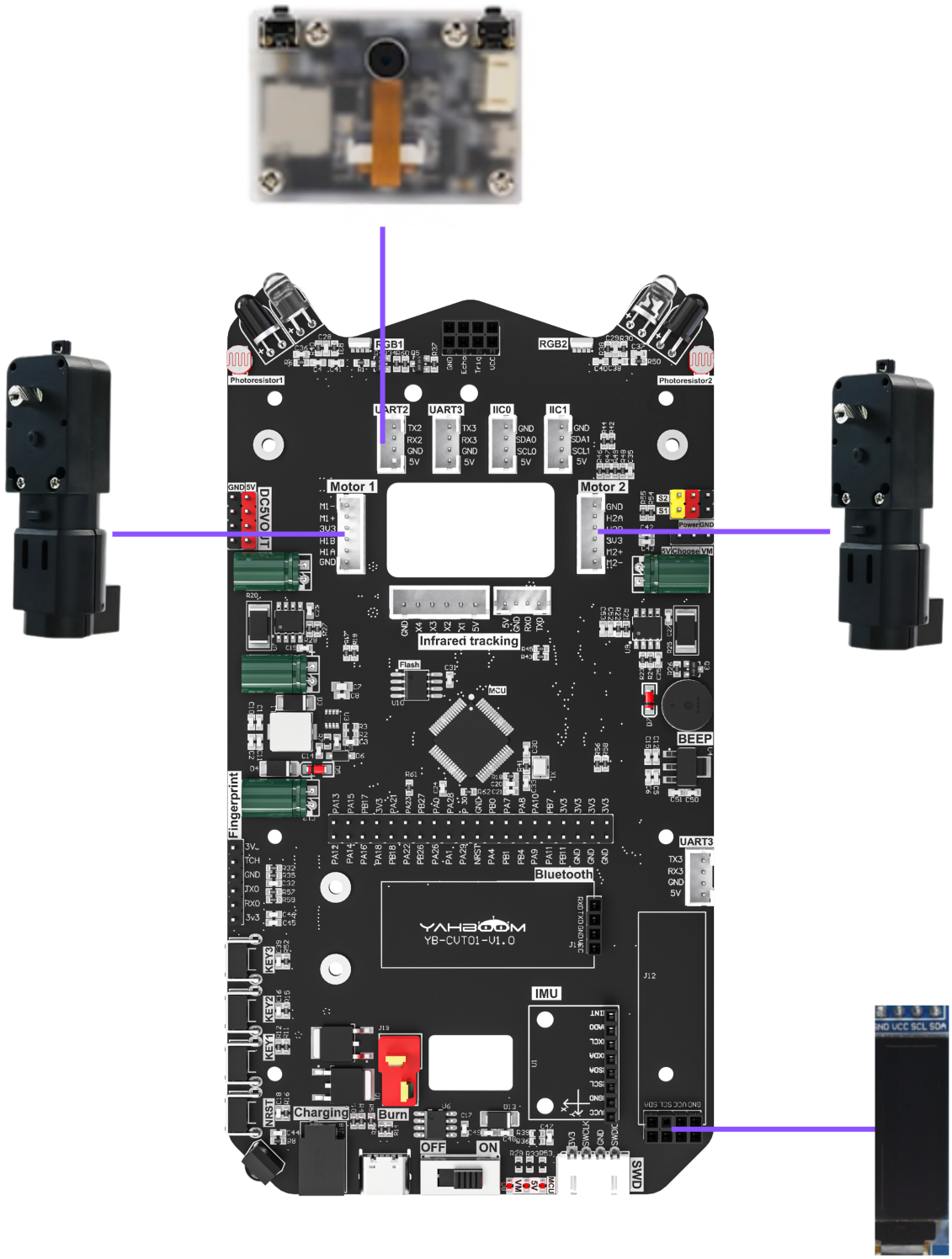
# K210 Following Machine Code



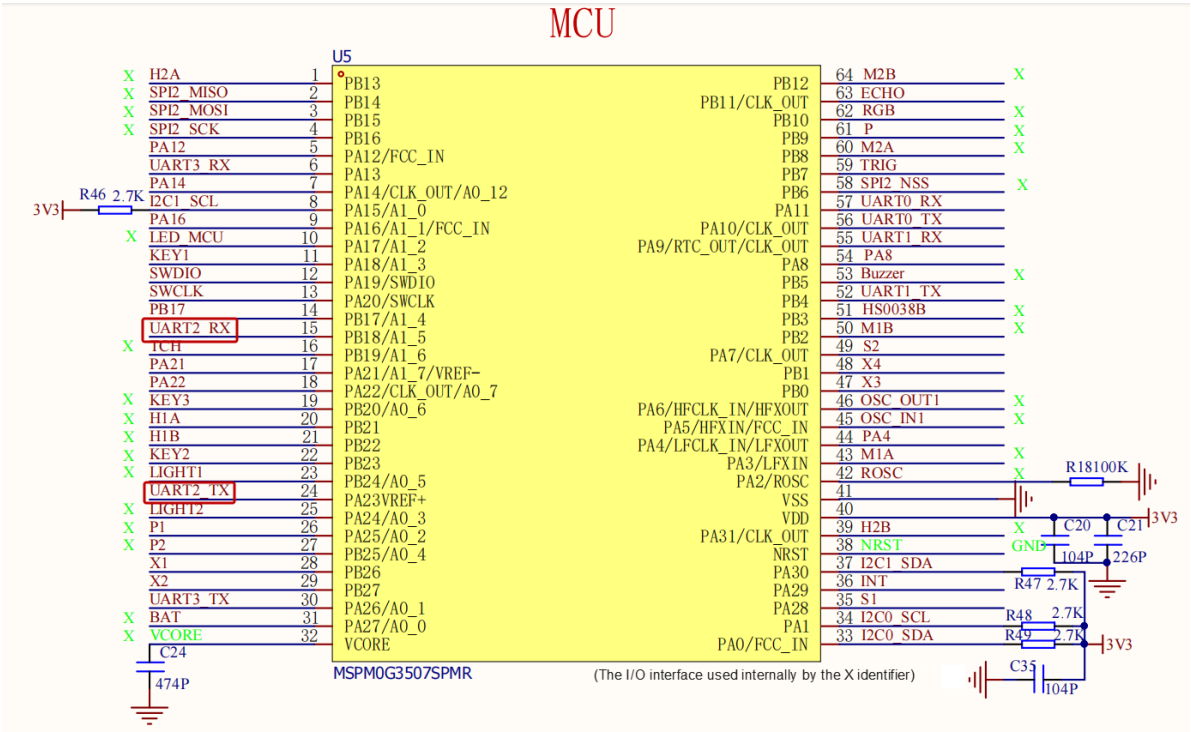
## 1. Hardware Connection

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

**Physical Connection Diagram**

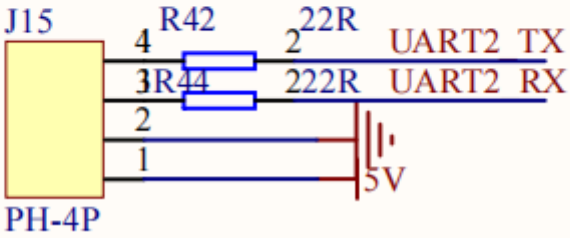


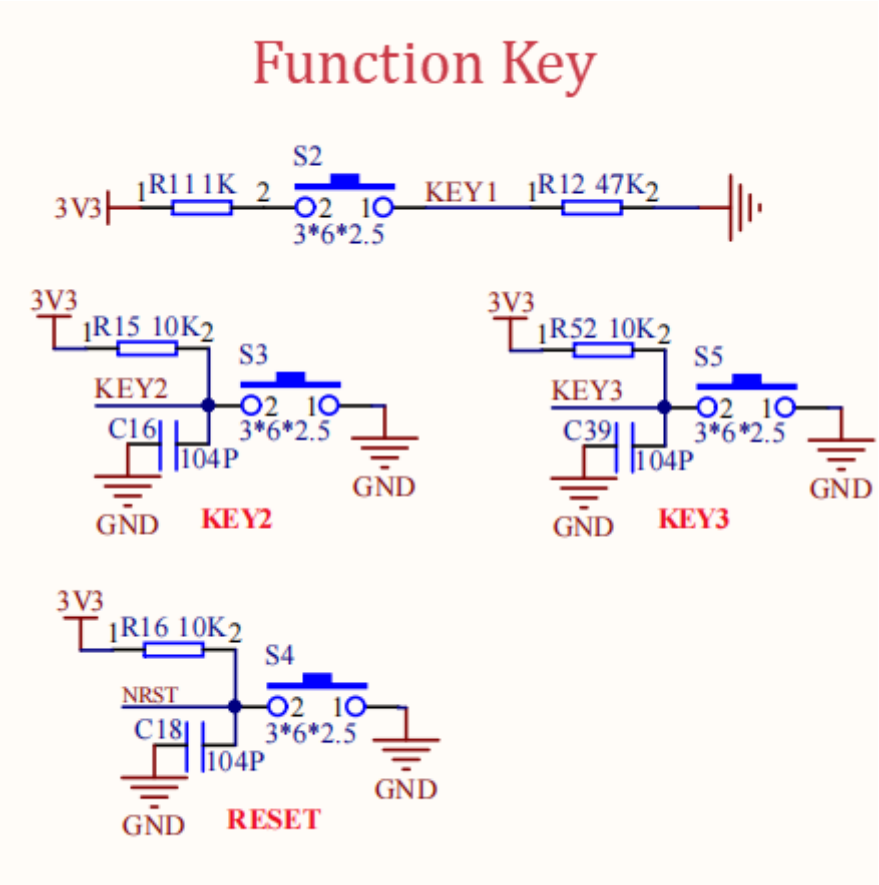

**Schematic Diagram**




# UART 2

## Vision module interface
## K210/K230 module interface






# 2. Code Analysis

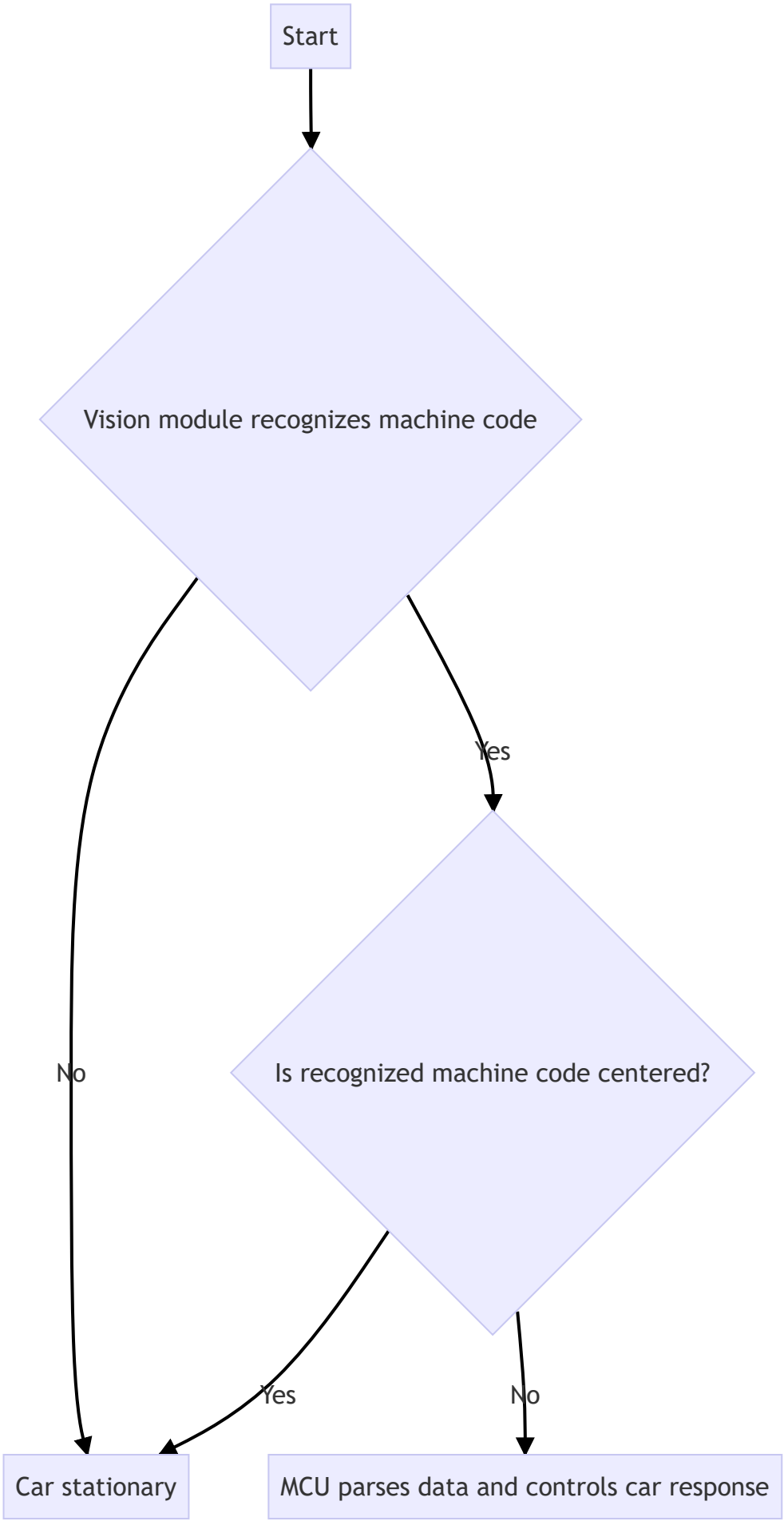
### Control Principle

### K210 Protocol

| Start Symbol | Length | Routine Number | Routine Group | Data Amount | x | Separator | y | Separator | w | Separator | h | Separator | ID | Separator | degrees | Separator | Checksum | End Symbol |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $ | XX | 04 | BB | 06 | XXX | , | XXX | , | XXX | , | XXX | , | XX | , | XX | , | 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)
id: Label of the recognized machine code
degress: Content of the recognized machine code

### Control Flow Chart


Start
Vision module recognizes machine code
No
Yes
Is recognized machine code centered?
Yes
No
Car stationary
MCU parses data and controls car response

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'$'，开始接收新消息 / Detect packet header '$', start receiving new
message
    if (recv_msg == '$')
    {
        new_flag = 1;
    }

    // 检测包尾'#'，结束接收并校验 / Detect packet tail '#', 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（抵消之前累加的校验和） / Checksum
calculation: subtract received checksum byte (offsets previously accumulated
checksum)
            buf_crc -= buf_msg[r_index-1];
            buf_crc %= 256;

            if(buf_crc == buf_msg[r_index-1])  // 校验正确，处理数据 / Checksum
correct, process data
            {
                deal_recvmsg();
            }
            else  // 校验失败，重置接收状态 / Checksum failed, reset receive state
            {
                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，标记已接收 / First
time receiving packet header, mark as received
        {
            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（用于后续验证） /
Accumulate checksum (for subsequent verification)

            if(len_flag == 0)  // 未获取长度时，从首字节读取数据总长度 / When length
not obtained, read total data length from first byte
```

```
            {
                buf_len = buf_msg[0];
                len_flag = 1;
            }
        }
    }
}
```

K210 Partial Source Code

```
# 数据发送函数（按自定义协议封装AprilTag的坐标、尺寸和ID信息，生成串口可发送帧） / Data sending function (encapsulates AprilTag coordinates, dimensions and ID information according to custom protocol, generates serial transmittable frame)
def send_data(x, y, w, h, msg):
    # 协议固定字段初始化 / Protocol fixed field initialization
    start = 0x24          # 帧头（对应ASCII的'$'，标识帧开始） / Frame header (corresponds to ASCII '$', marks frame start)
    end = 0x23            # 帧尾（对应ASCII的'#'，标识帧结束） / Frame tail (corresponds to ASCII '#', marks frame end)
    length = 5            # 基础长度（初始值=5，含后续4个固定字段+1个校验位占位） / Basic length (initial value=5, includes 4 subsequent fixed fields + 1 checksum placeholder)
    class_num = 0x04      # 例程编号（固定为0x04，标识当前是AprilTag识别任务） / Routine number (fixed at 0x04, identifies current AprilTag recognition task)
    class_group = 0xBB    # 例程组（固定为0xBB，与接收端约定的功能分类标识） / Routine group (fixed at 0xBB, function classification identifier agreed with receiver)
    data_num = 0x00       # 数据量（初始为0，后续统计实际数据元素个数） / Data amount (initially 0, counts actual data elements later)
    fenge = 0x2c          # 分隔符（对应ASCII的','，区分不同数据段） / Separator (corresponds to ASCII ',', distinguishes different data segments)
    crc = 0               # 校验和（初始为0，用于验证数据完整性） / Checksum (initially 0, used for data integrity verification)
    data = []             # 数据列表（存储待封装的坐标、消息等实际数据） / Data list (stores actual data to be encapsulated like coordinates, messages)

    # 若坐标和尺寸全为0，不封装这些参数（避免无效数据） / If coordinates and dimensions are all 0, don't encapsulate these parameters (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） / Encapsulate x coordinate (little-endian mode: low byte first, high byte after)
        low = x & 0xFF          # 取x的低8位（通过与0xFF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 / Add in "low,high," format
....
    # 若消息不为空，遍历字符并转换为十进制后封装 / If message is not empty, traverse characters and convert to decimal then encapsulate
    if msg is not None:
        for i in range(len(msg)):
            hec = str_int(msg[i])  # 调用转换函数，将字符转为十进制 / Call conversion function, convert character to decimal
            data.extend([hec, fenge])
```

```
....
    # 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 tail)
    send_merr.insert(0, start)  # 插入帧头 / Insert frame header
    send_merr.append(crc)       # 添加校验和 / Add checksum
    send_merr.append(end)       # 添加帧尾 / Add frame tail

# 核心数据（AprilTag识别相关参数，初始值均为默认状态） / Core data (AprilTag recognition
related parameters, all initial values are default state)
x_ = 0                    # AprilTag的x坐标（左上角x值） / AprilTag's x coordinate
(top-left x value)
y_ = 0                    # AprilTag的y坐标（左上角y值） / AprilTag's y coordinate
(top-left y value)
w_ = 0                    # AprilTag的宽度 / AprilTag's width
h_ = 0                    # AprilTag的高度 / AprilTag's height
msg_ = ""                 # 存储AprilTag的ID和家族信息 / Store AprilTag's ID and
family information

# 配置AprilTag识别的家族（启用多个常用家族，可注释禁用不需要的家族） / Configure AprilTag
recognition families (enable multiple common families, can comment to disable
unwanted families)
tag_families = 0
tag_families |= image.TAG16H5   # 启用TAG16H5家族 / Enable TAG16H5 family
.....

while True:

...
    # 遍历识别到的所有AprilTag，处理每个Tag的信息 / Traverse all recognized AprilTags,
process each Tag's information
    for tag in img.find_apriltags(families=tag_families):
        # 绘制Tag的红色矩形框（标记位置） / Draw Tag's red rectangle (mark position)
        img.draw_rectangle(tag.rect(), color=(255, 0, 0))
        # 绘制Tag的绿色中心点（标记中心） / Draw Tag's green center point (mark
center)
        img.draw_cross(tag.cx(), tag.cy(), color=(0, 255, 0))

        # 提取Tag的坐标和尺寸（x/y为左上角坐标，w/h为宽高） / Extract Tag's coordinates
and dimensions (x/y are top-left coordinates, w/h are width/height)
        x_ = tag.x()
        y_ = tag.y()
        w_ = tag.w()
        h_ = tag.h()

        # 生成消息：ID为个位数时补前导0（统一格式），拼接家族名称 / Generate message: add
leading 0 when ID is single digit (unified format), concatenate family name
        if tag.id() < 10:
            msg_ = '0' + str(tag.id()) + str(family_name(tag))
        else:
            msg_ = str(tag.id()) + str(family_name(tag))
        print(msg_)

    # 按识别结果发送对应数据 / Send corresponding data based on recognition results
    if msg_ != '':
        # 识别到Tag，发送Tag的坐标、尺寸和ID信息 / Tag recognized, send Tag's
coordinates, dimensions and ID information
        send_data(x_, y_, w_, h_, msg_)  # 调用发送函数封装数据 / Call sending
function to encapsulate data
```

```
        serial.send_bytearray(send_buf)  # 串口发送协议帧（字节数组形式） / Serial
send protocol frame (byte array form)
        msg_ = ''  # 重置消息，避免下一轮重复发送 / Reset message, avoid repeated
sending in next round
    else:
        # 未识别到Tag，发送全0数据（告知接收端无有效Tag） / No Tag recognized, send all
0 data (inform receiver no valid Tag)
        send_data(0, 0, 0, 0, "")
        serial.send_bytearray(send_buf)
...
```

# 3. Main Functions

**recv_k210msg**

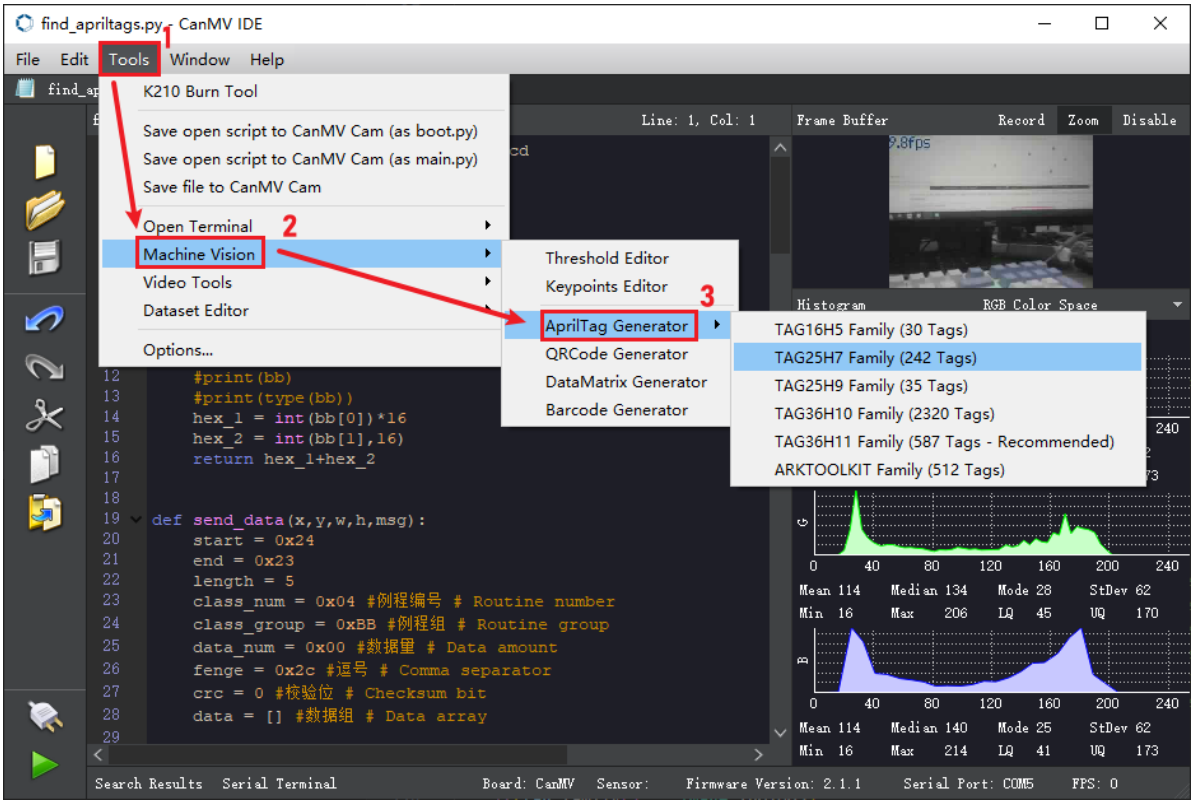
| **Function Prototype** | **void recv_k210msg(uint8_t recv_msg)** |
|---|---|
| Function Description | Processes single byte data from K210, implements message reception, verification and processing. Starts reception with packet header '$', ends and verifies with packet tail '#', calls deal_recvmsg() to process data if checksum is correct, otherwise resets receive state |
| Input Parameters | recv_msg: single byte data received from K210 |
| Return Value | None |

**Apr_Trace**

| **Function Prototype** | **void Apr_Trace(int x, int y, int w, int h)** |
|---|---|
| Function Description | Performs target tracking control based on input target area parameters (x, y, w, h). Applies first-order low-pass filtering to target center X coordinate and area, calculates X direction (left-right) and Y direction (forward-backward) control amounts through PID, finally calls Motion_Car_Control to control car movement; stops car if target area width/height is 0 |
| Input Parameters | x: target area top-left X coordinate y: target area top-left Y coordinate w: target area width h: target area height |
| Return Value | None |

# 4. CanMV IDE Generate Machine Code

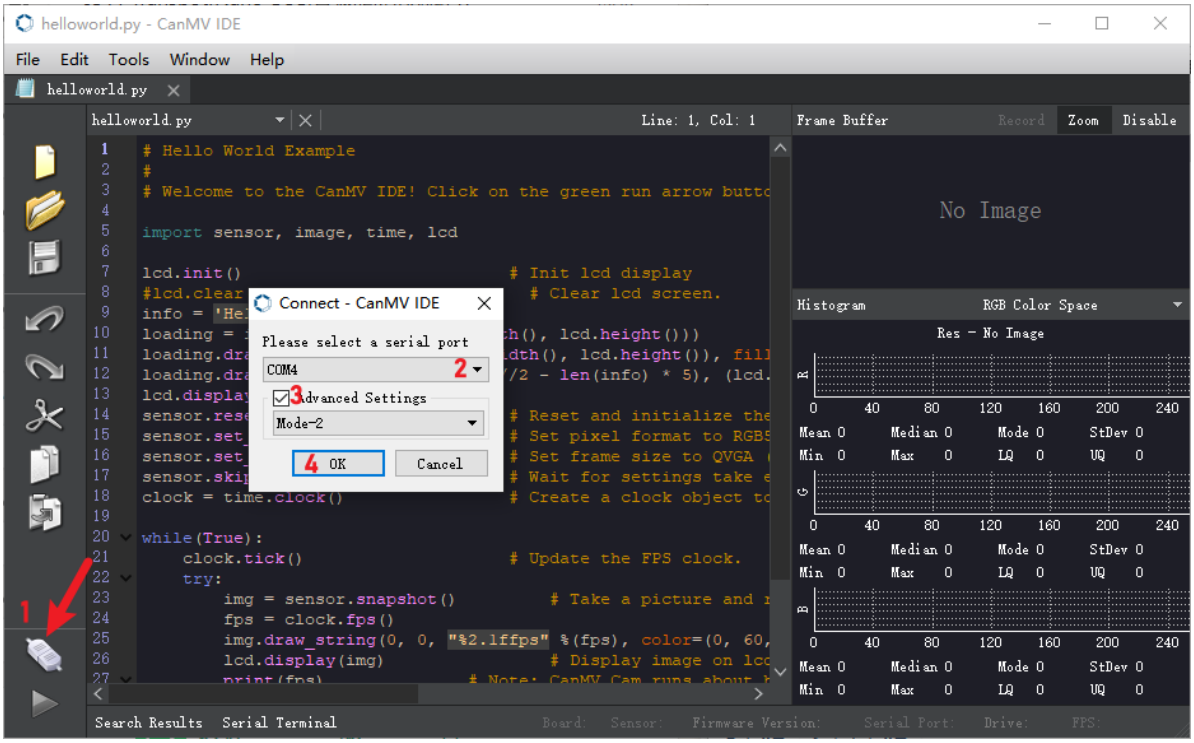
Open CanMV IDE menu bar -> Tools -> Machine Vision -> AprTag Generator to select corresponding Apr family



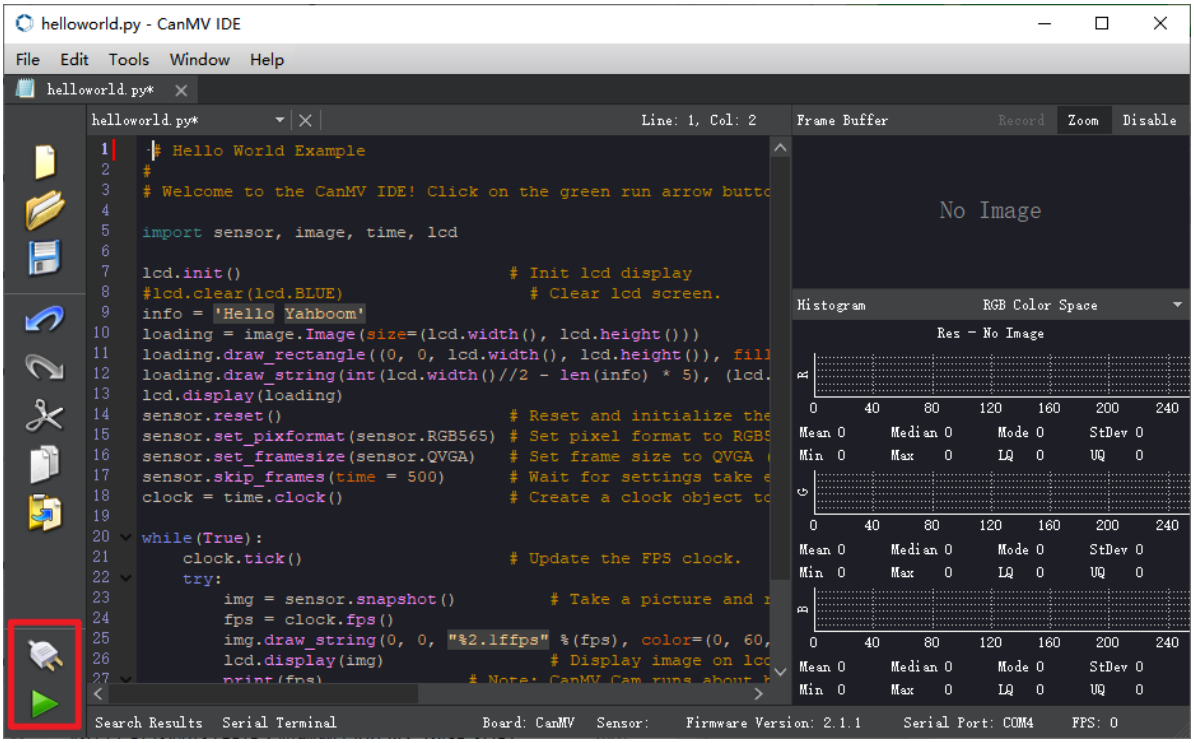

# 5. K210 Program Burning

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

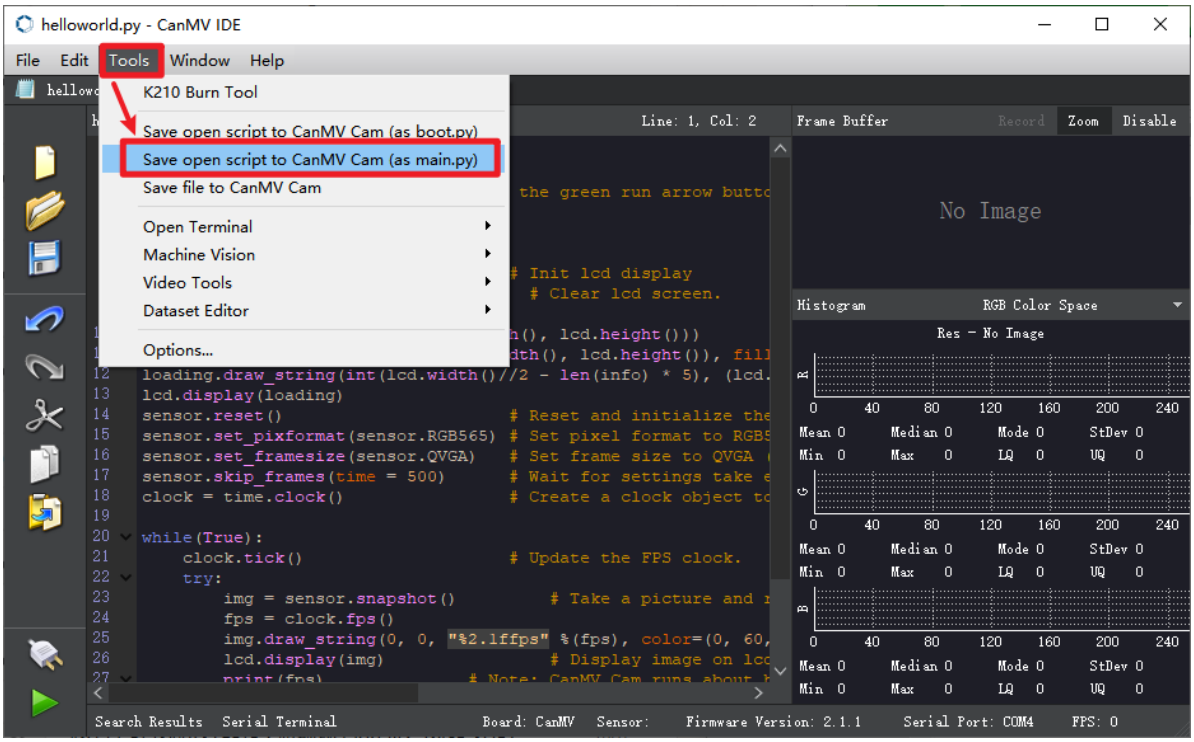
**here using helloworld.py as example**

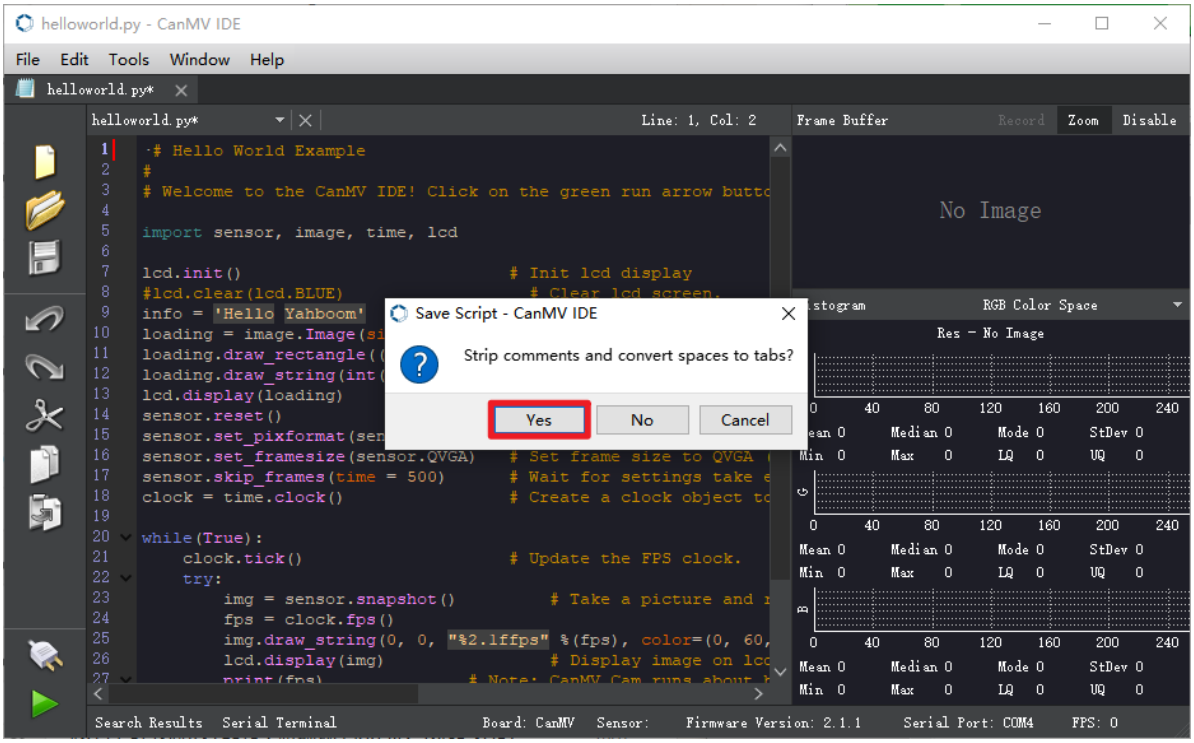



After successful IDE connection, the phenomenon is as follows

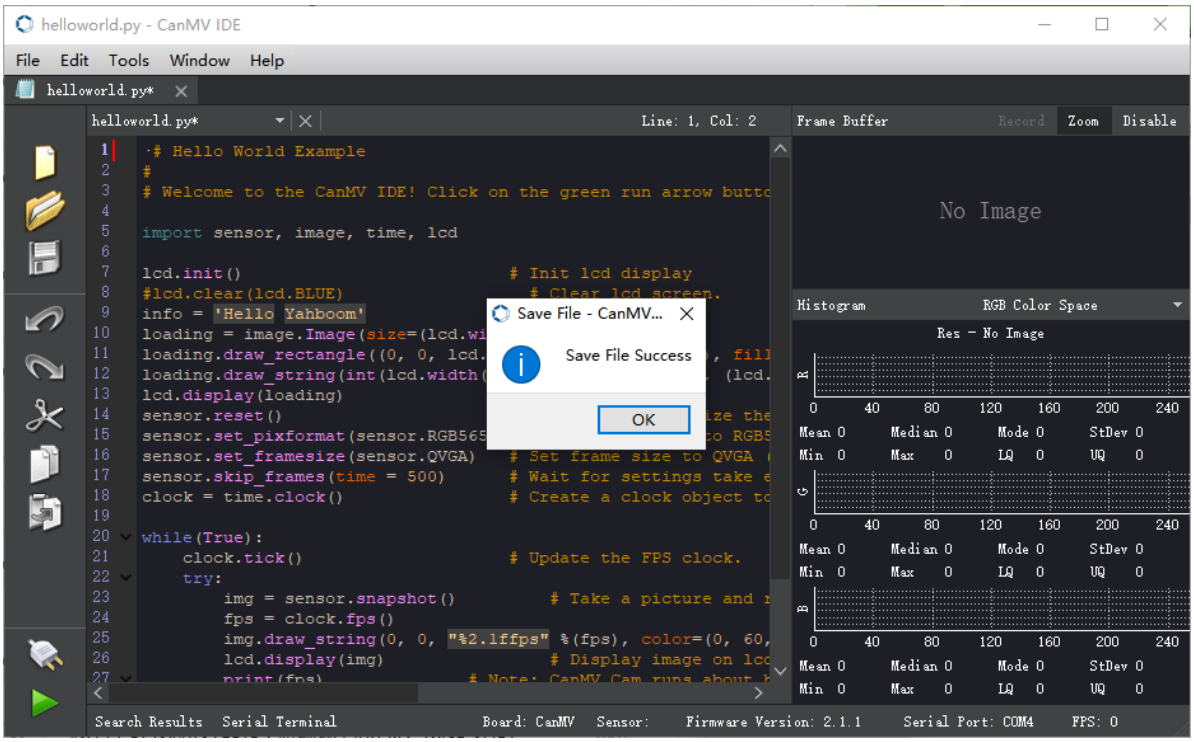



Here we use helloworld.py as an example, open the top menu bar Tools -> Save currently open script as (main.py) to CanMV Cam






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




# 6. Experimental Phenomena

After burning the program, the car will follow machine code by default. When the machine code moves away from the car, the car moves forward; when the machine code approaches the car, the car moves backward; when the machine code moves left/right, the car also moves left/right accordingly, keeping the recognized color in the center of the screen. It is recommended not to move the machine code too quickly, otherwise it may leave the recognition area.