs(str2.toInt()-syytime);}
            syytime = str2.toInt();
        }
}

}
        shi = (sytime-(sytime % 3600))/3600;
        fen = ((sytime % 3600)-(sytime % 3600)%60)/60;
        miao = sytime - shi*3600 - fen*60;
        QString str3 = QString::number(shi);
        str3.append(":");
        str3.append(QString::number(fen));
        str3.append(":");
        str3.append(QString::number(miao));
        ui->labelt->setText(str3);//已用时间
        shi = (shengyutime-(shengyutime % 3600))/3600;
        fen = ((shengyutime % 3600)-(shengyutime % 3600)%60)/60;
        miao = shengyutime - shi*3600 - fen*60;
        QString str4 = QString::number(shi);
        str4.append(":");
        str4.append(QString::number(fen));
        str4.append(":");
        str4.append(QString::number(miao));
        ui->labelt1->setText(str4);//剩余时间
}
```

### 3.6.5 键盘手动控制

1）存在问题

鼠标按按钮无法在对刀细调式，难以控制刀具位置。

2）设计思想

通过上位机键盘直接控制刀具移动，在 Qt 中使用键盘的控制实现较为简单：

方法一：在 pushbutton 对象中有一变量为 shortcut，可直接对其修改绑定键盘按键。

方法二：使用 setShortcut 函数即可。例：

```
m_0->setShortcut(QKeySequence(Qt::Key_0));
```

设置电脑键盘“A”、“D”、“W”、“S”、“J”和“K”分别进行三个轴的移动。可实现按键对应的功能，有按下运行、松开停止的效果。此处采用方法一，在 Qt 的 ui 界面中进行设置即可。

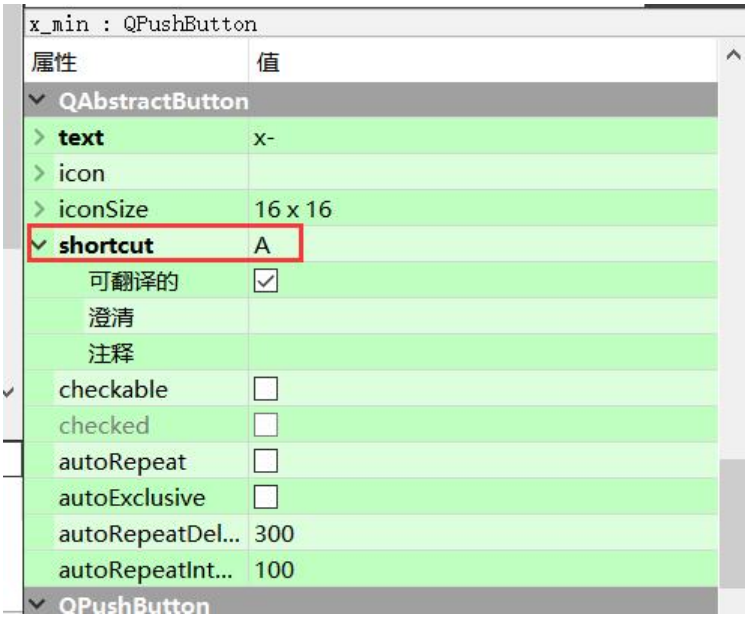


图 3- 25 键盘绑定设置

### 3.6.6 自动停止

1）设计思想

在完成文件所有雕刻后，主轴并未停止旋转，加入停止程序便可解决。增加 else if 语句，当计数 iiiii 大于行数，便下达停止指令。

2）程序实现

```
else if (iiiii>hangshu) {//将主轴停止
    ZMCExecute( ghandle, "MODBUSMREGSET(99,1,1)", 10, NULL, 0);
}
```

# 第四章 雕刻过程

## 4.1 制作 G 代码

ArtCAM 软件系列可以把如手绘稿、扫描文件、照片、灰度图、CAD 等文件一切平面数据，转化为生动而精致的三维浮雕数字模型，并生成能够驱动数控机床运行的代码。ArtCAM 包括了丰富的模块，这些模块功能齐备，运行快速，性能可靠，极富创造性。利用 Delcam ArtCAM 所生成的浮雕模型，可通过并、交、差等布尔运算，任意组合、叠加、拼接等产生出更复杂的浮雕模型。并且可以渲染处理设计完毕的浮雕。使用软件 ArtCAM 将图片转换成 G 代码。

### 4.1.1 设计步骤

1）打开 ArtCAM 文件，选择“新建项目”，将图形(.jpg .png 等)拖入项目中；

2）设置模型尺寸，选择“图像尺寸”，根据亚克力板尺寸(135mm×100mm×25mm)选择所雕刻图形的尺寸。

3）在右侧“项目”的位图中选中图片，右键“产生浮雕”，选择所雕刻图形的高度。

4）点击“三维查看”可看到图形的三维视图，可以看出图形表面并不光滑，因此选用“光顺浮雕”，并设定光顺次数为 2 次。

5）点击“刀具路径”，选择“产生加工浮雕刀具路径”，对于本次实习所使用的刀具，其具体参数设置如下：

| 参数 | 数值 |
|---|---|
| 刀具类型 | 平底刻刀 |
| 直径(D) | 3.175 |
| 半角(A) | 15 |
| 平底半径(F) | 0.2 |
| 下切步距 | 3.0 |
| 行距 | 0.24 |
| 主轴转速(转/分) | 10000 |
| 进给率(毫米/秒) | 10.0 |
| 下切速率(毫米/秒) | 8.0 |

6）选用所设定的刀具，对于本次实习，在“刀具区域清除策略”中选择 X 和 Y 方向平行加工(传统)或螺旋加工(传统)，并定义材料的厚度为 25 毫米。

7）选择“刀具路径”中的动态仿真，即可看到刀具在玻璃板上的雕刻路径。

### 4.1.2 注意事项

1）图片越高清，雕刻效果越好，但 G 代码同样会变长，设置大小时，设置长宽越长，Z 方向深度越深则代码行数越多。

2）可以通过 ArtCAM 中的“光顺”使图片变平滑。

3）一般用“X 和 Y 方向加工（传统）”。

4）选择刀具时，行距越短，雕刻越精细，但 G 代码同样会变长。

5）保存时，选择保存格式为 G-code（*.tap），以 mm 为单位，设置脉冲当量为 26214.4，此时和 G 代码单位吻合。

## 4.2 雕刻前软件准备

1）控制器连接

Pc 与控制器链接时，注意连接 ETHERNET 口。同时在配置 PC 网口，设置->网络和 Internet->状态->更改适配器选项，选择所连接的网口，右键属性，修改成下图所示。

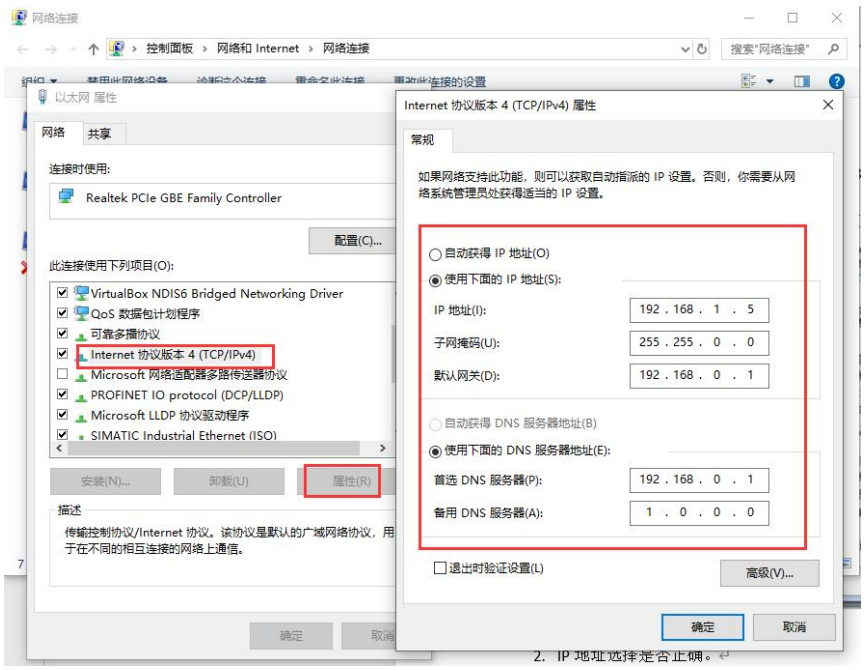


图 4- 1 控制器连接

在上位机程序中，设置各个轴的轴类型为 65 ，与仿真器通信时选择 0 或 1。

```
1. //设定轴类型 1-脉冲轴类型
2. ZAux_Direct_SetAtype(g_handle,0, 65);//轴号：0/
```

在上位机中链接运动控制器时，选择 IP 地址为 192.168.0.11


图 4- 2 运动控制器 IP 连接

2）参数设定

脉冲当量设为 pow(2.0,17)/5.0，此时坐标的单位 mm，速度的单位为 mm/s 运行速度不超过 10mm/s，在参数设定界面 xyz 轴的脉冲当量均为 26214.4。

3）限位设置

首先开启限位，打开 ZDevelop，连接到控制器，然后使用手动运动分别测试 X、Y、Z 三个轴正负方向的限位 IO 点。其次，限位参数设置要参考实际雕刻作品的大小，防止在雕刻过程中出现限位报警。

4）G 代码浏览

大致浏览 G 代码，确定 X,Y,Z 的大致范围，防止对刀零点确定错误，发生撞刀。

## 4.3 雕刻前硬件准备

### 4.3.1 变频器操作

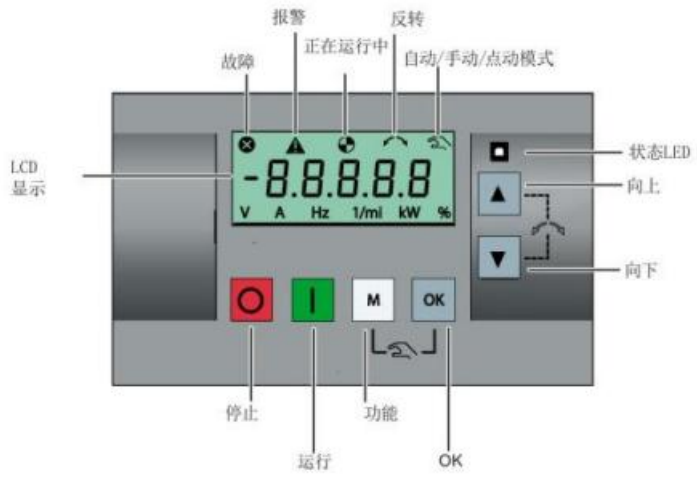


图 4- 3 变频器控制面板

1）常见操作：

①“M”+“OK”——自动、手动、点动 3 个模式间循环切换

②“OK”——确认键，进入数值编辑模式或换至下一位；还可清除故障

③“M”短按——相当于回退键，回到上一级内容

④“M”长按——显示主轴电机当前转速对应的频率

⑤“〇”——在手动模式下，可使主轴电机停车

⑥“I”——在手动模式下，可启动电机，使之运行在给定频率

⑦“ ▲”和“ ▼”——上下滚动菜单

2）变频器的重要参数设置

表 4- 1 变频器参数设置

| 参数 | 含义 | 推荐数值 | 单位 |
|---|---|---|---|
| P0003 | 用户访问级别 | 3（专家） | |
| P0100 | 50/60Hz 选择 | 0（50Hz） | Hz |
| P0304 | 额定电压 | 220 | V |
| P0305 | 额定电流 | 5.0 | A |
| P0307 | 额定功率 | 0.8 | kW |
| P0308 | 功率因数 | 0.82 | |
| P0310 | 额定频率 | 400 | Hz |
| P0311 | 额定转速 | 240000 | rpm |
| P2000 | 基准频率 | 100 或 200 | Hz |

① **P0003 用户访问级别**要设为 3（专家级别），设低了的话后面一些参数可能无法访问和修改。

② **P0310 额定频率**，这里指的是主轴电机额定转速（24000r/min）所对应的电源频率，根据公式：

$$n_1 = \frac{60 f_1}{p}$$

其中：$n_1 = 24000$, $p = 1$

则可算出 $f_1 = 400$，P0310 要设为 400Hz。

③ **P2000 基准频率**，主轴电机的运行转速等于上位机输入的频率 $f_{in}$ 与 16 进制的 4000（也就是 10 进制的 16384）的比值，乘以基准频率 P2000，公式如下：

$$\text{主轴电机电源频率：} f[Hz] = \frac{f_{in}(\text{Hex})}{4000(\text{Hex})} \cdot P2000 = \frac{f(\%)}{100\%} \cdot P2000$$

式中：$f_{in}$是上位机中输入的频率（16进制）

$$\text{主轴电机转速}(r/\min): \text{n} = \frac{60 f}{p} \quad (\text{电机最高转速24000r/min})$$

在实验中一般避免使电机运行负荷太重，因此 P2000 推荐设置为 100~200

之间。

### 4.3.2 换刀

当刀具磨碎大或者出现断刀时需要换刀，换刀的时候要注意安全，要将刀装在合适的位置，如果刀头太短，在雕刻时容易触发门限位，导致雕刻作品失败。

### 4.3.3 整平钳台

当待加工的图形边框是矩形时，如果图形的边框与亚克力板的边缘不平行，会导致最终作品偏离中心，因此需要在开始雕刻前调节钳台四个角的固定螺丝，使钳台边缘平行于基床的导轨，可使用直角尺辅助。

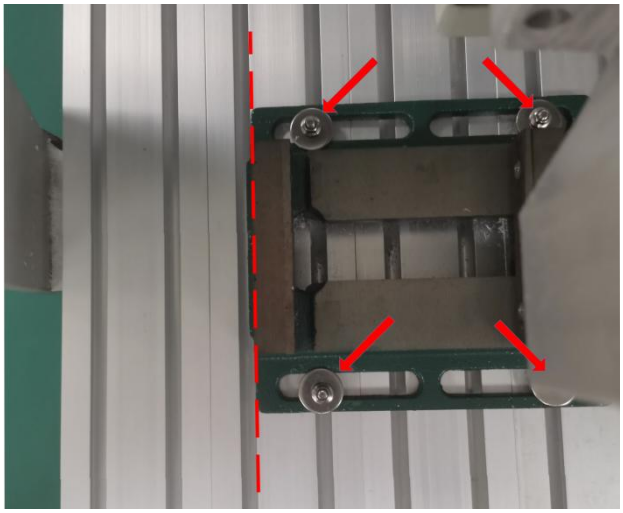

图 4- 4 钳台整平图

### 4.3.4 装夹工件

在装夹之前，为使作品在材料四周的留白对称美观，可先计算好 x、y 方向的留白尺寸，使雕刻图形尽量在板子的正中央，然后用尺子量取原点位置，并用记号笔标记“+”。夹装时注意使整个工件底部与钳台紧密接触，确保没有夹得一高一低，否则最后雕刻作品就会一边深一边浅。如果工件底面与钳台紧密贴合了也还是不平，可在工件底部垫纸解决。

## 4.4 操作设备开始雕刻

### 4.4.1 空跑

在正式雕刻前，Z 轴先调高一点，在板子上方空跑，观察大致雕刻范围、是

否会出现限位以及刀具的运动轨迹等，以确保正式雕刻的顺利完成。

### 4.4.2 对刀

在自动控制启动前要进行对刀，首先开启主轴，手动移动刀具位置，先调 X，Y，视线与预设原点平齐，再调 Z，直至刀尖即将与工件接触，有少许切削即可。

### 4.4.3 调节主轴转速

主轴转速的选取，既不能过低，也不能过高，保持合适的范围，才能平稳加工工件。主轴转速低容易导致亚克力融化，增大主轴转速有以下好处：

1）增加主轴转速会导致更多的切削热的产生：切削热对刀具的使用寿命有非常大的影响。

2）增大主轴转速可以减小每齿每转吃刀量：每齿每转吃刀量=进给速度/（主轴转速*齿数），减小每齿每转吃刀量可以减小切削力，使加工平稳。

3）增大主轴转速可以提高主轴电机的输出功率：从我们的主轴电机功率曲线中可以看出，转速越高，输出功率越大，这样加工越平稳。

总的来说，主轴转速高一些有好处，但是一定要在合理的转速范围内。

## 4.5 雕刻作品展示

### 4.5.1 作品一：花


图 4- 5 作品一

### 4.5.2 作品二：大展鸿图


图 4- 6 作品二：大展鸿图

### 4.5.3 作品三：海绵宝宝

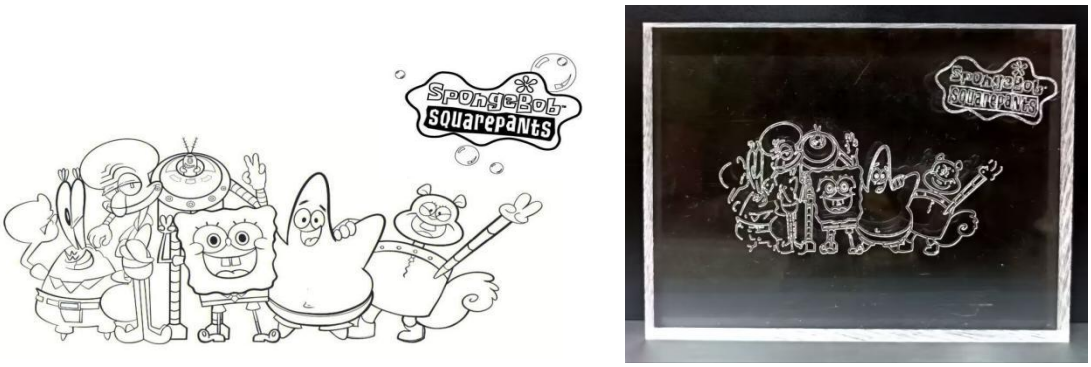


图 4- 7 作品三：海绵宝宝

## 4.6 问题解决

1）问题一：图形过于复杂，细节过多，尺寸比较小，成品分辨率低。

解决方法：换更简单的图形，并将图形尺寸调大。

我们原先准备雕刻一幅山水画，但由于山水画的细节过多、图形复杂且当时选的尺寸比较小，最终雕刻出的成品分辨率低、不清晰。因此，我们后续选择花

这副作品。

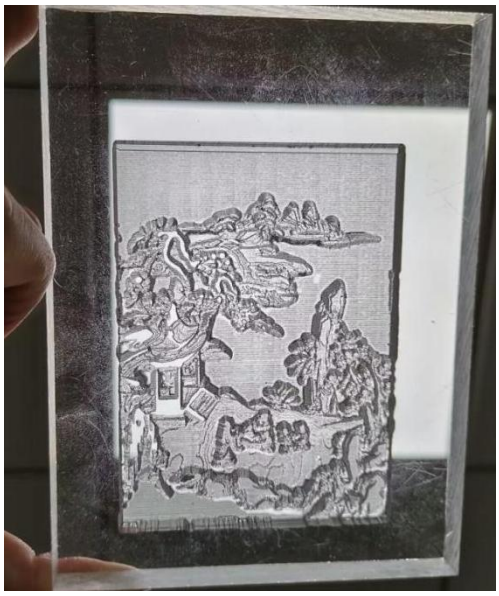

图 4- 8 山水画

2）问题二：变频器报警“F11”，电机过热。

解决方法：打开舱门加快散热，按“OK”键进行“故障响应”。

3）问题三：在 20° 刀具更换为 30° 刀具后，整个雕刻面十分粗糙，且有亚克力碎屑残留。故判断是刀具参数设置与实际刀具参数不匹配导致的。

解决方法：重新设置软件刀具参数，生成 G 代码。要注意，软件中设置的为刀具的半角度数和半间距。

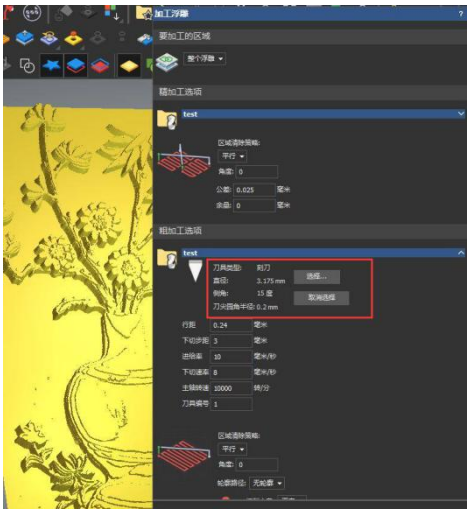


图 4- 11 G 代码参数设置

4）问题四：刀具断裂，雕刻停止

解决方法：可在原雕刻进度的基础上重新定点，删除 G 代码前半段，继续

雕刻。深层原因可能是 z 轴深度设置 5mm 过大导致的断刀，故后续将雕刻深度调整至 3mm。

图 4- 12 断刀示意图

5）问题二：图形细节集中，雕刻不清晰

解决方法：增大图片尺寸，去掉多余细节，使主体呈现更清晰

首次雕刻作品一花的过程中，我发现由于花瓶的云纹装饰和下方的小花朵连在了一起，雕刻成品不清晰。所以在第二次重新雕刻时，增大了图片的尺寸，去掉了云纹图案，作品更加清晰，整体性好。调整前后灰度图如 4-9 所示：

图 4- 9 调整前后灰度图

实际作品对比如下图：


图 4- 10 调整前后雕刻作品

# 第五章 总结和体会

本次实习我是我们小组的团队负责人，负责了部分的编程功能添加和 QT 界面设计工作，在我和程婷婷的共同协作下，我们补充完成了登录界面、密码修改界面、日志设计、时间计算、键盘绑定和自动停止等功能，并对软件界面进行了美化设计。

通过本次三维运动控制实习，我们对三维运控轨迹规划的实现过程有 了更深入的理解，对课本中的理论知识进行了实际的运用操作。在本次三维运控的实习中，我们遇到了一些大大小小问题，但是都在最后的自己努力独立思考和小组团队协作下，都得到了一一解决，并且取得了很不错的成果。在这其中，不仅学到了许许多多关于三位运控、变频器、控制器的相关的知识，也大大了锻炼了我独立思考问题、解决问题、实践动手能力。在完成作品的过程中，我们不断总结经验，调整图像尺寸和雕刻细节问题，完成了作品一、二和三，取得了不错的效果。

在这次实习中，不仅仅用到了运动控制的专业知识，还用到了之前学习的编程知识、数据库知识和一些课外的专业知识。比如，日志设计是之前我和同学开发软件时发现的创新点，可以记录软件的功能情况，方便进行故障检查和维修等。但，我们设计的日志功能比较简略，只是一个雏形，还不是很完善，这就要求我需要认真学习相关知识，努力提升自己。我们学习的专业知识是成体系的，在未来的工作和学习中都可能会用到。

另外，吴涛老师提到的直接通过 I/O 口控制运动控制器以及通过软件编写程序进行一些保护操作对我也很有启发：在实际的工业控制或者加工过程中，加工机器可能会出现各种问题，我们不能仅仅依靠硬件设备保护，还需要增加软件保护程序的编写，提供双重保险。

最后，感谢小组各位成员，作品的完成离不开每一个人的努力；感谢吴涛老师和郑世祺老师在实习过程中的指导，在本次实习中，我收获很多，在未来的工作或者学习过程中，希望有机会能够用到这两周实习学习的知识。