

## 综合设计与实践 A 综合实验设计报告

### 摘要:

本文介绍了综合设计与实践 A 课程的综合实验部分的设计过程, 开发基于 STM32G0 单片机及 16 个 ITR20001/T24 红外对射传感器的手势识别器。本课程设计涵盖电路原理图设计、PCB 版图绘制、K-Means 算法聚类、嵌入式语言开发等内容, 实现手势识别器能够感知靠近 PCB 放置的“石头”、“剪刀”、“布”手势, 并通过 LED 亮灯的形式显示判断的结果。

### Abstract:

This article introduces the design process of the comprehensive experiment part of the Comprehensive Design and Practice A course, and develops a gesture recognizer based on the STM32G0 microcontroller and 16 ITR20001/T24 infrared sensors. This course design covers circuit schematic design, PCB layout drawing, K-Means algorithm clustering, embedded language development, etc., so that the gesture recognizer can sense the "rock", "scissors", and "paper" gestures placed close to the PCB, and display the judgment results in the form of LED lights.

## 一. 题意分析

本课程设计需要开发基于 STM32G0 单片机及 16 个 ITR20001/T24 红外对射传感器的手势识别器, 在 15cm×15cm 的 PCB 上完成对于石头、剪刀、布三个手势的识别, 并将识别的结果以 LED 亮灯的形式显示。此外, 电路需设计在一块 PCB 上, 不得使用插接模块。

## 二. 原理分析与方案选择

### 1、 原理分析

要实现该手势识别器的预期功能, 首先需要运用电路理论的知识根据功能要求设计电路, 用 EDA 软件画出电路原理图。电路原理图是指说明电路中各个元器件的电气连接关系的图纸, 其不涉及元器件的具体大小、形状, 而

只是关心元器件的类型、相互之间的连接情况。绘制该部分时需要注意选择的封装，须在设计电路前确定。

其次，需要将电路原理图利用软件工具导入 PCB 版图，并将电子元件合理布局连线，得到完整的 PCB 版图。该部分需要符合绘制规范，比如直角容易留腐蚀液，因此布线时要走钝角；布线线宽要区分为接地线宽于电源线宽于信号线，布线间距要控制，布线长度要尽可能短。16 个 ITR20001/T24 红外对射传感器均匀分布在 PCB 上并以  $4 \times 4$  形式排列，使之能够最大程度覆盖手势感知识别区域；将三个显示结果的 LED 等尽可能放置在板子靠近边缘一侧，便于实验观察。

随后，将 PCB 版图发送至厂家加工制作成印刷电路板，其是用于安装、固定各个实际元器件，并利用铜箔走线实现其正确连接关系的一块基板。并在拿到板子后将电子元件安装焊接在印刷电路板上。

之后在焊接完成的板子上进行嵌入式开发。采集三种手势以及没有手势情况下的数据构成数据集，应用于相关算法，并完成推理计算，最终在 LED 灯上显示判别手势的结果。

## 2、 不同方案的比较

在实现手势识别的过程中，我们可以考虑以下几种方案来实现主要功能。

表 1 方案对比

方案	PC 端训练与推理	PC 端训练、单片机上推理	单片机端训练与推理
优点	基于 PC 的处理能力较强、开发调试便捷且易于修改优化	实时性较好、资源利用优化、系统更独立稳定	系统的集成度高且实时性与便携性最强
缺点	实时性不足且对硬件的依赖性较高，增加了系统的复杂度	训练与推理分离会增加调试过程的复杂度，需要分别在 PC 和单片机上验证	受限于处理能力与资源，开发调试也不方便

经过比较，故选择实现 PC 端训练与推理、单片机显示与 PC 端训练、单片机推理与显示的方案。

为实现对于手势的识别，需要选择使用的算法。K-Means 原理较为简单，

实现容易，收敛速度快；聚类效果较优；可解释度较强；主要需要调整的参数仅仅是簇数  $k$ 。也具有如下缺陷： $k$  值选取不好把握；对于不是凸的数据集比较难收敛；对噪音和异常点较为敏感。其他更为复杂的机器学习算法虽然可能达到更高的准确度，但由于计算量大、所需资源多，不适合在单片机上运行，对于本次课设的具体要求并不合适，故选择采用 K-Means 算法进行聚类。

### 三. 电路实现

使用 KiCad 软件绘制电路原理图，其中，图 1 主要为电源及 USB 接口部分。采用了 AMS1117 3.3 稳压器，将输入电压 5V 转化为稳定的输出电压 3.3V，满足各元件的供电需求。同时在稳压器输入端增加了 10uF 电容滤除低频噪声，在输出端增加了 0.1uF 电容抑制高频噪声，构成较为可靠稳定的电源滤波网络，减少波动与误差，确保电压的稳定性。

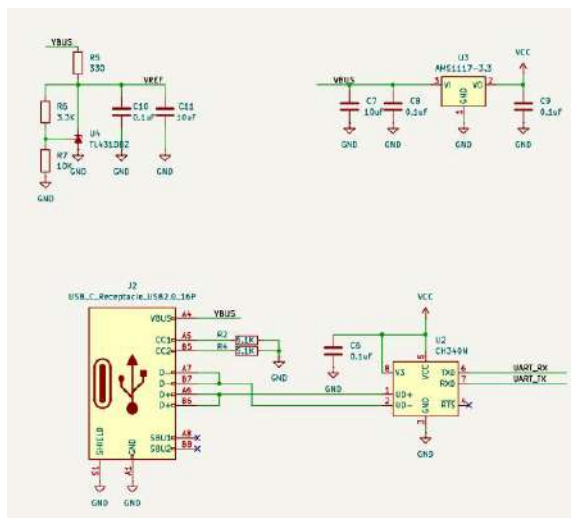


图 1 电源及 USB 接口部分电路原理图

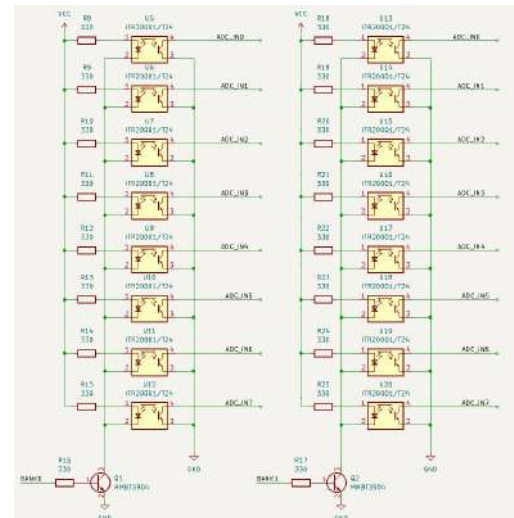


图 2 传感器阵列部分

图 2 所示部分为重要的传感器阵列部分，包含 16 个 ITR20001/T24 红外对射传感器。红外对管由红外发射管和接收管组成，其原理是由发射管发射光源，接收管接收光源并转换成电信号输出。当感应范围内有遮挡时，接收管能够接收到被反射的光线，输出信号变小；而当遮挡远离后，接收管未接收到反射光线，输出信号基本不变且较大。

传感器输出的是模拟信号，需要用模数转换器 ADC 将这些连续变化的信号转换为可以由微控制器处理的离散数字信号，将 ADC 端口与 stm32G0 的 PA0 至 PA7

端口相对应连接以采集接收到的红外信号强度。由于 ADC 采样分辨率为 12 位，将 ADC 采样数值转电压公式设置为

$$\text{Volt}[i]=((\text{float})\text{Data}[i])*3.3/4090 \quad \text{公式 1}$$

可将红外传感器所采集到的数据转化为 0 到 3.3V 之间。由于 ADC 通道不够，采用分时切换采集的方法解决。

将电路原理图转换为 PCB 版图，如下图所示，其尺寸为 15×15cm，红外对射传感器采用 4×4 均匀排列，确保覆盖预期的感知区域。

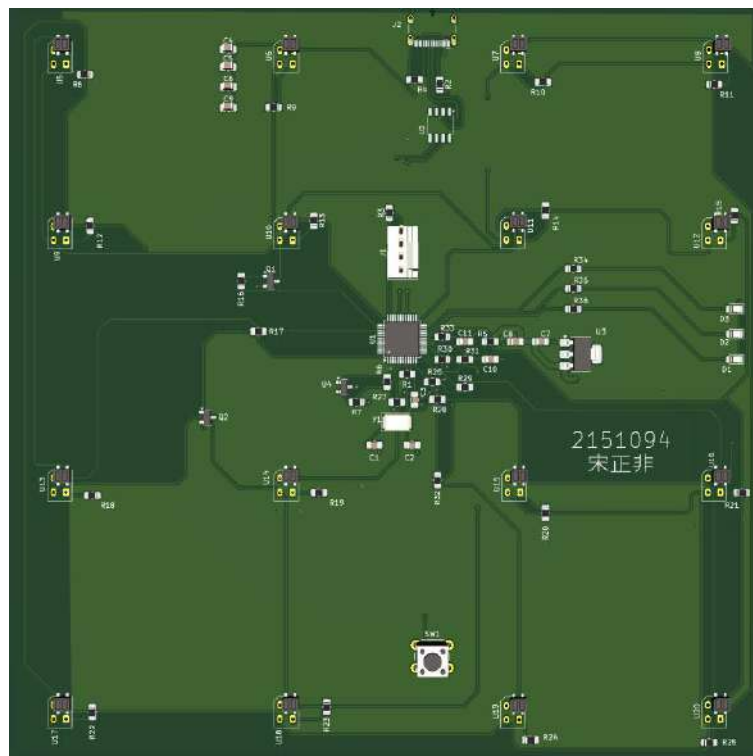


图 3 PCB 3D 仿真图

## 四. 软件算法

### 1、 聚类算法

K-Means 算法是无监督的聚类算法，其核心思想较为简单，对于给定的样本集，按照样本之间的距离大小，将样本划分为 k 个簇，将相似的点尽可能聚集在一个簇中，使用两点在欧式空间中的距离定义相似，

$$\text{EucDist}(x, y) = \sqrt{(x - y)^T (x - y)} = \sqrt{\sum_{i=1}^D (x^{(i)} - y^{(i)})^2} \quad \text{公式 2}$$

使得簇内的点尽量密集的连在一起，而簇间的距离尽量大。如下图所示为

K-Means 算法的流程图。

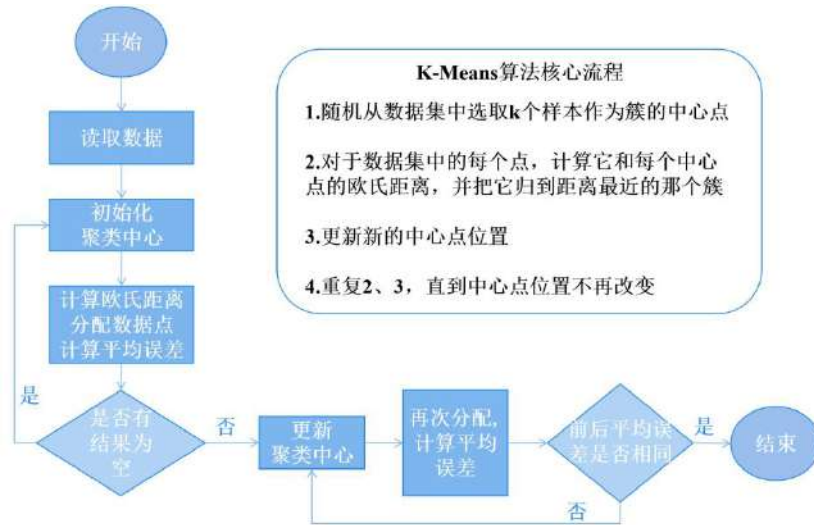


图 4 K-Means 算法流程图

在使用 K-Means 算法前，先使用 PCB 采集数据。通过串口调试助手获得不同手势或无遮挡时的红外对射传感器的值，汇总构成手势识别数据集，其具有 50 个无遮挡情况、50 个手势为“剪刀”、50 个手势为“石头”、50 个手势为“布”的数据。

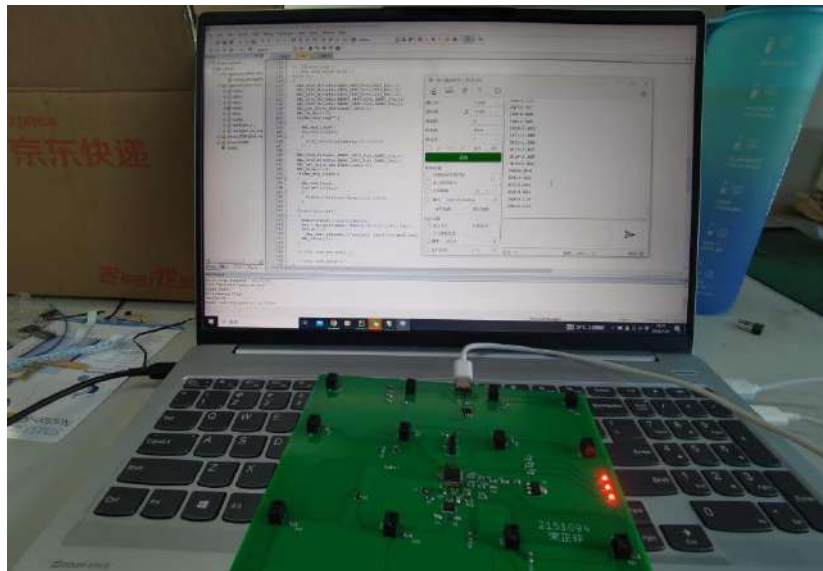
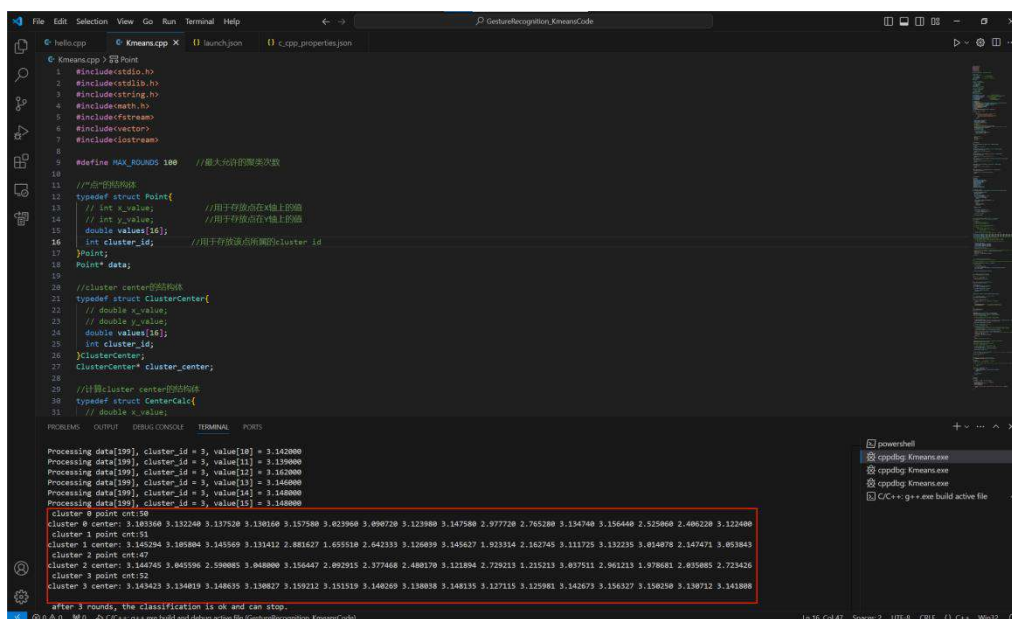


图 5 采集数据

对于本课设的具体情况，将簇数值设定为 4，并可规定四个簇的初始中心点为对应数据集中的点，加快算法收敛速度。最终通过 K-Means（详见附件 GestureRecognition\_KmeansCode 文件夹）得到如下图所示的聚类结果，用于后续单片机软件算法中，其中 cluster0 对应无遮挡情况，cluster1 对应“剪刀”，cluster2 对应“石头”，cluster3 对应“布”。



The image shows a C++ IDE with a file named `Kmeans.cpp`. The code defines a `Point` struct with `x_value`, `y_value`, and `values[16]` (for cluster ID). It also defines a `ClusterCenter` struct with `x_value` and `values[16]`. The main function processes data points and calculates cluster centers. The terminal output shows the results of the K-Means algorithm after 3 rounds, indicating that the classification is OK and can stop. The output lists the cluster centers and the data points assigned to each cluster.

```
Processing data[199], cluster_id = 3, value[16] = 3.142000
Processing data[199], cluster_id = 3, value[16] = 3.139000
Processing data[199], cluster_id = 3, value[16] = 3.142000
Processing data[199], cluster_id = 3, value[16] = 3.140000
Processing data[199], cluster_id = 3, value[16] = 3.140000
Processing data[199], cluster_id = 3, value[16] = 3.140000
cluster 0 point cnt:19
cluster 0 center: 3.183360 3.132240 3.137520 3.138160 3.157580 3.023960 3.090720 3.123800 3.147580 2.977720 2.765280 3.156440 2.525800 2.486220 3.122400
cluster 1 point cnt:51
cluster 1 center: 3.142294 3.185884 3.145569 3.131412 2.881627 1.655158 2.642333 3.126839 3.145627 1.923314 2.162745 3.111725 3.132235 3.014078 2.147471 3.053843
cluster 2 point cnt:47
cluster 2 center: 3.144745 3.045590 2.590885 3.040000 3.156447 2.092915 2.377466 2.480170 3.128894 2.729213 1.25213 3.037911 2.962213 1.970681 2.035085 2.723426
cluster 3 point cnt:52
cluster 3 center: 3.143423 3.134819 3.148635 3.138827 3.159212 3.151519 3.140289 3.138038 3.148135 3.127115 3.125981 3.142673 3.156327 3.150258 3.138712 3.141888
after 3 rounds, the classification is ok and can stop.
```

图 6 K-Means 算法聚类结果

## 2、 单片机软件算法

首先，实现在 PC 上利用 K-Means 算法实现手势的推理判别（详见附件 gesture 文件夹），即训练、推理流程在 PC 上实现，将结果推送至单片机并显示。在上述 PC 端上使用采集的数据集进行训练，随后通过串口调试助手获取某时刻的红外对射传感器信息，并输入到 PC 端进行推理，得到所属的簇，即得到对应的手势结果，随后将结果推送至在单片机，在对应算法中修改 LED 灯的亮灭情况，显示结果。如下图代码所示，若为无遮挡情况，则三个 LED 全亮。

```
HAL_GPIO_WritePin(LED0_GPIO_Port,LED0_Pin,1);
HAL_GPIO_WritePin(LED1_GPIO_Port,LED1_Pin,1);
HAL_GPIO_WritePin(LED2_GPIO_Port,LED2_Pin,1);
```

图 7 算法 1 显示部分代码

随后，实现在单片机上利用 K-Means 算法实现手势的推理判别（详见附件 gesture (2) 文件夹），即训练过程在 PC 上实现，推理过程在单片机上实现。由于 K-Means 算法较为简单，计算也较为迅速，可以直接写入单片机软件算法内，简化效果实现流程的复杂度。在 PC 端上使用采集的数据集进行训练后，推理时，PCB 板子实时获取红外对射传感器的信息，并在板子上即可完成聚类算法的推理，得到该时刻属于的簇，即判断手势的结果，并点亮对应的灯完成显示。



```
//Cluster0 Scissors
if(new_label == 0)
{
    HAL_GPIO_WritePin(LED0_GPIO_Port,LED0_Pin,1);
}
//Cluster1 Rock
if(new_label == 1)
{
    HAL_GPIO_WritePin(LED1_GPIO_Port,LED1_Pin,1);
}
//Cluster2 Paper
if(new_label == 2)
{
    HAL_GPIO_WritePin(LED2_GPIO_Port,LED2_Pin,1);
}
//Cluster3 Nothing
if(new_label == 3)
{
    HAL_GPIO_WritePin(LED0_GPIO_Port,LED0_Pin,1);
    HAL_GPIO_WritePin(LED1_GPIO_Port,LED1_Pin,1);
    HAL_GPIO_WritePin(LED2_GPIO_Port,LED2_Pin,1);
}
HAL_Delay(500);
```

图 8 算法 2 显示部分代码

## 五. 测试方法、结果与分析

使用 stm32G0 将单片机程序烧录至大板子，随后针对不同手势进行测试。测试时手需要尽量靠近一点板子，不能超出感知距离；同时避免光线干扰，在不发出红外光或较暗的照明条件下进行。



图 9 烧录程序

如下图所示，当 PCB 附近无手势无遮挡时，D1、D2、D3 三个 LED 灯全亮；而当手势为“剪刀”时，对应的 D1 亮；当手势为“石头”时，对应的 D2 亮；当手势为“布”时，对应的 D3 亮。部分测试过程可见附件视频，测试虽存在一定延迟或误差，但处于正常范围之内。

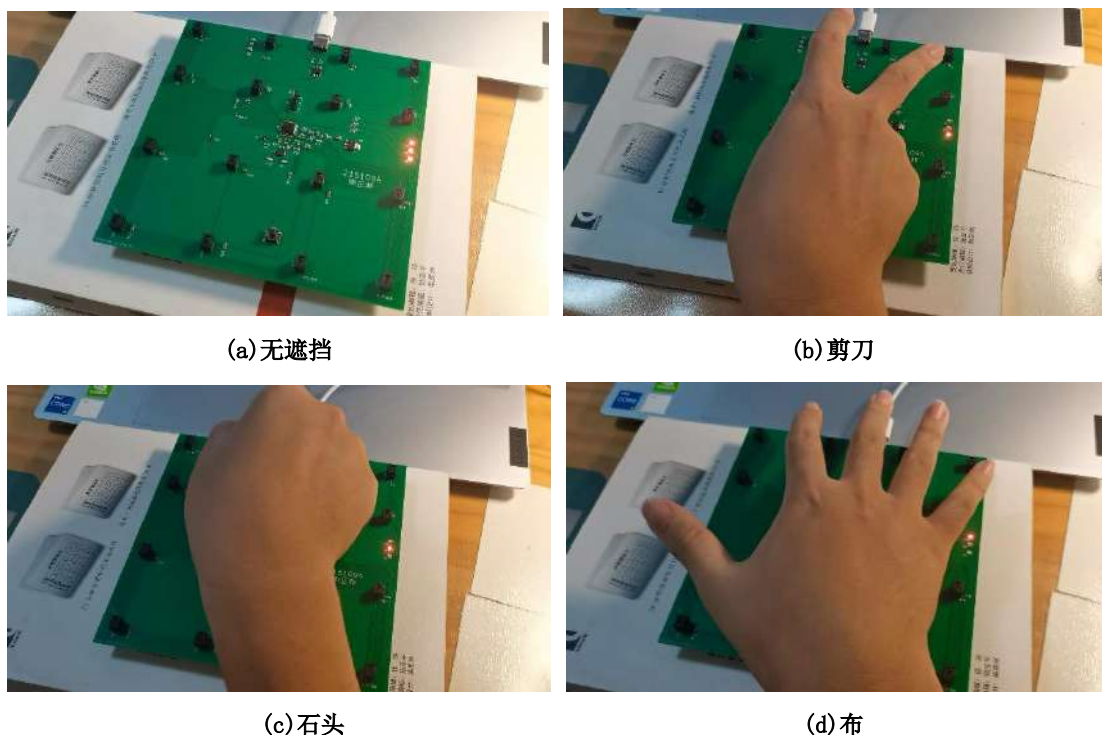


图 10 测试过程图片

测试结果统计如下，每一大组均在无遮挡、剪刀、石头与布四种情况下各测量 20 次。总而言之，手势识别器识别结果较为准确，成功实现了预期功能。

表 2 测试结果统计

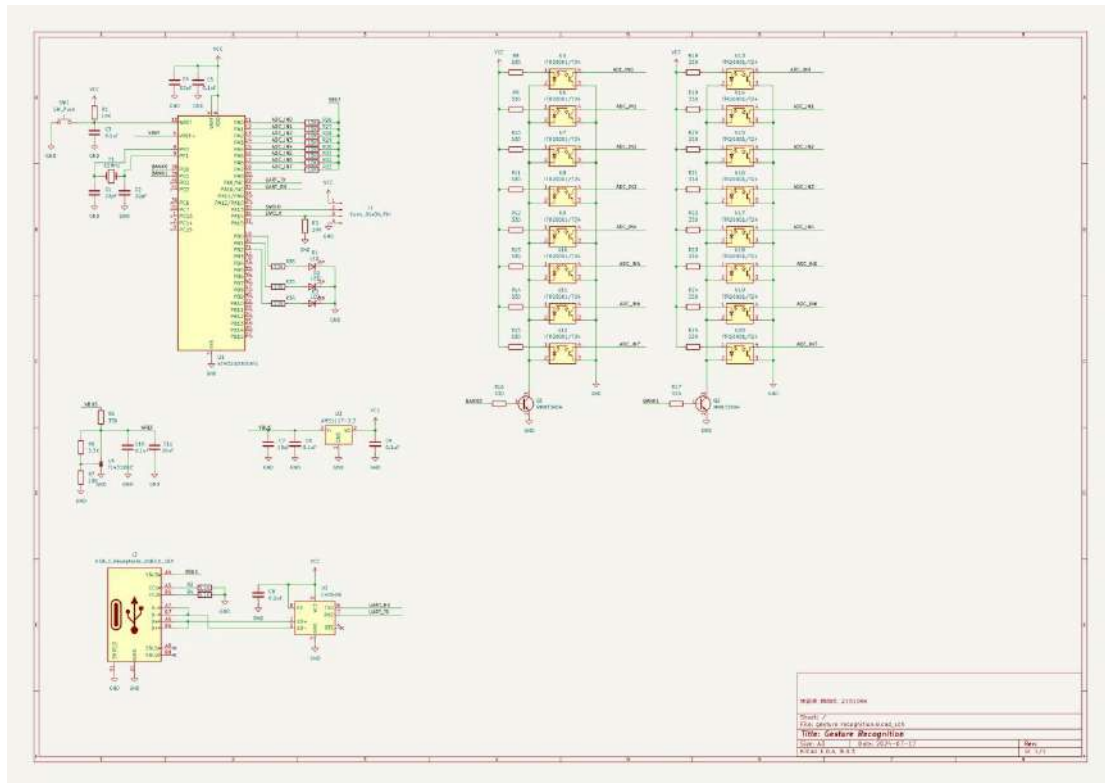
样本组	第一组	第二组	第三组	平均值
准确率	77.5%	73.75%	78.75%	76.67%

## 六. 总结

本次综合实验给了我一个全面深入了解并设计电子电路系统以实现目标功能的机会。内容涉及了硬件部分与软件部分，能够让我对一些专业知识有了一定的巩固与应用方面的实践，加深了我对自动化专业发展方向的认知。同时，在完成的过程中遇到了一些问题，比如焊芯片的时排焊后没有把锡吸干净导致相邻管脚短接、程序无法烧录到大板子上，采集数据时发现 IN0 和 IN8 的数据都很异常地接近 0 然后排查到是由于连接的电阻一端虚焊了导致接触不良断路了；从发现问题到寻找解决方法到最终解决问题，我收获了很多心得与感悟，也不断加深培养了排除故障、解决问题的工程思维。最终，我很高兴能够解决遇到的一系列问题，成功实现了基于 stm32G0 与红外对射传感器的手势识别器开发，完成了申优作业的挑战，非常感谢蒋老师的指导与帮助！

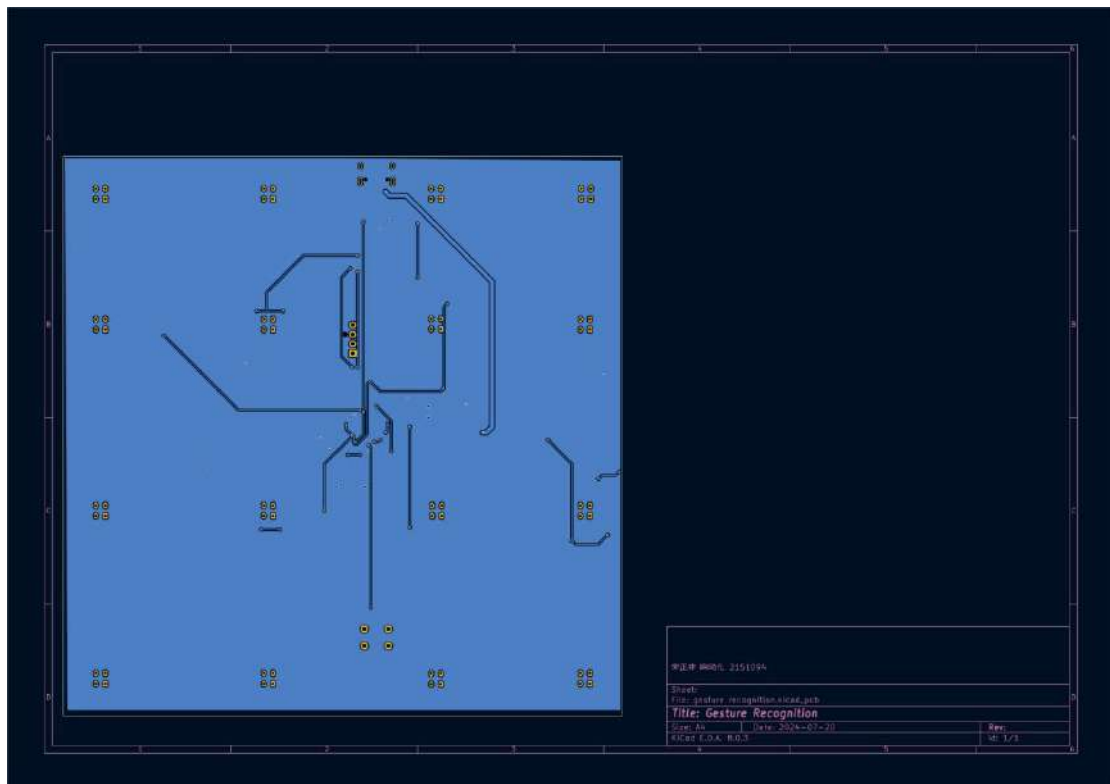


## 附录 A：电路原理图



## 附录 B：PCB 版图





## 附录 C：PC 端 Kmeans 算法主要核心函数

```
int main(int argc, char* argv[])
{
    if( argc != 4 )
    {
        printf("This application needs 3 parameters to run:"
            "\n the 1st is the size of data set,"
            "\n the 2nd is the file name that contains data"
            "\n the 3rd indicates the cluster_count"
            "\n");
        exit(1);
    }
    data_size_total = atoi(argv[1]);
    strcat(filename, argv[2]);
    cluster_count = atoi(argv[3]);
    std::cout<< "start memory alloc";
    //1, memory alloc
    memoryAlloc();
    //2, read point data from file
    std::cout<< "start read data from file"<<std::endl;
    std::vector<std::vector<double>> filedata; // data[i][j] means ith data, jth index
    readDataFromFile(filename, filedata);
    //3, initial cluster
```

```
initialCluster(filedata);  
for (int i=0; i < 200;i++) {  
    for (int j=0; j< 16; j++){  
        data[i].values[j] = filedata[i][j];  
        data[i].cluster_id = i/50;  
    }  
}  
  
//4, run k-means  
kmeans();  
  
//5, memory free & end  
memoryFree();  
  
return 0;  
}
```

```
//K-means 算法  
void kmeans()  
{  
    int rounds;  
    for( rounds = 0; rounds < MAX_ROUNDS; rounds++ )  
    {  
        printf("\nRounds : %d          \n", rounds+1);  
        partition4AllPointOneCluster();  
        calcClusterCenter();  
        if( 0 == is_continue )  
        {  
            printf("\n after %d rounds, the classification is ok and can stop.\n", rounds+1);  
            break;  
        }  
    }  
}
```

```
//在一轮的聚类中得到所有的 point 所属的 cluster center  
void partition4AllPointOneCluster()  
{  
    int i;  
    for( i = 0; i < data_size_total; i++ )  
    {  
        if( data[i].cluster_id != -1 ) //这个点就是 center, 不需要计算  
            continue;  
        else  
        {  
            calcDistance2AllCenters(i); //计算第 i 个点 to 所有 center 的 distance  
        }  
    }  
}
```

```
    partition4OnePoint(i);          //根据 distance 对第 i 个点进行 partition
}
}
}
```

```
//确定一个点属于哪一个 cluster center(取距离最小的)
void partition4OnePoint(int point_id)
{
    int i;
    int min_index = 0;
    double min_value = distance_from_center[0];
    for( i = 0; i < cluster_count; i++ )
    {
        if( distance_from_center[i] < min_value )
        {
            min_value = distance_from_center[i];
            min_index = i;
        }
    }

    data[point_id].cluster_id = cluster_center[min_index].cluster_id;
}
```

```
//重新计算新的 cluster center
void calcClusterCenter()
{
    int i, j;
    memset(center_calc, 0, sizeof(CenterCalc) * cluster_count);
    memset(data_size_per_cluster, 0, sizeof(int) * cluster_count);

    //分别对每个 cluster 内的每个点的 X 和 Y 求和, 并计每个 cluster 内点的个数
    for( i = 0; i < data_size_total; i++ )
    {
        for( j = 0; j < 16; j++ ) {
            printf("Processing data[%d], cluster_id = %d, value[%d] = %f\n", i, data[i].cluster_id, j,
data[i].values[j]);
            center_calc[data[i].cluster_id].values[j] += data[i].values[j];
        }
        data_size_per_cluster[data[i].cluster_id]++;
    }
}
```

```
//计算每个 cluster 内点的 X 和 Y 的均值作为 center
```

```
for( i = 0; i < cluster_count; i++ )
{
    if(data_size_per_cluster[i] != 0) {
        for (j = 0; j < 16; j++) {
            center_calc[i].values[j] /= (double)(data_size_per_cluster[i]);
        }
        printf(" cluster %d point cnt:%d\n", i, data_size_per_cluster[i]);
        printf("cluster %d center: ", i);
        for (j = 0; j < 16; j++) {
            printf("%f ", center_calc[i].values[j]);
        }
        printf("\n");
    }
    else
        printf(" cluster %d count is zero\n", i);
}
```

```
//计算一个点到一个 cluster center 的 distance
void calcDistance2OneCenter(int point_id, int center_id)
{
    double sum = 0.0;
    // 计算十六维向量的欧几里得距离
    for (int i = 0; i < 16; i++) {
        double diff = data[point_id].values[i] - cluster_center[center_id].values[i];
        sum += diff * diff;
    }
    distance_from_center[center_id] = sqrt(sum);
}
```

```
//计算一个点到每个 cluster center 的 distance
void calcDistance2AllCenters(int point_id)
{
    int i;
    for( i = 0; i < cluster_count; i++ )
    {
        calcDistance2OneCenter(point_id, i);
    }
}
```



## 附录 D：单片机软件部分关键性算法

```
int main(void)
{

    /* USER CODE BEGIN 1 */
    char txbuf[32];
    int len;
    int i;
    /* USER CODE END 1 */

    /* MCU Configuration-----*/

    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();

    /* USER CODE BEGIN Init */

    /* USER CODE END Init */

    /* Configure the system clock */
    SystemClock_Config();

    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_DMA_Init();
    MX_ADC1_Init();
    MX_TIM3_Init();
    MX_USART1_UART_Init();
    /* USER CODE BEGIN 2 */
    HAL_ADCEx_Calibration_Start(&hadc1);
    HAL_TIM_Base_Start(&htim3);
    /* USER CODE END 2 */

    float centroids[4][16]={ { 3.103360, 3.132240, 3.137520, 3.130160, 3.157580, 3.023960,
3.090720, 3.123980, 3.147580, 2.977720, 2.765280, 3.134740, 3.156440, 2.525060, 2.406220,
3.122400},
    {3.145294, 3.105804, 3.145569, 3.131412, 2.881627, 1.655510, 2.642333, 3.126039,
3.145627, 1.923314, 2.162745, 3.111725, 3.132235, 3.014078, 2.147471, 3.053843},
```

```
{3.144745, 3.045596, 2.590085, 3.048000, 3.156447, 2.092915, 2.377468, 2.480170,
3.121894, 2.729213, 1.215213, 3.037511, 2.961213, 1.978681, 2.035085, 2.723426},
{3.143423, 3.134019, 3.148635, 3.130827, 3.159212, 3.151519, 3.140269, 3.138038,
3.148135, 3.127115, 3.125981, 3.142673, 3.156327, 3.150250, 3.130712, 3.141808}};
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    HAL_GPIO_WritePin(LED0_GPIO_Port,LED0_Pin,0);
    HAL_GPIO_WritePin(LED1_GPIO_Port,LED1_Pin,0);
    HAL_GPIO_WritePin(LED2_GPIO_Port,LED2_Pin,0);
    HAL_GPIO_WritePin(BANK0_GPIO_Port,BANK0_Pin,1);
    HAL_GPIO_WritePin(BANK1_GPIO_Port,BANK1_Pin,0);
    HAL_ADC_Start_DMA(&hadc1,Data,8);
    HAL_Delay(400);
    if(dma_end_flag==1)
    {
        dma_end_flag=0;
        for(i=0;i<8;i++)
        {
            Volt[i]=((float)Data[i])*3.3/4090;
        }
    }
    HAL_GPIO_WritePin(BANK0_GPIO_Port,BANK0_Pin,0);
    HAL_GPIO_WritePin(BANK1_GPIO_Port,BANK1_Pin,1);
    HAL_ADC_Start_DMA(&hadc1,Data,8);
    HAL_Delay(400);
    if(dma_end_flag==1)
    {
        dma_end_flag=0;
        for(i=0;i<8;i++)
        {
            Volt[i+8]=((float)Data[i])*3.3/4090;
        }
    }
    for(i=0;i<16;i++)
    {
        memset(txbuf,0,sizeof(txbuf));
        len = sprintf(txbuf,"IN%d=%1.3f\r\n",i,Volt[i]);
        if(len > 0)
            HAL_UART_Transmit_IT(&huart1,(uint8_t*)txbuf,len);
        HAL_Delay(200);
    }
}
```

```
int new_label = classify(Volt, centroids);
//Cluster0 Scissors
if(new_label == 0)
{
    HAL_GPIO_WritePin(LED0_GPIO_Port,LED0_Pin,1);
}
//Cluster1 Rock
if(new_label == 1)
{
    HAL_GPIO_WritePin(LED1_GPIO_Port,LED1_Pin,1);
}
//Cluster2 Paper
if(new_label == 2)
{
    HAL_GPIO_WritePin(LED2_GPIO_Port,LED2_Pin,1);
}
//Cluster3 Nothing
if(new_label == 3)
{
    HAL_GPIO_WritePin(LED0_GPIO_Port,LED0_Pin,1);
    HAL_GPIO_WritePin(LED1_GPIO_Port,LED1_Pin,1);
    HAL_GPIO_WritePin(LED2_GPIO_Port,LED2_Pin,1);
}
HAL_Delay(500);
/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
}
```