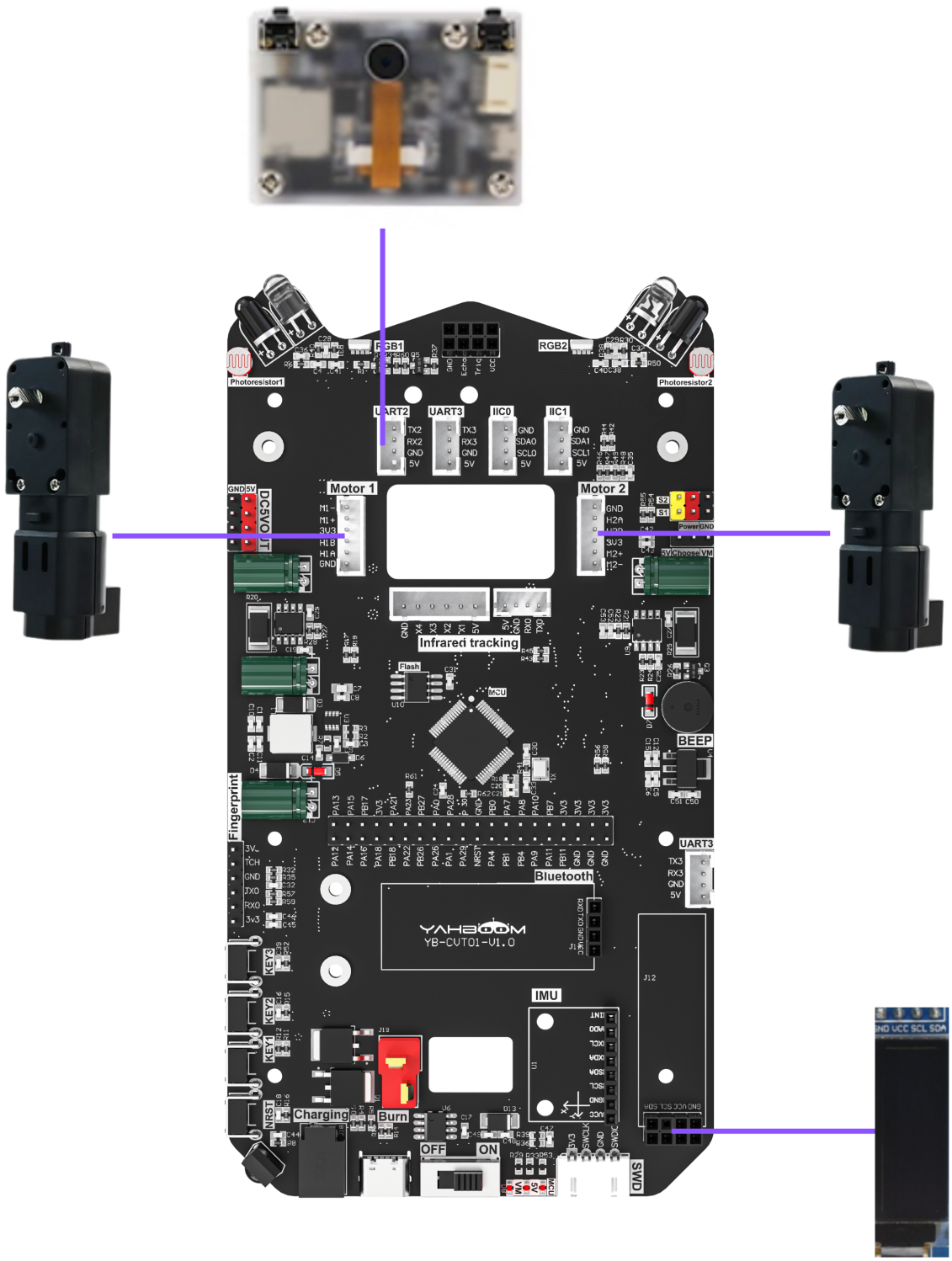
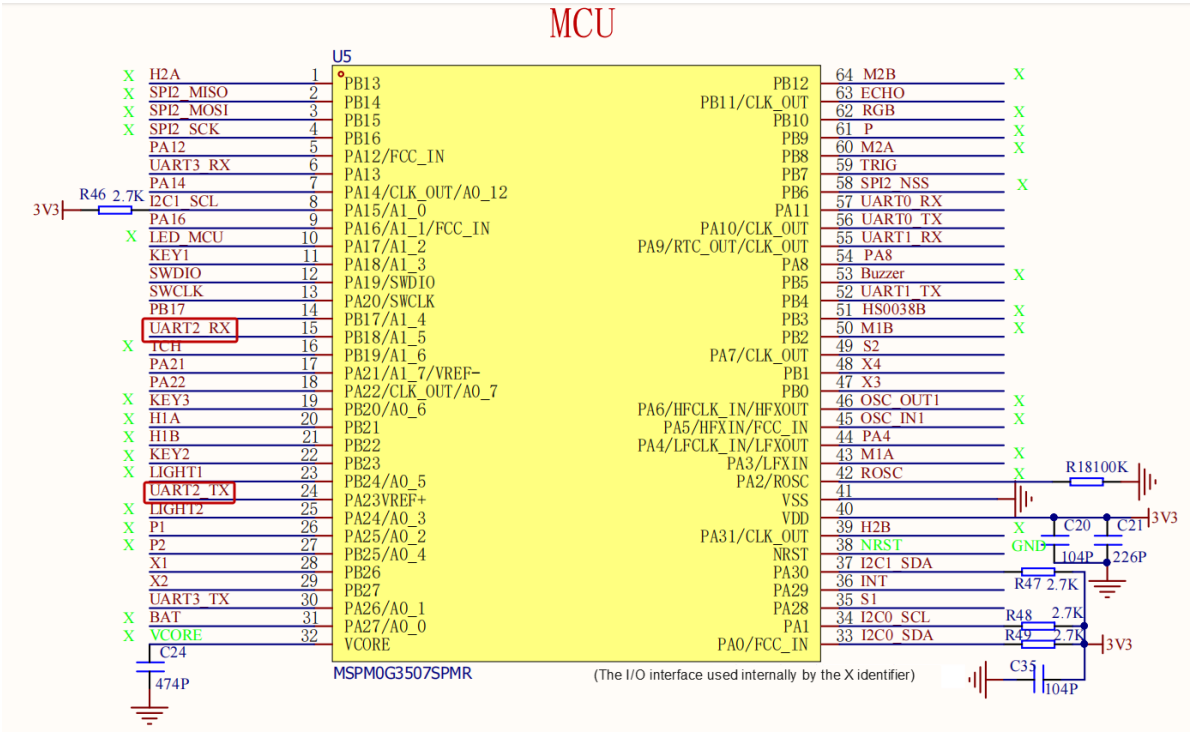
K210 QR Code Command

1. Hardware Connection

K210 Vision Module	MSPM0G3507
5V	5V
GND	GND
TX	RX2
RX	TX2

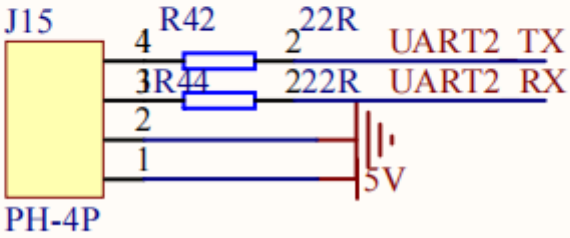
Physical Connection

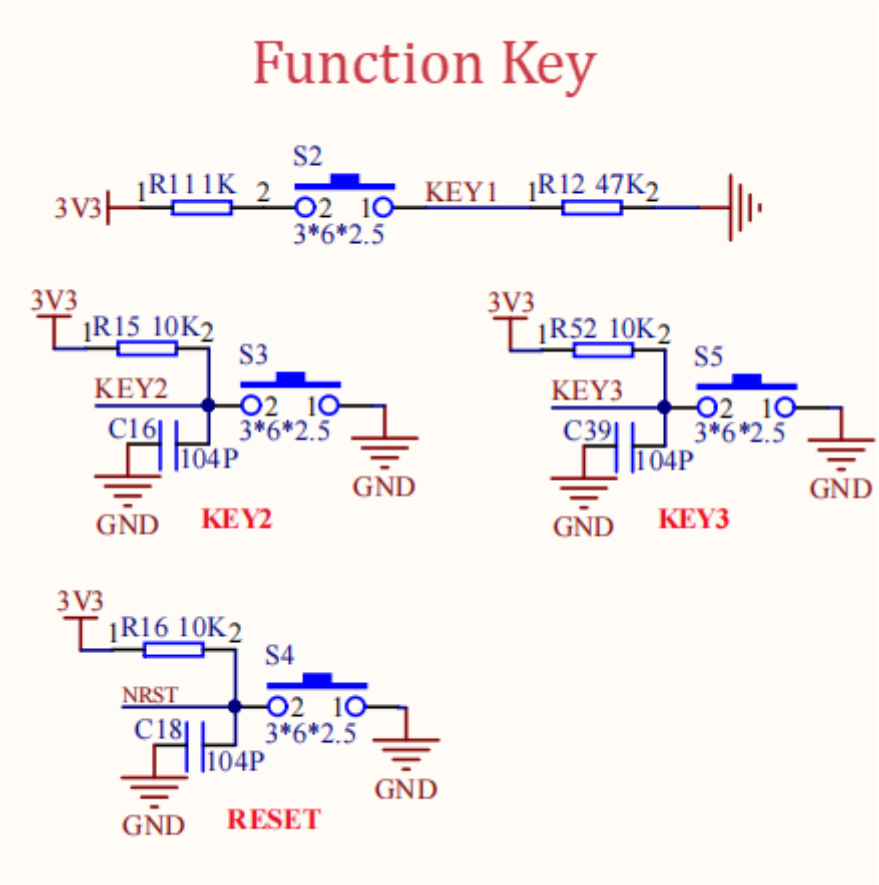

Photoresistor1
Photoresistor2
RGB1
RGB2
UART2
UART3
IIC0
IIC1
Motor 1
Motor 2
DC5VOUT
Infrared tracking
Flash
MCU
BEEP
Fingerprint
Bluetooth
YAHBOOM
YB-CUT01-V1.0
IMU
Charging
Burn
OFF
ON
SWD



UART 2

Vision module interface
K210/K230 module interface





2. Code Analysis

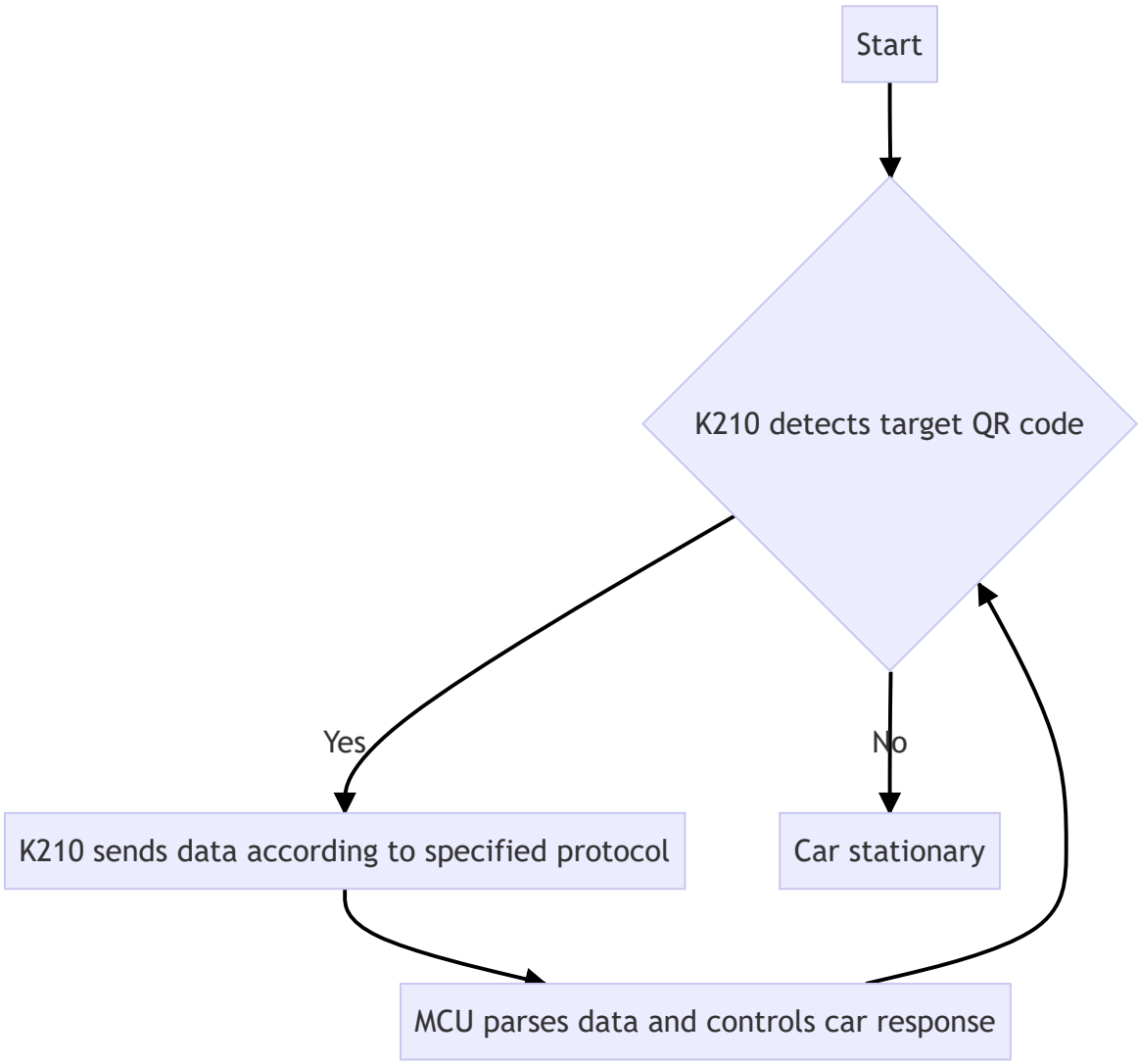
Control Principle

K210 Protocol

Start Symbol	Length	Routine Number	Routine Group	Data Amount	x	Separator	y	Separator	w	Separator	h	Separator	Payload	Separator	Checksum	End Symbol
$	XX	03	BB	05	XXX	,	XXX	,	XXX	,	XXX	,	...	,	XX	#

X: X-coordinate of the top-left corner of the detected box (range: 0-240)
Y: Y-coordinate of the top-left corner of the detected box (range: 0-320)
W: Width of the detected box (range: 0-240)
h: Height of the detected box (range: 0-320)
payload: Recognized string data

Control Flow Chart



bsp_k210_use.c

```
// 处理K210发来的单字节数据 / Process single-byte data from K210
void recv_k210msg(uint8_t recv_msg)
{
    // 检测包头'$'，开始接收新消息 / Detect header '$' to start receiving new message
    if (recv_msg == '$')
    {
        new_flag = 1;
    }

    // 检测包尾'#'，结束接收并校验 / Detect tail '#' to end reception and verify
    if(recv_msg == '#' )
    {
        if( buf_len == r_index)  // 数据长度匹配 / Data length matches
        {
            new_flag = 0;
            tou_flag = 0;
            len_flag = 0;

            // 校验和计算：减去接收的校验和字节 / Checksum calculation: subtract
received checksum byte
```

```
            buf_crc -= buf_msg[r_index-1];
            buf_crc %= 256;

            if(buf_crc == buf_msg[r_index-1])  // 校验正确 / Checksum verified
            {
                deal_recvmsg();  // 处理接收数据 / Process received data
            }
            else  // 校验失败 / Checksum failed
            {
                r_index = 0;
                buf_crc = 0;
            }
        }
    }

    // 正在接收新消息时的处理逻辑 / Processing logic when receiving new message
    if(new_flag == 1 )
    {
        if(recv_msg == '$' && tou_flag == 0)  // 首次接收包头 / Receive header for
first time
        {
            tou_flag = 1;
        }
        else
        {
            buf_msg[r_index++] = recv_msg;  // 存储数据并递增索引 / Store data and
increment index
            buf_crc += recv_msg;            // 累加校验和 / Accumulate checksum

            if(len_flag == 0)  // 未获取长度时，从首字节读取长度 / Read length from
first byte if not obtained
            {
                buf_len = buf_msg[0];
                len_flag = 1;
            }
        }
    }
}

// 解析接收的完整消息 / Parse received complete message
void deal_recvmsg(void)
{
    uint8_t index,data_i=0;
    uint8_t eg_num = buf_msg[1];//例程编号 / Routine number
    uint8_t number = buf_msg[3];//数据量(包含逗号) / Data count including commas
    uint8_t i_duo = 0;          // 逗号过滤标志 / Comma filter flag

    if(r_index!=buf_len)//长度校验 / Length verification
    {
        buf_len = 0;
        return ;
    }

    // 提取有效数据（过滤逗号） / Extract valid data (filter commas)
    for(index = 0 ;index<number;index++)
    {
        if(buf_msg[4+index] == 0x2c && i_duo ==0)//遇到逗号且未过滤 / Encounter
comma and not filtered
```

```
        {
            i_duo = 1;
            continue;
        }
        data[data_i++]=buf_msg[4+index];//存储有效数据 / Store valid data
        i_duo =0;
    }

    // 重置接收状态 / Reset receive state
    buf_crc = 0;
    r_index = 0;
    memset(buf_msg,0,sizeof(buf_msg));

    deal_data(eg_num);  // 处理解析后的数据 / Process parsed data
}
```

K210 Partial Source Code

```
def send_data(x, y, w, h, msg):
    # 协议固定字段定义 / Protocol fixed field definition
    start = 0x24          # 帧头（对应ASCII字符'$'，标识帧的开始） / Frame header (corresponds to ASCII character '$', marks frame start)
    end = 0x23            # 帧尾（对应ASCII字符'#'，标识帧的结束） / Frame tail (corresponds to ASCII character '#', marks frame end)
    class_num = 0x03      # 例程编号（固定为0x03，标识当前是二维码识别任务） / Routine number (fixed at 0x03, identifies current QR code recognition task)
    class_group = 0x00    # 例程组（固定为0x00，与接收端约定的功能分类标识） / Routine group (fixed at 0x00, function classification identifier agreed with receiver)
    fenge = 0x2c          # 数据分隔符（对应ASCII字符','，用于区分不同数据段） / Data separator (corresponds to ASCII character ',', used to distinguish different data segments)
    crc = 0               # 校验和（初始为0，用于验证数据传输完整性） / Checksum (initially 0, used for data transmission integrity verification)
    data = []             # 数据列表（存储待封装的坐标、二维码内容等实际数据） / Data list (stores actual data to be encapsulated like coordinates, QR code content)

    # 坐标(x,y)和尺寸(w,h)封装（仅当参数不全为0时处理，避免无效数据） / Coordinate (x,y) and dimension (w,h) encapsulation (only process when parameters are not all 0, avoid invalid data)
    if x == 0 and y == 0 and w == 0 and h == 0:
        pass
    else:
        # 封装x坐标（小端模式：低位字节在前，高位字节在后，符合多字节传输规范） / Encapsulate x coordinate (little-endian mode: low byte first, high byte after, conforms to multi-byte transmission specification)
        low = x & 0xFF          # 取x的低8位（通过与0xFF运算保留低位） / Get low 8 bits of x (retain low bits by AND with 0xFF)
        high = (x >> 8) & 0xFF  # 取x的高8位（右移8位后过滤高位） / Get high 8 bits of x (filter high bits after right shift 8)
        data.extend([low, fenge, high, fenge])  # 按「低位,高位,」格式添加到数据列表 / Add to data list in "low,high," format

        # 封装y坐标（逻辑与x一致，小端模式+分隔符） / Encapsulate y coordinate (logic same as x, little-endian mode + separator)
        low = y & 0xFF
        high = (y >> 8) & 0xFF
```

```
            data.extend([low, fenge, high, fenge])

            # 封装w（宽度，逻辑同上） / Encapsulate w (width, logic same as above)
            low = w & 0xFF
            high = (w >> 8) & 0xFF
            data.extend([low, fenge, high, fenge])

            # 封装h（高度，逻辑同上） / Encapsulate h (height, logic same as above)
            low = h & 0xFF
            high = (h >> 8) & 0xFF
            data.extend([low, fenge, high, fenge])

        # 二维码内容(msg)封装（仅当内容不为空时处理） / QR code content (msg) encapsulation
(only process when content is not empty)
        if msg is not None:
            # 遍历内容的每个字符，转十进制后添加到数据列表 / Traverse each character of
content, convert to decimal then add to data list
            for i in range(len(msg)):
                hec = str_int(msg[i])  # 调用简化的转十进制函数 / Call simplified
decimal conversion function
                data.append(hec)
                data.append(fenge)  # 每个字符后添加分隔符 / Add separator after each
character
            print(data)  # 打印数据列表（调试用，查看封装结果） / Print data list (for
debugging, view encapsulation result)

        # 移除数据列表末尾可能多余的分隔符（避免接收端解析错误） / Remove possible redundant
separator at end of data list (avoid receiver parsing error)
        if data and data[-1] == fenge:
            data.pop()  # 删除列表最后一个元素（多余的分隔符） / Delete last element of
list (redundant separator)
……
```

3. Main Functions

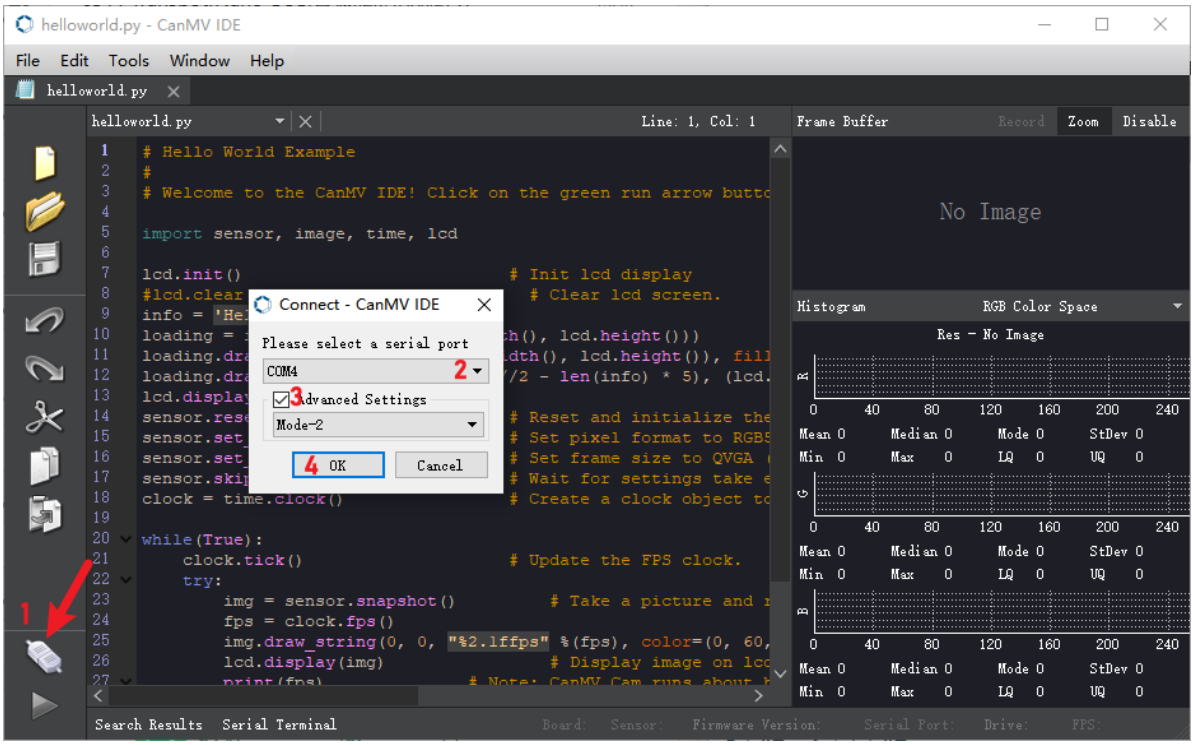
BSP_Loop

Function Prototype	**void BSP_Loop(void)**
Function Description	Main loop function: when K210 recognizes target of category 3, controls car movement (backward, forward, left turn, right turn) based on parsed string command (k210_msg.msg_msg), displays command on OLED, and resets category identifier after execution
Input Parameters	None
Return Value	None

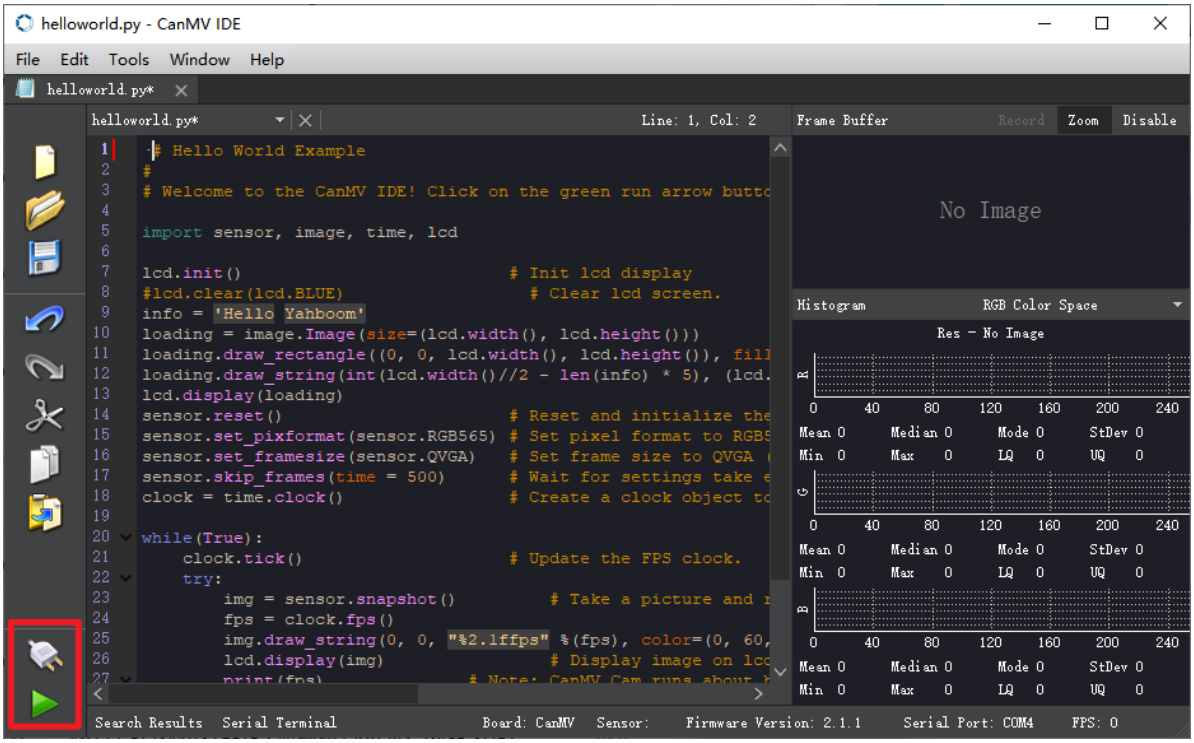
4. K210 Program Burning

After downloading and opening CanMV IDE, we need to first drag the k210_qrcode.py file from the k210 source code provided in this course to CanMV IDE to open it, then connect the IDE

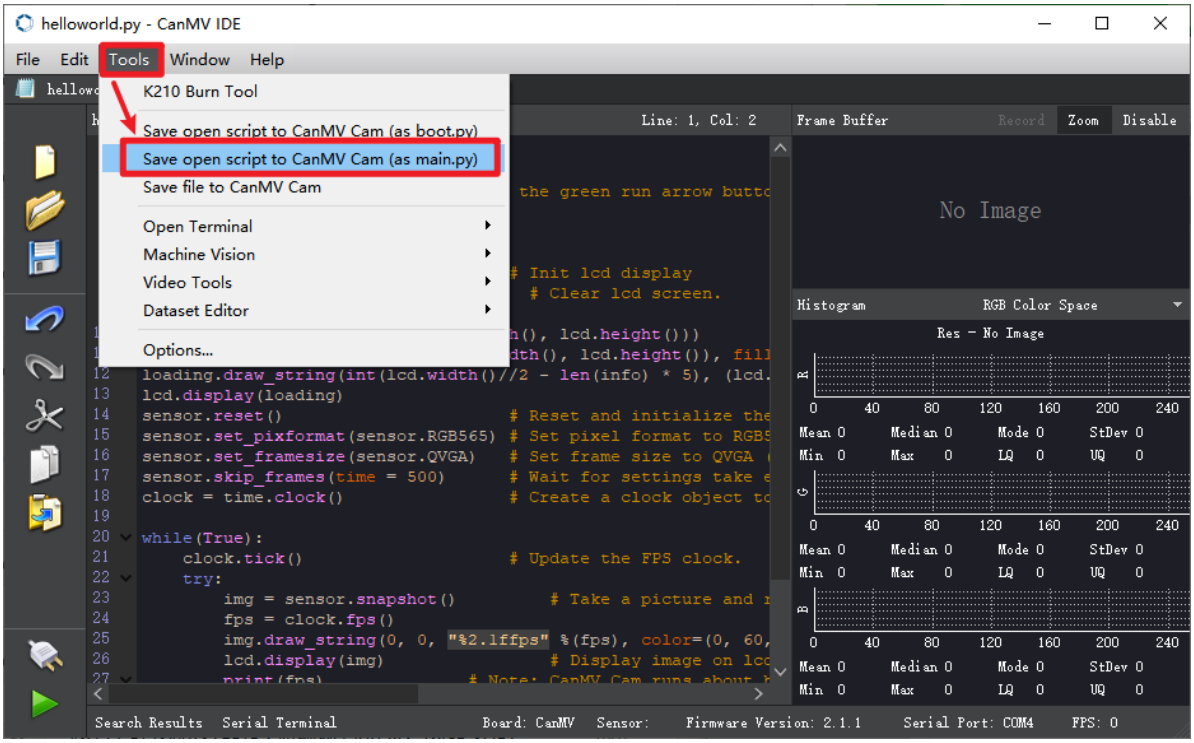
here using helloworld.py as example

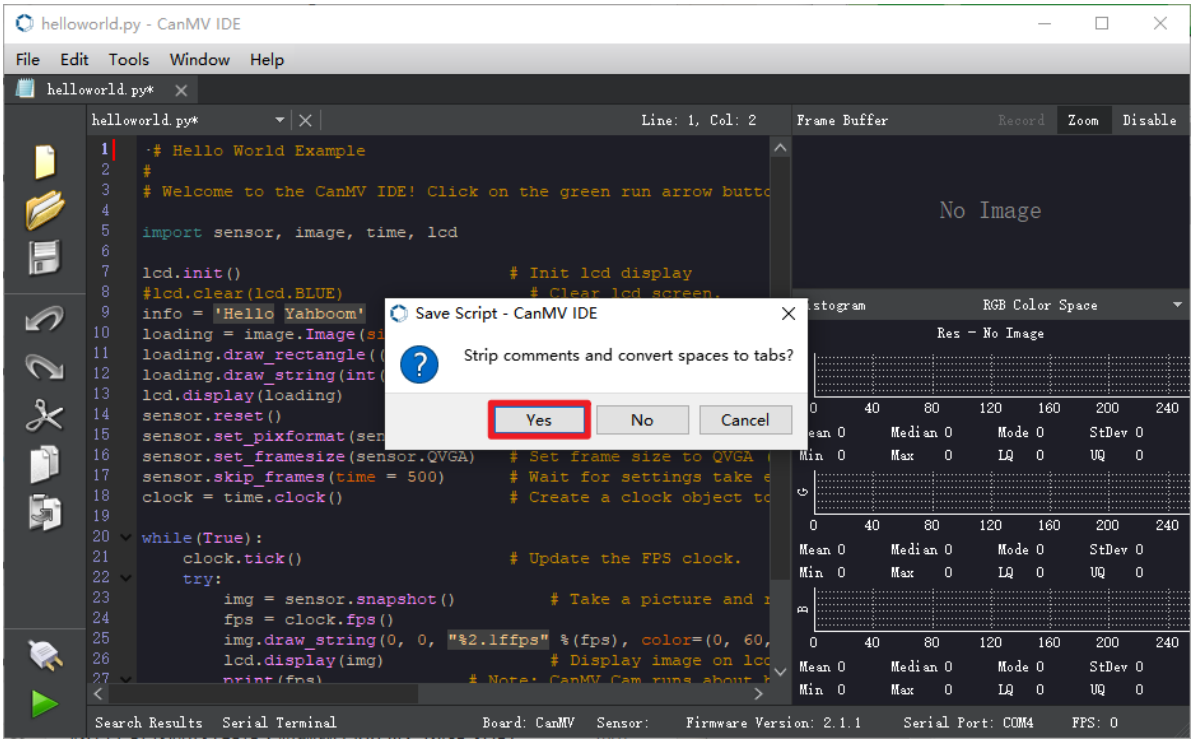


After successful IDE connection, the phenomenon is as follows

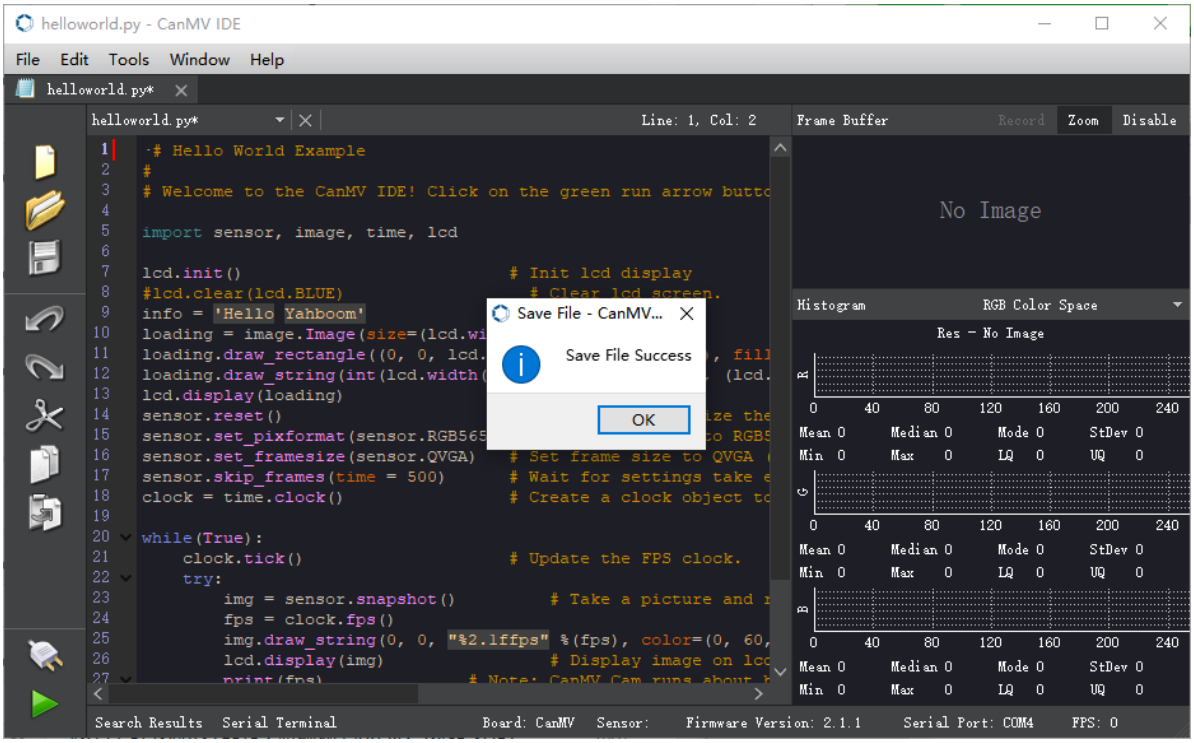


Here we use helloworld.py as an example, open the top menu bar Tools -> Save currently open script as (main.py) to CanMV Cam





Both Yes/No can be selected here. When the following status appears, the write is successful.

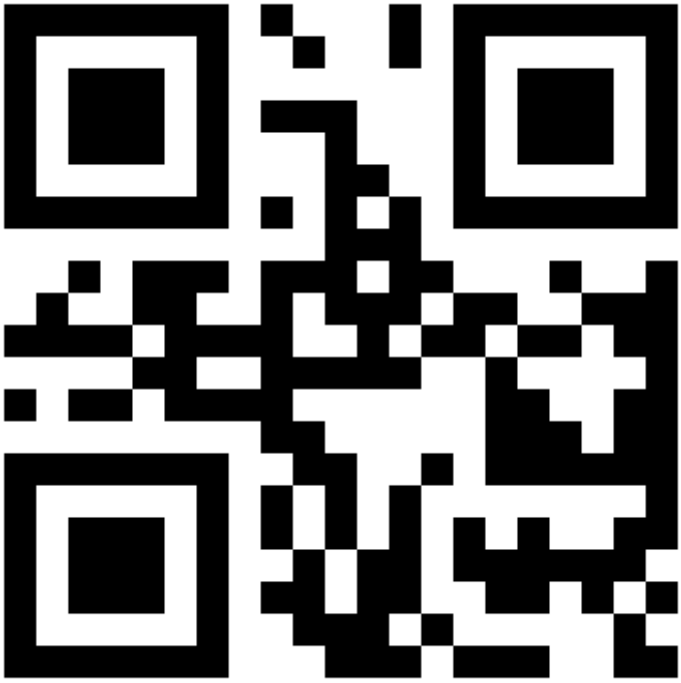


5. Experimental Phenomena

After burning the program, place QR codes in front of K210. When recognizing specified QR codes, the car will execute corresponding actions.

Below are the QR codes used in the tutorial:

goback



goahead

turnleft

turnright