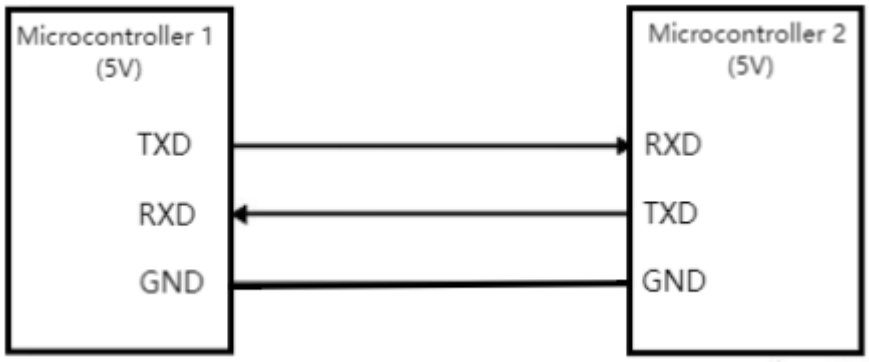
K210 Scenario-Based Autonomous Driving

This case requires the use of an eight-channel intelligent line-following module (infrared line-following module)

Serial Communication

USART: **Universal Synchronous Asynchronous Receiver/Transmitter**. It can automatically generate data frame timing based on a byte of data in the data register and send it out from the TX pin. It can also automatically receive data frame timing from the RX pin, concatenate them into a byte of data, and store it in the data register. This experiment uses UART2.

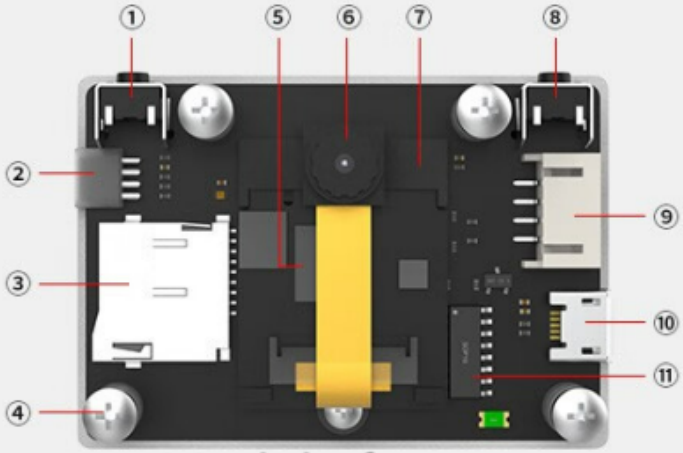


When using serial communication, both communicating parties must be "on the same channel." "On the same channel" refers to using the same communication protocol. USART (Ultra-Mobile Serial Port) Conventions: During communication, data must be transmitted in "frames." A single frame of data in a USART port includes: Start Bit + Data Bits + Parity Bit + Stop Bit. Where: 1) Start Bit: Fixed as a low-level signal for one cycle. 2) Data Bits: Can be agreed upon by both communicating parties, ranging from 5 to 9 bits. 3) Parity Bit: The USART port uses parity checking, which can be agreed upon by both communicating parties. 4) Stop Bit: Optional high-level signal for 0.5 to 2 cycles. Simultaneously, to synchronize the transmission and reception speeds of both parties, the number of data frames transmitted per second needs to be agreed upon, called the **baud rate**. Typical baud rates include 9600, 115200, 57600, etc.

K210 Vision Module

The Kendrick K210 is a system-on-a-chip (SoC) integrating machine vision and machine hearing capabilities. It utilizes TSMC's ultra-low-power 28nm advanced process, features a dual-core 64-bit processor with a total computing power of up to 1 TOPS, and incorporates various hardware acceleration units (KPU, FPU, FFT, etc.), while also boasting excellent power efficiency, stability, and reliability.

Functional Distribution as follows:



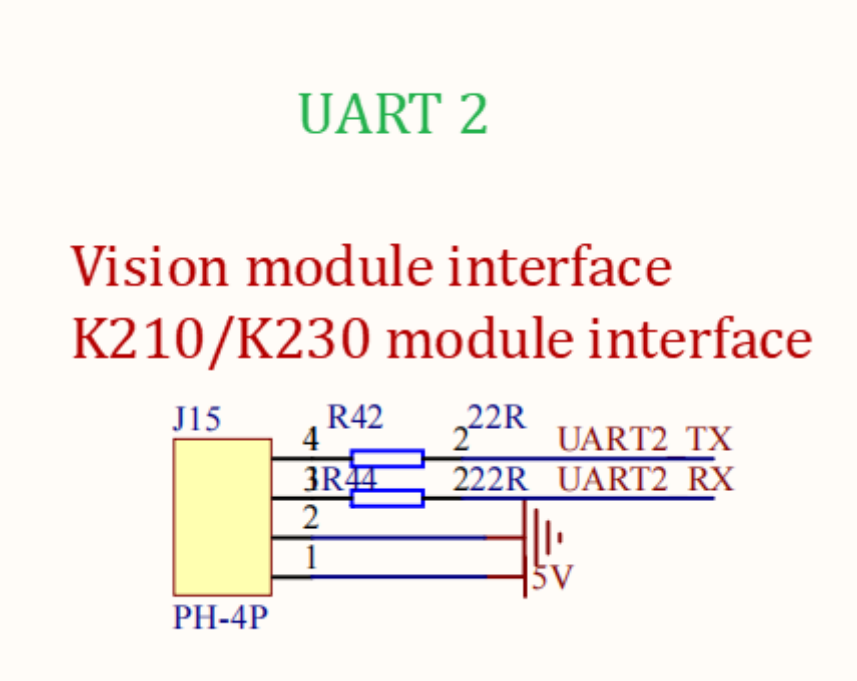
Module front

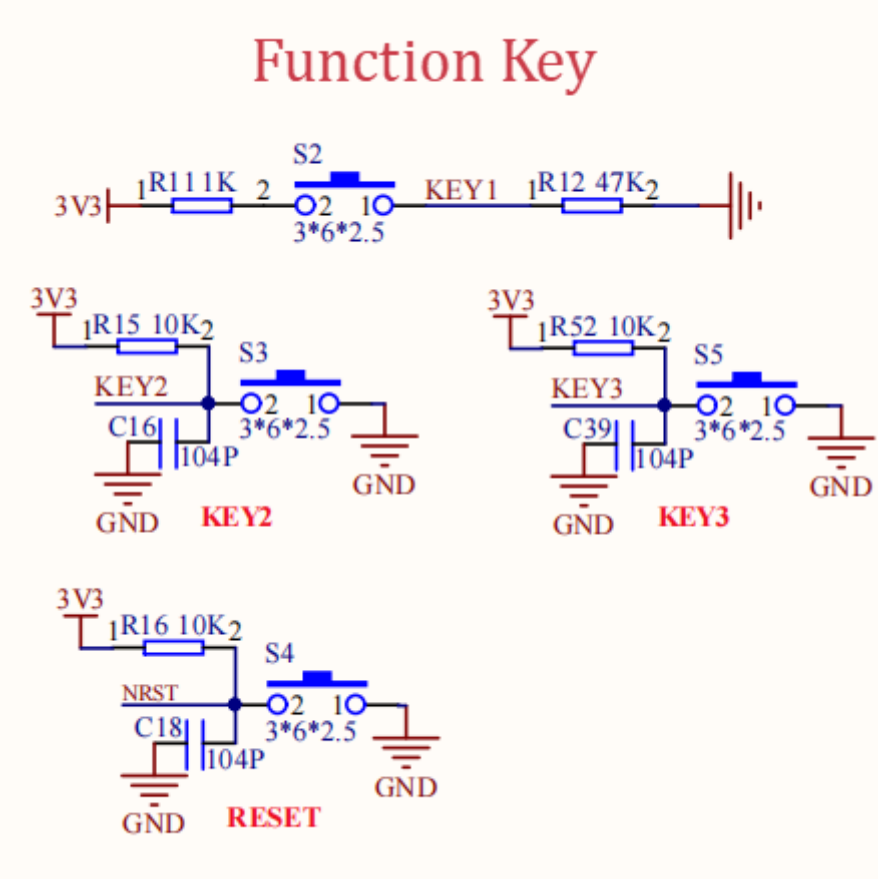
①Button K1: Hardware connection to I016, button function can be customized.
②RGB LEDs: Programmable display of red, green, blue, white and other colors.
③TF card slot: used to insert TF cards, with the gold fingers facing the module, to save program and model files.
④ Fixed copper pillars: There are a total of four fixed copper pillars with M3 diameter holes.
⑤K210 core module: contains the K210 chip operating system.
⑥ Camera: Captures images.
⑦ Module interface: Connects the K210 core module and the baseboard.
⑧ RST button: Reset button, used to restart the K210 chip.
⑨ External serial port: Connect to other serial port devices and output custom data.
⑩ microUSB interface: for downloading firmware, serial port debugging, connecting to IDE, etc.
⑪CH340 chip: Connects to the microUSB interface and converts USB signals into TTL signals.

Hardware Connection

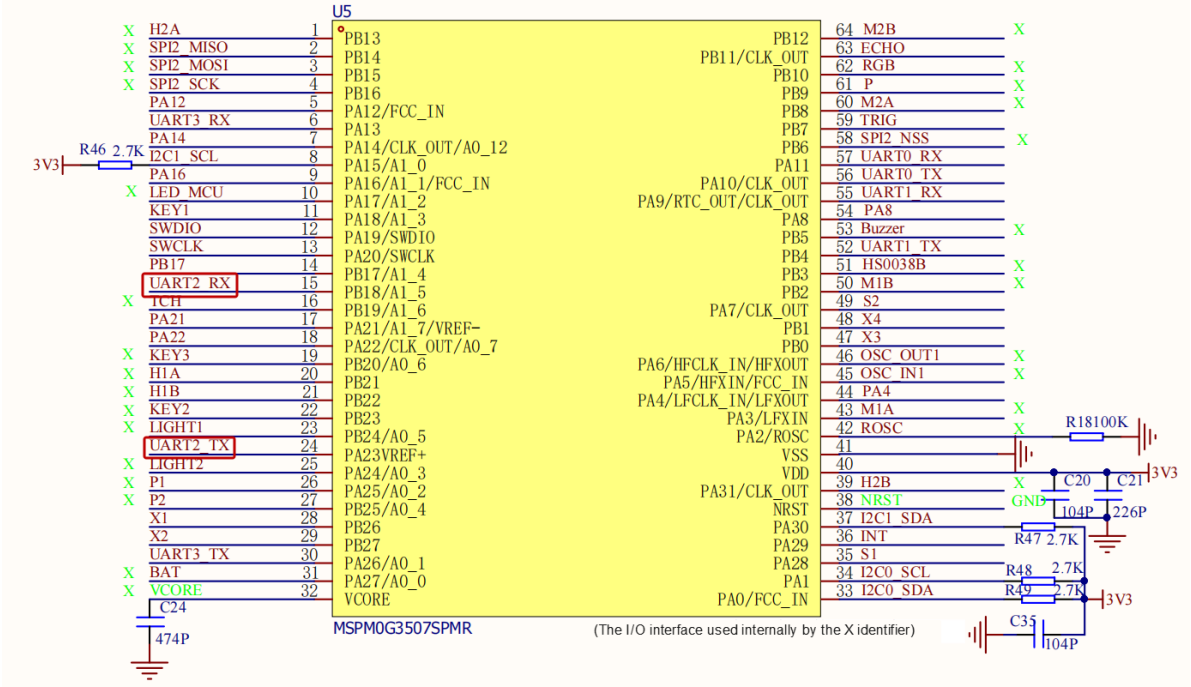
8-channel infrared tracking sensor module	MSPM0G3507
5V	5V
GND	GND
RX	TX0
TX	RX0

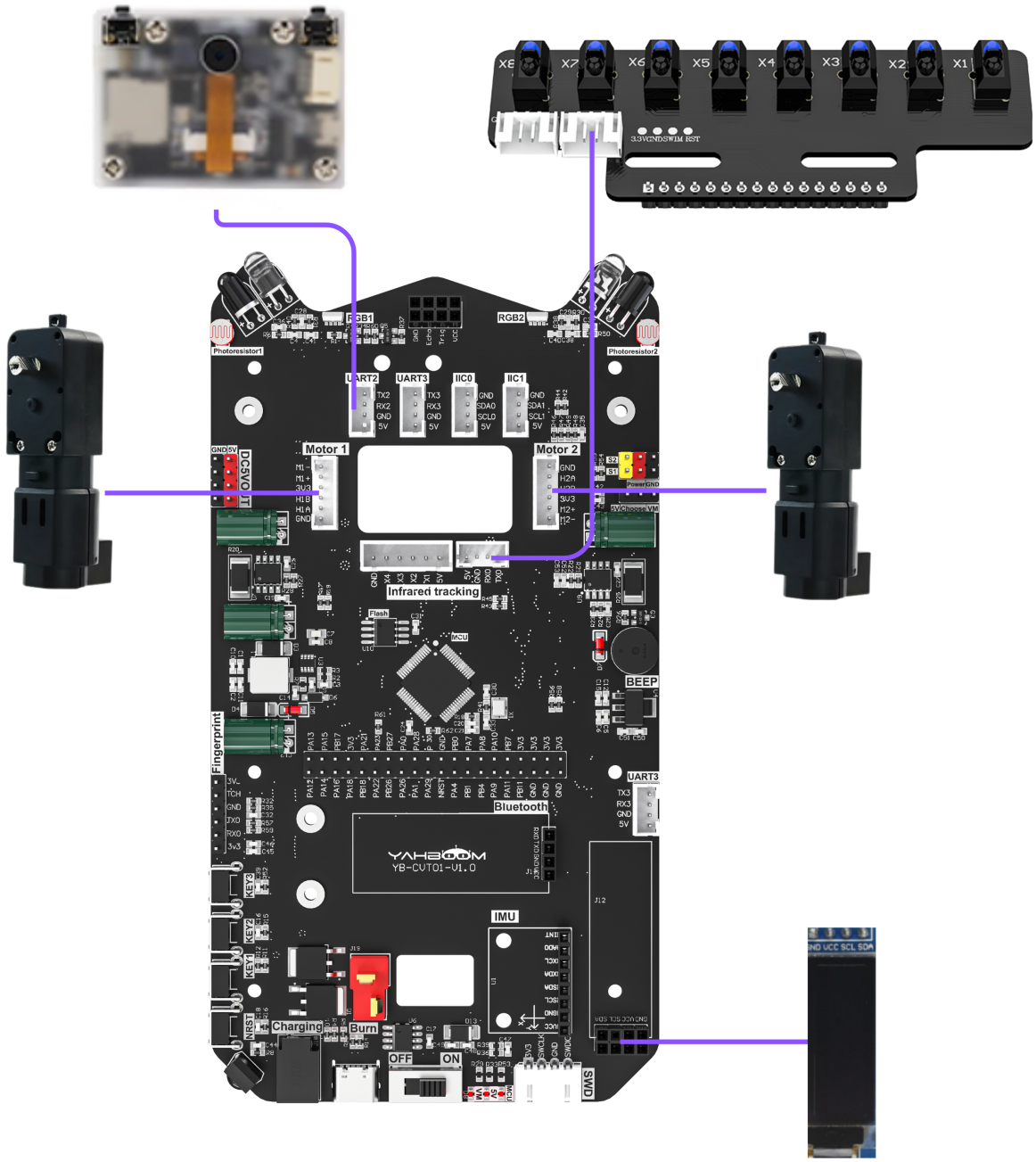
K210 vision module	MSPM0G3507
5V	5V
GND	GND
RX	TX2
TX	RX2





MCU


U5
X H2A 1
X SPI2_MISO 2
X SPI2_MOSI 3
X SPI2_SCK 4
PA12 5
UART3_RX 6
PA14 7
R46 2.7K
3V3
I2C1_SCL 8
PA16 9
X LED_MCU 10
KEY1 11
SWDIO 12
SWCLK 13
PB17 14
UART2_RX 15
X TCH 16
PA21 17
PA22 18
X KEY3 19
X H1A 20
X H1B 21
X KEY2 22
X LIGHT1 23
UART2_TX 24
X LIGHT2 25
X P1 26
X P2 27
X1 28
X2 29
UART3_TX 30
X BAT 31
X VCORE 32
C24
474P
PB13
PB14
PB15
PB16
PA12/FCC_IN
PA13
PA14/CLK_OUT/A0_12
PA15/A1_0
PA16/A1_1/FCC_IN
PA17/A1_2
PA18/A1_3
PA19/SWDIO
PA20/SWCLK
PB17/A1_4
PB18/A1_5
PB19/A1_6
PA21/A1_7/VREF-
PA22/CLK_OUT/A0_7
PB20/A0_6
PB21
PB22
PB23
PB24/A0_5
PA23VREF+
PA24/A0_3
PA25/A0_2
PB25/A0_4
PB26
PB27
PA26/A0_1
PA27/A0_0
VCORE
PB12
PB11/CLK_OUT
PB10
PB9
PB8
PB7
PB6
PA11
PA10/CLK_OUT
PA9/RTC_OUT/CLK_OUT
PA8
PB5
PB4
PB3
PB2
PA7/CLK_OUT
PB1
PB0
PA6/HFCLK_IN/HFXOUT
PA5/HFXIN/FCC_IN
PA4/LFCLK_IN/LFXOUT
PA3/LFXIN
PA2/ROSC
VSS
VDD
PA31/CLK_OUT
NRST
PA30
PA29
PA28
PA1
PA0/FCC_IN
MSPM0G3507SPMR
(The I/O interface used internally by the X identifier)
64 M2B X
63 ECHO
62 RGB X
61 P X
60 M2A X
59 TRIG
58 SPI2_NSS X
57 UART0_RX
56 UART0_TX
55 UART1_RX
54 PA8
53 Buzzer X
52 UART1_TX
51 HS0038B X
50 M1B X
49 S2
48 X4
47 X3
46 OSC_OUT1 X
45 OSC_IN1 X
44 PA4
43 M1A X
42 ROSC X
41
40
39 H2B X
38 NRST
37 I2C1_SDA
36 INT
35 S1
34 I2C0_SCL
33 I2C0_SDA
R18100K
C20
C21
3V3
GND
104P
226P
R47 2.7K
R48 2.7K
R49 2.7K
3V3
C35
104P

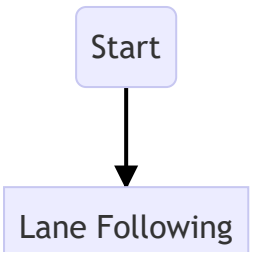


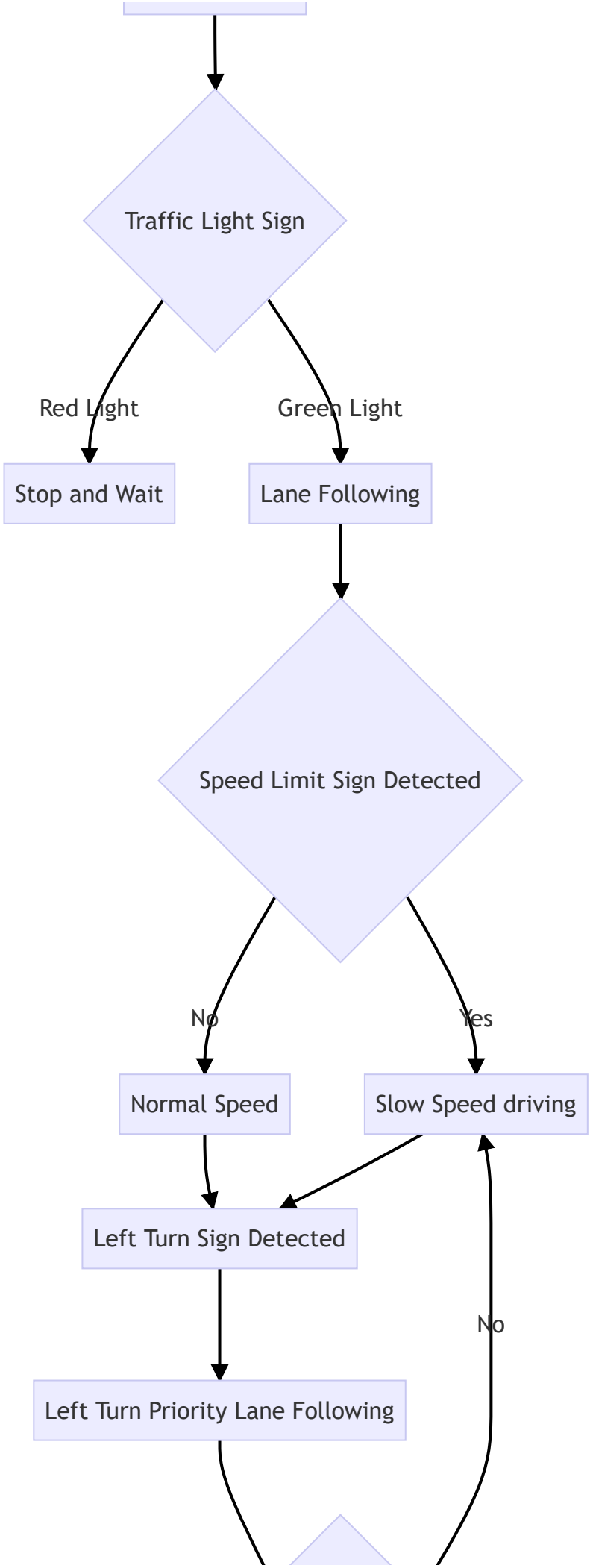
K210 Protocol

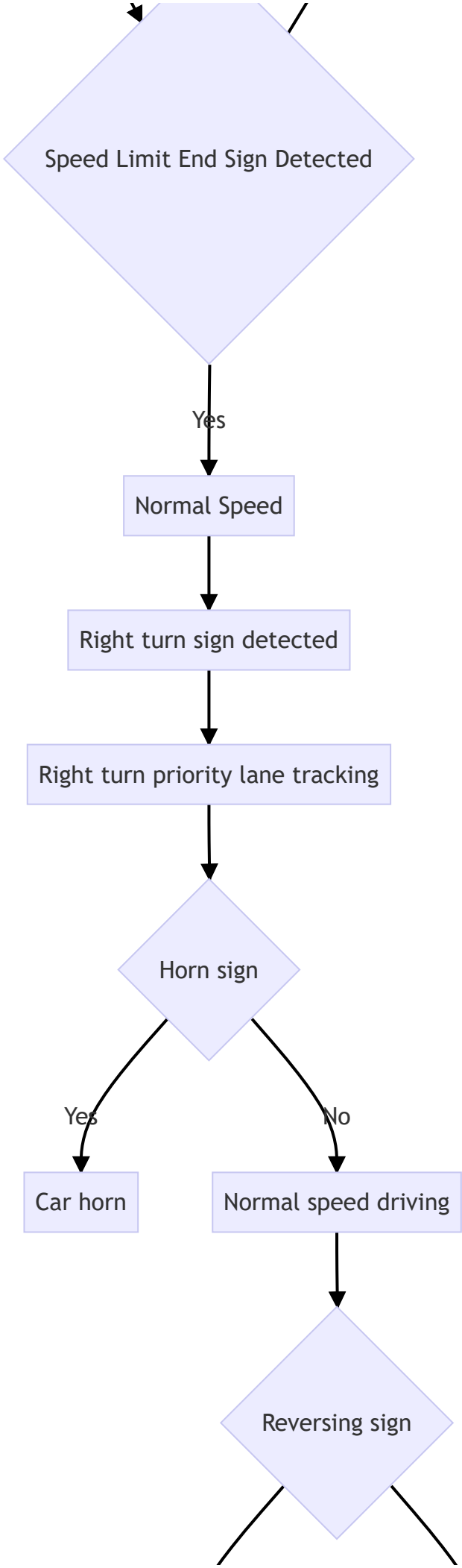
After the K210 identifies a road sign, it sends a data frame via serial port in the format **"$09" + "signature content ID," + "#"**. For example: "$091, #".

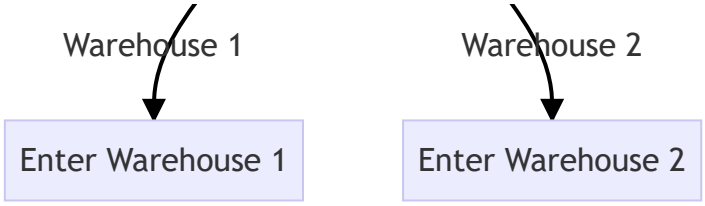
When the microcontroller receives this data frame, it checks each bit. If the content is "$09", it continues receiving. When it receives "#", it indicates that a data packet has been completed. The microcontroller parses the information transmitted by the K210 and compares the signature content ID with a specific command. If it matches the command, the vehicle will perform the corresponding operation.

Control Flowchart




Traffic Light Sign
Red Light
Green Light
Stop and Wait
Lane Following
Speed Limit Sign Detected
No
Yes
Normal Speed
Slow Speed driving
Left Turn Sign Detected
No
Left Turn Priority Lane Following


Speed Limit End Sign Detected
Yes
Normal Speed
Right turn sign detected
Right turn priority lane tracking
Horn sign
Yes
No
Car horn
Normal speed driving
Reversing sign



Partial Code Analysis

bsp_k210.c

```
void data_deal_k210(uint8_t data) {

    // Process data sent by K210 device, parse single-byte data according to
specific format
    switch(data_step) {
        case 0:

            // Reset all states to ensure a clean start
            len = 0;
            memset(k210_data, 0, K210_DATA_MAX_LEN);
            if(data == '$') {

                // Detected start character '$', move to next state
                data_step++;
            }
            break;

        case 1:
            if(data == 'O') {

                // Matched second feature character 'O', move to next state
                data_step++;
            } else {

                // Reset state if not matched
                data_step = 0;
            }
            break;

        case 2:
            if(data == '9') {

                // Matched third feature character '9', move to next state
                data_step++;
            } else {

                // Reset state if not matched
                data_step = 0;
            }
            break;

        case 3:
                 // Receiving x value (numeric digits)
```

```
            if(data == ',') {
                                // End of x reception, enter end character
verification phase
                if(len == 0 || len > K210_DATA_MAX_LEN) {

                    // x is empty or exceeds max length, invalid data
                    data_step = 0;
                } else {

                    // Valid data length, enter end character waiting state
                    data_step++;
                                    // Move to state 4 waiting for '#'
                }
            } else if(data >= '0' && data <= '9') {
                                                    // Only receive numeric
digits
                if(len < K210_DATA_MAX_LEN) {
                                                // Limit max length to prevent
overflow
                    k210_data[len++] = data;
                } else {

                    // Exceeds max length, treat as invalid data
                    data_step = 0;
                    len = 0;
                    memset(k210_data, 0, K210_DATA_MAX_LEN);
                }
            } else {

                // Received non-numeric and non-',' character, invalid data
                data_step = 0;
                len = 0;
                memset(k210_data, 0, K210_DATA_MAX_LEN);
            }
            break;

        case 4:
                 // Verify end character '#'
            if(data == '#') {
                                // Complete format match
                deal_msg(len);
                                 // Process received valid message
            }

            // Reset state regardless of whether '#' is received (to avoid
residue)
            data_step = 0;
            len = 0;
            memset(k210_data, 0, K210_DATA_MAX_LEN);
            break;
    }
}

void deal_msg(uint8_t length) {

    // Process parsed message, convert digits and verify range
    data_id = 0;
    if(length == 1) {
```

```
        // Single-byte digit conversion
        data_id = k210_data[0] - '0';
    } else if(length == 2) {

        // Two-byte digit conversion (tens place + units place)
        uint8_t data_H = k210_data[0] - '0';
        uint8_t data_L = k210_data[1] - '0';
        data_id = data_H * 10 + data_L;
    }


    // Verify if data_id is within 1-11 range
    if(data_id >= 1 && data_id <= MAX_ID) {
        stop_flag = 0;
                        // Valid data, normal processing
    } else {
        stop_flag = 1;
                        // Invalid data, set stop flag
    }
}
```

main.py

```
...

# Model selection: 1-tinybit_AI_01.kmodel 2-tinybit_AI_02.kmodel 3-
M0_AI_03.kmodel
kmdel = 3
...init...

# Label name list (order cannot be modified, must match training)
label_num =
["red","green","school","walk","one","right","two","freeSpeed","left","limitSpee
d","horn"]


# Configure parameters according to selected model
...
elif kmdel == 3 :
    labels =
["one","two","freeSpeed","green","horn","left","limitSpeed","red","right"]
    anchor = (1.78, 1.61, 2.03, 2.03, 2.22, 2.19, 2.50, 2.47, 3.19, 3.66)
    kpu.load_kmodel('/sd/M0_AI_03.kmodel')  # 加载模型 Load model


# Initialize YOLOv2 model parameters
kpu.init_yolo2(anchor,
               anchor_num=(int)(len(anchor)/2),
               img_w=320,
               img_h=240,
               net_w=320,
               net_h=240,
               layer_w=10,
               layer_h=8,
               threshold=0.4,
               nms_value=0.3,
```

```
                classes=len(labels))

msg_ = ""  # Recognition result message variable

while(True):
    gc.collect()  #  Clear memory

    clock.tick()  #  Update clock
    img = sensor.snapshot()  #  Capture a frame of image


    # Run KPU for object detection
    kpu.run_with_output(img)
    dect = kpu.regionlayer_yolo2()  #  Get detection results

    fps = clock.fps()  #  Get FPS


    # If targets are detected
    if len(dect) > 0:
        for l in dect :

            # Draw target rectangle on image
            img.draw_rectangle(l[0],l[1],l[2],l[3],color=(0,255,0))

            # Display label and confidence
            info = "%s %.3f" % (labels[l[4]], l[5])
            img.draw_string(l[0],l[1],info,color=(255, 255, 0),scale=1.5)
            print(info)
            del info  #  Free variable memory

          )
            # Convert label to corresponding ID (starting from 1)
            msg_ = labels[l[4]]
            for i in range(len(label_num)):
                if msg_ == label_num[i]:
                    idd = str(i+1)  # ID从1开始 ID starts from 1
                    print(idd)
                    break


    # If targets are detected, send data to lower computer
    if len(dect) > 0:
       send_data ="$"+"09"+ idd+','+"#"  # Data format: $09+ID+,#
       time.sleep_ms(5)  #  Short delay
       serial.send(send_data)  #  Send data
   # else :
    #    serial.send("#")  # Send empty command when no result (commented备用)


    # Display FPS on LCD
    img.draw_string(0, 0, "%2.1ffps" %(fps),color=(0,60,255),scale=2.0)
    lcd.display(img)  #  Display image
```

Main Functions

data_deal_k210

Function Prototype	void data_deal_k210(uint8_t data)
Function Description	Processes data sent from K210, parsing it according to a fixed format (starting with '$', identified by "09", data is numbers, separated by ',', ending with '#'). Valid data is processed through deal_msg; invalid data has its state reset.
Input Parameter	data: Single-byte data received from K210
Return Value	None

deal_msg

Function Prototype	void deal_msg(uint8_t length)
Function Description	Converts the parsed numeric data (length 1 or 2) into an ID value, checks if the ID is within the range of 1-11, and sets stop_flag if valid. Set to 0 otherwise, set to 1.
Input Parameter	length: Length of the parsed numeric data.
Return Value	None.

set_dataid

Function Prototype	void set_dataid(K210_TYPE id)
Function Description	Sets the value of data_id (used for clearing, etc.).
Input Parameter	id: The ID value to be set.
Return Value	None.

get_dataid

Function Prototype	uint8_t get_dataid(void)
Function Description	Gets the current data_id value.
Input Parameter	None.
Return Value	uint8_t type, the current data_id.

Reverse_parking_no2

Function Prototype	**void Reverse_parking_no2(void)**
Function Description	Control the car to reverse into parking space No. 2: stop after reversing for a specified time, stop after turning left for a specified time, then reverse for another specified time.
Input Parameters	None
Return Value	None

Reverse_parking_no1

Function Prototype	**void Reverse_parking_no1(void)**
Function Description	Control the car to reverse into parking space No. 1: stop after reversing for a specified time, stop after turning right for a specified time, then reverse for another specified time and stop.
Input Parameters	None
Return Value	None

Road_sign_right

Function Prototype	**void Road_sign_right(void)**
Function Description	Right turn control based on road sign: adjusts the speed of the left and right motors according to sensor status to achieve a right turn or correct direction.
Input Parameters	None
Return Value	None

Road_sign_left

Function Prototype	**void Road_sign_left(void)**
Function Description	Road sign left turn control (left turn priority lane following): Adjusts the speed of the left and right motors according to the sensor status to achieve left turn or direction correction.
Input Parameters	None
Return Value	None

Road_sign_speedlimit

Function Prototype	**void Road_sign_speedlimit(void)**
Function Description	Low-speed lane following control: Based on the speed limit, fine-tunes the speed of the left and right motors according to the sensor status to achieve turning or straight-line movement.
Input Parameters	None
Return Value	None

LineWalking_PWM

Function Prototype	**void LineWalking_PWM(void)**
Function Description	PWM-based Line-following Control: Acquires infrared sensor status and adjusts the PWM values of the left and right motors based on different detection conditions to achieve straight-line or turning control.
Input Parameters	None
Return Value	None

LineCheck

Function Prototype	int LineCheck(void)
Function Description	Detects whether the car is currently on a black or white line: Returns WHITE if no black line is detected by any sensor, otherwise returns BLACK.
Input Parameters	None
Return Value	int type, WHITE (white line) or BLACK (black line)

Experimental Phenomena

The road signs placed on the map in this tutorial are shown in the image below. After successfully downloading the program, place the car on the map, turn on the switch, and wait a while until the K210 screen lights up. The car will then begin line-following.

Upon detecting a speed limit sign, the car will move at a speed not exceeding the speed limit sign.

Upon detecting an end-of-speed-limit sign, the vehicle resumes normal lane-following movement.

Upon detecting a left/right turn, the vehicle prioritizes the left/right turn.

Upon detecting a horn, the vehicle stops, sounds the horn twice, and then resumes normal lane-following movement.

Upon recognizing traffic lights, the vehicle stops at red lights and proceeds at green lights.

Regarding parking spaces 1 and 2, if the vehicle recognizes space 1, it will enter space 1; if it recognizes space 2, it will reverse into space 2.