Verson ： 1.0

图像处理浅析文档

Author：村东头无敌的小瞎子

闲鱼店铺：玛卡巴卡的杂货铺

我想先打个广告，如果不需要可以忽略哈（@_@）：

文档免费分享，真的建议耐心看完。文档内容包括环岛状态机及相关新增内容，新版链表结构的边界提取的简单介绍，以及一些可行性验证，也有一些自己的图像调试常用的方法的分享。鄙人非电子信息专业，技术不好，表达可能不够准确，但都是用了心写的，希望能对你有所帮助，仅供学习使用，**如果照搬请标明出处！**

但是下面这些内容都不是无偿的，（明码标价，介意勿扰）本人平时要上班，仅周一至周五晚上 7 点以后，周末有时间。

1. 移植旧版源码 **30**
2. 环岛状态机源码(仅供学习使用，不能直接用于比赛) **200**
3. 新手入门指导（仅图像处理） **200**
4. 链表结构的八邻域程序 **300** （不是最优解但不贱卖，仅供学习使用，目前 STM2F103ZET6 试过可用）
5. 周末答疑:仅限本人自己编写的文档 **100**
6. 不需要指导，只是需要帮助解决 BUG **50(未解决可退)**

所有交易均在闲鱼平台完成，你有保障，我也有保障！

下图是本人唯一闲鱼店铺：

注：1.如果未及时回复就是在上班/加班，需要帮忙直接说需求就行了，价格都是可以谈的。

2.多看文档，一般大家问的问题，大部分都是在文档内回答过了的，问的次数多了，我不一定次次都耐心答复，避免已读未回，确保问的问题我没有在文档里提到过。

3.很多人自己尝试移植后，遇到各种问题，就好比总是有人直接问我“为什么卡死了啊？”，我的回答永远都是“不知道”，因为我不知道你漏了哪一步，我移植就没失败过。所以，看文档，要不就 printf()函数打印调试，我都是留了打印函数的。

4.在别处购买到我的源码，请勿到鄙人店铺提问，谢谢配合。

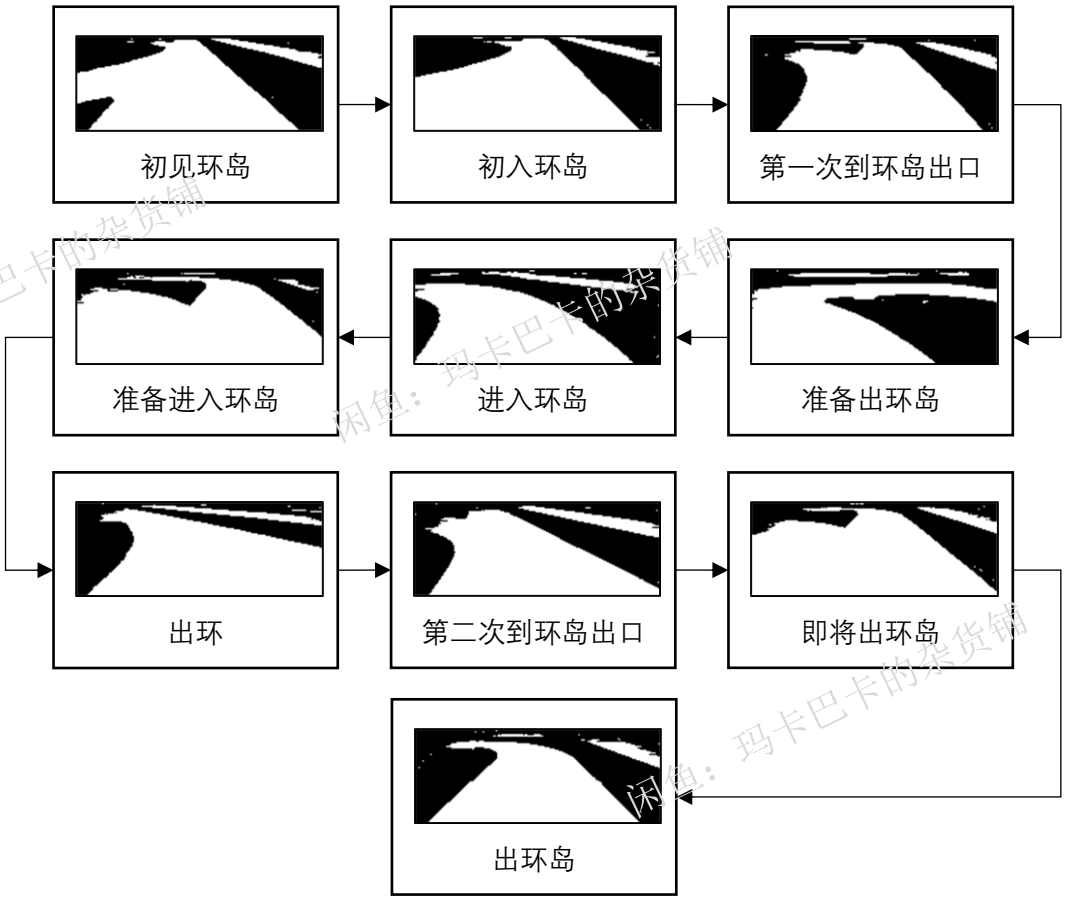
第一节：环岛状态机浅析

左右圆环，目前看来应该是比较棘手的元素，但是解决方法多种多样，例如 电磁辅助入环出环、编码集计步入环出环、陀螺仪偏航角积分入环出环、摄像头补线入环出环等。我这里主要分享的是摄像头补线做环岛状态机的内容，此方式形式虽然单一，但实现复杂度没那么高。

首先强调，咱们要做的是图像处理，不是研究破烂地图巡线算法，我们应该保证采集到的图像质量应当尽量的好，不然就真的是“图像质量差，调试两行泪”，能硬件处理好图像质量问题，就不用软件，能节约一点是一点，省下来的资源还可以干很多事情。

废话少说，咱们步入正题：

正常进出环岛大致为以下几个流程：



进出环岛流程

假如每一幅图都对应一个状态，那么找到当前状态，没到下一幅图之前，状态都不会改变，并且中间的顺序不能改变。

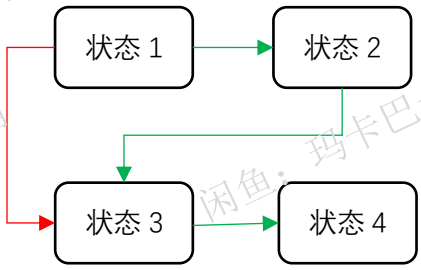
例如我刚开始到“初见环岛”，初始状态设为“1”，在没找到“初入环岛”前，我的状态都不会改变，当且仅当“初入环岛”时，状态才会变成 2，如果跳过这幅图，直接找到下一幅图“第一次到环岛出口”，状态依旧不会改变，顺序必须是“初见环岛”-》“初入环岛”-》“第一次到环岛出口”如图 1.1 所示。

基于上述思想的环岛状态机，我们希望在第一步就能准确、稳定的判断出：到底是不是环岛？是左环岛？还是右环岛？一旦确定了是环岛这个状态，那么这个状态一直都是锁死，

再根据锁死后状态进行后续的处理，直到最后成功出了环岛才会清楚回归正常巡线状态。

也就是说，如果一开始就判断到了这个元素是个环岛入口，并且是左环岛的入口，那么后续的所有处理（如下图 1.1 中伪代码所示）都基于这是左环岛来进行处理（右边同）。

```
if(这是个左环岛)/*左环岛状态机*/
{
    if(环岛状态 1 () )
    {
        补线 1 ();
    }
    ...
    if(环岛状态 n () )
    {
        补线 n ();
    }
}
```

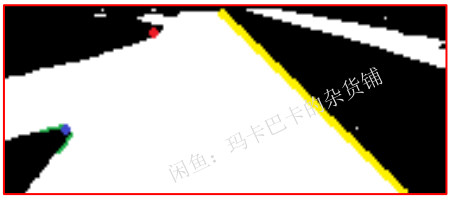


状态切换，必须是状态 1>状态 2>状态 3>状态 4,绿色为正确状态切换流程。
红色为错误切换流程。

1.1 状态切换

所以,(在图像质量得到保证的情况下)第一步判断环岛应当倾尽可能提高判断准确率。并且需要寻找有效的方法来避免误判断，环岛补线之道就在其中！

下图 1.2 是初见环岛时采集到的一帧图像：

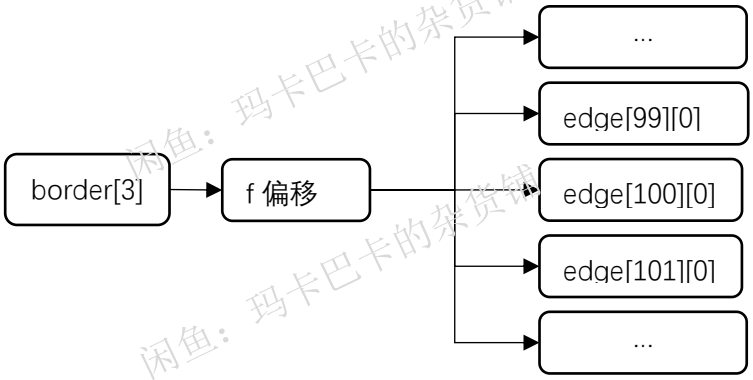


那么让我们一起来分析一下初见左环岛时的明显特征：右边界(黄)连续，左边界有断裂(未画出)，左边存在一个 A 点(蓝点)，以及一个 P 点(红点)三个有效特征，能有效区别于其它元素。

1.2 初见环岛

对于边界连续特征判断，只要满足从下往上 border 值单调递减，且相邻两个 border 值之间的差值在一定范围内，就认为它是连续的。

对于 A 点的查找，稍微复杂一点，则需要借助八邻域提取到的边缘（后面都以 edge 代称）来辅助判断，因为常规的左右巡线得到的边界以及从 edge 中提取出来的 border 都很难准确唯一的定位到该点的位置。我们可以直接访问 edge，通过 A 点与周围点的位置关系（A 点左右两边点的 y 都小于 A 点的 y,A 点左边点的 x 小于 A，右边点的 x 值大于 A）直接定位 A 点是否存在，但是有一个小问题，那就是如果每次都去完整的遍历一整条 edge,循环次数会太多，我们希望一次循环下来，即使没找到 A 点，也不要循环太多次数，最好不要超过图像的整体高度值，所以我们就需要利用到 border 搭配 edge 一起使用，来更快的定位 A 点，同时把循环次数降下来。简单来说就是粗定位和精定位两步（暂且这么表述吧），这里我们需要提前建立 edge 和 border 之间的联系（指能通过一个 border 对应的 index 顺利访问 border 在 edge 中的位置，如图 1.3 所示），粗定位时，我们利用 border 快速的找到边界断裂的位置，这个时候我们顺利得到断裂位置的 y 值,精定位时，根据查找到的断裂处的 y 值在 edge 内循环大约 30-50 个点（不完整遍历）继而查找 A 点，查找流程如图 1.4 所示。



我们现在有 border[3] = 10; edge[100][0] = 10; edge[100][1] = 3;
显然，在之前的程序中 border[3]就对应 edge[100][0],我们有一个映射关系 f,通过 f 的偏移,我们可以根据 border 的 3，顺利访问到 edge 的 100，并且不止 100，甚至 99、101、102…

图 1.3 建立联系

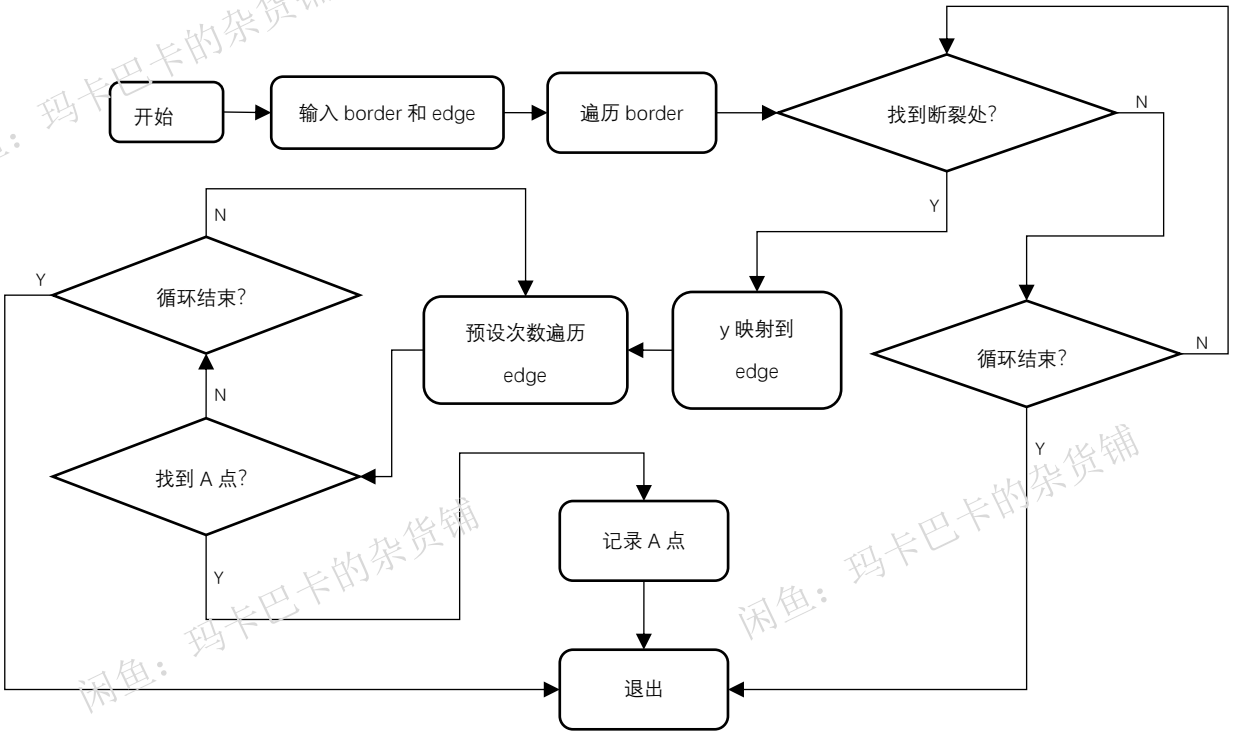


图 1.4 查找 A 点流程

对于 p 点查找判断，只要满足当前 border 的值同时大于上边和下边 border 的值,并且三处的 border 的值都不能是边界值（也就是我们所说的边界有效，即未丢线），就认为它是 p 点。

有了这些判断思路，在图像质量过关的情况下，已经可以有效判断“初见环岛”了。当然，可以选择把这些判断条件封装成子函数，多次调用，也可以直接一个函数 function（）

解决，下图 1.5 是我判断左环岛大致流程（右边对称）。

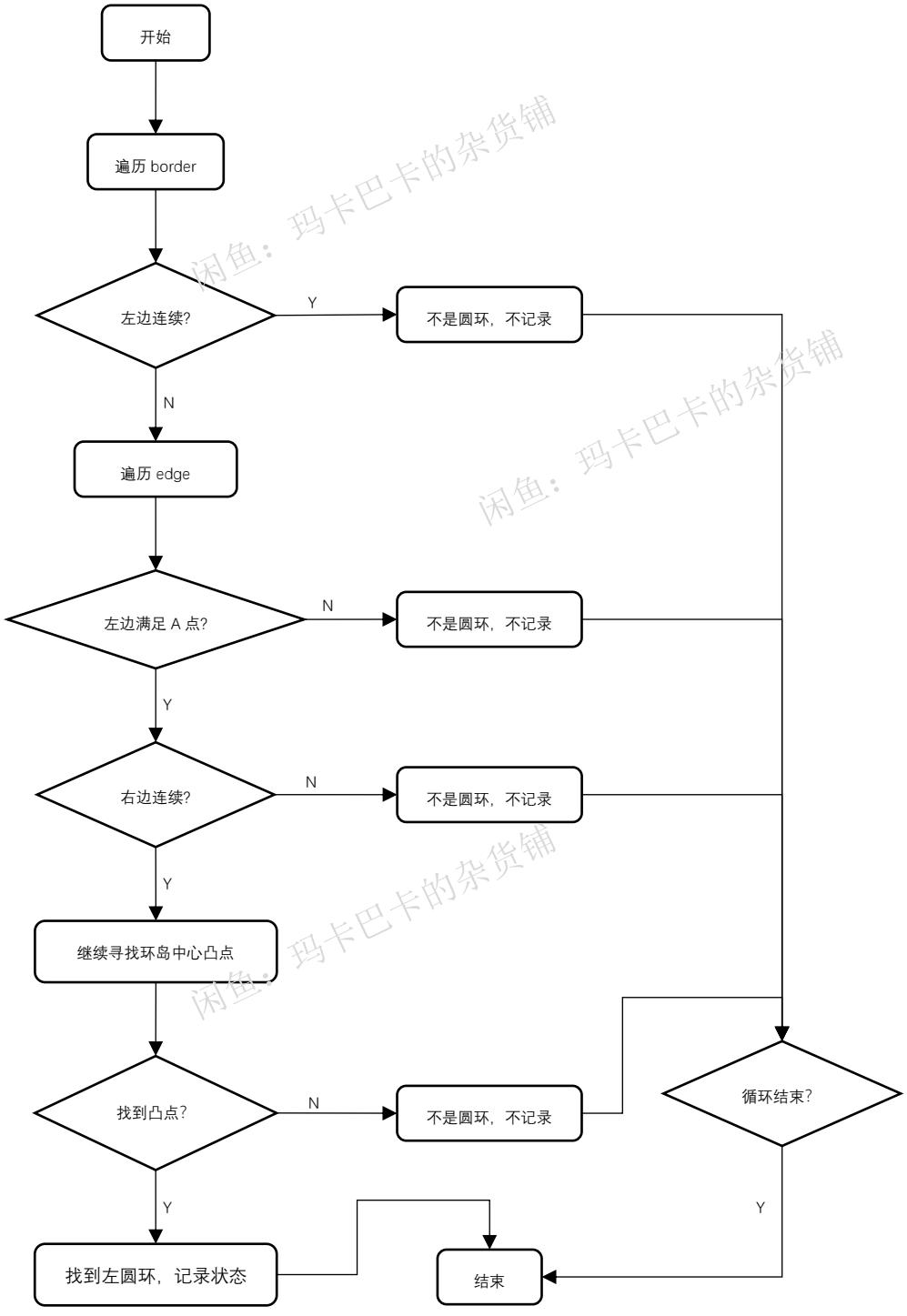


图 1.5 环岛判断流程

最后根据提取到的 A 点以及 P 点的坐标，计算斜率（k）和截距（b）进行补线就可以顺利的通过，这里不再推荐最小二乘法，因为太麻烦，所以直接用两点计算一样可以得到正确

的结果（后面都是用这个方法），简单便捷，可以说效果拔群，如图 1.6 所示。
计算公式也很简单：

$k = \triangle x / \triangle y;$

$b = y - kx;$

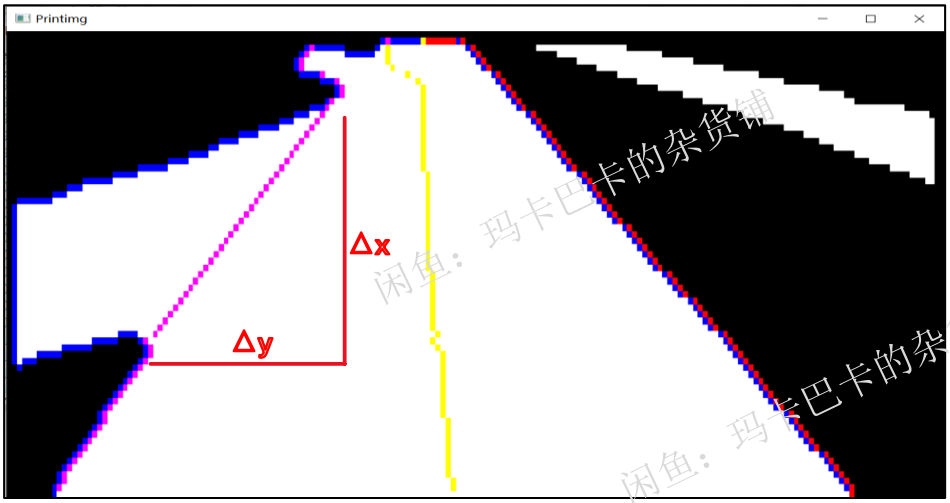


图 1.6　补线示意图

准确判断到环岛类型后，就可以针对性的做接下来的运行状态了，（因为后面的状态都是基于第一步确定的环岛类型得来的，所以第一步很重要），在小车识别到“初见环岛”，此时状态为 1，并且在补线状态下继续平稳行驶时，应当继续寻找下一个状态的特征，也就是“初入环岛”，在此时，左下角 A 点丢失，剩下的是一片空白，左上部分 p 点依旧存在，右边仍然是单调连续的，故通过左下角固定点（0，ymax）与 p 点计算斜率截距补线继续行驶，并更替环岛状态为 2，如图 1.7 所示。

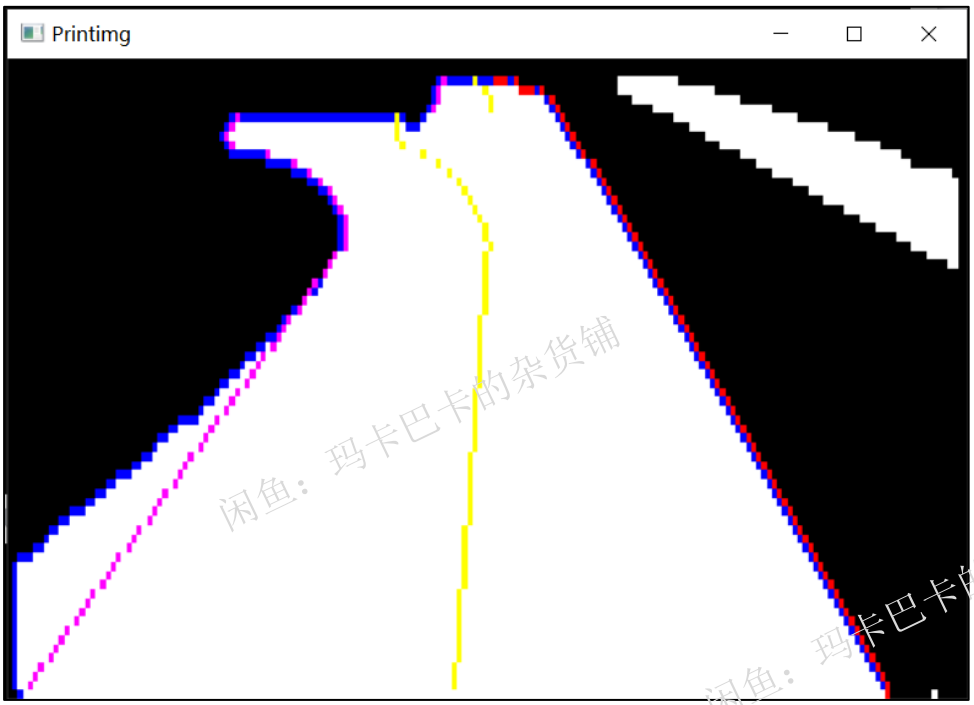


图 1.7 初入环岛补线示意图

保持此时的状态为 2 的同时，寻找下一个状态的特征，此时，我们小车运行到“第一次到环岛出口”位置，特征非常明显，右边依然连续，左边不连续，且存在一个 V 点如图 1.8

红圈所示，和 A 点查找思路一致，先粗定位，再精定位，只不过此时的 V 点的 y 值是比两边的点的 y 值都要大。找到 V 点后，切换状态为 3，使用 V 点坐标和右边界最下面的点一起计算斜率和截距，对右边界进行拉线处理即可，如图 1.8 所示。

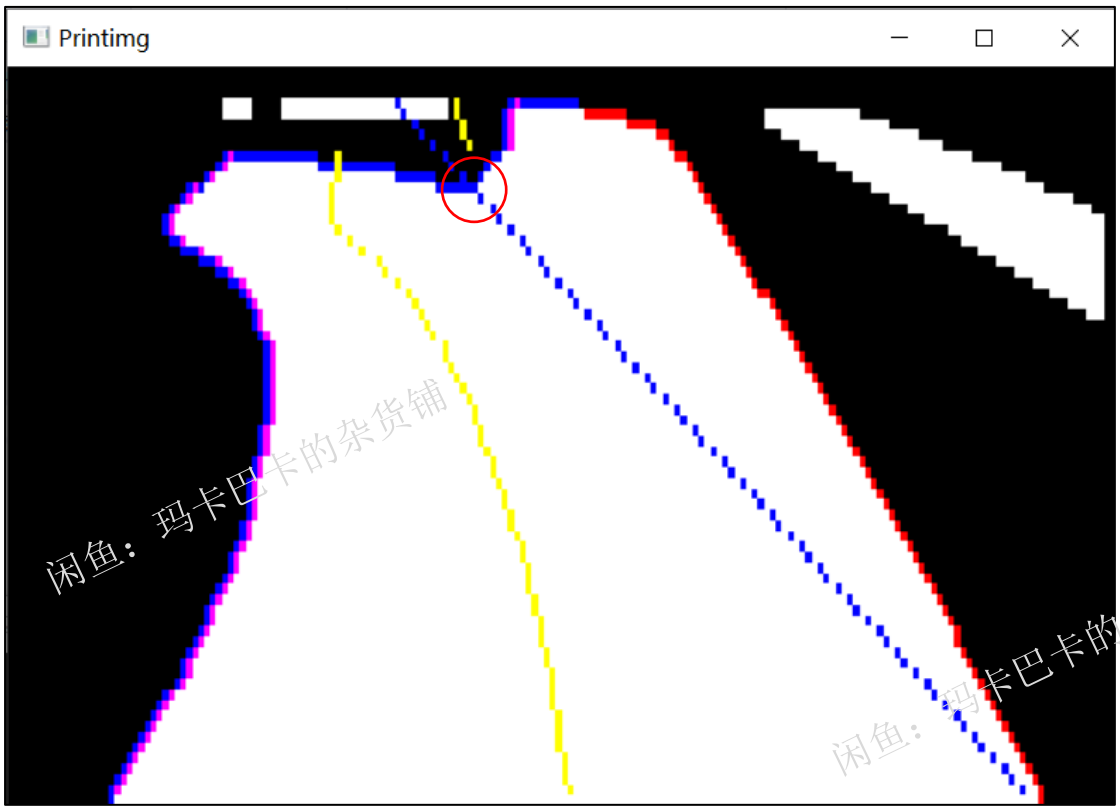


图 1.8 第一次到环岛出口补线示意图

保持状态为 3，继续寻找下一位置特征，在左边 V 点消失前，右边界补线状态会一直维持，直到左边界 V 点消失为止，就已经快接近入环了。

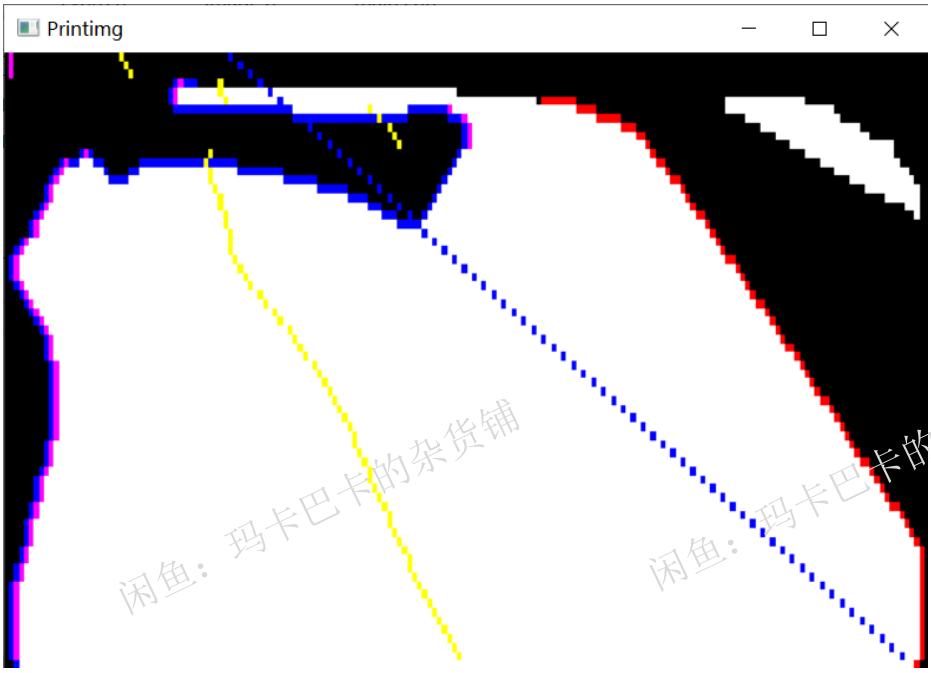


图 1.9 即将入环补线示意图

实际情况下，在即将入环前，V 点可能会跑到右边界去，如图 1.10 所示，只需按照同样思路查找右边界 V 点，拉线到右下角就可以了，在这里也可以更新一下状态为 4，如果此处对小车姿态影响不大，那么可以不用在这里更新状态（我这里为更新了状态）补线，直接寻找下一个状态就行了。

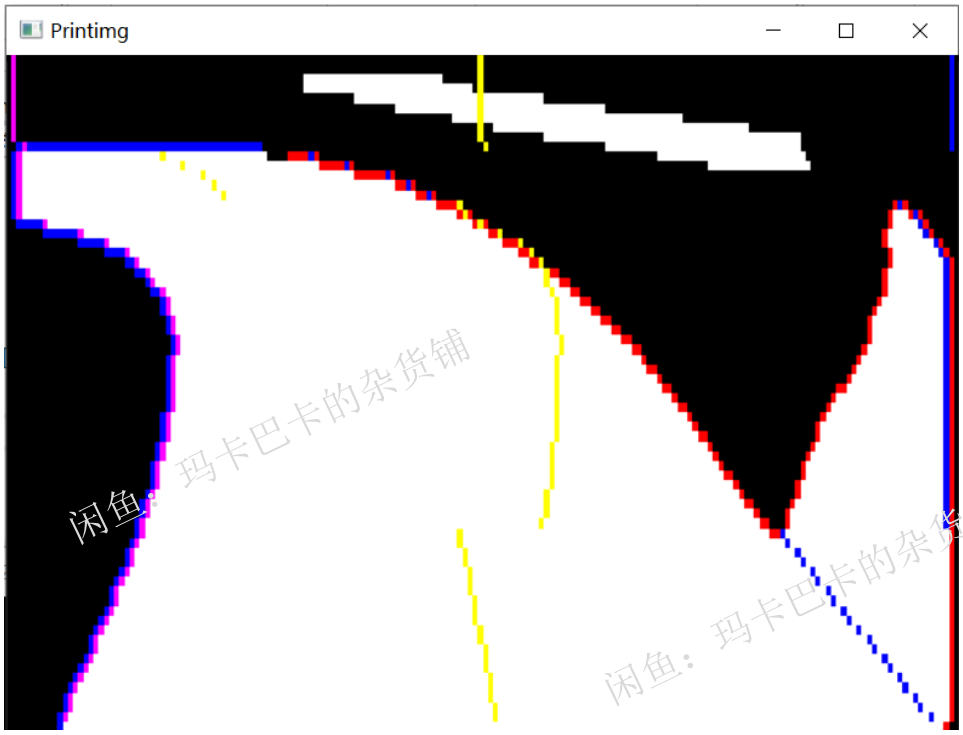


图 1.10 右边 V 点

保持当前状态为 4，继续寻找下一个状态，在小车完全进入环岛后，赛道的左右两边均有黑色部分，此时巡到的边界均为有效边界，或者说均不是图像边界，那么满足条件后直接切换状态为 5 就可以了。如图 1.11 所示。

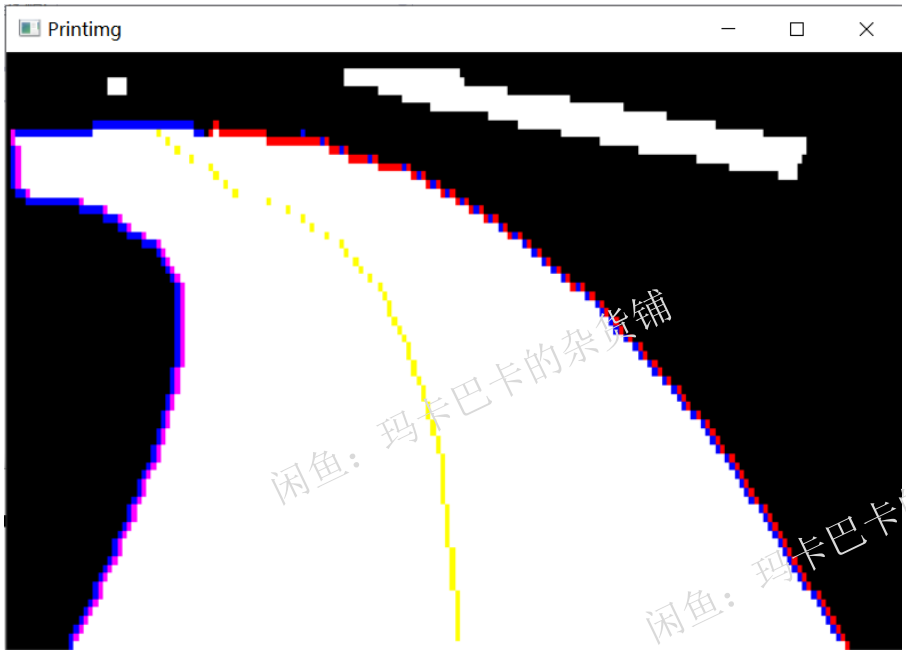


图 1.11 入环后状态

成功进环，那么接下来就是出环了，即将出环时，有边界会出现断裂，这个时候小车的姿态总是会向着左边运行就是正确的，所以这里我推荐直接对右边界进行判断，只要出现断裂，就从断裂处到图像左上角（0,0）直接拉线补线就可以了（如图 1.12），并且这个时候可以不改变状态值，直到走出弯道（如图 1.13）这一块再切换状态。

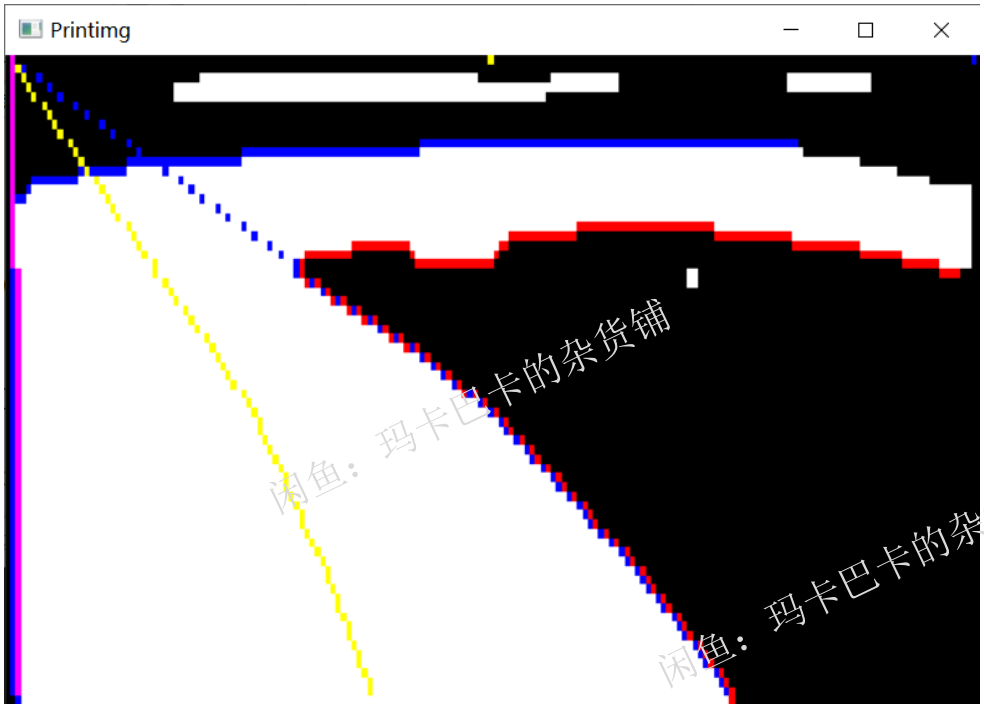


图 1.12

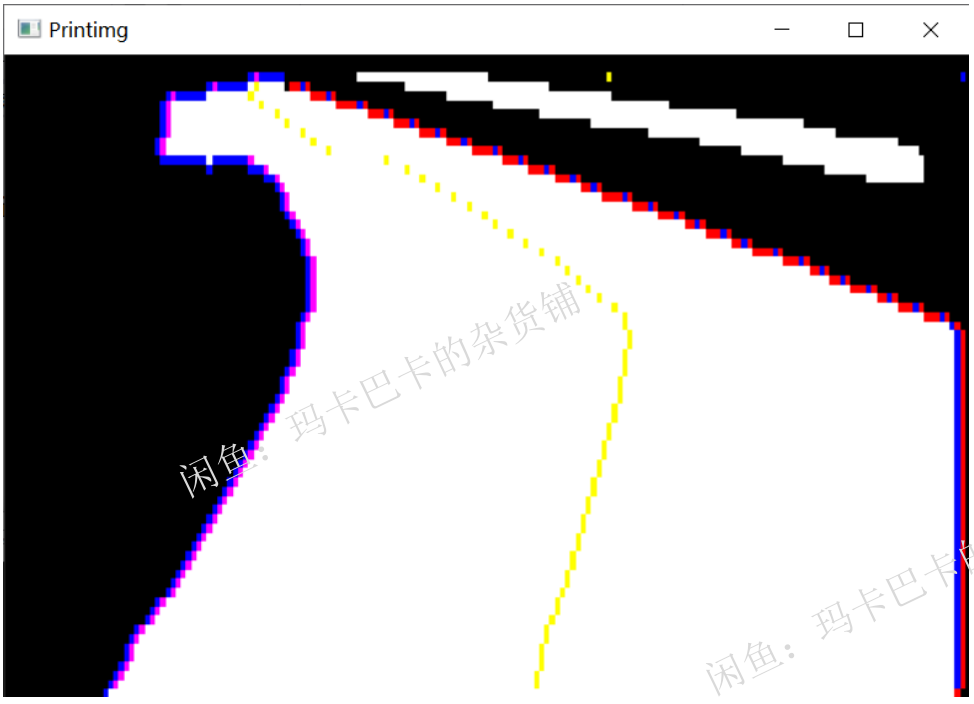


图 1.13

当小车运行到弯道结束后，左下角全白，右下角有黑色块，则认为已经走出了弯道了，此时可以切换状态为 6，继续寻找下一特征即可，其实到这里基本就是收尾了，因为这里正常巡线已经可以最稳定第二次走到环岛出口了，我们需要做的就是直接寻找左边的断裂处

（也可以寻找 V 点，我推荐使用断裂点，更快），拉线到左下角（0,ymax）切换状态为 7，就可以达到出环的目的了，如图 1.14 所示。

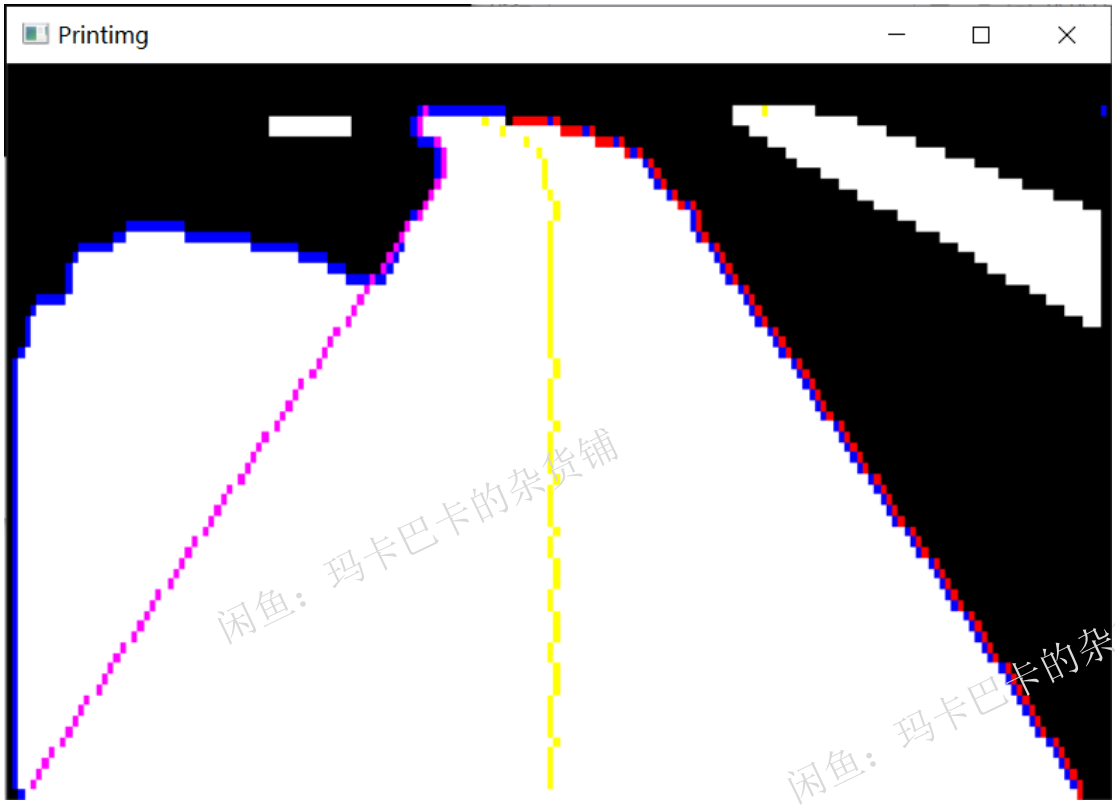


图 1.14

到这里已经就算大功告成了，按照这个补线状态（7）下去，就能正确出环了，剩下的就是清楚标志位了，和成功入环一样，成功出环后，左右边界都是正常的，没有丢线（或者丢线很少），并且左下角和右下角都是大黑块（如图 1.15），就认为已经出环成功了，清除状态为 0 即可进行下一次圆环判断了。

tips:在判断到环岛之后，什么十字，斑马线，坡道，三岔等元素的判断就都可以关掉了，避免元素之间相互影响，最好这些元素的判断分个优先级。

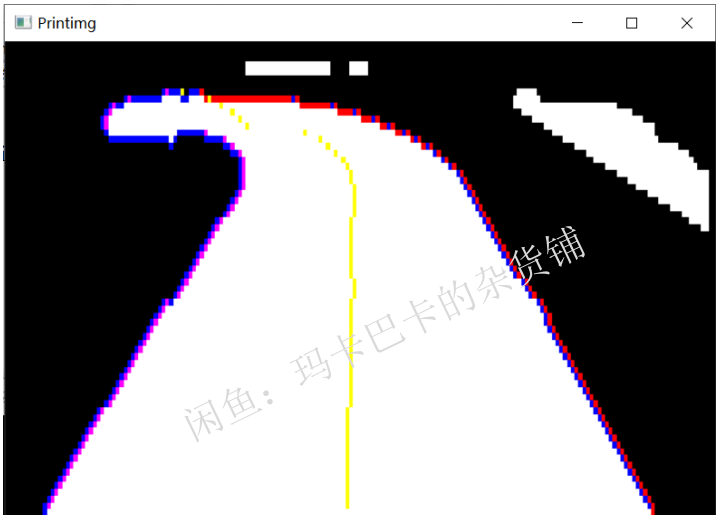
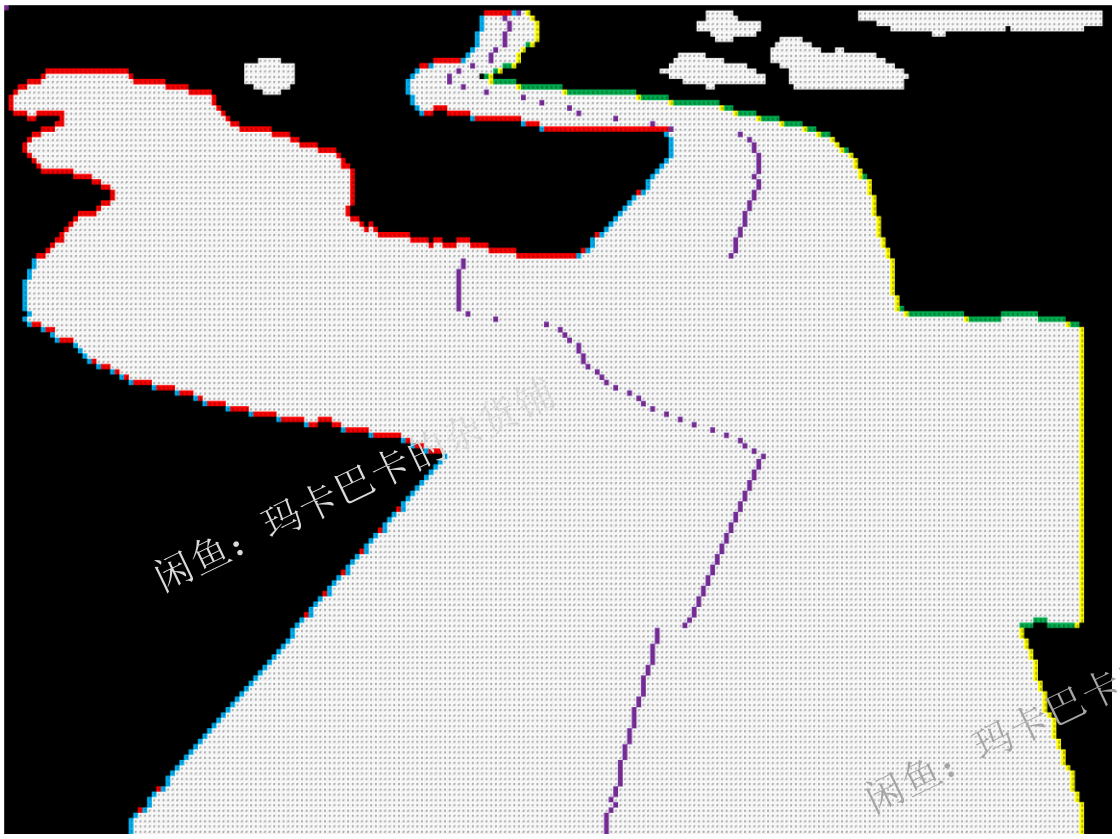


图 1.15

第二节：链表结构八邻域介绍

接下来分享一下基于链表结构的八邻域边界提取，最初有这个想法是还在比赛的时候，当时刚接触到图像处理，对什么都是一窍不通，也是吹牛逼的时候听群友们说八邻域拥有得天独厚的优势，于是自己也想弄一个，但是受限于时间以及自己悟性不好，最终没能如愿。现在工作了，回顾一下，还是想把这个空缺补上，毕竟这个才是最初自己想实现的八邻域中的“白月光”，虽然目前来看并不是最优解，甚至对图像的质量要求更高，但是还是想做出这个效果看一看，于是就有了这一版动态的八邻域边界提取。

先上一张寻边界效果（Excel 生成）：



首先我们需要了解一部分知识（图片来源于网络，侵删），不然会有点跳跃。

1. 堆和栈：

栈：在函数调用时，第一个进栈的是主函数中后的下一条指令（函数调用语句的下一条可执行语句）的地址，然后是函数的各个参数，在大多数的C编译器中，参数是由右往左入栈的，然后是函数中的局部变量。注意静态变量是不入栈的。
当本次函数调用结束后，局部变量先出栈，然后是参数，最后栈顶指针指向最开始存的地址，也就是主函数中的下一条指令，程序由该点继续运行。
堆：一般是在堆的头部用一个字节存放堆的大小。堆中的具体内容有程序员安排。

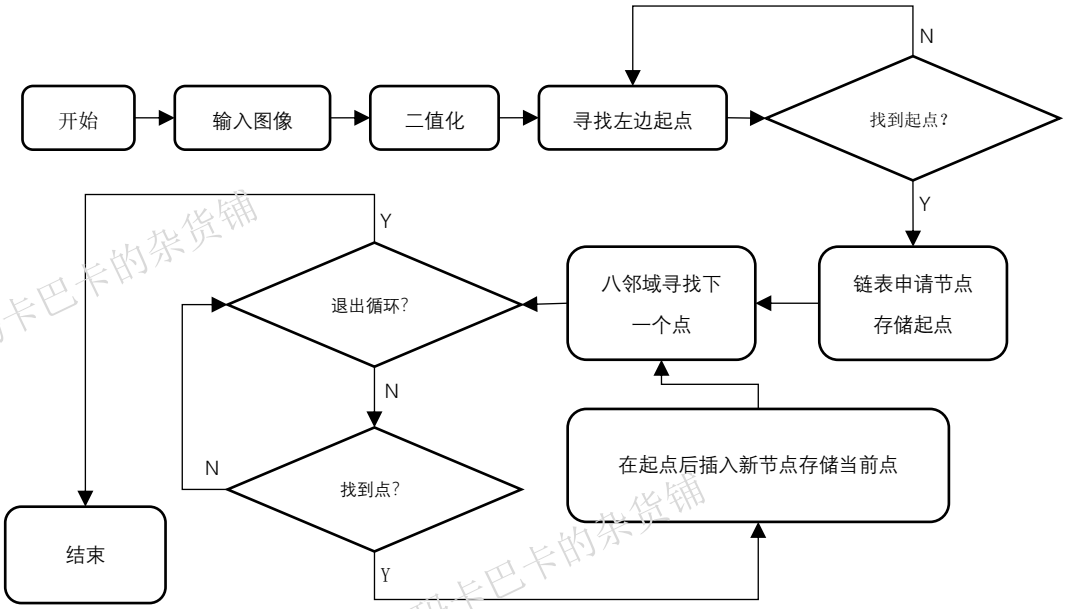
2. 申请效率比较：

栈由系统自动分配，速度较快。但程序员是无法控制的。
堆是由new分配的内存，一般速度比较慢，而且容易产生内存碎片,不过用起来最方便.
另外，在WINDOWS下，最好的方式是用VirtualAlloc分配内存，他不是在堆，也不是在栈是直接在进程的地址空间中保留一快内存，虽然用起来最不方便。但是速度快，也最灵活。

所以，在申请效率上，利用堆来建立顺序表，本身在速度上就已经落后于栈区自动分配的内存了，不过目前情况而言，依旧是寄希望于，八邻域本身的计算速度能够弥补上这部分

差距。（ps:链表这部分知识需要大家自行学习了，我的链表函数都是网上抄的）

流程图如下图所示，这里我以左边为例。



大致流程如此，除了结构更改其他的和旧版八邻域基本没有区别，该二值化的地方二值化，该滤波的滤波，该加黑框的地方加黑框。

链表结构分享到此为止。

第三节：常用方法分享

我们有一些想法需要验证的时候，当然只是软件层面，我们完全可以不用直接上单片机直接验证，单片机能办到的事情有限，对于可视化和调试方便程度都很难让人满意，所以，直接在电脑上把思路给跑通，再移植到单片机上运行，反正都是 C 语言编写的，大部分情况都不存在说什么电脑上能用，单片机跑不了的情况。

首先，就是 openCV 配置 C++，再用 C 语言编写进行图像处理，这个方法我已经分享过了，个人认为是最方便的了。

智能车灰度图像处理可视化分享，压缩包在简介自取哦。_哔哩哔哩_bilibili

然后就是 easyX 图形库了，甚至都不用你配置什么环境变量，直接把库下载下来放在文件夹，然后就可以调用了。

以上都是借用外部工具办到的，最后介绍一种不使用外部工具的方法。一样可以帮助调试图像。

这种方法就是使用 C 语言自带的文件系统，将像素点的数据导出到.csv 格式的文件里，再用 Excel 打开就可以了。黑色为 0，白色给 255，左边界 7，右边界 8，中线 9，最后在 Excel 里面用条件格式填充不一样的颜色就行了