《现代电子系统设计》综合/创新实验说明书

郑智骋 2023年12月

目录

一、综合实验：PSoC6 中 LED 亮度调节的多重交互方式

1.1 实现目标

1.1.1 基本要求　本综合实验的基本功能要求是利用 PSoC6 的双核完成亮度控制与通信、显示等任务。具体而言，在 CM4 核上分配任务，用 CapSense 控制 LED 亮度，显示在 EINK 墨水屏上，同时将滑条位置利用 BLE 发送到手机，将亮度信息利用 IPC 通道发送给 CM0+核，CM0+核接收数据后利用 UART 串口发送给计算机。

1.1.2 附加目标　基本目标中已经有了处理器之间的通信、处理器与计算机之间的通信，以及 CapSense 和 PWM、EINK、手机等设备之间的通信，涵盖了蓝牙 BLE、串口 UART、IPC 通信、多任务间通讯等方面的通信模式。笔者结合自身的兴趣，又在基本要求的基础上加入了如下的目标：

① 基本任务中的蓝牙 BLE 只进行了手机从蓝牙读取数据的操作，我们还可以在 PSoC 内读取手机中的数据，譬如用户的操作信息。

② 在墨水屏上除了可以显示亮度信息以外，还可以使得亮度的信息更加直观、动感化。即在屏幕上打印出一条具有渐变效果的条带，用光标指向当前的亮度状态，并且随 CapSense 的变化随时同步变化。

③ 蓝牙服务的状态需要与用户之间进行交互，否则会非常不方便使用。笔者将蓝牙的广告状态指示为 LED 的闪烁，将蓝牙广告、连接、收发信息的状态实时打印显示在 EINK 屏幕上。

④ 两个触摸感应的按键可用于控制 LED 的亮灭；在关闭 LED 的同时，也可以同时关闭蓝牙服务，达到节能的目标。

⑤ 蓝牙除了发送当前滑块位置以外，还可以发送当前的具体亮度值。

笔者逐一实现了如上目标并通过了验收。将在后文介绍实现细节。

1.2 系统结构图 & 各部分作用说明

整体方案的硬件和固件的结构框图如下所示：

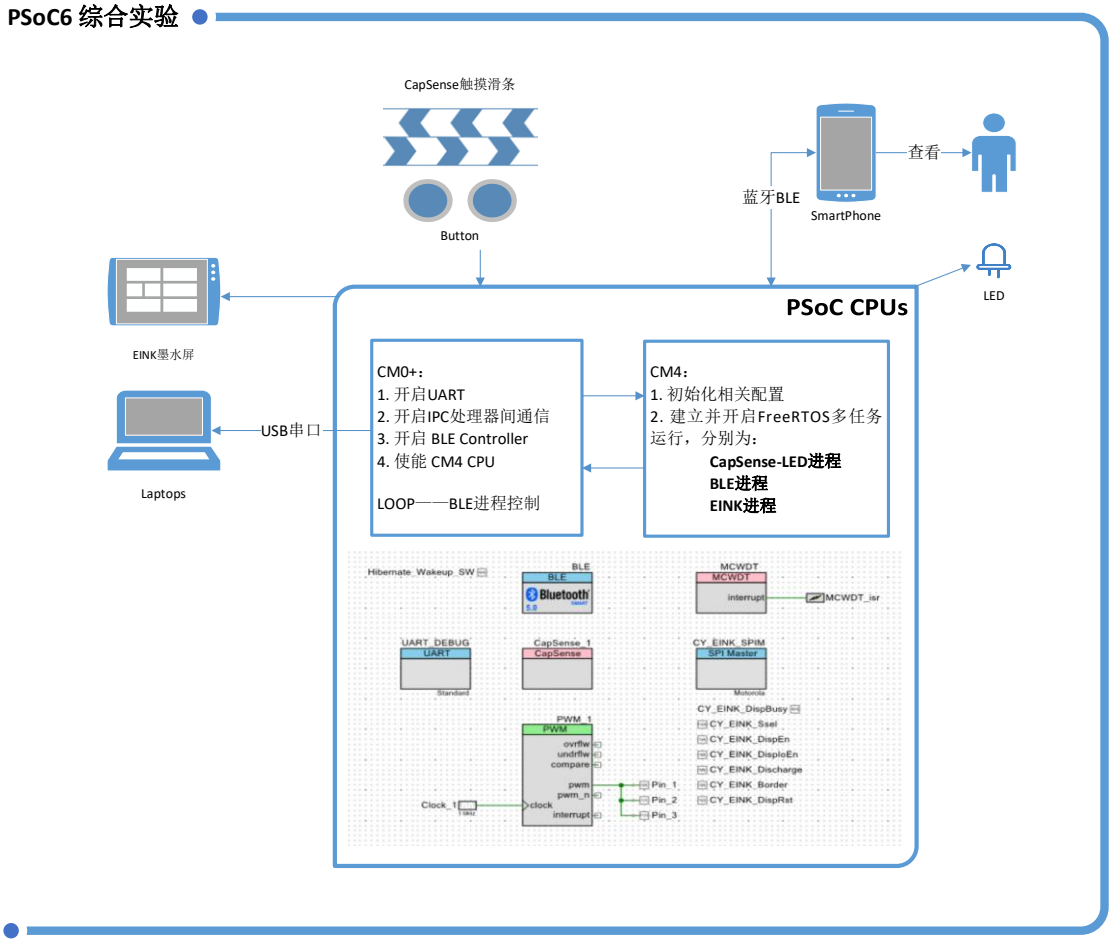


图 1 综合实验硬件和固件的系统框图

各硬件外设的作用分别为：CapSense 感应滑条用于控制 LED 亮度。Button 用于 LED 亮灭控制与 BLE 蓝牙服务的开闭控制（若关闭 LED，则同时关闭蓝牙服务模块，更加节能）。墨水屏用于实时显示亮度信息、CapSense 感应滑块的当前位置、实时显示蓝牙连接情况与用户查询情况。与手机和电脑连接用于亮度等信息的通讯。LED 灯的 RGB 三个引脚都接到 PWM 输出端，受其直接控制。

顶层电路图中各固件作用分别为：MCWDT 与 MCWDT_isr 用于产生频率为 4Hz 的中断信号，中断程序控制 PWM 的占空比，产生 LED 灯以 2Hz 闪烁的效果。EINK_SPIM 与 7 个进出引脚用于控制 EINK 的各个引脚信号传输。CapSense_1 模块用于控制和处理 CapSense 信号，UART_DEBUG 用于向上位机发送 Debug 信息与亮度信息。BLE 模块用于负责蓝牙服务。PWM_1 用于控制 RGB-LED 的亮暗。

其中各固件的参数设置情况如下：

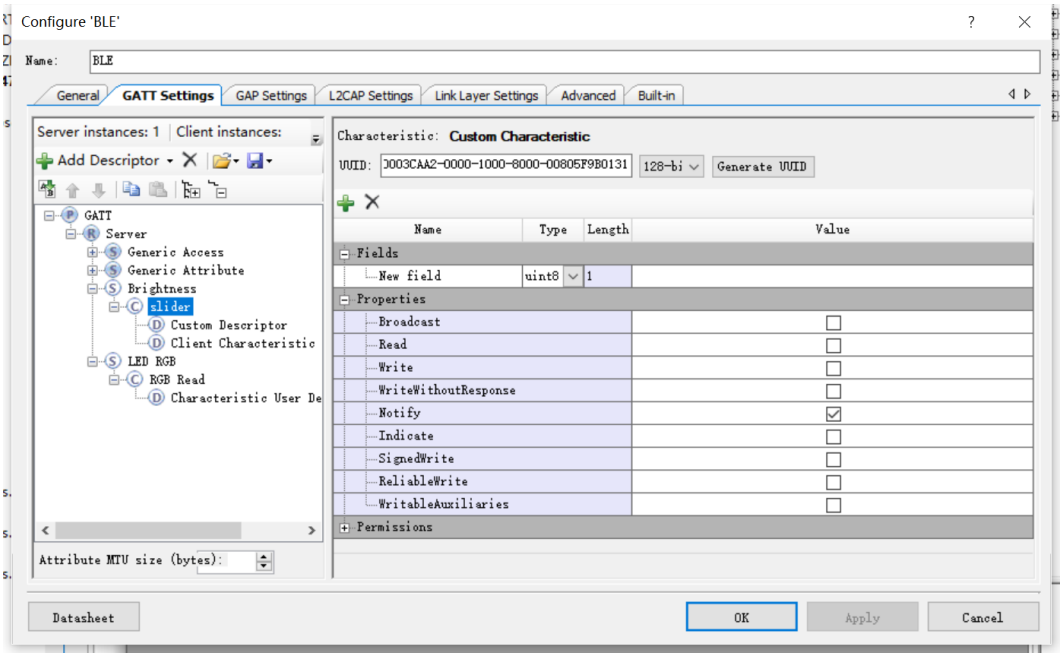


图 2 BLE 的 GATT 设置（滑块的 Characteristic（UUID）设置）

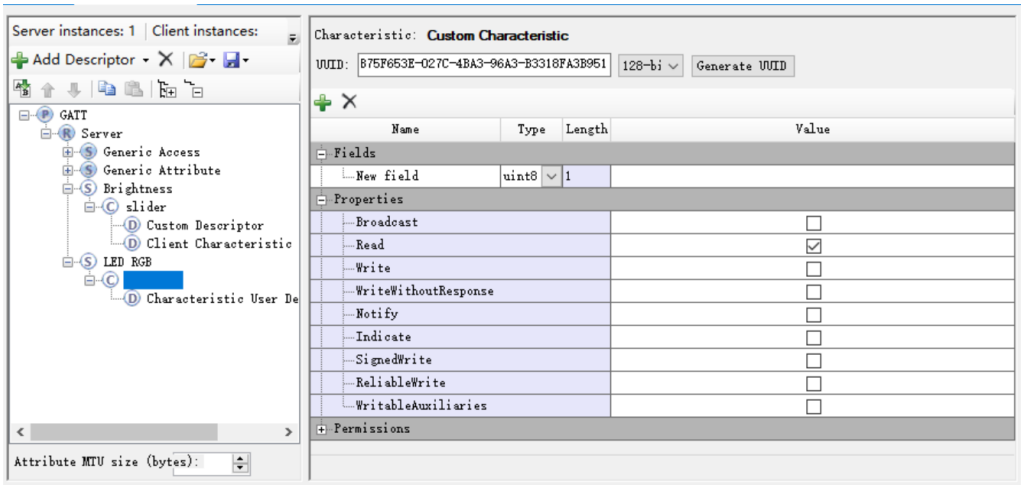


图 3 BLE 中 GATT 设置（RGB-LED 亮度信息 Characteristic 设置）

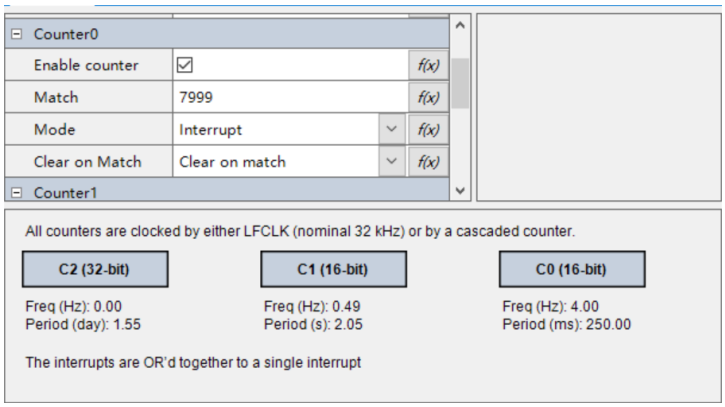


图 4 MCWDT 的时钟参数设置

Type	Name	Sensing mode	Sensing element(s)				Finger capacitance
●	Button0	CSX (Mutual-cap)	1	Rx	1	Tx	N/A
○	Button1	CSX (Mutual-cap)	1	Rx	1	Tx	N/A
⋙	LinearSlider0	CSD (Self-cap)	5	Segments			0.16 pF

图 5 CapSense 参数设置

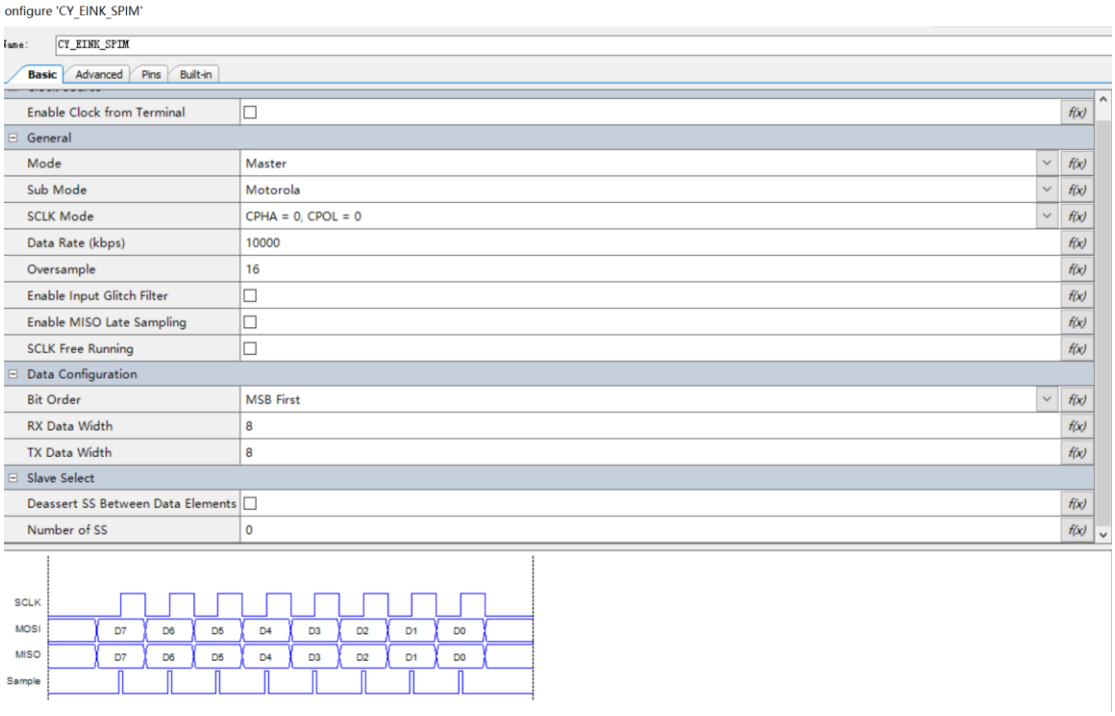


图 6 EINK_SPIM 的参数设置

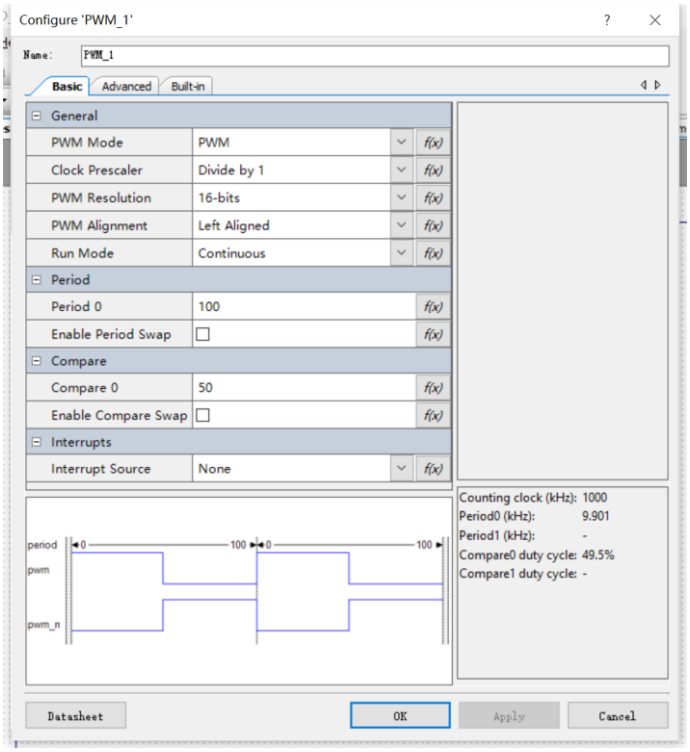


图 7 PWM 参数设置

1.3 各进程功能分配情况

在固件的各个 FreeRTOS 任务进程中，各个进程的功能罗列如下概览图所示：

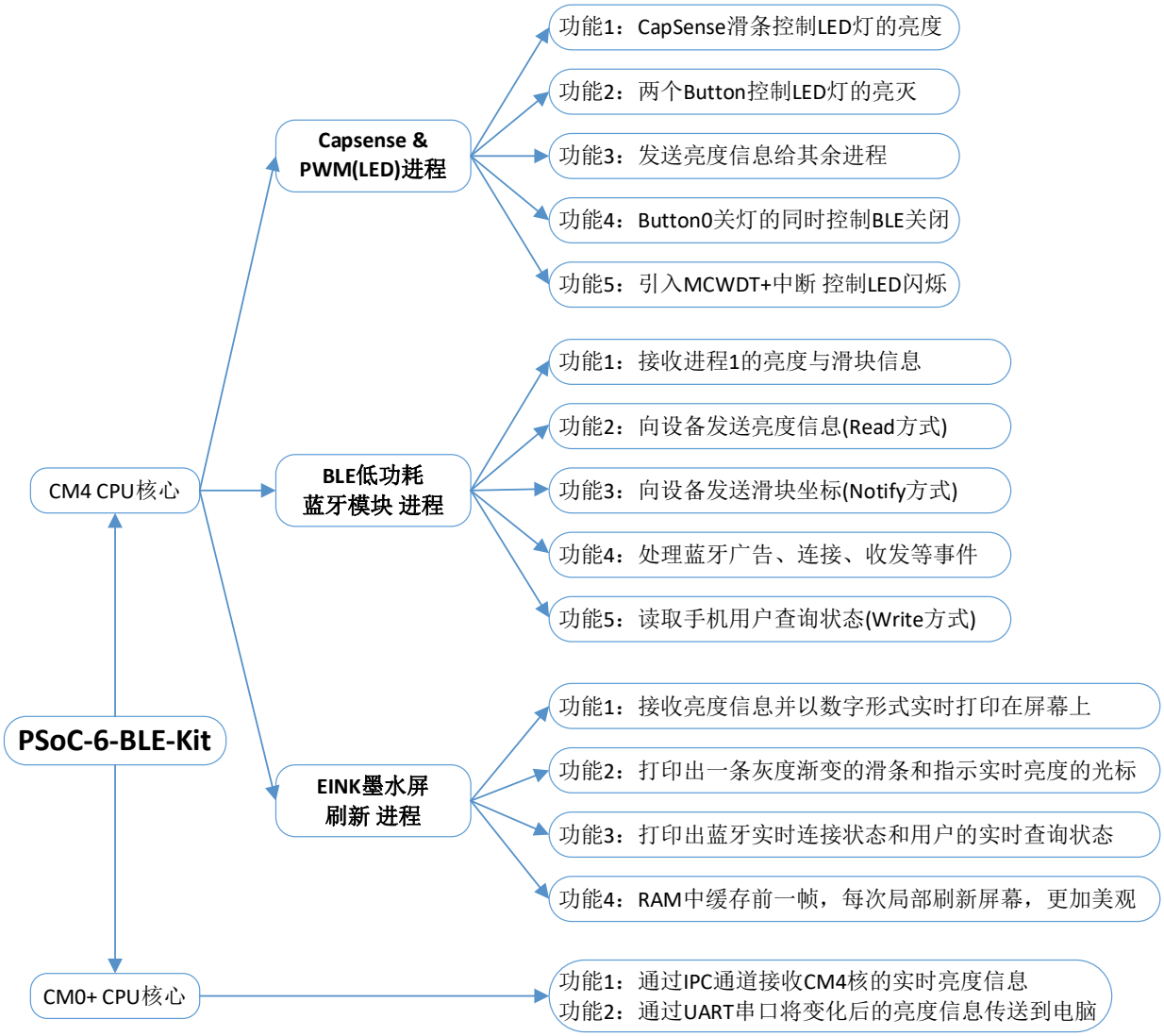


图 8 功能分配情况概览

1.4 固件代码与执行流程的具体说明

1.4.1 Cortex M0+ CPU 的固件代码

代码如下前图，为直观展示代码执行的流程，作出流程图如后图所示。

```
#include <stdio.h>
#include "project.h"

int main(void)
{
    uint8_t* brightness_read;
    uint8_t old_read;
    IPC_STRUCT_Type *ipcHandle;

    __enable_irq();
    UART_DEBUG_Start();
    Cy_IPC_Sema_Init(CY_IPC_CHAN_SEMA, sizeof(NULL), NULL);
    Cy_BLE_Start(NULL);
    Cy_SysEnableCM4(CY_CORTEX_M4_APPL_ADDR);
    ipcHandle = Cy_IPC_Drv_GetIpcBaseAddress(8u) ;
    while (Cy_IPC_Drv_ReadMsgPtr(ipcHandle, (void**)(&brightness_read)) != CY_IPC_DRV_SUCCESS){
        // pass
    };
    Cy_IPC_Drv_LockRelease(ipcHandle, 0ul) ;
    while(1)
    {
        Cy_BLE_ProcessEvents();
        if (*brightness_read != old_read) {
            printf("CM0+ read Brightness = %d\n", *brightness_read);
            old_read = *brightness_read;
        }
    }
}
```

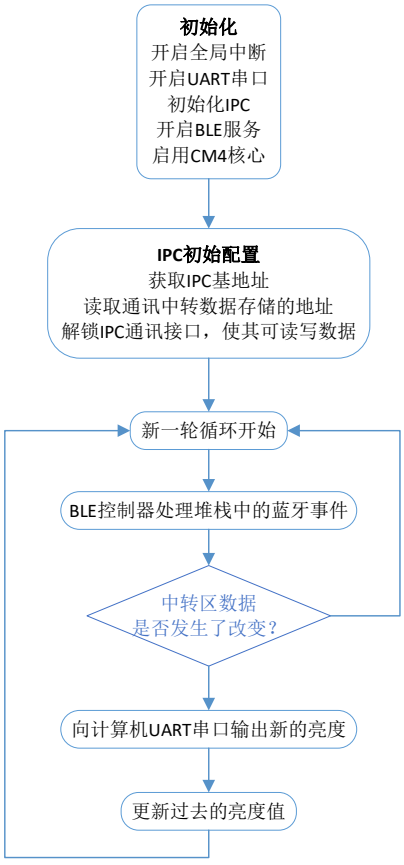


图 9 CM0+ CPU 核心流程图

1.4.2 Cortex M4 CPU 的固件代码

1.4.2.1 CM4 整体流程图及代码

CM4 的 main 函数运行代码如下所示。xTaskCreate 函数中的参数取值原因可见后文（问题解决部分）的详细说明。同样，为了展示直观，也作出了流程图。

```
int main()
{
    __enable_irq(); /* Enable global interrupts. */
    IPC_STRUCT_Type* ipcHandle ;
    ipcHandle = Cy_IPC_Drv_GetIpcBaseAddress(8u) ;
    while (Cy_IPC_Drv_SendMsgPtr(ipcHandle , 0, &common_bri) != CY_IPC_DRV_SUCCESS) ;
    while (Cy_IPC_Drv_IsLockAcquired(ipcHandle));
    UART_DEBUG_Start();
    int i = 0;
    while(i++<100){printf("\n");} // to clear the debug stdout cache
    PWM_1_Start();
    Init_Frame_Slider();
    xTaskCreate(Task_EINK, "Task_EINK", configMINIMAL_STACK_SIZE*10, 0, 0, 0);
    xTaskCreate(Task_LED, "Task_LED", configMINIMAL_STACK_SIZE, 0, 0, 0);
    xTaskCreate(Task_BLE, "Task_BLE", configMINIMAL_STACK_SIZE * 20 , 0, 0, 0);
    printf("BONJOUR!\n");
    vTaskStartScheduler();
    while(1) {printf("END!\n"); CyDelay(10);}
}
```

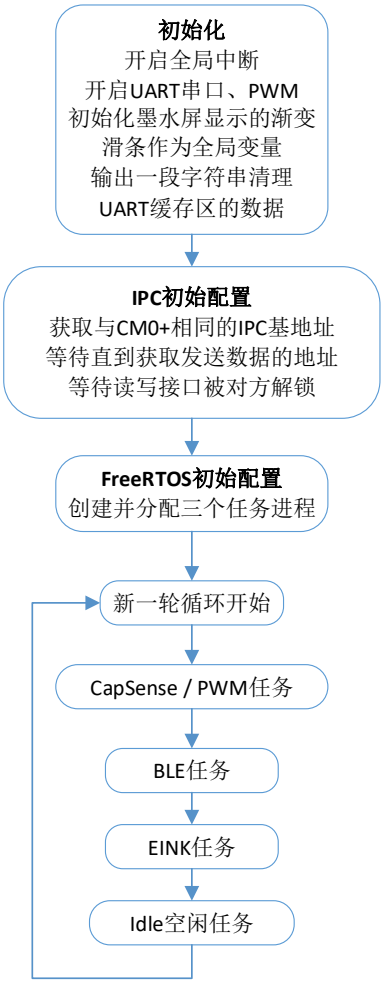


图 10 CM4 整体流程图（4 个任务的顺序不一定按照上图，因为四者同级）

1.4.2.2 任务一：CapSense & PWM(LED)任务

```
void Task_LED(void) {
    int b0new, blnew, slider;
    int old_slider=0, brightness=0, old_bri=0, b0old=0, blold=0;
    CapSense_1_Start();
    CapSense_1_ScanAllWidgets(); // capsense scan
    while(1) {
        if(!CapSense_1_IsBusy()){
            CapSense_1_ProcessAllWidgets();//data conversion
            b0new=CapSense_1_IsWidgetActive(CapSense_1_BUTTON0_WDGT_ID);//btn0
            blnew=CapSense_1_IsWidgetActive(CapSense_1_BUTTON1_WDGT_ID);//btn1
            slider=CapSense_1_GetCentroidPos(CapSense_1_LINEARSLIDER0_WDGT_ID);//slider
            if(b0new && b0old==0){
                brightness = 0;
                if (ble_killed == 0){
                    ble_killed = 1;
                    memset(Text_Conn, 0, 255*sizeof(char));
                    sprintf(Text_Conn, "Press Reset to Start BLE...");
                    memset(Text_Check, 0, 255*sizeof(char));
                    memset(Text_Check2, 0, 255*sizeof(char));
                    changed ++;
                    NVIC_DisableIRQ(MCWDT_isr_cfg.intrSrc);
                }
            }
            if(blnew && blold==0){
                brightness = 100;
            }
            if(slider != 0xFFFF) {
                brightness = slider;
            }
            if(slider != old_slider) {
                int res0 = xQueueSend(BLE_Slider_Queue, &slider, 0);
            }
            if (old_bri != brightness) {
                Cy_TCPWM_PWM_SetCompare0(PWM_1_HW, PWM_1_CNT_NUM, (uint)(brightness));
                old_bri = brightness;
                int res1 = (xQueueSend(Brightness_Queue, &old_bri, 0) == errQUEUE_FULL);
                int res2 = (xQueueSend(BLE_Bright_Queue, &old_bri, 0) == errQUEUE_FULL);
                vTaskDelay(100);
            }
            taskYIELD();
            b0old=b0new; blold=blnew; old_slider = slider;
            CapSense_1_ScanAllWidgets(); // scan widgets
```

该任务主要用于处理和 CapSense / Button / LED 亮度有关的各个事项。下面介绍该任务算法的流程。下图为算法流程图，方框中的 5 个语句块是依次判断并执行的。其中需要额外说明的是：Slider 值变化时，需要及时发送新值给 BLE 模块，是为了能够在手机 CySmart 软件中达到实时显示滑条位置的功能。而亮度值发生变化时则要及时将亮度的新值发送给 BLE 和 EINK，用于手机和墨水屏实时显示亮度值。

此处的任务间信息交互采用了 **xQueueSend** 的 API 接口，实现将数据加入空间为 1 字节的队列的功能。该函数的等待时间参数笔者设置为了 0，即不进行阻

塞等待，如果队列空间已经满了，就直接跳过即可。这样做是为了避免多任务联合运行时，各任务之间配合不默契，导致产生了过多的进程阻塞或挂起，造成栈溢出，超时卡死。

另外，此处还设置了 **taskYIELD** 的指令，这与 FreeRTOS 的任务调度规则有关。笔者将 FreeRTOS 的任务调度规则设置为“同级任务轮转 / 允许抢占资源 / 空闲任务主动出让资源”的模式，笔者在每一轮循环最后手动加入一个 taskYIELD，是为了保证各个任务每轮都只运行一次，尽量在运行完之后让出资源。使得任务之间尽可能同步，降低多任务调度卡死的可能性。

对于故障排除与算法改进的具体说明可见后文。

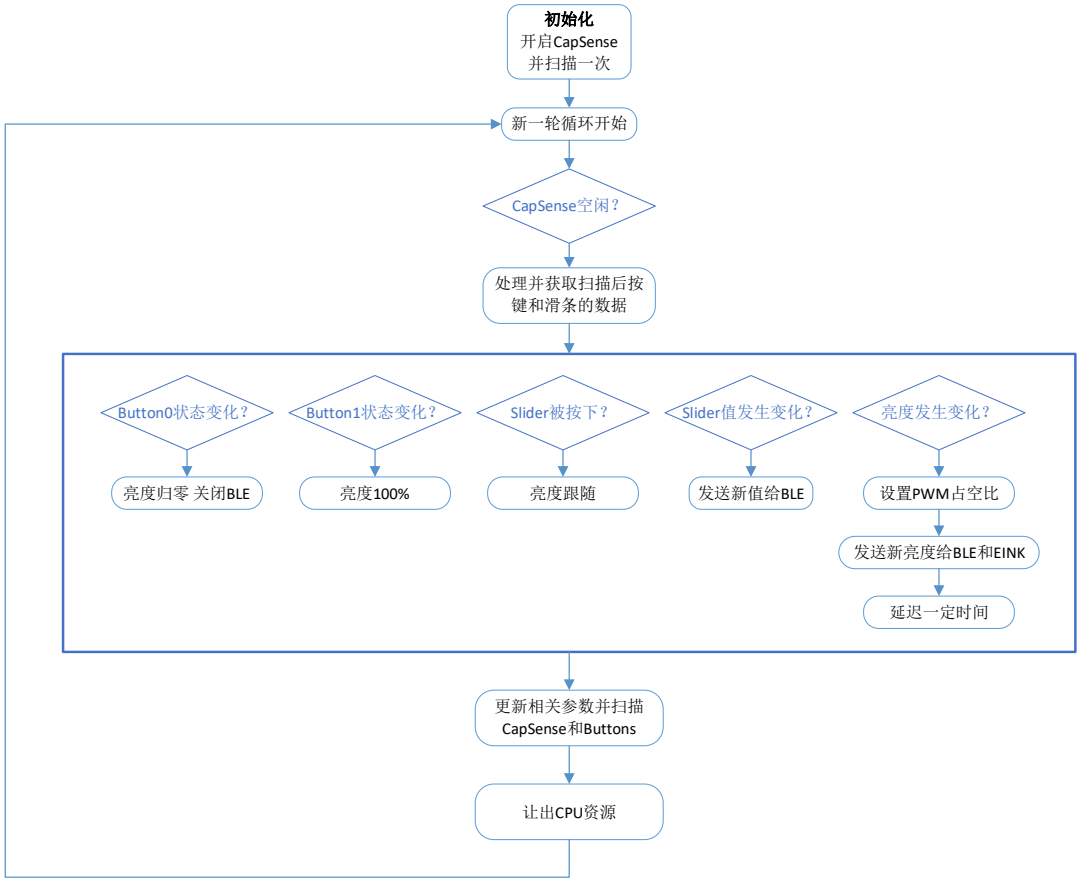


图 11 CapSense-PWM 任务的流程

1.4.2.3 任务二：BLE 低功耗蓝牙模块任务

如下所示，从上到下分别为 BLE 任务的主体函数代码、BLE 任务的初始化代码、循环处理的代码，以及单时钟看门狗 MCWDT 的中断处理函数。其中，进行初始化的函数流程基本是事先规定好的，因此笔者必须参考相关示例代码。

```
void Task_BLE(){
    BLE_Bright_Queue = xQueueCreate(1,sizeof(uint8_t));
    BLE_Slider_Queue = xQueueCreate(1,sizeof(uint8_t));
    BleDisplay_Init();
    BaseType_t recep_result0, recep_result1;// = pdFALSE;
    uint8_t bright, slider;
    while(1){
        if (ble_killed) Cy_BLE_Stop();
        else {
            BleDisplay_Process();
            recep_result0 = xQueueReceive(BLE_Slider_Queue, &slider, 0);
            recep_result1 = xQueueReceive(BLE_Bright_Queue, &bright, 0);
            if (recep_result0 == pdTRUE){
                BleSendNotify(slider, CY_BLE_BRIGHTNESS_SLIDER_CHAR_HANDLE);
            }
            if (recep_result1 == pdTRUE){
                BleSendNote(bright, CY_BLE_LED_RGB_RGB_READ_CHAR_HANDLE);
            }
        }
        taskYIELD();
    }
}

void BleDisplay_Init(void) {
    cy_en_ble_api_result_t          apiResult;
    cy_stc_ble_stack_lib_version_t  stackVersion;
    apiResult = Cy_BLE_Start(customEventHandler);
    Cy_MCWDT_Init(MCWDT_HW, &MCWDT_config);
    Cy_MCWDT_Enable(MCWDT_HW, CY_MCWDT_CTR0, 93);
    Cy_MCWDT_SetInterruptMask(MCWDT_HW, CY_MCWDT_CTR0);
    Cy_SysInt_Init(&MCWDT_isr_cfg, &MCWDT_Interrupt_Handler);
    NVIC_ClearPendingIRQ(MCWDT_isr_cfg.intrSrc);
    NVIC_EnableIRQ(MCWDT_isr_cfg.intrSrc);
    if(apiResult != CY_BLE_SUCCESS) printf("BLE Start Error! \n");
    else printf("BLE Successfully Start! \n");
    apiResult = Cy_BLE_GetStackLibraryVersion(&stackVersion);
    if(apiResult != CY_BLE_SUCCESS) printf("Error! : Cy_BLE_GetStackLibraryVersion! \n");
    else printf("Successful! : Cy_BLE_GetStackLibraryVersion: %d.%d.%d.%d \n",
        stackVersion.majorVersion,
        stackVersion.minorVersion,
        stackVersion.patch,
        stackVersion.buildNumber);
}

void BleDisplay_Process(void){
    Cy_BLE_ProcessEvents();
}

bool dsabled = 1;
void MCWDT_Interrupt_Handler(){
    Cy_MCWDT_ClearInterrupt(MCWDT_HW, CY_MCWDT_CTR0);
    /* Clear the CM4 NVIC pending interrupt for MCWDT */
    NVIC_ClearPendingIRQ(MCWDT_isr_cfg.intrSrc);
    if (dsabled == 1) {
        PWM_1_SetCompare0(2);
        dsabled = 0;
    }
    else {
        PWM_1_SetCompare0(30);
        dsabled = 1;
    }
}
```

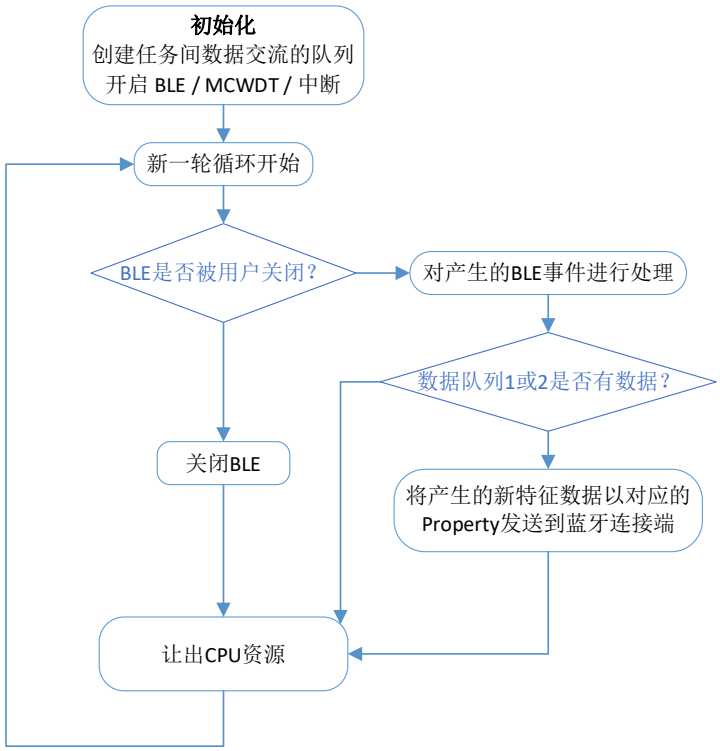


图 12 BLE 任务的流程图

上图为 BLE 任务的总体流程图。其中，使用 xQueueCreate 函数创建大小为 1 字节的数据队列，便于管理多任务间的数据收发。

BleDisplay_Init 用于处理 BLE、MCWDT 等固件的初始化问题。在该函数中，调用相关接口初始化（并使能）BLE、MCWDT、Interrupt。除此之外，还需要分别调用 Cy_MCWDT_SetInterruptMask、Cy_SysInt_Init、NVIC_ClearPendingIRQ、NVIC_EnableIRQ 等函数将 MCWDT 与其中断相配对。

MCWDT_Interrupt_Handler 用于处理 MCWDT 每隔一定时间呼叫的中断。在该函数中，每当中断到来，将 MCWDT 中的中断和之前挂起的中断清除，用一个变量来记录当前的 PWM 状态，如果 LED 正常亮，则将 PWM 占空比归零，反之则给出 50%的占空比。由于中断的时钟为 4Hz，因此 MCWDT 与 PWM 一同产生了 LED 灯 2Hz 闪烁效果。

BleDisplay_Process，即 Cy_BLE_ProcessEvents 函数用于对已经发生的蓝牙事件进行响应。其中在初始化时内嵌了 customEventHandler 函数作为 Hook 函数，用于方便开发者定义事件处理的逻辑。由于该函数的代码较冗长，故不在此贴出。下图总结展示了每一事件的响应流程与变量含义。

图 13 蓝牙事件与笔者定义的对应操作

（上方三个变量为实时打印在 EINK 上的字符串变量）

由上可见，该 Handler 作为钩子函数主要是用来处理开闭蓝牙广告、更新 EINK 关于蓝牙的显示信息、根据连接情况开闭 LED 闪烁模式、记录配对终端的连接句柄等等操作。

如下给出的代码则分别是 BLE 向配对终端发送 Read 信息和 Notify 信息的函数。Read、Notify、Write 等行为术语称为某种属性 Property。Read 信息需要用户发出 Read 请求才能获得，用于亮度查询；Notify 则能够连续不断地给出相关的数据，不需要时时刻刻发出读取请求，本实验中用于 CapSense 的手指实时位置

显示。我们需要将数据的值、长度、GATT 特征句柄、连接句柄等信息打包成一个实例送入 BLE 应用生成的 API 接口。即可向对应的设备发送 Read 信息或 Notify 信息。

```
void BleSendNote(uint8_t val, cy_ble_gatt_db_attr_handle_t dataType) {
    cy_stc_ble_gatt_handle_value_pair_t  data_pair;
    data_pair.value.val = &val;
    data_pair.attrHandle = dataType;
    data_pair.value.len = 1;
    Cy_BLE_GATTS_SendNotification(&appConnHandle, &data_pair);
}

void BleSendNotify(uint8_t val, cy_ble_gatt_db_attr_handle_t dataType) {
    cy_stc_ble_gatts_handle_value_ntf_t  data_ntf;
    data_ntf.handleValPair.value.val = &val;
    data_ntf.handleValPair.attrHandle = dataType;
    data_ntf.handleValPair.value.len = 1;
    data_ntf.connHandle = appConnHandle;
    Cy_BLE_GATTS_Notification(&data_ntf);
}
```

1.4.2.4 任务三：EINK 墨水屏刷新任务

EINK 任务代码和渐变滑条的初始化代码，如下所示：

```
void Task_EINK(void){
    Brightness_Queue = xQueueCreate(1, sizeof(int));
    memset(Text_Conn, 0, 255*sizeof(char));
    memset(Text_Check, 0, 255*sizeof(char));
    memset(Text_Check2, 0, 255*sizeof(char));
    changed = 0;
    int bri;
    Cy_EINK_Start(20, EINK_Delay);
    Cy_EINK_Power(CY_EINK_ON);
    Cy_EINK_Clear(CY_EINK_WHITE_BACKGROUND, CY_EINK_POWER_MANUAL);
    CY_EINK_SPIM_Start();
    memset(Frame[1], 255, CY_EINK_FRAME_SIZE*sizeof(cy_eink_frame_t));
    while(1){
        if (xQueueReceive(Brightness_Queue, &bri, 0) == pdTRUE || last_changed != changed) {
            if (last_changed != changed) {
                last_changed = changed;
            }
            else{
                common_bri = bri;
                memset(Text, 0, 255*sizeof(char));
                sprintf(Text, "BRIGHTNESS %d%%", bri);
            }
            TextCor[0] = TextCor[1] = Cor_1[0] = Cor_2[0] = Cor_3[0] = 1;
            Cor_1[1] = 4; Cor_2[1] = 6; Cor_3[1] = 5;
            Cor_s[0] = 0; Cor_s[1] = 32; Cor_s[2] = 120; Cor_s[3] = Cor_s[2] + 30;
            Cor_J[1] = 9; Cor_J[0] = (int)(32 * (double)(bri) / 100);
            memset(Frame[0], 255, CY_EINK_FRAME_SIZE*sizeof(cy_eink_frame_t));
            Cy_EINK_ImageToFrameBuffer(Frame[0], Frame_b, Cor_s); // slider
            Cy_EINK_TextToFrameBuffer(Frame[0], Text, CY_EINK_FONT_16X16BLACK, TextCor);
            Cy_EINK_TextToFrameBuffer(Frame[0], J, CY_EINK_FONT_8X12BLACK, Cor_J);
            Cy_EINK_TextToFrameBuffer(Frame[0], Text_Conn, CY_EINK_FONT_8X12BLACK, Cor_1);
            Cy_EINK_TextToFrameBuffer(Frame[0], Text_Check, CY_EINK_FONT_8X12BLACK, Cor_2);
            Cy_EINK_TextToFrameBuffer(Frame[0], Text_Check2, CY_EINK_FONT_8X12BLACK, Cor_3);
            Cy_EINK_ShowFrame(Frame[1], Frame[0], CY_EINK_PARTIAL, false);
            memcpy(Frame[1], Frame[0], CY_EINK_FRAME_SIZE*sizeof(cy_eink_frame_t));
        }
        taskYIELD();
    }
}
```

图 14 EINK 任务代码

同样为了流程展现更直观，作出该任务的流程图如下所示。在算法中，有以下几点值得注意：

① Text_Conn / Text_Check / Text_Check2 的含义在上文介绍过。用于在屏幕上打印出蓝牙连接情况和用户查询情况。

② 除了亮度信息发生改变时要刷新屏幕以外，蓝牙状态变化时也要刷新屏幕。

③ 为了避免整块屏幕重新刷新频繁产生黑影引起视觉不适，笔者采用了局部刷新的方式来刷新屏幕，这一方式需要记录前一帧和新一帧的像素矩阵，函数会对比前后两帧的差异，只刷新发生了变化的区域。刷新之后需要更新前一帧的矩阵，用于下一次刷新。

④ 下面具体介绍渐变滑块的生成和光标的移动算法。

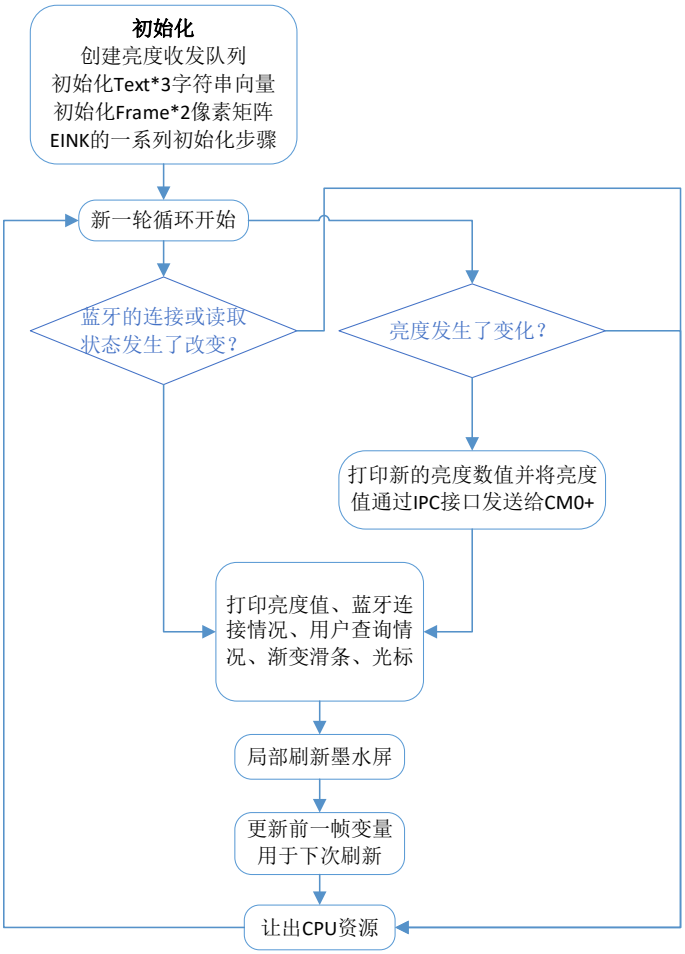


图 15 EINK 任务的流程图

```
void Init_Frame_Slider(){
    for (int t=0; t < 5808;t++){
        uint8_t b = 0;
        int prob = (int)(100 * (float)(t % 33) / 33);
        for(int j=0; j<8; j++){
            b = b << 1;
            if (rand() % 100 < prob) {b += 1;}
        }
        Frame_b[t] = b;
    }
}
```

图 16 渐变滑条的初始化代码

由于在 EINK 墨水屏上的每一个像素点只有黑白两种值，而笔者想要在屏幕上显示出一条灰度渐变的滑条，由于屏幕上的每一个像素点是相对密集微小的，所以笔者想到可以通过控制黑点的密度来在视觉上显示出一种灰度渐变的效果。初始化渐变滑条的代码如上图所示。具体逻辑就是对于每一像素点根据其横坐标计算其为黑的概率，通过控制为黑的概率来控制局部的密度，从而产生灰度变化的视觉效果。而滑条上的光标则是通过将其横坐标与亮度值建立联系，来控制其在滑条上的位置。

最终在墨水屏上的效果如下图所示：

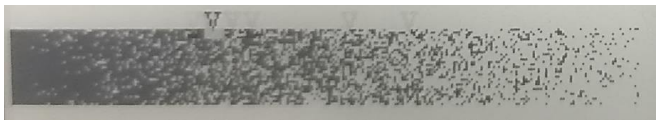


图 17 渐变滑条+光标的打印效果

1.5 实验与演示结果

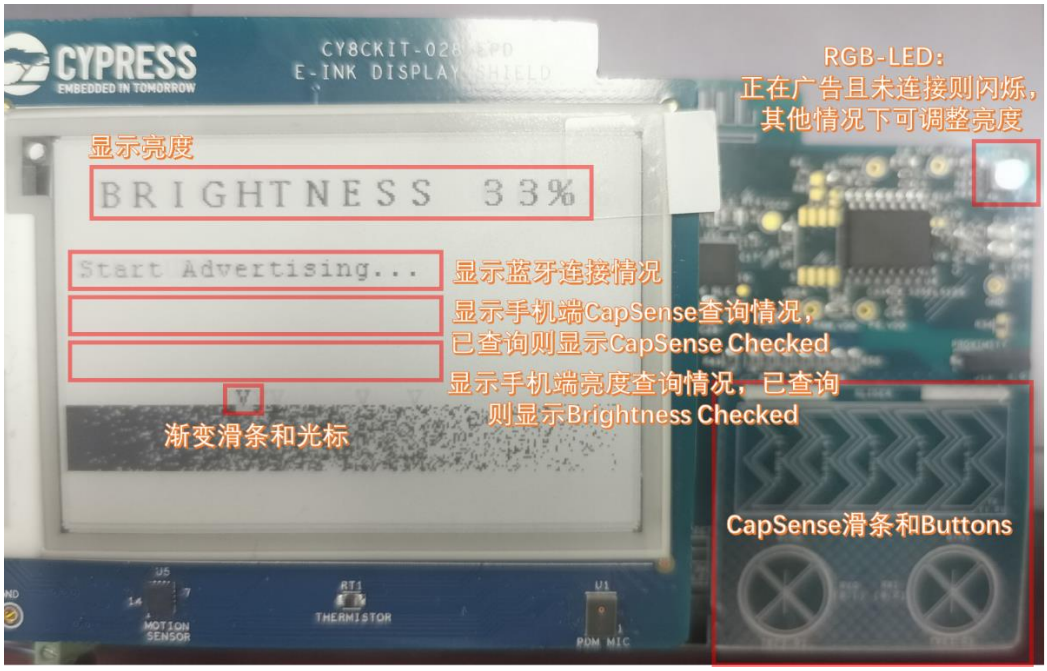


图 18 PSoC6 实验结果图

PSoC6 板上的实验效果如上图所示。经过测试，LED 亮度、打印的亮度值、亮度光标的变化、手机端 CySmart 中获取的实时滑条位置、手机端读取的亮度值以及计算机端接收到的亮度值都能够与手指在 CapSense 上的实时位置准确对应，且延迟均很小（估测在 0.5s 以内）。基本要求与附加功能均达到了预期，通过了助教的验收。

1.6 遇到的问题与解决方案

1.6.1 软件安装、驱动安装的问题

笔者下载 PSoC Creator 时不慎在安装路径中引入了中文路径，导致编译出现错误，经过不下三四次的卸载、重装，软件还是屡次出现中文相关的问题。笔者注意到，由于 Creator 是连带 Programmer 等一系列辅助程序一起安装的，因此卸载很容易不彻底，导致软件注册表仍然残留在系统中，其中保存了安装的设置，下一次安装即使设置了英文路径，安装程序也会默认使用原来的中文路径来安装，所以 Programmer 总是会被安装到中文路径上。后来笔者发现可以通过 Cypress 自己的 Software Manager 来统一 update 或 remove Cypress 旗下的各种软件，卸载是比较彻底的，重新下载就不会出现之前的问题了。

1.6.2 FreeRTOS 多任务运行到一半崩溃

在调试中，FreeRTOS 多任务并行常常崩溃,运行到一半卡住。由于 FreeRTOS 的调度算法被封装的很好，因此如果要深入调试其中的故障，还需要更深入了解 FreeRTOS 实现原理的相关细节。笔者自学了 FreeRTOS 的基本框架之后，大致掌握了其运行原理，经过不懈的调试，解决了代码中存在的若干问题。

这里的问题有很多，经过笔者的反复调试、反思、总结，出现的问题主要有以下几点：

① **xTaskCreate 函数中，堆栈深度大小、任务级别等参数控制不当。**PSoC6 上 RAM 的空间有限，最多只能放下 3 张 5808 字节的单帧墨水屏矩阵，因此如果开了过大的栈空间，可能整个固件烧入后就立即停在了初始阶段。笔者尝试将栈空间的参数适当调小一些，然后将 FreeRTOS_Config 文件中的 config-

TOTAL_HEAP_SIZE 等参数适当调大一些就可以了，这里需要不断试错，没有很好的估计方法。

```
xTaskCreate(Task_EINK, "Task_EINK", configMINIMAL_STACK_SIZE*10, 0, 0, 0);
xTaskCreate(Task_LED, "Task_LED", configMINIMAL_STACK_SIZE, 0, 0, 0);
xTaskCreate(Task_BLE, "Task_BLE", configMINIMAL_STACK_SIZE * 20 , 0, 0, 0);
```

图 19 任务创建时的参数设置

另外，任务的级别可以进行更好的设置，笔者这里将三个任务的级别都定义为 0 级，即与系统原生的空闲任务同级。并将 Config 文件中的若干参数进行了如下的配置。相关的解释如下所示。

```
#define configUSE_PREEMPTION                    1   // 允许高级任务抢占资源
#define configUSE_TIME_SLICING                  1   // 开启统计任务轮盘转模式
#define configIDLE_SHOULD_YIELD                 1   // 空闲任务应主动让出资源
```

注意到，IDLE 任务默认为 0 级。将 3+1 个任务设置为同级，是为了保证同级任务能够轮盘转，保证信息交互的同步性，尽量减少资源的空耗与过多阻塞任务被挂起的可能性。同时，空闲任务也得到了一定的运行机会，用于及时清理已处理的任务栈空间，释放空间，并输出相关任务运行状态的调试信息(利用 Hook 函数，见后文)。空闲任务运行一定时间后，应当及时让出资源。

② **任务间数据的收发不同步，阻塞超时**。笔者 Reset 板子之后，固件运行了一段时间，就完全卡住了，一开始笔者不知道发生了什么，束手无策。自学了 FreeRTOS 之后，笔者通过定义空闲任务的 Hook 函数 vApplicationIdleHook，并调用 API 读取各个任务状态的详细信息，来输出 FreeRTOS 当前运行状态的具体细节。笔者观测到随着程序运行，任务最后都逐一从 Blocked 状态到被 Suspended 了。基于此，更深入检查自己的代码之后，笔者注意到 xQueueReceive 函数的参数 xTickstoWait 不应被设置为 Maximal 模式，此时如果队列中没有数据，系统会一直阻塞进程直到队列中有了新的数据。这样一来如果任务之间配合不默契，积压的阻塞进程容易导致 FreeRTOS 超时 TimeOut，进而自动结束任务调度。最后就绪的程序就只剩下空闲任务了。发现了问题，解决方法就很简单了：尝试将收发消息的函数参数均设置为不阻塞等待（0），这样的话就可以避免阻塞等待时间过长。

③ **仍存在卡在初始化状态的问题，栈空间溢出**。笔者同样通过空闲任务的

钩子函数来输出 FreeRTOS 的调试信息，发现 EINK 任务的剩余栈空间只剩下几个字节，于是程序被挂起。笔者据此查看了任务代码中的存储占用情况，发现如下变量占据了大量存储字节。

```
cy_eink_frame_t    Frame[2][CY_EINK_FRAME_SIZE]={0};
```

解决方法就是将该变量定义为全局变量，而非函数内的变量，避免占用任务栈本身的空间。修改之后，任务就能正常运行了。

1.6.3 EINK 显示与滑块和亮度不同步

出现任务之间的输出结果错位，比如手指从 60%滑到了 99%，但是 EINK 显示与滑块和亮度有时不同步，显示的是上一个值 60%，但是有时又能够同步（显示 99%），不太稳定，所以这个问题困扰了笔者一段时间。

笔者最终的解决方法为，在亮度变化时立即 taskYIELD 让出 CPU 空间给其他任务，而 EINK 每刷新一次也 taskYIELD——每个任务只循环一次就换成下一个任务，由于任务迭代相当迅速，比人手的变化速度更快，所以在数据交互上，可以使二者基本同步。

【注】taskyield 应该是在该处暂时跳出给其他同级别的任务，下轮再从此处开始。也就是，不会整个函数重新运行一遍。可见，taskyield 放置的位置也非常重要。

1.6.4 显示灰度

如前文所述，灰度的显示也困扰了笔者一段时间，后来想到的方法已经介绍过。这里还可以补充一下墨水屏每一像素点的控制方法。

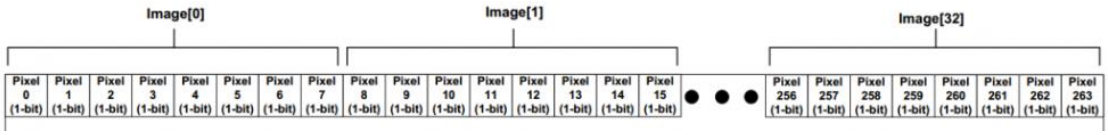


图 20 图源 https://www.21ic.com/evm/evaluate/MCU/201802/752221_2.htm

Frame 数组中存储的每一字节的 8 位二进制 01 值代表了相邻 8 个像素点的黑白二值状态。长度为 5808 字节，代表了横向 33*8 个像素，纵向 176 像素的像素矩阵。于是就可以通过给 uint8 的每一位二进制值赋值来控制每一个单独的像素值。

1.6.5 初始化不显示蓝牙信息

在上电测试时，发现始终不在 EINK 上显示预期应显示的蓝牙信息，后来发现这是一个问题耦合导致的故障，由两部分组成：

① 3+1 个 task 同级别，调度时是不一定按照预期的顺序执行的，所以可能在 EINK 初始化完之后，才初始化蓝牙 BLE 模块，导致 EINK 显示的变量没有及时被赋值就显示了。

② 由于后一帧是根据前一帧来进行局部刷新的，所以如果第一帧没有打印出来信息，后续该信息如果不更新，就一直不会打印出来。

解决上述第一个初始化问题即可。将变量的初始化放在主函数中，让其最先完成初始化即可。

1.7 综合实验总结

1.7.1 仍需改进的点

① 蓝牙 BLE 的原理细节没有做非常深入的研究，没有对 BLE 广告时的数据包进行更好的设计，使得手机端查看时缺乏用户体验，在未来进一步深入了解 GATT、GAP 等概念之后，可以做一些改进。

② 目前只能通过 CySmart 软件来收发信息，如果可以改进，可以考虑用树莓派、计算机等设备来直接向 PSoC 配对、连接、收发信息。

③ 墨水屏的视频流显示功能，笔者理论估算过大致是可行的，并完成了相关代码。但是树莓派系统出现了一些问题，无法使用 Python 进行 BLE 连接通讯。

1.7.2 体会

因为笔者很喜欢带有墨水屏的产品，因此对 PSoC6 也是相当喜爱，在墨水屏的显示上下了一些功夫，最终的结果也相对满意。但是如果未来有时间的话还是希望能够在墨水屏上显示蓝牙传输的视频流，这样会很酷！其实笔者已经实现了相关代码并设计了视频帧分 12 个包传输的数据结构，只是树莓派系统有些漏洞，无法安装针对 BLE 的 Python 蓝牙通信依赖库，因此不能发送视频，非常遗憾。在综合实验中，除了将基本实验中的众多功能整合起来以外，笔者还实现了一些较为创新的想法，正如前文介绍的那样。总结来说，综合实验锻炼了笔者 SoC 开发的基本能力，深化了笔者对计算机、电路中相关概念的认识，收获相当丰富。

1.7.3 小建议

在软件 PSoC Creator 中，老师介绍的 Datasheet 非常好用。此外也可以介绍一下软件中的 Go To Definition/Declaration 的功能（右键函数名或变量会出现），进入库函数的源代码可以进一步学习源代码的逻辑，更重要的是，源代码中有丰富的注释，这些注释针对某一个 API，这样的注释也可以在 Cypress 的官网 Doc 文档中找到，研究注释对于入门者的开发和学习是一种很大的辅助。另外，如果有时间，也可以展示一下 PSoC 的单步调试的某些技巧，用好调试功能能够为很多看似“黑箱”的故障找到一些可解释的原因，大大提高调试效率。

二、创新实验：多功能桌面辅助小车“桌团儿”

2.1 实验背景

笔者与王皓霖、贺一非共三人组队开展创新实验。我们受到 bilibili.com 视频网站上的知名博主“稚晖君”一条关于桌面机器人的创作视频的启发（来源见页面下方[1]），我们三个人开始探讨如何设计并实现我们想象中的兼具日常辅助功能与娱乐性质的桌面机器人。

概而论之，我们的预想目标大致为，设计并实现一款多功能的桌面辅助小车，能够辅助用户进行一系列的日常工作，且具有一定的娱乐功能，也就是有一定自动化程度的桌面行为、运动的能力，以及与用户进行识别、交互的能力。

我们将这一桌面辅助机器人取名为“阿桌团儿”。“团（发音 gia1）儿”是闽南方言，用来称呼孩童。如此称呼，一是寄托了我们亲切且美好的期许——这一机器人从上到下的每一实体部位都是我们组员收集（或实验室提供）、设计、拼装、调试出来的，从机械实体到硬件，再到软件算法都有较大程度的独创性，它的诞生倾注了我们两周的努力。二是表示这一桌面小车仍是新生，尚不成熟，未来可进一步改进。

2.2 实验目标与预期功能

具体而言，我们希望“桌团儿”具有如下的功能（或特征）：

2.2.1 娱乐功能

用户长时间坐在电脑桌前易感到乏味无趣，为了缓解无趣的心情，“桌团儿”可以像宠物一样与用户进行信息的交互，并能够在桌上完成面部、手势、边缘等的识别，或按指令行动等趣味功能。在娱乐模块中，又分为红外控制与语音+图像控制两种控制模式。此二者之外，还有 PSoC6 墨水屏和 LCD 触摸屏显示静态、动态待机图像的功能。

红外控制 指的是用红外遥控器控制小车前、后、左、右、斜向、或顺逆时针原地旋转，能够直接控制小车的桌上运动行为。同时能够根据超声雷达自动判断前方是否有障碍物，并亮灯警告，禁止用户控制小车继续前进。

[1] “稚晖君”:《【自制】我做了个能动的 电 脑 配 件!【软核】》，哔哩哔哩弹幕网，https://www.bilibili.com/video/BV1ka411b76m?spm_id_from=333.999.0.0&vd_source=5562a7dcc2934b931ead883489238509

语音+图像控制模式 下，机器人的状态大致可以划分为：初始状态、切换状态、旋转归位状态、跟随用户并收集手写涂鸦状态，共四种“大状态”。某些“大状态”中可能还有“小状态”的划分，我们将一一介绍。

对于第二种控制模式下的大状态，我们分别制定了如下的功能要求：

1. 在娱乐模式的“**初始状态**”下，机器人会借助麦克纳姆轮可以原地旋转的特殊功能，在原地旋转寻找用户的面部。如果旋转一圈没有用户的面部出现则停在原地，如果捕捉到用户的面部，则进一步旋转直到面部大致在视野的中央（设定了阈值，此处通过摄像头获取视野）。完成对准之后，自动进入“切换状态”。

2. 在娱乐模式的“**切换状态**”下，用户可以通过语音来控制桌面机器人进入两种娱乐状态的某一个状态中。

3. 娱乐模式的“**旋转归位状态**”。这是第一个娱乐状态，原地旋转，并归位（因此可分为两个小状态）。用户通过比划某种手势来控制机器人原地向左或向右转 45 度，经过若干秒自动回转归位。其中，比划“0”（握拳）为左转，比划“1”为右转。完成后自动回到“初始状态”。

4. 娱乐模式的“**跟随用户并收集手写涂鸦状态**”。这是第二个娱乐状态，识别并跟随用户的面部，并收集用户的涂鸦笔记。具体而言，又分为 5 个小状态。

第一个小状态是旋转寻找用户。

第二个小状态是向用户方向前进（要求面部不能够距离小车过远，否则车子容易摔下边界，此时要求小车不进行任何运动行为），若用户面部偏离视野中心则原地向用户方向旋转，再前进。

第三个小状态是预测摄像头到用户的面部距离小到一定阈值之后，控制舵机使得摄像头向下 90 度翻转，观察下方的桌子边界是否已经与车子的位姿对齐，并据此控制麦轮进行细微的调整来与桌子的边缘对齐，即正面面对用户。

第四个小状态是收起摄像头，然后等待用户在车上的 LCD 触摸屏上书写或涂鸦相关内容，并发送到树莓派上，树莓派再发送到用户个人电脑上。

第五个小状态是将小车按照原来的路径回退到初始的桌面位置上。

最后自动回到娱乐模式的“初始状态”下。

显示屏图像显示 我们要求在 PSoC6 和 LCD 触摸屏上能够显示待机图像。值得一提的是，我们原先还制定了在 PSoC6 EINK 墨水屏上二值化显示摄像头实时视频流数据的功能（交由笔者实现），该功能可用于给用户拍摄创意艺术自拍，但是由于设备（树莓派）存在众多系统缺口，很难安装调用蓝牙 BLE 的依赖库，因此没有在课上实现。

2.2.2 学习 / 办公辅助功能

“桌团儿”除了能够与用户进行交互娱乐，还要求其能够一定程度上辅助用户的学习或工作，使一些琐碎的日常工作更加便利。因此，我们制定了如下的功能要求：

1. **待办事项的管理 (To-Do List)**。手机可通过蓝牙 BLE 连接 PSoC6 开发板，手机上输入待办事项，可发送到 PSoC 端，实时显示在 EINK 显示屏上。当完成一条待办事项，可以通过 CapSense 滑条一条条滑掉已经完成的事项。并且要求能够显示出“滑动”的动态视觉效果。

2. **触摸屏的召唤与笔记采集 (Call and Note)**。这一点与娱乐模式中介绍的是相同的，该功能主要针对需要记录笔记时的情况。手写屏搭载在小车上。此时可以呼叫机器人，让其进入跟随用户并采集笔记数据的模式下，机器人就会向用户驶来，与桌边对齐，等待用户在触摸屏上写下笔记并发送到树莓派上之后，机器人自动回位。笔记数据会自动上传到用户的个人电脑上。而手写屏上的画笔可以改变颜色、改变线条的粗细等，满足大部分的日常学习笔记需求。

2.3 系统结构说明与各模块作用说明

2.3.1 系统结构图与实物图

如下各图为系统结构示意图与小车的多角度实物图。

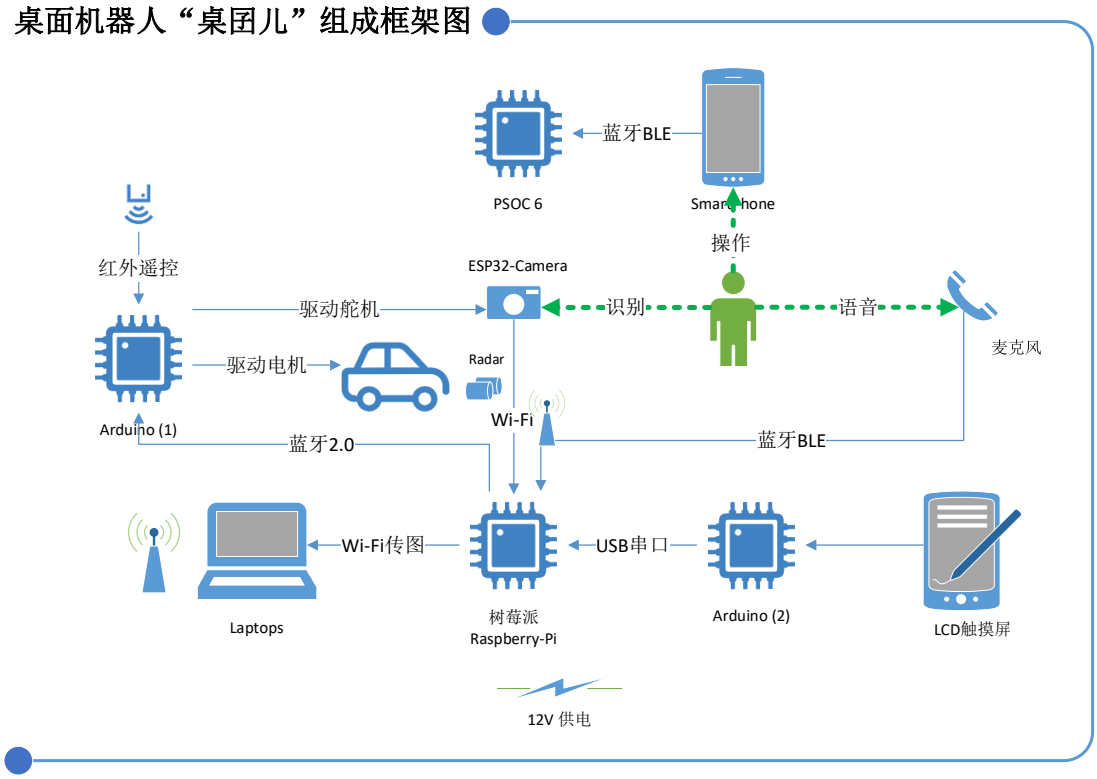


图 21 系统结构示意图

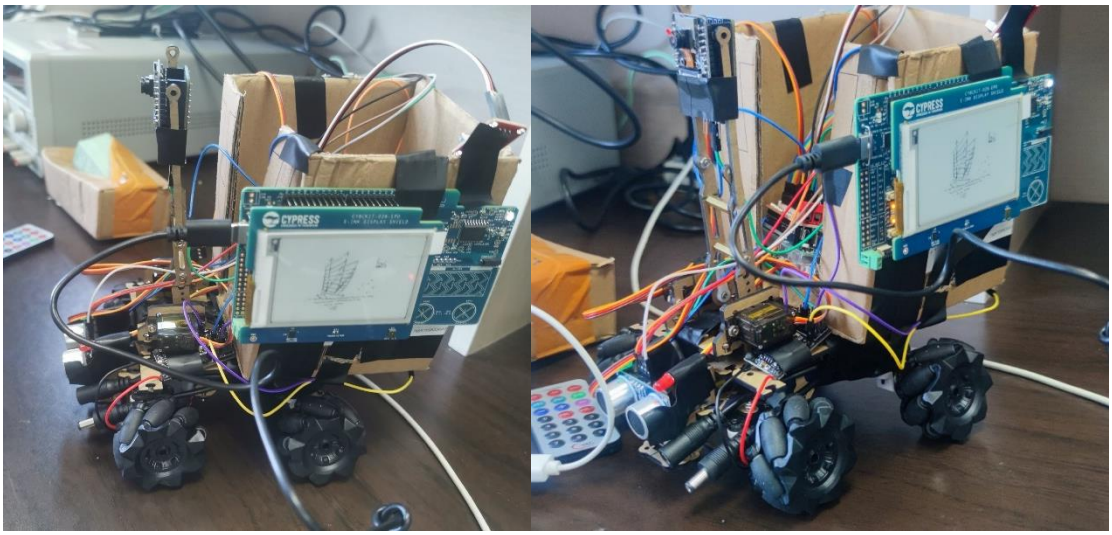


图 22 小车最终搭建的实物图

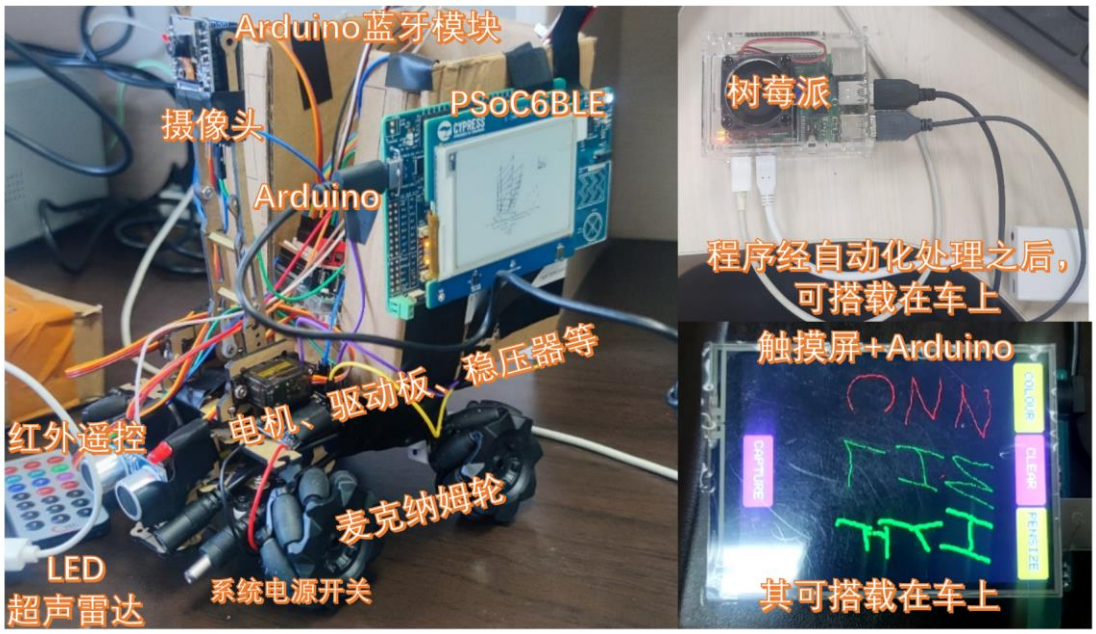


图 23 小车各部件的标注，树莓派与触摸屏

（为调试方便单独取出，实际使用时可搭载于车上）

需要额外说明的是，该小车中，能够机械运动的组件有:

① 四个麦克纳姆轮通过驱动模块控制前、后、左、右、转向等复杂的运动形式。

② 摄像头所在的连杆上含有一个舵机可以使得摄像头拍摄的俯仰角度受控制。

2.3.2 系统模块划分与通信链路说明

经过模块化梳理，本桌上小车项目的数据通讯连接框架大致可以划分为 1 条主线和 3 条支线——

主线 1：小车识别决策与运动控制链路。有两个输入源：用户向蓝牙耳机输入语音信息，蓝牙耳机通过蓝牙传送到树莓派；而 ESP32-Cam 摄像头采集环境图像，通过 Wi-Fi 将视频流传送到树莓派上。树莓派对传入的原始数据进行信息预处理、识别、决策，将决策命令通过经典蓝牙 2.0（HC-06 模块）发送给 Arduino1 用于控制麦克纳姆轮的电机与摄像头对应连杆上的舵机按照指示运动。

而运动完之后，由于小车的坐标与位姿改变，故摄像头采集的数据会相应变化，树莓派根据传入的变化后的数据进行决策，而后再次发送指令……如此不断循环，形成一个环状链路。

支线 2：PSoC6 辅助工具链路。在手机上连接 PSoC6 蓝牙并且完成信息输入，通过 BLE 建立快速链路传送到 PSoC6 内核，在墨水屏上进行显示。

支线 3：LCD 笔记传输链路。通过 LCD 触摸屏书写，将书写结果以图象的形式，通过 Arduino2 上的 USB 串口发送给树莓派，树莓派再将图像通过 Wi-Fi 连接传送给用户的个人电脑。

支线 4：Arduino 直接控制链路。可用红外遥控器向 Arduino1 发送指令，直接控制小车运动。超声雷达感应避障，控制小车避免向前运动。传感器与 Arduino 上的引脚直接相连。

如果与人类的神经系统进行类比，树莓派是小车“桌团儿”进行相对更“高级”的智能识别与决策的处理模块，用于处理人类语言（听觉）与图像（视觉），计算能力（思维）和存储空间（记忆）都较大，是小车的“大脑”。而 Arduino1/2、PSoC6 主要负责运动控制（控制骨骼与肌肉）与相对更简单直接的信号识别决策，譬如障碍物的远近、红外信号、LCD 触摸屏/PSoC 滑条信号（触觉）等，是小车的“周围神经系统”。其余的摄像头、电机、舵机、电池、触屏、麦轮、雷达等等模块可以看作小车不可或缺的直接输入/直接输出信号的一系列“器官”。

2.3.3 电路的物理连接情况

除了系统结构图中用连接线描绘的数据通路以外，有线的电路连接相对较少，主要有以下几部分：

1. 麦轮电机与电机驱动板之间的连接，驱动板与 Arduino 之间的连接。

   调节摄像头俯仰角的舵机与 Arduino 之间连接。

   雷达、LED、蓝牙 2.0 模块、红外模块等车上的独立配件直接连接 Arduino。

2. 各个设备连接电源供电。树莓派采用 5V/2A 电源适配器进行供电，PSoC6 采用 Type-C 从树莓派的 USB 接口取电。其他设备采用 12V 锂电池经过 5V 稳压器稳压转换之后，连接到各个设备（单片机、舵机、驱动板、摄像头等）进行供电。

3. 串口连接传输 LCD 触屏接收的图像信号。

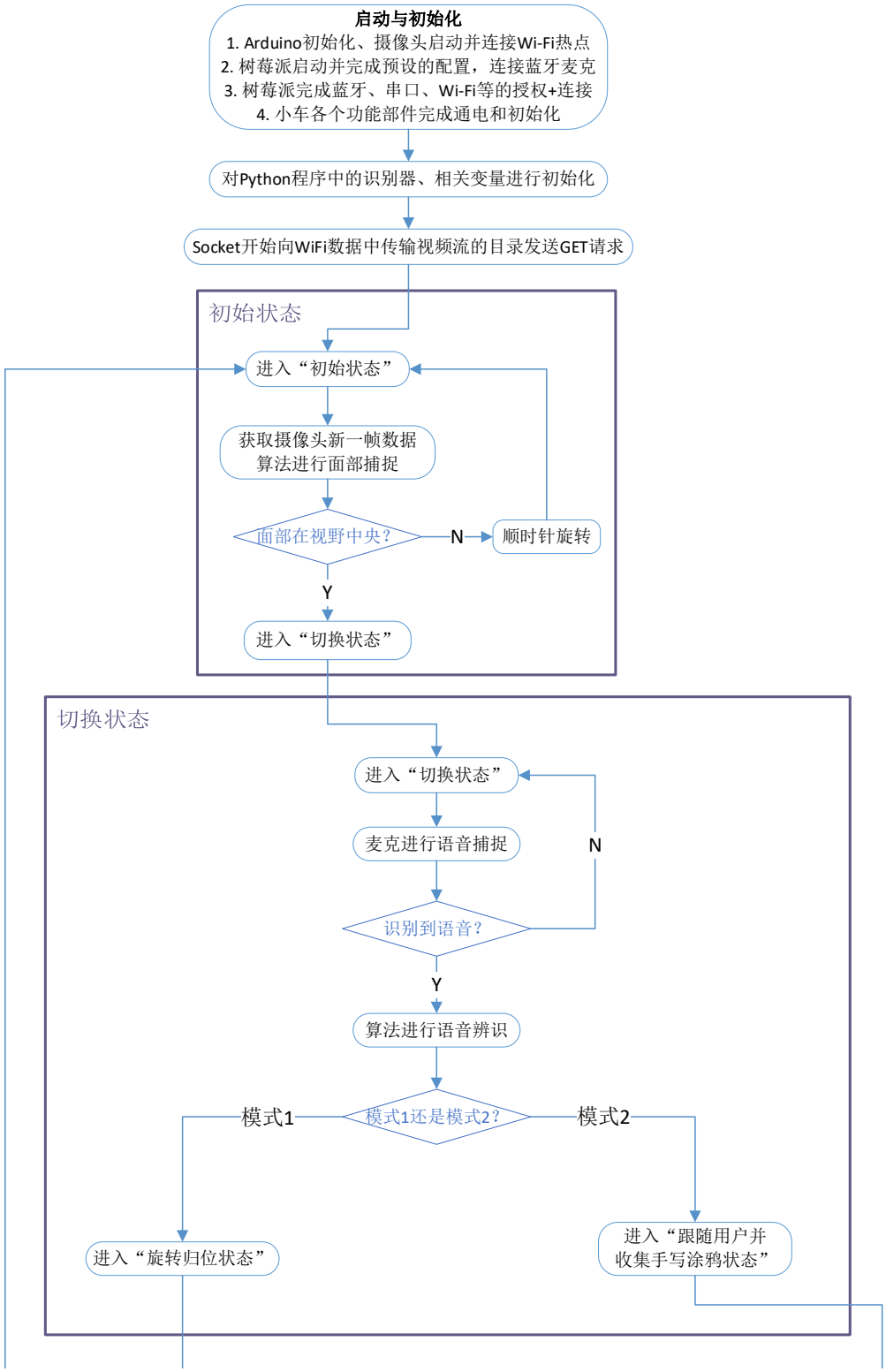
理论上，树莓派的 Type-C 电源适配器，是整个电子系统唯一与外界存在有线连接的地方。如果要改进，减少小车与外界世界的有线连接，可以寻找合适的电池来代替电源适配器给树莓派供电。

2.4 执行流程的具体说明

下面将针对笔者所负责的部分的运行流程进行详细的说明。

2.4.1 小车识别决策与运动控制链路的流程图

笔者负责了小车执行框架和识别决策算法的编写，作出主链路的执行流程图如下。



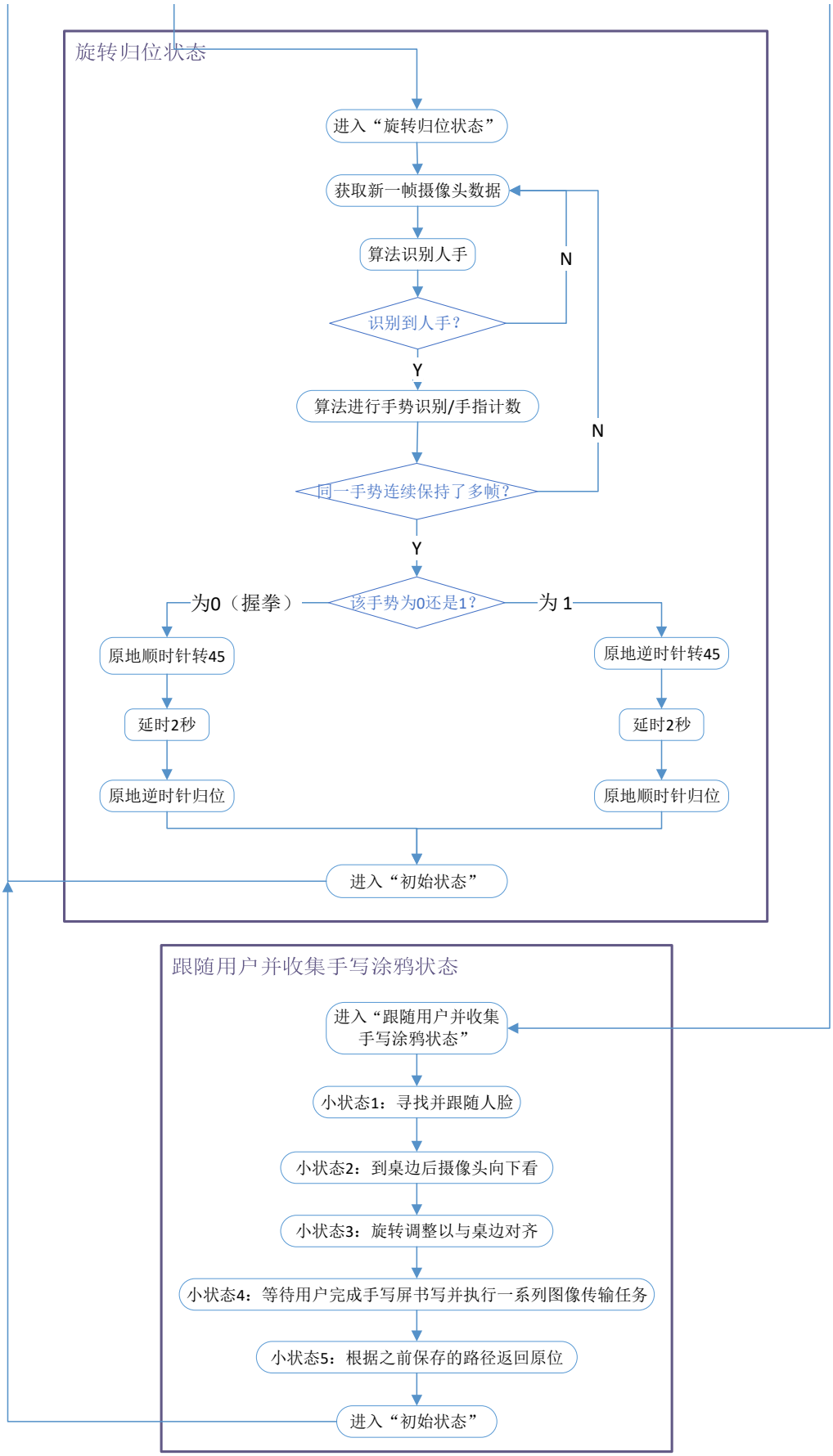
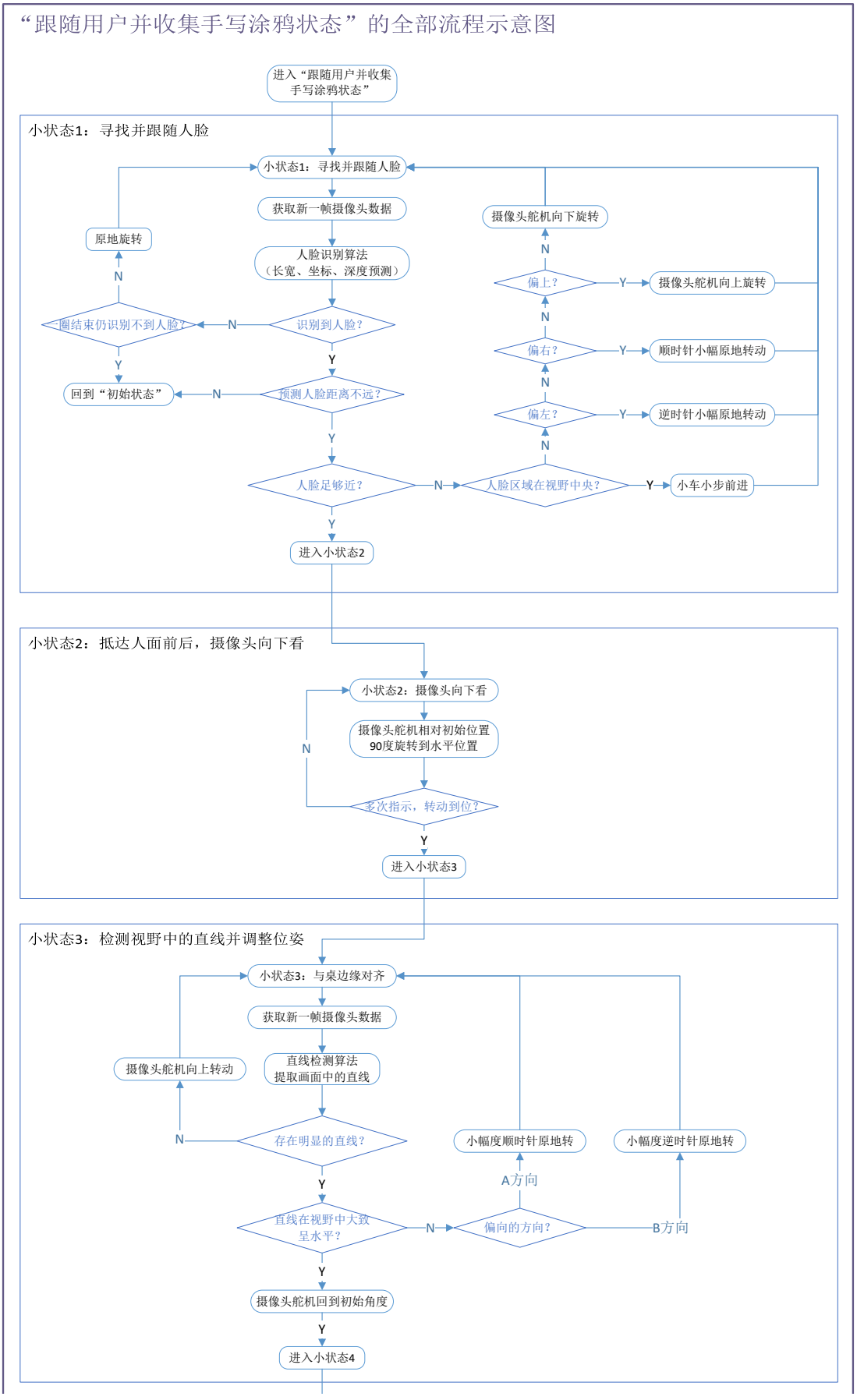


图 24 主链路的运行流程图

对“**跟随用户并收集手写数据**”大状态进行更具体的流程描述，如下图所示——



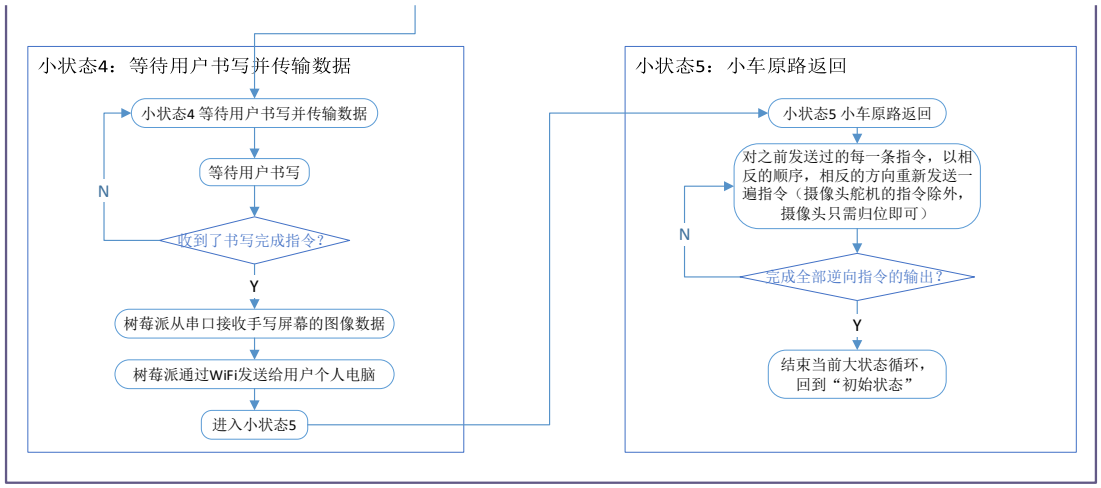


图 25 “跟随用户并收集手写数据”大状态的全部小状态 详细流程

以上的流程图展示了主链路的大致运行流程。由于篇幅有限，难免省略了一些细节，必要的细节会在后续问题解决部分进行具体说明。而更具体的算法设计思路则会在后文算法介绍部分详细说明。

2.4.2 PSoC6 链路的执行流程

该部分的执行流程与本文第一部分介绍的综合实验流程较为相似，笔者在其综合实验的代码基础上拓展了 ① Todo-List 管理 ② 待机图像显示 两大功能。在整体运行流程上，也是采用 BLE、EINK、CapSense、RGB-LED 等模块构建三个 Task 任务进程，进行适当的初始化之后，调用 FreeRTOS 系统进行多任务调度。因此固件运行的整体流程与综合实验是类似的，重复的部分在此不再赘述。针对所不同的部分，在整体流程的示意图上标注如下图所示。

对于各个任务中值得介绍的初始化和循环的流程及细节，将在算法部分一并说明。

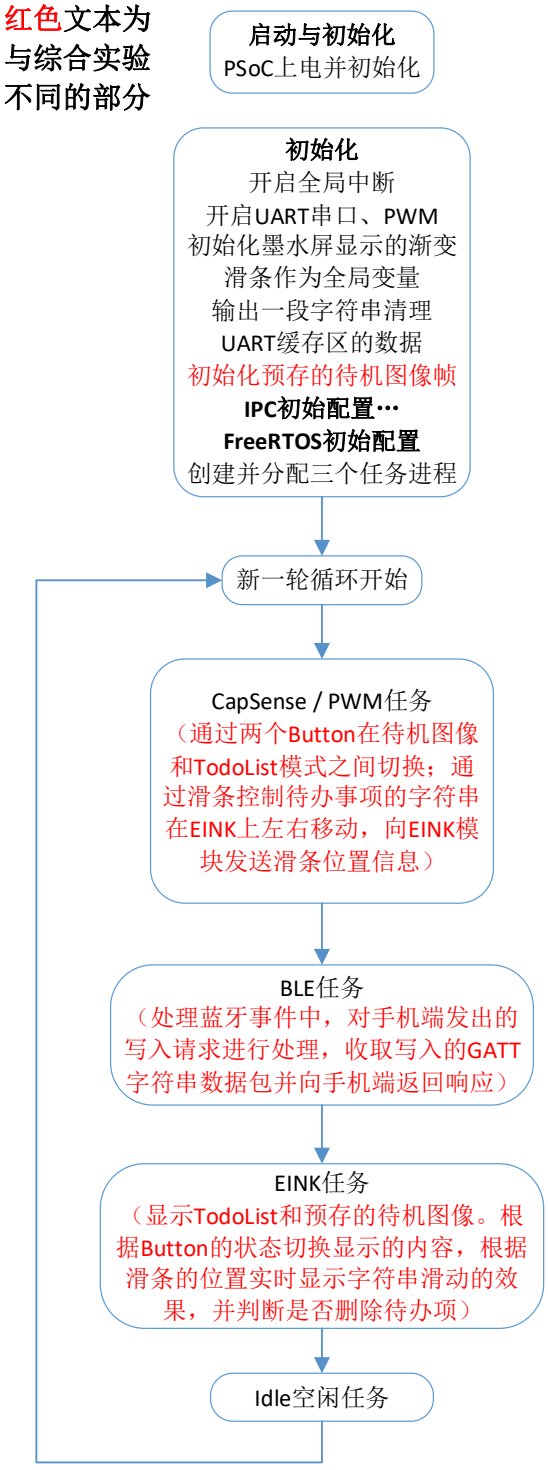


图 26 在创新实验中 PSoC6 程序运行的整体流程图

（任务执行顺序不一定严格按照上图）

2.5 笔者所负责部分的创新点/难点介绍

2.5.1 笔者在创新实验项目中的贡献

在整个桌面辅助小车的创新实验项目中，笔者负责的部分主要是：

【树莓派上的部分】

- 树莓派程序执行的系统配置与编译器、解释器、依赖库等依赖环境的配置
- 树莓派对摄像头视频流数据的请求、接收与解码
- 树莓派中小车运行逻辑的设计与流程框架的搭建（如上文流程图所示的绝大部分内容）
- 小车的识别（面部、手势、直线）与决策算法的设计和开发

【小车的部分】

- 树莓派执行逻辑与相关识别算法在小车上的统一调试
- 小车上可运动摄像头的机械结构设计

【PSoC6 辅助功能的部分】

- PSoC6 全部功能的设计和开发
  - 蓝牙、墨水屏、触摸滑块等基本功能模块的前期开发调试
  - 基于智能手机交互的 Todo List 管理系统的开发
  - 待机图像数据的生成与显示

【摄像头的开发与视频数据传输】

- 摄像头模块的二次开发及其与树莓派之间的 Wi-Fi 视频流传输

【其他事务】

在实验之外，笔者还负责了：

- 小车等部件的选购，并与商家对接相关资讯；
- 统一调试小车的主要功能，并在验收和视频展示中作为演示者；
- 展示 PPT 的整合与修改（包括统一整合组员的内容排版、绘制项目的整体系统结构图等）等。

上面介绍的是主要由笔者负责的部分。这些部分中也不乏组员们的付出，组员提出了一些宝贵的意见和建议，感谢其他两位组员！同样，在本小车项目的其他部分中，笔者也积极帮助组员，尽己所能提供帮助，譬如调试环境与代码、提

出故障解决方案等。因此我们的分工并不是绝对的，所以以上列出的贡献只是对大致分工的描述，特此说明。

2.5.2 笔者负责部分的难点

以下为笔者负责部分的创新点/难点罗列，是笔者个人的主观看法，仅供参考，针对算法的详细介绍可见后文的代码说明。

① 树莓派的系统初始配置与代码运行依赖环境的配置。

笔者此前没有接触过树莓派，虽然接触过 Linux 系统，但是很多 Linux 系统的观念用在树莓派上会带来问题，因为树莓派的性能相比一般的计算机是有很大的局限性的，人们一般用它来做数据通信等技术的嵌入式开发，如果要在树莓派上运行更加智能、复杂的算法，就会触发相当多的问题。首先就是环境配置的问题。

在初始配置时，由于树莓派存储空间有限，我们烧入了轻量级的 Linux 操作系统 Ubuntu-Lite。为了安装其他更大的插件，我们进一步利用 SD 卡的空间，将存储空间的宏扩展到了整块 SD 卡上。

轻量级的操作系统的缺点是没有进一步开发的生态环境，只有图形化界面、浏览网络等基本功能，要想进行我们需要的编程、调试与程序执行等更复杂的任务，就需要手动安装编译器、解释器、批处理工具以及众多底层的高级语言依赖库，涉及的工具如 g++、make、cmake、gettext、gLib、bluez 等等不胜枚举。在配置环境的过程中，常常是安装一个工具中途报错，缺少另一个工具，转手去安装另一个工具，而安装过程中再次报错，缺少又一个工具……轻量级系统的漏洞一环扣一环，需要付出耐心与时间一步步配置。

有时在网上查找到的指令或许并不适用我们的系统，一旦输入运行了，就很可能把系统弄崩溃。笔者在配置时，在安装 Bluez 和 Miniconda 时曾经出现过两次指令错误，导致系统崩溃了两次，只好从头再烧系统并配置，因此这一环节花费了笔者大量的时间。由于上课前接触也较少，因此是较为困难的。

② 摄像头的使用与二次开发

实验室提供的 ESP32-Cam 是一款已经开发好，有完善的使用流程的摄像头组件，但是这并不能满足我们的需求，我们着手进行二次开发。笔者了解到摄像头固定的底板 ESP32 是一种可开发板，通过连接 TTL-USB 转换头上的 RX/TX 接口能够烧入固件代码，因此理论上是可以二次开发的。国内对于这款国产模块的教程并不多，笔者在国外论坛上寻找到了一篇关于 ESP32-Cam 的使用教程，但是也发现了该教程中的许多错误，譬如教程中的接线不适用于笔者的设备。经过一番试错和测试之后，笔者完成了正确的连线，并从网上下载了示例代码，成功烧入了 ESP32，该示例代码涉及 HTML Web 前端网页开发，较为复杂。示例代码能够在局域网本地 IP 端口下设置图形化网页，用于可视化测试视频和相关参数的调整。

通过网络连接协议的学习，笔者了解到，要在树莓派上获取 Wi-Fi 局域网中的视频流数据，需要树莓派连接同一 Wi-Fi 局域网，再使用 socket 工具连接约定好的本地 IP、端口，向其发送 Request，方法为 GET，提取/stream 子文件夹中的二进制数据，根据事先约定好的图像帧数据包的格式来从二进制流中提取出单帧图像。这就是摄像头视频流数据传输给树莓派的总体逻辑。

③ 摄像头传输数据频繁出现的错误、破损、失真的处理

ESP32-Cam 摄像头的性能有限，其传输图像的质量受到外接电源信号波动的影响较大，如果小车在运动，则很容易引起连线的振动从而导致信号的波动，输出的图像就会出现波纹、色彩失真、大面积补丁等破损的现象，经笔者不严格估计，这样的现象在视频流中约有 30%左右的出现概率。

笔者通过设计缓冲时间的方法来减少这样的问题带来的干扰。

④ 摄像头传输数据与树莓派计算速度之间矛盾的调和。

由于摄像头传输数据的速率较快，而树莓派的计算速度较慢，两者的步调容易不一致，而通过 GET 读取的帧数据是前后连续的，所以可能会出现摄像头读取的某一帧数据在延迟了很久之后才被识别的情况。尤其是在识别人脸、手势等复杂的算法时，计算速率就会明显下降，而且树莓派的计算速率受温度影响较大，如果过热，则计算容易减慢，甚至莫名其妙地崩溃。笔

者通过降低视频流接收的帧率来使得两者步调一致。

⑤ 树莓派计算速度与 Arduino 接受并执行指令的过程存在延时这一对矛盾的调和。

树莓派向Arduino发出指令，Arduino接收指令之后，控制设备进行运动，设备接收到某一条指令之后，会运动一段时间，再执行下一个接收到的指令。树莓派虽然计算速度相对较慢，但是相比电机的转动是更快的，所以可能会出现某些指令被跳过的情况。这时也要考虑让两者的步调匹配。笔者也是通过一个变量记录处理的轮数，令程序相隔一定轮数再发送指令。于是解决了相关的问题。

⑥ 如何通过蓝牙 BLE 向 PSoC6 中写入信息。

官方文档、DataSheet、官方示例代码都没有介绍如何在 PSoC6 中通过 BLE 写入信息，仅介绍了一些固有服务的使用方法（如 Heart Rate、Instant Alert等）。因此在信息较为匮乏的情况下，笔者自学了BLE的基本运行原理，了解了 GAP、GATT 传输技术、特征、服务、描述等专业术语，并查阅了相关资料，完成了用户向 PSoC6 中写入字符串数据的流程。

⑦ 如何克服 PSoC6 开发板上 RAM 空间有限，编程时又无法烧入文本文件的问题。

RAM 最多只能同时保存 3 个 5808 字节的图像帧数组，而我们的图像显示任务和 TodoList 任务都各需要两个帧数组。笔者通过以时间换空间的方法（共用变量）来减少对 RAM 资源的占用，同时用了比较巧妙的初始化方法来初始化预存的待机图像，不需要读取外部数据文件。

2.5.3 笔者负责部分的创新点

① 在树莓派上搭载并运行相对复杂的识别算法

正如前文所示，树莓派的计算、存储性能有限，因此很少有人会在树莓派上部署较为复杂的模式识别模型，一般会部署在 PC 上，再通过 PC 回传

数据到控制模块。我们为了尽量增强小车的独立行为能力，减少与外界的有线/无线连接，于是将小车智能处理模块（树莓派）部署在小车的车身上，树莓派体型小巧，适合在小型移动机器人上搭载。为了让识别算法能够在树莓派上相对流畅地运行，我们烧入了轻量级的操作系统，并扩展了存储空间，手动安装了依赖库，并且采用了轻量级的面部识别和手势识别算法库MediaPipe，加快推理预测的速度，最终树莓派的计算速度较为可观，并能控制小车正常运行。

② 在 Python 中通过 Wi-Fi 接收 ESP32-Cam 摄像头向局域网传输的视频流数据。

如前文所述，ESP32Cam 中烧入的实例代码会建立一个本地网站来展示视频数据。笔者希望能获取每一帧的 RGB 数据用于 Python 中的一些识别算法来进一步处理。而 Python 抓取 ESP32Cam 数据流并解码为图像的代码在网络上较少。为了获得摄像头的原始 RGB 数据，笔者尝试自己实现了 Socket 连接与解码的算法，抓取并还原了视频流中的每一帧照片，使得摄像头可用于智能识别。

③ 可运动摄像头这一机械结构的构想和设计

为了让小车能够自主寻找人脸并将人脸放在视野正中心，以及判断是否到了边界且与边界对齐了，笔者利用连杆与舵机设计了可以灵活俯仰控制的摄像头，于是就可以使得摄像头更自主地与环境进行交互。

④ 小车向下看，与桌子边缘对齐的机制与算法设计

有了可运动的摄像头，车子靠近桌子边界时，就可以自动 90 向下转动，观看是否与边界对齐。笔者设计了 Canny 边缘提取、Hough 直线检测、转向决策的一套流程来控制小车的朝向与边界对齐。这一构思来自偶然的灵感。

⑤ PSoC6 显示图像

用 Python 实现了任意格式或尺寸的 RGB 图像一键向 EINK Frame 数组

的数据结构的转换。可以便捷地存储到 PSoC6 中并显示。同时创造性地定义了一种全新的数据初始化方法，避开了烧录程序不能固化外部数据文件的问题。

⑥ Todo List 的滑动动态视觉效果设计

这一灵感来自 Iphone 手机的滑动解锁效果。通过滑动删除每一项待办事务能够给使用者带来更好的使用体验与一定的成就感。

⑦ 实现了伸展的手指数目识别的算法设计

编写算法识别手部，并生成每一个手部关键点的空间坐标数组，考察关节点的相对位置来计算每一根手指是否处于伸展状态。汇总伸展手指的数目即可知用户手势指示的数字。

⑧ 实现了通过挥手控制小车随手的挥动方向左右旋转的代码。笔者实现了该部分代码，在个人计算机上运行流畅，但在树莓派上工作的效果不稳定，因而没有在验收中演示。

⑨ 笔者原先希望在 PSoC6 墨水屏上播放显示摄像头的视频流数据，因此在 Python 中实现了将视频流数据通过蓝牙分包发送的数据包结构设计与代码编写，但是由于在限定时间内未能解决树莓派上 BLE 依赖库的安装问题，最终没能如愿展示。

2.6 笔者所负责部分的软硬件代码说明

为了限制篇幅，代码量又较大，以下笔者只列举关键性的代码块来讲解。

2.6.1 树莓派运行流程中的关键代码说明

2.6.1.1 摄像头视频流的抓取与解码

连接并发送请求，socket 连接使用的 Python 库为 asyncio，支持多任务协同：

```
hostname = "192.168.137.95"
port = 81
path = "/stream"
reader, writer = await asyncio.open_connection(hostname, port)
query = (
    f"GET /stream HTTP/1.0\r\n"
    f"\r\n"
)
print(query)
writer.write(query.encode('latin-1'))
```

分行读取获得的数据流并进行简单的识别，拼凑出完整的图像。

ESP32Cam 上约定的视频流数据包结构如下：

- 起始信号：会先后传来即将传输的总字节数与相应字节数的数据
  总字节数：xx\r\n
  数据：--123456789000000000000987654321 \r\n
- 相关的图像信息（大小、起止时间等）：数据共三行
  总字节数：xx\r\n
  数据：
  XXX：XXX\n
  XXX：XXX\n
  XXX：XXX \r\n
- JPG 格式图像数据：不固定行数，遇到 ascii 码为\n 的地方会分行
  总字节数：xx\r\n
  数据：
  XXXXXXXXXXX\n
  XXXXXXXXXXXXX\n
  ……
  XXXXXXX\r\n

JPG 图像数据传输完之后，会继续开始下一轮读取，重新从起始信号开始。

对 JPG 图像二进制数据的解码，首先是采用 string 类型的 decode 方法来解码为“latin-1”格式的字符串，再将各行的图像数据进行连接，用 Pillow 库的 Image.open(BytesIO(img_binary))来转换为 Image 格式的 jpg 数据，再使用 numpy 库的 np.array(img_jpg)函数来转换为 H*W*3 的 RGB 张量，便于后续处理。

分行读取 stream 流数据包的代码如下，可见，只需要在每一次起始信号到来

时处理前一轮收集到的流数据即可。代码中通过 num 的操作来跳过无用信息数据的读取，通过 img_bin 来拼接各行的图像数据。

```
while True:
    # loop += 1
    line = await reader.readline()
    #print("line: ", line)
    if not line:
        break
    line2check = line.decode('latin1').rstrip()
    if line2check:
        if line2check == "--123456789000000000000987654321":
            num = 0
            img_bin = img_bin[:-2]
            if not yep:
                yep = True
                continue
            img_jpg = Image.open(BytesIO(img_bin))
            img_np = np.array(img_jpg)
            print(TASK)
            index += 1
            if TASK == 3:
                frame_rate = 20
            else:
                frame_rate = 60
            if index % frame_rate == 0:...
            img_bin = b''
            # time.sleep(0.)
            continue
        num += 1
        # print(num, line)
        if line == b'24\r\n':
            continue
        elif num > 5:
            img_bin += line
# Ignore the body, close the socket
writer.close()
```

2.6.1.2 面部识别与面部跟踪的代码说明

调用 MediaPipe 库中的面部识别算法，获取图像中的面部数据。定义一个类用于本任务的面部识别，提供 centre_接口输出面部的中心坐标，width_接口输出面部的宽度（百分比形式）。代码逻辑较为简单，不再赘述。

```
class face_detector():
  def __init__(self):
    self.face_detection = mp_face_detection.FaceDetection(model_selection=0,
                                                          min_detection_confidence=0.5)
    self.results = None

  def detect_(self, image_np):
    self.results = self.face_detection.process(image_np)

  def centre_(self):...

  def width_ (self):...
```

针对树莓派识别决策模块与 Arduino 控制模块之间的通讯，定义如下的通讯协议。

```
# DEFINE          int   send
# forward          1     "a"
# backward         2     "b"
# leftward         3     "c"
# rightward        4     "d"
# left-front       5     "e"
# right-front      6     "f"
# left-behind      7     "g"
# right-behind     8     "h"
#
# clockwise        9     "i"
# anticlockwise    10    "j"
#
# camera_up5       11    "k"
# camera_down5     12    "l"
#
# camera_init            "m"
# camera_90deg           "n"
# clockwise_fast         "o"
# anticlockwise_fast     "p"
```

图 27 小车指令的通讯协议

“初始状态”下寻找面部并发出指令的函数如下所示，对面部的中心坐标设定一个阈值，在阈值内则返回 None，即表示已经完成初始化任务。否则返回顺时针旋转指令，同时也返回一个相反的动作指令，用于必要时归位，后续的代码也是一样，同时输出动作与逆动作。

```
def find_face(face_catcher, img_np, thres):
    face_catcher.detect_(img_np)
    x, y = face_catcher.centre_()
    if not x:
        return "i", "j"
    if not abs(x - 0.5) < thres:  # no face
        # cmd spin
        return "i", "j"
    else:
        return None, None
```

“跟踪用户并收集手写涂鸦数据状态”下，**第一个小状态**（向用户靠近）的识别与决策的算法如下图所示。判断逻辑与上文的流程图基本一致。

需要解释的是，因为我们的小车是通过人脸的宽度来预测距离人脸的远近的，也是通过人脸的宽度来判断是否已经到达合适的位置的。它能够根据人脸的位置来自适应地停在离人脸有着固定距离的位置，使得人的书写体验相对舒适。但是

这并没有考虑人坐的太远，车是否会掉下桌子，所以笔者加入了一个限制条件，即在车子初始状态下观察到的人脸不能太小。可以通过实际调试来改变对应的阈值，如果人脸过小，说明人距离桌子是较远的，此时车子会不断前进，直至掉下桌子。而安全范围内的人脸大小能够保证车子正常情况下不会掉下桌子。阈值通过 width_min 控制。 而 width_max 决定了人脸接近到什么程度可以认为是小车已经到了合适的位置。

```
def go_for_face_status1(face_catcher, img_np, thres, width_min=0.1, width_max=0.4):

    face_catcher.detect_(img_np)
    x, y = face_catcher.centre_()
    w = face_catcher.width_()
    if not x:  # no face
        # cmd spin
        return None, None
    elif w < width_min:
        return None, None
    elif w >= width_max:
        return "OK", None
    elif abs(x - 0.5) < thres and abs(y - 0.5) < thres:  # have face in the centre
        # cmd forward
        return "a", "b"
    else:  # have face not in the centre, should spin or adjust the camera
        if abs(x - 0.5) > thres:  # should spin the car
            if x > 0.5:
                # cmd cw
                return "i", "j"  # todo NEED TO SPECIFY !!!
            else:
                # cmd acw
                return "j", "i"  # todo NEED TO SPECIFY !!!
        if abs(y - 0.5) > thres:  # should adjust the camera
            if y > 0.5:
                # cmd camera up
                return "l", "k"
            else:
                # cmd camera down
                return "k", "l"
```

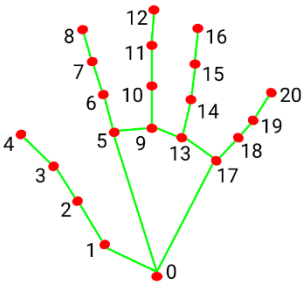
2.6.1.3 手势识别与决策的算法说明

笔者定义了一个 hand_detector 类，用于：① 识别手部并提取关键点，② 计算手部的平均坐标，③ 计算伸展的手指个数。①和②逻辑都较为简单，③的逻辑有必要进行具体的介绍。

笔者调用了 Mediapipe 库中的手部识别算法。速度较快。其 API 的相关参数可以灵活调整，譬如 static_image_mode 定义了是视频输入还是单张图像输入。由于算法对于视频连续帧输入的识别效果更具置信度，因此该参数可以选为视频输入模式。

```
self.hands = mp_hand.Hands(self.static_image_mode,
                           self.max_num_hands,
                           self.model_complexity,
                           self.min_detection_confidence,
                           self.min_tracking_confidence)
```

为了识别手势，首先要介绍手部的 21 个关键点的定义，如下图示。其中 4，8，12，16，20 分别为各个手指的末端 ID。其中手部的关键点提取算法有参考过一位博主（来源见[2]），因此这里不作介绍。



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

图 28 手部关键点的 ID 及定义 图源[3]

为了识别伸展的手指个数，笔者的算法需要对手的摆放位姿进行一定的限制，即要求手是五指朝上，掌心面对摄像头，而且识别的为右手。于是可以开始编写手势识别算法。对于大拇指，4 号末端与 3 号关节之间的水平相对位置在展开和收拢状态下是明确不一样的，由此可以分辨大拇指的伸展状态。对于其余四根手指，在限定条件下，手指末端与往下第二个关节在竖直方向上的相对位置定义了手指是伸展还是收拢。因此遍历每一根手指即可判断出总伸展手指个数，也就能够完成比划数字的手势识别。代码如下所示。

```
# 该函数用于检测伸出来的手指个数
def finger_recog(self, img_np):
    lms_list = self.generate_landmarks(img_np)
    # print(lms_list)
    if lms_list:
        fingers_num = 0
        if lms_list[20][2] < lms_list[18][2]:
            fingers_num += 1
        if lms_list[16][2] < lms_list[14][2]:
            fingers_num += 1
        if lms_list[12][2] < lms_list[10][2]:
            fingers_num += 1
        if lms_list[8][2] < lms_list[6][2]:
            fingers_num += 1
        if lms_list[4][1] > lms_list[3][1]:
            fingers_num += 1
        return fingers_num
    return -1
```

[2] Simon Kiruri: https://www.section.io/engineering-education/creating-a-hand-tracking-module/

[3] Mediapipe: https://google.github.io/mediapipe/images/mobile/hand_landmarks.png

2.6.1.4 直线识别与决策的算法说明

代码如下：

```
def matching_edge_line_status3(img_np, thres=0.16, problem_thres=0.1, loop=0):
    gray = cv2.cvtColor(img_np, cv2.COLOR_RGB2GRAY)
    edges = cv2.Canny(gray, 120, 20, apertureSize=3)
    edges[:5, :] = 0
    lines = cv2.HoughLines(edges, 1, np.pi / 180, 100)
    # print(lines.shape)
    # cv2.imwrite("./img/" + str(loop) + ".jpg", edges)
    # print(loop)
    if lines is not None:
        line = lines[:2, 0, :]
        # print(line)
        angle_avg = line[:, 1].mean()
        if line.shape[0] == 2 and abs(line[0, 1] - line[1, 1]) > problem_thres:
            print("may have recognizing problems!!")
            angle_avg = line[0, 1] \
                if abs(line[0, 1] - 1.5707964) < abs(line[1, 1] - 1.5707964)\
                else line[1, 1]
        if angle_avg - 1.5707964 > thres:
            # todo need check
            return "i", "j"
        elif angle_avg - 1.5707964 < -thres:
            # todo need check
            return "j", "i"
        else:
            return "OK", None
    else:
        print("No line !")
        return None, None
```

笔者先对直线进行灰度化处理，然后使用 Canny 算法（通过适当设置相关阈值参数）提取边缘，之后采用 Hough 算法对边缘图片进行直线检测，其原理是将图像二维平面中的直线 y=ax+b 唯一地映射到对偶空间中的一个点(a, b)，统计直线上边缘点的个数，在对偶空间中边缘点最密集的区域即表明了桌子边缘对应直线的最可能表达式。

通过调整阈值可以调节识别出的直线数目，如果识别出多条直线，为了提高鲁棒性，笔者设计了如下选取边缘直线的算法：

只考虑边缘点最多的前两条提取出的直线，如果两条直线的倾斜角度差距很小，则取平均值再输入决策算法即可，如果差距很大，则表明图像中可能出现了相对明显的其他直线，此时取最接近水平线的一条，因为小车到边时的直线姿态是较为接近水平的。其他直线大多是垂直于水平线的（如人腿）。

边缘提取的例图如下所示，可以看到视野中的直线形态是较为明显的。

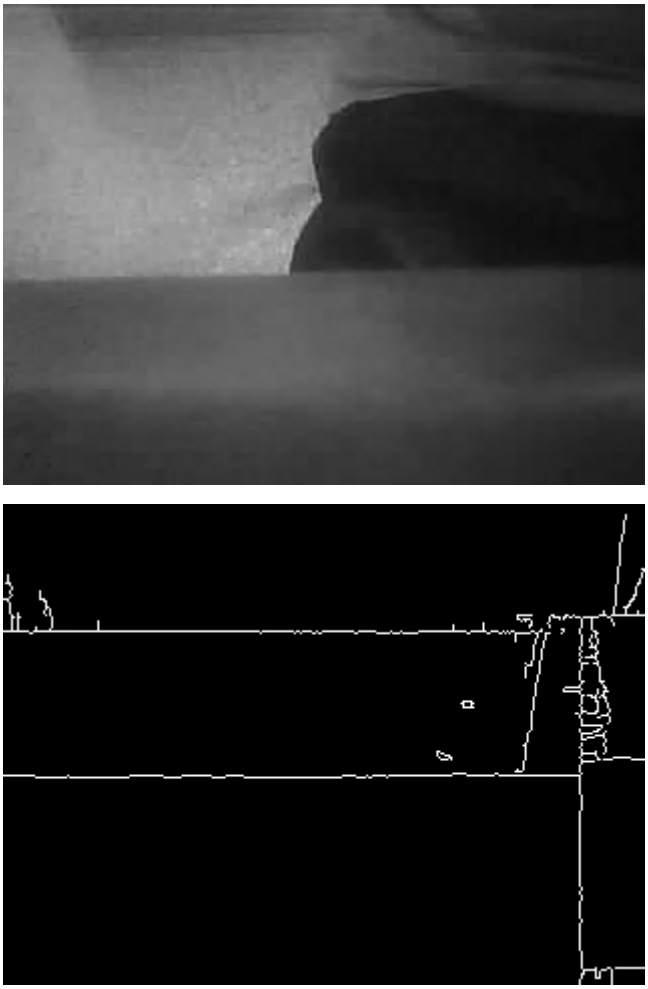


图 29 原图与 Canny 边缘提取的结果

在后续决策中，判断直线的倾斜程度是否接近水平线，如果较为接近，则结束识别与调整状态，认为已经与桌边对齐，否则根据识别的倾斜方向来原地小幅旋转调整位姿。

2.6.1.5 流程中状态转换代码的细节说明

① 通过 TASK 参数来控制大状态的切换：

1 为初始状态

2 为切换状态

3 为原地旋转归位

4 为跟随并收集书写数据

② 通过 status 参数来控制小状态的转换，如“跟随并收集书写数据状态”中的 5 个小状态切换。

③ 为了解决前文提及的树莓派识别速率与摄像头传输频率不匹配的问题，笔者设计了可调节处理帧率的算法：

```
if TASK == 3:
    frame_rate = 20
else:
    frame_rate = 60
if index % frame_rate == 0:...
```

图 30 树莓派处理的帧率切换方法

（三点省略中的语句块为大状态执行的语句）

问题的本质是树莓派处理速率过慢，而导致图像压栈过多，延迟过大，此时引入 frame rate 参数，使程序每 frame rate 张图片才进行一次处理、识别、决策的流程，否则直接跳过该帧图片。而不同任务的运行速率也各不相同，譬如手势识别的速率明显低于直线检测的速率，所以不同任务的 frame rate 设置也应当有所区别，如上图的切换方法所示。引入 frame rate 之后，机器人的识别与摄像头采集数据的同步性就更好了。

④ 为了解决前文提及的树莓派发送指令与小车执行指令速率不匹配的问题，同样引入了类似上一条的方法来使二者同步。但由于帧率调节较小之后，小车已经能够比较实时地对观察到的实际环境作出反应，所以相关的参数的调控力度并不大。

⑤ 对于图像频繁存在的破损问题与识别模型可能发生的错误识别的情况，需要进行必要的去干扰处理。在各个状态和子状态中普遍需要对偶然的干扰进行处理。

以手部的识别为例，由于图像采集和算法本身的局限性，在某些帧中手部目标检测会出现错误，所以不能仅仅是在某一帧出现了手部就对结果直接进行决策和运动控制，可能是手刚放入视野或手只是偶然扫过了视野（用户并没有指示的意图）。也不能因为某一帧没有识别到手部就立即结束识别，很可能是因为算法本身出现了小概率的错误识别或者图像传输出现了信号的波动。

笔者使用的方法类似按键防抖的“滤波”算法，引入变量来构建一个“缓冲阶段”，只有在相邻的若干帧手部图像的识别结果**连续**稳定地输出了相同的预测值，才能最终确定用户要表达的意图，即手势是 0 还是 1,2,3,4,5。中途如果出现了干扰帧，则变量归零，重新开始缓冲阶段；如果中途更换了其他手势，则变量也归零，修改保存的预测值变量，重新进行缓冲阶段。同样，对于面部的识别也

是一样的防干扰技巧。实验表明，引入缓冲阶段能够提高算法识别的稳定性。

手势识别（“旋转归位状态”）中手势识别与决策流程的代码如下所示。通过对变量 finger_endurance(手势连续保持的帧数)和 finger_num_now（当前记录的手势指代的数字）进行操作实现了如上的思路。

```
# task3 hand scanning
elif TASK == 3:
    img = detector.find_hand(img_np)
    finger_num = detector.finger_recog(img_np)
    if not task3_delaying:
        # print(end_time)
        if hand_show_up:
            if finger_num == -1:
                end_time += 1
                if end_thres == end_time:
                    end_time = 0
                    hand_show_up = False
                    finger_num_now = 5
            else:
                end_time = 0
        if not hand_show_up and finger_num != -1:
            if finger_num != finger_num_now:
                finger_num_now = finger_num
                finger_endurance = 0
            elif finger_num != 5:
                finger_endurance += 1
                if finger_endurance == threshold:
                    finger_endurance = 0
                    finger_num_now = 5
                    print(finger_num)
                    # go to given task id
                    hand_show_up = True
                    if finger_num == 0:
                        direction = "o"
                        inv_direction = "p"
                    else:
                        direction = "p"
                        inv_direction = "o"
                    print(direction)
                    ser.write(str2bin(direction))
                    time.sleep(SLEEP_AFTER_WRT)
                    task3_delaying = True
```

2.6.1.6 代码实现小车随手的挥动方向转动的效果

该部分实现并且在 PC 上测试过，但因为在树莓派上运行并不稳定，故没有在验收中演示。通过识别手部关键点的平均横坐标值，来判断手的总体方位，记录相邻 4 帧的手部方位，对手部离开视野前的四次结果的相对关系进行判断，即可预测出人手的运动方向。

```
img = detector.findHands(img_np)
hor = detector.horizon_global(img_np)  # if no hand return None
if not task3_delaying:
    if hor:  # 有手
        if hand_quit_time:
            hand_quit_time = 0
        if hand_quit:
            hand_quit = False
        if not hand_on:  # 手未准备好
            hand_wait_time += 1
            if hand_wait_time_thres == hand_wait_time:
                hand_wait_time = 0
                hand_on = True  # 手准备好了
        if hand_on:
            # if hand_buffer_hor:
            #     if abs(hand_buffer_hor - hor) > thres_v:
            #         print(hand_buffer_hor - hor > 0)
            #         hand_buffer_hor = None
            #         hand_on = False
            hand_buffer_hor_3 = hand_buffer_hor_2
            hand_buffer_hor_2 = hand_buffer_hor_1
            hand_buffer_hor_1 = hand_buffer_hor
            hand_buffer_hor = hor
    else:  # 没手
        if hand_wait_time:
            hand_wait_time = 0
        if not hand_quit:
            hand_quit_time += 1
            if hand_quit_time == hand_quit_thres:
                hand_quit_time = 0
                hand_quit = True
        if hand_on and hand_quit:
            hand_on = False
            if hand_buffer_hor_3:
                if hand_buffer_hor + hand_buffer_hor_1 > hand_buffer_hor_2 + hand_buffer_hor_3:
                    direction = "o"

                    inv_direction = "p"
                else:
                    direction = "p"
                    inv_direction = "o"
                print(direction)
                ser.write(str2bin(direction))
                time.sleep(SLEEP_AFTER_WRT)
                task3_delaying = True
```

2.6.1.7 小车原地返回的运动代码

```
# todo: all the historic actions reversely cmd
if way_num + len(way_memory) > 0:
    way_num -= 1
    print(way_num, len(way_memory) )
    print(way_memory)
    ser.write(str2bin(way_memory[way_num]))
    time.sleep(SLEEP_AFTER_WRT_4)
else:
    status = 1
    way_num = -1
    way_memory = []
    reverse = False
    # 退出这个任务
    TASK = ORIGINAL_TASK
```

通过在此前的状态中维护一个列表来记录每一个发出指令的逆向指令，之后

通过如上代码反向遍历每一个逆向指令，即可实现小车归位的效果。

2.6.2 PSoC6 运行过程中的关键代码说明

2.6.2.1 实现 TodoList 管理与字符串滑动动态视觉效果的算法

TodoList 的创建：采用 BLE 从手机端输入字符串，维护若干个存储 TodoList 字符串数组的变量，分条打印并实时显示在墨水屏上。

TodoList 的删除：感应 CapSense 的方位，如果在大于 85%的位置上，则删除当前首条 TodoList 项，并更新屏幕显示内容。

字符串滑动动态视觉效果的实现方法类似笔者综合实验中的亮度光标的算法。对于 CapSense 的位置变化，不断据此修改首条 TodoList 项的打印起点并刷新屏幕，当刷新频率较高时，人眼就会观察到 TodoList 项随手指的左右移动而移动的效果，延迟很小，效果较好。

相关代码片段如下所示，具体可见代码中注释。

```
else { // Slider Changed
    common_bri = bri;
    Cor_J[0] = (int)(16 * (double)(bri) / 100); // refresh first item's head position
    if (bri > 85 && this_idx > 0) { // delete the first item
        for (int i = 0; i < this_idx - 1; i++){
            memset(todo_list[i], 0, 30 * sizeof(char));
            memcpy(todo_list[i], todo_list[i + 1], 30 * sizeof(char));
        }
        memset(todo_list[this_idx - 1], 0, 30 * sizeof(char));
        this_idx--;
        pair_taken--;
    }
}
memset(Frame[0], 255, CY_EINK_FRAME_SIZE*sizeof(cy_eink_frame_t));
Cy_EINK_TextToFrameBuffer(Frame[0], Text_Conn, CY_EINK_FONT_8X12BLACK, TextCor);
switch (this_idx){ // print all the items and the number of items matters how much to print
    case 0: {
        break;
    }
    case 1: {
        Cy_EINK_TextToFrameBuffer(Frame[0], todo_list[0], CY_EINK_FONT_16X16BLACK, Cor_J);
        break;
    }
    case 2: {
        Cy_EINK_TextToFrameBuffer(Frame[0], todo_list[0], CY_EINK_FONT_16X16BLACK, Cor_J);
        Cy_EINK_TextToFrameBuffer(Frame[0], todo_list[1], CY_EINK_FONT_16X16BLACK, Cor_1);
        break;
    }
    case 3: {
        Cy_EINK_TextToFrameBuffer(Frame[0], todo_list[0], CY_EINK_FONT_16X16BLACK, Cor_J);
        Cy_EINK_TextToFrameBuffer(Frame[0], todo_list[1], CY_EINK_FONT_16X16BLACK, Cor_1);
        Cy_EINK_TextToFrameBuffer(Frame[0], todo_list[2], CY_EINK_FONT_16X16BLACK, Cor_2);
        break;
    }
    default: {
        Cy_EINK_TextToFrameBuffer(Frame[0], todo_list[0], CY_EINK_FONT_16X16BLACK, Cor_J);
        Cy_EINK_TextToFrameBuffer(Frame[0], todo_list[1], CY_EINK_FONT_16X16BLACK, Cor_1);
        Cy_EINK_TextToFrameBuffer(Frame[0], todo_list[2], CY_EINK_FONT_16X16BLACK, Cor_2);
        Cy_EINK_TextToFrameBuffer(Frame[0], todo_list[3], CY_EINK_FONT_16X16BLACK, Cor_3);
        break;
    }
}
```

2.6.2.2 实现手机端 BLE 写入 PSoC6 的算法

在缺乏示例代码等资料的情况下，笔者自学了 BLE 的基础知识，实现了从手机中写入 BLE 数据库的功能。

要从手机中写入 PSoC BLE DB，我们需要完成如下的流程：

① 手机端连接完毕之后，进入相关服务的特征输入接口，输入 TodoList 字符串，点击发送。

② PSoC 端处理蓝牙堆栈，生成了一个新的事件：

CY_BLE_EVT_GATTS_WRITE_REQ

判断该事件的特征句柄是否是 Todo List，再对该事件进行处理。

③ 先获取一同输入的参数值，保存到 RAM 中，用于 TodoList 的更新。

④ 之后将参数值保存至 BLE GATT 数据库中，方便之后访问读取。

相关 API 为 Cy_BLE_GATTS_WriteAttributeValuePeer

输出相关错误信息（如有）

⑤ 【重要】向手机端发送对于写入请求的响应。相关 API 为：

Cy_BLE_GATTS_WriteRsp(appConnHandle)

⑥ 在任务函数中更新 Todo List，于是完成了一次写入的操作。

蓝牙事件处理流程的代码如下：

```
case CY_BLE_EVT_GATTS_WRITE_REQ:
    if ((*((cy_stc_ble_gatt_write_param_t*)eventParam)).handleValPair.attrHandle
                                        == CY_BLE_TODOLIST_MESSAGES_CHAR_HANDLE)
    {
        pair = (*((cy_stc_ble_gatt_write_param_t*)eventParam)).handleValPair;
        pair_taken ++;
        cy_en_ble_gatt_err_code_t gatt_err_msg;
        cy_stc_ble_gatt_write_param_t *writePtr = (cy_stc_ble_gatt_write_param_t *)eventParam;
        gatt_err_msg = Cy_BLE_GATTS_WriteAttributeValuePeer(&writePtr->connHandle,
                                                            &writePtr->handleValPair);
        if(gatt_err_msg != CY_BLE_GATT_ERR_NONE)
        {
            cy_stc_ble_gatt_err_info_t err_msg ={
                .opCode = CY_BLE_GATT_WRITE_REQ,
                .attrHandle = writePtr->handleValPair.attrHandle,
                .errorCode = gatt_err_msg
            };
            Cy_BLE_GATTS_SendErrorRsp(&writePtr->connHandle, &err_msg);
        }
        // very IMPORTANT !!!
        cy_en_ble_api_result_t api_rsp = Cy_BLE_GATTS_WriteRsp(appConnHandle);
        printf("%d", pair.value.len);
        printf("GOT: %s \n", pair.value.val);
        int i = 1;
        printf("%d", i);
        printf("done!!!");
    }
```

2.6.2.3 图像显示的算法

在 PSoC6 EINK 墨水屏上像素控制的细节已经在本文的第一部分详细阐述。其基本原理是以每八块的 01 像素值组合成一个字节的整数来存储的。于是就可以控制该整数的值来控制相邻八块的像素值。

先将任意尺寸 RGB 图像输入如下笔者编写的代码，先进行 resize 伸缩为适合墨水屏显示的尺寸 176*264，之后转化为灰度图像，再调用 threshold 函数转化为二值图像，阈值为 128。之后将值为 0/255 的项分别转化为 0/1 二进制数列，再将相邻的每 8 位输入笔者编写的 tobyte 函数，转化为 uint8 整数值，最后以代码的格式写入 frame.txt 文件。代码如下图所示。

```python
def wrote_frame_buffer_to_file(img):
    img2 = cv2.resize(img, (176, 264), interpolation=cv2.INTER_CUBIC)
    img2 = cv2.cvtColor(img2, cv2.COLOR_RGB2GRAY)
    img2 = cv2.threshold(img2, 128, 255, cv2.THRESH_BINARY)[1]
    img_bin = (img2 == 255).tobytes()
    img_byte = b''
    for i in range(5808):
        img_byte = tobyte(img_bin[i * 8: i * 8 + 8])
        with open("frame.txt", "w") as f:
            f.write("Frame_x[" + str(i) + "] = " + str(img_byte[0]) + "; \n")


def tobyte(bin):
    all = 0
    for i in range(8):
        all += bin[i] * (2 ** (7 - i))
    return all.to_bytes(1, "big")
```

将生成的代码串复制粘贴在 PSoC 的代码文件中，如下图所示，能够完成图像帧的初始化。之所以这么做，是因为笔者经尝试发现没有办法实现 PSoC Creator 向硬件烧入外部的文本文件或者图像文件，所以无法在板子内部进行打开文件并读写的操作。于是笔者突然想到可以转换成初始化的代码直接对每一项进行初始化，于是就完成了上图所示的代码。

```
void init_Frame_Buffer(cy_eink_frame_t* Frame_x){
    Frame_x[0]=255;
    Frame_x[1]=255;
    Frame_x[2]=255;

……

    Frame_x[5805]=255;
    Frame_x[5806]=255;
    Frame_x[5807]=255;
}
```

显示的效果如下，可以看到图像的显示大致是较为理想的。

图 31 图像显示效果（帆船）

2.7 实验与演示结果

经过笔者的不懈调试与组员们的共同努力，前文列出的难点得到克服，小车终于能够正常执行我们预想的任务。当然，由于时间有限，在系统中也仍旧存在一些暂时未解决的问题可待改进，将在之后的部分介绍。

图 32 滑动删除的演示效果

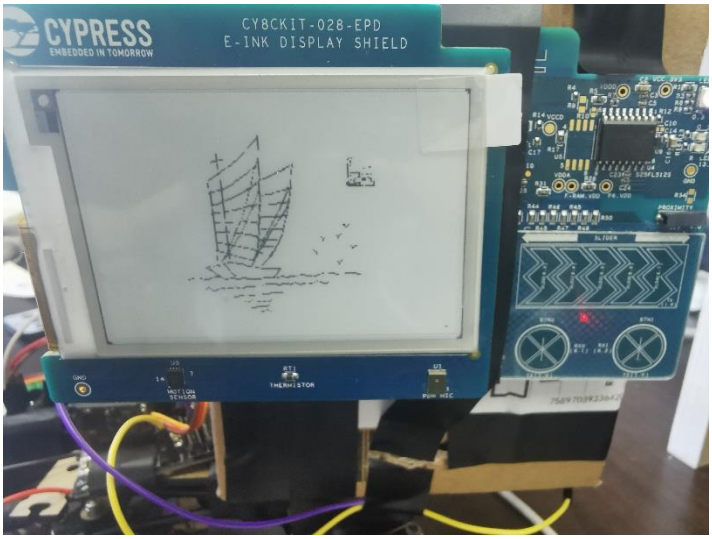


图 33 切换为图像显示模式

小车的演示视频可在云盘上查看：

https://cloud.tsinghua.edu.cn/d/a9f029c7353240f0918a/

实现结果中可见，我们完成了大部分实验前预想的功能效果。

2.8 遇到的问题与解决方案

遇到的问题数不胜数，过去的半个月我们几乎每时每刻都在遇到问题并尝试解决问题。难以在有限的文章中全部囊括。

笔者负责部分的最主要的问题及其解决方案大多已在前文“**难点/创新点说明**”和“**代码说明**”中介绍过，就不再重复说明。以下另外列举一部分其他的困难与解决方案。

2.8.1 PSoC6 存储空间有限

PSoC6 的 RAM 存储空间有限，而在模式切换中，两个模式各需要 2 个帧数组来存放刷新前后的帧像素信息。但是 RAM 难以存下 4 张 5808 字节的数组，于是笔者采用了共用数组的方法来解决该问题。具体方法是在需要用到帧数组时重新打印需要显示的像素数据。

2.8.2 自行购置小车的机械结构较小，无法放下多个模块

为了适应桌面上有限的空间，我们在网络上购置了小车的框架。但是小车的

机械结构已经固定化：其机械结构较小，无法放下树莓派、两块 Arduino、显示屏、PSoC6 板等。于是我们将结构进行了重新设计、改装，并加装了可运动的摄像头，用纸板箱来制作一个类似收纳盒的装置拓展小车的收纳空间，可放下更多的核心部件。并且能够在侧面安装 PSoC6 以方便展示墨水屏，亦可以安装 LCD 触摸屏用于采集笔记。使得小车车体结构紧凑有序。

2.8.3 对于树莓派必须要连接显示屏、键、鼠才能运行程序的问题

对于树莓派必须要连接显示屏才能运行程序的问题，笔者编写了系统脚本使得树莓派能够在开机之后自动执行笔者定义的指令和程序，如蓝牙自动连接、WiFi 自动连接、自动授予接口访问权限、为了避免输入密码，笔者同时也取消了开机密码输入的环节 (编辑修改相关的系统文件)。

如下代码所示：

```
# nmcli device wifi connect TPLinkZZC password 00000000
# nmcli connection up TPLinkZZC
# sudo rfcomm bind 0 00:22:01:00:25:85
# cd /dev
# sudo chmod +777 rfcomm0
# sudo chmod +777 ttyACM0
# cd /home/ww/pro
# sudo chmod +777 main.py
# python main.py
```

2.8.4 Python 中二进制数据类型与实际数据之间的转换问题

对于 JPG RGB 图像，需要将二进制与图像数据之间互相转化；对于指令，需要在二进制与字符数据之间互相转化；对于 PSoC EINK 墨水屏的显示，则需要在二进制位与十进制整数之间互相转化……其中会用到一些诸如 ByteIO(binary_img)、str.encode(“utf-8”)、bytes.decode(“utf-8”)、integer.to_bytes()等等接口，应当多搜索、多尝试，善用 Python Console、Jupter 等在线编程的工具来进行接口测试。

2.8.5 ESP32Cam 的资料缺乏问题

ESP32Cam 芯片的官网与助教提供的资料相对较少，因此笔者尝试广泛寻找

其使用的经验或教程。网络上的资料依然较少，但最终笔者还是在国外的某个论坛上查找到了一些相对详细的 ESP32Cam 技术教程，以及 ESP32 开发板本身的教程。具体的网站可见页末。[4][5] 需要说明的是，这些网站上也存在并非正确或并非适用于笔者所测试的设备的内容，所以应当客观地看待教程。

2.8.6 树莓派计算速率随处理器温度升高而显著减小

风扇散热能力有限，相关的智能识别算法耗能又较大，因此运行一段时间之后，树莓派温度升高，由于处理器内置的自我保护机制，运算速率会显著下降，以至于接近 10 秒才发出一次指令，且与摄像头采集数据的时间点之间出现了较大的延迟。

暂时的处理方法是，每运行一段时间，闲置待机冷却一段时间再重新调试，但是调试效率会较低。

2.9 创新实验总结

2.9.1 仍需改进的点

1. 小车判断并识别桌子边缘的方法有待改进。笔者原本想的是用红外或超声装置来向下检测是否已到边界，但是这需要从小车上引出一条杆件来试探前方的边界情况，可能不太方便，所以实际上还是用人脸的远近预测+初始状态下的预判等方法来避免小车掉下桌子。其实是很粗糙且局限的，所以未来可以进一步修改。另外，检测桌子边缘直线斜率的方法也相对粗糙，如果要改进，可以采用分辨率更高的摄像头、计算能力更强大的模块来识别直线。但是对于本项目的预期功能，这些识别算法已经能够满足基本需求。

2. 摄像头的运动自由度可以更高。实验中只加入了一个自由度——俯仰角的控制，如果希望达成机器与环境的更加智能的交互，可以再加入一个自由度。，用于查看左右的情况。

3. 手势的识别具有一定先决条件的限制，即只能识别五指朝上、掌心朝前的右手手势。未来如果使用更好的计算模块，可以引入更复杂的识别模型来识别更

[4] Randomnerdtutorials: https://randomnerdtutorials.com/esp32-cam-save-picture-firebase-storage/

[5] Espressif: https://docs.espressif.com/projects/arduino-esp32/en/latest/index.html

复杂的手势情况。但在本项目中，为了能够在树莓派中流畅运行智能算法，笔者选择使用了相对快速的算法，也达到了预期的效果。

4. 树莓派的运算速度随温度变化较大，可以考虑设计散热装置，或采用性能更强、体积又不大的计算模块来代替树莓派。

2.9.2 体会

三周的课程与笔者预想的一样，非常充实，笔者担心我们的项目不能达到预想的效果，因此每天都会和伙伴在实验室认真学习并动手实验。在此感谢我的队友王皓霖、贺一非，向我热心提供了很多帮助。

1. 技术能力的提升。总结来说，我们从最底层的摄像头、显示屏、存储器、总线、串口、蓝牙、局域网等技术原理开始一步步学习，然后一步步搭建，一步步测试，解决一个又一个问题，最终搭建出来的小车总体达到了笔者想象的效果，但是笔者也深知有很多不完满之处需要改进。笔者明白，这是一门提升能力的课程，目标是在有限的时间里尽可能多的学习并应用新知识。笔者在这一项目中饶有兴趣地探索了不同方面的知识，并获得了很大程度的技术经验的提升与知识面的拓展，这一点就已经让自己感到足够喜悦了。

2. 解决问题与独立开发能力的提升。在课程中，我们没有依赖已有的设备和已有的完善的资料库，而是自己购置价格合理的设备，并进行改装，实现我们需要的功能，这一过程也大大提升了我们独立开发的能力。只是运气略有不佳，购买的设备频频出现问题，重新购买并安装零件也投入了不少精力。但从积极的角度看，这一过程也锻炼了我们解决问题的能力和抵抗挫折的能力。

3. 团队协作能力的提升。在团队合作之中，我们三人各司其职，从综合实验开始，我们就分别分担了通信、识别、传感三个侧重方向的学习任务，并在后续的创新试验中都展现了自己学习到的知识，同时在交流中也加快了对其他方面知识的学习，使得我们每一个人都能够对通信、识别、传感等方面的电子技术有更全面的了解。我们在课前课后的联系一直都互通有无，整体体验非常愉快。

这一篇实验报告即将抵达尾声，尽管笔者尽量精炼语言，减少不必要的内容输出，但篇幅仍然显得过长了。在此衷心感谢叶老师、赵老师的耐心批阅与指导！

2.9.3 小建议

课程体验非常好，笔者会向学弟学妹积极推荐本课程！另外有一个小建议：如果可能的话，老师可以提醒一下学生，如果要自己购买项目的主体器材或设备（如小车、飞行器等主体），最好要尽量结合实际需求，对产品调研清楚了再购买，没有特殊需求的话使用实验室提供的设备或许会更好。因为我们购买的小车出现了很多的问题（机械结构、零部件、运动控制等方面），虽然解决其中的一些问题很锻炼人，但也有很多非常麻烦的故障（比如导线太短无法挪动、接头不牢固需要手动焊接、驱动质量较差需要重新购买延迟工作进程等等），会比较头疼，所以我们认为项目主体如果能用实验室的器材会更好。而其他非主要的零部件则可以再根据需求来收集或购买。这是我们上完课之后的感想与小建议。

2022 年 7 月 19 日

手册完善时间：2023年12月19日