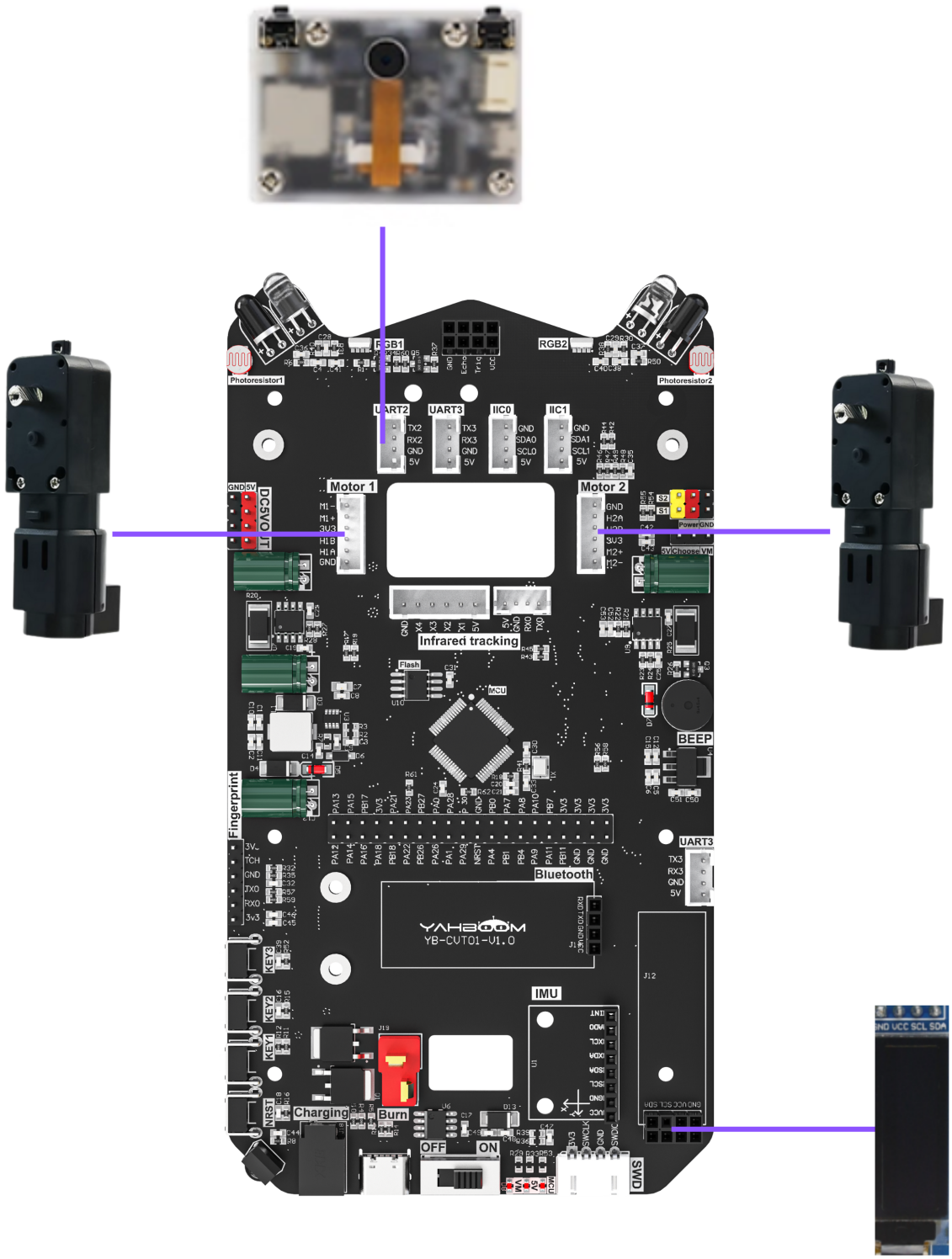
K210 Autonomous Driving

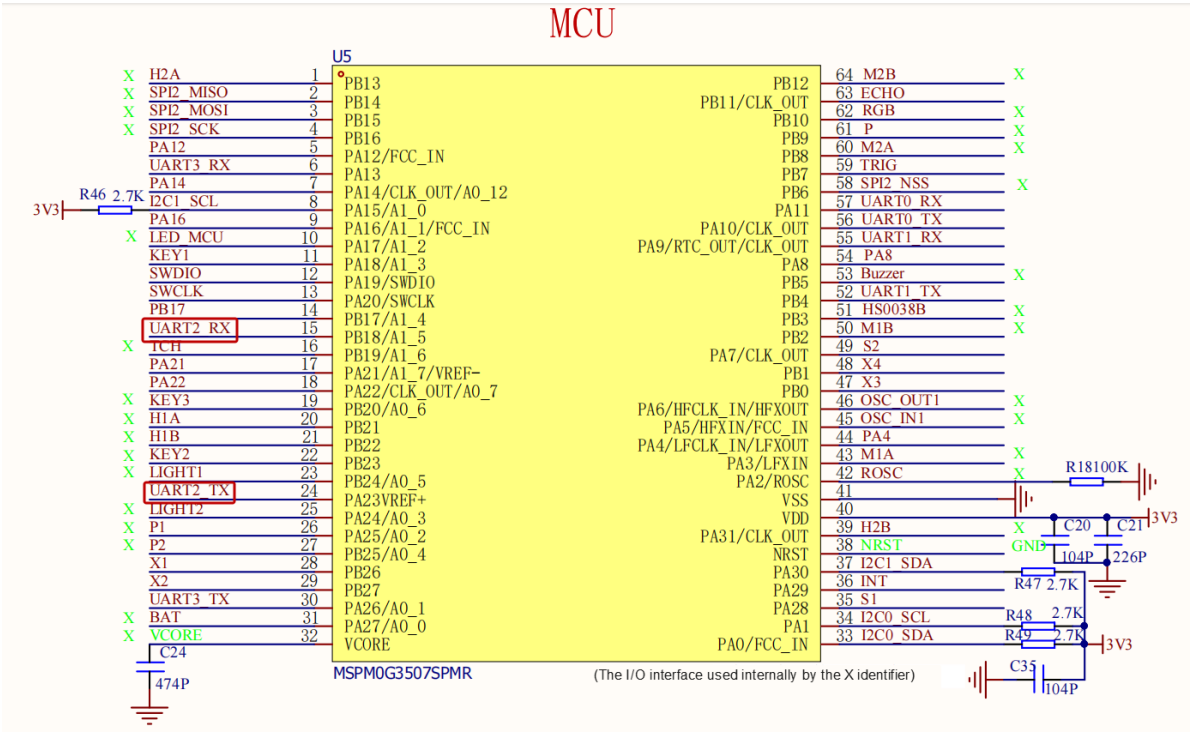
1. Hardware Connection

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

Physical Connection Diagram

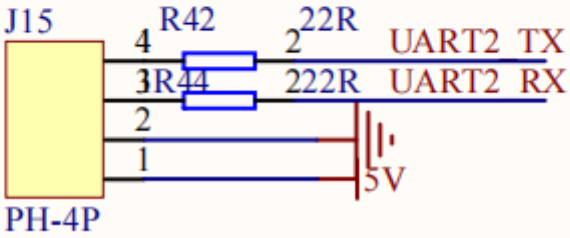


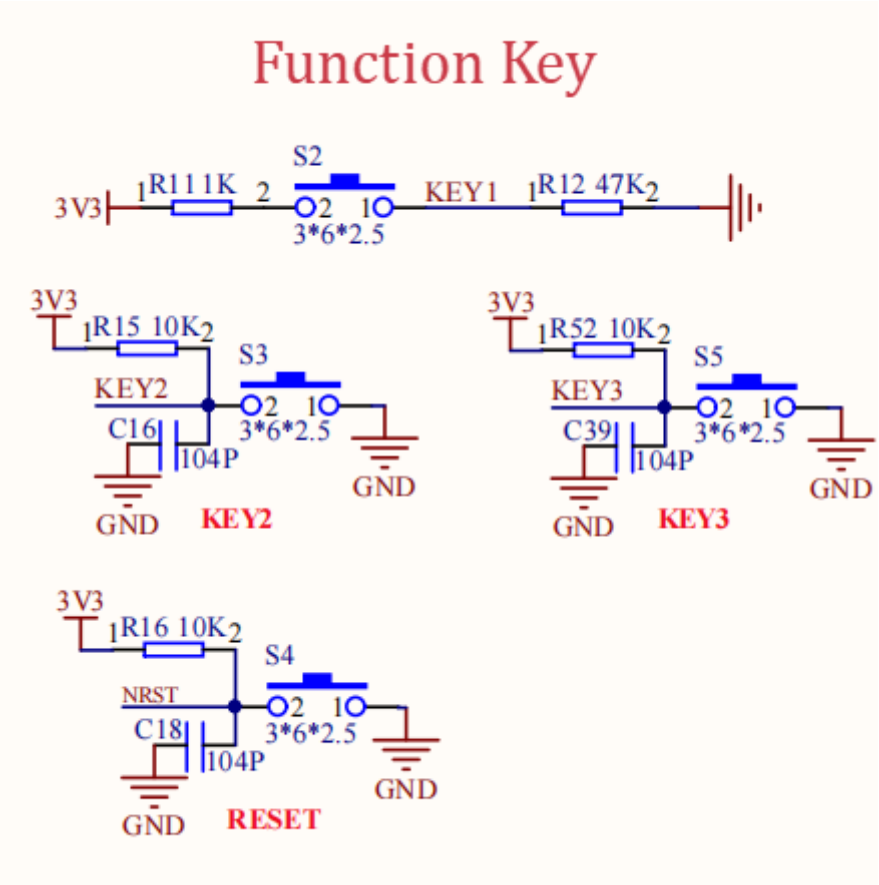
Schematic Diagram



UART 2

Vision module interface
K210/K230 module interface

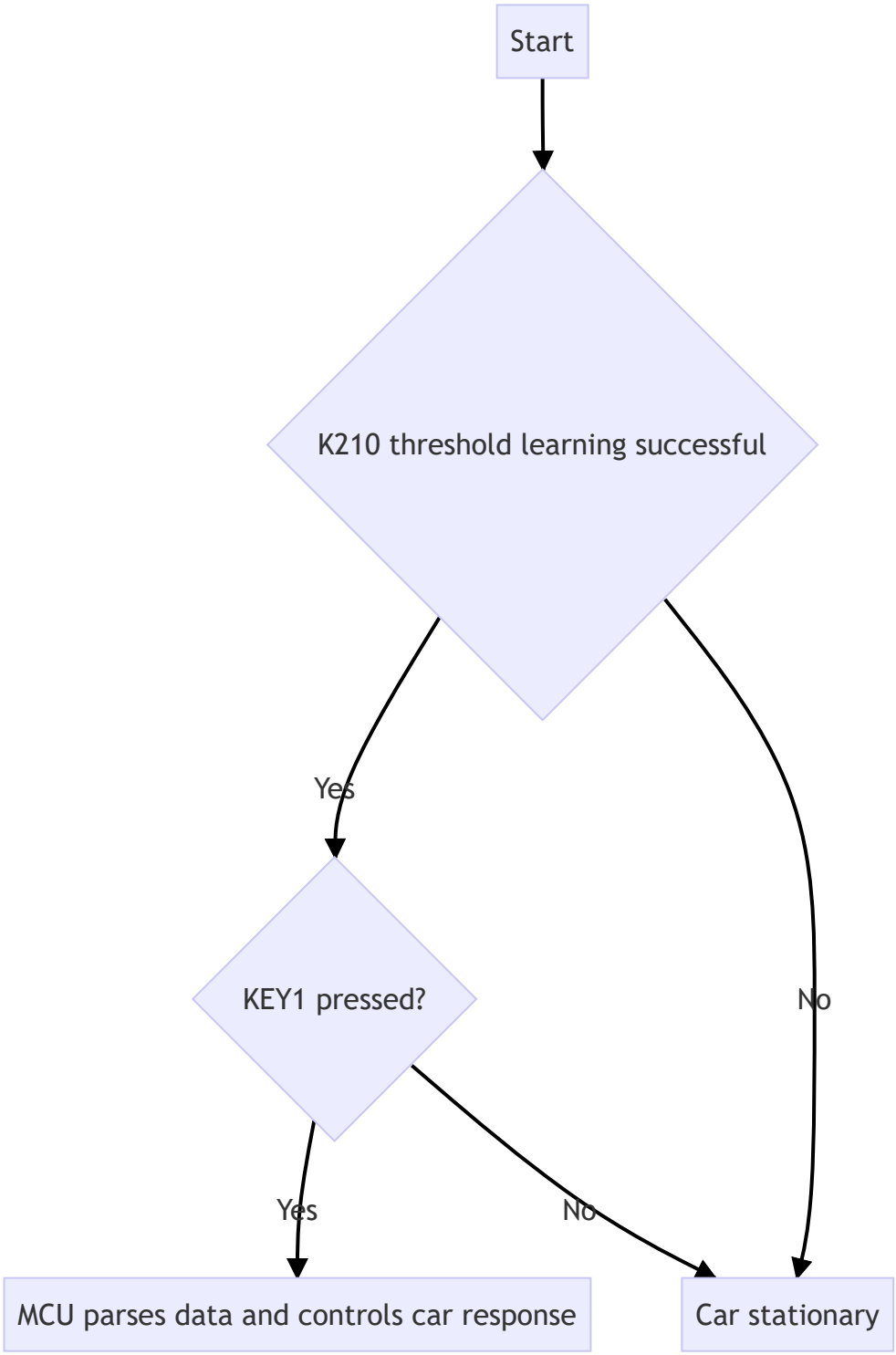




K210 Protocol

Experiment Routine	Start Symbol	Length	Routine Number	Routine Group	Data Amount	x	Separator	y	Separator	w	Separator	h	Separator	Checksum	End Symbol
Color Recognition	$	XX	01	BB	04	XXX	,	XXX	,	XXX	,	XXX	,	XX	#

Control Flow Chart



2. Code Analysis

bsp_k210_use.c

```
// 解析接收的完整消息 / Parse received complete message
void deal_recvmsg(void)
{ ...
    if(r_index!=buf_len)
    {
        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)
        {
            i_duo = 1;
            continue;
        }
        data[data_i++]=buf_msg[4+index];
    }

    // 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);
}
```

K210 Partial Source Code

```
# 数据发送函数：将坐标、尺寸、消息封装成协议格式并准备发送 / Data sending function:
encapsulates coordinates, dimensions, messages into protocol format and prepares
for sending
def send_data(x, y, w, h, msg):
    # 协议帧头（对应ASCII的'$'） / Protocol frame header (corresponds to ASCII '$')
    start = 0x24
    # 协议帧尾（对应ASCII的'#'） / Protocol frame footer (corresponds to ASCII '#')
    end = 0x23
    # 基础长度（初始值，后续会根据数据长度更新） / Base length (initial value, will be
updated based on data length)
    length = 5
    # 例程编号（固定为0x01） / Routine number (fixed as 0x01)
    class_num = 0x01
    # 例程组（固定为0xBB） / Routine group (fixed as 0xBB)
    class_group = 0xBB
    # 数据量（初始值为0，后续统计实际数据个数） / Data amount (initial value 0, will
count actual data items later)
    data_num = 0x00
    # 分隔符（对应ASCII的逗号','） / Separator (corresponds to ASCII comma ',')
    fenge = 0x2c
    # 校验和（初始值为0，后续计算） / Checksum (initial value 0, will be calculated
later)
    crc = 0
    # 存储数据的列表（用于组装待发送数据） / Data list for storage (used to assemble
data to be sent)
    data = []

    # 若坐标和尺寸全为0，不添加数据（跳过参数封装） / If coordinates and dimensions are
all 0, don't add data (skip parameter encapsulation)
    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） / Encapsulate x coordinate (little-endian: low byte first, high byte second)
        low = x & 0xFF  # 取x的低8位（低位字节） / Get low 8 bits of x (low byte)
        high = x >> 8 & 0xFF  # 取x的高8位（高位字节） / Get high 8 bits of x (high byte)
        data.append(low)
        data.append(fenge)  # 添加分隔符 / Add separator
        data.append(high)
        data.append(fenge)  # 添加分隔符 / Add separator

        # 封装y坐标（小端模式） / Encapsulate y coordinate (little-endian)
        low = y & 0xFF
        high = y >> 8 & 0xFF
        data.append(low)
        data.append(fenge)
        data.append(high)
        data.append(fenge)
        ...
    # 若消息不为空，封装消息数据 / If message is not empty, encapsulate message data
    if msg is not None:
        # 遍历消息的每个字符，转换后添加到数据列表 / Iterate through each character of the message, convert and add to data list
        for i in range(len(msg)):
            hec = str_int(msg[i])  # 字符转十进制 / Convert character to decimal
            data.append(hec)
            data.append(fenge)  # 添加分隔符 / Add separator

    # 更新数据量（数据列表的元素个数） / Update data amount (number of elements in data list)
    data_num = len(data)
    # 更新总长度（基础长度 + 数据长度） / Update total length (base length + data length)
    length += len(data)

    # 组装核心数据（长度、例程编号、例程组、数据量 + 实际数据） / Assemble core data (length, routine number, routine group, data amount + actual data)
    send_merr = [length, class_num, class_group, data_num]
    for i in range(data_num):
        send_merr.append(data[i])

    # 计算校验和（对核心数据的所有字节求和，再对256取模） / Calculate checksum (sum all bytes of core data, then modulo 256)
    for i in range(len(send_merr)):
        crc += send_merr[i]
    crc = crc % 256

    # 组装完整协议帧：添加帧头、校验和、帧尾 / Assemble complete protocol frame: add frame header, checksum, frame footer
    ...
# 定义中心采样方框（50x50像素，位于QVGA画面中心） / Define center sampling box (50x50 pixels, located at center of QVGA screen)
BOX = 50
r = [(320//2)-(BOX//2), (240//2)-(BOX//2), BOX, BOX]
...

    # 组装完整协议帧（帧头 + 核心数据 + 校验和 + 帧尾） / Assemble complete protocol frame (frame header + core data + checksum + frame footer)
    send_merr.insert(0, start)
```

```
    send_merr.append(crc)

    send_merr.append(end)
    ……
    img = sensor.snapshot()
    # 获取采样区域的颜色直方图（统计该区域内颜色分布，用于计算阈值） / Get color histogram of sampling area (counts color distribution in this area for threshold calculation)
    hist = img.get_histogram(roi=sample_roi)
    # 获取直方图1%分位值（对应颜色下限，过滤极暗/极淡的噪声像素） / Get 1% percentile of histogram (corresponds to color lower limit, filters extremely dark/light noise pixels)
    lo = hist.get_percentile(0.01)
    # 获取直方图70%分位值（对应颜色上限，覆盖目标物体的主要颜色范围） / Get 70% percentile of histogram (corresponds to color upper limit, covers main color range of target object)
    hi = hist.get_percentile(0.7)

    ……
```

3. Main Functions

APP_K210X_Line_PID

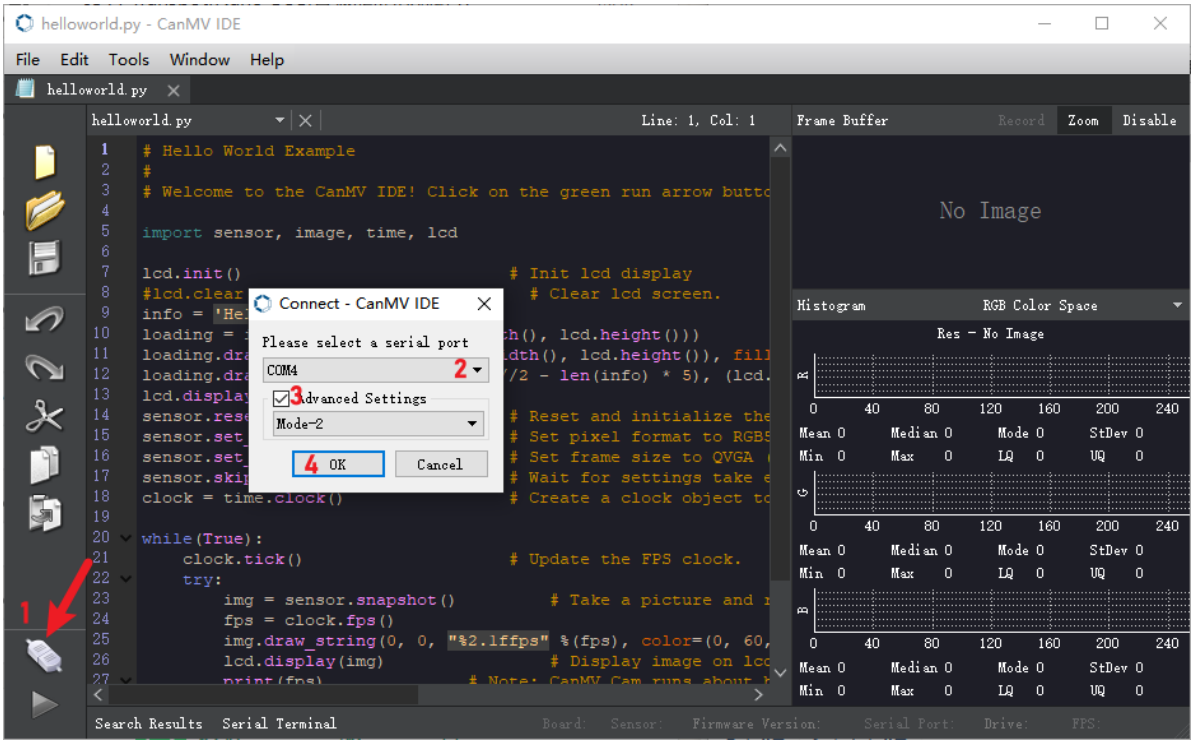
Function Prototype	**void APP_K210X_Line_PID(void)**
Function Description	Line following PID control function based on K210 camera feedback: calculates position deviation of black line in camera view (based on screen center 160, combined with offset and target area parameters), calls APP_K210X_PID_Calc to get steering control amount, controls car movement through Motion_Ctrl (straight line speed is K210X_SPEED)
Input Parameters	None
Return Value	None

BSP_Loop

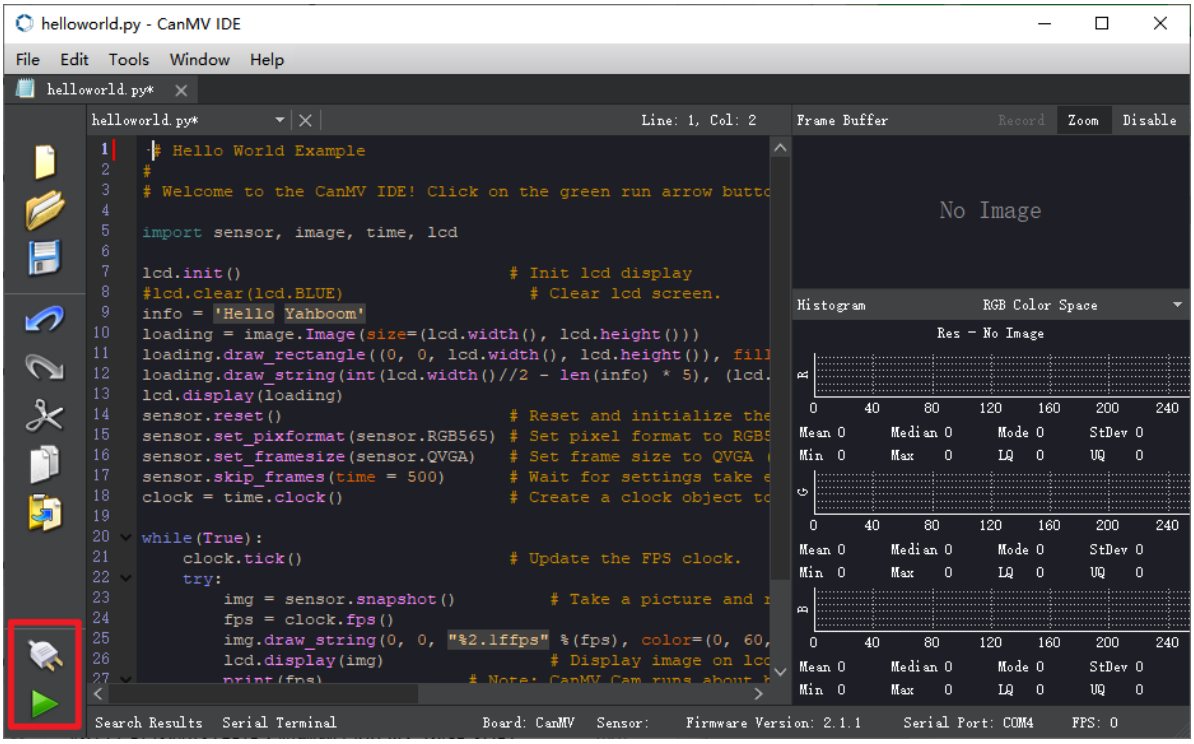
Function Prototype	**void BSP_Loop(void)**
Function Description	Main loop function: detects short press events to switch g_key_flag state. When g_key_flag is 0, stops motors and resets motor PID; when 1, executes APP_K210X_Line_PID for camera line following control
Input Parameters	None
Return Value	None

4. K210 Program Burning

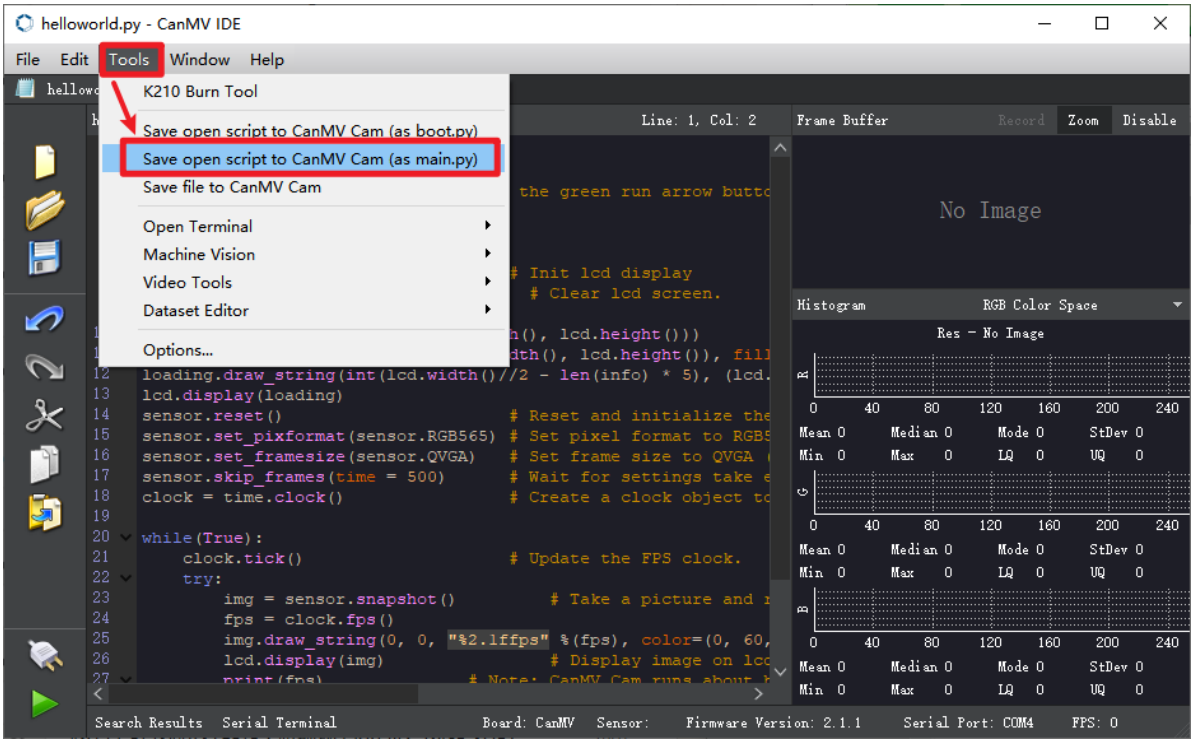
After downloading and opening CanMV IDE, we need to first drag the follow_line.py file from the k210 source code provided in this course to CanMV IDE to open it, then connect the IDE, **here using helloworld.py as example**

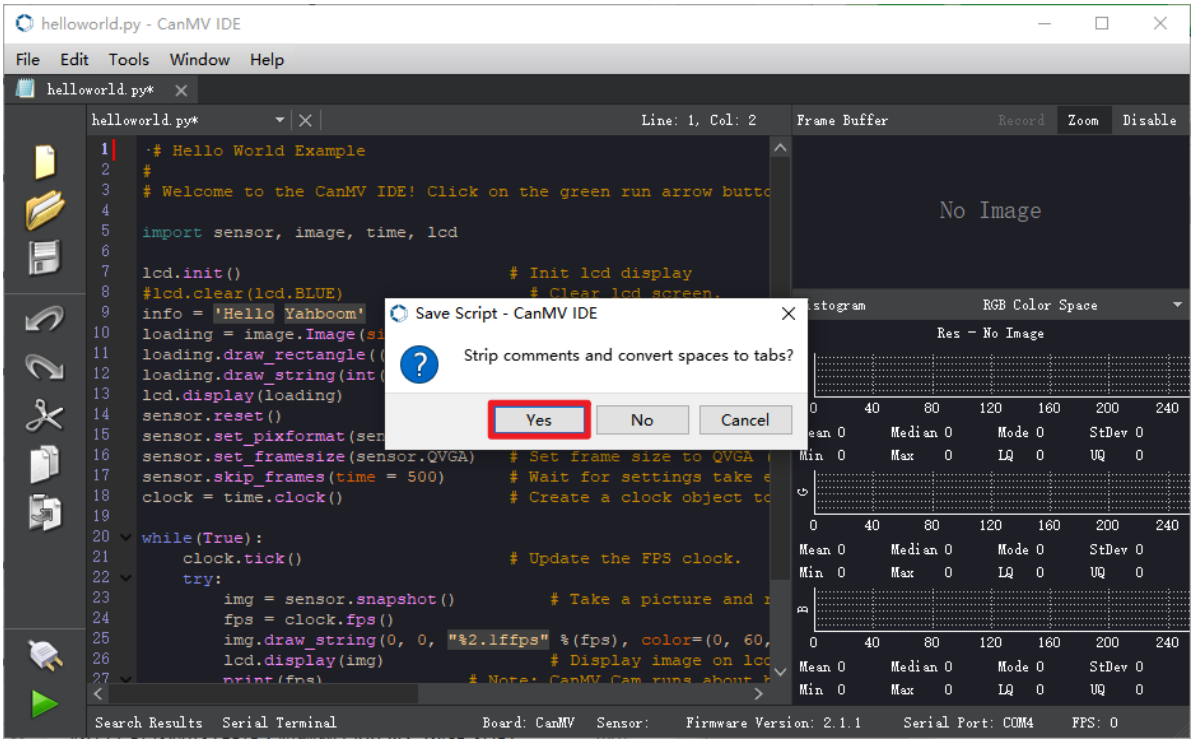


After successful IDE connection, the phenomenon is as follows

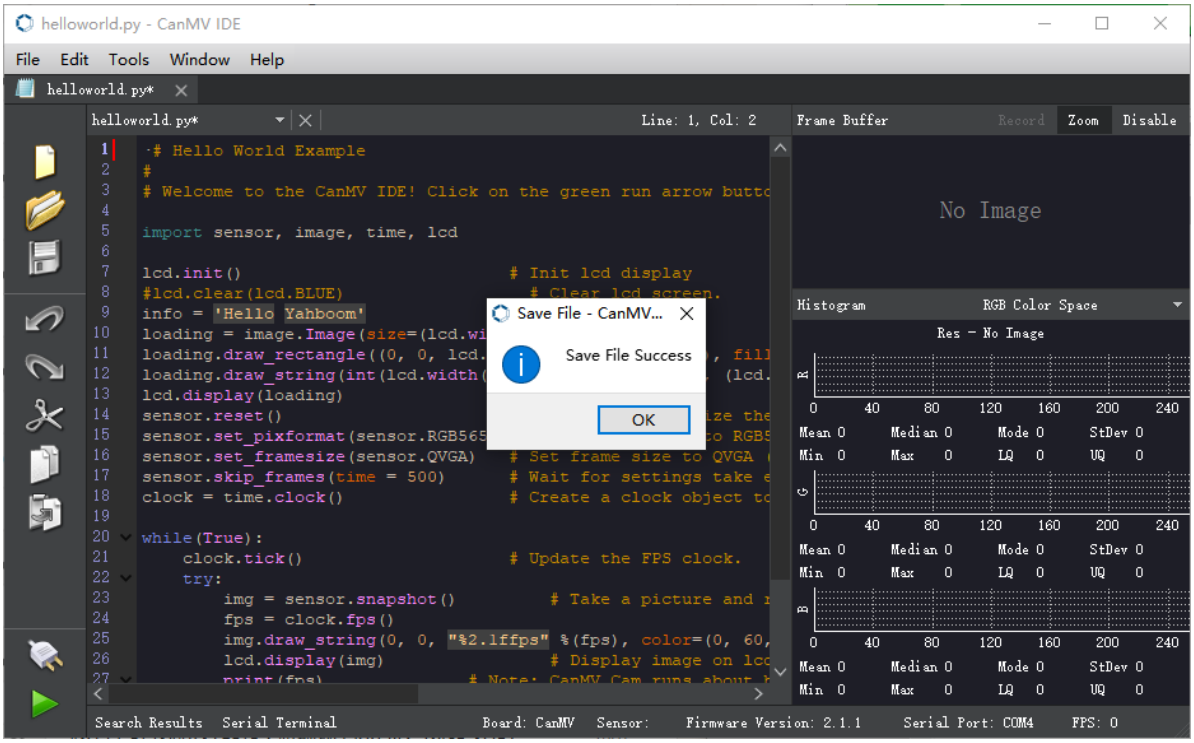


Here we use follow_line.py as an example, open the top menu bar Tools -> Save currently open script as (main.py) to CanMV Cam





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



5. Experimental Phenomena

After burning the program, turn on the power switch, wait for system initialization to complete, the LCD displays the camera image, and there is a white box in the center of the screen. Please place the color to be recognized in the white box, wait for the white box to turn green to start color collection. After collection is complete, the green box disappears and the program starts running. After color learning is successful, press KEY1 on the development board, the car will start line following. Note that learning must be successful before pressing KEY1, otherwise you need to adjust the histogram percentile values according to the environment.