基于嵌入式系统的运动控制器设计

课程实习报告

班　　级：_____

学　　号：_____

姓　　名：_____

组　　员：_____

二〇二二年一月

目录

第一章 实习概述

1.1 实习目标

运动控制系统实验是综合应用了：控制理论、单片机原理及接口技术、电机拖动、微机接口技术、自动控制系统等课程知识的重要实践环节。其目的在于：

通过试验来验证和研究运动检测控制，增强感性认识, 以促进认识的深化，培养学生科学的分析能力，使学生掌握一般运动控制的操作方法和基本技能；培养学生严肃认真和实事求是的科学作风，锻炼科学实验的能力。

运动控制是控制领域比较经典和最常见的控制类型。运动控制系统实践要求学生熟练掌握：

① 一般的运动控制方法和原理；

② 伺服电机驱动技术、伺服电机控制方法；

③ 微机接口技术；

④ 单片机原理及接口技术；

⑤ 数控轮廓插补原理；

了解：

① 计算机高级语言硬件编程；

② 一般运动控制系统常用机械结构；

③ 一般交流伺服电机控制方法等；

1.2 实习内容

利用实验室提供的运动控制实验仪与 51 系统板，设计并制作一个二维运动控制系统，在完成点动控制、直线插补、圆弧插补三个功能的基础上，探索并扩展其他特色功能（如任意函数曲线绘制），最终完成一个功能相对完整的二维运动控制系统。

1.3 实习设备

运动控制实验箱
EP3L 伺服驱动器
MK 交流伺服电机
限位开关/接近开关
实习用 51 系统板
个人电脑
万用表、螺丝刀、六角扳手、电工胶带、尼龙扎条、圆珠笔等

1.4 本小组系统特色

①上电后 LCD 显示手写体的“中国地质大学”作为开机界面
②8 方向手动移位，按着就走，不按不走，速度可调，lcd 显示当前速度和一张 32*32 的箭头图片，后者可指示当前移动方向
③绘制任意半径 R 和圆心角 θ 的圆弧，且绘制中途可退回主菜单
④绘制任意半长轴和半短轴的椭圆
⑤绘制任意振幅 A、任意角频率 ω 、任意初相 φ 的 sin 函数
⑥上位机发送指令，控制电机绘制任意长度的直线，任意半径的圆，控制绘制速度。开始绘制和结束绘制时，单片机向电脑发送开始绘制和绘制完成的信息。

1.5 本人分工（大致）

lcd 显示和键盘输入、
改进手动控制功能、
圆的绘制、
sin 函数绘制、
椭圆绘制与改进、
整体程序逻辑设计、
各功能整合、
联合调试

第二章 硬件设计

2.1 硬件的总体设计

本系统的硬件组成主要分为实验箱、51 系统板两大部分，51 系统版是控制核心，实验箱是执行部分，同时也承担部分控制功能。

51 系统板上有 51 单片机、8255、lcd、键盘、数码管、光耦等一系列硬件设备，在系统中承担：采集用户输入、显示、向驱动器发出控制信号的作用。

实验箱上有一个开关电源，可将 220V50Hz 的市电，转换成 24V 的直流电，供给驱动器和限位开关；每个实验箱有两个方向正交的丝杆和与之配套的伺服电机、以及两个用于控制伺服电机的驱动器。固定在移动台上的金属座可连接金属杆和笔，电机的转动后由丝杆传动，移动台就带着笔一起移动，从而实现图形的绘制。

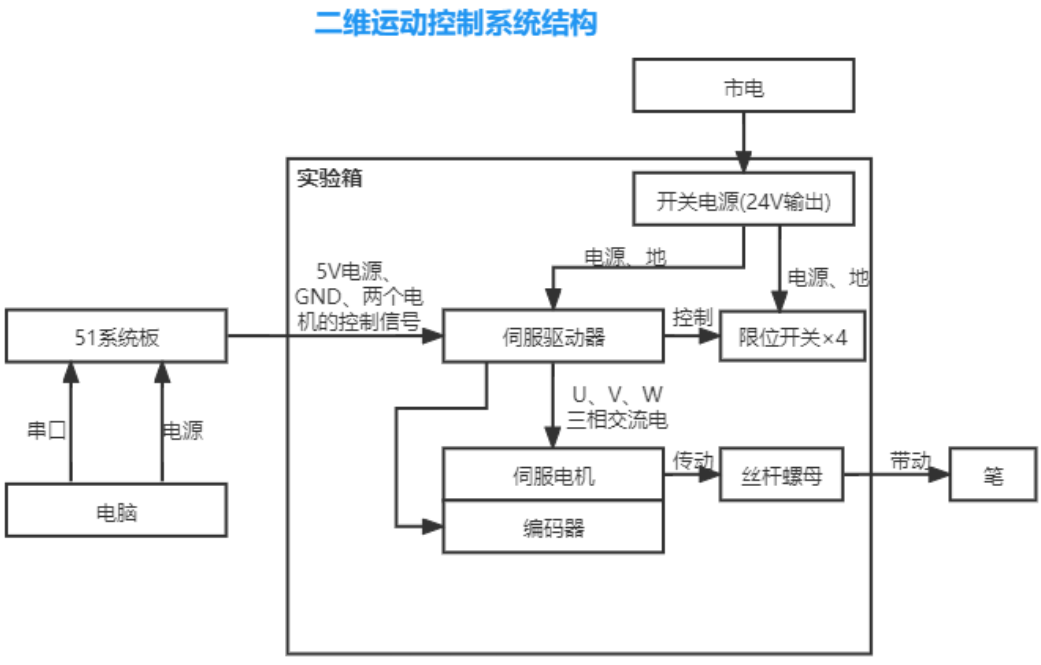


图 2- 1 二维运动控制系统整体框架图

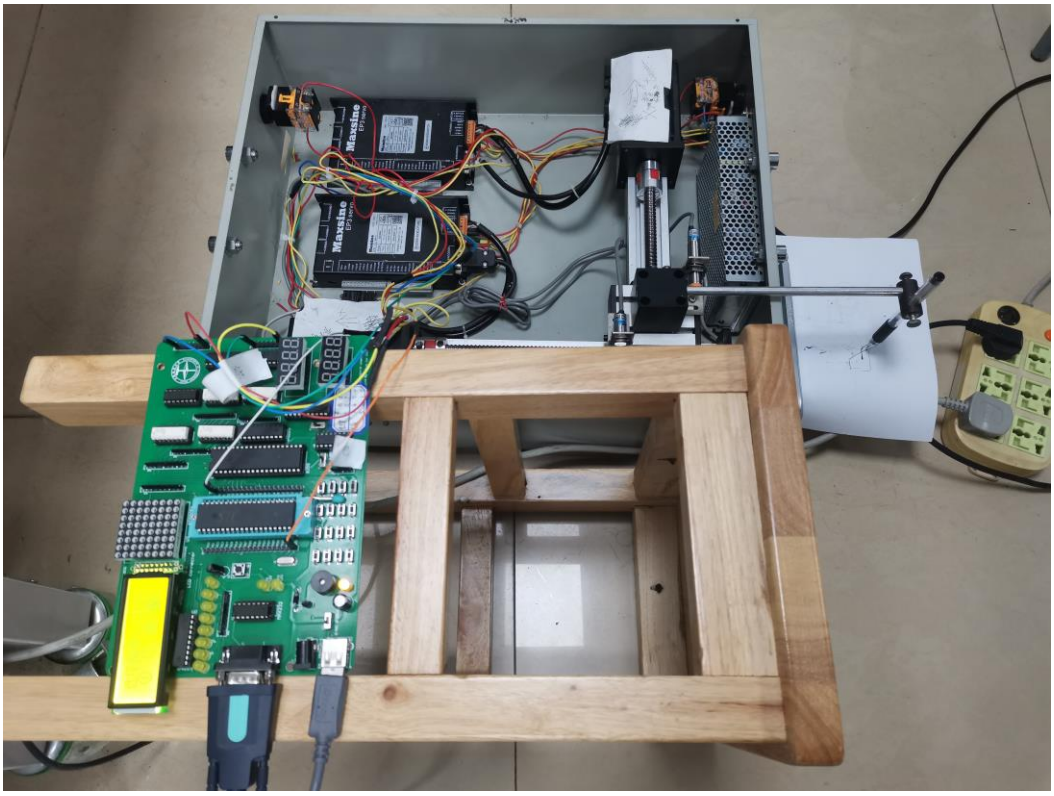


图 2- 2 二维运动控制系统实物图（整体）

2.2 运动控制实验箱

运动控制实验箱是一个平面两轴驱动的运动装置，机械部分采用丝杆螺母的传动方式，如图 2-2 所示。它可以由单片机控制，控制系统用一套伺服电机驱动和检测电路板。驱动和检测电路板上有单片机的控制接口，使用十分方便。该运动控制实验仪配备了一只画笔，通过编制相应的控制程序，可以在图板上绘出不同的图形，使运动过程更加明了，便于观察。

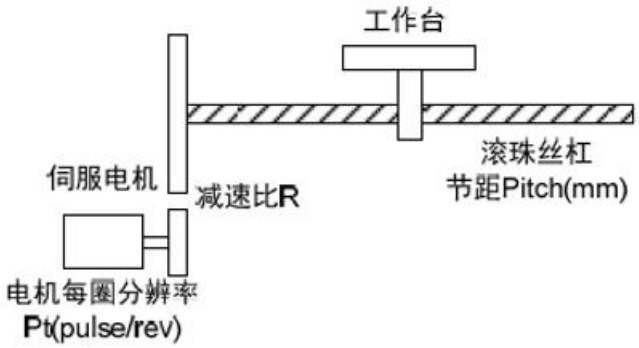


图 2- 3

2.3 MK 交流伺服电机

伺服系统（servomechanism）是使物体的位置、方位、状态等输出被控量能够跟随输入目标（或给定值）的任意变化的自动控制系统。

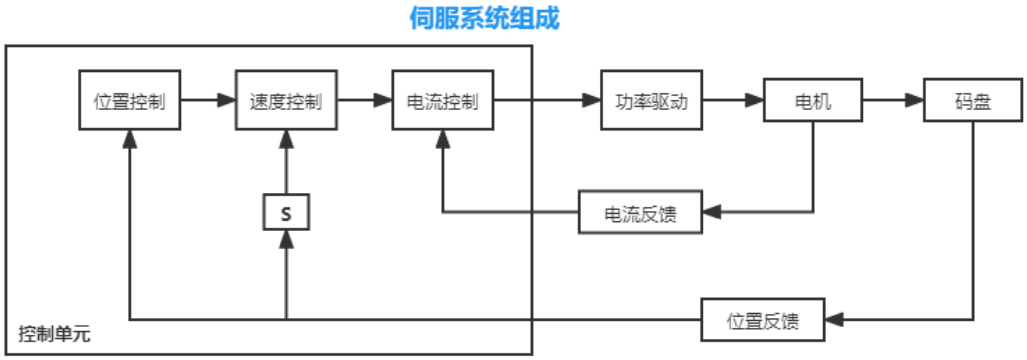


图 2- 4 伺服系统的组成

MK 系列交流永磁伺服电机：低电压，中惯量，高转速，旋转伺服电机，转矩范围为 0.64N.m~1.27N.m。

本实验箱所用 MK 伺服电机内部的转子是永磁铁，驱动器控制的 U/V/W 三相电形成电磁场，转子在此磁场的作用下转动，同时电机自带的编码器反馈信号给驱动器，驱动器根据反馈值与目标值进行比较，调整转子转动的角度，形成闭环，因而控制精度可以做得很高。伺服电机的精度决定于编码器的精度。

伺服电机有三种控制方式：转矩控制方式、速度控制方式、位移控制方式。本系统采用的是位移控制方式，通过外部输入的脉冲的频率来确定转动速度的大小，通过脉冲的个数来确定转动的角度，控制输入脉冲数量、频率及脉冲的方向，就可以得到需要的运行特性。位置模式可以对速度和位置都有很严格的控制，所以一般应用于定位装置，如数控机床、印刷机械等等。

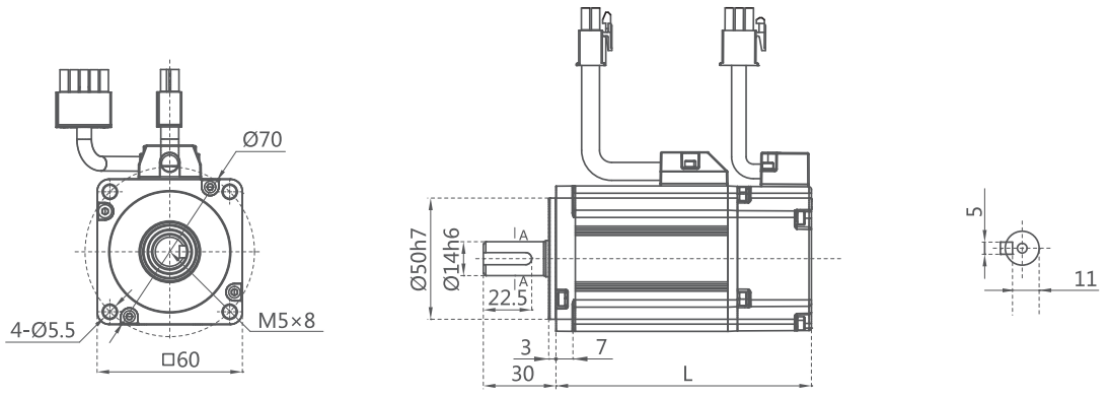


图 2- 5 MK 系列伺服电机外形尺寸

2.4 EP3L 伺服驱动器

EP3L 低压小功率伺服驱动器：输入电压 24V~60V，功率 0.2kW~0.4kW。

电源及电机端子用来给驱动器和伺服电机提供电源；编码器接口与伺服电机的编码器引出线对应端接在一起；IO 端口主要接收上位控制器（这里是单片机）的输入信号和行程开关的输入信号；显示板端子用来接键盘（手操器），通过键盘来修改伺服驱动器的参数和调机试运行；指示灯用来显示驱动器的状态是正常还是有报警发生。其他没有介绍的端子或接口本实验仪没有用到。

表 2- 1 驱动器端子定义

名称	端子符号	说明
电源及电机端子	VDD、GND	驱动器主电源输入
	VBUS+、VBUS-	功率端子
	BRAKE	刹车电阻
	PHASE(U、V、W)	输出到电机 U、V、W 相电源
编码器接口	ENCODER	编码器反馈信号
控制器信号线端子	I/O	用于连接上位机(控制器、PLC 等)

PHASE：相位；　　ENCODER：编码器

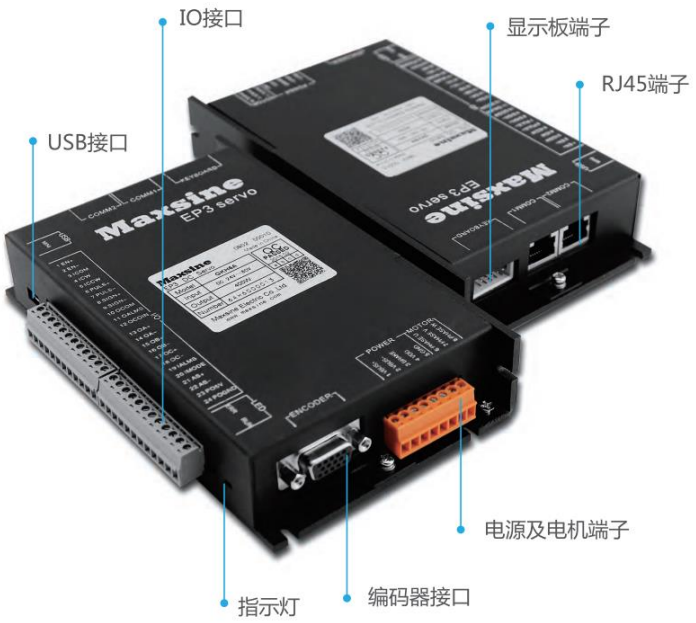


图 2- 6 驱动器及其端子定义

2.5 实习用 51 系统板

这是一款已经由老师设计好的 51 单片机系统板，搭载的是 STC 公司的 89C52RC 单片机，12 时钟周期/机器周期，工作频率 12MHz，它拥有 8K 字节的程序空间和 512K 字节的 RAM，具有 EEPROM 和看门狗以及 3 个 16 位定时/计数器、4 路外部中断、通用异步串行口。除此之外，系统板上还有 8255、12232 液晶、8×8 点阵、流水灯、八位数码管、4×4 键盘、四个光耦、排针、串口等资源，作为本控制系统的核心控制部分，向伺服驱动器输出控制信号，从而控制电机运动，最终达到绘制图形的目的。

使用时机箱 24V 开关电源的 0V、伺服驱动器的地、51 系统板的地要共地。两个驱动器的 6 脚（PULS+）和 8 脚（SIGN+）均接到单片机的 VCC（由单片机给驱动器的脉冲和方向输入信号提供电源）。驱动器 7 脚（PULS-）9 脚（SIGN-）接到 51 系统板的 P6(8 脚排针)或 P5（8 脚排针）上。这两个排针经过了光耦隔离，可避免驱动器一侧电流过大而烧坏板子。

实验中我们组的设计是：

P6 的低 4 位输出 X 轴的脉冲信号、P6 的高 4 位输出 X 轴方向信号；

P5 的低 4 位输出 Y 轴的脉冲信号、P5 的高 4 位输出 Y 轴方向信号。

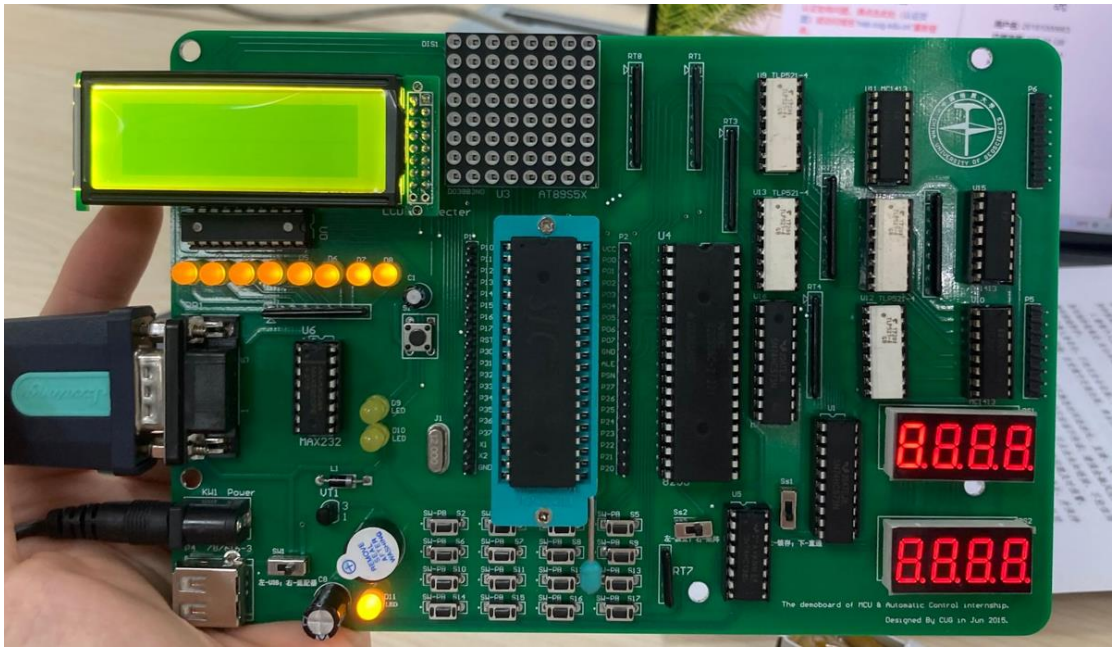


图 2- 7 实习用 51 系统板

2.6 个人电脑

应事先装配好实验所需的开发平台和工具软件，如 keil4/keil5、STC-ISP、串口助手等。用于编写编译代码、烧录程序到实验板上的单片机中。

2.7 遇到的问题及解决

（1）最开始编了个简单程序输出脉冲，但电机不转

问题分析：虽然一般情况下是脉冲频率越高，电机转速越快，但经过不断调整脉冲频率，我们发现电机有一个最高可响应的启动频率值，当以高于该频率的脉冲启动时，电机是启动不了的。而且不同的电机，这个频率还会不同，比如我们组的 Y 轴电机就比 X 轴电机支持的最大频率稍小一点。

问题解决：保证输出脉冲频率在最大可响应值以内，电机即可正常启动与运行。

（2）丝杆有些许弯曲

问题解决：队友在后排中箱子里找到了一根好些的丝杆，使用螺丝刀、六角扳手等工具进行了拆卸和更换。

2.8 小结

硬件部分的连线和组装、维修主要是由队友负责的，我只参与了很少的一部分。我们拿到的箱子，没有存在编码器故障、电机故障等大问题，只是进行了连线和整理，并更换了丝杆，紧固了螺丝，过程较为顺利。

这是第一次接触伺服控制系统，感觉很新奇也很有意思。在伺服系统中，驱动器的使用十分重要，我们对此也进行了一些学习。所有的硬件，我们基本上都是用的实验室提供的，不过也正是因为所选用的 52 单片机性能太过有限，所以给后续的开发带来了不少的麻烦和限制，如果可以再来一次，我应该会选择 STM 32 或者是 S3C 2440，他们不仅有更大的内存、更丰富的硬件资源，而且频率更高，可以控制电机以更高速度移动。

第三章 软件设计

3.1 软件的总体设计

进入主函数，先设置 8255 的工作方式，PA 口和 PB 口都是方式 0、输出，PC 口的低四位输入，高四位输出。

本程序是一个两层菜单结构，第一层是在主函数的 while 循环中检测按键，当某个功能模块对应的按键被按下时，进入各功能模块的函数，即进入两级结构的下面一层，各模块函数的内部都有一个 while(1)死循环，在这个死循环内循环检测按键、执行手动移位、画直线、画圆、画 sin 函数等功能，当检测到按下退出键时，则用 break 跳出该 while(1)，退出子功能，然后刷新主菜单，即可回到上一级，然后重新选择要进入的模式。

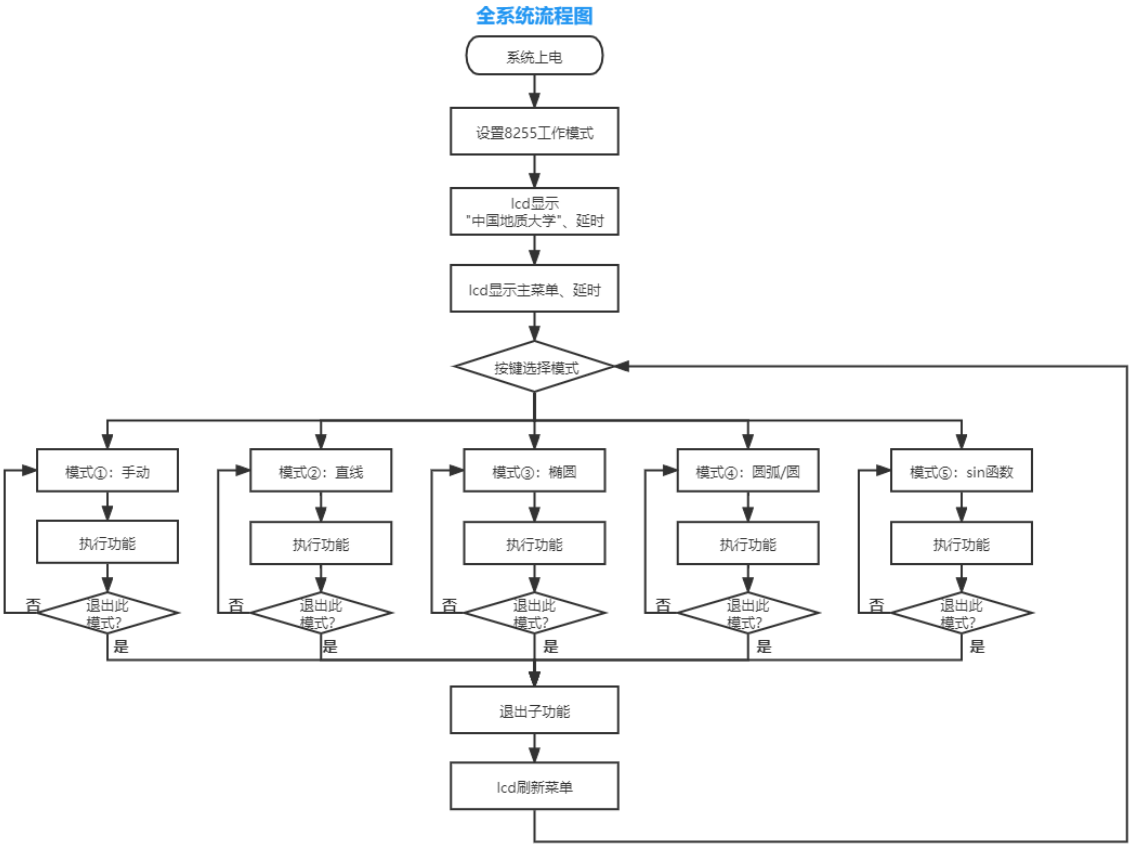


图 3-1 全系统流程图

主函数代码如下：

```
1.  void main()
2.  {
3.   CAddr= mode;
4.  show_CUGlogo();
5.  delayms(1000);
6.  show_menu();
7.  delayms(1000);
8.  while(1)
9.  {
10.  menuCiShu--;
11.  if(menuCiShu<0)
12.  {
13.    menuCiShu=80000;
14.    show_menu();
15.    delayms(10);
16.  }
17.  if(TestKey()==0)//没键按下
18.  {
19.    keynum=100;
20.  }
21.  else//有键按下
22.  {
23.    keynum=keyscan() & 0x0f;
24.    switch (keynum)  //keynum 就等于按键的数值
25.      {
26.          case 0x0f:  {Moshi_1();break;}  //模式一：手动移位
27.          case 0x0e: {Moshi_2();break;}  //模式二：画直线
28.          case 0x0d: {Moshi_3();break;}  //模式三：画椭圆
29.          case 0x0c: {Moshi_4();break;}  //模式四：画任意圆弧
30.          case 0x0b: {Moshi_5();break;}  //模式五：画 sin
31.      }
32.   }
33. }
34. }
```

3.2 人机交互部分——LCD 显示

因为本系统用到的字模较多，为使 lcd.c 程序不过于冗长，我在这里将字模库都建成.c 文件独立出去，然后在 lcd.c 中#include 它们。共有 8×16，16×16，32×32 和 60×24 四种不同大小的字模，分别用了 4 个不同的.c 文件来装。

```
1.  #include <reg52.h>
2.  #include "8x16zimo_lib.c"     //8*16
3.  #include "16x16zimo_lib.c"    //16*16
4.  #include "directionArrows.c" //32*32
5.  #include "lcd_CUG_logo.c"    //60*24
```

事先设计好每个功能模块的界面，并确定字模大小，然后用取模软件“文字图形取模.exe”取出对应的字模，存到字模库里。字模库的本质是一个数组，例如 8×16 的字模库是 DIGTAB[][16]，每个字模 16 字节；16×16 的字模库是 CCTAB[][32]，每个字模 32 字节。

下面讲一下关于软件“文字图形取模.exe”的使用：首先，在给本实习所用的恒科液晶取模时，“参数设置”—“其它选项”要勾选“纵向取模”和“字节倒序”。（若是给 led 点阵取模，则应勾选“横向取模”和“字节倒序”，同时，若是 16x16 点阵，则还需要将图形旋转 180°，若是 8x8 点阵，则不用旋转）。经个人在实践中总结，获取字模的方式总的来说有三种：

①文字输入法：在文字输入区输入要取模的汉字、字母、数字、符号等内容，然后按 Ctrl + Enter 键即可生成模拟液晶显示的图形，再选择“取模方式”——“C51 格式”，即可得到字模。

②位图导入法：把目标图像先用 Windows 自带的画图软件转换成 bmp 格式，再在“文字图形取模.exe”中打开，即可得到模拟液晶图像，然后点“C51 格式”取得字模（或许更准确一点应该叫“图模”）即可。

③自行抠图法：“新建图像”选择高宽尺寸，然后“模拟动画”—“放大格点”至合适大小，用鼠标点击屏幕上白色区域内的格点，它便会在黑与白之间切换，从而实现自行抠图，此种方法最为灵活，能取得各种自己想要的图片，不过工作量稍大，只在图片较小时比较适用。在实践中可结合方法②和③，在导入图片后，再通过鼠标点击修改图像。

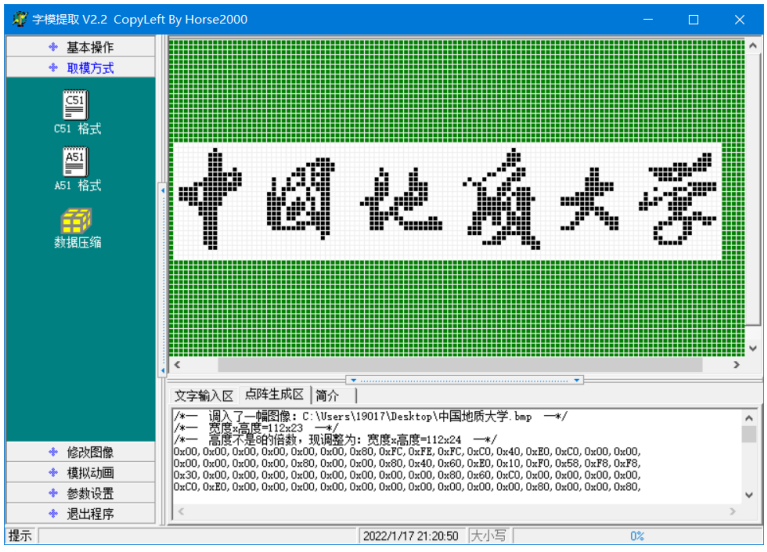


图 3- 2 “文字图形取模.exe”打开 bmp 文件

每一个页面内容的显示都是通过调用一个函数实现的，比如Display_1()~Display_4()分别用来显示手动、画直线、画圆弧等子功能模块的界面，show_menu()用来显示主菜单界面，而 show_CUGlogo()则用来在系统上电后显示手写体的中国地质大学作为欢迎界面。虽然这些函数的显示内容不同，但其逻辑结构都是一致的，可用如下流程图表示：

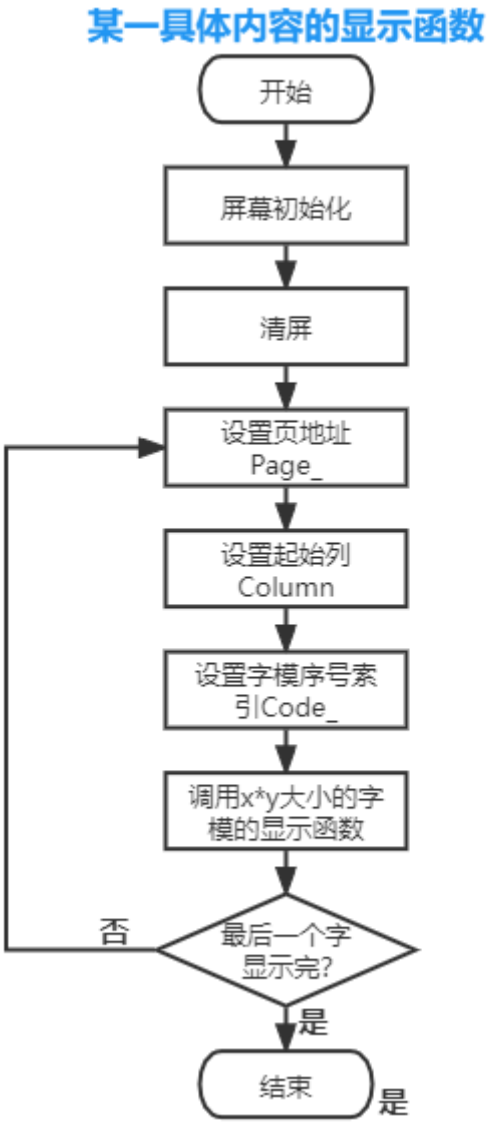


图 3- 3 某一具体内容显示函数

从上图可看出，诸如 show_menu()这些函数都调用了另一类函数——x*y 大小的字模显示函数，这个函数实现着更基础的功能，正是它把一个个特定大小的字模显示在屏幕上，通过对该函数的修改，可以实现各种尺寸的字模显示，所以掌握了这个函数，才算是掌握了恒科 12232 液晶的使用。我这里所用字模有四种大小，所以该类函数共有四个： WriteNUM8x16()、 WriteCHN16x16()、WritePicture32x32()、WritePicture60x24()。它们的逻辑结构可用如下流程图表示：

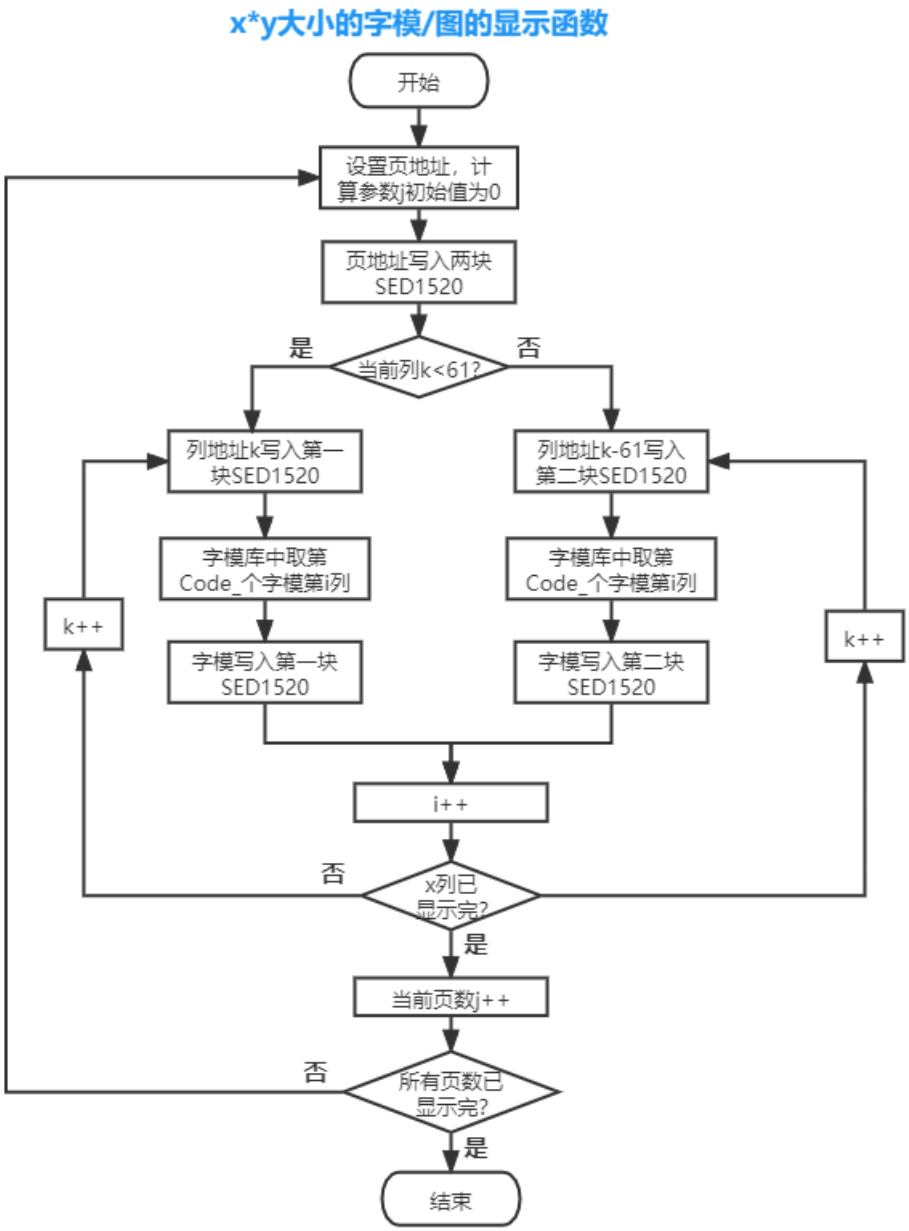


图 3- 4 x*y 大小的字模显示函数

下面以显示中国地质大学的函数 WritePicture60x24()为例，分析其代码。

当我们要显示一个宽 x 列高 y 行的字模的时候，修改函数中的两个参数就行，一是第 7 行的 while(j＜3)，表示字模所占据的页数，而该液晶中一 Page 是 8 行，所以当字模的高是 1～8、9～16、17～24、25～32 时，括号内的常数分别填 1、2、3、4。二是第 13 行的 while(k＜Column+60)，括号中的常数代表一个字模的宽，虽然这款 lcd 总共有 122 列，但受硬件限制（具体原因在后文 3.10 节“遇到的问题及解决”中论述），一个字模最多只允许有 80 列。更多细节请参阅代码及注释。

```
// *******60x24 显示子程序*******
void WritePicture60x24()   //*********调用一次显示一个字模*****
{
  unsigned char i,j,k;
  i = 0,   j = 0;
 while(j<3)//用到 3 页就改为 3************************************
 {
    Command = ((Page_ + j) & 0x03) | 0xb8;  //设置页地址
    WriteCommandE1();
    WriteCommandE2();
    k = Column;  // 列地址值
    while(k < Column + 60)//*****60 列*******硬件最多支持 80 列*********
  {
     if (k < PD1) // 为左半屏显示区域(E1)
   {
       Command = k;
       WriteCommandE1();       // 设置列地址值
       LCDData = CUGlogoTAB[Code_][i]; // 取汉字字模数据
       WriteDataE1();          // 写字模数据
     }
   else   // 为右半屏显示区域(E2)
   {
       Command = k-PD1;
       WriteCommandE2();       // 设置列地址值
       LCDData = CUGlogoTAB[Code_][i]; // 取汉字字模数据
       WriteDataE2();          // 写字模数据
     }
     i++;
     if( ++k >= PD1 * 2) break;  // 列地址是否超出显示范围
   }
   j++;
  }
}
```

本次实习一共设计了七个界面，分别是:

①手写体“中国地质大学”

②主菜单

③手动模式

④直线模式

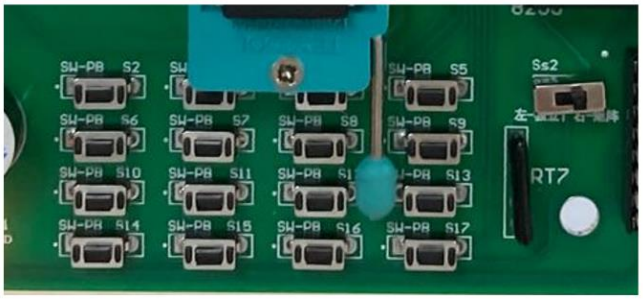
⑤画圆弧

⑥画椭圆

⑦画 sin 函数

图 3- 5 本系统的所有 lcd 界面

3.3 人机交互部分——键盘输入



f	e	d	c
b	a	9	8
7	6	5	4
3	2	1	0

图 3- 6 键盘实物(左)与对应的键值(右)

开关 Ss2 向左拨时，最左边一列的 4 个按键就是独立按键，而当 Ss2 向右拨时，这 4×4 共 16 个键就组成矩阵键盘。

键盘扫描部分主要用到了两种函数：①TestKey()，用来检测有无键被按下，只要有就返回非零；②keyscan()/keyscan2()，键盘扫描函数，用于返回所按键的键值。

按键检测函数 TestKey()的原理：8255 的 PC 口高四位做列扫描，低四位做行读入。PC 口是自带锁存功能的，PC 的低四位接到排阻 RT7 上，RT7 起上拉电阻的作用，无键按下时，PC0~PC3（各行的读入）都被拉为高电平，因为 P6~P7 一直输出低电平，所以只要一有键按下，其所在行就会被拉为低电平。

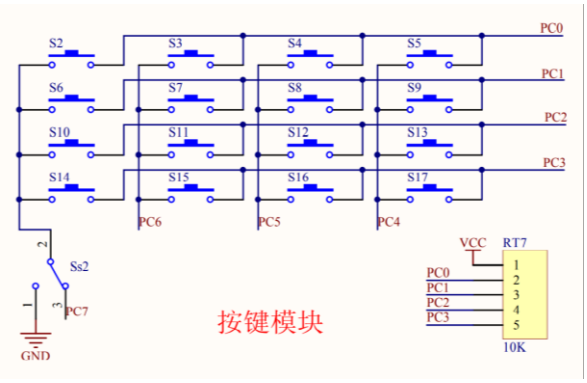


图 3- 7 按键模块电路

TestKey()函数代码：

```
1.  unsigned char TestKey() //有无键按下检测，有就返回非零
2.  {
3.    PortC = 0x0f;          // 输出 PC4~PC6 置为 0，四列一起扫描
4.    return (~INC & 0x0f); //读回四行状态（PC0~PC3），被按下的被拉成 0，取反为 1
5.  }
```

键盘扫描函数 keyscan()/keyscan2()的原理： 我们通常都是先使用按键检测函数 TestKey()确定有键按下之后，才调用键盘扫描函数。与按键检测函数不同，这时我们不是同时给 4 列都输出低电平，而是逐列输出低电平，即逐列扫描，当扫描到某一列时，若读取到某一行为低，则结束扫描，然后用当前的列和为低的行计算出键值在键值表中的索引值 i，在 keyscan()中会等待按键松开后再返回键值，而在 keyscan2()中则不等按键松开，直接返回键值。

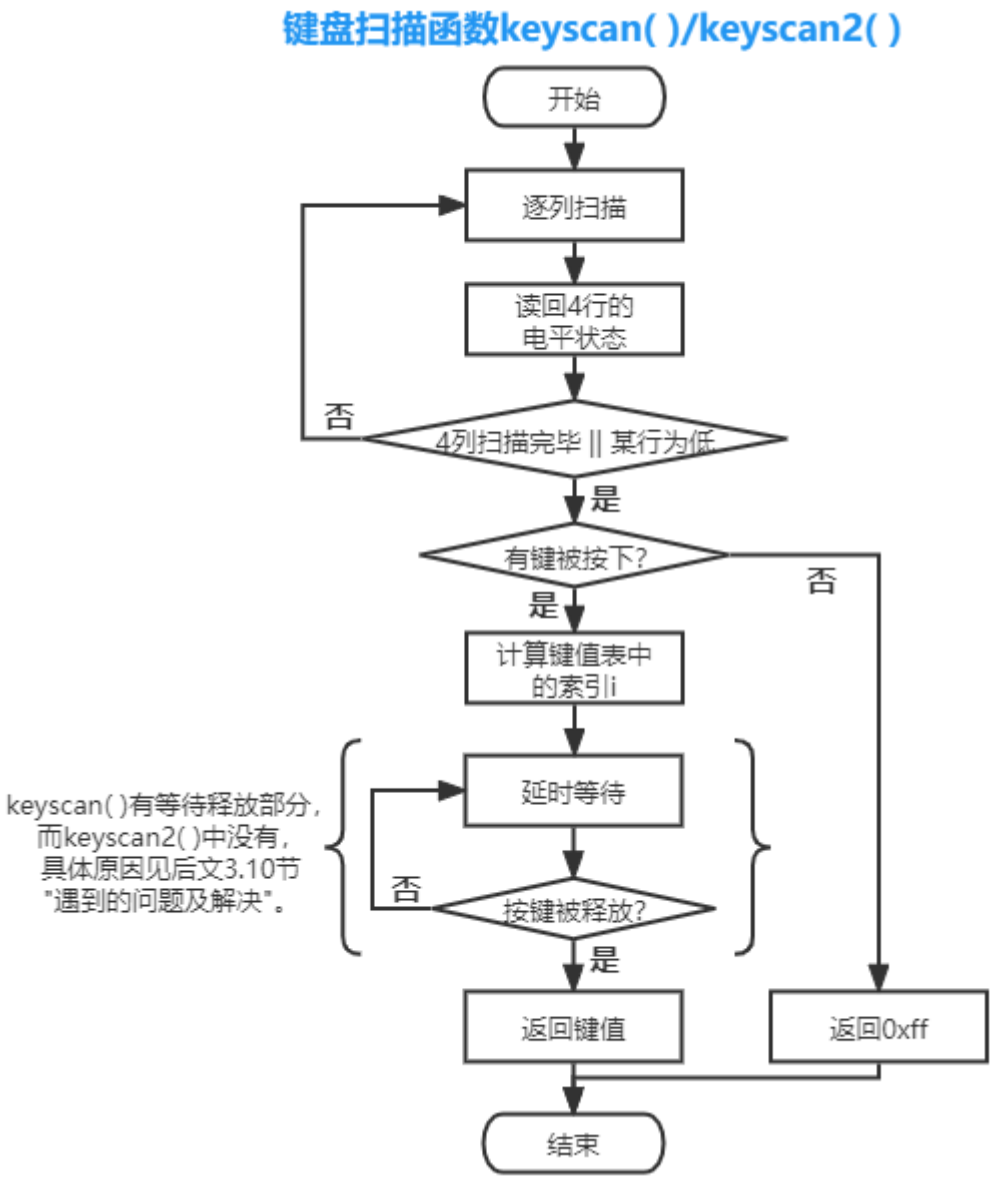


图 3- 8 键盘扫描函数 keyscan()/keyscan2()

```
1.  unsigned char keyscan() //键盘扫描函数，返回键值
2.  {
3.    unsigned char Pos , i , k;
4.    i = 4;
5.    Pos = 0x80;    // 1000 0000，扫描用
6.    do {
7.      PortC = ~ Pos;
8.      Pos >>= 1;
9.      k = ~IN & 0x0f;
10.   } while ((--i != 0) && (k == 0));
11.   if (k != 0) {//如果有键按下
12.     i *= 4;
13.     if (k & 2)
```

```
14.      i += 1;
15.    else if (k & 4)
16.      i += 2;
17.    else if (k & 8)
18.      i += 3;
19.   do {Delaykey(10);}
20.   while (TestKey()); // deng 键松开
21.    return(KeyTable[i]); //有键按下返回键值
22.  }
23. else return(0xff);//无键按下，则返回 0xff
24. }
```

3.4 运动控制部分——手动模式

3.4.1 手动模式总体设计

手动模式是由函数 Moshi_1()实现的，而 Moshi_1()的实现是基于另两个函数：①手动移位函数 shoudong()。②手动模式的 lcd 界面显示函数 Display_1()。

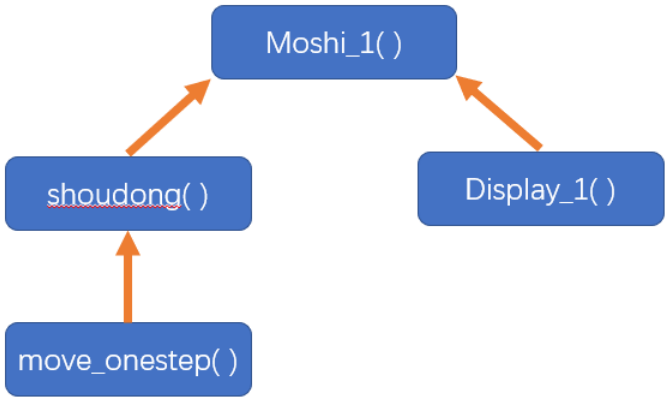


图 3- 9 手动模式的各函数依托关系

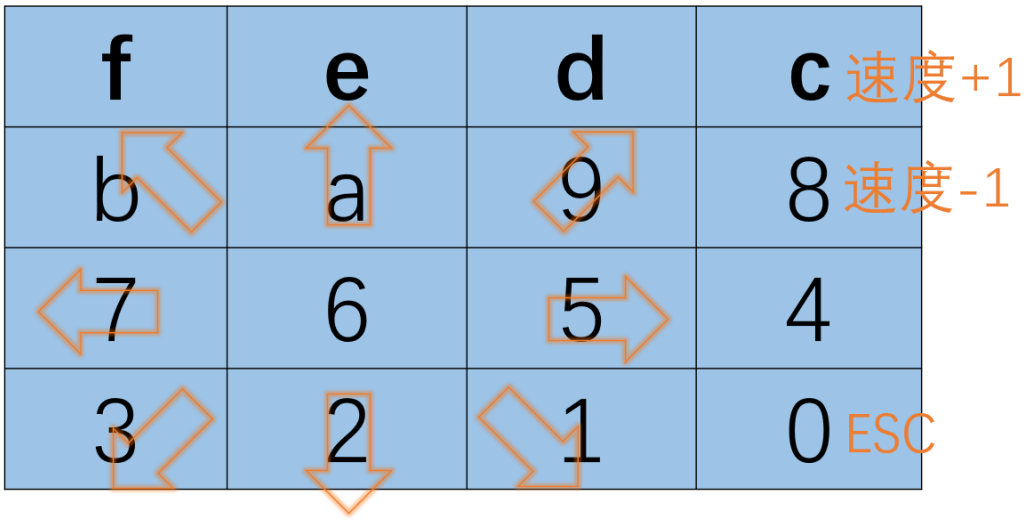


图 3- 10 手动模式的按键定义

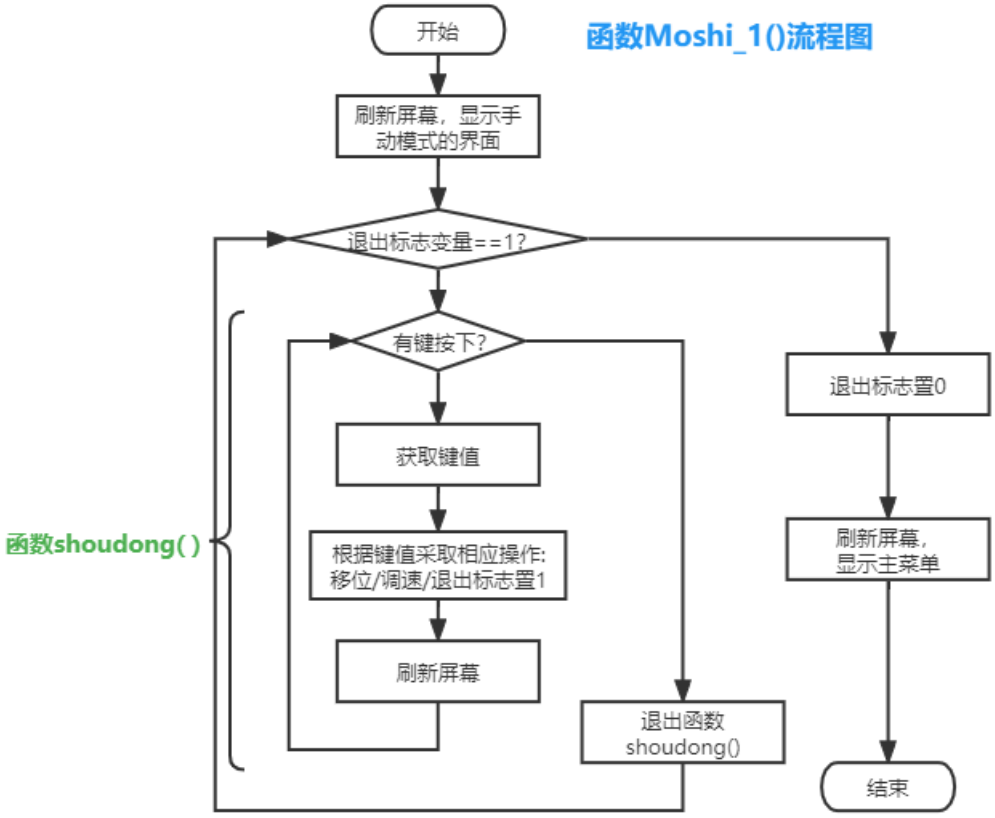


图 3- 11 函数 Moshi_1()流程图

```
1.  void Moshi_1 (void)//moshi1，手动
2.  {
3.    Display_1(SDspeed,index);
4.    delayms(100);
5.    while(noTuiChu)
6.    {
7.     shoudong();
8.    }
9.    show_menu();delayms(1000);noTuiChu=1;
10. }
```

3.4.2 手动模式运动设计

手动移位的运动功能由函数 shoudong()完成，它的逻辑流程见图 3-11 。

而 shoudong()函数的实现又是基于另一个至关重要的函数——单步移动函数 move_onestep()。move_onestep()是所有移动功能的基础，不论是手动模式还是画直线、画弧、画椭圆、画 sin 都是以它为基础的，都要调用 move_onestep()。给它设置了三个参数：坐标轴（‘x’ / ‘y’）、方向（‘+’ / ‘-’）、速度（一个无符号的整型数），通过设置着三个参数，就能实现以一定速度对 x 或 y 轴的正向或负向移位。

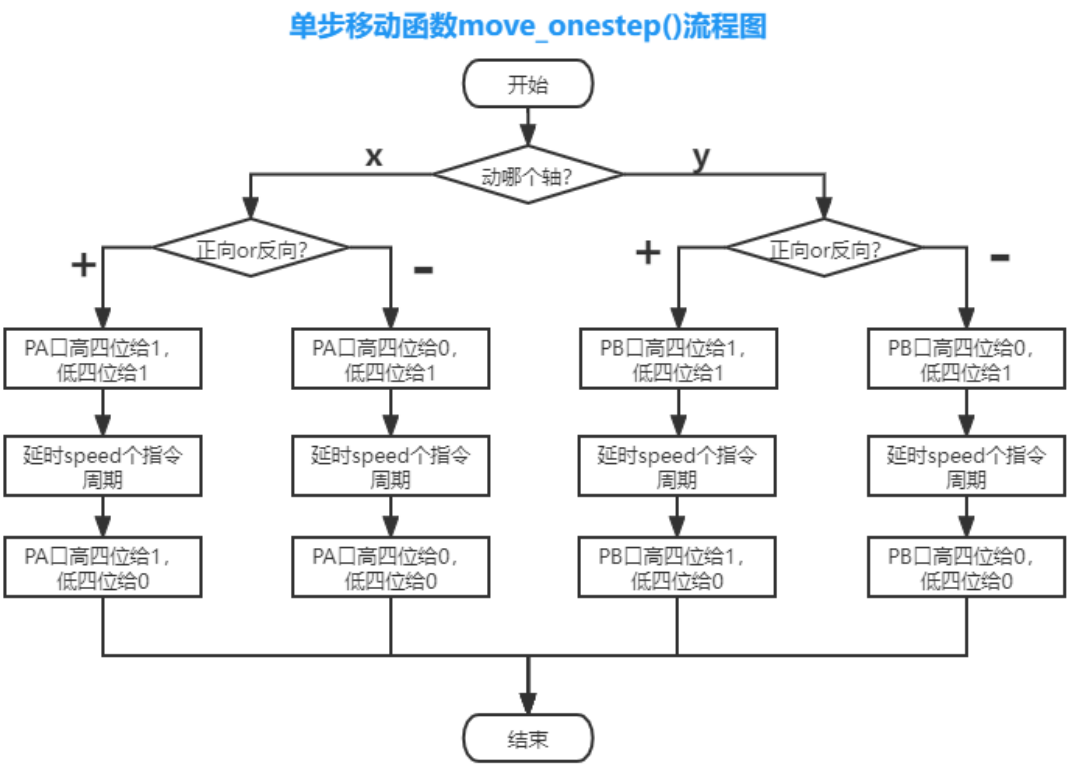


图 3- 12 函数 move_onestep()流程图

3.4.3 手动模式人机交互设计

手动函数 shoudong()是用来执行手动模式中的移位功能的，为了实现长按长动，不按不动的功能，我设置了一个 while(TestKey()){…}语句，只要按键按下没有松开，就会一直在这个循环中不断的用函数 keyscan2()获取键值，然后根据键值判断方向，进行移动。

在这里我总共设置了上、下、左、右、左上、左下、右上、右下八个方向，并且设置了速度增加键和速度减小键，以及用于返回主菜单的退出键。

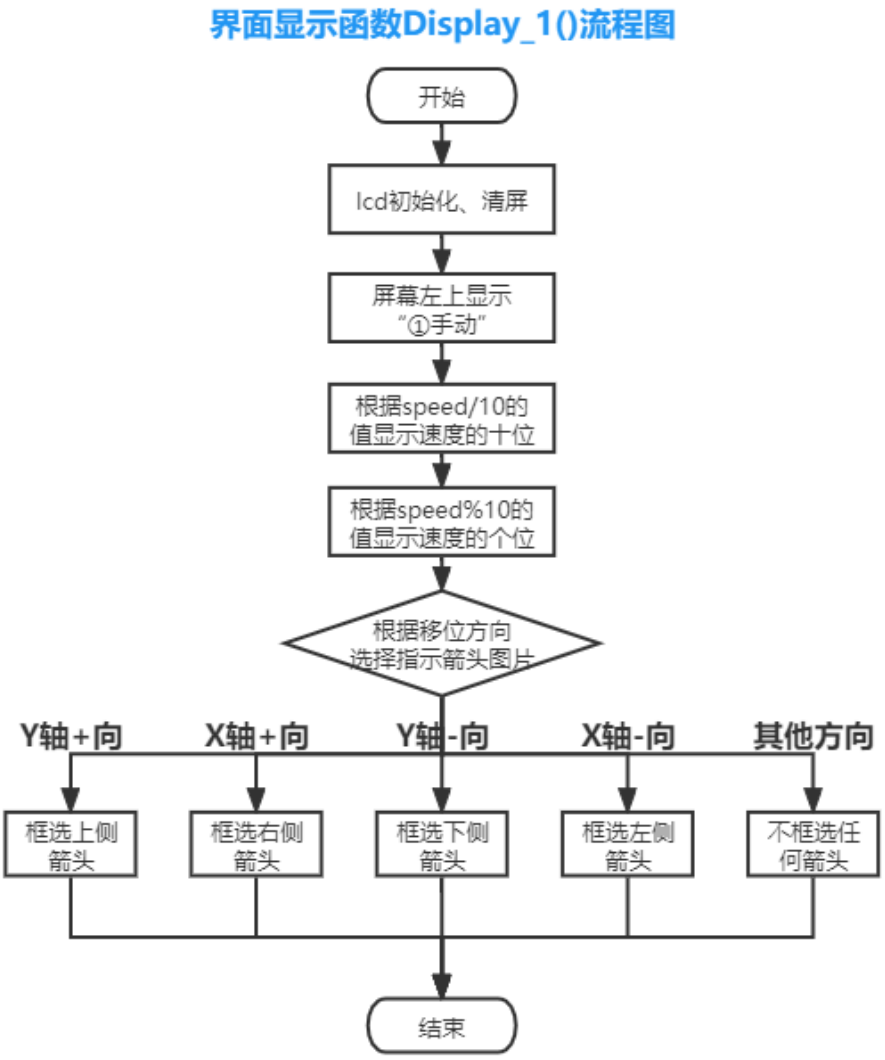


图 3- 13 手动模式界面显示函数 Display_1()流程图

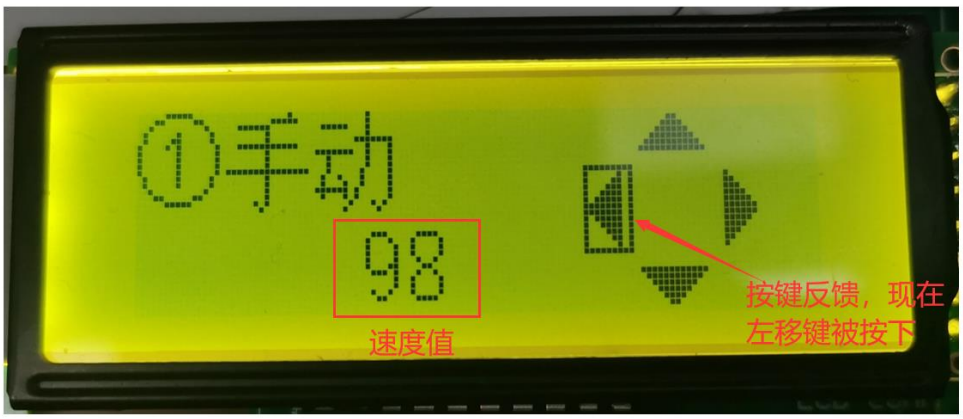


图 3- 14 手动模式的 lcd 界面

3.5 运动控制部分——直线绘制

3.5.1 直线插补原理

平面直角坐标系内任一直线 AB，起点 A（X0，Y0），终点 B(Xe，Ye)，直线方程为：

$$\frac{X-X0}{Y-Y0}=\frac{Xe-X0}{Ye-Y0}$$

取判别函数 $F=(Y-Y_0)(X_e-X_0)-(X-X_0)(Y_e-Y_0)$

用逐点比较法加工时，每一次只在一个坐标方向给出一个脉冲，使运动件在该坐标方向上进给一步，因此刀具的运动轨迹是折线，而不是斜线 AB。折线拐点 M 与斜线 AB 之间的位置关系有如下三种情况：

1)M 点在 AB 线的上方．判别函数 F>0;

2)M 点在 AB 线上，F=0

3)M 点在 AB 线的下方，F<0

为控制方便，将 F>0 和 F=0 两种情况作为 F>=0 一种方式判别。当判别函数 F>=0 时，刀具一定处在 AB 线的外侧，或直线 AB 上，这里我们以 Y 坐标绝对值大为“外”，Y 坐标绝对值小为“内”。

例如图 3-15 中的 M1(X1,Y1)点，这时刀具只有沿+X 方向进给才更接近 AB 线，因此，根据这个判别结果，计算机在 x 铀方向输出一个脉冲，使刀具在+x 方向前进一个脉冲当量的距离，到达 M2 点。若 F2<0 则 M2 点判定处在 AB 线 的下方,应向 y 方向送出一个脉冲，使刀具向+y 方向移动一步，到达从点 M3。

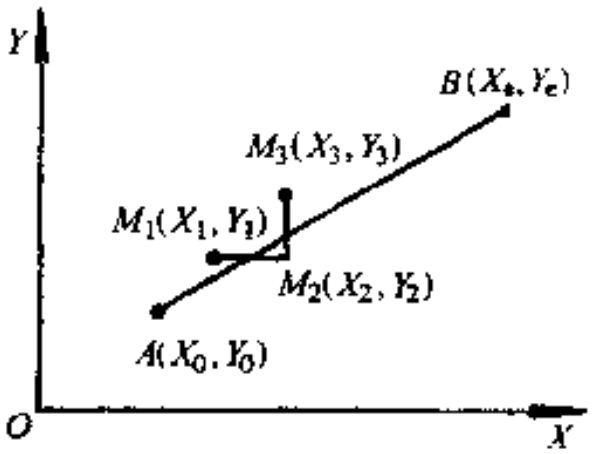


图 3- 15 第一象限直线插补示意图

表 3- 1 象限判别和电机方向

方向	第一象限	第二象限	第三象限	第四象限
Xe-X0	>0	<0	<0	>0
Ye-Y0	>0	>0	<0	<0
X 向电机	正	反	反	正
Y 向电机	正	正	反	反

根据对线段加工方向的不同，可将直线所在分为四个象限，如图 3-16。

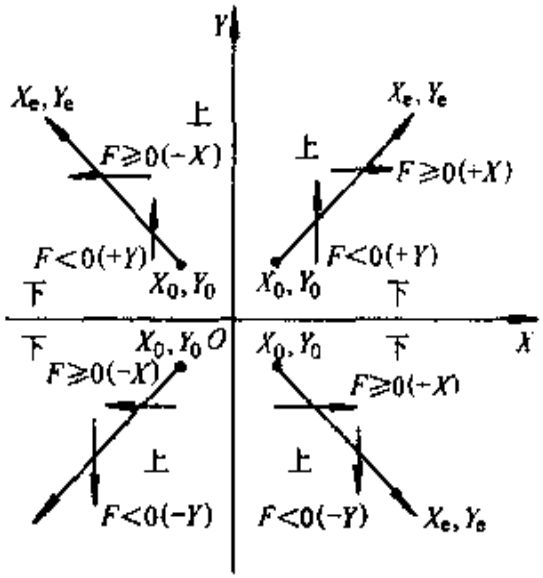


图 3- 16 四象限的直线插补示意图

直线插补的终点判断方法：（两种）

①每走一步都要计算|(Xi-X0)|，|(Yi—Y0)|的数值，并判断|(Xi-X0)|>= |(Xe-X0)|且|(Yi—Y0)|>= |(Ye—Y0)|是否成立，若成立则插补结束，否则继续。

②把被加工线段的 Xe-X0，Ye—Y0 的长度单位换算成脉冲数值(若长度单位为 mm，则把上述的坐标增量值除以脉冲当量),然后求出各坐标方向所需的脉冲数总和 n：即 n＝|(Xe—X0)+(Ye—Y0)|，计算机无论向哪个方向输出一个脉冲都作 n—1 计算，直到 n=0 为止。

本次实习中，我们组采用的是方法②。

3.5.2 直线插补模式总体设计

核心函数 Moshi_2()逻辑流程如下图：

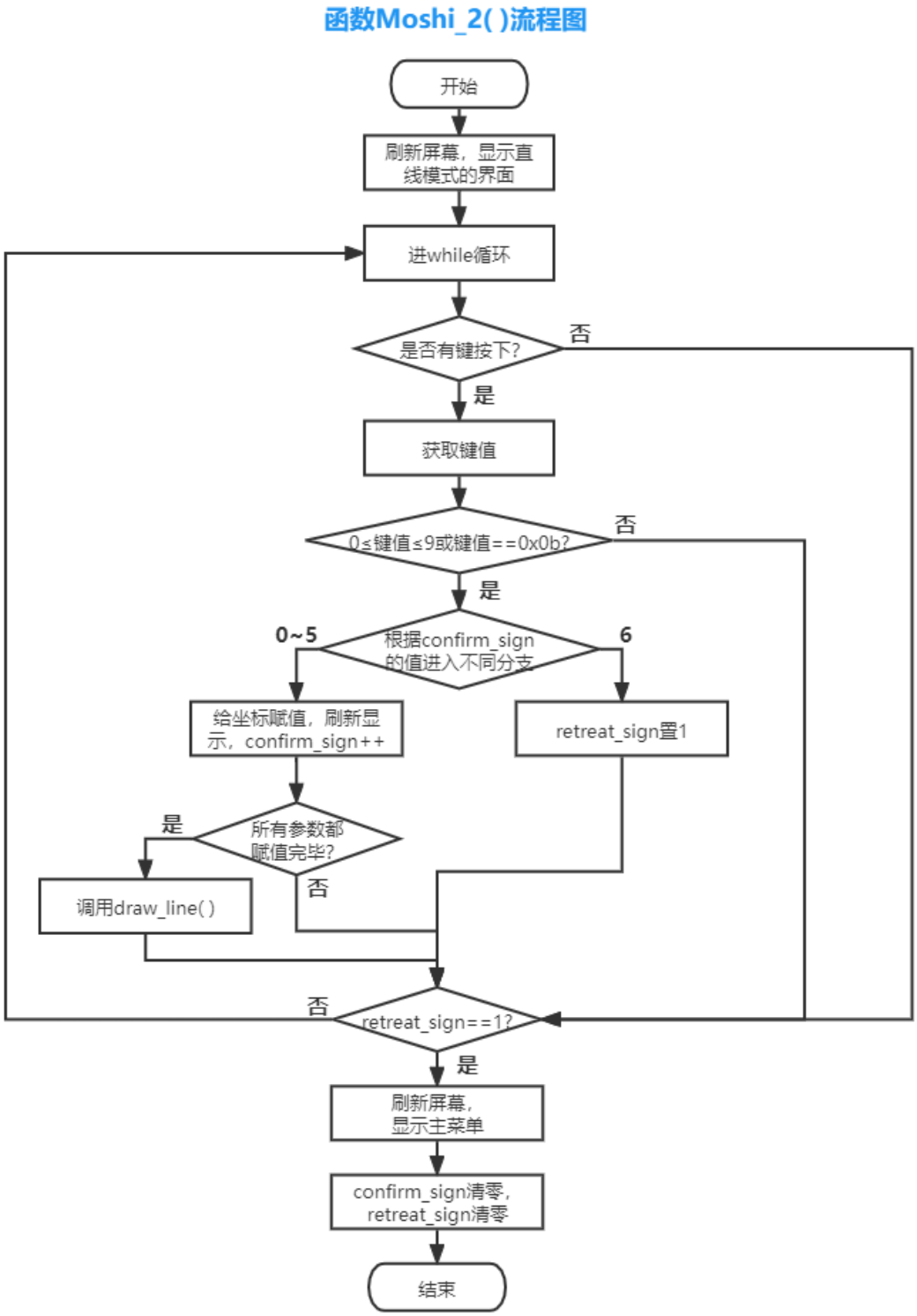


图 3- 17 函数 Moshi_2()流程图

3.5.3 直线插补模式运动设计

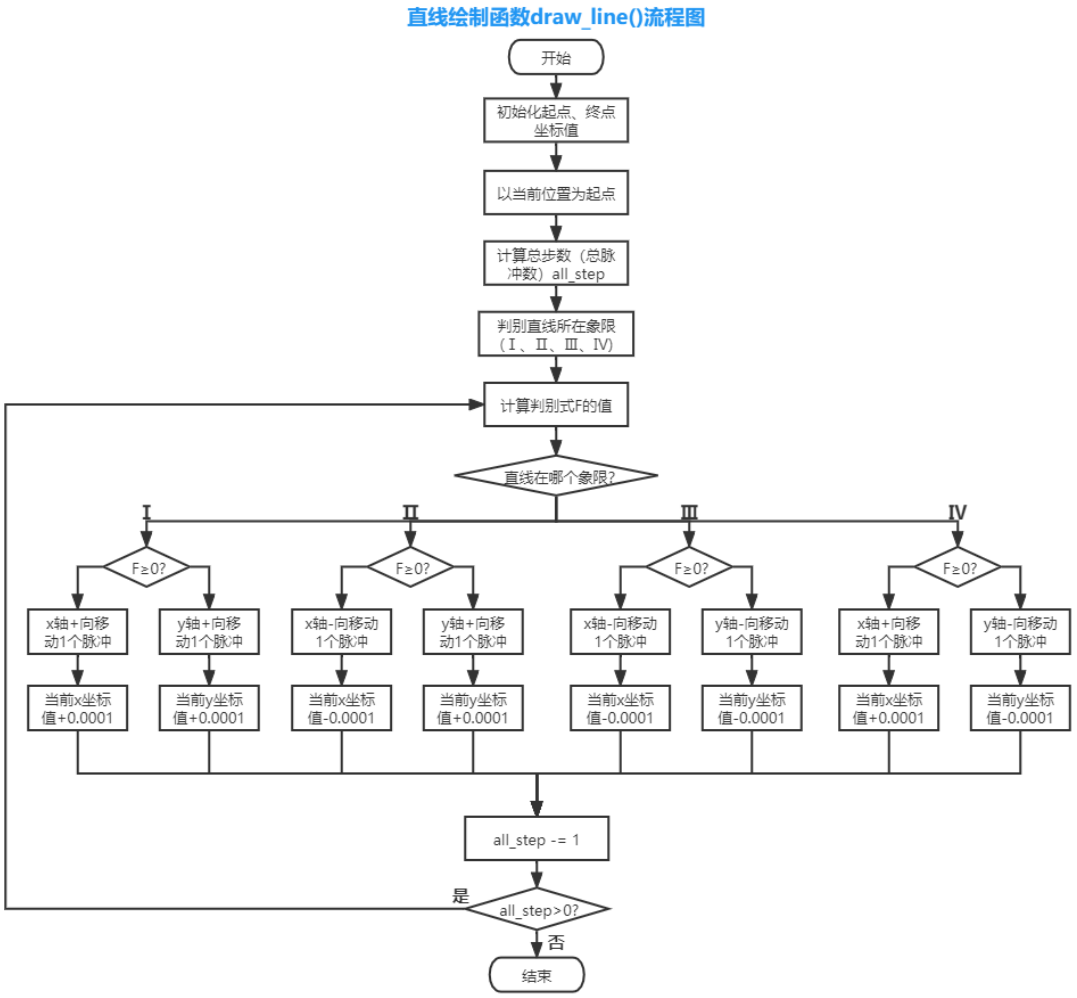


图 3- 18 函数 draw_line()流程图

3.5.4 直线插补模式人机交互设计

直线终点的横纵坐标均各有 3 位数字，单位均是 0.1mm。直线模式的 lcd 界面显示函数 void Display_2(int indexX1,int indexX2,int indexX3,int indexY1,int indexY2,int indexY3)的六个形参，分别对应终点横坐标 X0 的 1~3 位数值、终点纵坐标 Y0 的 1~3 位数值。每输入一位的值，lcd 都会刷新，当 6 位全部输完之后，便会开始画直线。绘制完毕，电机停转后，按下 b 键即可退出直线模式。

图 3- 19 直线模式的 lcd 界面

3.6 运动控制部分——圆弧/圆的绘制

3.6.1 圆弧插补原理

圆弧插补与直线插补的思想类似，也是用一系列的阶梯形状去近似，只不过这里近似的目标是圆弧而不是直线。

在平面直角坐标系中，以原点为圆心、半径为 R 的圆的标准方程为：$\frac{X^2}{R^2}+\frac{Y^2}{R^2}=1$

我们取判别函数为：$F = X^2 + Y^2 - R^2$

那么可知，当 F＞0 时，就说明点在圆外；F＜0 时，就说明点在圆内。

以逆时针方向画圆为例，分析应采取怎样的运动控制策略：

①当前点在第一象限（包含 Y 轴正半轴）时，

若 F≥0，则向 X 轴负方向移动一步；

若 F＜0，则向 Y 轴正方向移动一步。

②当前点在第二象限（包含 X 轴正半轴）时，

若 F≥0，则向 Y 轴负方向移动一步；

若 F＜0，则向 X 轴负方向移动一步。

③当前点在第三象限（包含 Y 轴负半轴）时，

若 F≥0，则向 X 轴正方向移动一步；

若 F＜0，则向 Y 轴负方向移动一步。

④当前点在第四象限（包含 X 轴负半轴）时，

若 F≥0，则向 Y 轴正方向移动一步；

若 F＜0，则向 X 轴正方向移动一步。

总结正圆、逆圆在四个象限的所有情况，可得如下图 3-20 和表 3-2：

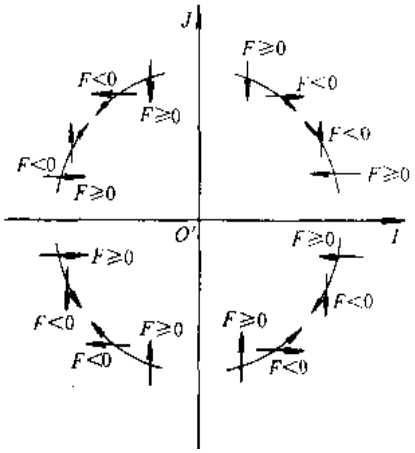


图 3- 20 四个象限内顺、逆圆加工时，判别式符号和进给方向关系

表 3- 2 象限判断和电机转向

		第一象限	第二象限	第三象限	第四象限
Ii 的符号		+	-	-	+
Ji 的符号		+	+	-	-
X 向电机	顺圆	+	+	-	-
	逆圆	-	-	+	+
Y 向电机	顺圆	-	+	+	-
	逆圆	+	-	-	+

圆弧插补的终点判断方法：

其思想与判断直线的终点方法相似，整个圆周的总脉冲数=8*半径*5000，那么圆心角为 θ 的圆弧的总脉冲数 all_step =（8*半径*5000*（θ/360°））

3.6.2 圆弧插补模式的总体设计

圆弧插补功能的核心函数 Moshi_4()的逻辑流程与直线模式基本一致：

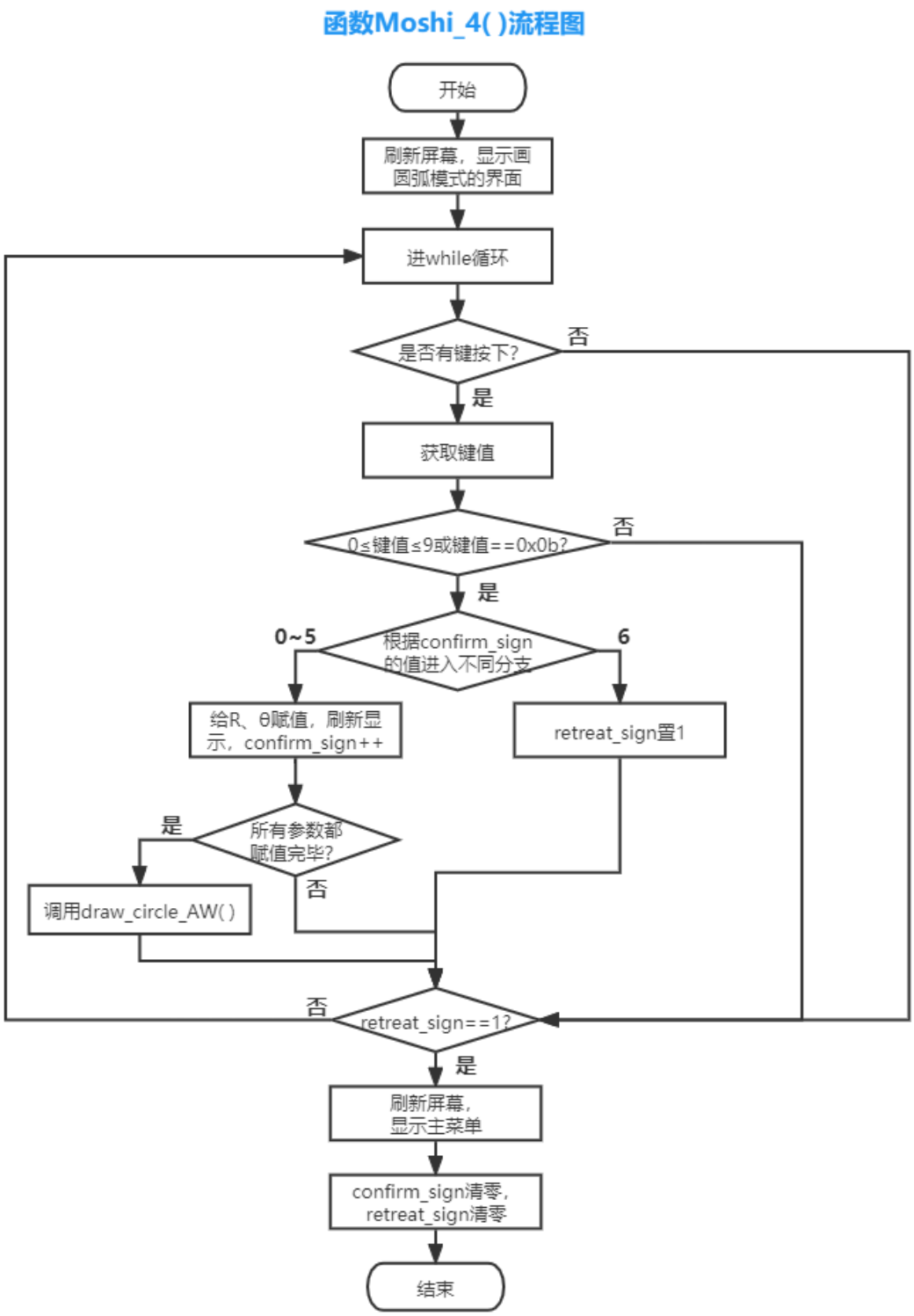


图 3- 21 函数 Moshi_4()流程图

3.6.3 圆弧插补模式的运动设计

以逆时针方向绘制任意半径和圆心角的圆弧的函数如下：

```
1.  //画一个任意圆心角、任意半径的弧/圆
2.  void draw_circle_AW(Cdt_circle circle,unsigned int speed)
3.  {
4.   long int count=8*circle.radius*5000*(circle.angle/360.0);
5.   double f;//判别式
6.   int keynum;
7.   //当前坐标等于起始点（以圆心为原点）
8.   x_now=circle.x_s-circle.x_center;
9.   y_now=circle.y_s-circle.y_center;
10.  //开始画圆
11.  while(count>0)
12.  {
13.   f=x_now*x_now+y_now*y_now-circle.radius*circle.radius;
14.   if(x_now>=0 && y_now>0) //第一象限
15.   {
16.    if(f>=0) {move_onestep('x','-',speed);x_now=x_now-0.0002;}
17.    else {move_onestep('y','+',speed);y_now=y_now+0.0002;}
18.   }
19.  else if(x_now<0&&y_now>=0)  //第二象限
20.   {
21.    if(f>=0) {move_onestep('y','-',speed);y_now=y_now-0.0002;}
22.    else {move_onestep('x','-',speed);x_now=x_now-0.0002;}
23.   }
24.  else if(x_now<=0&&y_now<0) //第三象限
25.   {
26.    if(f>=0) {move_onestep('x','+',speed);x_now=x_now+0.0002;}
27.    else {move_onestep('y','-',speed);y_now=y_now-0.0002;}
28.   }
29.  else if(x_now>0&&y_now<=0) // 第四象限
30.   {
31.    if(f>=0) {move_onestep('y','+',speed);y_now=y_now+0.0002;}
32.    else {move_onestep('x','+',speed);x_now=x_now+0.0002;}
33.   }
34.  count--;
35.  if(TestKey())
36.  {
37.   keynum=keyscan() & 0x0f;
38.   if(keynum==0x0b) { count=-1;} //b 键做退出键，按下后剩余步数置为-1
39.  }
40. }
41. }
```

3.6.4 圆弧插补模式的人机交互设计

圆弧插补与直线模式的人机交互结构相同，圆弧的半径 R 和圆心角 θ 均各有 3 位数字，R 的单位是 0.1mm， θ 的单位是度。lcd 界面显示函数 void Display_4(int indexR1,int indexR2,int indexR3,int indexAng1,int indexAng2,int indexAng3)有六个形参，分别对应 R 的 1~3 位数值、θ 的 1~3 位数值。每输入一位的值，lcd 都会刷新，当 6 位全部输完之后，便会开始执行画圆弧函数。绘制完毕，电机停转后，按下 b 键即可退出画圆模式。

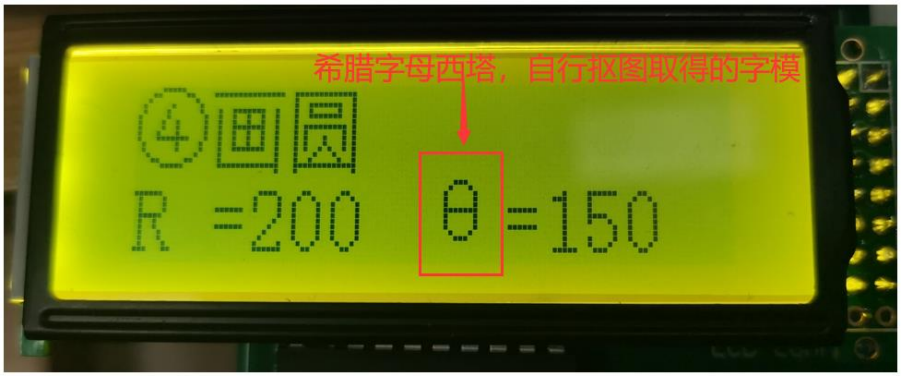


图 3- 22 圆弧模式的 lcd 界面

3.6.5 绘制结果展示

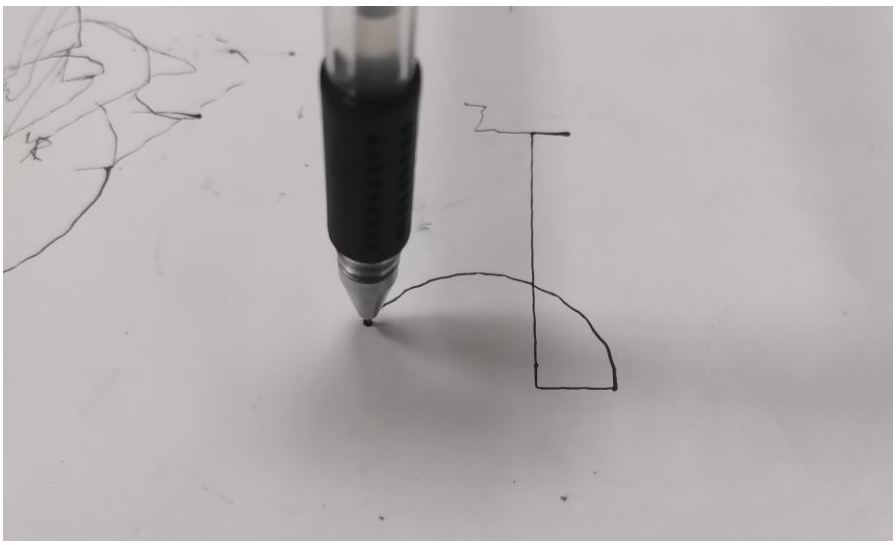


图 3- 23 绘制的 R=2cm， θ =150° 的圆弧

3.7 运动控制部分——椭圆绘制

3.7.1 椭圆插补的原理

平面直角坐标系中，焦点位于坐标轴上、半长轴和半短轴分别 a 和 b 的椭圆标准方程为：$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1(a>b>0)$

因为把坐标系内一点(x，y)代入椭圆标准方程，当点在椭圆内时，就有方程左边＜1；当点在椭圆外时，就有方程左边＞1；点在椭圆上时，就有方程左边=1 成立。因此我们取判别函数为：$F=b^2x^2+a^2y^2-a^2b^2$

那么可知，当 F＞0 时，就说明点在椭圆外；F＜0 时，就说明点在椭圆内。

椭圆插补与普通圆弧插补类似，也是分正逆圆、分四个象限，每种不同情况分别采取不同的移动策略。具体每种情况与表 3-2 所述相同。

3.7.2 椭圆绘制模式的整体设计

椭圆绘制的核心函数是 Moshi_3()，其逻辑结构与圆弧插补模式函数 Moshi_4 一致，不同之处仅在于 Moshi_3()中需要键盘设置的两个参数是 a 和 b，而非 R 与 θ，故不再另画流程图。

3.7.3 椭圆绘制模式的运动设计

参数 a 和 b 输入完之后即调用函数 darw_oval()画椭圆，darw_oval()与 draw_circle_AW()相比，核心差异是判别函数 F 不同、脉冲总数=4*(a+b)*5000，其他地方基本相同，故不再过多赘述。

3.7.4 椭圆绘制模式的人机交互设计

椭圆绘制模式的人机交互仅仅是界面所显示的汉字和参数名与圆弧模式不同，其他都一样。

图 3- 24 椭圆模式的 lcd 界面

3.7.5 绘制结果展示

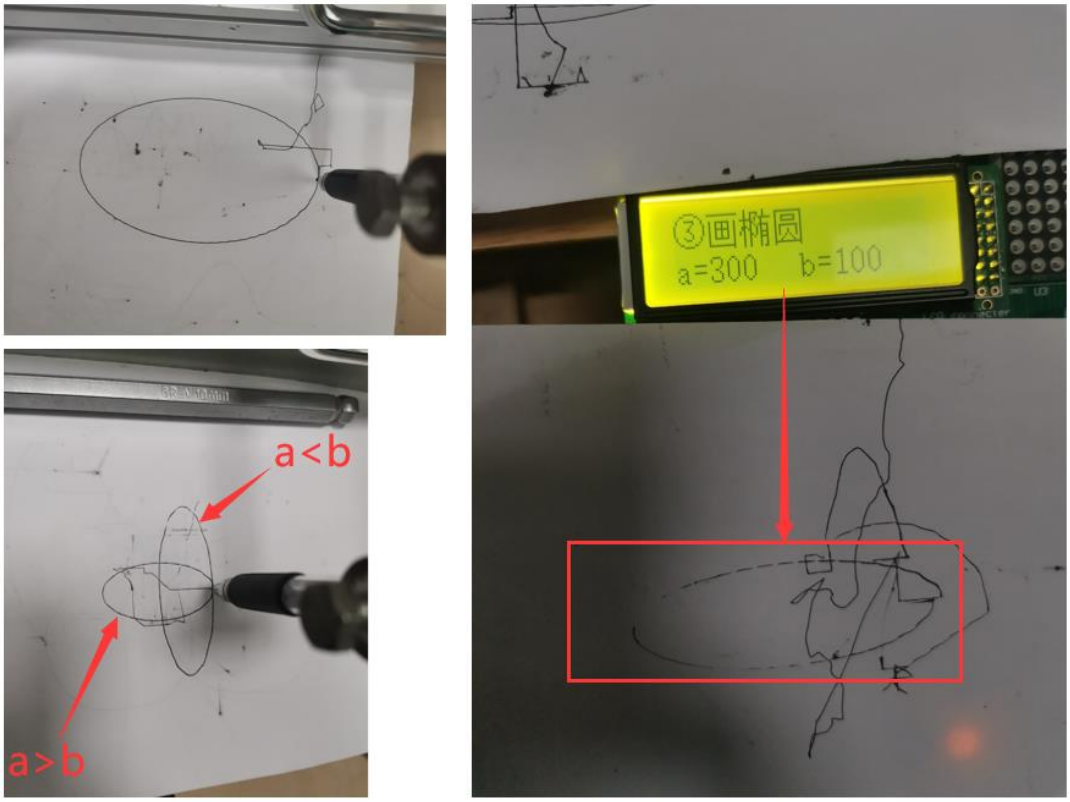


图 3- 25 椭圆绘制的结果展示

3.8 运动控制部分——sin 函数绘制

3.8.1 sin 函数插补原理

一样是用阶梯形状来近似曲线，根据当前 x 坐标所在的范围以及判别式 F 的符号来采取不同的运动策略，具体规律如下表：

表 3- 3

	0~(pai/2w)	(pai/2w)~(pai/w)	(pai/w)~3(pai/2w)	3(pai/2w)~(2pai/w)
F≥0	x，正	y，负	y，负	x，正
F＜0	y，正	x，正	x，正	y，正

sin 函数插补的终点判断方法：

依旧使用计算总脉冲数的办法，单周期的 sin 曲线的总脉冲数=(pai/w+2*A)*2*5000，每运动一步就使 count 减 1，当 count≤0 就结束绘制。

3.8.2 sin 函数插补总体设计

Sin 函数模式的核心函数是 Moshi_5()，与 Moshi_4()相比，差别仅在于 Moshi_5()中需要键盘设置的两个参数是振幅 A 和角频率ω、调用的绘图函数是 draw_sin()，其他部分两者基本一致，故不再另画流程图。

3.8.3 sin 函数插补运动设计

sin 函数插补的运动功能由 draw_sin()函数来执行，其流程图如下：

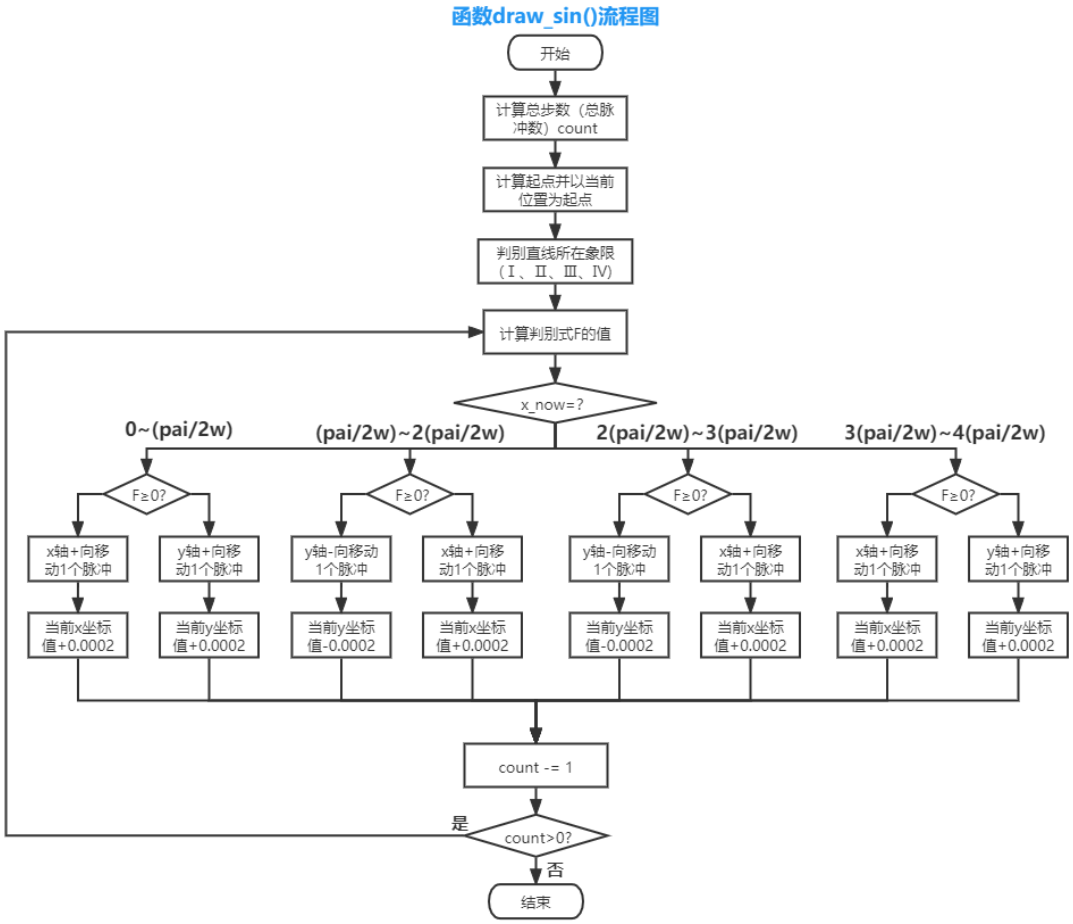


图 3- 26 函数 draw_sin()流程图

代码如下：

```
1.  void draw_sin(double A,double w,double fai,unsigned int speed)
2.  {
3.   unsigned long int count=(pai/w+2*A)*2*5000;//计算总步数（总脉冲数）
4.   double f;//判别式
5.   //设置当前坐标
6.   x_now=0;
7.   y_now=A*sin(w*x_now+fai);
8.   //开始画 sin
```

```
9.    while(count>0)
10.   {
11.    f=y_now-A*sin(w*x_now+fai);
12.    if(x_now <= (pai/(2*w))) //0~pai/2w
13.    {
14.     if(f>=0) {move_onestep('x','+',speed);x_now=x_now+0.0002;}
15.     else {move_onestep('y','+',speed);y_now=y_now+0.0002;}
16.    }
17.    else if(x_now <= 2*(pai/(2*w))) //pai/2w ~ pai/w
18.    {
19.     if(f>=0) {move_onestep('y','-',speed);y_now=y_now-0.0002;}
20.     else {move_onestep('x','+',speed);x_now=x_now+0.0002;}
21.    }
22.    else if(x_now <= 3*(pai/(2*w))) //pai/w ~ 3pai/2w
23.    {
24.     if(f>=0) {move_onestep('y','-',speed);y_now=y_now-0.0002;}
25.     else {move_onestep('x','+',speed);x_now=x_now+0.0002;}
26.    }
27.    else if(x_now <= 4*(pai/(2*w)) ) // 3pai/2w ~ 2pai/w
28.    {
29.     if(f>=0) {move_onestep('x','+',speed);x_now=x_now+0.0002;}
30.     else {move_onestep('y','+',speed);y_now=y_now+0.0002;}
31.    }
32.    count--;
33.   }
34.}
```

3.8.4 sin 函数插补人机交互设计

这部分与圆弧模式的思路是一致的，只是 lcd 显示的具体内容稍有不同、显示的变量名不同。

图 3- 27 sin 模式的 lcd 界面

3.8.5 绘制结果展示

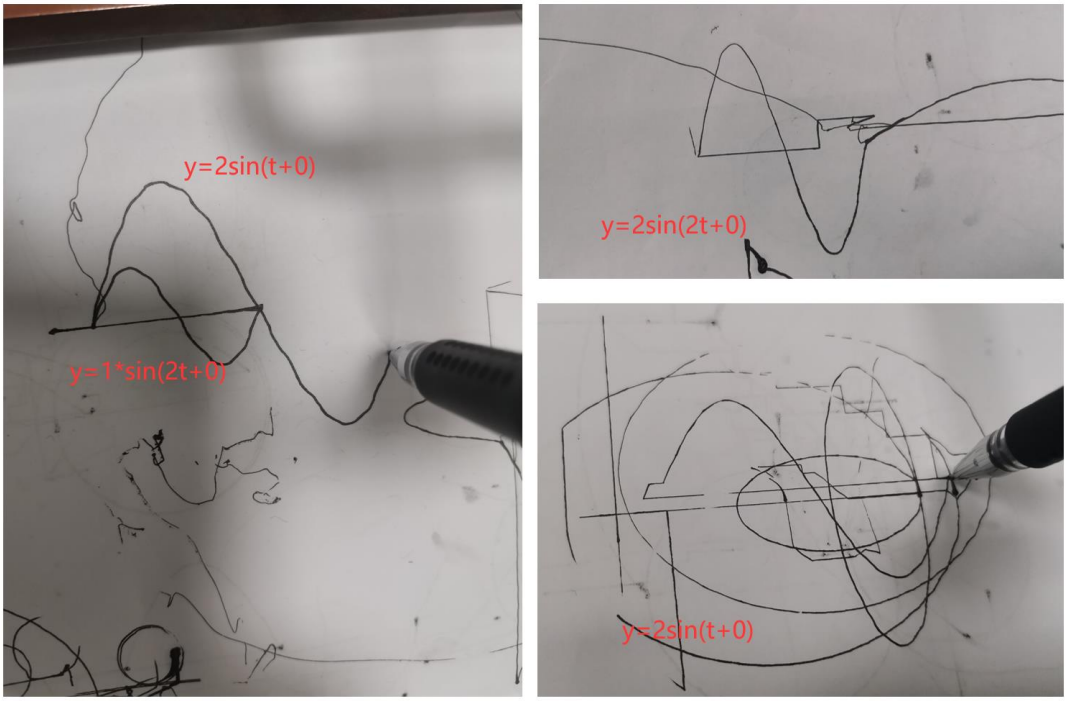


图 3- 28 sin 函数绘制结果展示

3.8.6 draw_sin()函数的改进——任意初相

通过总结 3.8.1 节的表 3-3，我们可以发现，运动策略概括起来主要分两种情况：上升段和下降段。

那么这个优化问题的关键就在于，当初相不同的时候，怎么构造统一形式的表达式来判断当前 x 坐标处函数 y=Asin(ωt+φ)是上升还是下降？而导数的正负不就正好可以判断斜率的正负嘛！就能判断当前处于 sin 函数的上升段还是下降段。对原三角函数求导得：y'=Aωcos(ωt+φ)

表 3- 4

	cos(ωt+φ)>0	cos(ωt+φ)<0	cos(ωt+φ)==0
F≥0	x，正	y，负	x，正
F＜0	y，正	x，正	x，正

改进后的 draw_sin2()函数代码如下：

```
1.  //sin 曲线进阶版（任意初相）
2.  void draw_sin2(double A,double w,double fai,unsigned int speed)
3.  {
4.   unsigned long int count=(pai/w+2*A)*2*5000;//计算总步数（总脉冲数）
5.   double f;//判别式
6.   //设置当前坐标
7.   x_now=0;
8.   y_now=A*sin(w*x_now+fai);
9.   //开始画 sin
10.  while(count>0)
11.  {
12.   f=y_now-A*sin(w*x_now+fai);
13.   if(cos(w*x_now+fai)>0) //上升段
14.   {
15.    if(f>=0) {move_onestep('x','+',speed);x_now=x_now+0.0002;}
16.    else {move_onestep('y','+',speed);y_now=y_now+0.0002;}
17.   }
18.   else if(cos(w*x_now+fai)<0) //下降段
19.   {
20.    if(f>=0) {move_onestep('y','-',speed);y_now=y_now-0.0002;}
21.    else {move_onestep('x','+',speed);x_now=x_now+0.0002;}
22.   }
23.   else { move_onestep('x','+',speed);x_now=x_now+0.0002;} //波峰波谷
24.   count--;
25.  }
26. }
```

因为在学校那几天非常忙，有一天夜里尝试过做这个优化，不过可能是因为熬夜时状态不好，脑子不清醒，在那儿总结了半天规律，最后也没能成功画出任意初相的 sin 曲线。这部分改进内容是回家后想出来的，所以暂时没有机器能够测试，开学之后一定要去 204 实验室验证一下!

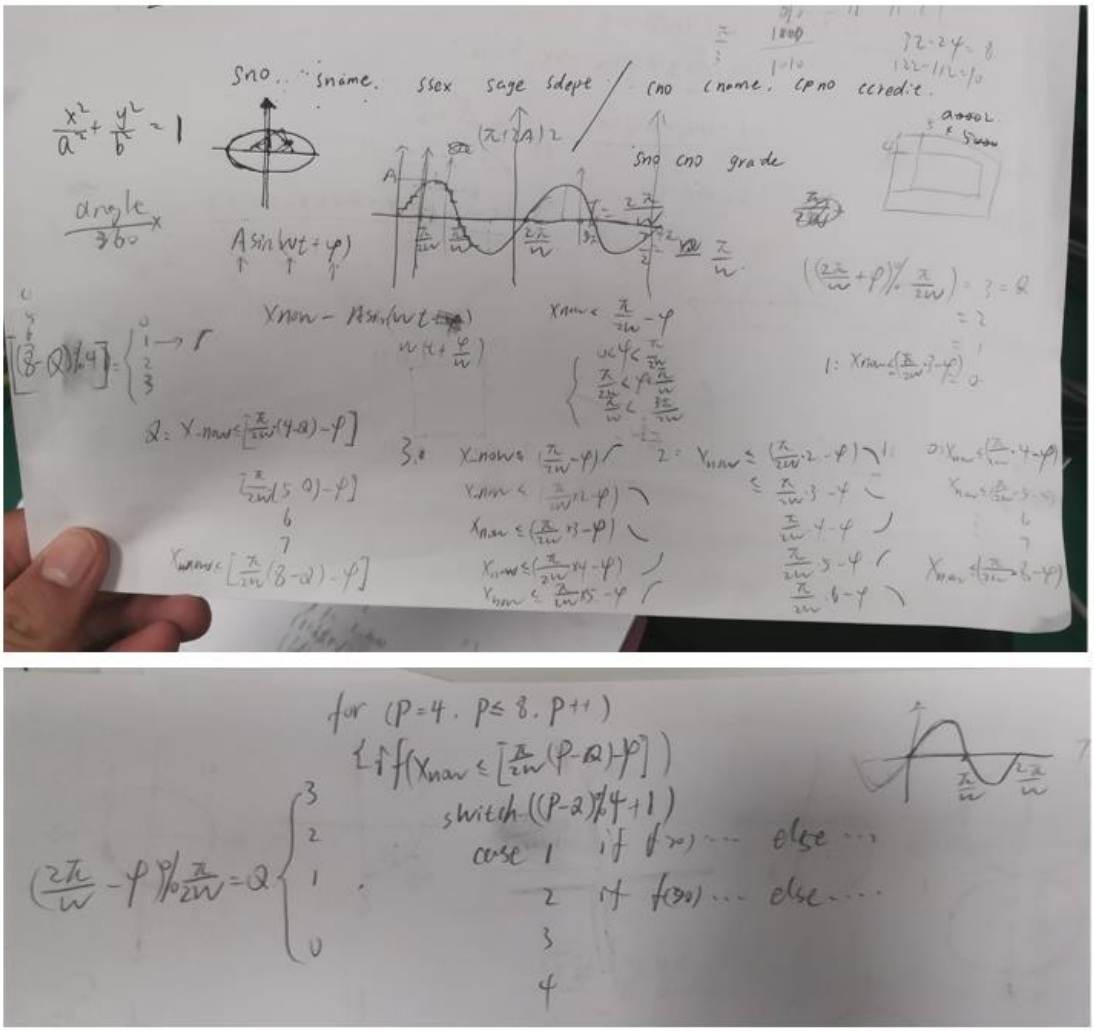


图 3- 29 当初尝试写任意初相的 draw_sin()的手稿

3.9 任意函数曲线绘制的灵感

3.9.1 任意函数曲线插补原理

在利用导数实现了任意初相的 sin 函数绘制之后，于是我延伸地想到：其实对任意一个函数也能够实现它的绘制了，关键就是要利用求导！我们一般意义上的函数，是指对每一个自变量，我们都有一个唯一的函数值与之对应，所以 sin 是函数而圆它就不是函数。那么对于函数的绘制，我们就可以根据 sin 的绘制触类旁通了：其实对任一个函数就分三种情况：导数大于零、小于零、等于零，采取 3 种运动策略（见表 3-4）。对，原来就这么简单！

插补终点的判断：用当前 x 坐标，当 x_now≥终点 x 的坐标时，插补结束。

对于求导，牛顿莱—莱布尼茨法在 c 语言中是难以实现的，因此，我们采用定义法来实现对任意函数的求导。

3.9.2 任意函数曲线插补代码

代码如下：

```
double myfunc(double x) //把待绘制函数封装成“函数”
{
 double value=0;
 value=sin(x); //待绘制的函数，以 sin 为例
 return value;
}

double differentiate(double x0) //求导函数
{
 double dx,dd1,dd2;
 dx=0.01;  //设 dx 初值
 do{
   dd1=(myfunc(x0) - myfunc(x0+dx))/dx;  //计算导数 dd1
   dx = 0.5 * dx; // 减小步长
   dd2=(myfunc(x0) - myfunc(x0+dx))/dx;  //计算导数 dd2
 }while(abs(dd1-dd2) >= 1e-06); //判断新旧导数值之差是否满足精度
 return dd2;
}

//任意函数曲线绘制
void draw_random_func(double x1,double x2,unsigned int speed) //起点 x、终点 x、速度
{
 //这里用当前 x 值 x_now 来判断插补终点，不再用计算总脉冲数的办法
 double F;//判别式
 //设置当前坐标
 x_now=x1;
 y_now=myfunc(x_now);
 //开始画绘制
 while(x_now<=x2) //只要 x_now 还没到达终点 x2 便继续运动
 {
  F=y_now-myfunc(x_now); //任意函数的判别函数 F 的表达式
  if(differentiate(x_now)>0) //上升段
  {
   if(F>=0) {move_onestep('x','+',speed);x_now=x_now+0.0002;}
   else {move_onestep('y','+',speed);y_now=y_now+0.0002;}
  }
  else if(differentiate(x_now)<0) //下降段
  {
   if(F>=0) {move_onestep('y','-',speed);y_now=y_now-0.0002;}
```

```
    else {move_onestep('x','+',speed);x_now=x_now+0.0002;}
  }
  else { move_onestep('x','+',speed);x_now=x_now+0.0002;} //波峰波谷
 }
}
```

3.9.3 任意函数曲线插补总结

不过，现在有一个很大的局限就是，还只能通过在程序代码中去写函数表达式来完成不同函数的绘制，还不能实现用户输入表达式然后绘制，若要实现该功能，则需要做词法分析，还需要进行更多的学习，需要一定的时间。

3.10 系统拓展部分——上位机发送指令与接受反馈

3.10.1 功能概述

本部分内容由 xxx 同学负责，实现了上位机发送指令，控制电机绘制任意长度的直线，任意半径的圆，并可控制绘制速度。开始绘制和结束绘制时，单片机向电脑发送开始绘制和绘制完成的信息。

3.10.2 效果展示

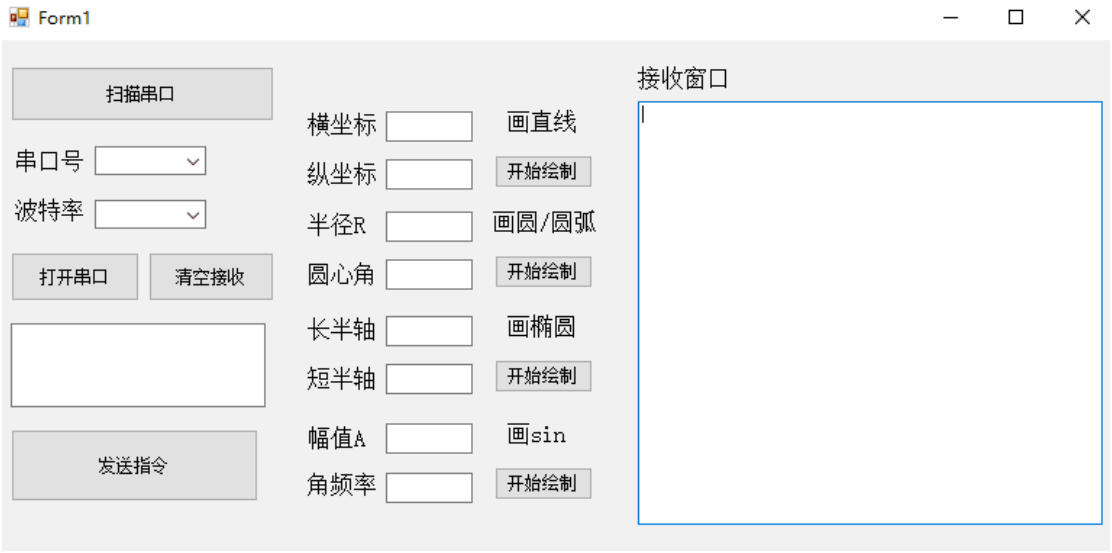


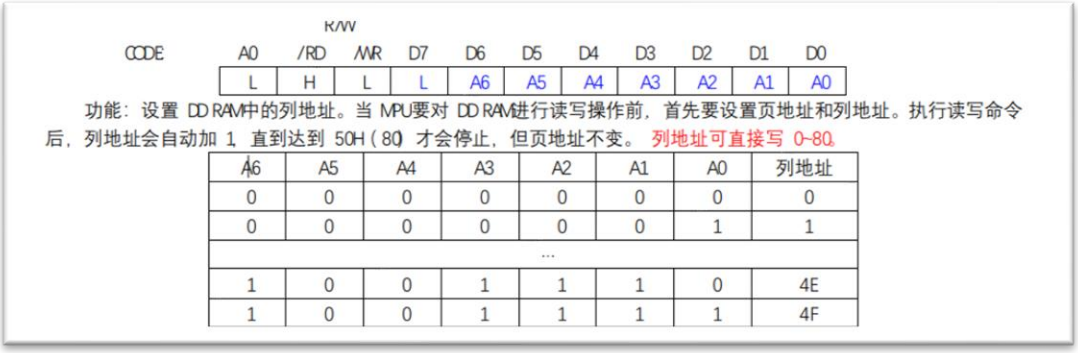
图 3- 30 上位机界面

3.11 遇到的问题及解决

（1）122*32 的手写体“中国地质大学”图片无法正常显示（如图）

图 3- 31 122*32 的图片显示出错

问题分析：因为这块 lcd 是 122×32 像素的， 所以我一开始直接把校名图片制作成一个 122*32 大小的字模，然后写了专用的 122*32 字模显示函数，但每次显示出来都是如上图的错误，我反复检查，取模以及程序都没有问题，困扰了我好几天之后， 我突然想起上学期李老师发过这个液晶的说明书，通过阅读说明书，我终于知道，原来虽然这块屏共有 122 列，但是因为寄存器值的定义表中只定义了 0～80，所以单个字模的最大宽度其实被限制为了 80 列！所以呀，当软件找不出问题原因时，最好的办法就是马上去读硬件资料。



R/W

CODE	A0	/RD	/WR	D7	D6	D5	D4	D3	D2	D1	D0
	L	H	L	L	A6	A5	A4	A3	A2	A1	A0

功能：设置 DD RAM中的列地址。当 MPU要对 DD RAM进行读写操作前，首先要设置页地址和列地址。执行读写命令后，列地址会自动加 1，直到达到 50H（80）才会停止，但页地址不变。 列地址可直接写 0-80。

A6	A5	A4	A3	A2	A1	A0	列地址
0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1
				...			
1	0	0	1	1	1	0	4E
1	0	0	1	1	1	1	4F

图 3- 32 恒科液晶说明书截图

问题解决：于是我把“中国地质大学”拆成两左右两幅图，每一幅宽 60，终于显示成功了！

（2）手动模式下只能按一下动一下，但我希望实现一直按着就一直动

问题分析：原本的按键检测函数中有一句 do{Delaykey(10;)}while(TestKey());用来检测按键是否释放，只要按键还没有释放，程序就会一直停在这个循环中等待，无法重新读取按键值，所以每按一下只能移动一步，不能连续运动。

问题解决：于是我另写了一个 keyscan2()，与原版唯一的差别就是把等待按键松开的语句去除了，计算出键值的索引之后，就直接返回键值。这样，就实现了按下按键未松开时，也能扫描获取键值。

但是这个 keyscan2()函数不宜用在参数输入中，因为可能导致我只按了一下按键，却被连续读取，使该参数的所有位都被设置成了这个按下的数字。所以在其他地方应该依旧使用原来的按键扫描函数 keyscan()，避免重复读取导致出错。

（3）手动模式下，按键加减速反应不灵敏，经常按了后 lcd 却没变化，要长按之后再松开才能看到变化，但长按又容易调过头。

问题分析：因为使用了不必等按键松开就可获取键值的 keyscan2()函数，所以导致加减速识别得不准确。为了实现长按长动，keyscan2()还是不能丢的，那就要改代码，让程序走到 switch 下的加速/减速分支里之后，按键不松开就不能继续往下走。

问题解决：更新速度值、刷新屏幕之后，加一条循环等待语句，如第 5、第 12 行。

```
1.          case 0x0c: //c 键加速
2.           {
3.            SDspeed-=1;
4.            Display_1(SDspeed,keynum);
5.            do{Delaykey(10);}while((keyscan2())==0x0c);
```

```
6.          break;
7.         }
8.        case 0x08: //8 键减速
9.         {
10.         SDspeed+=1;
11.         Display_1(SDspeed,keynum);
12.         do{Delaykey(10);}while((keyscan2())==0x08);
13.         break;
14.        }
```

（4）开始绘制之前，在通过键盘输入直线、圆、椭圆、sin 的参数时，发现 lcd 上显示的内容经常乱码

问题分析：检查过数字字模取得没有问题，Lcd 显示部分的代码也没有错误，那就说明是按键识别出错了。

问题解决：于是我把输入参数部分用一个 if 语句套起来，只有当返回的键值是 0~9 当中的一个时，才会把这个值赋给参数，当键值是 0x0b（做返回键）时将退出标志置 1。

（5）输入参数出错，比如正在设置圆的半径时，只按下数字键 3 一次，结果从 lcd 上看到半径 R 的所有位都变成了 3。

问题分析：参数赋值语句处于 while 循环中，虽然只按了一次键，但会因为循环而重复执行赋值语句，于是后面的各个位也都被设置成了相同的数值。要解决这个问题，就给参数的每个不同的位的幅值语句都设置一个进入条件。

问题解决：这里我用了一个变量 confirm_sign，当半径的任何一位都没有被设置时，confirm_sign==0，半径的第一位被赋值后 confirm_sign==1，第二位被赋值后 confirm_sign==2…以此类推。用一个 switch 分支根据 confirm_sign 的不同值，把当前获取的键值赋给参数的不同位。

```
if((keynum>=0&&keynum<=9)||keynum==0x0b)
{
  switch (confirm_sign)
  {
    case 0:{ if(keynum!=0x0b){indexX1=keynum;confirm_sign=1;Display_2(indexX1,indexX2,indexX3,indexY1,indexY2,indexY3);}break;}
    case 1:{ if(keynum!=0x0b){indexX2=keynum;confirm_sign=2;Display_2(indexX1,indexX2,indexX3,indexY1,indexY2,indexY3);}break;}
    case 2:{ if(keynum!=0x0b){indexX3=keynum;confirm_sign=3;Display_2(indexX1,indexX2,indexX3,indexY1,indexY2,indexY3);}break;}
    case 3:{ if(keynum!=0x0b){indexY1=keynum;confirm_sign=4;Display_2(indexX1,indexX2,indexX3,indexY1,indexY2,indexY3);}break;}
    case 4:{ if(keynum!=0x0b){indexY2=keynum;confirm_sign=5;Display_2(indexX1,indexX2,indexX3,indexY1,indexY2,indexY3);}break;}
    case 5:
          { if(keynum!=0x0b)
            {
              indexY3=keynum;Display_2(indexX1,indexX2,indexX3,indexY1,indexY2,indexY3);
              confirm_sign=6;
              Cdt_test.x_end= indexX1*1+indexX2*0.1+indexX3*0.01;
              Cdt_test.y_end= indexY1*1+indexY2*0.1+indexY3*0.01;
              draw_line(Cdt_test, SDspeed);
            }
           break;
          }
    case 6: { if(keynum==0x0b){retreat_sign=1;confirm_sign=7;}break;}
  }
}
```

（6）绘制圆心角小于 360 度的圆弧时，机器总是一动不动。

问题分析：原来是判别式中，我没有注意两个 int 型数相除之后还是 int 型，所以只要角度值小于 360，(θ /360)的结果都是零，最后求得的总脉冲数也就是 0，所以循环只执行了一次就结束了，肉眼看起来就像没有动似的。

问题解决：把判别式 F 中的 360，改成 360.0。

```
//画任意圆弧
void draw_circle_arc(Cdt_circle circle, unsigned int angle,unsigned int
{
  //Cdt_struct_now Cdt_now;//当前坐标
  //long int count=8*circle.radius*5000;
  long int count=8*circle.radius*5000*(angle/360.0);
  double f;//判别式

  int keynum;

  //当前坐标等于起始点（以圆心为原点）
```

图 3- 33 任意圆弧的总脉冲数计算

（7）画完一条直线之后，退回主菜单，再次进入直线模式，却发现 lcd 上显示着上一次的终点坐标值。

问题分析：因为我把终点坐标值放在全局变量中，而退出直线模式前没有将该全局变量清零。

问题解决：在退出直线模式前添加将该全局变量清零的语句。

（8）当我们的工程写好了两三个功能之后再写上新的功能，再编译，结果烧录就异常了，提示“文件大小超出程序区范围，超出部分已被自动移到 EEPROM 区”，这时烧录完之后，板子上程序是完全不能正常运行的，具体表现为：lcd 没有显示，按键无反应等。

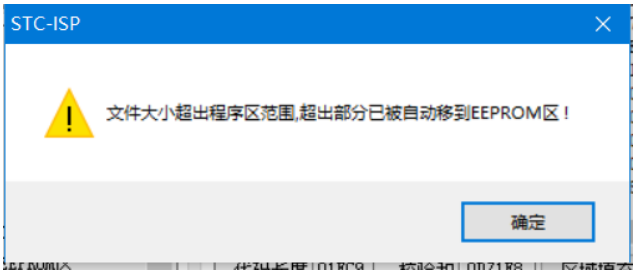


图 3- 34 程序写爆了之后烧录软件的提示

问题分析：开始时候 lcd 显示部分我写的内容比较多，于是我砍掉了一些显示内容，删了一些字模库，但发现还是杯水车薪，后续加点东西就又爆了。考虑到 52 芯片只有 8K 字节的程序空间和 512K 字节的 RAM，第三个功能还没写完就爆了，看来 52 是实在难以装下我们所有的功能模块。

问题解决：于是我们最后只得把系统拆分成 4 个不同的版本，每个版本搭在不同的模块，不用的部分就注释掉。

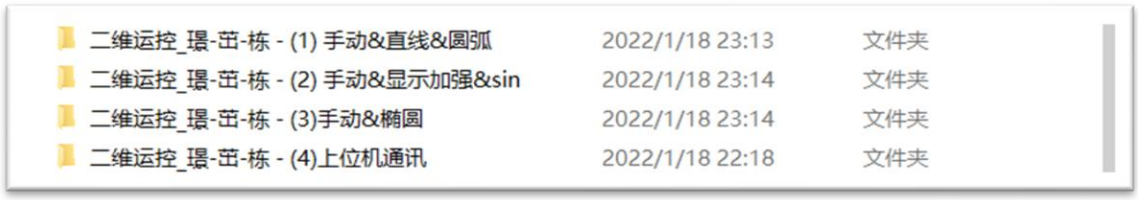


图 3- 35 我们组的四个版本的工程文件

3.12 小结

软件部分除了上位机，其他的内容我基本都有参与，所以对本次设计有一个比较全面的认识。我重点负责的是键盘输入和 lcd 显示、以及几个插补算法。都接触过一遍之后，我发现其实总地来说插补算法说到底就那么点东西，几个之间大同小异，只要把圆的插补弄清楚了，其他几种插补都可以触类旁通，比如椭圆的插补，只是把判别式 F 换了一下、sin 函数的插补依旧用到的是判别式&不同区域采取不同运动策略的思想、而直线插补则是根据斜率分出四个象限，即四种情况。画直线、圆弧、椭圆、sin 函数这四种模式的实现函数 Moshi_2()～Moshi_5()都具有一致的结构、他们的显示函数 Display_2～Display_5 也都具有一致的结构。不过整个系统要协调工作无 bug 还是经过了不少修改，写程序就是个逐渐完善的过程。同时也要注意细节（比如 360.0），不然就可能给自己制造麻烦，增加工作量。

最令我们头疼的问题还是 52 单片机内存太小，导致不得不把系统拆分成四个不同的工程来实现，下次做这种设计我一定要用 ARM，坚决不再用 51 了。

第四章 设想与展望

4.1 本次设计的不足

①使电机达到的最高速度仍旧较慢，导致绘图速度慢，图像稍微大一点就要等很久。

②52 单片机内存太小，使得无法将所有功能模块都集合到一个工程文件中，就是这个问题使本次设计受到了很大的局限，给我们造成了不少麻烦。

③人机交互较为单调，主要是靠 lcd 和矩阵键盘，以及队友后来开发出的上位机通讯。

4.2 改进方案

（1）弃用 52 单片机，改用 ARM，选 STM32 或 S3C2440，STM32 课外接触过，S3C2440 则是课上学过的。

（2）增加当前坐标的实时显示，lcd 或者数码管。

（3）利用蜂鸣器、流水灯、数码管设计更多的人机交互，比如绘图期间亮灯、数码管显示绘图时间、绘图结束时蜂鸣器鸣响提醒等。

（4）增加上位机、串口屏、WiFi 或蓝牙模块通讯功能，实现在电脑和手机上下达指令、设置参数、实时显示绘制轨迹等。

（5）想尝试一下用 lcd 同步显示画笔绘制 sin 曲线的轨迹，大致思路我已经有了，原理也不复杂，如果能实现，这将非常有意思。

第五章 实习总结与体会

5.1 实习总结

通过本次二维运动控制实习，我主要

（1）学到了

①伺服系统的一般控制方法和基本技能、

②数控轮廓插补原理（包括直线、圆、椭圆、sin 函数），

（2）巩固了单片机相关的知识与嵌入式系统开发技术，

（3）了解了伺服运动控制系统常用的机械结构，

（4）锻炼了实践与动手能力，

（5）在实践中磨砺了意志，使我更加领悟到了永不放弃、百折不挠的进取精神。

5.2 实习体会

虽然是头一次接触伺服控制系统，但是这并没有成为我学习的阻碍。

就实习过程而言，本次实习经历了相当的麻烦和挫折，尤其是在前半阶段，即元旦之前，那段时间忙得不可开交，而且运控实习经常被各种事耽搁，我转专业补修的《工程制图》实训和上课就耽搁了两个半天、期末复习又耽搁了两天，那段时间还有其他的课和实验一并进行，一周七天，有四~五天都在实习或实验，同时还要忙着期末复习，所以那段时间就特别忙乱，熬到三四点已是常态。而且最令人崩溃的是，最初几天我编写键盘和 lcd 的程序时老是出 bug，刚改了一个又来一堆，导致内心十分挣扎，很多代码后来都改掉了，那段时期真的做了很多无用功，就像是一个傻子在沙地里挖坑，刚挖走坑底的沙，马上就又有旁边的沙子滑下来……反复挣扎却又反复失败，付出时间却没有进展的感觉真的让人煎熬。

真正开始有进展是到阳历新年之后，那时只剩下最后两门考试和两个课内实验、三个课内实验报告，压力相对小了一点，元月一号二号我和队友 xxx 都泡在 204 实验室搞这个实习，后来又熬了几个夜，才终于有了一写成果，不过比其他组还是落后不少，感谢老师在时间上的宽容，让我们能把设想的功能基本做完，虽然成为了倒数第二个验收的小组，但结局还算圆满。

就系统开发而言，没有 bug 是很重要的，而修改 bug 的过程可能是漫长而艰苦的，可能需要很长时间，也可能要改很多东西，在本次设计中，我们就遇到了诸如退不回主菜单、进不去子函数、键盘输入参数时 lcd 显示乱码、“中国地质大学”的整幅图片显示不成功等许多问题，也算是走了不少弯路，费了很多功夫，但当程序终于顺利运行的那时候，总会觉得辛苦都值得。同时我也明白了要学会取舍，先易后难，不要死磕一个问题，一时处理不了的难题就先放着，待后面抽空想一想，说不定什么时候就能想出解决方案（比如 sin 函数的改进）。

就团队协作而言，这次实习使我更体会到了什么叫“分工协作”。一个团队的成员之间既要相互配合，又要分工明确。每个人应该有各自分别负责的部分，不要两个人做同一个东西，这样才能提高效率、加快进展。一个部分也不一定就是一整个功能模块，可以只是实现该模块的部分环节。当遇到问题时，大家要相互配合，比如当队友遇到 bug 没有思路时，可以帮对方排除 bug（比如整个程序的二级结构），或者一方写的程序，由另一方加以改进（比如我们的任意圆弧插补）；队友相互之间也要保持良好的沟通，经常商量设计思路、预期目标，保持认识和想法上的一致性，这样才能劲儿往一处使，更好更顺利地完成任务。

就实习结果而言，本次设计的功能基本都达到了，最后的系统也较为完整，而且几乎没有什么 bug。虽然一些地方本可以做得更好，但可惜的是，受我们当前能力和时间的限制，未能实现。在以后的学习和实践中要争取做到更好！

就实习收获而言，还是那句话——“纸上得来终觉浅，绝知此事要躬行。”单纯的理论课学习是远远不够的，只有通过在实践中磨砺、加深、领悟，才能真正掌握一门知识、学会一种技能，本次实习很好地提升了理论课的学习效果，使我在成长为一名合格自动化人的道路上又迈出了坚实的一步。

虽然这次实习过程坎坷、心路艰难，但我没有放弃，咬牙坚持了下来，通过查资料、看视频、问同学问老师、自己钻研摸索，成功把遇到的问题一一解决，我希望把事情尽力做到最好！成长和进步就是在遇到问题和解决问题的循环往复中悄然发生的，在这个过程中不仅仅是学到知识，更重要的是锻炼自己解决难题的能力、思维以及永不言弃的意志品质，我想，这才是这段课设经历给予我最宝贵的东西。

就实习建议而言，想到一点：老师们不在实验室的时候，可以安排一两个助教进行指导，这样我们就不至于有问题的时候找不到人问，因为如果有人指导，往往可以少走很多弯路、加快进度。

几经辛苦，又翻过一座山头，再接再厉，前方定有更大的胜利！

最后，对在本次实习中辛勤付出的各位老师以及帮助过我的各位同学表示衷心的感谢！