智能车图像处理辅助文档(By 闲鱼：玛卡巴卡的杂货铺)：

移植：

第一步就是移植，这一步请务必自行完成，为什么呢？因为已经有先例（不止一例），图省事或者不会移植，说让我代劳，我也图省事(也没收钱)就代劳了，毕竟程序是我自己写的，我移植也就几分钟的事，于是乎他就跳过了自己移植这一步，这就导致了他连函数看都不看，遇到问题啥都不想，直接拿来问，甚至连显示自己的图像效果都不会，觉得是我的程序有问题，我就又帮忙找问题，结果发现是自己的库里的显示例程调用都错了。如果都是这样，智能车谈何容易。

好了，题外话结束，咱们还是正经开始移植。

1.首先.c和.h文件可以自己建，也可以直接改后缀名。然后在自己的工程里添加就行了。(添加文件根据自己的IDE不同选择不同的方式)

2.等.c和.h都放到工程模板里了，不要急着编译，因为有些东西需要改，接下来说那些地方需要改：

Tips：如果打开注释乱码，edit→encoding→把编码改成UTF-8就行了。

首先输入图像要改，改成你自己的摄像头采集到的图片的地址。

```
Get_image(mt9v03x_image);
```

改成你自己摄像头采集到的图像的首地址

类型定义需要改，如果你是用的逐飞的库函数，把#include "headfile.h"放进来，然后直接删掉就行。如果是龙邱，也有类型定义，在#include"Platform_Types.h"里，但是int8和int16以及int32，需要改成sint8，sint16，以及sint32（如下图）。

```
//数据类型声明（方便移植—移植的时候可以删掉，改成你自己的）
typedef    signed          char int8;
typedef    signed short     int int16;
typedef    signed           int int32;
typedef  unsigned          char uint8;
typedef  unsigned short     int uint16;
typedef  unsigned           int uint32;
```

```
21
22  sint16 limit_a_b(sint16 x, int a, int b)
23  {
24      if(x<a) x = (sint16)a;
25      if(x>b) x = (sint16)b;
26      return x;
27  }
28
```

显示函数需要改（这里多说两句，改之前请仔细阅读你自己所用的屏幕的资料，例程至少用对，不要上来就问，这个画点的怎么改，哪个图像的显示怎么又不对）画点需要填写的横纵坐标已经给出，根据自己屏幕的画点函数填入参数显示就行，有的库颜色定义不一样，请根据自身情况更改。

```
ips154_drawpoint(l_border[i], i, RED);//显示起点 显示左边线
ips154_drawpoint(r_border[i], i, BLUE);//显示起点 显示右边线
```

横坐标　纵坐标　颜色

显示图像的函数因为各家库函数的不同，写的各有特点，参数位置不一，但是图像首地址是肯定是一样的。所以我给个简单的例子，具体情况具体分析，请自行判断。

```
diplayimage(image[0], 188, 120);
```

图像首地址　图像宽度　图像高度

再多唠叨一句，不管是逐飞库还是龙邱库，例程真的写的很好，参数该输入什么，怎么调用，都写得很详细的，一定要看，一定要亲自动手操作一下。

```
//-------------------------------------------------------------------------------------------------------------------
// @brief       TFT180 显示二值图像 数据每八个点组成一个字节数据
// @param       x               坐标x方向的起点 参数范围 [0, tft180_x_max-1]
// @param       y               坐标y方向的起点 参数范围 [0, tft180_y_max-1]
// @param       *image          图像数组指针
// @param       width           图像实际宽度
// @param       height          图像实际高度
// @param       dis_width       图像显示宽度 参数范围 [0, tft180_x_max]
// @param       dis_height      图像显示高度 参数范围 [0, tft180_y_max]
// @return      void
// Sample usage:                tft180_show_binary_image(0, 0, ov7725_image_binary[0], OV7725_W, OV7725_H, OV7725_W / 2, OV7725_H / 2);
//-------------------------------------------------------------------------------------------------------------------
void tft180_show_binary_image (uint16 x, uint16 y, const uint8 *image, uint16 width, uint16 height, uint16 dis_width, uint16 dis_height)
{
    // 如果程序在输出了断言信息 并且提示出错位置在这里
    // 那么一般是屏幕显示的时候超过屏幕分辨率范围了
    zf_assert(x < tft180_x_max);
    zf_assert(y < tft180_y_max);
```

最后就可以把头文件放在 main.c，调用 imag_process()放循环就可以编译了。

```
#include "image.h"

int main(void)
{
//关闭总中断
__disable_irq();
board_init();//务必保留，本函数用于初始化MPU 时钟 调试串口
//此处编写用户程序
LED_Init();
ips154_init();
//总中断最后开启
__enable_irq();
  while(1)
  {
  led_toggle(100);
  image_process();
  }
}
```

至此，移植就已经完成了，烧录就可以看效果了（请放到正经赛道上看，不然会卡循环）。

补充：放循环时记得加上摄像头采集结束标志位。

```
    tft180_set_color(RGB565_BLUE, RGB565_RED);
    while(1)
    {
       if (mt9v03x_finish_flag_dvp == 1)
        {
            mt9v03x_finish_flag_dvp=0;
            image_process();
        }

    }
}
```

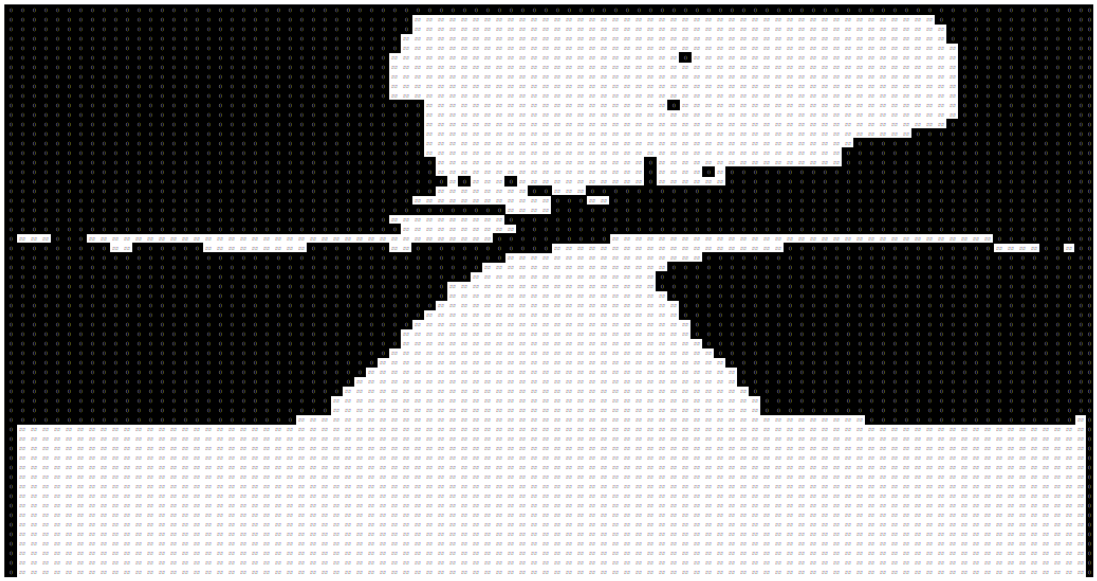
常见报错：

内存不够报错：这个必须强制压缩图片，把图像高度宽度（包括摄像头采集到的宽度和高度也改为一致大小），减小一点，直到不爆栈为止。

```
//宏定义
#define image_h 120//图像高度
#define image_w 188//图像宽度
```

广告：如果实在不想自己移植，可以代劳， 10 元保证移植成功。

原理：



接下来我们来解析一下原理，梳理一下框架，演示一下程序是如何运行的，这里为了更好的演示效果，我就用串口打印了一张二值化之后的图放在了 EXCEL 里，方便单个像素点的演示，感兴趣的可以学着打印出来，放 excel 里分列一下就可以了，没什么操作难度。

首先就是这八个邻域：我定义了两个方向循环，一个顺时针，一个逆时针，循环起点为 1 号位置。

4	5	6
3	中心点	7
2	1	8（0）

顺时针

6	5	4
7	中心点	3
8（0）	1	2

逆时针

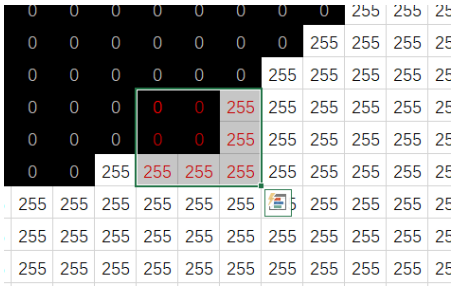
变量如下：

```
static int8 seeds_l[8][2] = { {0,  1},{-1,1},{-1,0},{-1,-1},{0,-1},{1,-1},{1,  0},{1, 1}, };
     //{-1,-1},{0,-1},{+1,-1},
     //{-1, 0},  中点  {+1, 0},
     //{-1,+1},{0,+1},{+1,+1},  左边顺时针
static int8 seeds_r[8][2] = { {0,  1},{1,1},{1,0}, {1,-1},{0,-1},{-1,-1}, {-1,  0},{-1, 1}, };
     //{-1,-1},{0,-1},{+1,-1},
     //{-1, 0},  中点  {+1, 0},
     //{-1,+1},{0,+1},{+1,+1},  右边逆时针
```

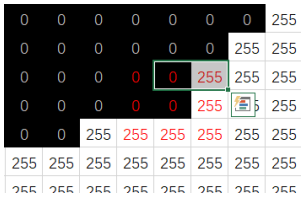
上面的中心点假设为(1,1),那么程序里要表示这八个点也很简单，用中心点加上上面的变量就行了。如下图所示：

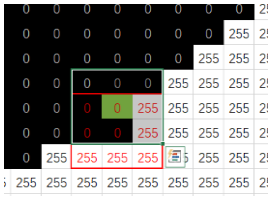
(0.0)	(0.1)	(0.2)		(1.1)	(1.1)	(1.1)		(-1.-1)	(0.-1)	(1.-1)
(1.0)	(1.1)	(1.2)	=	(1.1)	(1.1)	(1.1)	+	(-1.0)	(1.1)	(1.0)
(2.0)	(2.1)	(2.2)		(1.1)	(1.1)	(1.1)		(-1.1)	(0.1)	(1.1)

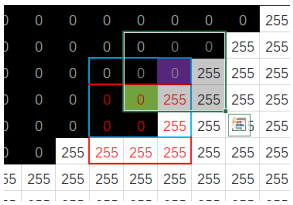
八个邻域知道了，那么八领域循环简单来说就是以某一个点为中心点，根据限制条件来在八个邻域中找到满足条件的点记录下来作为新的中心点的循环。



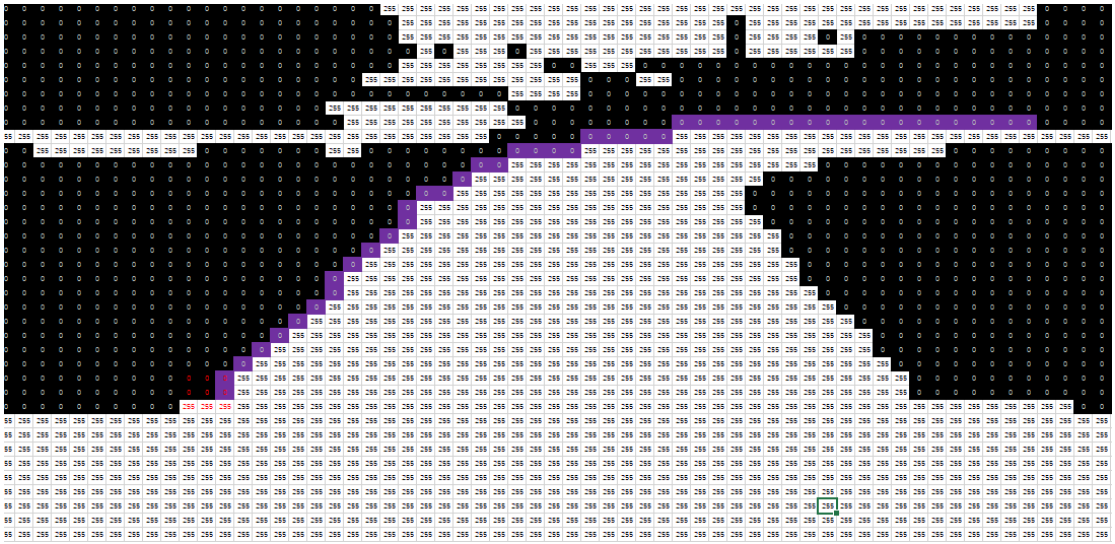
就比如上图这9个点，我们围绕中间的0为中心，从1号位开始，逆时针循环进行比对，当满足黑点跳到白点（也就是0→255）时，就更新坐标中心点(本程序去的是黑点所以就取的值为0的点也就是下图填充绿色的点)，然后以新的中心点开始新的逆时针循环再找到下一个的中心点，以此类推，紫色点就为下一个中心点。(右边同理，不再赘述)







一直这样循环下去，一直取黑点我们就能等得到一条完整的边界了。

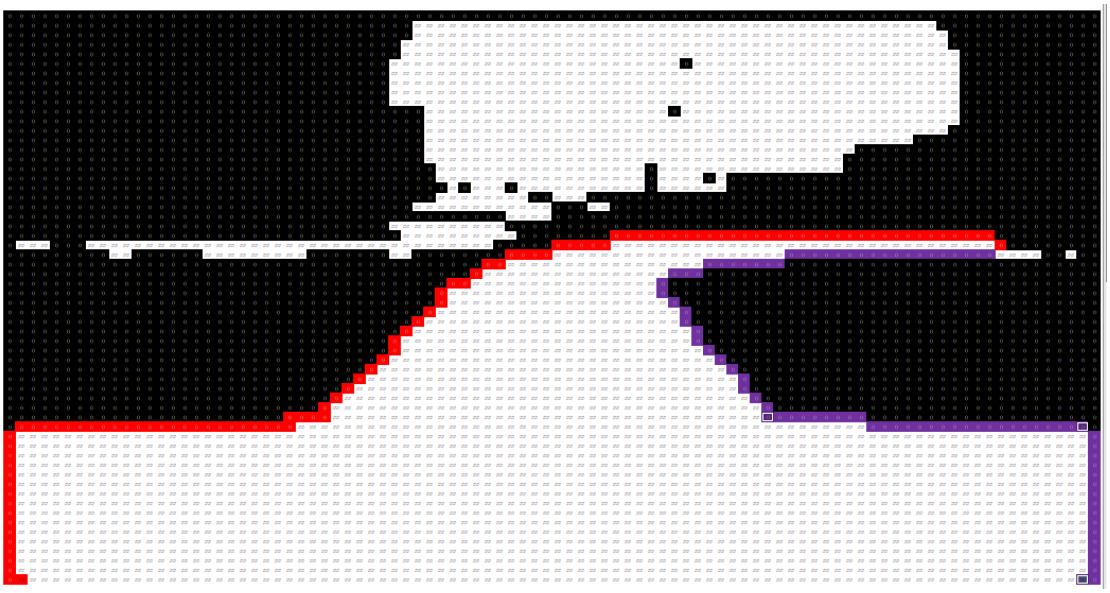


领域判断大致思想就是如此。这也是下面这一段程序的主要思想，也就是本程序的精华所在。

Tips：程序循环八个点是写的取余是%8，但是为了优化，改成了&7，但是作用是一样的，望周知。

```
if (image[search_filds_l[i][1]][search_filds_l[i][0]] == 0
    && image[search_filds_l[(i + 1) % 8][1]][search_filds_l[(i + 1) % 8][0]] == 255)
{
    temp_l[index_l][0] = search_filds_l[(i)][0];
    temp_l[index_l][1] = search_filds_l[(i)][1];
    index_l++;
    dir_l[data_statics-1] = (i);
    //printf("dir_l[data_statics-1]:%d\n", dir_l[data_statics - 1]);
}
```

于是乎，从图像最底部的中间开始往两边先找到两个白→黑的点作为起点，然后从下往上执行领域循环，加上其他的(限制条件)最终就能得到如下图所示效果。



那么领域循环就到此打住，我们根据领域循环函数（下图）可以得到左右轮廓和生长方向，下面演示一下有效边界如何获得。

```
// 八邻域提取边界
search_l_r((uint16)USE_num,bin_image,&data_stastics_l, &data_stastics_r,start_point_l[0], start_point_l[1], start_point_r[0], start_point_r[1],&hightest);
```

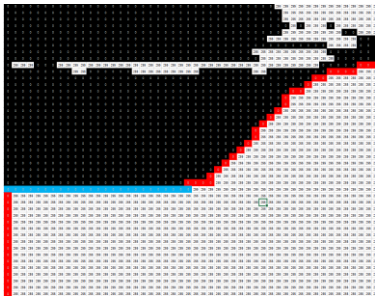
Tips:这里贴一个 continue 的用法，有的小伙伴不是很熟悉,C 语言基本功，务必掌握。

(10 条消息) continue 的用法详解_无赖 H4 的博客-CSDN 博客_continue 的用法

正常情况我们应该得到类似上图红色/紫色部分的有一系列边界点组成的轮廓边界，红色为左边轮廓，紫色为右边轮廓。显然直接拿这样的轮廓求中线肯定是不行的，中线大概率会很

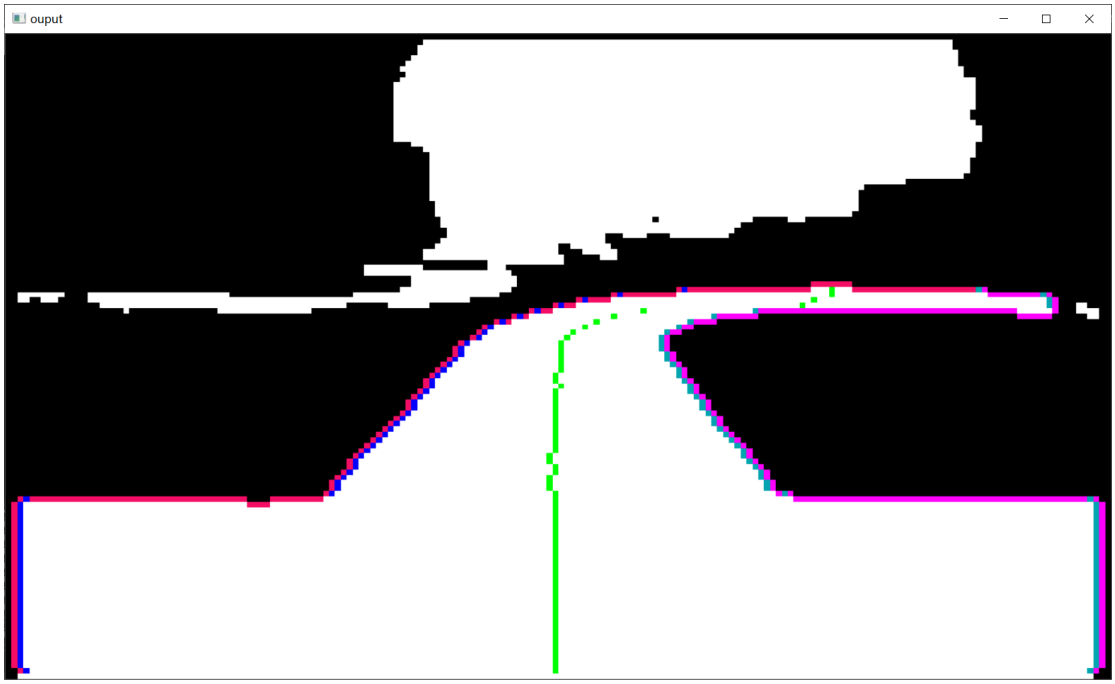
怪，而且图形越怪，中线越飘，所以我们要想办法从中提取有用的边界信息从而得出全新的中线。

这里我的思路是（以左边为例子，右边是一样的道理）在找到的轮廓中每一行只取一个点，当遇到蓝色部分时（如下图），他们的 y 坐标都是相同的，并且一行有多个点，但是一行点里面的大部分都是没用的，所以我们只按着 for 循环的顺序依次取第一个点就行了（后面的以此类推）



```
//左边
for (j = 0; j < total_L; j++)
{
    //printf("%d\n", j);
    if (points_l[j][1] == h)
    {
        l_border[h] = points_l[j][0]+1;
    }
    else continue; //每行只取一个点，没到下一行就不记录
    h--;
    if (h == 0)
    {
        break;//到最后一行退出
    }
}
```

按照每一行只取一个点的逻辑，我们就可以得到图是效果（Open CV 打印），这里为了方便观察，提取出来的有效边界我往中间(左边右移，右边左移)挪了一个点的位置，避免原来的边界点被覆盖，但是程序确实是取得第一个点。



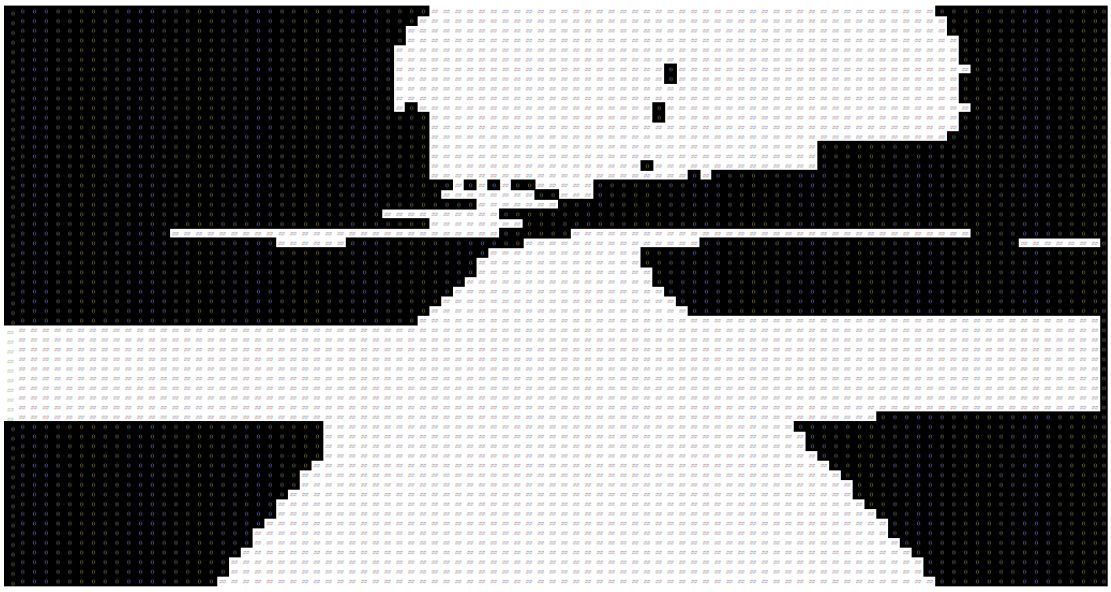
至此，边界提取就已经完成了。

从上面的过程中我们了解了邻域循环的过程并且解释了怎么提取有用的边界，如果没看懂可以多看两遍，接下来我们来说生长方向的问题。

Tips：dir 记录的生长方向信息只是个过程信息，不带有 x，y 坐标，x，y 坐标是储存在 points 数组里面的。不过 dir 的数组下标和 points 的数组下标是一致的，所以只需要访问相应的下标对应的 points，就可以得到 dir 的 x，y 坐标。列如 dir[35]对应 points[35]。

那么生长方向有什么用呢？提前学习了解过的就应该知道，它可以作为判断条件来判断元素，并且十分便捷。

这里我就拿十字举例了，同样的我用串口打印了一份图纸。



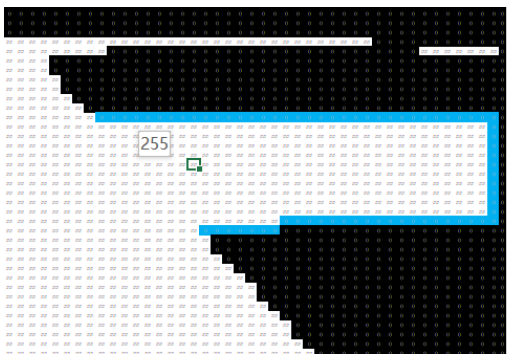
接下来我们来看分析他的生长方向，我们首先用串口打印一下它的生长方向看看效果：

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	dir[0]	3		dir[21]	2		dir[42]	4		dir[63]	6		dir[84]	4
2	dir[1]	5		dir[22]	2		dir[43]	4		dir[64]	6			
3	dir[2]	5		dir[23]	2		dir[44]	4		dir[65]	6			
4	dir[3]	5		dir[24]	2		dir[45]	4		dir[66]	6			
5	dir[4]	4		dir[25]	2		dir[46]	4		dir[67]	6			
6	dir[5]	5		dir[26]	2		dir[47]	5		dir[68]	6			
7	dir[6]	5		dir[27]	2		dir[48]	6		dir[69]	6			
8	dir[7]	5		dir[28]	2		dir[49]	6		dir[70]	6			
9	dir[8]	4		dir[29]	2		dir[50]	6		dir[71]	6			
10	dir[9]	5		dir[30]	2		dir[51]	6		dir[72]	6			
11	dir[10]	5		dir[31]	2		dir[52]	6		dir[73]	6			
12	dir[11]	5		dir[32]	2		dir[53]	6		dir[74]	6			
13	dir[12]	5		dir[33]	2		dir[54]	6		dir[75]	6			
14	dir[13]	4		dir[34]	2		dir[55]	6		dir[76]	6			
15	dir[14]	3		dir[35]	2		dir[56]	6		dir[77]	6			
16	dir[15]	2		dir[36]	2		dir[57]	6		dir[78]	6			
17	dir[16]	2		dir[37]	3		dir[58]	6		dir[79]	6			
18	dir[17]	2		dir[38]	4		dir[59]	6		dir[80]	6			
19	dir[18]	2		dir[39]	4		dir[60]	6		dir[81]	5			
20	dir[19]	3		dir[40]	4		dir[61]	6		dir[82]	5			
21	dir[20]	2		dir[41]	4		dir[62]	6		dir[83]	5			
22														

Tips：dir 是通过 debug 串口用 printf 函数打印出来的，每个人情况不同，关于 printf 怎么才能用这个请不要来问我，我真的帮不了你。

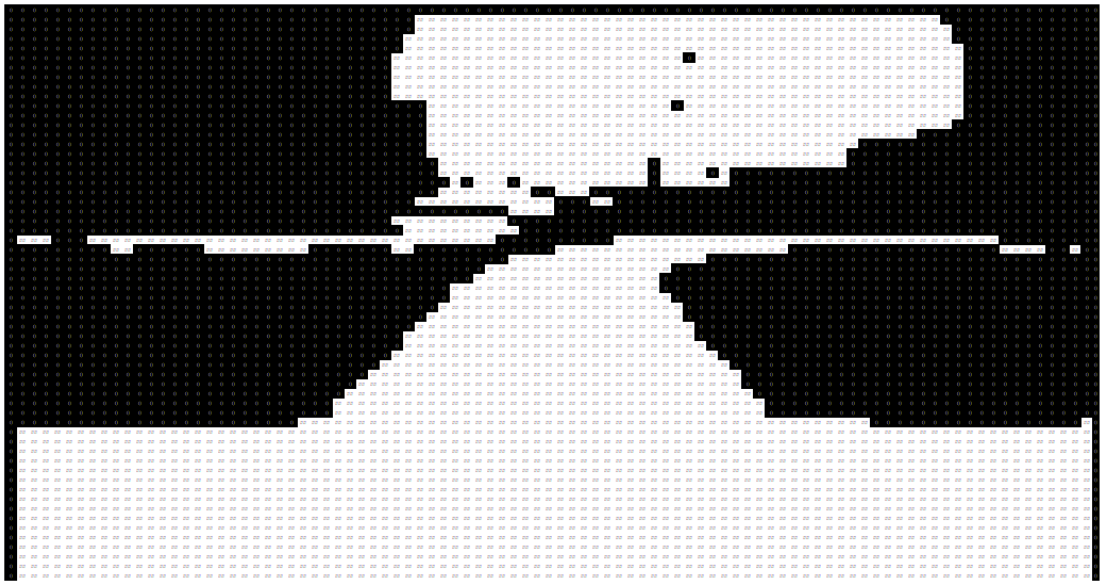
一共 84 组数组，我们可以发现其中，2,4,6 三个方向对应的都有较长的连续的一段，这就说明八领域找出来的轮廓至少有三段直线，根据上面我们自主定义的八个方向，我们安全可以得出结论，这三条直线就是下图中对应的蓝色部分。

D	E	F	G	H	I	J	K
dir[21]	2		dir[42]	4		dir[63]	6
dir[22]	2		dir[43]	4		dir[64]	6
dir[23]	2		dir[44]	4		dir[65]	6
dir[24]	2		dir[45]	4		dir[66]	6
dir[25]	2		dir[46]	4		dir[67]	6
dir[26]	2		dir[47]	5		dir[68]	6
dir[27]	2		dir[48]	6		dir[69]	6
dir[28]	2		dir[49]	6		dir[70]	6
dir[29]	2		dir[50]	6		dir[71]	6
dir[30]	2		dir[51]	6		dir[72]	6
dir[31]	2		dir[52]	6		dir[73]	6
dir[32]	2		dir[53]	6		dir[74]	6
dir[33]	2		dir[54]	6		dir[75]	6
dir[34]	2		dir[55]	6		dir[76]	6
dir[35]	2		dir[56]	6		dir[77]	6
dir[36]	2		dir[57]	6		dir[78]	6
dir[37]	3		dir[58]	6		dir[79]	6
dir[38]	4		dir[59]	6		dir[80]	6
dir[39]	4		dir[60]	6		dir[81]	5
dir[40]	4		dir[61]	6		dir[82]	5
dir[41]	4		dir[62]	6		dir[83]	5



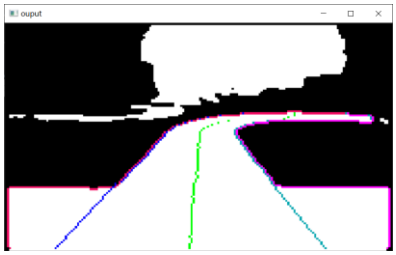
通过对比，也不难看出 2 对应下面较短的横线，4 的个数最少，对应竖着的一段，6 的个数最多必然对应上面的一段最长的横线。所以我们就可以根据相同的 dir 特征判断十字。

理论存在，那么我们直接趁热打铁，给下面的这幅图写一个补线函数。



首先，我们上面得出了结论，也就是**竖着的轮廓生长方向为 4，黑色在上面的横轮廓生长方向为 6**，所以我们就以此为判断条件写一个补线程序。(点击查看补线程序)

因为 EXCEL 看不了结果且我身边没有车模和赛道，但是补线结果还是得拿出来看一看，证明确实补到位了，所以我就移植到 C8T6 上显示了一下结果，再加上一份 opencv 显示效果。

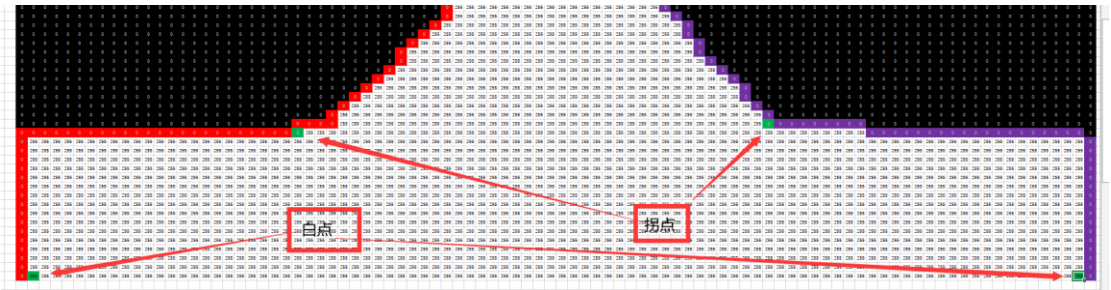


稳稳能跑，效果还不错。

补充：这个补线函数本来打算是不公开的，但是禁不住很多人问，再加上不给个例子出来对部分人来说确实很难上手，毕竟上课做题还得给个例题不是嘛，所以程序内涉及到的最小二乘法和计算斜率截距的函数我也会在后面给出（点击查看斜率截距函数）

然后接下来解释一下这段程序：

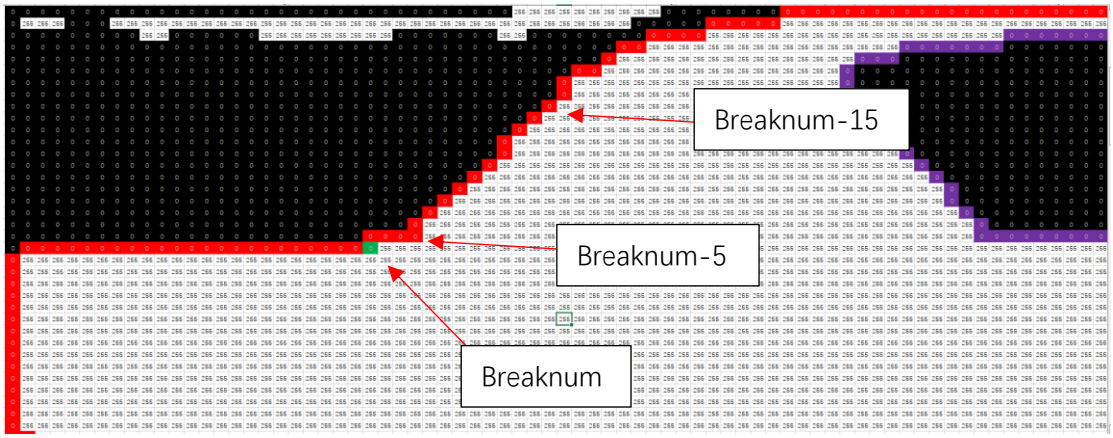
根据图像的特征我们可以发现图像的生长方向由 4 到 6 的转折找到第一个拐点和以及图像下方大量留白我们也可以作为辅助判断条件。(如下图绿色的点：上面的为拐点，下面的为白点)



通过遍历 dir 数组我们可以找到拐点的对应的 dir 下标，记录 dir 下标，然后传递给 points，再传递给 breaknum(`break_num_l = points_l[i][1];//传递 y 坐标`)把 breaknum 减去 15 作为计

算斜率的起点，breaknum 减去 5 作为计算斜率的终点，进而求得斜率和截距。类似下图所示（数据对不上请勿较真，知道是那么个意思就行了）：

```
//计算斜率
start = break_num_r - 15;//起点
start = limit_a_b(start, 0, image_h);//限幅
end = break_num_r - 5;//终点
calculate_s_i(start, end, r_border, &slope_l_rate, &intercept_l);
```



然后就从 breaknum 开始，一直到图像的最底部补线：这里有人不理解，为什么就用到这个 border 了，不应该是 points 嘛？他们有什么关系？

这里解释一下（因为前面已经说过了，dir 和 points 是对应的，而 border 是从 points 里提取出来的，所以 border 和 points 有一定的对应关系　　例如 border[5] = points[5][0]）我们补线最终补得也是 border，并非 points 所以是对 border 进行操作，其中 border 存的是横坐标，那么 y 去哪了呢？其实啊 border[35] = = 50 中 35 就是 y，50 就是 x

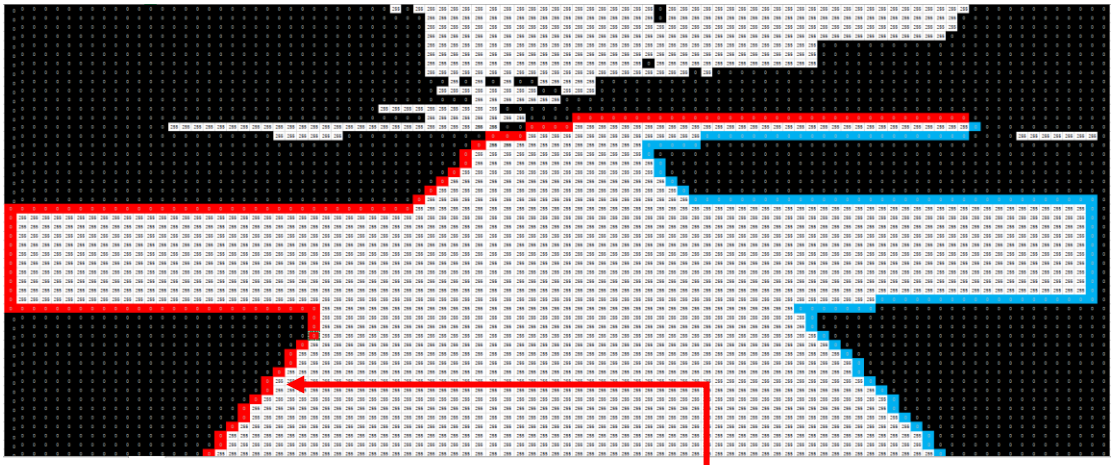
```
for (i = break_num_l - 5; i < image_h - 1; i++)
{
    l_border[i] = slope_l_rate * (i)+intercept_l;//y = kx+b
    l_border[i] = limit_a_b(l_border[i], border_min, border_max);//限幅
}
```

通过 y=kx+b 的思路，以 breaknum 为起点，往下依次给 border 赋值。就能补好这个十字了，讲的不好，请在给出的程序中细细体会。这就是整个补线的过程了。

Tips：肯定会有人问，判断条件里的那两个白点是干什么的，是用来加强判断条件的，因为不加两个白点，可能会跟（点击查看图片）判断成一样的十字，这样就不够严谨。也就是下图给出的条件。

```
&&image[image_h - 1][4] && image[image_h - 1][image_w - 4]
```

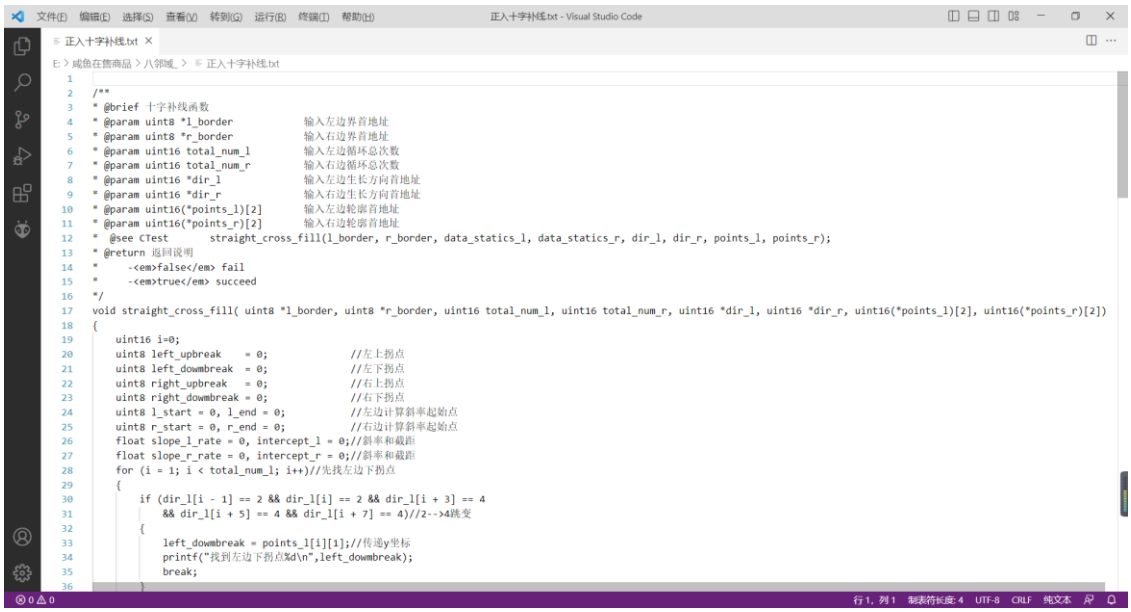
那么我自己给的例子也就分享完了，下面我再分享出一张图的思路，请自行编写处理程序。



1. 生长方向 2→4→6 可以找出四个拐点（左右各两个），2→4 和 4→6 (左右都一样)
2. 左下角和右下角大量黑点
3. 图像下方斜率较稳定，可利用下方斜率补线
4. 也可以不用最小二乘法，直接连接两个拐点也可以 y = kx+b ，已知两个点求直线表达式。

请自己对自己负责，认真思考后编写程序，出了问题请尝试 debug 自己找问题，自己不写程序就永远都不会写，自己不找问题就永远都不知道怎么找，永远只能当个只会问的大笨蛋呀，所以这个图就不给示例了。

皮一下：如果实在不想对自己负责，诶，我就想用现成的，店铺下单上图十字的程序，只有注释且不包讲解，嘿，你小子，请慎重考虑哦。



正入十字补线.txt - Visual Studio Code

正入十字补线.txt

E: > 咸鱼在售商品 > 八邻域_ > 正入十字补线.txt

```
/**
* @brief 十字补线函数
* @param uint8 *l_border            输入左边界首地址
* @param uint8 *r_border            输入右边界首地址
* @param uint16 total_num_l         输入左边循环总次数
* @param uint16 total_num_r         输入右边循环总次数
* @param uint16 *dir_l              输入左边生长方向首地址
* @param uint16 *dir_r              输入右边生长方向首地址
* @param uint16(*points_l)[2]       输入左边轮廓首地址
* @param uint16(*points_r)[2]       输入右边轮廓首地址
*  @see CTest       straight_cross_fill(l_border, r_border, data_statics_l, data_statics_r, dir_l, dir_r, points_l, points_r);
* @return 返回说明
*     -<em>false</em> fail
*     -<em>true</em> succeed
*/
void straight_cross_fill( uint8 *l_border, uint8 *r_border, uint16 total_num_l, uint16 total_num_r, uint16 *dir_l, uint16 *dir_r, uint16(*points_l)[2], uint16(*points_r)[2])
{
    uint16 i=0;
    uint8 left_upbreak    = 0;              //左上拐点
    uint8 left_dowmbreak  = 0;              //左下拐点
    uint8 right_upbreak   = 0;              //右上拐点
    uint8 right_dowmbreak = 0;              //右下拐点
    uint8 l_start = 0, l_end = 0;           //左边计算斜率起始点
    uint8 r_start = 0, r_end = 0;           //右边计算斜率起始点
    float slope_l_rate = 0, intercept_l = 0;//斜率和截距
    float slope_r_rate = 0, intercept_r = 0;//斜率和截距
    for (i = 1; i < total_num_l; i++)//先找左边下拐点
    {
        if (dir_l[i - 1] == 2 && dir_l[i] == 2 && dir_l[i + 3] == 4
            && dir_l[i + 5] == 4 && dir_l[i + 7] == 4)//2-->4跳变
        {
            left_dowmbreak = points_l[i][1];//传递y坐标
            printf("找到左边下拐点%d\n",left_dowmbreak);
            break;
        }
```

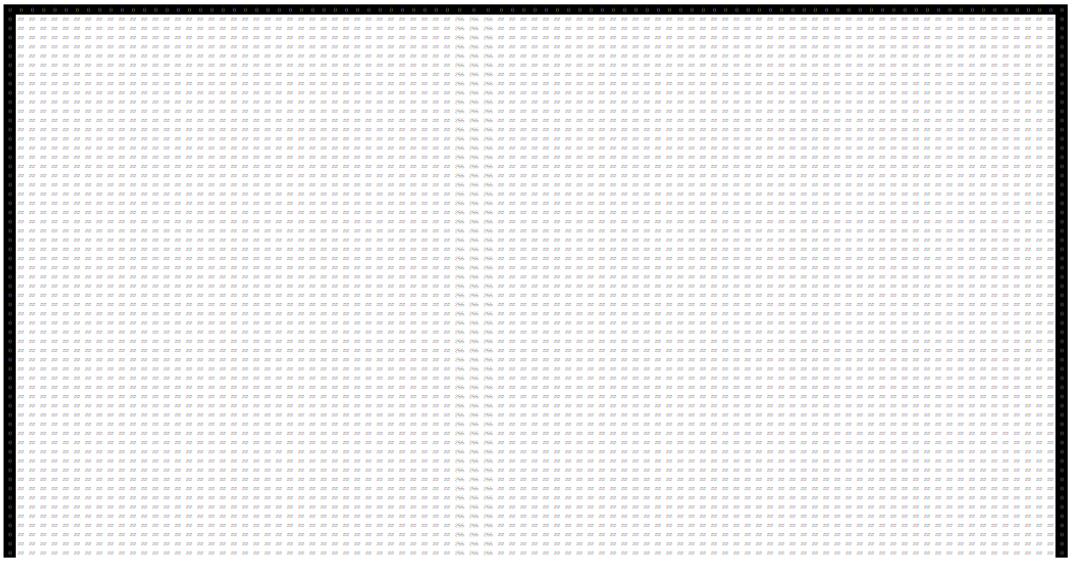
0 △ 0 行 1，列 1 制表符长度: 4 UTF-8 CRLF 纯文本

接下来讲一下程序里的一些杂七杂八的处理，避免产生疑惑，滤波函数我就不多说了，网上一大堆，我也是仿写的，不必较真，可以自行替换更好更快的算法。

首先 Get_image(mt9v03x_image);//请务必填原图首地址

然后 image_preprocessing(bin_image); //预处理

这个函数的作用就是给图像“罩”上一个框，目的是防止卡循环，不多解释，请勿删除。如下图所示。



Tips:本程序左右边线，轮廓线，包括生长方向，都是对称的。也有人问过为什么弄一个逆时针，一个顺时针，原因就在这里了，为了实现高度对称，只要掌握了其中一边的逻辑，或者写了其中一边的处理程序，另一边直接复制粘贴都是可行的。

元素处理放的位置也有讲究，放在边界提取之后，中线计算之前。

```
//获得起点
get_start_point(image_h-2);//先找起点  从底部倒数第二行开始找线

// 八邻域提取边界
search_l_r((uint16)USE_num,bin_image,&data_stastics_l, &data_stastics_r,start_point_l[0], start_point_l[1], s

// 从爬取的边界线内提取边线 ，  这个才是最终有用的边线
get_left(data_stastics_l );
get_right(data_stastics_r);

//元素处理写这个位置


//显示图像   改成你自己的就行
IPS130_displayimage032_zoom(bin_image[0], image_w, image_h, image_w, image_h);

    //根据最终循环次数画出边界点
    for (i = 0; i < data_stastics_l; i++)
    {
        IPS130_drawpoint(points_l[i][0]+2, points_l[i][1], BLUE);//显示起点
```

最后就是 hightest 了：在左右边界相遇时，会保存一个 y 值（如下图所示），这里我习惯性的把这个值称为最高值，这个值在遇到断道或大弯道的时候会变大，遇到长直道会接近于 1，具体怎么使用，我就不多说了，就看机智如你怎么使用了。

```
if (my_abs(points_r[r_data_statics][0] - points_l[l_data_statics - 1][0])<2
    && my_abs(points_r[r_data_statics][1] - points_l[l_data_statics - 1][1]<2)
    )
{
//printf("\nbreak:触发自然退出条件\n\n");
    *hightest = (points_r[r_data_statics][1] + points_l[l_data_statics - 1][1]) >> 1;//取出最高点
    break;
}
```

限制条件：

左右两边相遇，跳出循环；

```
if (my_abs(points_r[r_data_statics][0] - points_l[l_data_statics - 1][0])<2
    && my_abs(points_r[r_data_statics][1] - points_l[l_data_statics - 1][1]<2)
    )
{
//printf("\nbreak:触发自然退出条件\n\n");
    *hightest = (points_r[r_data_statics][1] + points_l[l_data_statics - 1][1]) >> 1;//取出最高点
    break;
}
```

左边比右边高了，左边等待右边；

```
if ((points_r[r_data_statics][1] < points_l[l_data_statics - 1][1]))
{
    continue;//如果左边比右边高了，左边等待右边
}
```

可以自己加一个条件，陷入死循环就退出（三次更新到同一个中心坐标点）；自己发挥

没有完美的程序，是程序就必然存在不足，很多地方有待发掘，本人现在上班，时间不多，加上身边没有车模，没有赛道，无法继续更新，请发挥你的想象力和创造力把它发扬光大吧。

辅助文档在此就结束了，相信细细品读就会有收获。

```
/**
 * @brief 十字补线函数
 * @param uint8(*image)[image_w]    输入二值图像
* @param uint8 *l_border            输入左边界首地址
* @param uint8 *r_border            输入右边界首地址
* @param uint16 total_num_l         输入左边循环总次数
* @param uint16 total_num_r         输入右边循环总次数
* @param uint16 *dir_l              输入左边生长方向首地址
* @param uint16 *dir_r              输入右边生长方向首地址
* @param uint16(*points_l)[2]       输入左边轮廓首地址
* @param uint16(*points_r)[2]       输入右边轮廓首地址
 *  @see CTest      cross_fill(image,l_border, r_border, data_statics_l, data_statics_r, dir_l, dir_r, points_l, points_r);
 * @return 返回说明
 *     -<em>false</em> fail
 *     -<em>true</em> succeed
 */
void cross_fill(uint8(*image)[image_w], uint8 *l_border, uint8 *r_border, uint16 total_num_l, uint16 total_num_r,
                uint16 *dir_l, uint16 *dir_r, uint16(*points_l)[2], uint16(*points_r)[2])
{
    uint8 i;
    uint8 break_num_l = 0;
    uint8 break_num_r = 0;
    uint8 start, end;
    float slope_l_rate = 0, intercept_l = 0;
    //出十字
    for (i = 1; i < total_num_l; i++)
    {
        if (dir_l[i - 1] == 4 && dir_l[i] == 4 && dir_l[i + 3] == 6 && dir_l[i + 5] == 6 && dir_l[i + 7] == 6)
        {
            break_num_l = points_l[i][1];//传递y坐标
            printf("brea_knum-L:%d\n", break_num_l);
            printf("I:%d\n", i);
            printf("十字标志位：1\n");
            break;
        }
    }
    for (i = 1; i < total_num_r; i++)
    {
        if (dir_r[i - 1] == 4 && dir_r[i] == 4 && dir_r[i + 3] == 6 && dir_r[i + 5] == 6 && dir_r[i + 7] == 6)
        {
            break_num_r = points_r[i][1];//传递y坐标
            printf("brea_knum-R:%d\n", break_num_r);
            printf("I:%d\n", i);
            printf("十字标志位：1\n");
            break;
        }
    }
    if (break_num_l&&break_num_r&&image[image_h - 1][4] && image[image_h - 1][image_w - 4])//两边生长方向都符合条件
    {
        //计算斜率
        start = break_num_l - 15;
        start = limit_a_b(start, 0, image_h);
        end = break_num_l - 5;
        calculate_s_i(start, end, l_border, &slope_l_rate, &intercept_l);
        //printf("slope_l_rate:%d\nintercept_l:%d\n", slope_l_rate, intercept_l);
        for (i = break_num_l - 5; i < image_h - 1; i++)
        {
            l_border[i] = slope_l_rate * (i)+intercept_l;
            l_border[i] = limit_a_b(l_border[i], border_min, border_max);
        }

        //计算斜率
        start = break_num_r - 15;
        start = limit_a_b(start, 0, image_h);
        end = break_num_r - 5;
        calculate_s_i(start, end, r_border, &slope_l_rate, &intercept_l);
        //printf("slope_l_rate:%d\nintercept_l:%d\n", slope_l_rate, intercept_l);
        for (i = break_num_r - 5; i < image_h - 1; i++)
        {
            r_border[i] = slope_l_rate * (i)+intercept_l;
            r_border[i] = limit_a_b(r_border[i], border_min, border_max);
        }


    }

}
```

```
/**
* @brief 最小二乘法
* @param uint8 begin                输入起点
* @param uint8 end                  输入终点
* @param uint8 *border              输入需要计算斜率的边界首地址
 *  @see CTest      Slope_Calculate(start, end, border);//斜率
 * @return 返回说明
 *     -<em>false</em> fail
 *     -<em>true</em> succeed
 */
float Slope_Calculate(uint8 begin, uint8 end, uint8 *border)
{
    float xsum = 0, ysum = 0, xysum = 0, x2sum = 0;
    int16 i = 0;
    float result = 0;
    static float resultlast;

    for (i = begin; i < end; i++)
    {
        xsum += i;
        ysum += border[i];
        xysum += i * (border[i]);
        x2sum += i * i;

    }
    if ((end - begin)*x2sum - xsum * xsum) //判断除数是否为零
    {
        result = ((end - begin)*xysum - xsum * ysum) / ((end - begin)*x2sum - xsum * xsum);
        resultlast = result;
    }
    else
    {
        result = resultlast;
    }
    return result;
}

/**
* @brief 计算斜率截距
* @param uint8 start                输入起点
* @param uint8 end                  输入终点
* @param uint8 *border              输入需要计算斜率的边界
* @param float *slope_rate              输入斜率地址
* @param float *intercept               输入截距地址
 *  @see CTest      calculate_s_i(start, end, r_border, &slope_l_rate, &intercept_l);
 * @return 返回说明
 *     -<em>false</em> fail
 *     -<em>true</em> succeed
 */
void calculate_s_i(uint8 start, uint8 end, uint8 *border, float *slope_rate, float *intercept)
{
    uint16 i, num = 0;
    uint16 xsum = 0, ysum = 0;
    float y_average, x_average;

    num = 0;
    xsum = 0;
    ysum = 0;
    y_average = 0;
    x_average = 0;
    for (i = start; i < end; i++)
    {
        xsum += i;
        ysum += border[i];
        num++;
    }

    //计算各个平均数
    if (num)
    {
        x_average = (float)(xsum / num);
        y_average = (float)(ysum / num);

    }

    /*计算斜率*/
    *slope_rate = Slope_Calculate(start, end, border);//斜率
    *intercept = y_average - (*slope_rate)*x_average;//截距
}
```