# 运动目标追踪

回传JSON：

{

    "FUNC": "MOTION DETECT V1.0",

    "DIFF TOTAL": 10000, //画面变动率

    "DIFF MAX": 75, // 最高变化率

    "TOTAL": 3, //边界框数量

    "0": {

        "x": 45,

        "y": 18,

        "w": 126,

        "h": 72,

        "area": 342 //该边界框内变化像素的数量

    },

    "1": {

        "x": 0,

        "y": 169,

        "w": 130,

        "h": 24,

        "area": 173

    },

    "2": {

        "x": 39,

        "y": 204,

        "w": 276,

        "h": 34,

        "area": 141

    }

}

设置JSON：

{

    "MOTION DETECT": 1.0, *//功能标记，不可缺省*

    "mode": "COMPUTE\_MODE\_STATIC", *//可缺省 "COMPUTE\_MODE\_STATIC" 静态检测模式 or "COMPUTE\_MODE\_DYNAMIC" 动态检测模式*

    "thr\_w": 20, *//可缺省 边界框宽阈值，[3,200]*

    "thr\_h": 20, *//可缺省 边界框长阈值，[3,200]*

    "stepx": 1, *//可缺省 X扫描间隔，[0, 40]，设置为0则关闭边界框检测*

    "stepy": 2, *//可缺省 Y扫描间隔，[0, 40]，设置为0则关闭边界框检测*

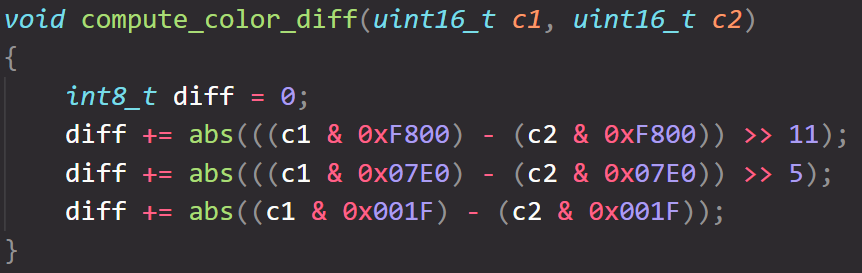
    "delta": 20, *//可缺省 变化率阈值，[0, 99]*

"merge": 10 *//可缺省 边界框合并阈值，[0, 40]*

}

\*设置JSON中标注的缺省项可以省略，省略即为保持默认/上次设置结果。

1. 画面变动率：两帧之间变化像素的变化量。假设有2个像素发生了变化，像素1变化了27，像素2变化了10，则该数值为27+10=37。将两个像素点R,G,B三个分量的差值求和即为变化量。



1. 最高变化率：变化最剧烈的像素的变化量
2. 目标ROI：目标的边界框，xy为矩形左上角坐标，wh为长宽。
3. area：变化像素的数量，假设有2个像素发生了变化，该值就是2。
4. 边界框长宽阈值：若（3）中目标ROI的长宽小于该值，则不发送这个边界框。
5. 扫描间隔：检测目标时的步进值，假设step x = 2， step y = 4，则横向每隔2个像素检测一次目标，纵向每隔4个像素检测一次目标。加大该项数值可以有效提升目标检测速度，但有可能漏掉较小的目标。
6. 变化率阈值：按（1）中的计算方式，变化量小于该数值的像素将不被认为发生了变化。
7. 动态检测模式：不断拍摄图像，比对前后两帧之间的变化。
8. 静态检测模式：拍一张照片并保存，比对新拍摄的图像与照片直接的变化。发送该指令时则自动拍摄，若要重新拍摄新的照片，需要先切换回动态检测模式再发送该指令。
9. 边界框合并阈值：合并两个距离小于该值的边界框。



# 目标追踪

回传JSON：

{

    "FUNC": "TARGET TRACKER V1.0",

    "x": 282,

    "y": 165,

    "w": 13,

    "h": 15

}

设置JSON：

{

    "TARGET TRACKER": " V1.0",

    "x": 282, //xywh均不可缺省

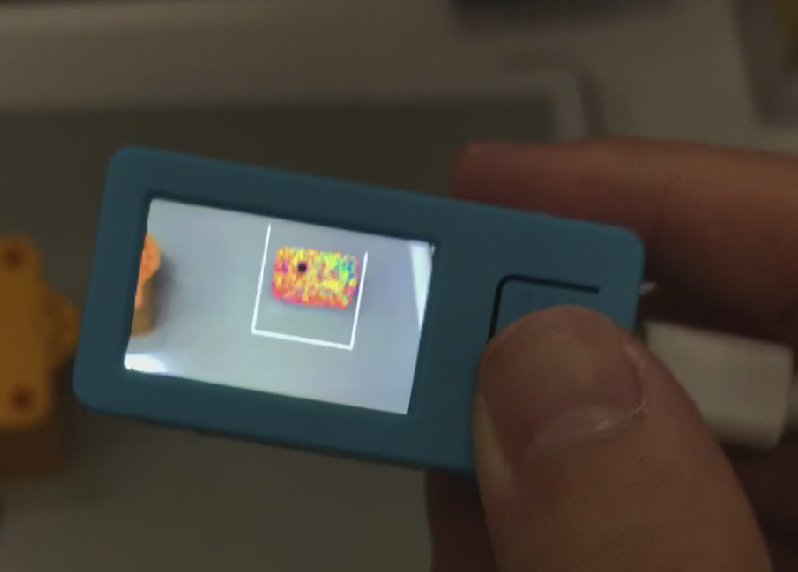
    "y": 165,

    "w": 13,

    "h": 15

}

1. 框出目标在画面中的位置，即可自动追踪框中的目标。目标最好具有显著的颜色特征。



# 色块识别

回传JSON：

{

    "FUNC": "COLOR TRACKER V1.0",

    "TOTAL": 3, *//边界框数量*

    "0": {

        "x": 45,

        "y": 18,

        "w": 126,

        "h": 72,

        "area": 342 *//该边界框内变化像素的数量*

    },

    "1": {

        "x": 0,

        "y": 169,

        "w": 130,

        "h": 24,

        "area": 173

    },

    "2": {

        "x": 39,

        "y": 204,

        "w": 276,

        "h": 34,

        "area": 141

    }

}

设置JSON：

{

    "COLOR TRACKER": 1.0, *//功能标记，不可缺省*

    "thr\_w": 20, *//可缺省 边界框宽阈值，[3,200]*

    "thr\_h": 20, *//可缺省 边界框长阈值，[3,200]*

    "stepx": 1, *//可缺省 X扫描间隔，[0, 40]，设置为0则关闭边界框检测*

    "stepy": 2, *//可缺省 Y扫描间隔，[0, 40]，设置为0则关闭边界框检测*

    "merge": 10, *//可缺省 边界框合并阈值，[0, 40]*

    "Lmin": 0, *//可缺省 L阈值下限 [0, 100]*

    "Lmax": 0, *//可缺省 L阈值上限 [0, 100]*

    "Amin": 0, *//可缺省 A阈值下限 [0, 255]*

    "Amax": 0, *//可缺省 A阈值上限 [0, 255]*

    "Bmin": 0, *//可缺省 B阈值下限 [0, 255]*

    "Bmax": 0, *//可缺省 B阈值上限 [0, 255]*

}

1. area：该颜色的像素数量
2. LAB颜色阈值：是LAB颜色空间的颜色值，在该范围外的颜色将会被过滤。这个值的选取需要专用的阈值编辑器实现，后续我会写一个小工具。
3. 检测步进，长宽阈值，合并阈值与运动目标追踪中所述一致。

# 人脸识别

1. 使用JSON作为数据包格式
2. {
3. "FUNC": "FACE DETECT",  *// 功能说明*
4. "count": 3,   *// 识别到的人脸数量*
5. "2": {  *// 人脸编号*
6. "x": 97,    *// ROI*
7. "y": 26,
8. "w": 64,
9. "h": 86,
10. "value": 0.859508,  *// 置信率*
11. "classid": 0,
12. "index": 2,
13. "objnum": 3
14. },
15. "1": {
16. "x": 70,
17. "y": 157,
18. "w": 38,
19. "h": 63,
20. "value": 0.712100,
21. "classid": 0,
22. "index": 1,
23. "objnum": 3
24. },
25. "0": {
26. "x": 199,
27. "y": 145,
28. "w": 31,
29. "h": 40,
30. "value": 0.859508,
31. "classid": 0,
32. "index": 0,
33. "objnum": 3
34. }
35. }

# 二维码识别

回传JSON

QRCODE模式

{

    "count": 1,

    "FUNC": "FIND QRCODE",

    "0": {

        "x": 57,

        "y": 16,

        "w": 197,

        "h": 198,

        "payload": "m5stack", //二维码数据

        "version": 1, //二维码版本

        "ecc\_level": 1, //二维码ECC水平

        "mask": 2, //二维码掩码

        "data\_type": 4, //二维码数据类型

        "eci": 0 //返回二维码的ECI。

    }

}

DATAMATRIX模式

{

    "0": {

        "x": 20,

        "y": 116,

        "w": 96,

        "h": 96,

        "payload": "m5stack",

        "rotation": 1.588250, //DM码旋转角度

        "rows": 16, //DM码行数

        "columns": 16, //DM码列数

        "capacity": 12, //DM码容量（字节）

        "padding": 1 //DM码剩余容量（字节）

    },

    "count": 1,

    "FUNC": "FIND DATAMATRIX"

}

APRILTAG模式

{

    "0": {

        "x": 71,

        "y": 5,

        "w": 85,

        "h": 88,

        "id": 1,

        "family": 16,// AprilTag的类别

        "cx": 115,

        "cy": 49,

        "rotation": 6.219228,// 返回以弧度计的AprilTag的旋度(int)。

        "decision\_margin": 0.451959,// AprilTag匹配的色饱和度（取值0.0 - 1.0），其中1.0为最佳。

        "hamming": 0,// AprilTag的可接受的数位误差数值

        "goodness": 0.000000, //AprilTag图像的色饱和度

        "x\_translation": 0.868200,

        "y\_translation": 0.245313,

        "z\_translation": -2.725188,

        "x\_rotation": 3.093776,

        "y\_rotation": 0.065489,

        "z\_rotation": 6.219228

    },

    "count": 1,

    "FUNC": "FIND APRILTAG"

}

BARCODE模式

{

    "0": {

        "x": 62,

        "y": 90,

        "w": 100,

        "h": 45,

        "payload": "123", *//数据*

        "type": 15, *//条码类别*

        "rotation": 0.000000, *//条码旋转角度*

        "quality": 28 *//条码在图像中被扫描的次数*

    },

    "count": 1,

    "FUNC": "FIND BARCODE"

}

设置JSON:

{

    "FIND CODE": 1.0,

    "mode":"DATAMATRIX" *//识别模式，可选QRCODE，APRILTAG，DATAMATRIX,BARCODE*

}

# JPEG图像传输

回传格式 JPEG：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | | 13 | 14 | | 15 | 16 | | 17 | | 18 | 19 | 20 | 20+len | |
| A5 | 5A | AA | 55 | frame，当前帧计数，uint64\_t | | | | | | | | | w，图像宽度 uint16\_t | | | h，图像高度 uint16\_t | | | jpeg图像质量 | | jpeg图像大小，uint32\_t字节 | | | | | jpeg图像数据 |
| 数据包头 | | | |

回传格式 BMP：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | | 13 | 14 | | 15 | 16 | |
| 5A | 55 | A5 | AA | frame，当前帧计数，uint64\_t | | | | | | | | | w，图像宽度 uint16\_t | | | h，图像高度 uint16\_t | | | BMP图像数据，长度w\*h\*2+66 | |
| 数据包头 | | | |

设置json格式：

{

    "JPEG TRANSFER": 1.0, *//不可缺省*

    "takepic": 1, *//可缺省，值任意，只要有这个键就会进入一次BMP回传模式*

    "imgdiv": 1, *//可缺省，值可取1，2，4。对应分辨率320\*240， 160\*120， 80\*60*

    "quality": 60 *//可缺省，值0~99，但实际上只有3个等级的质量。 阈值为 >=60, >35, >0*

}

关于一次BMP图像的传输：

1. 主机发送包含“takepic”键的json
2. 设备回传bmp数据包的包头，5A 55 A5 AA
3. 主机发送任意数据给设备
4. 设备开始回传BMP数据包

\*Arudino端接收图像可参考<https://github.com/m5stack/V_IMG_Reciver>，协议有细微差别，仅供参考。

# 自定义标签识别

回传JSON格式

{

    "FUNC": "TAG READER V2.0",

    "TOTAL": 1,

    "0": {

        "x": 113,

        "y": 65,

        "w": 117,

        "h": 105,

        "p0x": 113, *// p0x ~ p3y: TAG 4个顶点的坐标*

        "p0y": 77,

        "p1x": 211,

        "p1y": 65,

        "p2x": 230,

        "p2y": 156,

        "p3x": 127,

        "p3y": 170,

        "rotation": 8, *// TAG 的相对旋转角度*

        "rows": 8, *// TAG 的行数（本数值不含定位框）*

        "columns": 8, *// TAG 的列数（本数值不含定位框）*

        "size": 64, *// TAG 实际内容的数据长度，该值 = 内容的行数 \* 内容的列数 = (rows) \* (columns)*

        "code": "0x003C42425A424200", *// uint64\_t类型的内容二进制代码，本键值最大编码64位（8 x 8）的TAG*

        "binstr": "0000000000111100010000100100001001011010010000100100001000000000" *//二进制数据的字符串形式，本键值可以编码任意长宽的TAG*

    }

}

目前程序限制识别7x7 与 10x10两种格式，其它格式会被过滤掉。

\*关于code与binstr的解码：

Code键值需要转换为对应的2进制序列，如0x003C42425A424200就对应0000000000111100010000100100001001011010010000100100001000000000，随后按照行列数从左至右进行排列，如示例中的8行8列可以转换为

00000000

00111100

01000010

01000010

01011010

01000010

01000010

00000000

将1提取出来，可得对应的字符

00000000      00000000

00111100      00@@@@00        @@@@

01000010      0@0000@0       @    @

01000010  ->  0@0000@0  ->   @    @

01011010      0@0@@0@0       @ @@ @

01000010      0@0000@0       @    @

01000010      0@0000@0       @    @

00000000      00000000