Autonomous driving car

Autonomous driving car

1. Software-Hardware
2. Brief Principle
   - 2.1 Hardware Schematic Diagram
   - 2.2 Physical Connection Diagram
   - 2.3 Control Principle
3. Code Analysis
4. Experimental Operation
5. Experimental Phenomenon

This tutorial is a comprehensive experiment combining multiple peripherals. You can understand individual peripherals before conducting this experiment.

1. Software-Hardware

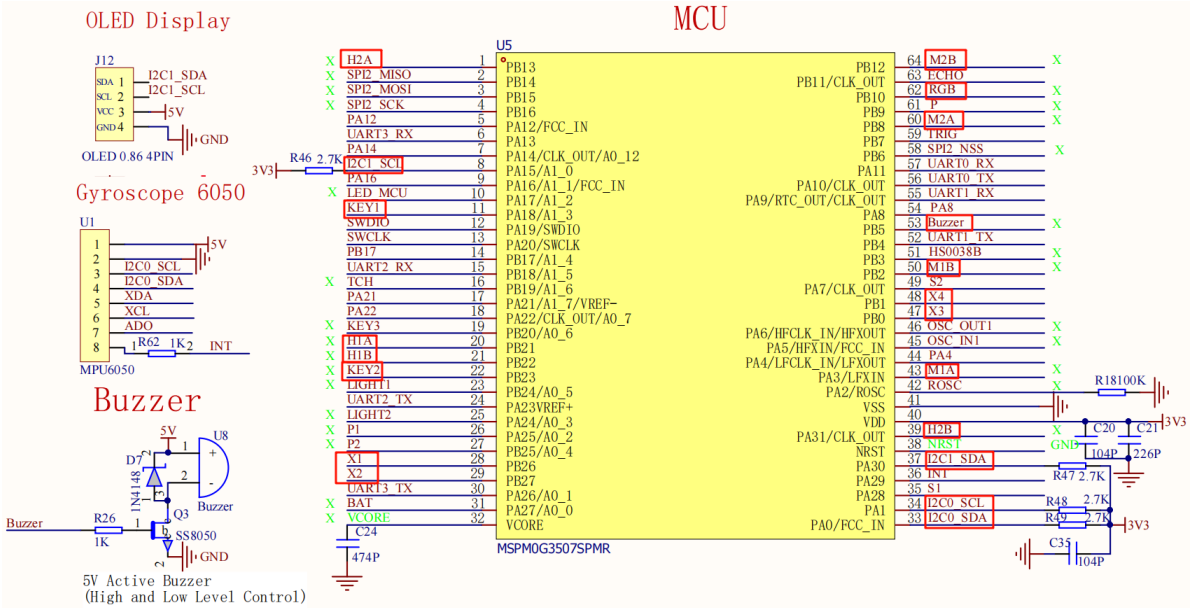
- **KEIL5**
- **MSPM0G3507 Robot Development Board**

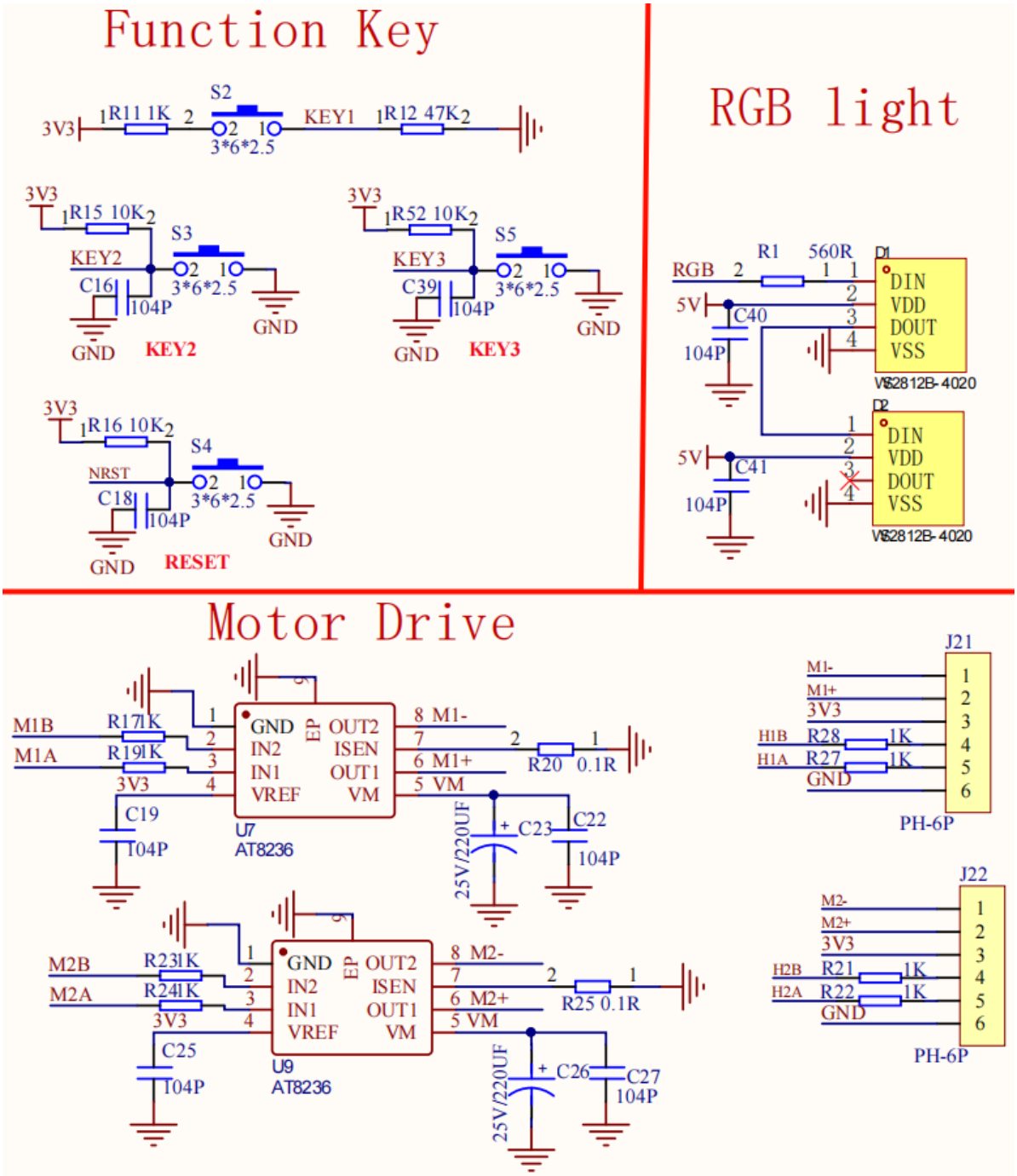
  MPU6050, buzzer, 4-channel infrared tracking sensor module, RGB, TT encoder motor
- **Type-C data cable or DAP-Link**

  For program download or simulation to the development board

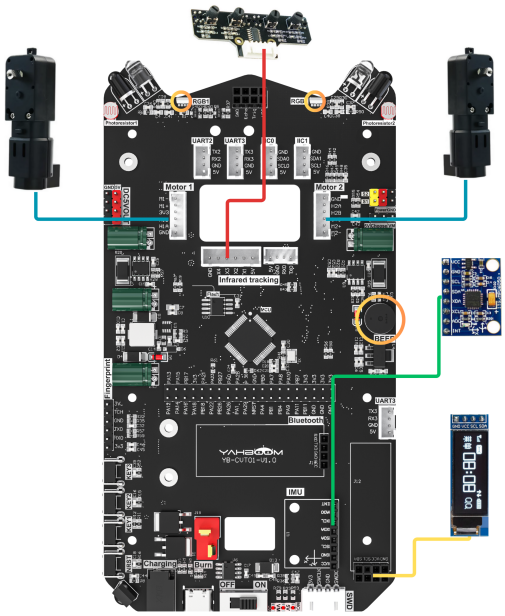
2. Brief Principle

2.1 Hardware Schematic Diagram



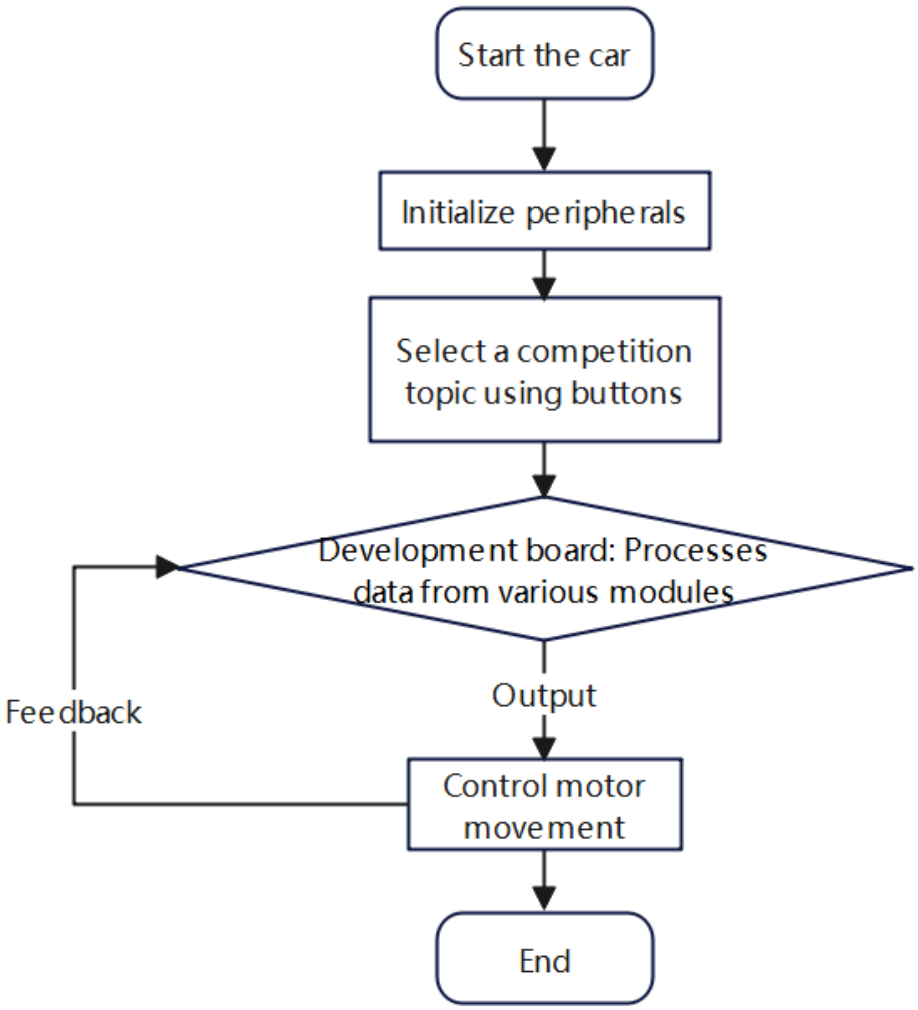


2.2 Physical Connection Diagram



2.3 Control Principle

- **Program Flowchart**



Control Principle:

PID Control

PID control is a common closed-loop control algorithm that minimizes the error between the actual output and expected output of the system by continuously adjusting the controller output.

PID Parameter	Function	Purpose
P (Proportional)	Determines the response speed of PID controller output	Accelerate system response speed
I (Integral)	Determines the response degree of PID controller to accumulated error	Eliminate steady-state error
D (Differential)	Determines the response degree of PID controller to error change rate	Improve system steady-state performance

3. Code Analysis

Main Function Explanation

> **Function: mode_4**

Function Prototype	void mode_4(void)
Function Description	**Task 4 cross straight line total function**
Input Parameter	**None**
Output Parameter	**None**

> > **Function: Scheduler_Run**
>
> | Function Prototype | void Scheduler_Run(void) |
> | --- | --- |
> | Function Description | **Small state machine configuration** |
> | Input Parameter | **None** |
> | Output Parameter | **None** |

- control.c

```
void mode_3(void)
{


        static float Second_yaw;
        static float first_yaw;

if(mode3_flag==0&&mode3_stop==0&&Line_flag==0&&yaw_flag==0)
    {
        delay_ms(1000);
        first_yaw =calibratedYaw;
      object_yaw  = navigetion_0_360_limit(first_yaw-38);
```

```
        mode3_flag=1;
    }

else if(mode3_flag==1&&mode3_stop==0&&Line_flag==0&&yaw_flag==0)
    {




        balance_yaw =get_minor_arc(object_yaw,calibratedYaw);
        yaw_out=Dir_PID(balance_yaw);
        Set_PID_Motor(53 ,53,yaw_out);
        mode3_stop= LineStop();
    }

else if(mode3_flag==1&&mode3_stop==1&&Line_flag==0&&yaw_flag==0)
{

    yaw_out=0;
    object_yaw=0;

     Motor_Stop(1) ;


    Buzzer_open_state();
    Control_RGB_ALL(Cyan_RGB);

    delay_ms(10);
    Buzzer_close_state();
     Control_RGB_ALL(OFF);
    Line_flag =1;

}



else if(mode3_flag==1&&mode3_stop==1&&Line_flag==1&&yaw_flag==0)
{


    LineWalking(60,60,1);
    line_stop=LineStop();
    if((calibratedYaw < 200 &&  calibratedYaw > 135)&&line_stop==0)
    {
    yaw_flag=1;
             Motor_Stop(1) ;

                        Buzzer_open_state();
        Control_RGB_ALL(Cyan_RGB);

        delay_ms(10);
        Buzzer_close_state();
        Control_RGB_ALL(OFF);
        delay_ms(1000);
    Second_yaw=calibratedYaw;
    mode3_stop= 0;
```

```
    object_yaw  = navigetion_0_360_limit(Second_yaw+36);




    }else
    {

        yaw_flag=0;

    }

}




else if(mode3_flag==1&&mode3_stop==0&&Line_flag==1&&yaw_flag==1)
{

        LineWalking(60,60,0);
        balance_yaw =get_minor_arc(object_yaw,calibratedYaw);
        yaw_out2=Dir_PID(balance_yaw);

        Set_PID_Motor(50 ,50,yaw_out2);
        encoder_odometry_flag = 1;
        line_stop=LineStop();
        if(odometry_sum>30&&line_stop==1)
        {
        Motor_Stop(1) ;
        Buzzer_open_state();
        Control_RGB_ALL(Cyan_RGB);

        delay_ms(10);
        Buzzer_close_state();
        Control_RGB_ALL(OFF);
        yaw_flag=2;
        mode3_stop=1;


        }


}
else if(mode3_flag==1&&mode3_stop==1&&Line_flag==1&&yaw_flag==2)
{
    object_yaw=0;
    yaw_out2=0;
    encoder_odometry_flag = 0;
    odometry_sum=0;


    yaw_flag=3;

}
```

```
else if(mode3_flag==1&&mode3_stop==1&&Line_flag==1&&yaw_flag==3)
{

        line_stop=LineStop();
        encoder_odometry_flag = 1;
    if(odometry_sum>30&&line_stop==0)
    {



        Motor_Stop(1) ;



    }



    else{

        LineWalking(60,60,1);

    }
}



}
```

In mode_3: `object_angle = navigetion_0_360_limit(first_angle - 38);` sets the angle the car needs to rotate when running. Here it's the starting angle - 38°. If 38° cannot reach the corresponding position during your own run, modify the number 38 according to your car's offset. Left turn is subtraction, right turn is addition.

`if(odometry_sum>30&&line_stop==0)` uses the line tracking sensor to detect whether the black line has been reached, and whether the odometry count is greater than or equal to 30. If the track conditions are poor and the line tracking sensor triggers falsely, causing early entry to the next state, you can increase `odometry_sum`.

`(calibratedYaw < 200 &&  calibratedYaw > 135)&&line_stop==0` uses the angle from the MPU6050 to determine whether the car is in the 135°-200° range and has reached the white background area. But sometimes when the car reaches this point, the angle may exceed this range, preventing the car from entering the next state. In this case, you can choose to modify the angle values here. The specific modification amount needs to be tested by yourself.

- get_mpu6050.c

```
//获取已校准的角度  Get the calibrated angle
void Get_CalibratedAngles(void)
{



    if(Filter_out < -2)
```

```
    {
        calibratedYaw = -Filter_out;
    }
    else if(Filter_out >= 2)
    {
        calibratedYaw = 360 - Filter_out; //车头朝前yaw顺时针为负数  逆时针为正数 //
Car front facing, yaw clockwise is negative, counterclockwise is positive
    }
    else
    {
        calibratedYaw = 0;  //过滤掉0和360附近的数据 // Filter out data near 0 and
360

    }

}
```

Used to convert yaw data to 0-360 degree range

- bsp_PID_motor.c

```
// PID单独计算一条通道 PID calculates one channel separately
float PID_Calc_One_Motor(uint8_t motor_id, float now_speed)
{
    if (motor_id >= MAX_MOTOR)
        return 0;

        return PID_Location_Calc(&pid_motor[motor_id], now_speed);


}

void Set_PID_Motor(float set_l ,float set_r,float turn_out)
{


        l_pid_out = PID_Calc_One_Motor(0, set_l);
        r_pid_out = PID_Calc_One_Motor(1, set_r);

        PWM_Control_Car(l_pid_out+turn_out , r_pid_out-turn_out );

}

// 位置式PID计算方式 Positional PID calculation method
float PID_Location_Calc(PID_t *pid, float actual_val)
{
    /*计算目标值与实际值的误差 Calculate the error between target value and actual
value*/
    pid->err = pid->target_val - actual_val;
    /* 限定闭环死区 Limit closed-loop dead zone */
    if ((pid->err >= -40) && (pid->err <= 40))
    {
        pid->err = 0;
        pid->integral = 0;
```

```
    }

    /* 积分分离，偏差较大时去掉积分作用 Integral separation, remove integral action
when deviation is large */
    if (pid->err > -1500 && pid->err < 1500)
    {
        pid->integral += pid->err; // 误差累积 Error accumulation

        /*限定积分范围，防止积分饱和 Limit integral range to prevent integral
saturation */
        if (pid->integral > 4000)
            pid->integral = 4000;
        else if (pid->integral < -4000)
            pid->integral = -4000;
    }

    /*PID算法实现 PID algorithm implementation*/
    pid->output_val = pid->Kp * pid->err +
                      pid->Ki * pid->integral +
                      pid->Kd * (pid->err - pid->err_last);


    /*误差传递 Error transmission*/
    pid->err_last = pid->err;

    /*返回当前实际值 Return current actual value*/
    return pid->output_val;
}
```

Set_PID_Motor: Sets the target values for 2 motors and the steering loop output value.

PID_Calc_One_Motor: Calculates the position loop PID value for one channel and returns the calculated value.

PID_Location_Calc: Position loop PID calculation formula and complete position loop structure.

- main.c

```
    bsp_init();//需要的外设初始化  Required peripheral initialization




    Control_RGB_ALL(OFF);
    delay_ms(100);

            OLED_ShowString(0,0,"yaw:",8,1);
            OLED_Refresh();

                get_mode =  switch_mode();
//
    while (1)
    {
```

```
            Scheduler_Run();

            switch(get_mode){
                case 1:{
                    mode_1();
                    break;
                }
                case 2:{
                    mode_2();
                    break;
                }
                case 3:{
                     mode_3();
                    break;
                }
                case 4:{
                 mode_4();
                    break;
                }
            }
//

            OLED_ShowSNum(75,0, calibratedYaw, 3, 8, 1);

                OLED_Refresh();
```

First initialize all peripherals, and after success, MPU6050 detection is ok. Then run the task scheduler to obtain the required data in sequence. The while loop continuously determines whether a competition topic has been selected through button press.

4. Experimental Operation

1. Connect according to the hardware connection instructions in the tutorial.
2. Open the project used by MSPM0G3507, then compile and burn, finally reset or power on again.
3. Place the car on the competition map, then power on, the car enters initialization.
4. After initialization is complete, enter the mode selection interface, displayed on the OLED.
5. After selecting the mode, the car will automatically coordinate various modules to start moving.

5. Experimental Phenomenon

After the car is powered on. The peripherals used will perform initialization operations. You need to wait for a while, after MPU6050 initialization is successful, OK will be displayed on the OLED interface, then the OLED will display numbers. Press the K1 button to select the competition topic. The corresponding number represents the corresponding competition topic. After selecting the topic, first press and hold K1 without releasing, then press K2 to start running the task.