基于图像识别的机械操作台

第一部分 设计概述

1.1 设计目的

随着科学技术的发展，越来越多的工业级加工设备推出了家庭版本，例如平面雕刻机，3D 打印机等一系列产品，目前工业上使用的切割机一种是传统的锯片切割，需要人手动控制材料成型，而高端的全自动切割机大部分使用的是激光切割，由于制作过程复杂，成本高昂导致其难以低端化，使得家庭级全自动切割机的市场一直处于空白状态。因此我们队采用钻头多层旋转切割的方式开发出了一款小型化的全自动切割操作台，利用机械方式避开激光，并且完成了自动切割，用于填补这一市场空白。

1.2 应用领域

①**家庭级的低端加工设备：**由于其制作成本低廉，操作使用简单，只需要简单的绘图即可以得到想要的切割图形，更加易于被普通用户接受。

②**自由开发人员的桌面级工具：**由于作品同时提供 AutoCAD 专业软件的接口，因此也同样适用于较为专业的开发人员，他们可以更加发挥出作品的性能。

③**小型企业的低成本代替方案：**由于业内通用解决方案成本相对较高，这个作品可以为很多小型企业提供一个低成本的加工方案。

1.3 主要技术特点

机械操作台的基础功能便是利用图像完成全自动切割。作品使用 NUCLEO-F767ZI 开发板，可以通过摄像头直接采集手绘图像或者利用脚本分析 Auto_CAD 软件绘制的图形，然后将数据通过串口发送到主控板，主控板接收到信号后将数据解析，然后将轨迹路线传出到控制电机部分，从而使电机精准定位，达到全自动切割的目的。

① **Corexy 结构：**机器整体采用 Corexy 结构，这种结构在很多工业机器中都得到了运用，具备高精度，稳定可靠，抗干扰等特点。

② **Bresenham 变化算法：**程序在 Bresenham 绘图算法的基础上加以改进，通过对轨迹的切割与拼接，使其适用于连续机械运动。

③ **DRV8825 电机驱动芯片：**作品采用德州仪器公司的 DRV8825 作为电机驱动芯片，拥有过流保护（OCP）、热关断（TSD）、VM 欠压锁定（UVLO）等保护功能。

④ **简单图形处理方式：**简单图形可通过手绘后摄像头会自动采集图像数据，然后通过算法得到移动轨迹，机器会自动处理完成操作。

⑤ **AutoCAD 图形接口：** 同时机器也针对开发者提供了 AutoCAD 软件接口来完成更加复杂图形的绘制，用户只需要在 AutoCAD 指定范围内绘制线条，完成后数据便会通过串口发送给主控芯片，最终完

成全自动切割操作。

⑥ **具备一定的可扩展性：**作品连接件全部由我们自己建模设计，可拆卸，能够外接打孔，焊接等模块，具备一定的可拓展性，适合于多种平台。

1.4 关键性能指标

①系统可实现两种识别方式：手绘图像识别与 CAD 图形识别。这两种图像识别方式均可控制步进电机的动作，测试结果表明，机器步进精度均可达 1mm。

②机身较市面上传统的切割机体积更小，更加轻便，如表 1 所示。

表 1 本机机身尺寸

结构尺寸	X,Y 轴底座	钻头连接件	整体结构
长	80mm	30mm	150mm
宽	80mm	30mm	150mm
高	150mm	45mm	40mm

③部分图形完成所需的切割时间如表 2 所示。数据表明，该工作台工作速度较快。

表 2 部分图形的切割时间

工作时间	整体时间	单位时间
切割矩形（40mm×30mm）	9min 23s	0.24mm/s
切割圆形（r=20mm）	7min 50s	0.26mm/s

第二部分 系统组成及功能说明

2.1 整体介绍

当用户使用摄像头或者 AutoCAD 软件绘制好轨迹图像后，数据通过串口传输至主控 NUCLEO-STM32F767ZI 开发板进行数据解析，然后通过 DRV8825 步进电机驱动模块驱动步进电机转动带动进行丝杆螺母直线运动，并且由电调控制无刷电机及钻头开始高速转动，通过多层轨迹切割将目标图形切割下来。系统框图见图 1 所示，主体 3D 模型图见图 2 所示。

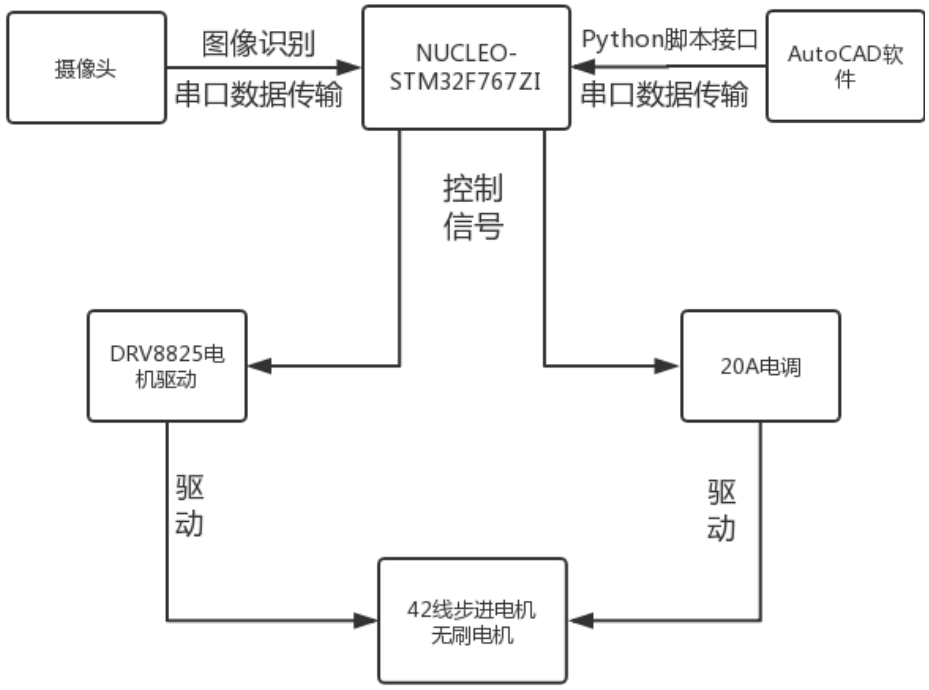


图 1 系统框图

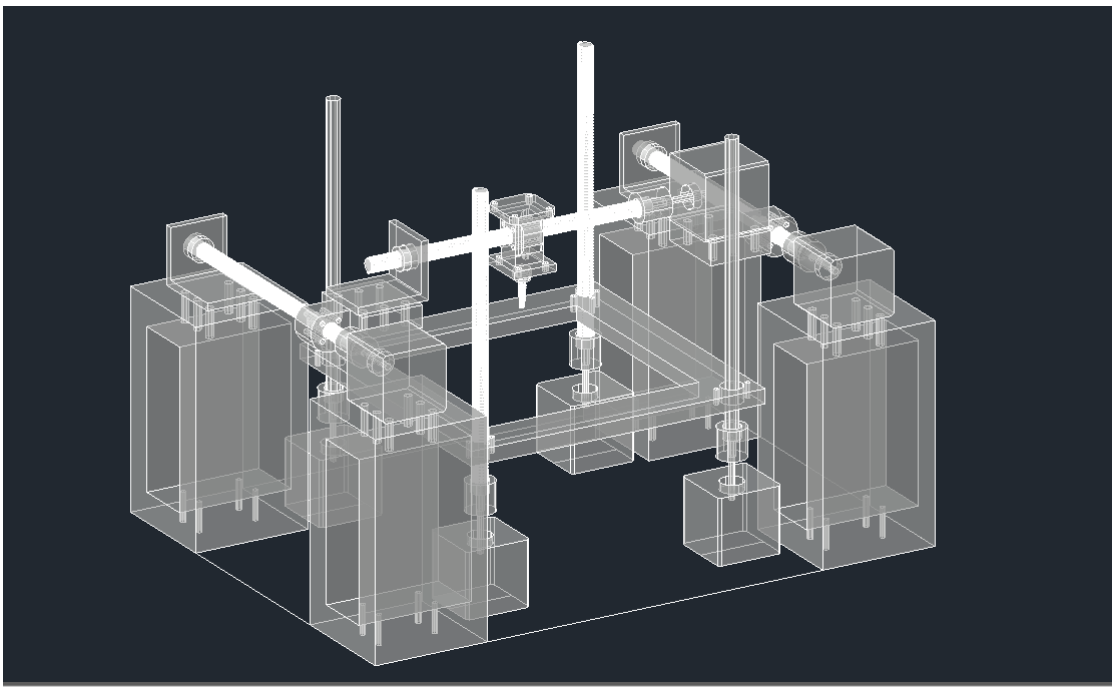


图 2 作品主体 3D 模型图

2.2 各模块介绍

2.2.1 主控芯片简介

NUCLEO-F767ZI 是 ST 公司推出的一款针对 STM32F7 系列设计的 Cortex-M7 Nucleo-144 开发板，支持 mbed，兼容 Arduino，还带有 ST Zio 和 ST Morpho 扩展接口，可连接微控制器的所有周边外设。

开发板基于 STM32F767ZI 设计，还集成了 ST-LINK / V2-1 仿真下载器（但仅对外提供 SWD 接口），免除另外采购仿真器或下载器的麻烦。并且具备 Arduino 的接口，可接入 Arduino 巨大生态系统的各种扩展板，让用户能够轻松快速增加特殊功能。

NUCLEO-F767ZI 开发板特点：

(1) 采用 LQFP144 封装的 STM32 微控制器,主频可达 216MHz

(2) 带 SWD 连接器的板载 ST-LINK / V2-1 调试器/编程器

(3) 三个用户 LED

(4) 两个按钮：USER 和 RESET

(5) 全面的免费软件 HAL 库包括各种软件示例

(6) 多种可供选择的集成开发环境（IDE），包括 IAR™，Keil 公司的支持® ，基于 GCC 的 IDE，ARM ® mbed™

2.2.2 DRV8825 步进电机驱动模块

电机驱动的选择决定了最终机器工作的精度以及准确度，并且对于这样一个多电机协同工作的工作台来说驱动的选择至关重要。

最终机器采用的是德州仪器的 DRV8825 步进电机驱动模块广泛应用于打印机，扫描仪和其他自动化设备上，该器件具有两个 H 桥驱动器和一个微步进分度器，用于驱动双极步进电机。输出驱动器模块由 N 沟道功率 MOSFET 组成，配置为全 H 桥，用于驱动电机绕组。DRV8825 能够从每个输出驱动高达 2.5 A 的电流（适当的散热，24 V 和 25° C）。简单的 STEP / DIR 接口可轻松连接控制器电路。模式引脚允许以高达 1/32 步模式全步配置电机。衰减模式是可配置的，因此可以使用缓慢衰减，快速衰减或混合衰减。提供低功

耗睡眠模式，关闭内部电路以实现非常低的静态电流消耗。可以使用专用的 nSLEEP 引脚设置此休眠模式。

驱动模块外围电路设计如图 3 所示：

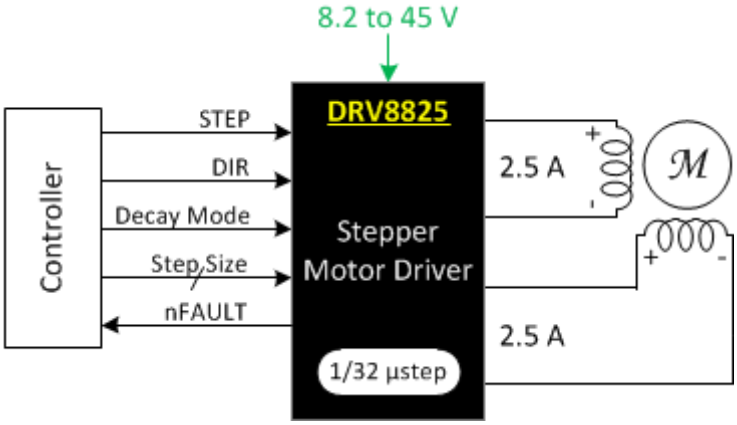


图 3 驱动模块外围电路

为了能够提高机器的一体化程度，我们专门设计了整块驱动转接板，电路图如图 4 所示：

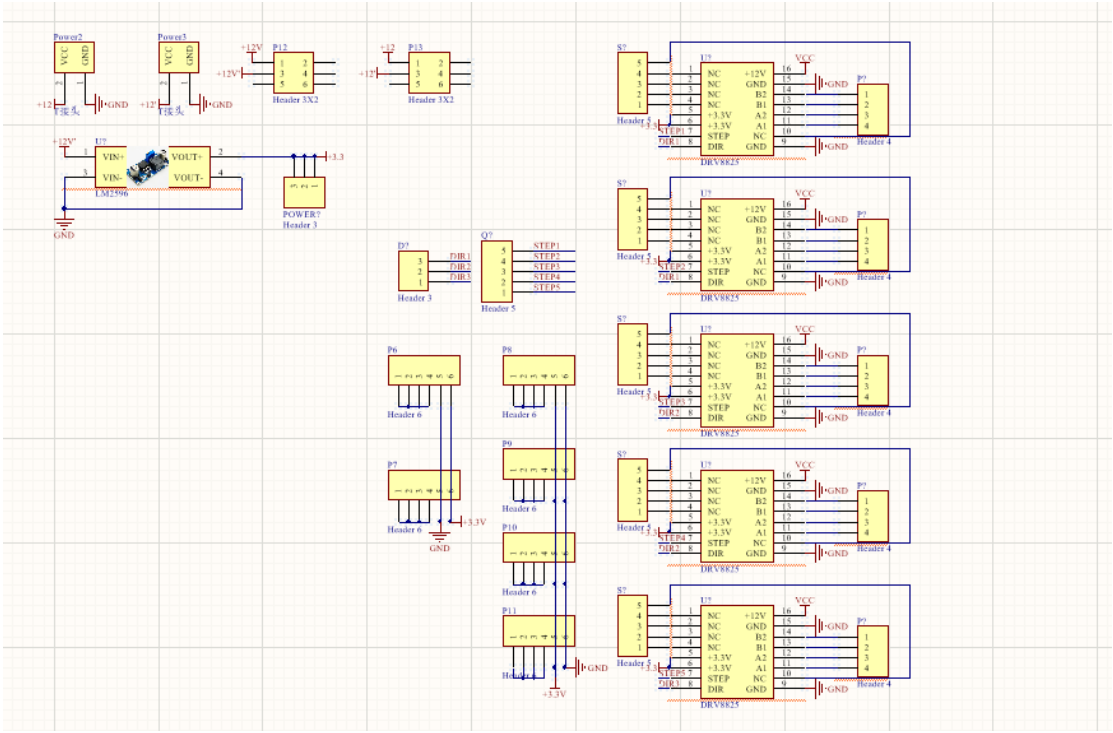


图 4 电机驱动转接板电路图

2.2.3 图像识别平台

在进行简单的图像识别时，我们选择了智能摄像头 OpenMV，OpenMV 搭载 MicroPython 解释器，这允许用户在嵌入式上使用 Python 来编程 (Python 3 to be precise)。由于其内置了众多算法，同时 Python 的语法也使机器视觉的编程变得简单得多。

通过 3D 打印设计的图像识别平台(如图 5 所示.)使得用户操作更加便利，只需要自行将图纸画好后放置在平台上，摄像头即可以识别出图像轨迹，然后提取出关键数据通过串口发送至主控芯片处理。

由于 OpenMV 属于开源器件，所以允许用户使用其专门的 IDE 去自定义程序，并且可以实时监测图像状态，大大提高了机器的可扩展性。

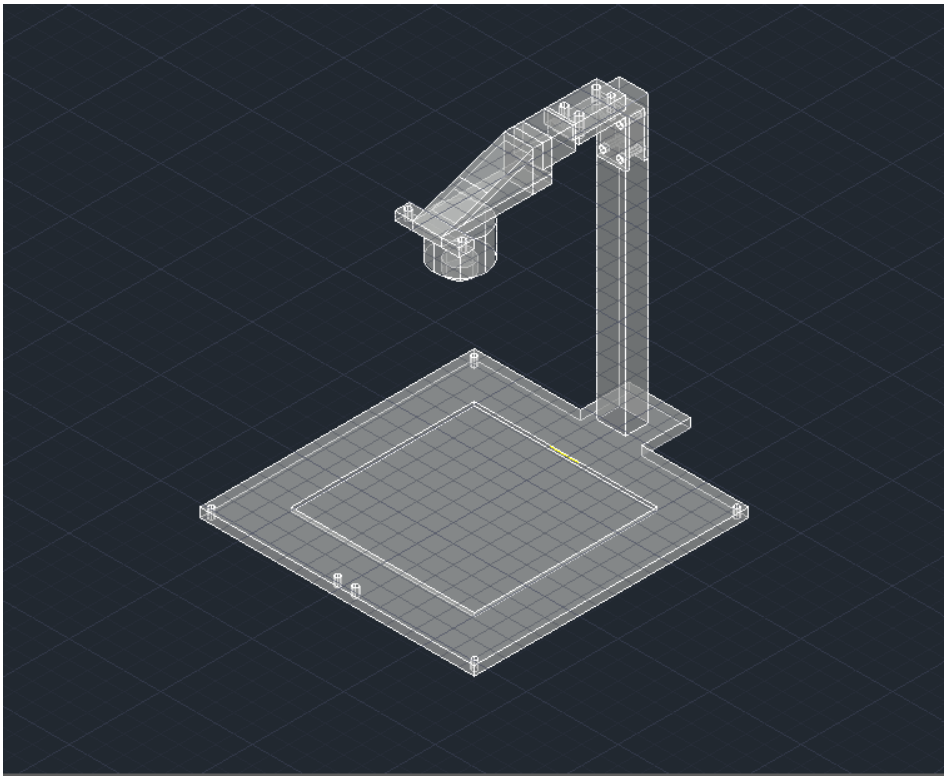


图 5 图像识别平台 3D 图

2.2.4 AutoCAD 软件 Python 脚本

AutoCAD（Autodesk Computer Aided Design）是 Autodesk（欧特克）公司首次于 1982 年开发的自动计算机辅助设计软件，用于二维绘图、详细绘制、设计文档和基本三维设计，现已经成为国际上广为流行的绘图工具。

为了兼容绝大多数开发者的使用习惯，机器专门为该软件预留了一个图像数据处理脚本，通过 Python 语言分析设计图内的对象，提取数据，然后依靠算法对数据进行处理后将数据写入串口，最后由主控芯片进行解析。

受面向对象编程思想的影响，可以将 CAD 图纸中的每一个图形都最为一个对象处理，依靠 Auto_CAD 的拆解指令，那么所有的图形都可以用多段线和圆弧来表示，然后提取出各个对象的重要数据后，写入对应的数据协议区。 CAD 图纸分析方法流程图见图 6 所示。

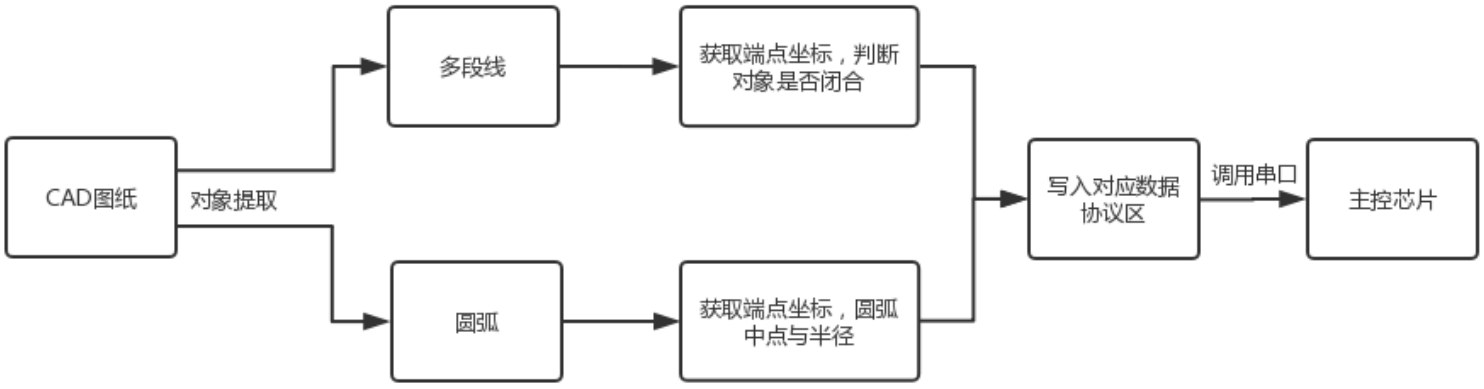


图 6 CAD 图纸分析方法流程图

整套脚本程序首先连接 Auto_CAD 软件，依靠上述的设计思路得到可靠图形数据，然后进行面积排序后可以分配好图形的切割顺序，接着依靠 Python 方法将所有的数据变换为同一列表，在列表对应的地方添加上帧头帧尾以便主控芯片接收分析，最后调用串口相关的方法找到可用的串口，这样再调用串口发送给主控芯片后主控即可以还原出对应的图形。

脚本运行的程序框图如图 7 所示。

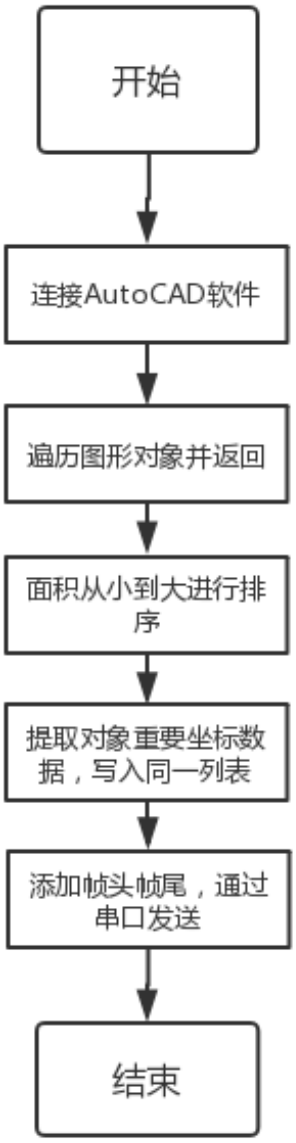


图 7 Auto_CAD 脚本程序框图

2.2.5 轨迹运动核心算法——Bresenham 变化算法

布雷森汉姆直线算法（英语：Bresenham's line algorithm，如图 8 所示）是用来描绘由两点所决定的直线的算法，它会算出一条线段在 n 维位图上最接近的点。是计算机图形学中最先发展出来的算法。经过少量的延伸之后，原本用来画直线的算法也可用来画圆。且同样可用较简单的算术运算来完成，避免了计算二次方程式或三角函数，或递归地分解为较简单的步骤。

以上特性使其仍是一种重要的算法，常常用在绘图仪、绘图卡的绘图芯片中，以及各种图形程式库。

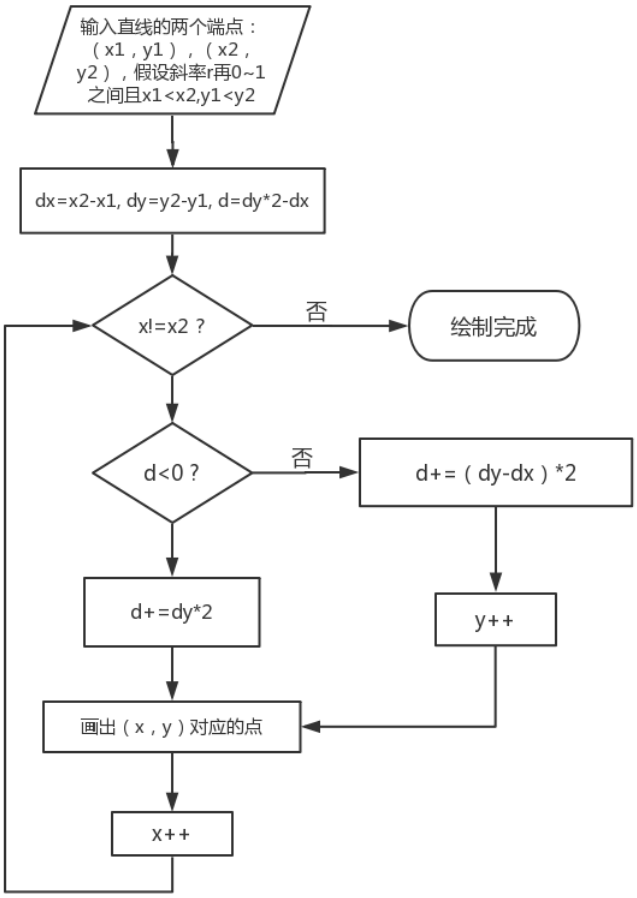


图 8 Bresenham 直线算法流程

为了可以在机器准确的按照轨迹执行，程序对 Bresenham 算法进行了改进，将坐标通过剪裁与拼接后模拟成连续量，然后交由步进电机执行。

以下以画圆算法为例进行简单介绍。

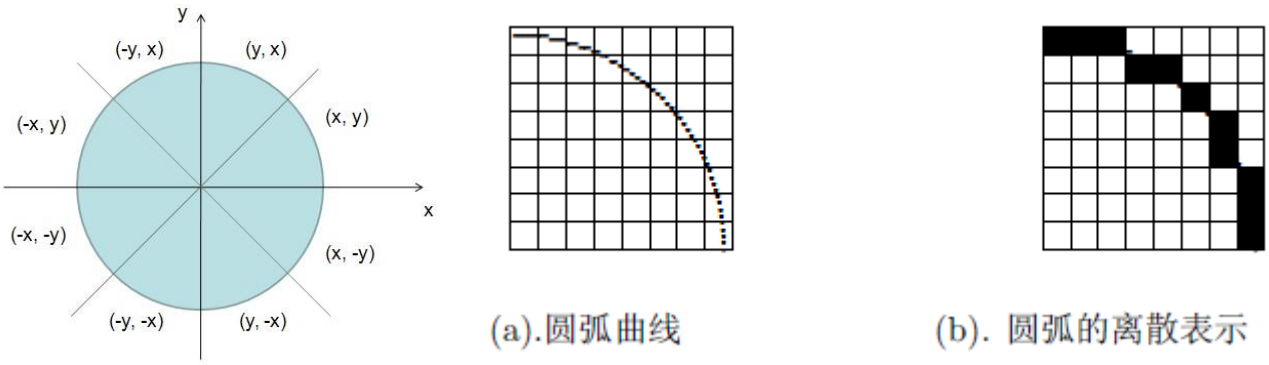


图 9 传统的 Bresenham 八分画圆算法表示

传统的 Bresenham 画圆算法是利用判别变量来判断选择最近的像素点，判别变量的数值仅仅用一些加、减和移位运算就可以计算出来。为了简便起见，考虑一个圆心在坐标原点的圆，而且只计算八分圆周上的点，其余圆周上的点利用对称性就可得到。但是由对称性得到的圆坐标具有跳跃性，不适合机器连续执行，如图 9 所示。

变化后的算法通过条件判别仅提取出圆的第一象限坐标点，然后通过冒泡排序消除传统算法的坐标跳跃性，接着利用图像旋转重新拼接出一个坐标连续的完整圆，再将其等价成多条连续直线后即可由机器执行(如图 10 所示)。切割结果表明，提取的圆较光滑，不会出现传统画圆时的跳跃。

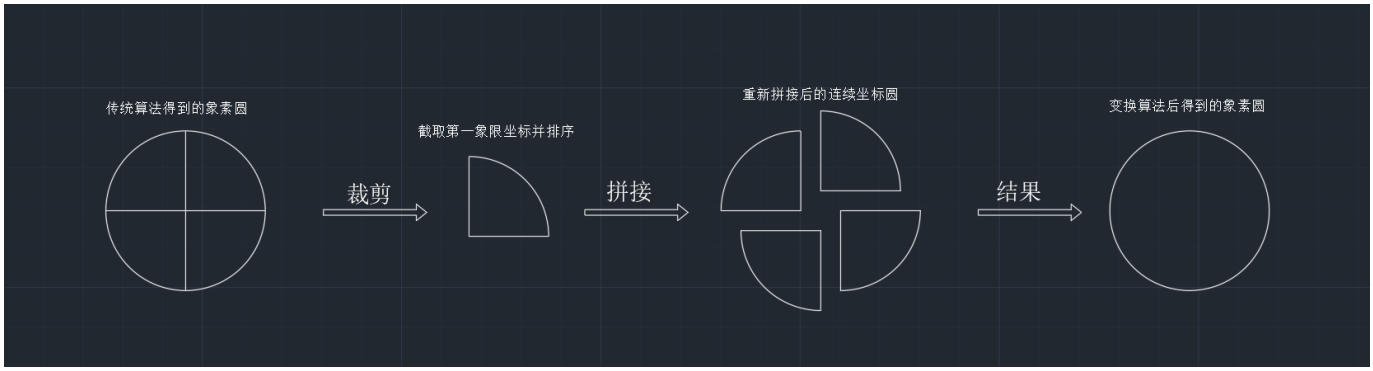


图 10 改良后的 Bresenham 变化画圆算法示意图

2.2.6 模仿图像显示的电机运动程序设计

为了保证各个电机在运动时的协同性，不会出现电机速度不同而导致轨迹失真的情况，运动部分的程序设计借鉴了图像在屏幕上的显示方式——将每个坐标等价成一个像素点。在获取了各个像素点坐标值后，不是直接提取首尾坐标进行运动，而是离散性地在每一个像素点间运动，这样子不仅使得运动轨迹更加准确可靠，也在宏观上保证了运动的连续性。

具体做法是配合开发板的中断功能，在传入相邻坐标值后计算出电机的步进距离，然后得到对应的脉冲数传入中断函数开启 PWM 控制电机运动，一旦脉冲结束则再在中断中关闭 PWM，然后传入下一次运动的数据，直至运动结束。

具体流程图如图 11 所示。

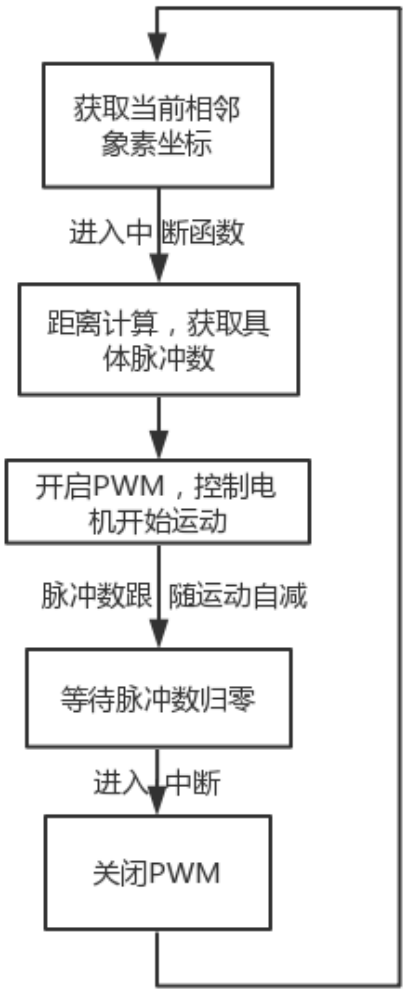


图 11 电机运动程序流程图

第三部分 完成情况及性能参数

3.1 具备两种输入图像方式

为了方便现场绘制任意图和接受客户的给定图，系统给出了两种绘图方式，即手绘和 AutoCAD 绘制。接收到绘制指令后，自动完成图像处理与坐标提取，所绘图形坐标范围最大可达到 150mm×150mm。

3.2 切割图形展示

机器操作台可以精准切割满足精度要求的实物，如圆形、正方形、

星形等。无明显失真，效果图如图 12 所示。

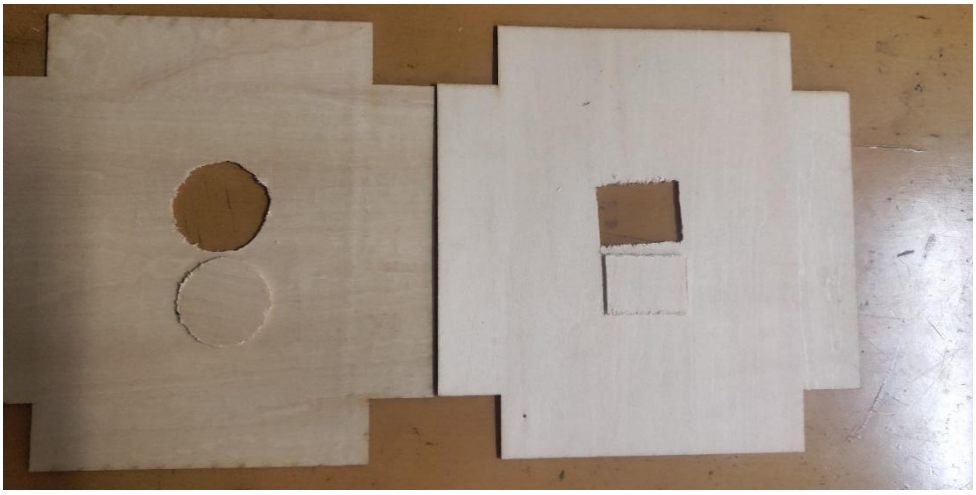


图 12 效果图（左：圆形，右：矩形）

3.3 切割误差

切割误差基本满足实际工程需要，给定图形和实际测量数据（3 次切割情况）对照表，见表 3。

表 3 给定图形和实际测量数据对照表

形状	数据指标	理论值	实际值(3 次切割情况)			平均误差
圆形	直径	40mm	41mm	42mm	39mm	3.3%
矩形	长	40mm	40mm	39mm	42mm	2.5%
矩形	宽	30mm	31mm	31mm	32mm	4.4%

3.4 切割速度

机器切割时速度大约为 0.24mm/s，切割材料每层上升速度大约为 0.2mm/层，可以通过软件调参后增加速度。

第四部分 总结

4.1 主要创新点

①通过机械切割方式代替了成本高昂的激光切割方式，成功将全自动切割机小型家庭化。

②AutoCAD 数据接口脚本，为画图软件拓展出了更多的可能性，一定程度上简化了开发者的开发周期。

③改良了 Bresenham 画图算法，使其不仅仅局限于计算机图形学的应用，可以扩展到更多的工业设备上去。

4.2 功能扩展

①目前机器只支持工作台的切割功能，但是由于我们自主设计的机械连接结构使得完成更多的功能成为了可能性，未来可以开发出更多的组件进行装配，使其成为完全的图像工作台。

②切割部分钻头与无刷电机的同轴度依旧不高，为切割带来了一定无法消除的误差，可以选用更加优异的切割方案。

③机器在运行时噪音过大，可以考添加吸音材料来减缓噪音。

④可以将摄像头与个人电脑与主控芯片的接口更换成串口转 WIFI 模块，全部实现无线通信，为用户提供便利。

⑤可运用“互联网+”模式，开发手机绘图终端软件，通过网络直接控制机器。

⑥同时增进云端功能，带有历史项目保存功能，方便开发者或用户对工作进度进行跟进。

4.3 心得体会

这一次智能互联的比赛，从选题，方案的确定，到最后完成产品，我们遇到了各种各样的问题，比如自主设计的机械结构精度不够高，图像数据处理过于复杂等问题。但是还是靠我们的不懈努力最终克服了这些困难。通过这次比赛，让我们的专业知识与素养有了一大步的提升，更是让我们认识到，“只要思想不滑坡，办法总比问题多”，一个方案不行就换下一方案，一个 BUG 调不通过就不停的 Debug 肯定可以发现问题出在哪。同时团队的合作也是至关重要的，毕竟这是一个团队比赛，不是一个人逞英雄的时候，有些时候与他人交流远比自己一个人单独思考要好得多。

总之，十分感谢这一次比赛给我们团队提供了这样一个提升和锻炼自己的机会，也非常感谢在比赛中帮助了我们的老师们。

第五部分 参考文献

[1] 邵贝贝.单片机嵌入式应用的在线开发方法[M].清华大学出版社, 2004.

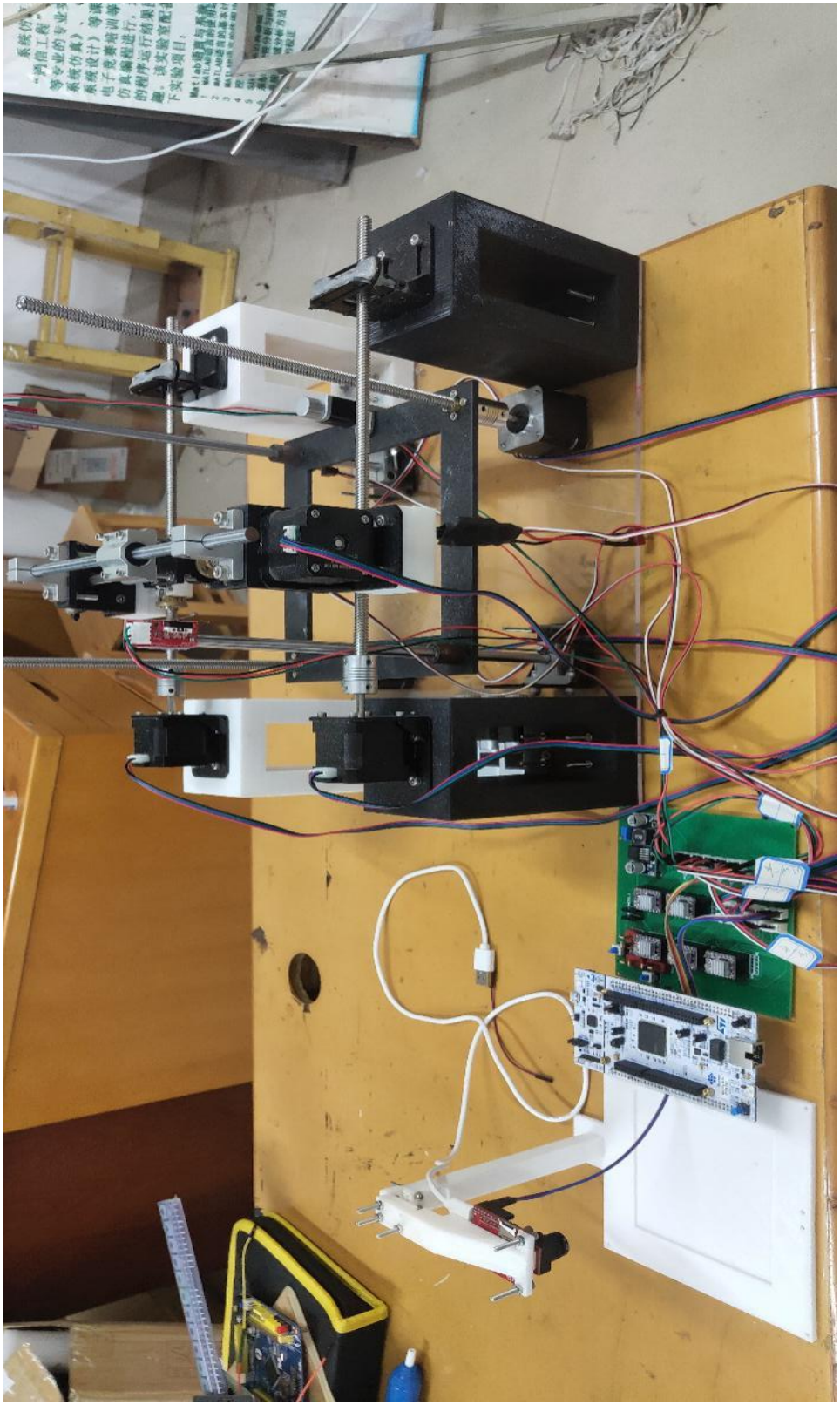
[2] 张波,耿在丹,苏国平.基于 C 语言的嵌入式系统编程[J]. 河南科技,2008,（2）： 30-31.

[3]. Python 3.7.4 文档” https://docs.python.org/zh-cn/3/“

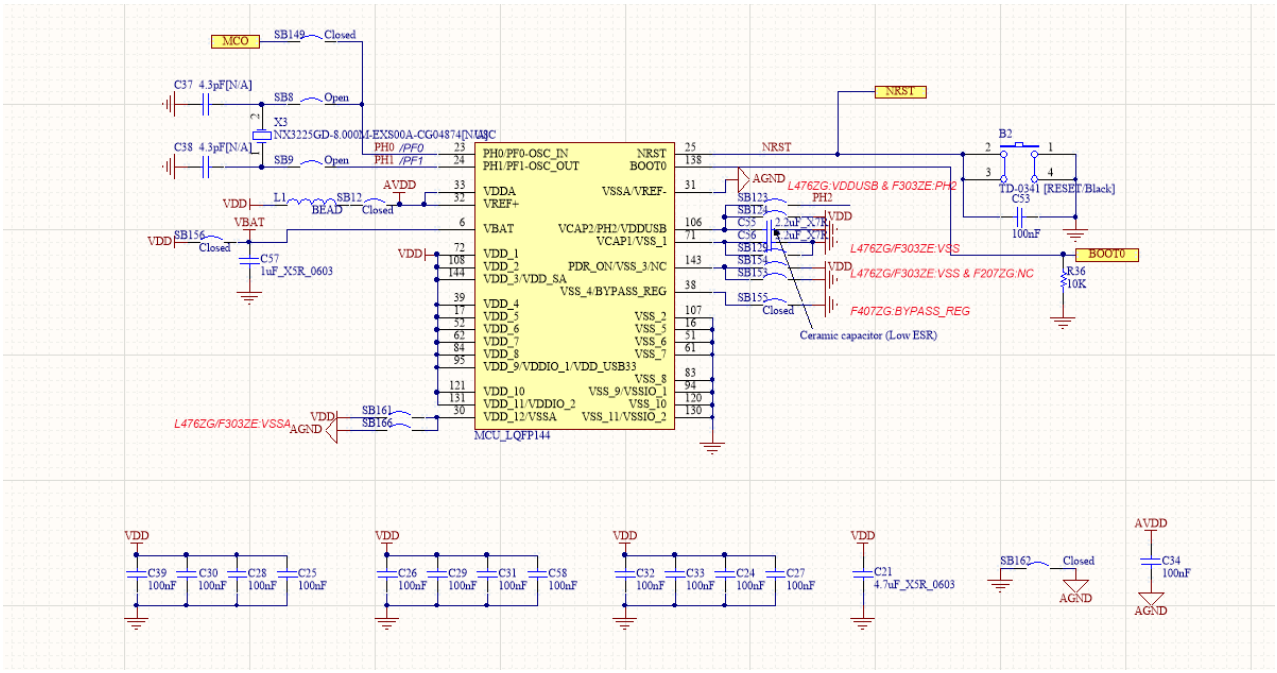
[4].Bresenham 直线算法 维基百科“ https://zh.wikipedia.org/zh-cn/%E5%B8%83%E9%9B%B7%E6%A3%AE%E6%BC%A2%E5%A7%86%E7%9B%B4%E7%B7%9A%E6%BC%94%E7%AE%97%E6%B3%95

第六部分 附录

附录 A 作品实物图片：



附录 B NUCLEO-F767ZI 开发板最小系统电路图



附录C Bresenham变化算法核心代码：

```
#include "data_analysis.h"
#include "Motor.h"
#include "usart.h"
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "delay.h"

int16_t cycle_coor_original[5000][2]; //初始坐标数据
int16_t cycle_coor_filter[5000][2];   //过滤后的第四象限坐标数据
int16_t cycle_coor[5000][2];           //重新拼接后的坐标数据，即最终
数据

/*坐标数据过滤，提取出单一象限坐标*/
static uint16_t Coor_Filter(int16_t X, int16_t Y, uint16_t length)
{
        uint16_t i = 0;
        uint16_t j = 0;
        uint16_t length_filter;
        for(i = 0;i < length;i++)
        {
if((cycle_coor_original[i][0] < X)&&(cycle_coor_original[i][1] >= Y))
                {
                  cycle_coor_filter[j][0] = cycle_coor_original[i][0];
                  cycle_coor_filter[j][1] = cycle_coor_original[i][1];
                   j++;
                }
        }
        length_filter = j;
        return length_filter;
}

/*冒泡排序，将象限点连续化*/
static void Cycle_BubbleSort(uint16_t len)
{
    int16_t temp;
    int i, j;
    for (i=0; i<len-1; i++)
        {
                        for (j=0; j<len-1-i; j++)
             {
            if (cycle_coor_filter[j][0] > cycle_coor_filter[j+1][0])
               {
                temp = cycle_coor_filter[j][0];
                cycle_coor_filter[j][0] = cycle_coor_filter[j+1][0];
                cycle_coor_filter[j+1][0] = temp;
                temp = cycle_coor_filter[j][1];
```

```
                cycle_coor_filter[j][1] = cycle_coor_filter[j+1][1];
                cycle_coor_filter[j+1][1] = temp;
            }
        }
        }

}

/*初始坐标数据保存*/
static void Cycle_Data_save(int32_t x,int32_t y,uint8_t flag)
{
        static uint16_t len = 0;
        if(flag)
        {
                len = 0;
                return;
        }
        cycle_coor_original[len][0] = x;
        cycle_coor_original[len][1] = y;
        len++;
}

/*Bresenham 算法,画圆函数*/
static uint16_t Circle_Bresenham(int16_t X,int16_t Y,uint16_t R)
{
    int8_t  a;
         int8_t  b;
    int32_t c;
         uint16_t length = 0;
    a=0;
    b=R;
    c=3-2*R;
    while (a<b)
      {
        Cycle_Data_save(X+a,Y+b,0);     //        7
        Cycle_Data_save(X-a,Y+b,0);     //        6
        Cycle_Data_save(X+a,Y-b,0);     //        2
        Cycle_Data_save(X-a,Y-b,0);     //        3
        Cycle_Data_save(X+b,Y+a,0);     //        8
        Cycle_Data_save(X-b,Y+a,0);     //        5
        Cycle_Data_save(X+b,Y-a,0);     //        1
        Cycle_Data_save(X-b,Y-a,0);     //        4

        length = length+8;
        if (c<0) c=c+4*a+6;
        else
        {
            c=c+4*(a-b)+10;
            b-=1;
```

```
        }
       a = a + 1;
    }

    if (a==b)
      {
       Cycle_Data_save(X+a,Y+b,0);
       Cycle_Data_save(X+a,Y+b,0);
       Cycle_Data_save(X+a,Y-b,0);
       Cycle_Data_save(X-a,Y-b,0);
       Cycle_Data_save(X+b,Y+a,0);
       Cycle_Data_save(X-b,Y+a,0);
       Cycle_Data_save(X+b,Y-a,0);
       Cycle_Data_save(X-b,Y-a,0);
                              length = length+8;

     }

         Cycle_Data_save(0,0,1);
         return length;
}

static uint16_t Cycle_Coor_Connect(int16_t X, int16_t Y, int16_t
length)
{
        uint16_t i = 0;
        uint16_t len;
        for(i=i; i < length; i++)
        {
                      cycle_coor[i][0] = cycle_coor_filter[i][0];
                      cycle_coor[i][1] = cycle_coor_filter[i][1];
        }

                for(i=i; i < 2*length; i++)
        {
         cycle_coor[i][0] = X + (cycle_coor_filter[i-length][1] - Y);
         cycle_coor[i][1] = Y + (X - cycle_coor_filter[i-length][0]);
        }

                for(i=i; i < 3*length; i++)
        {
       cycle_coor[i][0] = X + (X - cycle_coor_filter[i-2*length][0]);
       cycle_coor[i][1] = Y - (cycle_coor_filter[i-2*length][1] - Y);
        }

                for(i=i; i < 4*length; i++)
        {
      cycle_coor[i][0] = X - (cycle_coor_filter[i-3*length][1] - Y);
      cycle_coor[i][1] = Y - (X - cycle_coor_filter[i-3*length][0]);
```

```
        }

        len = 4*length;
        return len;
}

void Data_Cycle_Reset(uint16_t len_original,uint16_t
len_filter,uint16_t len)
{
        uint16_t i;
        for(i = 0; i < len_original; i++)
        {
                cycle_coor_original[i][0] = 0;
                cycle_coor_original[i][1] = 0;
        }

                for(i = 0; i < len_filter; i++)
        {
                cycle_coor_filter[i][0] = 0;
                cycle_coor_filter[i][1] = 0;
        }

                for(i = 0; i < len; i++)
        {
                cycle_coor[i][0] = 0;
                cycle_coor[i][1] = 0;
        }

}

void Cycle_Analysis(int16_t X,int16_t Y,uint16_t R)
{
        uint16_t length_original;//原始坐标数据长度
        uint16_t length_filter;//过滤后坐标数据的长度
        uint16_t length;//最终坐标数据的长度
        uint16_t i = 0;

        length_original = Circle_Bresenham(X, Y, R);//画圆算法
        length_filter = Coor_Filter(X, Y, length_original);//数据过滤
        Cycle_BubbleSort(length_filter);//冒泡排序
        length = Cycle_Coor_Connect(X , Y, length_filter);//数据重拼
接

        /*此处调用圆切割函数*/
        for(i = 0; i < length-1; i++)
        {
                while(flag_x|flag_y)
                {
                }
```

```
Motor_Action(cycle_coor[i][0],cycle_coor[i+1][0],cycle_coor[i][1],cyc
le_coor[i+1][1]);
                delay_ms(500);
        }

        Data_Cycle_Reset(length_original, length_filter, length);
}


int16_t line_coor[5000][2];

/*坐标数据保存函数*/
static void Line_Data_save(int16_t x,int16_t y,uint8_t flag)
{
                static uint16_t len = 0;
                if(flag)
        {
                len = 0;
                return;
        }

        line_coor[len][0] = x;
        line_coor[len][1] = y;
        len++;
}

static uint16_t Line_Bresenham(uint16_t x0, uint16_t y0, uint16_t x1,
uint16_t y1)
{
  int16_t dx, dy;
  int16_t dx2, dy2;
  int16_t x_inc, y_inc;
  int16_t error, index;
  uint16_t length = 0;
        dx = x1-x0;
        dy = y1-y0;

        if (dx>=0)
        {
                x_inc = 1;
        }
        else
        {
                x_inc = -1;
                dx    = -dx;
        }

        if (dy>=0)
        {
```

```
        y_inc = 1;
}
else
{
        y_inc = -1;
        dy    = -dy;
}

dx2 = dx << 1;
dy2 = dy << 1;

if (dx > dy)
{
        error = dy2 - dx;
        for (index=0; index <= dx; index++)
        {
                Line_Data_save(x0,y0,0);
                length++;
                if (error >= 0)
                {
                        error-=dx2;
                        y0+=y_inc;
                }
                error = error + dy2;
                x0 = x0 + x_inc;
        }
}
else
{
        error = dx2 - dy;

        for (index=0; index <= dy; index++)
        {

                Line_Data_save(x0,y0,0);
                length++;
                if (error >= 0)
                {
                        error = error - dy2;
                        x0 =x0 + x_inc;
                }
                error =error + dx2;

                y0 =y0 + y_inc;
        }
}
Line_Data_save(0,0,1);

return length;
```

```
}


void Data_Line_Reset(int16_t len)
{
        uint16_t i;
        for(i = 0; i < len; i++)
        {
                line_coor[i][0] = 0;
                line_coor[i][1] = 0;
        }
}

//int16_t line_coor[5000][2];

void Line_Analysis(int16_t x0, int16_t y0, int16_t x1, int16_t y1)
{
        uint16_t length;
        uint16_t i;
        length = Line_Bresenham(x0, y0 ,x1, y1);
        /*此处调用线条切割函数*/
        for(i = 0; i < length-1; i++)
        {
                while(flag_x|flag_y)
                {
                }

        Motor_Action(line_coor[i][0],line_coor[i+1][0],line_coor[i][1
],line_coor[i+1][1]);

        }

        Data_Line_Reset(length);
}
```

① 电机驱动转接板 PCB 图

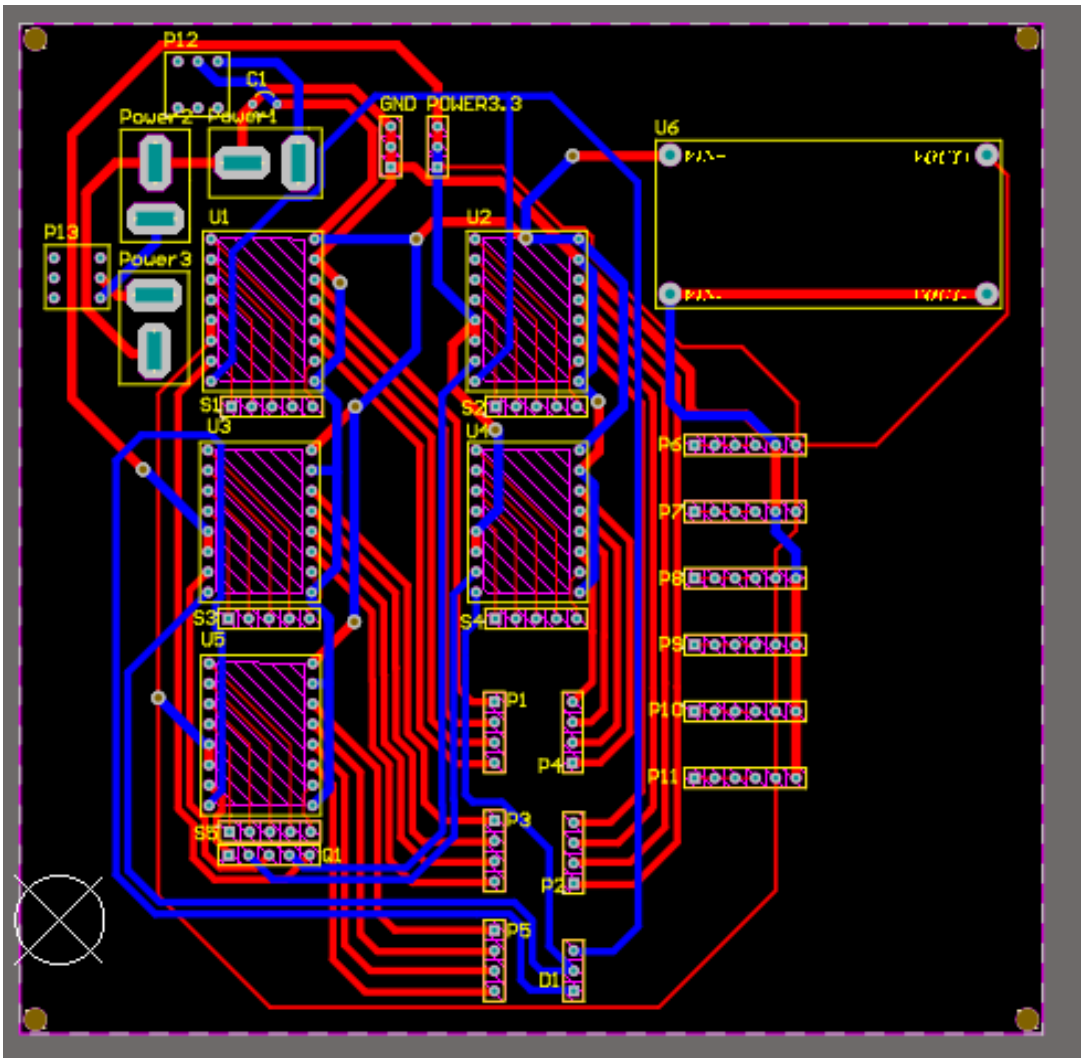


图 15：电机驱动转接板 PCB