K230 Gaze Direction Tracking

This tutorial is a comprehensive experiment combining multiple peripherals. It is recommended to understand individual peripherals before proceeding with this experiment.

1. Software-Hardware Requirements

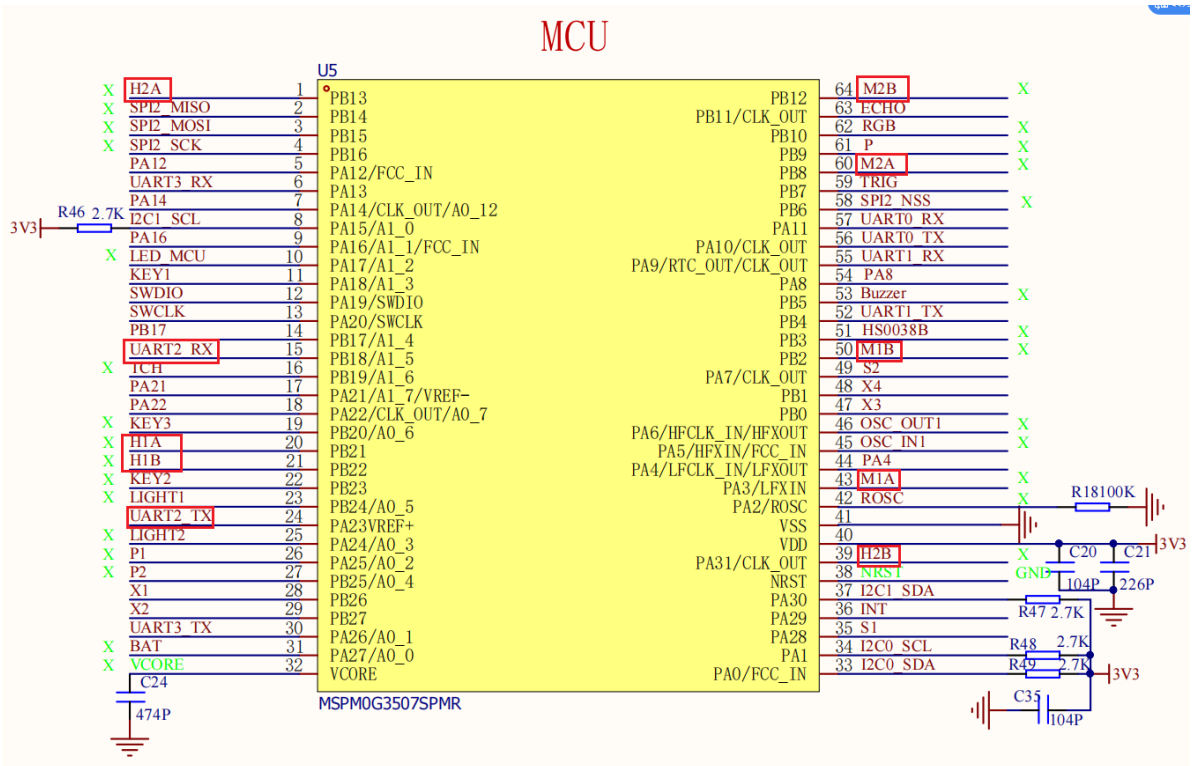
- **KEIL5**
- **MSPM0G3507 Robot Development Board**

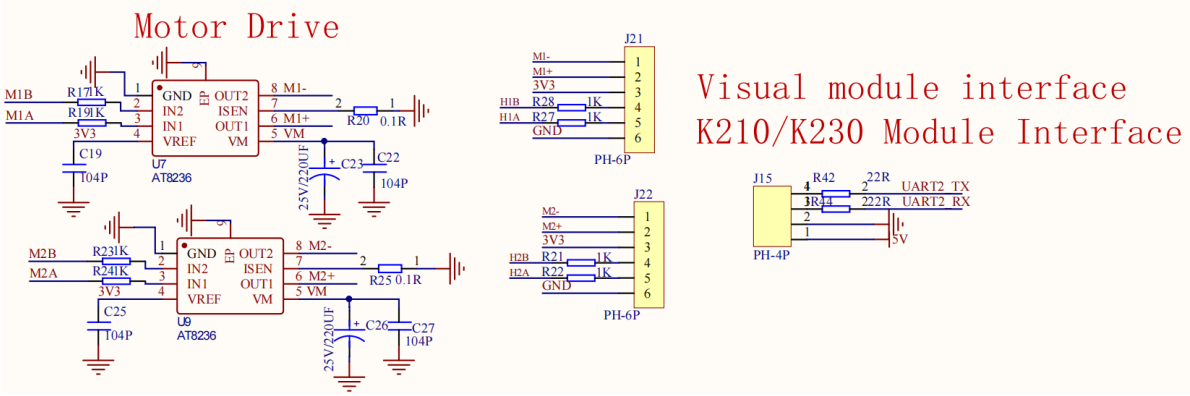
  K230 module: External connection
- **Type-C Data Cable or DAP-Link**

  For program download or simulation to the development board

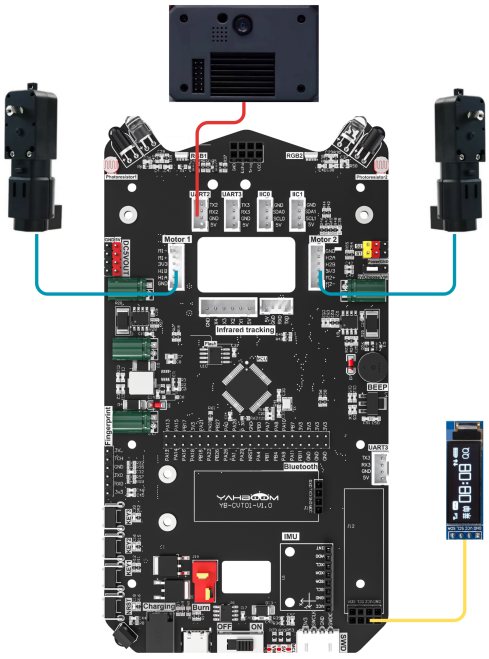
2. Basic Principles

2.1 Hardware Schematic Diagram





2.2 Physical Connection Diagram



Wiring Pins

Motor Wiring (Note: The wiring in the diagram below is for position reference only. Our factory provides 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

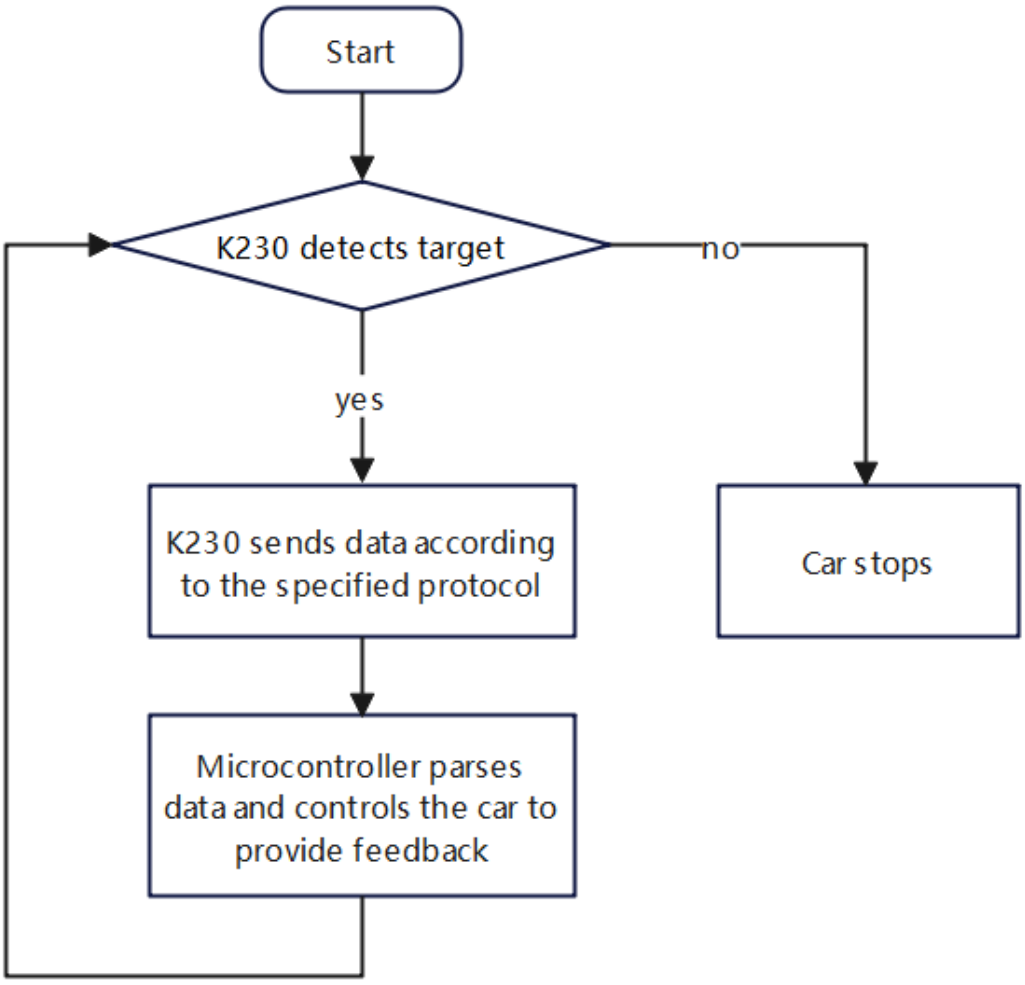
TT Encoder Motor	MSPM0G3507
Encoder Ground	GND
Encoder Output A Phase	H2A
Encoder Output B Phase	H2B
Encoder Power	3V3
Motor Wire +	M2+
Motor Wire -	M2-

K230 Wiring (Note: The wiring in the diagram below is for position reference only. Our factory provides 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 Flowchart**



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 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 errors	Eliminate steady-state errors
D (Differential)	Determines the response degree of PID controller to error change rate	Improve system steady-state performance

Module	Function
K230 Vision Module	Recognize gaze direction, then send data to microcontroller via serial port

Communication Protocol:

When K230 recognizes the gaze direction, 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	,	07	,	XXX	,	XXX	,	XXX	,	XXX	#

Microcontroller part:

Receives data sent by K230 via serial port and processes each byte of data. Extracts valid data from the data frame content and stores it in an array. Then performs PID processing on the array data, and the microcontroller drives the motor movement using the processed data.

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 = 2;  //1: Patrol line, 2: Tracking


int main(void)
{

    bsp_init(); // Required peripheral initialization
    Control_RGB_ALL(OFF); // Turn off RGB
    delay_ms(100);
    OLED_ShowString(0,0,"pid_output:",8,1); // Display PID output for left-right control
    OLED_ShowString(0,20,"pid_output1:",8,1); // Display PID output for forward-backward control
    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();

    }
```

```
}
```

color_mode: Use 1 when running vision patrol line case, use 2 when running tracking case

BSP_init: Hardware initialization

pto__Loop: Continuously processes received K230 messages.

- yb_protocol.c

```
/**
 * @Brief: Data analysis
 * @Note:
 * @Parm: 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))
    {
        //DEBUG_PRINT("pto error:pto_head=0x%02x , pto_tail=0x%02x\n", pto_head,
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 Field
    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  Parsing 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);

}
```

Parses received data that conforms to the protocol, obtains the function ID and data information from the data, and executes the corresponding function handler. This project uses a metadata parsing method. The core idea is to separate the protocol format description (field positions, types, etc.) from parsing logic, defining protocol format through data structures (like struct arrays), and parsing functions automatically process data according to these descriptions.

Different types of data in the protocol can also be parsed. If you want to understand implementation details, you can study it yourself; no excessive explanation will be provided. This function has been adapted for all K230 communication protocols in our store, so it can be used normally without complete understanding.

- 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
//Position 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 the target value and the 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, removing the integral effect when the deviation is
large*/
    if (pid->err > -1500 && pid->err < 1500)
    {
        pid->integral += pid->err; // error accumulation 误差累积

        /* Limit the integration range to prevent integration saturation 限定积分
范围，防止积分饱和 */
        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*/ /*Returns the current actual value*/
    return pid->output_val;
}
```

Set_PID_Motor: Sets target values for 2 motors and the turning 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.

- bsp_K230_control.c

```
void GazeDire_Track(int x_start, int x_end)
{

    PID_TypeDef k230_pid;
    PID_param_init(&k230_pid);

    set_p_i_d(&k230_pid, k230_p3, 0, k230_d3);
    int err = x_start - x_end;
    if(err<30 && err>-30)
    {
        err=0;
        Motor_Stop(1) ;
    }
```

```
    // 边界检查：禁止任何越界方向的转向 Boundary check: any turn in the out-of-bounds
direction is prohibited
    bool allow_motion = true;  // 默认允许运动  Movement is allowed by default

    if (x_start < 270 && err < 0) {
        // 左边界且需要向右转 → 禁止    Left boundary and need to turn right → No
//        DEBUG_PRINT("STOP: x_start=%d < 270, err=%d (no right turn)\r\n",
x_start, err);
        allow_motion = false;
    }
    else if (x_start > 400 && err > 0) {
        // 右边界且需要向左转 → 禁止    Right boundary and need to turn left → No
//        DEBUG_PRINT("STOP: x_start=%d > 400, err=%d (no left turn)\r\n",
x_start, err);
        allow_motion = false;
    }

    // 根据条件执行动作 Perform actions based on conditions
    if (allow_motion && (err > 30 || err < -30)) {
        pid_output = PID_Calculate(&k230_pid,err);

        // 限制输出范围 Limit output range
        if (pid_output > 70) pid_output = 70;
        if (pid_output < -70) pid_output = -70;

//        DEBUG_PRINT("pid_output:%d\r\n", pid_output);
//        DEBUG_PRINT("\r\n");
        Set_PID_Motor(0, 0, pid_output);
    }
    else if (!allow_motion) {
           Motor_Stop(1) ;
    }
}
```

GazeDire_Track: Calculates the deviation between the start and end positions of gaze direction, used to control the car's in-place rotation, and limits the PID output value and car rotation range to avoid the car being unable to continue recognizing gaze direction due to excessive rotation.

4. Experimental Operation

1. Connect hardware according to the tutorial's wiring instructions.
2. Open the project used by MSPM0G3507. First, check the beginning part of the main.c file and modify color_mode to 2. This enables tracking mode. After correct modification, compile and burn the program. Finally, reset or power on again.
3. Place GazeDire_Track.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 sure how to flash programs to 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 supply.
5. When K230 initialization is successful, it will start calculating the deviation between the start and end positions of gaze direction. Then press the car's reset button, wait for three seconds, and the car will start following.

Note:

If the tracking effect is not satisfactory, you can modify the PID values in the bsp_PID_motor.c file of the APP project.

```
#define k230_p3   (000.3f)
#define k230_i3   (0.00f)
#define k230_d3   (00.001f)
```

5. Experimental Phenomena

Turn on the power switch, wait for system initialization to complete. The screen will display the camera feed and recognize the human eye's gaze direction. When a person faces the car and looks to the right, the car will also rotate to your right. When looking to the left, the car will rotate back to the left. However, the rotation angle will not be particularly large. This is to ensure that when changing gaze direction, K230 can still recognize it and the car can respond accordingly.