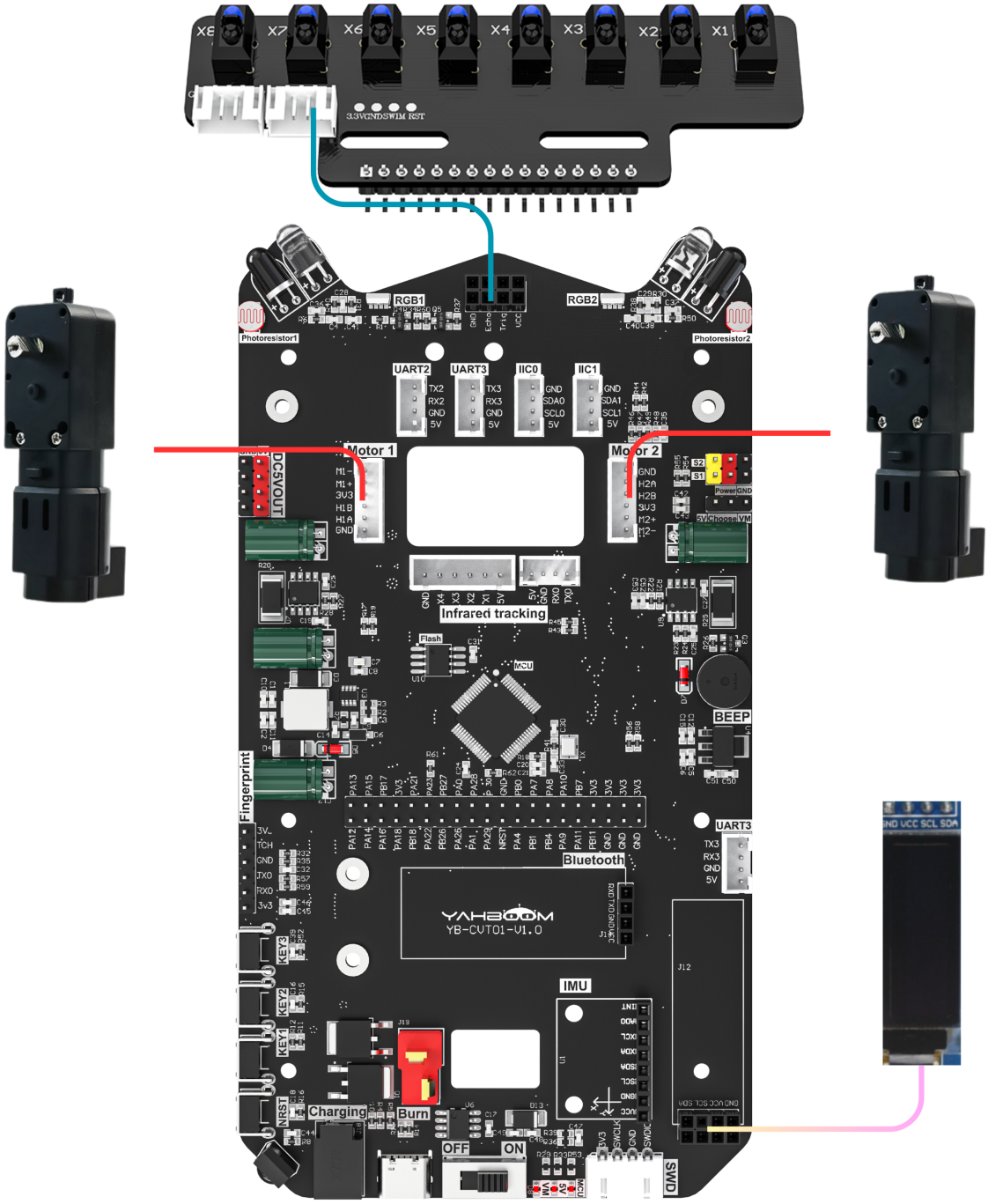
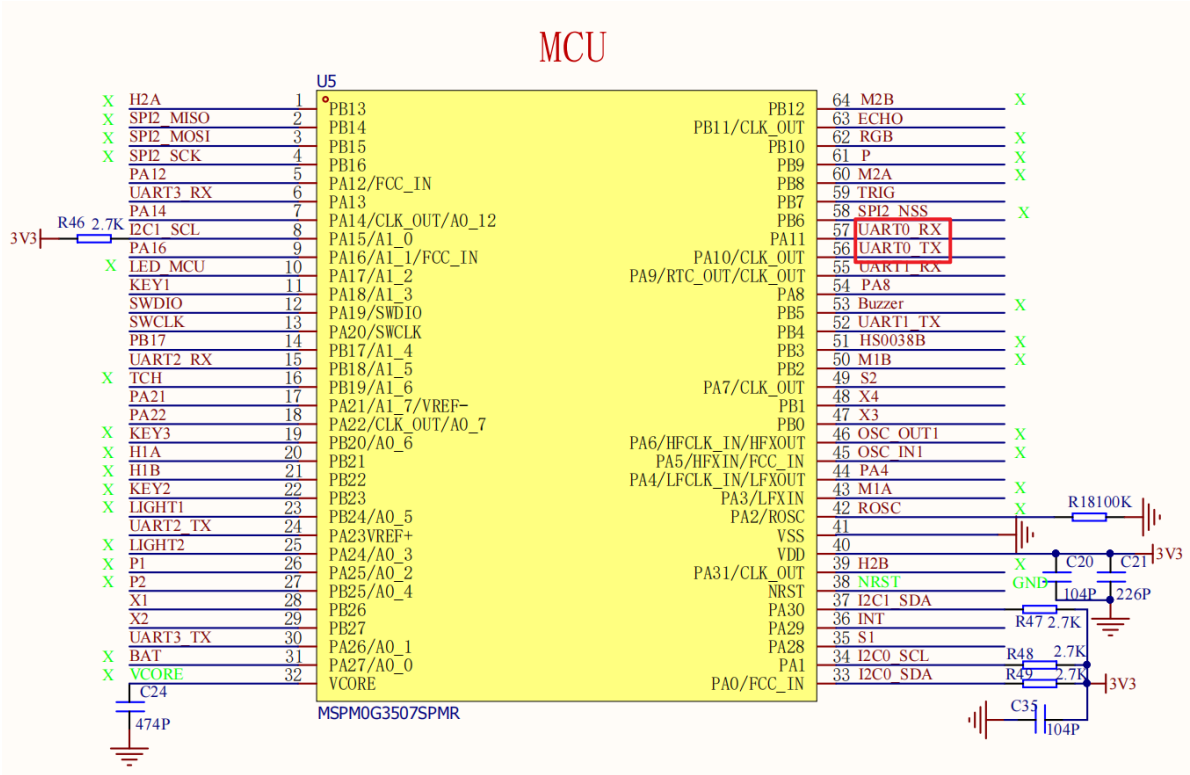
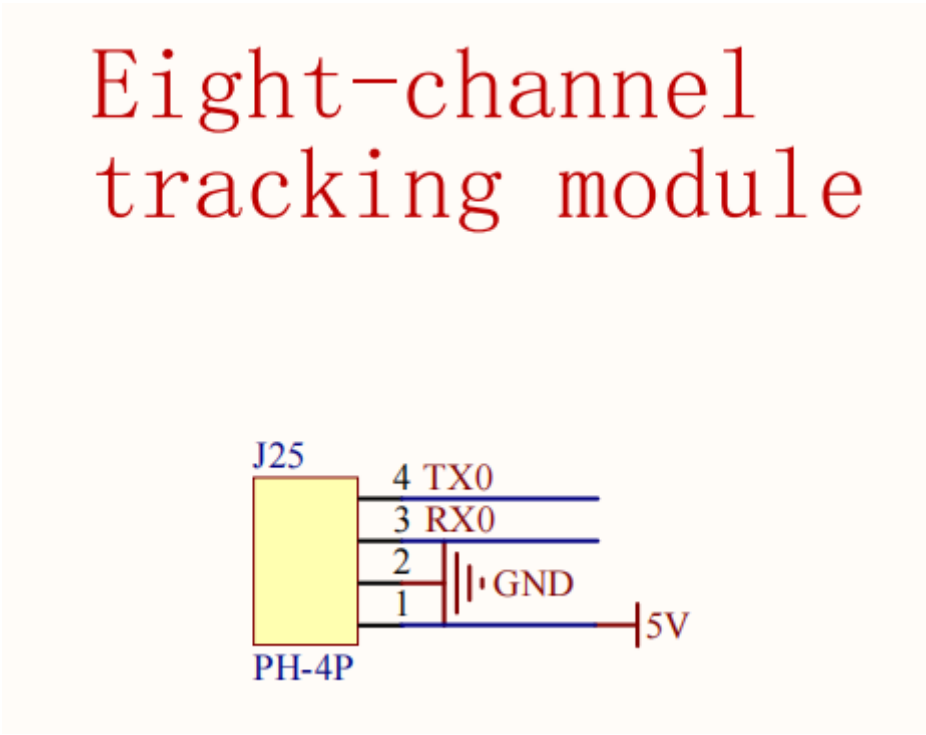
Drawing a Circle - 8 Channel Infrared Tracking Sensor Module

1. Hardware Connection

8 Channel Infrared Tracking Sensor Module	MSPM0G3507
5V	5V
GND	GND
TX	RX0
RX	TX0

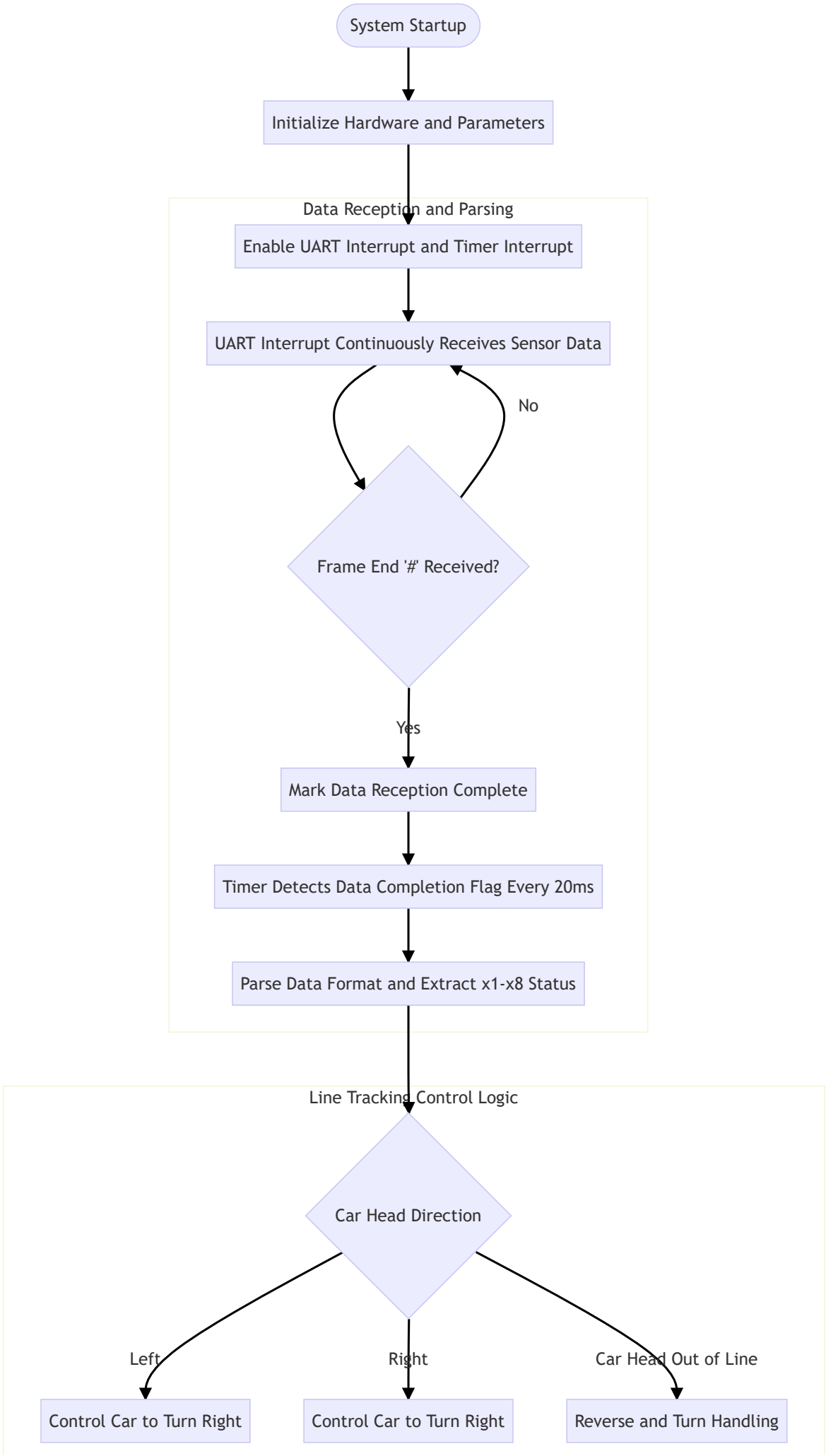

X8
X7
X6
X5
X4
X3
X2
X1
3.3VGNDSWIM RST
RGB1
RGB2
Photoresistor1
Photoresistor2
UART2
UART3
IIC0
IIC1
Motor 1
Motor 2
DC5VOUT
Infrared tracking
Flash
MCU
BEEP
Fingerprint
Bluetooth
YAHBOOM
YB-CUT01-V1.0
IMU
Charging
Burn
OFF
ON
SWD
KEY3
KEY2
KEY1
NRST





2. Partial Code Analysis

Control Principle


System Startup
Initialize Hardware and Parameters
Data Reception and Parsing
Enable UART Interrupt and Timer Interrupt
UART Interrupt Continuously Receives Sensor Data
No
Frame End '#' Received?
Yes
Mark Data Reception Complete
Timer Detects Data Completion Flag Every 20ms
Parse Data Format and Extract x1-x8 Status
Line Tracking Control Logic
Car Head Direction
Left
Right
Car Head Out of Line
Control Car to Turn Right
Control Car to Turn Right
Reverse and Turn Handling

bsp_ir.c

```
/**
 * @brief 红外数据解析函数 / IR data analysis function
 *数据格式如下 $D,x1:1,x2:0,x3:1,x4:0,x5:1,x6:0,x7:1,x8:0# / Data format is as
follows $D,x1:1,x2:0,x3:1,x4:0,x5:1,x6:0,x7:1,x8:0#
 */

void IRDataAnalysis(void)
{
    char temp[60] = {0};
    char *buff = NULL;
    int head = 0;
    int end = 0;


    // 没有接收到数据 或者 数据没有接收完成 则不进行处理 / If no data received or data
reception not complete, do not process
    if (IR_recv_complete_flag == 0) return;

    // 关中断，防止在解析过程中接收到新数据 / Disable interrupt to prevent receiving
new data during parsing
    __disable_irq();

    // 找到格式的头的第一个 '$' / Find the first '$' of the format header
    while ((IR_recv_buff[head] != '$') && (head < IR_recv_length))
    {
        head++;
    }
    if (head == IR_recv_length)
    {
        // 清除接收完成标志位，等待下一次接收 / Clear reception completion flag, wait
for next reception
        IR_recv_complete_flag = 0;
        IR_recv_length = 0;
        __enable_irq();
        return;
    }
    buff = &IR_recv_buff[head];

    // 找到结尾 '#' / Find the ending '#'
    while ((buff[end] != '#') && (end < IR_recv_length))
    {
```

```
        end++;
    }
    if ((head + end) == IR_recv_length)
    {
        // 清除接收完成标志位，等待下一次接收 / Clear reception completion flag, wait
for next reception
        IR_recv_complete_flag = 0;
        IR_recv_length = 0;
        __enable_irq();
        return;
    }

    // 复制数据到 temp 缓冲区 / Copy data to temp buffer
    if (end + 1 < sizeof(temp))
    {
        strncpy(temp, buff, end + 1);
        temp[end + 1] = '\0';  // 确保字符串终止 / Ensure string termination

        // 检查数据格式是否正确（$D,...） / Check if data format is correct ($D,...)
        if (temp[0] == '$' && temp[1] == 'D')
        {
            char *token = strtok(temp, ",");  // 分割字符串 / Split string
            int index = 0;

            while (token != NULL && index < IR_Num)
            {
                // 解析 x1:1, x2:1, ..., x8:1 / Parse x1:1, x2:1, ..., x8:1
                if (strstr(token, "x") != NULL)
                {
                    char *colon = strchr(token, ':');
                    if (colon != NULL)
                    {
                        IR_Data_number[index] = (colon[1] - '0');  // 提取 '0' 或
'1' / Extract '0' or '1'
                        index++;
                    }
                }
                token = strtok(NULL, ",");
            }

        }
    }

    // 清除接收完成标志位，等待下一次接收 / Clear reception completion flag, wait for
next reception
    IR_recv_complete_flag = 0;
    IR_recv_length = 0;
    memset(IR_recv_buff, 0, USART_RECEIVE_LENGTH);
    __enable_irq();
}
//串口中断中一直接收传感器返回的数据，然后再定时器中断中每20ms判断是否接收到了帧尾，接收到帧尾关
闭中断处理新的数据 / Continuously receive sensor returned data in UART interrupt,
then check every 20ms in timer interrupt if frame end is received, close
interrupt to process new data when frame end is received
void UART_0_INST_IRQHandler(void)
{
    uint8_t RecvDATA = 0;  // 接收的字节数据 / Received byte data
```

```
    // 检查中断来源 / Check interrupt source
    switch (DL_UART_getPendingInterrupt(UART_0_INST))
    {
        case DL_UART_IIDX_RX:
            RecvDATA = DL_UART_Main_receiveData(UART_0_INST);

            // 检查缓冲区是否已满 / Check if buffer is full
            if (IR_recv_length >= USART_RECEIVE_LENGTH - 1)
            {
                // 缓冲区满，重置接收状态（丢弃数据） / Buffer full, reset reception
state (discard data)
                IR_recv_complete_flag = 0;
                IR_recv_length = 0;
                break;
            }

            // 存储接收的数据到缓冲区 / Store received data to buffer
            IR_recv_buff[IR_recv_length++] = RecvDATA;
            IR_recv_buff[IR_recv_length] = '\0';

            // 收到 '#' 时标记接收完成 / Mark reception complete when '#' is
received
            if (RecvDATA == '#')
            {
                IR_recv_complete_flag = 1;
            }
            break;

        default:
            break;
    }
}
```

app_irtracking.c

```
//带死区处理的位置式PID / Positional PID with dead zone processing
float APP_IR_PID_Calc(int8_t actual_value)
{

    float IRTrackTurn = 0;
    int8_t error;
    static int8_t error_last=0;
    static float IRTrack_Integral;//积分 / Integral

    error=actual_value;

    IRTrack_Integral +=error;

    //位置式pid / Positional PID
    IRTrackTurn=error*IRTrack_Trun_KP
                            +IRTrack_Trun_KI*IRTrack_Integral
                            +(error - error_last)*IRTrack_Trun_KD;



    if (IRTrackTurn > (MAX_SPEED - MOTOR_DEAD_ZONE))
            IRTrackTurn = (MAX_SPEED - MOTOR_DEAD_ZONE);
```

```
    if (IRTrackTurn < (MOTOR_DEAD_ZONE - MAX_SPEED))
            IRTrackTurn = (MOTOR_DEAD_ZONE - MAX_SPEED);
    return IRTrackTurn;
}
//画地为牢即巡线反逻辑 / Drawing a circle is the reverse logic of line tracking
void LineWalking(void)
{
     int8_t err = 0;
            u8 x1,x2,x3,x4,x5,x6,x7,x8;

    deal_IRdata(&x1,&x2,&x3,&x4,&x5,&x6,&x7,&x8);

        if(x4 ==0 && x5 ==0)
        {
            if(trun_flag == 1)
            {
             err =100;

            }
            else
            err = -100;

            // 应用PID控制 这里根据实际情况调节 / Apply PID control, adjust according
to actual situation here
            pid_output_IRR = (int)(APP_ELE_PID_Calc(
            -err));
            Motion_Car_Control(-IRR_SPEED, 0, pid_output_IRR);
            delay_ms(500);
            return ;
        }
        else if(x1 == 0)
            { err = -20;
                trun_flag = 0 ;
        }
    else if(x1 == 0 && x8 == 1) err = -20;    // 最左 / Leftmost
        else if(x1 ==0 && x2==0) err =-50;
        else if(x7==0 && x8 ==0) err =50;
        else if(x7 == 0) err = 30;
        else if(x8 == 0)
            { err = 20;
                trun_flag = 1 ;
        }
            else err = 0;

    // 应用PID控制 / Apply PID control
    pid_output_IRR = (int)(APP_ELE_PID_Calc(
            -err));

    // 控制小车运动 / Control car movement
    Motion_Car_Control(IRR_SPEED, 0, pid_output_IRR);
}
```

3. Main Functions

LineWalking

Function Prototype	**void LineWalking(void)**
Function Description	Drawing a circle main function: reverse logic of line tracking, reads 8-channel sensor data, controls car to move in a circular pattern based on sensor detection conditions, uses APP_ELE_PID_Calc for control calculation, and controls car movement through Motion_Car_Control
Input Parameters	None
Return Value	None

4. Experimental Phenomenon

After connecting the car properly, connecting the OLED module, and burning the program to MSPM0, place the car on a white background with black lines map, the car will move in a circular pattern within the bounded area, and sensor data will be displayed on the OLED