

四川大學



嵌入式系统设计自主项目 实验报告

实验项目名称：“新型宠物”——基于平衡车的智能环境感知系统

目录

一. 实验应用场景	4
二. 功能模块描述	4
三. 硬件连接方式	4
四. 硬件模块描述	6
1. 陀螺仪	6
2. 电机控制	6
3. 蓝牙远程控制	7
4. 超声波测距	7
5. 传感器采集	8
6. 按键控制 OLED 显示	8
五. 程序流程图	9
1. 系统总体流程图	9
2. 核心板块流程图	9
2.1 陀螺仪	9
2.2 电机驱动	10
2.3 蓝牙远程控制	10
2.4 超声波测距	10
2.5 传感器采集	11
2.6 按键控制 OLED 显示	11
六. 核心代码简述	11
1. 陀螺仪	11
2. 电机控制	16
3. 蓝牙远程控制	18
4. 超声波测距	19
5. 传感器采集	21
5.1 温湿度传感器	22
5.2 烟雾传感器	23
5.3 空气质量传感器	23
5.4 声音检测传感器	24
6. 按键控制 OLED 显示	25
七. 系统创新	28
1. 蓝牙远程控制	28
2. 陀螺仪 DMP 算法姿态融合	28

3. 电机 PID 算法控制.....	29
4. OLED 汉字取模显示	30
5. 自主 PCB 打板.....	31
八. 实验心得	32

一. 实验应用场景

在智能家居监控领域，两轮平衡车以其多功能性和创新性，为用户提供了全面的家居环境监控解决方案。

首先，平衡车作为一个移动监控单元，能够实时检测家中的温湿度、空气质量、烟雾浓度等关键环境参数，并将这些数据传输到用户的智能设备上，使用户能够随时随地掌握家中的环境状况。这种环境感知与交互的能力得益于平衡车上配备的超声波模块，使其能够在室内自由移动并智能避开障碍物，从而在不同区域进行环境数据的采集，为用户提供全面的家居环境信息。

其次，平衡车增强了远程控制与自动化的功能。用户可以通过手机应用远程控制平衡车的移动，实现对家中环境的定点监测或特定区域的深入分析。在安全预警方面，平衡车通过检测烟雾浓度，成为家庭火灾预警系统的重要组成部分，能够在发现异常时立即通知用户，从而提高家庭安全。同时，它还能通过环境声音强度的检测，评估家中或办公室的噪音水平，为用户提供噪音污染的数据支持。

最后，在教育与娱乐方面，平衡车的趣味功能可以作为教育工具，向儿童介绍编程和机器人技术，同时也能作为家庭娱乐的一部分，与家庭成员互动。作为一个多功能集成平台，平衡车未来可以扩展更多的传感器和功能，例如通过摄像头进行视觉识别，或者集成语音识别模块，使其更加智能化，满足家庭多样化的需求。

二. 功能模块描述

这款基于平衡车的智能环境感知系统，集成了多种传感器和控制模块，具备丰富的功能。

1. **陀螺仪姿态检测：** 它通过 MPU6050 传感器实现，确保小车在行进过程中的平衡。
2. **PID 电机控制：** 小车通过精确的 PWM 信号控制左轮和右轮电机的转动，无论是速度还是方向，都能得到有效控制。编码器的反馈机制，为小车提供了精确的速度和位置信息，进一步增强了运动控制的准确性。
3. **超声波测距：** 让小车能够测量与障碍物的距离，实现智能避障。
4. **蓝牙远程控制：** 使得小车可以通过手机等蓝牙设备进行遥控，增加了操作的灵活性。
5. **智能环境检测：** 小车能够监测温湿度、空气质量、声音强度以及烟雾浓度，为室内环境的全面监控提供了可能。
6. **串口打印调试：** 小车可以实现串口打印显示，便于调试和状态监控。
7. **按键交互 OLED 显示：** 使得小车能够直观地展示各种信息，如环境参数和系统状态。

总体而言，这款平衡小车是一个集自主导航、环境监测、用户交互于一体的智能移动平台。

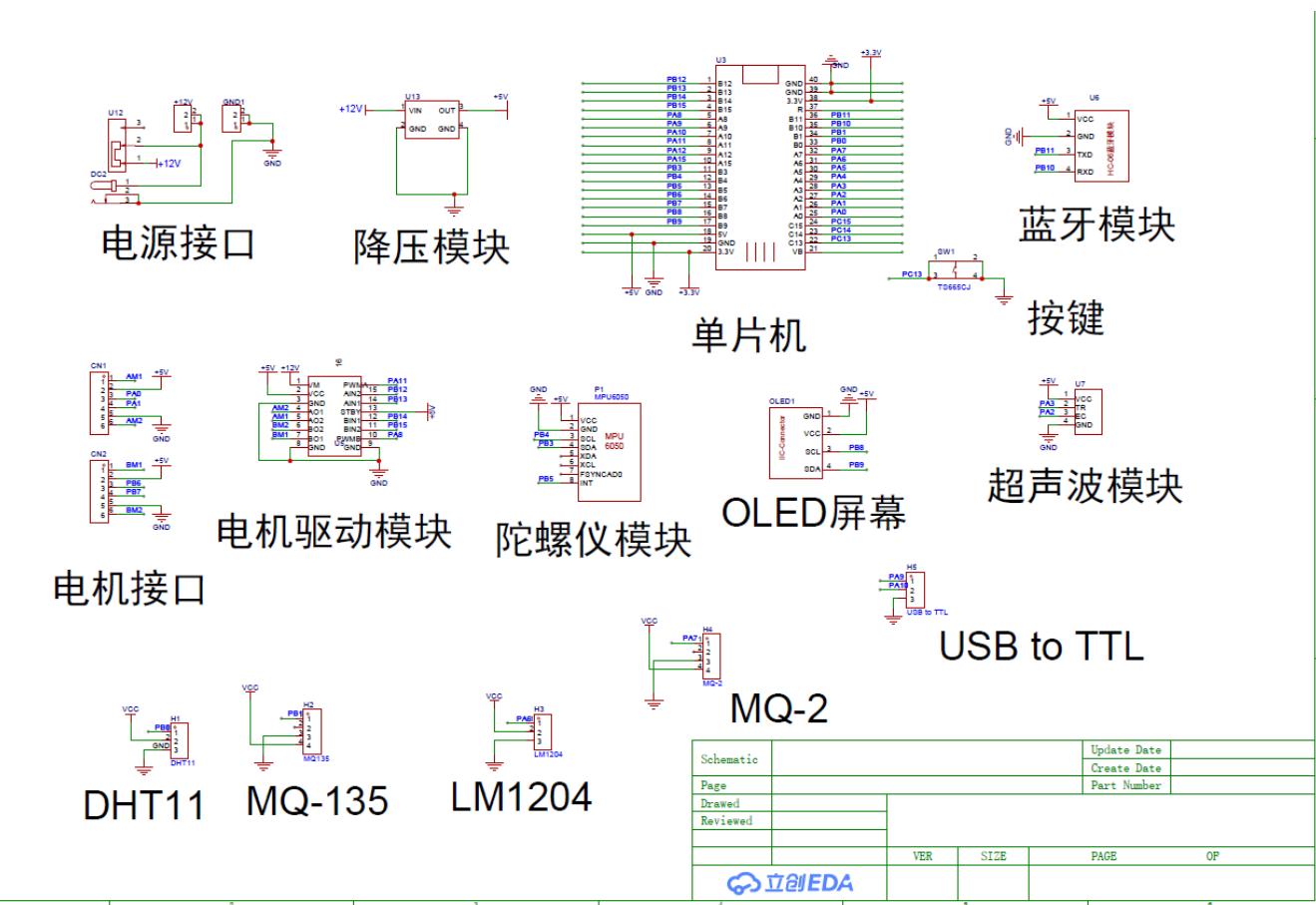
三. 硬件连接方式

本组基于平衡车的智能环境感知系统的引脚分配图、PCB 打版图展示如下。

第十五组平衡小车引脚分配图

IO编号	资源说明	外设	备注
PB3 PB4 PB5	GPIO	MPU6050	软件模拟IIC
PB10 PB11	USART3	蓝牙模块	遥控小车，使用中断
PA2	TIM2_CH3	超声波模块	捕获超声波回波脉冲，计算距离
PA3	TIM2_CH4	超声波模块	给超声波模块发送开始信号
PB12 PB13 PB15 PB14	GPIO	左右轮电机	电机控制信号：电机方向
PA8 PA11	TIM1_CH3、CH4	左右轮电机	电机控制信号：PWM大小
PA0 PA1	TIM2_CH1、CH2	左轮编码器	编码器采集
PB6 PB7	TIM4_CH1、CH2	右轮编码器	编码器采集
PB0	GPIO	温湿度传感器 DHT11	内部ADC转换
PB1	ADC1_CH9	空气质量传感器 MQ-135	ADC采集
PA6	ADC1_CH6	声音强度传感器 LM1204	ADC采集
PA7	ADC1_CH7	烟雾传感器 MQ-2	ADC采集
A9、A10	USART1	MicroUSB	串口打印显示
PB8 PB9	IIC GPIO	OLED显示屏	复用硬件IIC
PC13	GPIO	按键	按键控制OLED显示屏翻页

图表 1 引脚分配图



图表 2 PCB 打版图

四. 硬件模块描述

本实验系统主要包括 mpu6050 陀螺仪、25GA370 带编码器的电机、TB6612 电机驱动模块、MP1584EN 降压模块、HC-05 蓝牙模块、HC-SR04 超声波模块、OLED 显示屏、四脚按键、DHT11 温湿度传感器、MQ-135 空气质量传感器、MQ-2 烟雾传感器、LM1204 声音强度传感器这 12 个模块组成，除此外还有 12V 锂电池、开关模块及电源充电模块。

本实验对时钟、GPIO、USART、IIC、定时器 TIM、ADC、中断、DMA 这八项实验内容均有应用。典型应用内容如下：

1. 陀螺仪

基于 STM32F013C8T6 和 MPU6050 陀螺仪模块，实现：①解析小车在 x y z 轴上的角速度，对小车进行直立环和转向环控制②计算小车在 x y z 轴上的角度，并显示在 OLED 显示屏上

功能应用：

1) 时钟

系统初始化时配置系统时钟为 72MHz，保证外设的正确运行。

2) GPIO

将 PB5 配置为上拉输入，使外部数据传入传感器。

3) 外部中断

开启 PB5 外部中断。当陀螺仪传感器检测到快速运动或方向变化，可以实时响应。

4) 软件 IIC 协议读写

使用软件模拟 IIC 协议来控制 MPU6050，以达到向指定的寄存器连续读或写数据，得到陀螺仪所测得在 x y z 轴上的角度值及加速度值。

5) 硬件 IIC 显示 OLED

将陀螺仪模块读取到的角度显示在 OLED 显示屏上。

2. 电机控制

基于 STM32F013C8T6 和 25GA370 带编码器的电机、TB6612 电机驱动模块，实现：①通过 PWM 信号和转向控制信号对左右轮电机进行精确控制。②PID 控制算法根据实时姿态数据计算出误差，控制调节 PWM 信号和转向信号来调整电机的转速和方向，确保小车能够稳定运行、维持平衡。

功能应用：

1) 时钟

系统初始化时配置系统时钟为 72MHz，保证外设的正确运行。

2) GPIO

启用 GPIO 时钟。将 PB12、PB13、PB14 和 PB15 引脚设置为推挽输出模式，通过 GPIO_Init 初始化这些引脚的输出功能，设置为 50MHz 的输出速度。配置 PA8 和 PA11 为复用推挽输出模式，用于输出 TIM1 的 PWM 信号控制 motorA 和 motorB，即左轮、右轮的控制信号。

3) 定时器中断

配置定时器 TIM1，计数并产生 PWM 信号，通过改变占空比调节电机接收到的电压来控制电机的速度。

4) ADC 模数转换

通过编码器从 MPU6050 传感器的模拟引脚读取电压值并转换为数字信号，以获取角度、加速度、角速度等数据，控制电机的输出信号。

3. 蓝牙远程控制

基于 STM32F103C8T6 微控制器和蓝牙模块 HC05，实现：①串口接收蓝牙数据；②根据蓝牙接收数据控制平衡车运动方向

功能应用：

1) 时钟

系统初始化时配置系统时钟为 72MHz，保证外设的正确运行。

2) GPIO

配置 PB10 为推挽输出，PB11 为浮空输入。

3) 串口中断

配置串口波特率为 9600，开启全双工模式，实现同时收发的功能。若接收中断开启，则小车根据接收信号进行移动。

4) USART

数据发送：平衡车系统通过 USART3 向手机蓝牙发送字符。

数据接收：通过 USART3 实现蓝牙通讯的接收中断，由手机 APP 控制设备运动方向，

4. 超声波测距

基于 STM32F103C8T6 微控制器和超声波传感器，实现：①发送超声波信号；②接收超声波回波信号；③测量距离；④OLED 显示测量距离

功能应用：

1) 时钟

系统初始化时配置系统时钟为 72MHz，保证外设的正确运行。

2) 定时器中断

初始化 TIM3 定时器中断，测量超声波信号的往返时间，处理定时器中断事件，记录 TIM3 溢出次数，获取 TIM3 计数值从而计算距离。

3) GPIO

将 PA3 均设置为推挽输出，PA2 设置为浮空输入，由 TIM3 定时触发超声波传感器发送超声波信号，再由 PA2

接收超声波传感器的回波信号。

4) 硬件 IIC

OLED 显示屏通过硬件 IIC 与微控制器相连，显示距离参数。

5. 传感器采集

基于 STM32F103C8T6 微控制器和 DHT11 温湿度传感器、MQ-135 空气质量传感器、MQ-2 烟雾传感器、LM1204 声音强度传感器这四项传感器，实现：①数字传感器 DHT11 通过内置 ADC 直接输出数字信号 ②其余传感器通过外部 ADC 采样，将模拟信号转换为数字信号输出 ③多路 ADC 采样并转存至 DMA，从 DMA 连续读取数据传输至内存 ④通过 OLED 显示传感器数据。

功能应用：

1) 时钟

系统初始化时配置系统时钟为 72MHz，保证外设的正确运行。

2) GPIO

将数字传感器 DHT11 的端口配置为将相应的端口配置为推挽输出，则为直接输出数字信号；将其余三个传感器配置为模拟输入，读取外部模拟信号。

3) ADC 模数转换

通过将 GPIO 口配置为模拟输入，读取外部模拟信号。同时配置 ADC1 的多路通道，将其初始化为独立模式，软件触发，连续转换和扫描模式，实现多路 ADC 采样。最后进行 ADC1 使能及 ADC 校准流程，。

4) DMA 数据传送

初始化 DMA，将数据传输方向设置为从外设到存储器，转运次数与通道数一致，最后进行 DMA 使能。

5) 硬件 IIC

OLED 显示屏通过硬件 IIC 与微控制器相连，显示各项传感器参数。

6. 按键控制 OLED 显示

基于 STM32F103C8T6 微控制器和 0.96 寸 OLED 显示屏、四脚按键，实现：①按键触发外部中断，实现 OLED 翻页功能 ②OLED 每一页显示一个传感器的值，并对中英文进行取模显示。

1) 时钟

系统初始化时配置系统时钟为 72MHz，保证外设的正确运行。

2) GPIO

将按键接入的 PB13 设置为上拉输入，将 OLED 接入的 PB8 和 PB9 设置为推挽输出。

3) 外部中断 EXTI

将按键的 GPIO 设置为外部中断，为上升沿触发，若按键按下则触发中断，对 OLED 进行翻页。

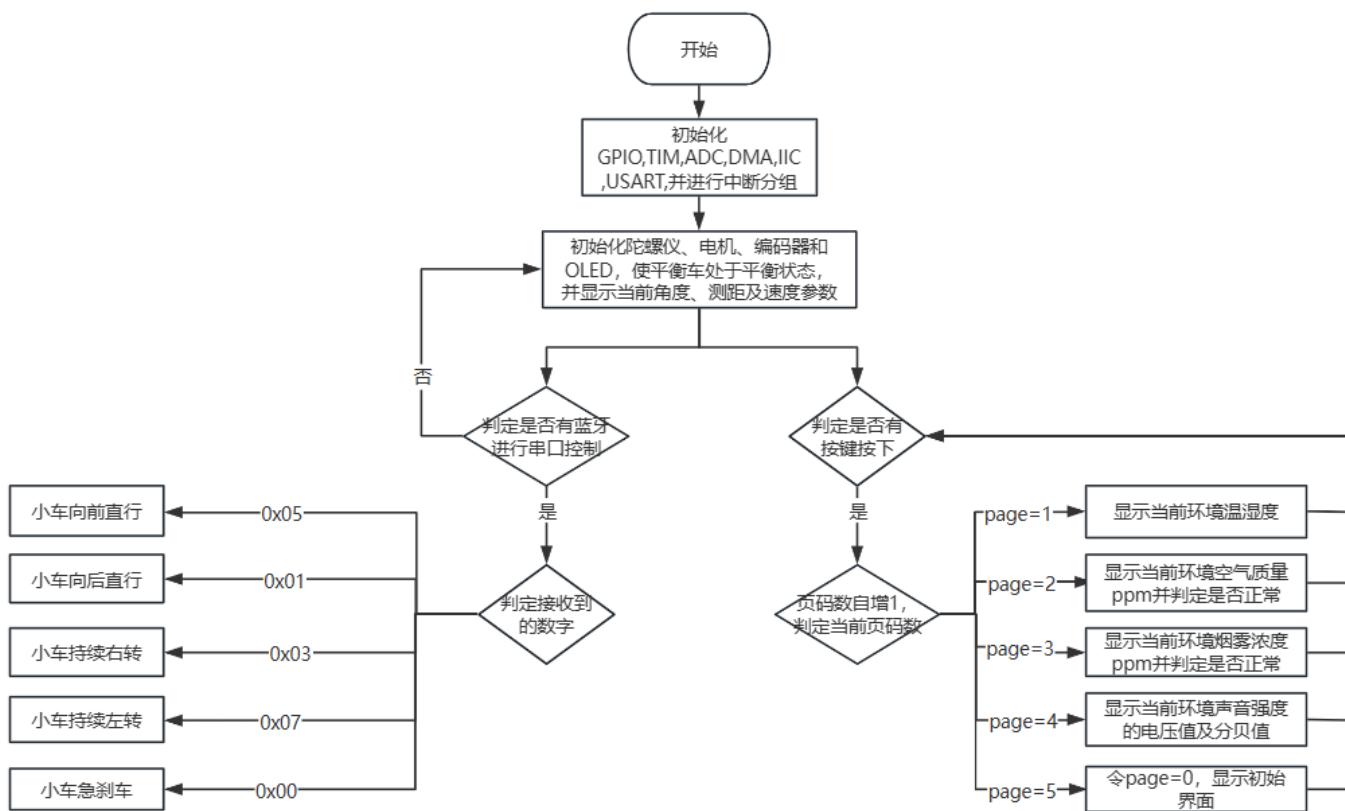
4) 硬件 IIC

将 PB8 和 PB9 分别映射为 IIC 的 SCL 和 SCA，并关闭其 JTAG 功能，释放引脚供 IIC 使用。通过 OLED_Write_IIC_Command 和 OLED_Write_IIC_Data 函数用于向 OLED 显示屏发送命令和数据。这些函数首先发

送起始信号，然后发送 OLED 的 I2C 地址（左移一位并根据写操作设置最低位），接着发送命令或数据，最后等待应答信号。

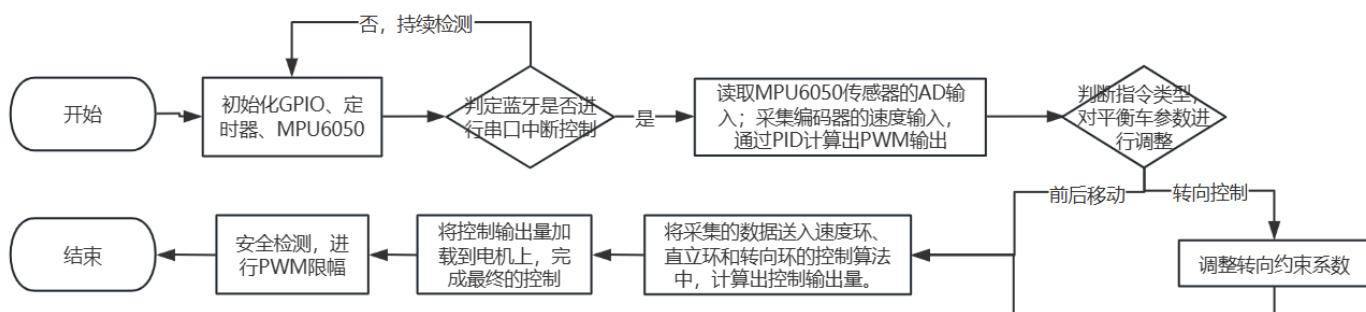
五. 程序流程图

1. 系统总体流程图

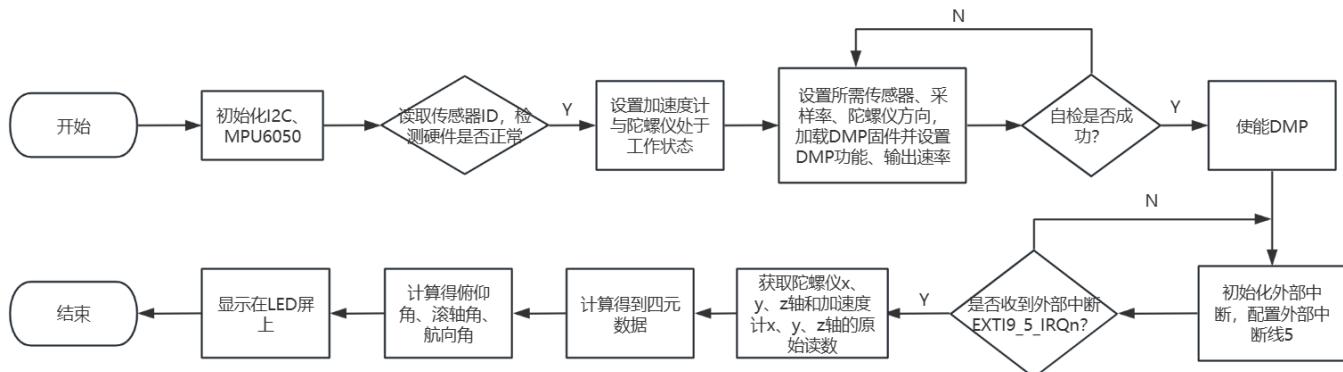


2. 核心板块流程图

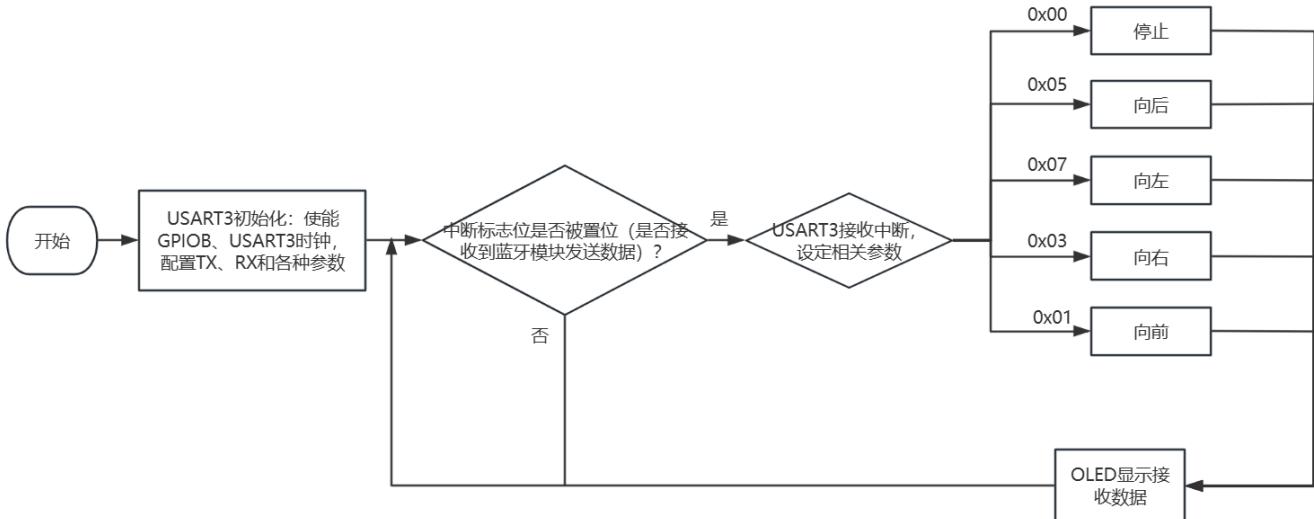
2.1 陀螺仪



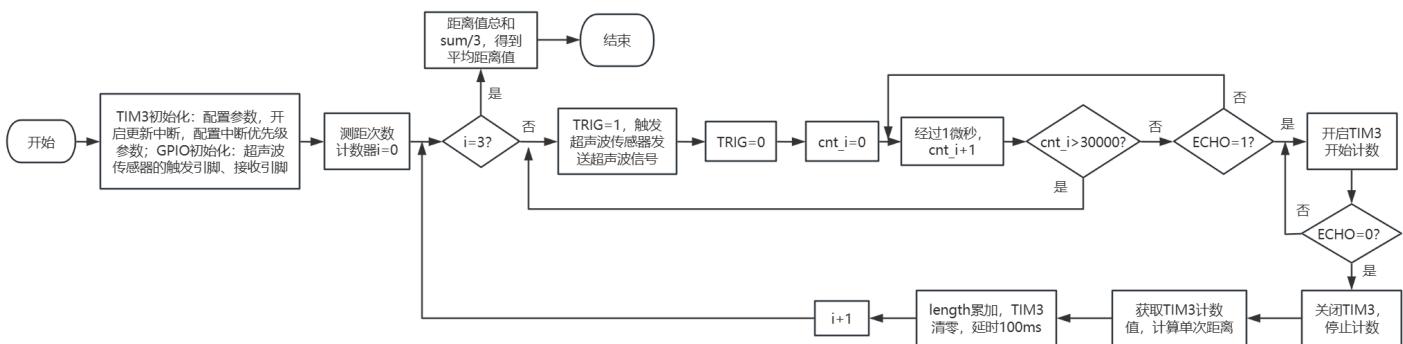
2.2 电机驱动



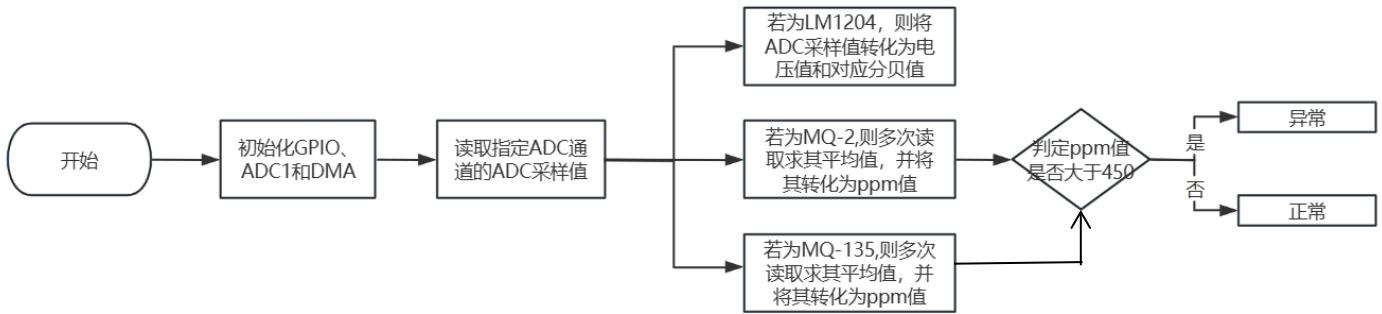
2.3 蓝牙远程控制



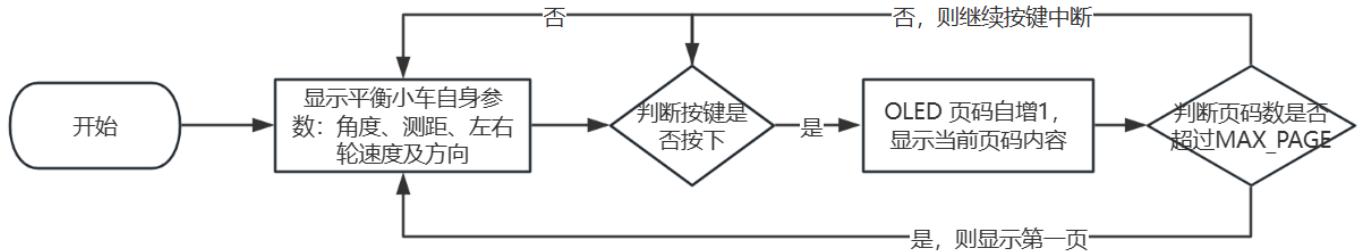
2.4 超声波测距



2.5 传感器采集



2.6 按键控制 OLED 显示



六. 核心代码简述

1. 陀螺仪

- 对 MPU6050 进行初始化，并读取传感器 ID，检测硬件是否正常。若传感器检测正常，设置加速度计与陀螺仪处于工作状态

```
u8 MPU_Init(void)
{
    u8 res;
    MPU_IIC_Init(); // 初始化IIC总线
    MPU_Write_Byte(MPU_PWR_MGMT1_REG, 0X80); // 复位MPU6050
    delay_ms(100);
    MPU_Write_Byte(MPU_PWR_MGMT1_REG, 0X00); // 唤醒MPU6050
    MPU_Set_Gyro_Fsr(3); // 陀螺仪传感器,±2000dps
    MPU_Set_Accel_Fsr(0); // 加速度传感器,±2g
    MPU_Set_Rate(200); // 设置采样率50Hz
    MPU_Write_Byte(MPU_INT_EN_REG, 0X00); // 关闭所有中断
    MPU_Write_Byte(MPU_USER_CTRL_REG, 0X00); // I2C主模式关闭
    MPU_Write_Byte(MPU_FIFO_EN_REG, 0X00); // 关闭FIFO
    MPU_Write_Byte(MPU_INTP_CFG_REG, 0X80); // INT引脚低电平有效
    res=MPU_Read_Byte(MPU_DEVICE_ID_REG);
```

```

    res=MPU_Read_Byte(MPU_DEVICE_ID_REG);
    if(res==MPU_ADDR)//器件ID正确
    {
        MPU_Write_Byte(MPU_PWR_MGMT1_REG,0X01); //设置CLKSEL,PLL X轴为参考
        MPU_Write_Byte(MPU_PWR_MGMT2_REG,0X00); //加速度计与陀螺仪都工作
        MPU_Set_Rate(100); //设置采样率为50Hz
    }else return 1;
    return 0;
}

```

- 2) 设置所需传感器、FIFO、采样率、陀螺仪方向，加载DMP固件并设置DMP功能、输出速率，

```

//设置陀螺仪、加速度计的全量程范围、低通滤波器、采样率和FIFO配置
if (mpu_set_gyro_fsr(2000))
    return -1;
if (mpu_set_accel_fsr(2))
    return -1;
if (mpu_set_lpf(42))
    return -1;
if (mpu_set_sample_rate(50))
    return -1;
if (mpu_configure_fifo(0))
    return -1;

//    if (int_param)
//        reg_int_cb(int_param);

#ifndef AK89xx_SECONDARY
    setup_compass();
    if (mpu_set_compass_sample_rate(10))
        return -1;
#else
    /* Already disabled by setup_compass. */
    if (mpu_set_bypass(0))
        return -1;
#endif

    mpu_set_sensors(0);
    return 0;
}

```

- 3) 自检成功后使能DMP

```

u8 run_self_test(void)
{
    int result;
    //char test_packet[4] = {0};
    long gyro[3], accel[3];
    result = mpu_run_self_test(gyro, accel);
    if (result == 0x7)
    {
        /* Test passed. We can trust the gyro data here, so let's push it down
         * to the DMP.
        */
        float sens;//用于存储陀螺仪的灵敏度
        unsigned short accel_sens;//用于存储加速度计的灵敏度
        mpu_get_gyro_sens(&sens);//获取陀螺仪的灵敏度
        gyro[0] = (long)(gyro[0] * sens);//将陀螺仪x轴的数据乘以灵敏度，进行校准
        gyro[1] = (long)(gyro[1] * sens);
        gyro[2] = (long)(gyro[2] * sens);
        dmp_set_gyro_bias(gyro);//设将校准后的陀螺仪数据设置为DMP的偏置
        mpu_get_accel_sens(&accel_sens);
        accel[0] *= accel_sens;
        accel[1] *= accel_sens;
        accel[2] *= accel_sens;
        dmp_set_accel_bias(accel);
        return 0;
    }else return 1;
}

```

4) 初始化外部中断，配置外部中断线 5 以便 GPIOB 的 PB5 检测到下降沿时触发中断

```
#include "exti.h"

void MPU6050_EXTI_Init(void)
{
    EXTI_InitTypeDef EXTI_InitStruct;
    GPIO_InitTypeDef GPIO_InitStruct;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_AFIO, ENABLE); //开启时钟

    //配置GPIOB的第5脚为上拉输入模式
    GPIO_InitStruct.GPIO_Mode=GPIO_Mode_IPU;/**[1]**//GPIO_Mode_AF_PP
    GPIO_InitStruct.GPIO_Pin=GPIO_Pin_5;//PB5配置为上拉输入
    GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &GPIO_InitStruct);

    GPIO_EXTILineConfig(GPIO_PortSourceGPIOB, GPIO_PinSource5); //连接EXTI Line5到GPIOB的第5脚

    //配置中断线5
    EXTI_InitStruct EXTI_Line=EXTI_Line5;//选择外部中断线5
    EXTI_InitStruct EXTI_LineCmd=ENABLE;//使能外部中断线
    EXTI_InitStruct EXTI_Mode=EXTI_Mode_Interrupt;//配置为中断模式
    EXTI_InitStruct EXTI_Trigger=EXTI_Trigger_Falling;//配置为下降沿触发
    EXTI_Init(&EXTI_InitStruct); //初始化外部中断
}
```

5) 收到外部中断 EXTI9_5_IRQHandler 后，获取陀螺仪 x、y、z 轴的角速度和加速度的原始读数

```
//得到陀螺仪值(原始值)
//gx,gy,gz:陀螺仪x,y,z轴的原始读数(带符号)
//返回值:0,成功
// 其他,错误代码
u8 MPU_Get_Gyroscope(short *gx,short *gy,short *gz)
{
    u8 buf[6],res;
    res=MPU_Read_Len(MPU_ADDR,MPU_GYRO_XOUTH_REG,6,buf);
    if(res==0)
    {
        *gx=((ul6)buf[0]<<8)|buf[1];
        *gy=((ul6)buf[2]<<8)|buf[3];
        *gz=((ul6)buf[4]<<8)|buf[5];
    }
    return res;;
}

//得到加速度值(原始值)
//ax,ay,az:加速度计x,y,z轴的原始读数(带符号)
//返回值:0,成功
// 其他,错误代码
u8 MPU_Get_Accelerometer(short *ax,short *ay,short *az)
{
    u8 buf[6],res;
    res=MPU_Read_Len(MPU_ADDR,MPU_ACCEL_XOUTH_REG,6,buf);
    if(res==0)
    {
        *ax=((ul6)buf[0]<<8)|buf[1];
        *ay=((ul6)buf[2]<<8)|buf[3];
        *az=((ul6)buf[4]<<8)|buf[5];
    }
    return res;;
}
```

6) 利用 DMP 计算四元数据，具体过程如下：

① **解析 DMP 数据包**：根据 DMP 的特征掩码（dmp.feature_mask），解析 FIFO 中的数据包。这可能包括四元数、加速度计数据和陀螺仪数据。

② **四元数处理**：

如果 DMP 配置了四元数输出（DMP_FEATURE_LP_QUAT 或 DMP_FEATURE_6X_LP_QUAT），从 FIFO 中读取四元数数据。四元数数据被转换为长整型（long），并存储在 quat 数组中。增加索引 ii 以跳过已读取的四元数数据。

③ **四元数完整性检查**（如果启用了 FIFO_CORRUPTION_CHECK）：通过检查四元数的平方和是否接近 1，来检测

FIFO 数据是否可能已损坏。如果四元数的平方和超出了可接受的阈值，重置 FIFO，清除传感器掩码，并返回错误。

⑤ 加速度计数据处理：如果 DMP 配置了原始加速度计输出 (DMP_FEATURE_SEND_RAW_ACCEL)，从 FIFO 中读取加速度计数据。加速度计数据被转换为短整型 (short)，并存储在 accel 数组中。更新传感器掩码以反映已读取加速度计数据。

⑥ 陀螺仪数据处理：如果 DMP 配置了任何陀螺仪输出 (DMP_FEATURE_SEND_ANY_GYRO)，从 FIFO 中读取陀螺仪数据。陀螺仪数据被转换为短整型 (short)，并存储在 gyro 数组中。更新传感器掩码以反映已读取陀螺仪数据。

```
/* @brief      Get one packet from the FIFO. 从DMP的FIFO队列中读取数据
 * If @e sensors does not contain a particular sensor, disregard the data
 * returned to that pointer.
 * \n @e sensors can contain a combination of the following flags:
 * \n INV_X_GYRO, INV_Y_GYRO, INV_Z_GYRO
 * \n INV_XYZ_GYRO
 * \n INV_XYZ_ACCEL
 * \n INV_WXYZ_QUAT
 * \n If the FIFO has no new data, @e sensors will be zero.
 * \n If the FIFO is disabled, @e sensors will be zero and this function will
 * return a non-zero error code.
 * @param[out] gyro          Gyro data in hardware units.
 * @param[out] accel         Accel data in hardware units.
 * @param[out] quat          3-axis quaternion data in hardware units.
 * @param[out] timestamp     Timestamp in milliseconds.
 * @param[out] sensors       Mask of sensors read from FIFO.
 * @param[out] more          Number of remaining packets.
 * @return      0 if successful.
 */
int dmp_read_fifo(short *gyro, short *accel, long *quat,
                  unsigned long *timestamp, short *sensors, unsigned char *more)
{
    unsigned char fifo_data[MAX_PACKET_LENGTH];
    unsigned char ii = 0;

    /* TODO: sensors[0] only changes when dmp_enable_feature is called. We can
     * cache this value and save some cycles.
     */
    sensors[0] = 0;

    /* Get a packet. */ //从FIFO读取一个数据包
    if (mpu_read_fifo_stream(dmp.packet_length, fifo_data, more))
        return -1;
```

```

/* Parse DMP packet. */
if (dmp.feature_mask & (DMP_FEATURE_LP_QUAT | DMP_FEATURE_6X_LP_QUAT)) {
#endif FIFO_CORRUPTION_CHECK
    long quat_q14[4], quat_mag_sq;
#endif
//解析四元数据
quat[0] = ((long)fifo_data[0] << 24) | ((long)fifo_data[1] << 16) |
    ((long)fifo_data[2] << 8) | fifo_data[3];
quat[1] = ((long)fifo_data[4] << 24) | ((long)fifo_data[5] << 16) |
    ((long)fifo_data[6] << 8) | fifo_data[7];
quat[2] = ((long)fifo_data[8] << 24) | ((long)fifo_data[9] << 16) |
    ((long)fifo_data[10] << 8) | fifo_data[11];
quat[3] = ((long)fifo_data[12] << 24) | ((long)fifo_data[13] << 16) |
    ((long)fifo_data[14] << 8) | fifo_data[15];
ii += 16;//更新索引
#endif FIFO_CORRUPTION_CHECK
/* We can detect a corrupted FIFO by monitoring the quaternion data and
 * ensuring that the magnitude is always normalized to one. This
 * shouldn't happen in normal operation, but if an I2C error occurs,
 * the FIFO reads might become misaligned.
 *
 * Let's start by scaling down the quaternion data to avoid long long
 * math.
 */
//检查四元数据的完整性
quat_q14[0] = quat[0] >> 16;
quat_q14[1] = quat[1] >> 16;
quat_q14[2] = quat[2] >> 16;
quat_q14[3] = quat[3] >> 16;
quat_mag_sq = quat_q14[0] * quat_q14[0] + quat_q14[1] * quat_q14[1] +
    quat_q14[2] * quat_q14[2] + quat_q14[3] * quat_q14[3];
if ((quat_mag_sq < QUAT_MAG_SQ_MIN) ||
    (quat_mag_sq > QUAT_MAG_SQ_MAX)) {
    /* Quaternion is outside of the acceptable threshold. */
    mpu_reset_fifo();
    sensors[0] = 0;
    return -1;//重置FIFO并返回错误
}
sensors[0] |= INV_WXYZ_QUAT;//更新传感器标志
#endif
}

//解析原始加速度计数据
if (dmp.feature_mask & DMP_FEATURE_SEND_RAW_ACCEL) {
    accel[0] = ((short)fifo_data[ii+0] << 8) | fifo_data[ii+1];
    accel[1] = ((short)fifo_data[ii+2] << 8) | fifo_data[ii+3];
    accel[2] = ((short)fifo_data[ii+4] << 8) | fifo_data[ii+5];
    ii += 6;
    sensors[0] |= INV_XYZ_ACCEL;
}

//解析原始陀螺仪数据
if (dmp.feature_mask & DMP_FEATURE_SEND_ANY_GYRO) {
    gyro[0] = ((short)fifo_data[ii+0] << 8) | fifo_data[ii+1];
    gyro[1] = ((short)fifo_data[ii+2] << 8) | fifo_data[ii+3];
    gyro[2] = ((short)fifo_data[ii+4] << 8) | fifo_data[ii+5];
    ii += 6;
    sensors[0] |= INV_XYZ_GYRO;
}

/* Gesture data is at the end of the DMP packet. Parse it and call
 * the gesture callbacks (if registered).
 */
if (dmp.feature_mask & (DMP_FEATURE_TAP | DMP_FEATURE_ANDROID_ORIENT))
    decode_gesture(fifo_data + ii);

get_ms(timestamp);
return 0;
}

```

7) 计算俯仰角 pitch、横滚角 roll、航向角 yaw 的具体角度值。

```

u8 mpu_dmp_get_data(float *pitch, float *roll, float *yaw)
{
    float q0=1.0f,q1=0.0f,q2=0.0f,q3=0.0f;//初始化四元数的分量, q0是实部, q1,q2,q3是虚部
    unsigned long sensor_timestamp;//定义一个变量用于存储传感器的时间戳
    short gyro[3], accel[3], sensors;//定义数组用于存储陀螺仪和加速器的原始数据
    unsigned char more;//定义一个变量用于存储FIFO中是否有更多数据
    long quat[4]; //存储四元数数据
    if(dmp_read_fifo(gyro, accel, quat, &sensor_timestamp, &sensors,&more))return 1;
    /* Gyro and accel data are written to the FIFO by the DMP in chip frame and hardware units.
     * This behavior is convenient because it keeps the gyro and accel outputs of dmp_read_fifo and mpu_read_fifo consistent.
     */
    /*if (sensors & INV_XYZ_GYRO )
    send_packet(PACKET_TYPE_GYRO, gyro);
    if (sensors & INV_XYZ_ACCEL)
    send_packet(PACKET_TYPE_ACCEL, accel); */
    /* Unlike gyro and accel, quaternions are written to the FIFO in the body frame, q30.
     * The orientation is set by the scalar passed to dmp_set_orientation during initialization.
     */
    if(sensors&INV_WXYZ_QUAT)
    {
        q0 = quat[0] / q30; //q30格式转换为浮点数
        q1 = quat[1] / q30;
        q2 = quat[2] / q30;
        q3 = quat[3] / q30;
        //计算得到俯仰角/横滚角/航向角
        *pitch = asin(-2 * q1 * q3 + 2 * q0 * q2) * 57.3; // pitch
        *roll = atan2(2 * q2 * q3 + 2 * q0 * q1, -2 * q1 * q1 - 2 * q2 * q2 + 1) * 57.3; // roll
        *yaw = atan2(2*(q1*q2 + q0*q3), q0*q0+q1*q1-q2*q2-q3*q3) * 57.3; //yaw
    }else return 2;
    return 0;
}

```

2. 电机控制

1) 编码器初始化

配置定时器 TIM2 以读取连接到 PA0 和 PA1 的编码器信号，**配置编码器模式**，配置溢出更新中断标志位，初始化输入捕获。配置定时器 TIM4，与 TIM2 初始化几乎相同，只是使用 PB6、PB7 串口。

```

5 void Encoder_TIM2_Init(void)
6 {
7     GPIO_InitTypeDef GPIO_InitStruct;
8     TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStruct;
9     TIM_ICInitTypeDef TIM_ICInitStruct;
10
11    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);//开启时钟
12    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);
13
14    GPIO_InitStruct.GPIO_Mode=GPIO_Mode_IN_FLOATING;//初始化GPIO--PA0、PA1
15    GPIO_InitStruct.GPIO_Pin=GPIO_Pin_0 |GPIO_Pin_1;
16    GPIO_Init(GPIOA,&GPIO_InitStruct);
17
18    TIM_TimeBaseStructInit(&TIM_TimeBaseInitStruct);//初始化定时器。
19    TIM_TimeBaseInitStruct.TIM_ClockDivision=TIM_CKD_DIV1;
20    TIM_TimeBaseInitStruct.TIM_CounterMode=TIM_CounterMode_Up;
21    TIM_TimeBaseInitStruct.TIM_Period=65535;
22    TIM_TimeBaseInitStruct.TIM_Prescaler=0;
23    TIM_TimeBaseInit(TIM2,&TIM_TimeBaseInitStruct);
24
25    TIM_EncoderInterfaceConfig(TIM2,TIM_EncoderMode_TI12,TIM_ICPolarity_Rising,TIM_ICPolarity_Rising);//配置编码器模式
26
27    TIM_ICStructInit(&TIM_ICInitStruct);//初始化输入捕获
28    TIM_ICInitStruct.TIM_ICFilter=10;
29    TIM_ICInit(TIM2,&TIM_ICInitStruct);
30
31    TIM_ITConfig(TIM2,TIM_IT_Update,ENABLE);//配置溢出更新中断标志位
32    |
33    TIM_SetCounter(TIM2,0);//清零定时器计数值
34
35    TIM_Cmd(TIM2,ENABLE);//开启定时器
36 }

```

2) PID 算法读取编码器速度

代码亮点：通过 PID 算法调节控制输出，确保机器人能够根据实时反馈稳定地行驶，并通过各种传感器、编码器和电机接口进行高效的控制。

```
73 /******  
74 | 编码器  
75 | 速度读取函数  
76 | 入口参数:定时器  
77 |*****/  
78 int Read_Speed(int TIMx)  
79 {  
80     int value_1;  
81     switch(TIMx)  
82     {  
83         case 2:value_1=(short)TIM_GetCounter(TIM2);TIM_SetCounter(TIM2,0);break;//IF是定时器2, 1.采集编码器的计数值并保存。2.将定时器的计数值清零。  
84         case 4:value_1=(short)TIM_GetCounter(TIM4);TIM_SetCounter(TIM4,0);break;  
85         default:value_1=0;  
86     }  
87     return value_1;  
88 }  
89  
90
```

3) 电机初始化

通过 RCC_APB2PeriphClockCmd 启用 GPIOB 时钟。将 PB12、PB13、PB14 和 PB15 引脚设置为推挽输出模式。通过 GPIO_Init 初始化这些引脚的输出功能，设置为 50MHz 的输出速度。并使用限幅函数限制电机输出。

```
3 /*电机初始化函数*/  
4 void Motor_Init(void)  
5 {  
6     GPIO_InitTypeDef GPIO_InitStruct;  
7     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,ENABLE); //开启时钟  
8  
9     GPIO_InitStruct.GPIO_Mode=GPIO_Mode_Out_PP; //初始化GPIO--PB12、PB13、PB14、PB15为推挽输出  
10    GPIO_InitStruct.GPIO_Pin=GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;  
11    GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz;  
12    GPIO_Init(GPIOB,&GPIO_InitStruct);  
13 }  
  
14 /*限幅函数*/  
15 void Limit(int *motoA,int *motoB)  
16 {  
17     if(*motoA>PWM_MAX)*motoA=PWM_MAX;  
18     if(*motoA<PWM_MIN)*motoA=PWM_MIN;  
19  
20     if(*motoB>PWM_MAX)*motoB=PWM_MAX;  
21     if(*motoB<PWM_MIN)*motoB=PWM_MIN;  
22 }  
23  
24
```

4) 控制电机方向和 PWM 输出

根据 moto1 和 moto2 的符号来控制两个电机的旋转方向：如果值为正，电机正转；如果值为负，电机反转。通过 TIM_SetCompare1 和 TIM_SetCompare4 函数，分别控制左轮电机和右轮电机的 PWM 输出。如果当前位置与目标位置误差大于 60，则停止电机。

```
33 /*赋值函数*/  
34 /*入口参数：PID运算完成后的最终PWM值*/  
35 void Load(int moto1,int moto2)//moto1=-200:反转200个脉冲  
36 {  
37     //1.研究正负号, 对应正反转  
38     if(moto1>0) AIn1=1,Ain2=0;//正转  
39     else AIn1=0,Ain2=1;//反转  
40     //2.研究PWM值  
41     TIM_SetCompare1(TIM1,GFP_abs(moto1));  
42  
43     if(moto2>0) Bin1=1,Bin2=0;  
44     else Bin1=0,Bin2=1;  
45     TIM_SetCompare4(TIM1,GFP_abs(moto2));  
46 }
```

```

49     char PWM_Zero=0,stop=0;
50     void Stop(float *Med_Jiaodu,float *Jiaodu)
51     {
52         if(GFP_abs(*Jiaodu-*Med_Jiaodu)>60)
53         {
54             Load(PWM_Zero,PWM_Zero);
55             stop=1;
56         }
57     }

```

3. 蓝牙远程控制

USART3 串口通信功能，包括串口的初始化、接收中断处理以及数据发送函数等。

1) `uart3_init` 函数：用于初始化 USART3 串口及其相关的 GPIO 引脚。

具体实现步骤：时钟使能、GPIO 引脚配置、串口参数配置、中断与使能配置

2) `USART3_IRQHandler` 函数：作为 USART3 的中断服务函数，用于处理串口接收中断事件。

具体实现步骤：中断条件判断、数据接收与处理、

3) `USART3_Send_Data` 函数：用于通过 USART3 串口发送单个字符数据。

4) `USART3_Send_String` 函数：用于通过 USART3 串口发送字符串数据。

```

//uart3_init 初始化USART3串口和相关GPIO引脚, 配置串口的相关参数, 开启接收中断和使能串口功能
void uart3_init(u32 bound)
{
    //GPIO初始化结构体变量
    GPIO_InitTypeDef GPIO_InitStructure;
    //USART初始化结构体变量
    USART_InitTypeDef USART_InitStructure;

    //使能GPIOB时钟, USART3时钟
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3, ENABLE);

    //配置USART3_TX发送引脚 PB10
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//设置引脚输出速度为GPIO_Speed_50MHz
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;//设置为复用推挽输出模式, 用于串口数据发送
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    //配置USART3_RX接收引脚 PB11
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;//设置为浮空输入模式, 适用于接收外部传来的串口数据, 引脚处于浮空状态等待外部信号输入
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    //USART3 初始化设置
    USART_InitStructureUSART_BaudRate = bound;//USART3的波特率 一般设置为9600;
    USART_InitStructureUSART_WordLength = USART_WordLength_8b;//设置串口字长为8位
    USART_InitStructureUSART_StopBits = USART_StopBits_1;//设置停止位为1位, 用于表示一个数据帧的结束标志
    USART_InitStructureUSART_Parity = USART_Parity_No;//设置无奇偶校验
    USART_InitStructureUSART_HardwareFlowControl = USART_HardwareFlowControl_None;//设置无硬件流控制,
    USART_InitStructureUSART_Mode = USART_Mode_Rx | USART_Mode_Tx;//使能USART3的接收(Rx)和发送(Tx)模式, 实现全双工通信
    USART_Init(USART3, &USART_InitStructure);

    //开启USART3的接收中断, 当有新数据接收到串口数据寄存器时, 会触发该中断, 允许系统进入相应的中断服务函数进行处理, 实现实时响应串口接收数据的功能
    USART_ITConfig(USART3, USART_IT_RXNE, ENABLE);
    USART_Cmd(USART3, ENABLE);
}

```

```

u8 Fore,Back,Left,Right;//控制车辆运动方向的不同状态
extern u8 lock;
//USART3_IRQHandler是USART3串口的中断服务函数, 用于处理串口接收中断事件, 根据接收数据来设置方向的变量值
void USART3_IRQHandler(void)
{
    int Bluetooth_data;
    //检查USART3的接收中断标志位(USART_IT_RXNE)是否被置位, 即判断是否有新的数据接收到串口的数据寄存器中
    if(USART_GetITStatus(USART3,USART_IT_RXNE)!=RESET)//接收中断标志位拉高, 发生了接收中断, 有新数据到达
    {
        Bluetooth_data=USART_ReceiveData(USART3);//读取并保存接收的数据
        // if(Bluetooth_data==0xf5)lock=0;
        // sprintf((char *)uart_buf,"%x",Bluetooth_data);
        // OLED_ShowString(0,0,"      ",16);
        // OLED_ShowString(0,0,uart_buf,16);
        if(Bluetooth_data==0x00)Fore=0,Back=0,Left=0,Right=0;//所有方向变量均设置为0, 表示停止状态
        else if(Bluetooth_data==0x01)Fore=1,Back=0,Left=0,Right=0;//向前运动
        else if(Bluetooth_data==0x05)Fore=0,Back=1,Left=0,Right=0;//向后运动
        else if(Bluetooth_data==0x03)Fore=0,Back=0,Left=0,Right=1;//向右运动
        else if(Bluetooth_data==0x07)Fore=0,Back=0,Left=1,Right=0;//向左运动
        else
            Fore=0,Back=0,Left=0,Right=0;//接收到无效或不识别的数据情况, 设备停止运动
    }
}

```

```

//发送单个字符
void USART3_Send_Data(char data)
{
    USART_SendData(USART3,data); //将要发送的字符数据放入USART3的数据寄存器，触发数据发送操作，数据将从串口发送出去
    while(USART_GetFlagStatus(USART3,USART_FLAG_TC)!=1); //循环检测USART3的发送完成标志位，为1时表示数据成功发送
}

//发送字符串
void USART3_Send_String(char *String)
{
    ul6 len,j;
    len=strlen(String); //获取传入字符串的长度
    for(j=0;j<len;j++) //循环逐个发送字符
    {
        USART3_Send_Data(*String++);
    }
}

```

4. 超声波测距

```

//TIM3_Init函数用于初始化TIM3
void TIM3_Init()
{
    GPIO_InitTypeDef GPIO_InitStruct; //定义GPIO初始化结构体变量，用于配置GPIO引脚的相关参数
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure; //定义定时器基本参数初始化结构体变量，用于配置TIM3的相关参数
    NVIC_InitTypeDef NVIC_InitStruct; //定义中断向量初始化结构体变量，用于配置TIM3中断的相关优先级的参数

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE); //使能TIM3时钟
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE); //使能GPIOA的时钟

    //配置PA3相关参数
    GPIO_InitStruct.GPIO_Mode=GPIO_Mode_Out_PP; //设置为推挽输出模式
    GPIO_InitStruct.GPIO_Pin=GPIO_Pin_3;
    GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz; //设置引脚输出速度为50MHz
    GPIO_Init(GPIOA,&GPIO_InitStruct);

    //配置PA2相关参数
    GPIO_InitStruct.GPIO_Mode=GPIO_Mode_IN_FLOATING; //设置为浮空输入模式
    GPIO_InitStruct.GPIO_Pin=GPIO_Pin_2;
    GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_Init(GPIOA,&GPIO_InitStruct);

    //定时器3初始化
    TIM_TimeBaseStructure.TIM_Period = 999; //ARR 定时器的自动重装数值
    TIM_TimeBaseStructure.TIM_Prescaler = 7199; //PSC 预分频器
    TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; //设置定时器计数模式为向上计数模式
    TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);

    TIM_ITConfig(TIM3, TIM_IT_Update,ENABLE ); //使能制定TIM3中断，允许更新中断
    //当定时器发生计数溢出(即达到自动重装数值后开始计数时)，触发该中断，系统进入相应的中断服务函数

    //中断设置
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); //设置中断优先级分组为第2组，抢占优先级占2位，子优先级占2位
    NVIC_InitStruct.NVIC_IRQChannel = TIM3_IRQn;
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0; //设置抢占优先级为0(抢占优先级决定在多个中断同时触发时哪个中断优先得到响应并执行中断服务函数)
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 3; //设置子优先级为3(子优先级在抢占优先级相同的情况下，用于进一步区分中断的响应顺序)
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStruct);

    TIM_Cmd(TIM3, DISABLE);
}

```

```

//测距函数
int Senor_Using()
{
    unsigned int sum=0;
    unsigned int tim;//用于存储TIM3的计数值，该数值与超声波信号传播时间相关
    unsigned int i=0;//用于控制循环进行多次距离测量操作
    unsigned int length;//用于临时存储单次测量计算得到的距离值
    ul6 cnt_i=0;//用于等待超声波回波信号(ECHO引脚变为高电平)时的计数，防止长时间等待无响应导致程序卡死

    //通过循环进行3次距离测量操作，获取测量结果并求取平均值
    while(i!=3)
    {
        TRIG=1; //触发超声波传感器发送超声波信号 TRIG-超声波传感器的触发端
        delay_us(20);
        TRIG=0; //触发信号发送完毕后，将触发引脚拉低电平

        cnt_i=0;
        while(ECHO==0) //等待超声波传感器的回波信号 ECHO-传感器的回波接收端，变为高电平表示接收到反射回来的超声波信号
        {
            cnt_i++; //记录等待时长
            delay_us(1);

            if(cnt_i>30000) //如果等待超过30000微秒，则重新触发传感器发送信号，并重置计数变量
            {
                TRIG=1;
                delay_us(20);
                TRIG=0;cnt_i=0;
            }
        }

        TIM_Cmd(TIM3,ENABLE); //当检测到回波信号后，开启TIM3开始计数，用于计算超声波信号的往返时间
        i++; //测量次数计数器+1 表示完成了一次有效的信号发送和接收检测过程

        while(ECHO==1); //等待至ECHO变为低电平，表示超声波回波信号接收完毕，一次完整的超声波信号发送、反射和接收过程结束
        TIM_Cmd(TIM3,DISABLE); //关闭TIM3，此时定时器的计数值代表了超声波信号往返所用的时间
    }

    tim=TIM_GetCounter(TIM3); //获取TIM3计数值

    length=(tim*100)/58.0; //根据超声波传播速度(约340m/s)及定时器计数值来计算单次测量得到的距离值
    //距离计算原理 距离=(时间*声速)/2
    sum=length+sum; //将本次测量距离值累加到sum变量中 用于求多次测量的平均距离
    TIM3->CNT=0; //TIM3计数器清零
    overcount++; //重置TIM3溢出次数的记录变量
    delay_ms(100);
}

length=sum/3; //将累加距离值总和除以测量次数，得到多次测量后的平均距离值
return length; //返回最终的平均距离值，作为本次测量函数的结果
}

//TIM3_IRQHandler TIM3的中断服务函数
void TIM3_IRQHandler(void)
{
    if (TIM_GetITStatus(TIM3,TIM_IT_Update) != RESET) //检查是否发生TIM3中断
    //当TIM3计数溢出，硬件会自动将该更新标志位置位，表示发生了中断更新事件
    {
        TIM_ClearITPendingBit(TIM3, TIM_IT_Update ); //清除中断更新标志
        overcount++; //记录TIM3溢出次数
    }
}

```

1) TIM_Int_Init 函数

功能：初始化通用定时器 TIM3 及其相关的 GPIO 引脚，并配置中断相关参数，为后续基于定时器的定时或计数操作以及中断处理做准备。

具体实现步骤：时钟使能、GPIO 引脚配置——PA3，用于后续作为输出触发信号等用途（与距离传感器的触发有关）；PA2，用于接收外部输入信号（距离传感器的回波信号等）、定时器基本参数配置、中断配置：

2) Senor_Using 函数

功能：通过超声波传感器实现距离测量功能，多次触发传感器发送和接收信号，利用 TIM3 记录信号往返时间来计算距离，最后取多次测量结果的平均值作为最终距离值返回 int Senor_Using()

具体实现步骤：变量初始化与准备、多次测量循环——每次测量时，先将触发引脚(TRIG)拉高 20 微秒(delay_us(20))来触发超声波传感器发送超声波信号，然后拉低触发引脚，等待接收引脚 (ECHO) 变为高电平，表示接收到回波

信号开始。在等待 ECHO 变为高电平的过程中，有一个超时处理机制，如果等待时间超过 30000 微秒，则重新触发传感器发送信号，防止程序一直卡在等待状态；当接收到回波信号（ECHO 变为 1）后，开启定时器 3（TIM_Cmd(TIM3,ENABLE)）开始计数，然后等待 ECHO 变为低电平，表示回波接收结束，此时关闭定时器 3，通过 TIM_GetCounter(TIM3) 获取定时器的计数值 tim，并根据超声波传播速度计算出单次测量的距离值 length，累加到 sum 中，同时将定时器的计数器清零（TIM3->CNT=0），重置溢出次数（overcount=0），并延迟 100 毫秒后进行下一次测量、求平均距离并返回

在 mean 函数的 while(1) 无限循环中，首先调用 length = Senor_Using(); 函数获取距离测量值，然后通过一系列 OLED_ 开头的函数（在 OLED 显示屏上显示不同类型的数据，如距离（length）

3) TIM3_IRQHandler 函数

功能：用于处理定时器 3 的更新中断事件，对定时器溢出次数进行计数（overcount 变量累加）

具体实现步骤：通过 TIM_GetITStatus 函数检查定时器 3 的更新中断标志位是否被置位，如果被置位，表示发生了定时器更新中断，进入中断处理逻辑。然后使用 TIM_ClearITPendingBit 函数清除该更新中断的挂起标志位，防止重复进入中断，接着将 overcount 变量加 1，完成对定时器溢出次数的计数操作。

5. 传感器采集

首先初始化 ADC1 和 DMA，达到能够多路 ADC 采样并将值连续存储在 DMA 中的功能。将 LM1204、MQ2、MQ135 中的 ADC 采样分别配置到 ADC1 的通道 6、7、9，并在 DMA 初始化时选择连续扫描，3 次转运，与 ADC1 的采样次数项对应。

```

1 #include "stm32f10x.h"                                // Device header
2 #include "AD.h"                                         // 定义用于存放AD转换结果的全局数组
3 uint16_t AD_Value[3];
4
5 /**
6  * 函 数: AD初始化
7  * 参 数: 无
8  * 返 回 值: 无
9 */
10 void AD_Init(void)
11 {
12     /*开启时钟*/
13     RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);    //开启ADC1的时钟
14     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);    //开启GPIOA的时钟
15     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);    //开启GPIOB的时钟
16     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);      //开启DMA1的时钟
17
18     /*设置ADC时钟*/
19     RCC_ADCCLKConfig(RCC_PCLK2_Div6);                      //选择时钟6分频, ADCCLK = 72MHz / 6 = 12MHz
20
21     /*GPIO初始化*/
22     GPIO_InitTypeDef GPIO_InitStructure;
23     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
24     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
25     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
26     GPIO_Init(GPIOA, &GPIO_InitStructure);                  //将PA6、PA7引脚初始化为模拟输入
27
28     /*GPIO初始化*/
29     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
30     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
31     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
32     GPIO_Init(GPIOB, &GPIO_InitStructure);                  //将PB1引脚初始化为模拟输入
33
34     /*规则组通道配置*/
35     ADC-RegularChannelConfig(ADC1, ADC_Channel_6, 1, ADC_SampleTime_55Cycles5); //规则组序列1的位置, 配置为通道6 LM1204
36     ADC-RegularChannelConfig(ADC1, ADC_Channel_7, 2, ADC_SampleTime_55Cycles5); //规则组序列2的位置, 配置为通道7 MQ2
37     ADC-RegularChannelConfig(ADC1, ADC_Channel_9, 3, ADC_SampleTime_55Cycles5); //规则组序列3的位置, 配置为通道9 MQ135

```

```

39 /*ADC初始化*/
40 ADC_InitTypeDef ADC_InitStructure;
41 ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
42 ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
43 ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
44 ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
45 ADC_InitStructure.ADC_ScanConvMode = ENABLE;
46 ADC_InitStructure.ADC_NbrOfChannel = 3;
47 ADC_Init(ADC1, &ADC_InitStructure);
48
49 /*DMA初始化*/
50 DMA_InitTypeDef DMA_InitStructure;
51 DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)&ADC1->DR; //外设地址，给定形参AddrA
52 DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord; //外设数据宽度，选择半字，对应16位的ADC数据寄存器
53 DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable; //外设地址自增，选择不能，始终以ADC数据寄存器为源
54 DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)&AD_Value; //存储器基址，给定存放AD转换结果的全局数组AD_Value
55 DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord; //存储器数据宽度，选择半字，与源数据宽度对应
56 DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable; //存储器地址自增，选择使能，每次转运后，数组移到下一个位置
57 DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC; //数据传输方向，选择由外设到存储器，ADC数据寄存器转到数组
58 DMA_InitStructure.DMA_BufferSize = 4; //转运的数据大小（转运次数），与ADC通道数一致
59 DMA_InitStructure.DMA_Mode = DMA_Mode_Circular; //模式，选择循环模式，与ADC的连续转换一致
60 DMA_InitStructure.DMA_M2M = DMA_M2M_Disable; //存储器到存储器，选择不能，数据由ADC外设触发转发到存储器
61 DMA_InitStructure.DMA_Priority = DMA_Priority_Medium; //优先级，选择中等
62 DMA_Init(DMA1_Channel1, &DMA_InitStructure); //将结构体变量交给DMA_Init，配置DMA1的通道1
63
64 /*DMA和ADC使能*/
65 DMA_Cmd(DMA1_Channel1, ENABLE); //DMA1的通道1使能
66 ADC_DMACmd(ADC1, ENABLE); //ADC1触发DMA1的信号使能
67 ADC_Cmd(ADC1, ENABLE); //ADC1使能
68
69 /*ADC校准*/
70 ADC_ResetCalibration(ADC1); //固定流程，内部有电路会自动执行校准
71 while (ADC_GetResetCalibrationStatus(ADC1) == SET); //等待校准完成
72 ADC_StartCalibration(ADC1);
73 while (ADC_GetCalibrationStatus(ADC1) == SET); //等待校准完成
74
75 /*ADC触发*/
76 ADC_SoftwareStartConvCmd(ADC1, ENABLE); //软件触发ADC开始工作，由于ADC处于连续转换模式，故触发一次后ADC就可以一直连续不断地工作
77 }

```

5.1 温湿度传感器

由于温湿度传感器为数字传感器，无需模拟输入，只需将数字口先配置为推挽输出进行唤醒，再配置为输入，即可对外界温度和湿度进行读取。DHT11 会发送 40 位的数据，包括 8 位湿度整数部分，8 位湿度小数部分，8 位温度整数部分，8 位温度小数部分，以及 8 位校验和。读取 40 位数据后，解析出温度的整数和小数部分。

```

103 //初始化DHT11的IO口 DQ
104 void DHT11_Init(void)
105 {
106     GPIO_InitTypeDef GPIO_InitStructure;
107     RCC_APB2PeriphClockCmd(DHT11_GPIO_CLK, ENABLE); //使能PA端口时钟
108     GPIO_InitStructure.GPIO_Pin = DHT11_GPIO_PIN; //端口配置
109     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; //推挽输出
110     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
111     GPIO_Init(DHT11_GPIO_PORT, &GPIO_InitStructure); //初始化IO口
112     GPIO_SetBits(DHT11_GPIO_PORT, DHT11_GPIO_PIN); //输出高
113
114     DHT11_Rst(); //复位DHT11
115 }
116
117
118 void DHT11_Mode(u8 mode)
119 {
120     GPIO_InitTypeDef GPIO_InitStructure;
121
122     if(mode)
123     {
124         GPIO_InitStructure.GPIO_Pin = DHT11_GPIO_PIN;
125         GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
126         GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
127     }
128     else
129     {
130         GPIO_InitStructure.GPIO_Pin = DHT11_GPIO_PIN;
131         GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
132     }
133     GPIO_Init(DHT11_GPIO_PORT, &GPIO_InitStructure);
134 }

```

```

42 //从DHT11读取一个位
43 //返回值: 1/0
44 u8 DHT11_Read_Bit(void)
45 {
46     u8 retry=0;
47     while(GPIO_ReadInputDataBit(DHT11_GPIO_PORT, DHT11_GPIO_PIN)&&retry<100)//等待变为低电平
48     {
49         retry++;
50         delay_us(1);
51     }
52     retry=0;
53     while(!GPIO_ReadInputDataBit(DHT11_GPIO_PORT, DHT11_GPIO_PIN)&&retry<100)//等待变高电平
54     {
55         retry++;
56         delay_us(1);
57     }
58     delay_us(40); //等待40us
59     if(GPIO_ReadInputDataBit(DHT11_GPIO_PORT, DHT11_GPIO_PIN))return 1;
60     else return 0;
61 }
62 //从DHT11读取一个字节
63 //返回值: 读到的数据
64 u8 DHT11_Read_Byte(void)
65 {
66     u8 i,dat;
67     dat=0;
68     for(i=0;i<8;i++)
69     {
70         dat<=1;
71         dat|=DHT11_Read_Bit();
72     }
73     return dat;
74 }
75 }

77 //从DHT11读取一次数据
78 //temp:温度值(范围:0~50°)
79 //humi:湿度值(范围:20%~90%)
80 //返回值: 0,正常;1,读取失败
81 u8 DHT11_Read_Data(u8 *temp,u8 *humi)
82 {
83     u8 buf[5];
84     u8 i;
85     DHT11_Rst();
86     if(DHT11_Check()==0)
87     {
88         for(i=0;i<5;i++)//读取40位数据
89         {
90             buf[i]=DHT11_Read_Byte();
91         }
92         if((buf[0]+buf[1]+buf[2]+buf[3])==buf[4])
93         {
94             *humi=buf[0];
95             *temp=buf[2];
96         }
97     }
98     else return 1;
99 }
100 }

```

5.2 烟雾传感器

烟雾传感器为模拟传感器，需要 stm32f103c8t6 中的外置 ADC1 进行模数转换。通过 MQ2_ADC_Read 函数读取 ADC 相应通道的采样值后，由 MQ2_GetData 函数进行多次读取求平均值以达到读数稳定。

MQ2_GetData_PPM 函数用来将读取到的稳定采样值转化为 PPM 值。

```

12 #if MODE
13 uint16_t MQ2_ADC_Read(void)
14 {
15     return AD_Value[1];
16 }
17 #endif
18
19 uint16_t MQ2_GetData(void)
20 {
21
22 #if MODE
23     uint32_t tempData = 0;
24     uint8_t i;
25     for (i = 0; i < MQ2_READ_TIMES; i++)
26     {
27         tempData += MQ2_ADC_Read();
28         delay_ms(5);
29     }
30
31     tempData /= MQ2_READ_TIMES;
32     return tempData;
33
34 #else
35     uint16_t tempData;
36     tempData = !GPIO_ReadInputDataBit(MQ2_DO_GPIO_PORT, MQ2_DO_GPIO_PIN);
37     return tempData;
38 #endif
39 }

```



```

42 float MQ2_GetData_PPM(void)
43 {
44 #if MODE
45     float tempData = 0;
46
47     uint8_t i;
48     for (i = 0; i < MQ2_READ_TIMES; i++)
49     {
50         tempData += MQ2_ADC_Read();
51         delay_ms(5);
52     }
53     tempData /= MQ2_READ_TIMES;
54
55     float Vol = (tempData*5/4096);
56     float RS = (5-Vol)/(Vol*0.5);
57     float R0=6.64;
58
59     float ppm = pow(11.5428*R0/RS, 0.6549f);
60
61     return ppm;
62 #endif
63 }

```

5.3 空气质量传感器

空气质量传感器为模拟传感器，需要 stm32f103c8t6 中的外置 ADC1 进行模数转换。通过 MQ135_ADC_Read 函数读取 ADC 相应通道的采样值后，由 MQ135_GetData 函数进行多次读取求平均值以达到读数稳定。

MQ135_GetData_PPM 函数用来将读取到的稳定采样值转化为 PPM 值。

```

11 #if MODE
12     uint16_t MQ135_ADC_Read(void)
13 {
14     //设置指定ADC的规则组通道,采样时间
15     return AD_Value[2];
16 }
17 #endif
18
19 uint16_t MQ135_GetData(void)
20 {
21
22 #if MODE
23     uint32_t tempData = 0;
24     for (uint8_t i = 0; i < MQ135_READ_TIMES; i++)
25     {
26         tempData += MQ135_ADC_Read();
27         delay_ms(5);
28     }
29
30     tempData /= MQ135_READ_TIMES;
31     return tempData;
32
33 #else
34     uint16_t tempData;
35     tempData = !GPIO_ReadInputDataBit(MQ135_DO_GPIO_PORT, MQ135_DO_GPIO_PIN);
36     return tempData;
37 #endif
38 }
39
40 float MQ135_GetData_PPM(void)
41 {
42 #if MODE
43     float tempData = 0;
44
45
46     for (uint8_t i = 0; i < MQ135_READ_TIMES; i++)
47     {
48         tempData += MQ135_ADC_Read();
49         delay_ms(5);
50     }
51     tempData /= MQ135_READ_TIMES;
52
53     float Vol = (tempData*5/4096);
54     float RS = (5-Vol)/(Vol*0.5);
55     float R0=6.64;
56
57     float ppm = pow(11.5428*R0/RS, 0.6549f);
58
59     return ppm;
60 #endif
61 }
62 }
```

5.4 声音检测传感器

声音检测传感器为模拟传感器，需要 stm32f103c8t6 中的外置 ADC1 进行模数转换。通过 LM1204_ADC_Read 函数读取 ADC 相应通道的采样值后，由 calculate_dB 函数用来将读取到的采样值转化为分贝值。

```

3 uint16_t LM1204_ADC_Read(void)
4 {
5     return AD_Value[0];
6 }
7
8
9 // 计算分贝值
10 float calculate_dB(float voltage) {
11     if (voltage <= 0.0f) {
12         return -INFINITY; // 防止 log10(0) 的错误, 返回负无穷
13     }
14     return 20.0f * log10(voltage / V_REF);
15 }
```

6. 按键控制 OLED 显示

- 1) 首先进行按键 GPIO 初始化，再将其配置为外部中断，在 KEY1_IRQHandler 函数中，每按一下次，则 page 数加 1，通过 page=page%5，使得 page 在 0-4 之间循环。

```
21  /* @brief 配置 IO为EXTI中断口，并设置中断优先级
22  * @param 无
23  * @retval 无
24  */
25
26 void EXTI_Key_Config(void)
27 {
28     GPIO_InitTypeDef GPIO_InitStructure;
29     EXTI_InitTypeDef EXTI_InitStructure;
30     /*开启按键GPIO口的时钟*/
31     RCC_APB2PeriphClockCmd(KEY1_INT_GPIO_CLK, ENABLE);
32     /*-----KEY1配置-----*/
33     /* 选择按键用到的GPIO */
34     GPIO_InitStructure.GPIO_Pin = KEY1_INT_GPIO_PIN;
35     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
36     GPIO_Init(KEY1_INT_GPIO_PORT, &GPIO_InitStructure);
37     /* 选择EXTI的信号源 */
38     GPIO_EXTILineConfig(KEY1_INT_EXTI_PORTSOURCE, KEY1_INT_EXTI_PINSOURCE);
39     EXTI_InitStructure.EXTI_Line = KEY1_INT_EXTI_LINE;
40     EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
41     EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
42     EXTI_InitStructure.EXTI_LineCmd = ENABLE;
43     EXTI_Init(&EXTI_InitStructure);
44
45
46 }
172 void KEY1_IRQHandler(void)
173 {
174     //按键按下后，则翻页
175     if(EXTI_GetITStatus(KEY1_INT_EXTI_LINE) != RESET)
176     {
177         OLED_Clear();
178         OLED_Showtitle();
179         page++;
180         page = page % MAX_PAGE; //page仅在0-4间取值
181         //清除中断标志位
182         EXTI_ClearITPendingBit(KEY1_INT_EXTI_LINE);
183     }
184 }
```

- 2) 在 OLED.C 文件中，对 OLED 进行软件模拟 IIC 配置。

```
86  ****
87  // IIC Write byte
88  ****
89
90 void OLED_Write_IIC_Byte(unsigned char IIC_Byte)
91 {
92     unsigned char i;
93     unsigned char m, da;
94     da=IIC_Byte;
95     OLED_SCLK_Clr();
96     for(i=0;i<8;i++)
97     {
98         m=da;
99         // OLED_SCLK_Clr();
100        m=m&0x80;
101        if(m==0x80)
102            {OLED_SDIN_Set();}
103        else OLED_SDIN_Clr();
104        da=da<<1;
105        OLED_SCLK_Set();
106        OLED_SCLK_Clr();
107    }
108 }
109 ****
110 // IIC Write Command
111 ****
112 void OLED_Write_IIC_Command(unsigned char IIC_Command)
113 {
114     OLED_IIC_Start();
115     OLED_Write_IIC_Byte(0x78); //Slave address, SA0=0
116     OLED_IIC_Wait_Ack();
117     OLED_Write_IIC_Byte(0x00); //write command
118     OLED_IIC_Wait_Ack();
119     OLED_Write_IIC_Byte(IIC_Command);
120     OLED_IIC_Wait_Ack();
121     OLED_IIC_Stop();
122 }
123 ****
124 // IIC Write Data
125 ****
126 void OLED_Write_IIC_Data(unsigned char IIC_Data)
127 {
128     OLED_IIC_Start();
129     OLED_Write_IIC_Byte(0x78); //D/C#=0; R/#=
130     OLED_IIC_Wait_Ack();
131     OLED_Write_IIC_Byte(0x40); //write data
132     OLED_IIC_Wait_Ack();
133     OLED_Write_IIC_Byte(IIC_Data);
134     OLED_IIC_Wait_Ack();
135     OLED_IIC_Stop();
136 }
137 void OLED_WR_Byte(unsigned dat,unsigned cmd)
138 {
139     if(cmd)
140     {
141         OLED_Write_IIC_Data(dat);
142     }
143     else
144     {
145         OLED_Write_IIC_Command(dat);
146     }
147 }
```

a) **GPIO 配置:** 首先，配置两个 GPIO 引脚（通常为 PB8 和 PB9）为推挽输出模式，用于模拟 I2C 的时钟线（SCL）和数据线（SDA）。

b) **IIC 起始条件和停止条件:** 通过将 SDA 线从高拉低，然后 SCL 线从高拉低，生成 I2C 起始条件。这表示一次 I2C 通信的开始；通过将 SDA 线从低拉高，然后 SCL 线从高拉低，生成 I2C 停止条件。这表示一次 I2C 通信的结束。

c) **等待应答:** 在发送数据后，通过将 SDA 线设置为高阻态，然后拉高 SCL 线，检查 SDA 线上的应答信号。如果设备应答（SDA 线被拉低），则继续通信；如果没有应答（SDA 线保持高电平），则停止通信。

d) **发送字节:** 将一个字节的数据通过 SDA 线发送出去，每次发送一位，从最高位到最低位。在发送每一位数据后，SCL 线被拉高，允许接收设备读取该位数据，然后 SCL 线被拉低，准备发送下一位数据。

e) **发送命令和数据:** 使用 OLED_Write_IIC_Command 和 OLED_Write_IIC_Data 函数，分别发送命令和数据到 OLED 显示屏。这些函数内部会调用 OLED_IIC_Start、OLED_Write_IIC_Byte、OLED_IIC_Wait_Ack 和 OLED_IIC_Stop 等函数来完成整个 I2C 通信过程。

f) **OLED 初始化:** 在 OLED_Init 函数中，通过软件模拟 IIC 发送一系列命令到 OLED 显示屏，以配置显示参数，如显示方向、对比度、时钟分频等。

g) **显示操作:** 在显示文本、数字或图像时，通过软件模拟 IIC 发送相应的命令和数据到 OLED 显示屏，以实现显示操作。

```
531 //初始化SSD1306
532 void OLED_Init(void)
533 {
534     GPIO_InitTypeDef GPIO_InitStructure;
535     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB|RCC_APB2Periph_AFIO, ENABLE); //使能B端口时钟
536     GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);
537     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8|GPIO_Pin_9;
538     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; //推挽输出
539     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz; //速度50MHz
540     GPIO_Init(GPIOB, &GPIO_InitStructure); //初始化GPIOB8,9
541     GPIO_SetBits(GPIOB,GPIO_Pin_8|GPIO_Pin_9);
542
543
544     delay_ms(800);
545     OLED_WR_Byte(0xAE, OLED_CMD); //--display off
546     OLED_WR_Byte(0x00, OLED_CMD); //--set low column address
547     OLED_WR_Byte(0x10, OLED_CMD); //--set high column address
548     OLED_WR_Byte(0x40, OLED_CMD); //--set start line address
549     OLED_WR_Byte(0xB0, OLED_CMD); //--set page address
550     OLED_WR_Byte(0x81, OLED_CMD); // contract control
551     OLED_WR_Byte(0xFF, OLED_CMD); //--128
552     OLED_WR_Byte(0xA1, OLED_CMD); //set segment remap
553     OLED_WR_Byte(0xA6, OLED_CMD); //--normal / reverse
554     OLED_WR_Byte(0xA8, OLED_CMD); //--set multiplex ratio(1 to 64)
555     OLED_WR_Byte(0x3F, OLED_CMD); //--1/32 duty
556     OLED_WR_Byte(0xC8, OLED_CMD); //Com scan direction
557     OLED_WR_Byte(0xD3, OLED_CMD); //--set display offset
558     OLED_WR_Byte(0x00, OLED_CMD); //
559
560     OLED_WR_Byte(0xD5, OLED_CMD); //set osc division
561     OLED_WR_Byte(0x80, OLED_CMD); //
562
563     OLED_WR_Byte(0xD8, OLED_CMD); //set area color mode off
564     OLED_WR_Byte(0x05, OLED_CMD); //
565
566     OLED_WR_Byte(0xD9, OLED_CMD); //Set Pre-Charge Period
567     OLED_WR_Byte(0xF1, OLED_CMD); //
568
569     OLED_WR_Byte(0xDA, OLED_CMD); //set com pin configuartion
570     OLED_WR_Byte(0x12, OLED_CMD); //
571
572     OLED_WR_Byte(0xDB, OLED_CMD); //set Vcomh
573     OLED_WR_Byte(0x30, OLED_CMD); //
574
575     OLED_WR_Byte(0x8D, OLED_CMD); //set charge pump enable
576     OLED_WR_Byte(0x14, OLED_CMD); //
577
578     OLED_WR_Byte(0xAF, OLED_CMD); //--turn on oled panel
579 }
```

3) 在主函数中，通过 switch case 语句，根据 page 的值来选择页面；若 page 为 0，则显示初始界面；若 page 为 1，选择温湿度界面；page 为 2、3、4 选择空气质量、烟雾浓度、声音检测界面。

在显示初始界面时，通过 Sensor_Using 调用测距函数显示距离，而其余陀螺仪角度、左右电机速度均为全局变

量，在 control.c 文件中得到。

在显示温湿度界面时，通过 DHT11_Read_Data 函数得到当前温湿度 temp 和 humi

在显示空气质量界面时，通过 MQ135_GetData 和 MQ135_GetData_PPM 函数得到空气质量的原始值和 ppm 值

在显示烟雾浓度界面时，通过 MQ2_GetData 和 MQ2_GetData_PPM 函数得到烟雾浓度的原始值和 ppm 值

在显示声音强度界面时，通过 LM1204_ADC_Read 得到声音强度初始值，由 $voltage = (Voice_Value / 4096) * 3.3$

将初始值转化为电压值，再由 calculate_dB(voltage)将其转化为分贝值。

```
71     switch (page){  
72         case 1://显示平衡车自身参数  
73             length = Senor_Using(); //调用测距函数  
74             OLED_ShowString(0, 2, "Angle:", 16);  
75             OLED_ShowString(0, 4, "Dist:", 16);  
76             OLED_ShowString(0, 6, "Speed:", 16);  
77             //显示陀螺仪角度、超声波距离及当前左右电机速度  
78             OLED_Float(3, 60, Pitch, 1);  
79             OLED_Float(5, 60, length, 1);  
80             OLED_Num3(10, 6, Encoder_Left);  
81             OLED_Num3(10, 7, Encoder_Right);  
82             delay_ms(100);  
83             break;  
84         case 0://显示温湿度  
85             DHT11_Read_Data(&temp, &humi);  
86             OLED_ShowString(0, 2, "Temp:    C", 16);  
87             OLED_Num2(8, 3, temp);  
88             OLED_ShowString(0, 4, "Humi:    %", 16);  
89             OLED_Num2(8, 5, humi);  
90             delay_ms(100);  
91             break;  
92         case 2://显示空气质量  
93             MQ135_value = MQ135_GetData();  
94             MQ135_ppm = MQ135_GetData_PPM(); // 计算 ppm 值  
95             OLED_ShowMQ135(0, 2, 0);  
96             OLED_ShowMQ135(20, 2, 1);  
97             OLED_ShowMQ135(40, 2, 2);  
98             OLED_ShowMQ135(60, 2, 3);  
99             OLED_ShowMQ135(80, 2, 4);  
100            OLED_Num4(100, 3, MQ135_value);  
101            sprintf((char*)MQ135_buff, ".2fppm      ", MQ135_ppm);  
102            OLED_ShowString(48, 4, MQ135_buff, 16);  
103            if (MQ135_ppm>450){  
104                OLED_ShowString(48, 6, "Abnormal", 16); //异常  
105            }  
106            else{  
107                OLED_ShowString(48, 6, "Normal", 16); //正常  
108            }  
109            delay_ms(100);  
  
110         case 3://显示声音强度  
111         Voice_Value = (float)LM1204_ADC_Read();  
112         voltage = (Voice_Value / 4096) * 3.3;  
113         dB_value = calculate_dB(voltage); //计算分贝值  
114         sprintf((char*)LM1204_buff, "Voice_val:%0.0f      ", Voice_Value);  
115         OLED_ShowString(0, 2, LM1204_buff, 16);  
116         //OLED_Num4(100, 3, Voice_Value);  
117         // 在OLED上显示电压值  
118         sprintf((char*)LM1204_buff, "Voltage:%0.2fV    ", voltage);  
119         OLED_ShowString(0, 4, LM1204_buff, 16);  
120         // 在 OLED 上显示分贝值  
121         sprintf((char*)LM1204_buff, "dB:%.2f", dB_value);  
122         OLED_ShowString(32, 6, LM1204_buff, 16);  
123         delay_ms(1000);  
124         break;  
125     case 4://显示烟雾强度  
126     MQ2_value = MQ2_GetData(); // 获取烟雾浓度的原始数据  
127     MQ2_ppm = MQ2_GetData_PPM(); // 计算 ppm 值  
128     OLED_ShowMQ2(0, 2, 0);  
129     OLED_ShowMQ2(20, 2, 1);  
130     OLED_ShowMQ2(40, 2, 2);  
131     OLED_ShowMQ2(60, 2, 3);  
132     OLED_ShowMQ2(80, 2, 4);  
133     OLED_Num4(100, 3, MQ2_value);  
134     sprintf((char*)MQ2_buff, ".2fppm      ", MQ2_ppm);  
135     OLED_ShowString(48, 4, MQ2_buff, 16);  
136     if (MQ2_ppm>450){  
137         OLED_ShowString(48, 6, "Abnormal", 16); //异常  
138     }  
139     else{  
140         OLED_ShowString(48, 6, "Normal", 16); //正常  
141     }  
142     delay_ms(100);  
143     break;  
144     default:  
145         page = 0;  
146         break;}
```

七. 系统创新

1. 蓝牙远程控制

```
35 u8 Fore, Back, Left, Right;
36 extern u8 lock;
37 void USART3_IRQHandler(void)
38 {
39     int Bluetooth_data;
40     if(USART_GetITStatus(USART3, USART_IT_RXNE)!=RESET)//接收中断标志位拉高
41     {
42         Bluetooth_data=USART_ReceiveData(USART3);//保存接收的数据
43         //if(Bluetooth_data==0xf5)lock=0;
44         //sprintf((char *)uart_buf,"%x",Bluetooth_data);
45         //OLED_ShowString(0,0," ",16);
46         //OLED_ShowString(0,0,uart_buf,16);
47         if(Bluetooth_data==0x00)Fore=0,Back=0,Left=0,Right=0;//刹
48         else if(Bluetooth_data==0x05)Fore=1,Back=0,Left=0,Right=0;//前
49         else if(Bluetooth_data==0x01)Fore=0,Back=1,Left=0,Right=0;//后
50         else if(Bluetooth_data==0x03)Fore=0,Back=0,Left=0,Right=1;//右
51         else if(Bluetooth_data==0x07)Fore=0,Back=0,Left=1,Right=0;//左
52         else
53             Fore=0,Back=0,Left=0,Right=0;//刹车
54     }
55 }
```

我们实现了通过蓝牙模块与智能手机的无缝连接，用户可以利用手机应用远程控制平衡车的运动。这种远程控制不仅提高了用户体验，还为平衡车的应用场景提供了更多可能性，如远程监控和智能家居集成。

2. 陀螺仪 DMP 算法姿态融合

```
981 /**
982 * @brief 启用DMP功能。
983 * 此函数根据输入掩码启用指定的DMP功能。
984 * 输入掩码中使用的宏定义包括：
985 * - DMP_FEATURE_TAP：启用敲击检测。
986 * - DMP_FEATURE_ANDROID_ORIENT：启用安卓方向检测。
987 * - DMP_FEATURE_LP_QUAT：启用低功耗四元数。
988 * - DMP_FEATURE_6X_LP_QUAT：启用6倍低功耗四元数。
989 * - DMP_FEATURE_GYRO_CAL：启用陀螺仪校准。
990 * - DMP_FEATURE_SEND_RAW_ACCEL：发送原始加速度计数据。
991 * - DMP_FEATURE_SEND_RAW_GYRO：发送原始陀螺仪数据。
992 * 注意：DMP_FEATURE_LP_QUAT和DMP_FEATURE_6X_LP_QUAT互斥。
993 * 注意：DMP_FEATURE_SEND_RAW_GYRO和DMP_FEATURE_SEND_CAL_GYRO也互斥。
994 *
995 * @param[in] mask 要启用的功能掩码。
996 * @return 成功返回0。
997 */
998 int dmp_enable_feature(unsigned short mask) {
999     unsigned char tmp[10];
1000
1001     // 设置积分比例因子。
1002     tmp[0] = (unsigned char)((GYRO_SF >> 24) & 0xFF);
1003     tmp[1] = (unsigned char)((GYRO_SF >> 16) & 0xFF);
1004     tmp[2] = (unsigned char)((GYRO_SF >> 8) & 0xFF);
1005     tmp[3] = (unsigned char)(GYRO_SF & 0xFF);
1006     mpu_write_mem(D_0_104, 4, tmp); // 写入DMP内存地址D_0_104。
1007
1008     // 发送传感器数据到FIFO。
1009     tmp[0] = 0xA3; // FIFO配置的默认值。
1010     // 如果启用了，配置FIFO发送原始加速度计数据。
1011     if (mask & DMP_FEATURE_SEND_RAW_ACCEL) {
1012         // 如果启用了，配置FIFO发送原始陀螺仪数据。
1013         if (mask & DMP_FEATURE_SEND_ANY_GYRO) {
1014             tmp[7] = 0xA3;
1015             tmp[8] = 0xA3;
1016             tmp[9] = 0xA3;
1017             mpu_write_mem(CFG_15, 10, tmp); // 写入DMP内存地址CFG_15。
1018
1019             // 发送手势数据到FIFO。
1020             if (mask & (DMP_FEATURE_TAP | DMP_FEATURE_ANDROID_ORIENT))
1021                 tmp[0] = DIN_A20; // 启用敲击或安卓方向。
1022             else
1023                 tmp[0] = 0xD8; // 禁用敲击和安卓方向。
1024             mpu_write_mem(CFG_27, 1, tmp); // 写入DMP内存地址CFG_27。
1025         }
1026     }
1027 }
```

```

1042 // 如果指定了，启用陀螺仪校准。
1043 if (mask & DMP_FEATURE_GYRO_CAL)
1044     dmp_enable_gyro_cal(1);
1045 else
1046     dmp_enable_gyro_cal(0);
1047
1048 // 配置原始陀螺仪数据发送。
1049 if (mask & DMP_FEATURE_SEND_ANY_GYRO) {
1050     // 如果指定了，启用敲击检测。
1051     if (mask & DMP_FEATURE_TAP) {
1052         // 如果指定了，启用安卓方向检测。
1053         if (mask & DMP_FEATURE_ANDROID_ORIENTATION) {
1054             tmp[0] = 0xD9; // 启用安卓方向。
1055         } else {
1056             tmp[0] = 0xD8; // 禁用安卓方向。
1057         }
1058         mpu_write_mem(CFG_ANDROID_ORIENTATION_INT, 1, tmp); // 写入DMP内存地址CFG_ANDROID_ORIENTATION_INT。
1059
1060         // 启用低功耗四元数或6倍低功耗四元数。
1061         if (mask & DMP_FEATURE_LP_QUAT)
1062             dmp_enable_lp_quat(1);
1063         else
1064             dmp_enable_lp_quat(0);
1065         if (mask & DMP_FEATURE_6X_LP_QUAT)
1066             dmp_enable_6x_lp_quat(1);
1067         else
1068             dmp_enable_6x_lp_quat(0);
1069
1070         // 计步器总是启用的。
1071         dmp.feature_mask = mask | DMP_FEATURE_PEDOMETER;
1072         mpu_reset_fifo(); // 重置FIFO。
1073
1074         // 根据启用的功能计算数据包长度。
1075         dmp.packet_length = 0;
1076         if (mask & DMP_FEATURE_SEND_RAW_ACCEL)
1077             dmp.packet_length += 6;
1078         if (mask & DMP_FEATURE_SEND_ANY_GYRO)
1079             dmp.packet_length += 6;
1080         if (mask & (DMP_FEATURE_LP_QUAT | DMP_FEATURE_6X_LP_QUAT))
1081             dmp.packet_length += 16;
1082         if (mask & (DMP_FEATURE_TAP | DMP_FEATURE_ANDROID_ORIENTATION))
1083             dmp.packet_length += 4;
1084
1085     }
1086
1087     return 0; // 返回成功。
1088 }
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999

```

通过使用 MPU6050 陀螺仪的数字运动处理器（DMP），我们实现了对平衡车姿态的精确控制。DMP 算法能够提供稳定的角速度和角度数据，这些数据被用于姿态融合算法，以实现对平衡车姿态的精确估计和控制，从而提高了平衡车的稳定性和响应速度。

3. 电机 PID 算法控制

```

114 /* ****
115 直立环PD控制器: Kp*Ki*Kd
116
117 入口: 期望角度、真实角度、真实角速度
118 出口: 直立环输出
119 ****/
120 int Vertical(float Med, float Angle, float gyro_Y)
121 {
122     int PWM_out;
123
124     PWM_out=Vertical_Kp*(Angle-Med)+Vertical_Kd*(gyro_Y-0);
125     return PWM_out;
126 }
127
128
129 /* ****
130 速度环PI: Kp*Ki*Kd
131 ****/
132 int Velocity(int Target, int encoder_left, int encoder_right)
133 {
134     static int Encoder_S,EnC_Err_Lowout_last,PWM_out,Encoder_Err,EnC_Err_Lowout;
135     float a=0.7;
136
137     //1.计算速度偏差
138     Encoder_Err=(encoder_left+encoder_right)-Target;//舍去误差--我的理解：能够让速度为“0”的角度，就是机械中值。
139     //2.对速度偏差进行低通滤波
140     //low_out=(1-a)*Encoder_Err+a*low_out_last;
141     EnC_Err_Lowout=(1-a)*Encoder_Err+a*EnC_Err_Lowout_last;//使得波形更加平滑，滤除高频干扰，防止速度突变。
142     EnC_Err_Lowout=last=EnC_Err_Lowout;//防止速度过大影响直立环的正常工作。
143
144     //3.对速度偏差积分，积分出位移
145     Encoder_S+=Encoder_Err_Lowout;
146
147     //4.积分限幅
148     Encoder_S=Encoder_S>10000?10000:(Encoder_S<-10000)?-10000:Encoder_S;
149
150     if(stop==1)Encoder_S=0,stop=0;//清零积分量
151
152     //5.速度环控制输出计算
153     PWM_out=Velocity_Kp*Encoder_Err+Velocity_Ki*Encoder_S;
154
155 }
156
157
158 /* ****
159 转向环:系数*Z轴角速度+系数*遥控数据
160 ****/
161 int Turn(int gyro_Z, int RC)
162 {
163     int PWM_out;
164
165     //这不是一个严格的PD控制器，Kd针对的是转向的约束，但Kp针对的是遥控的转向。
166     PWM_out=Turn_Kd*gyro_Z + Turn_Kp*RC;
167     return PWM_out;
168 }
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193

```

项目中采用了先进的 PID 控制算法来调节电机的速度和方向。PID 控制器能够根据实时反馈调整 PWM 信号，确保电机运行的精确性和稳定性。这种控制策略使得平衡车能够在各种环境下保持平衡，即使在面对突发的外部干扰时也能迅速恢复稳定。

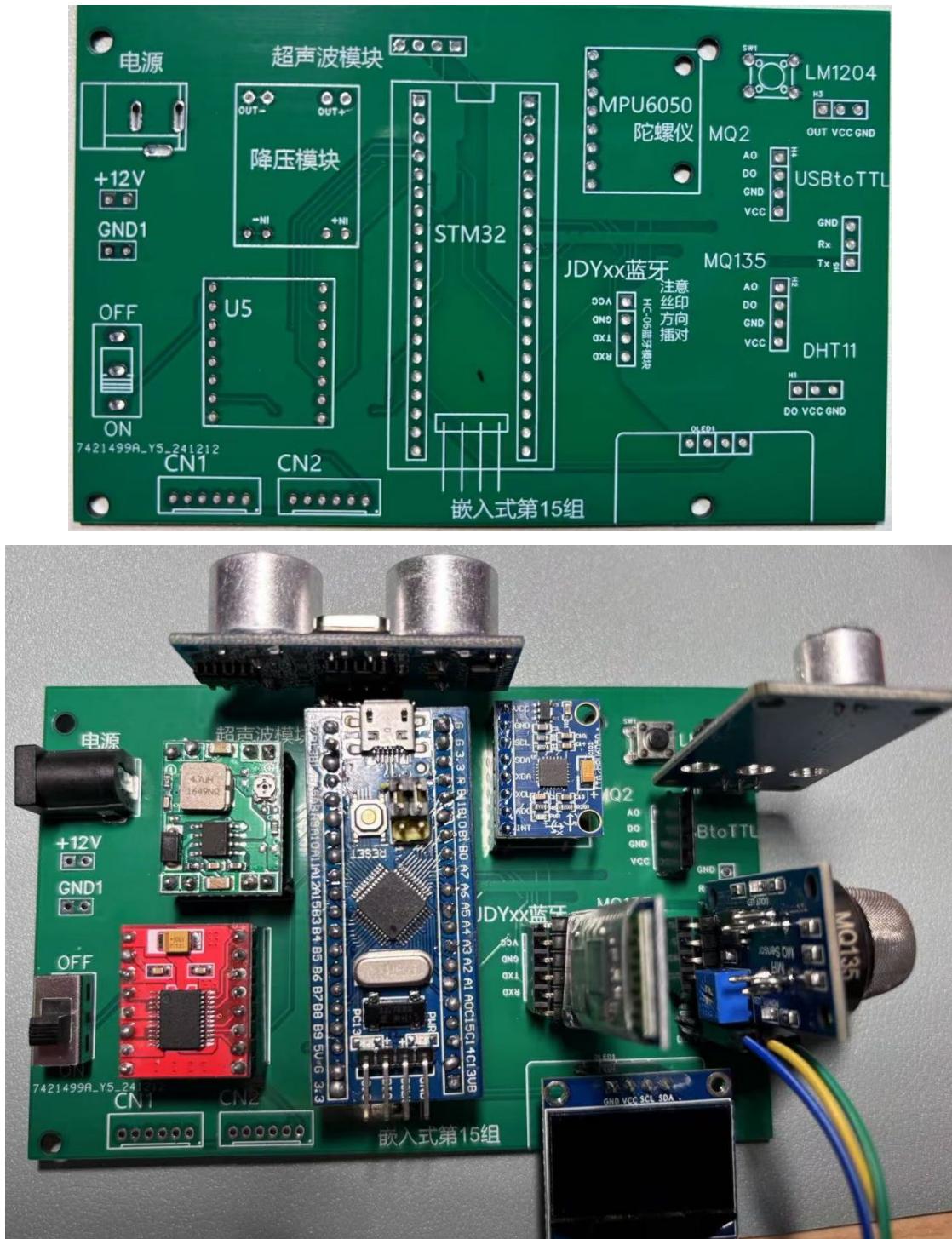
4. OLED 汉字取模显示

- 1) 确定汉字编码与字模数据：汉字的显示依赖于字模数据，即汉字的点阵表示。每个汉字都有一个对应的字模数组，该数组包含了构成该汉字的点阵数据。
 - 2) 设置显示位置：使用 `OLED_Set_Pos(x, y)` 函数设置 OLED 显示屏上的起始坐标 (x, y) ，这是要显示汉字的左上角位置。
 - 3) 分页显示汉字：由于 OLED 显示屏的内存是按页组织的，每个汉字的点阵数据需要分两页来显示（假设使用的是 16×16 的点阵）。第一页（上半部分）包含前 16 个字节的数据，第二页（下半部分）包含接下来的 16 个字节的数据。
 - 4) 写入字模数据：通过循环，将汉字的上半部分字模数据写入 OLED 的显存。循环变量 t 从 0 到 15，对应汉字点阵的上半部分。之后，再次设置光标位置到下一行 $(x, y+1)$ ，以准备写入汉字的下半部分。再次循环，将汉字的下半部分字模数据写入 OLED 的显存。循环变量 t 从 16 到 31，对应汉字点阵的下半部分。
 - 5) 数据传输：使用 `OLED_WR_Byte(data, OLED_DATA)` 函数将字模数据写入 OLED。这个函数负责将数据发送

到 OLED 显示屏，其中 `data` 是要写入的字节，`OLED_DATA` 表示正在发送的是数据。

完成两页数据的写入后，一个完整的汉字就被显示在了 OLED 屏幕上。

5. 自主 PCB 打板



在硬件设计方面，我们进行了自主的 PCB 设计和打板，这不仅降低了成本，还提高了系统的集成度和可靠性。自主 PCB 设计使得我们能够根据项目的具体需求优化电路布局，减少信号干扰，提高电源效率，从而提升了整个系统的性能。

八. 实验心得

在这次“新型宠物”——基于平衡车的智能环境感知系统的项目实践中，我们小组经历了一段充实且富有成效的探索旅程。通过将课堂所学的理论知识与实际应用相结合，我们深刻体会到了跨学科知识在解决实际问题中的核心价值。在项目实施的每一个环节，从电路设计、编程到用户界面的实现，我们都亲力亲为，这不仅加深了我们对 PID 控制算法和数据融合技术等关键概念的理解，也锻炼了我们的实践技能。

在团队协作方面，我们小组展现出了卓越的合作精神。面对项目的复杂性，我们通过有效的沟通和明确分工，共同克服了重重困难。这一过程不仅提升了我们的工作效率，也让我们学会了如何在团队中发挥个人优势，共同推动项目向前发展。

在解决项目中遇到的硬件和软件问题时，我们小组的创新思维和解决问题的能力得到了充分的锻炼。我们不断尝试新的解决方案，从不同角度思考问题，这不仅丰富了我们的设计思路，也增强了我们的技术实力。

我们小组在项目中的创新实践，如自主 PCB 打板、蓝牙远程控制的实现，以及陀螺仪 DMP 算法的融合应用，都体现了我们对技术细节的极致追求。这些创新点不仅提升了项目的技术含量，也让我们在实践中学会了如何将创新思维转化为实际成果。

通过这次实验，我们小组的每一位成员都在专业技能和问题解决能力上得到了显著提升。我们深刻认识到了持续学习的重要性，并坚信这些经验和体会将为我们未来的学术探索和职业发展奠定坚实的基础。我们为能在这样一个充满挑战和机遇的项目中共同成长感到自豪，并期待将所学知识应用于未来的更多创新项目中。

报告评分：

指导教师签字：