K230 Autonomous Avoidance

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

1. Software-Hardware Requirements

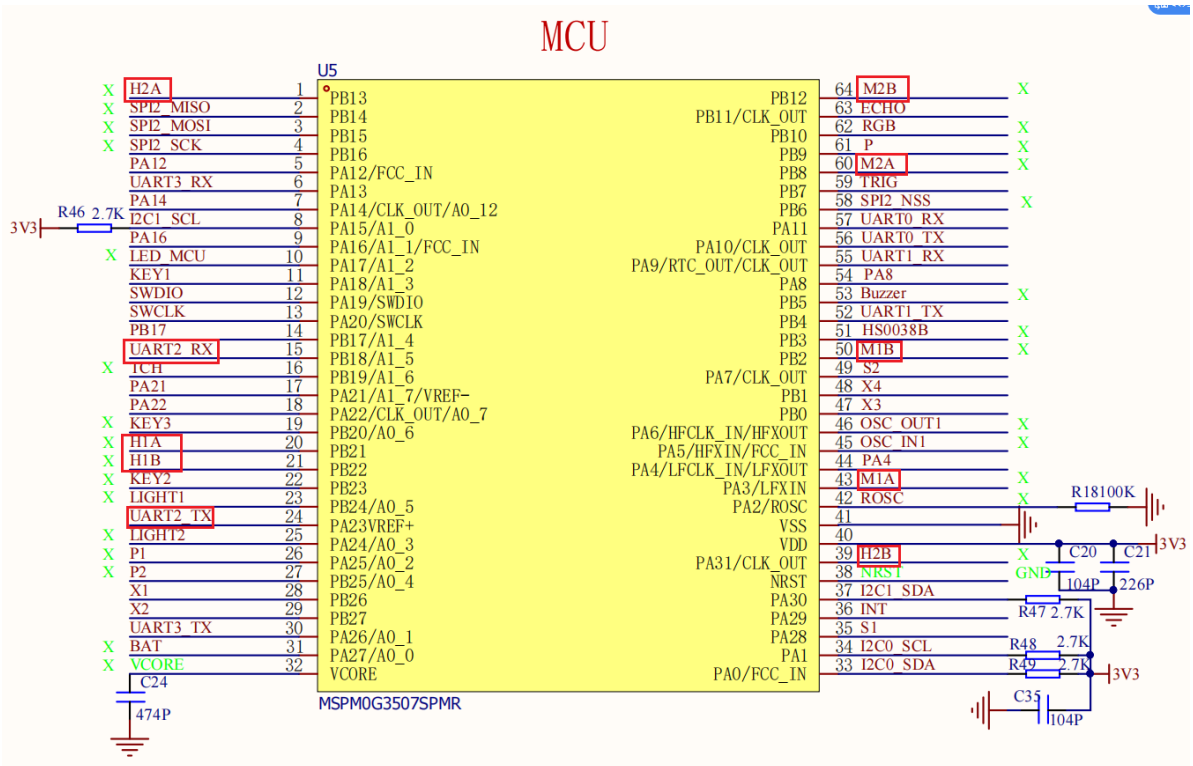
- **KEIL5**
- **MSPM0G3507 Robot Development Board**

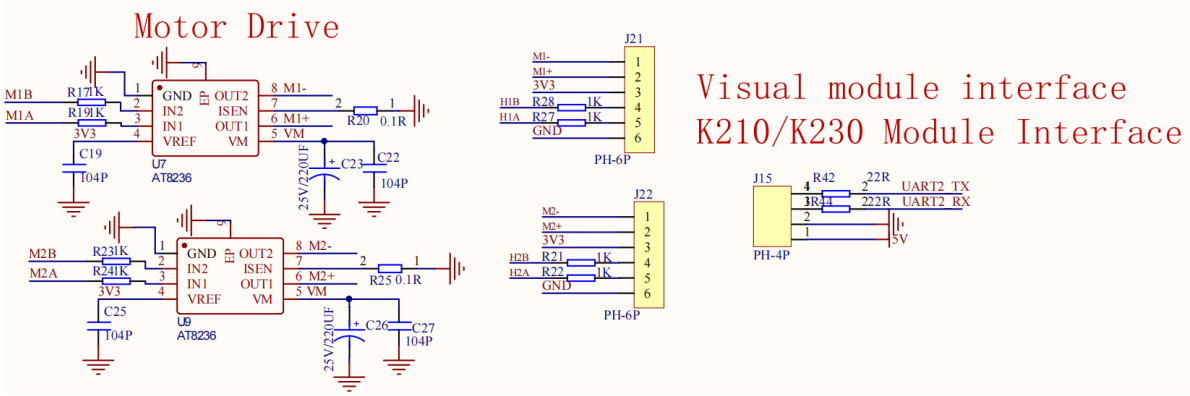
  K230 Module: External
- **Type-C Data Cable or DAP-Link**

  For program download or simulation to the development board

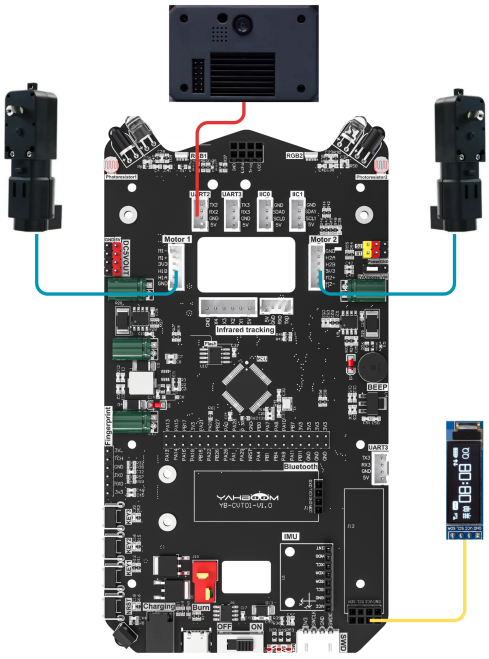
2. Brief Principle

2.1 Hardware Schematic Diagram





2.2 Physical Connection Diagram



Wiring

Motor Wiring (Note: The wiring in the diagram below is for position reference only. We provide dual-head PH2.0 6Pin all-black cables with anti-mistake design, so you don't need to worry about wiring issues)

TT Encoder Motor	MSPM0G3507
Motor Wire-	M1-
Motor Wire+	M1+
Encoder Power	3V3
Encoder Output B Phase	H1B
Encoder Output A Phase	H1A
Encoder Ground	GND

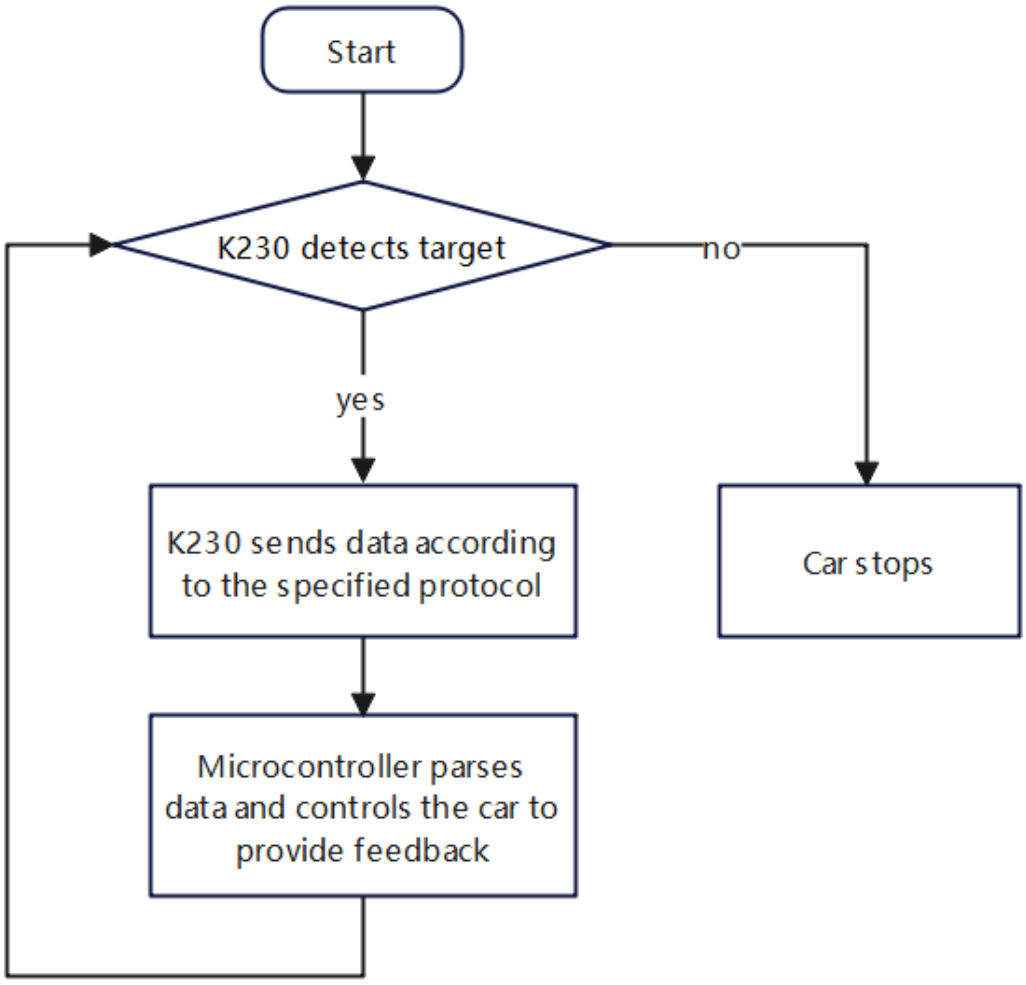
TT Encoder Motor	MSPM0G3507
Encoder Ground	GND
Encoder Output A Phase	H2A
Encoder Output B Phase	H2B
Encoder Power	3V3
Motor Wire+	M2+
Motor Wire-	M2-

K230 Wiring (Note: The wiring in the diagram below is for position reference only. We provide K230 dual-head PH2.0 4Pin all-black cables with anti-mistake design, so you don't need to worry about wiring issues)

K230	MSPM0G3507
RX	TX2
TX	RX2
GND	GND
5V	5V

2.3 Control Principle

- **Program Flow Chart**



Control Principle:

PID Control

PID control is a commonly used closed-loop control algorithm that continuously adjusts the controller output to minimize the error between the system's actual output and expected output.

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

Module	Function
k230 Vision Module	Recognize the object you want to identify, then send data to the microcontroller via serial port

Communication Protocol:

When k230 recognizes the color track, it will send a data frame via serial port with the following format

Experiment Routine	Start Symbol	Length	Separator	Routine Number	Separator	x	Separator	y	Separator	w	Separator	h	End Symbol
Color Recognition	$	XX	,	16	,	XXX	,	XXX	,	XXX	,	XXX	#

Microcontroller part:

Receive data sent by k230 via serial port and process each byte of data. Extract valid data from the data frame content and store it in an array. Then perform PID processing on the array data, and the microcontroller drives the motors based on the processed data to move forward and stop.

3. Code Analysis

Main Function Analysis

> **Function: Pto_Loop**

Function Prototype	void Pto_Loop(void)
Function Description	Main function for running the program
Input Parameters	None
Output Parameters	None

Partial Function Analysis

- main.c

```
ColorMode color_mode = 1;  //1: Patrol, 2: Tracking

//Select whether the car moves forward automatically after initialization, 0 for no, 1 for yes (used in autonomous avoidance case)


int Car_Auto_Drive = 0;

int main(void)
{

    bsp_init();//需要的外设初始化 / Required peripheral initialization
        Control_RGB_ALL(OFF);//关闭RGB / Turn off RGB
    delay_ms(100);
 OLED_ShowString(0,0,"pid_output:",8,1);//显示pid输出用于控制左右 / Display pid output for controlling left and right
    OLED_ShowString(0,20,"pid_output1:",8,1);//显示pid输出用于控制前后 / Display pid output for controlling forward and backward
    OLED_Refresh();


    while (1)
    {
```

```
            Pto_Loop();
            OLED_ShowSNum(75,0, pid_output, 3, 8, 1);
            OLED_ShowSNum(75,20,pid_output1,3,8,1);
            OLED_Refresh();

    }
}
```

Car_Auto_Drive：Select whether the car moves forward automatically after initialization, 0 for no, 1 for yes (used in autonomous avoidance case)

BSP_init：Hardware initialization

pto__Loop：Continuously processes received K230 messages.

- yb_protocol.c

```
/**
 * @Brief: 数据分析 Data analysis
 * @Note:
 * @Parm: 传入接受到的一个数据帧和长度    Pass in a received data frame and length
 * @Retval:
 */
void Pto_Data_Parse(uint8_t *data_buf, uint8_t num)
{
    uint8_t pto_head = data_buf[0];
    uint8_t pto_tail = data_buf[num-1];

    //校验头尾  Check head and tail
    if (!(pto_head == PTO_HEAD && pto_tail == PTO_TAIL))
    {

        return;
    }

    uint8_t data_index = 1;
    uint8_t field_index[PTO_BUF_LEN_MAX] = {0};
    int i = 0;
    int values[PTO_BUF_LEN_MAX] = {0};
    char msgs[][PTO_BUF_LEN_MAX] = {0};

    //分割字段  Split fields
    for (i = 1; i < num-1; i++)
    {
        if (data_buf[i] == ',')
        {
            data_buf[i] = 0;
            field_index[data_index] = i;
            data_index++;
        }
    }

    //解析长度与功能ID  Parse length and function ID
    for (i = 0; i < 2; i++)
    {
```

```
        values[i] = Pto_Char_To_Int((char*)data_buf+field_index[i]+1);
    }

    uint8_t pto_len = values[0];
    uint8_t pto_id = values[1];


 ParseCommonFields(pto_id,data_buf,pto_len,field_index,data_index,values,msgs);

}
```

Parse the received protocol-compliant data to obtain the function ID number and data information in the data, and execute the corresponding function processing function. This project uses a metadata parsing method. The core idea is to separate the protocol format description (field position, type, etc.) from the parsing logic, define the protocol format through data structures (such as struct arrays), and the parsing function automatically processes data according to these descriptions.

Different types of data contained in the protocol can also be parsed. If you want to understand the implementation details, you can study it yourself. No excessive explanation will be provided. This function has been adapted to all K230 communication protocols in our store. You don't need to fully understand it to use it normally.

- bsp_PID_motor.c

```
// PID calculation for a single channel
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 );

}

// 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 error between target value and actual
value*/

    pid->err = pid->target_val - actual_val;

    /* 限定闭环死区 */
    /*Limited 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
effect 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 the integration range to prevent
integration 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：Set target values for 2 motors and steering loop output value.

PID_Calc_One_Motor：Calculate position loop PID value for one channel and return the calculated value.

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

- bsp_K230_control.c

```
void Autonomous_Avoid(int w,int h)
{
    target_area = 200;

    int now_area = w*h/100;
```

```
    if(now_area >= target_area)
    {
        Set_PID_Motor(0, 0, 0);
    }
    delay_ms(100);
}
```

Autonomous_Avoid：When an obstacle is recognized and the distance is close enough that the area is greater than the set target_area, the car will stop.

4. Experimental Operation

1. Wire according to the hardware wiring instructions in the tutorial.
2. Open the project used by MSPM0G3507. First, check the beginning part of the main.c file and modify Car_Auto_Drive to 1. This enables automatic obstacle avoidance mode. After correct modification, compile and burn the program. Finally, reset or power on again.
3. Place Autonomous_Avoid.py in the root directory of K230's SD card and rename it to main.py, then reset or power on again.

If you are not clear about how to burn programs into K230, please first check the K230 Vision Module tutorial's [Quick Start] chapter.

4. Place the car in an open area, adjust the K230 module bracket to a suitable angle, and connect the power.
5. When K230 initialization is successful, it will start recognizing whether there are obstacles. Then press the car reset button, wait for three seconds, and the car will start recognizing obstacles and performing autonomous avoidance.

5. Experimental Phenomena

Turn on the power switch, wait for a few seconds, after system initialization completes, the car will move forward automatically. If an obstacle is recognized and the distance is close, the car will stop by itself.

Obstacle traffic cone: