浙 江 大 學

智 能 车 培 训 资 料

连 通 域 测 算

指导教师 姚维、韩涛

编写作者 邢毅诚

联系方式 3190105197@zju.edu.cn

完成日期 2023 年 3 月 10 日

目录

一、连通域介绍

连通区域分析是一种在 CVPR 和图像分析处理等众多应用领域较为常见和基本的一种方法，如：OCR 识别中的字符分割提取（车牌识别，文本识别，字幕识别等等）。他通过对二值图像中目标像素的标记，让每个单独的连通区域形成一个被标识的块，进一步的我们就可以获取这些块的轮廓、外接矩形、质心、不变矩等集合参数。

在智能车竞赛中，由于赛道往往跟其他元素分割开来，即我们在获得赛道图片并对其进行二值化后，赛道往往是白的，而赛道旁边的元素往往是黑的。在最理想情况下，我们获取的整张图片应呈现赛道为白色，而除去赛道外的其他所有元素都为黑的这样一个分布。但在实际情况中，由于环境，光照，地面反光，以及其他物品的影响，部分赛道可能被识别为黑色，而有些非赛道部分可能被识别为白色。这些情况的出现，会给智能车接下来的判断造成极大的困扰，因此，从获得图像的一瞬间开始，我们就需要进行一系列的处理，如大津法二值化，滤波，连通域/边线搜索，图像修正与补全等一系列的措施。在这其中，比较重要的一环节就是连通域/边线搜索，本文将着重对连通域进行介绍。当然，对于本文提到的一些其他预处理方法，我们也会简单提及，这些方法比较简单，也有现成的代码，因此我们不会花费过多的笔墨在这方面。

在智能车比赛中，由于种种原因，如反光，看到了对面的赛道，看到了赛道的挡板，看到了别的东西，摄像头的抖动等等，如下图所示：

图 1.1: 摄像头图片

这是我们摄像头拍摄到的一张图片，可以看到在最上方有着很多不相干的元素，但这些不相干的元素又会在很大程度上影响我们的判断，因此，我们需要采用寻找赛道边线，

或者寻找赛道主体的方式来去除这些干扰。我们可以注意到，由于智能车比赛的特殊性，我们拍摄到的赛道图片必然是相连的，而由于赛道旁边的黑边和背景的存在，与赛道不相干的元素大概率与赛道是不相连的，因此，我们只需要搜索一张图片中，有哪些点与赛道主体是相连的，并把与赛道相连的部分置为白色，而其余的部分一律置为黑色，这样便可以在最大程度上去除掉与赛道不相干的元素，具体如下图所示：

图 1.2: 连通域测算

可以看到，与赛道相连的部分，都被置为了白色，而与其不相连的部分则均被置为了黑色（图像中赛道中央的黑线为我后期寻找的中线，请忽略），而这种寻找某个区域是否相连的方式便是连通域测算。

二、连通域测算方法

1 连通域算法原理

在我们讨论连通域标记的算法之前，我们先要明确什么是连通域，怎样的像素邻接关系构成连通。在图像中，最小的单位是像素，每个像素周围有 8 个邻接像素，常见的邻接关系有 2 种：4 邻接与 8 邻接。4 邻接一共 4 个点，即上下左右，如下左图所示。8 邻接的点一共有 8 个，包括了对角线位置的点，如下图所示。

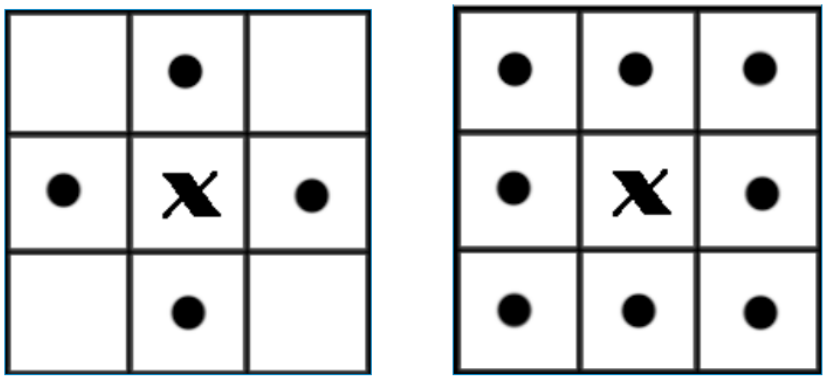

图 2.1: 4 邻接与 8 邻接

如果像素点 A 与 B 邻接，我们称 A 与 B 连通，于是我们不加证明的有如下的结论：

如果 A 与 B 连通，B 与 C 连通，则 A 与 C 连通。

在视觉上看来，彼此连通的点形成了一个区域，而不连通的点形成了不同的区域。这样的一个所有的点彼此连通点构成的集合，我们称为一个连通区域。

下面这幅图中，如果考虑 4 邻接，则有 3 个连通区域；如果考虑 8 邻接，则有 2 个连通区域。（注：图像是被放大的效果，图像正方形实际只有 4 个像素）。

图 2.2: 4 邻接与 8 邻接

在智能车中，由于图像的像素较低，使用 8 邻接时容易出现误判的情况，因此，我们采取 4 邻接的方式来进行连通域的识别。

在确定基本的连接方法之后，我们需要确定连通域分析所使用的算法，最简单的连通域搜索方式当然是确定一个点后进行深度优先搜索，但深度优先搜索需要大量的递归，同时调用大量的堆栈，其时间复杂度也较高，出于对单片机性能的考量，我们必须使用其他的连通域测算算法。通常来讲，实现连通域有以下两种方法：

- Two-Pass(两遍扫描法): 通过扫描两边图像，就可以将图像中存在的所有连通区域找

出并标记。

- Seed Filling(种子填充法): 种子填充方法来源于计算机图形学，常用于对某个图形进行填充。其思路是选取一个前景像素点作为种子，然后根据连通区域的两个基本条件（像素值相同，位置相邻）将与种子相邻的前景像素合并到同一个像素集合中，最后得到的该像素集合则为一个连通区域。

这两种方法各有优劣，接下来将分别对这两种方法进行介绍（更详细的讲解请参照：https://blog.csdn.net/tiandijun/article/details/51279643）：

1.1 Two-Pass 算法

Two-Pass 的思路如下所示：

第一遍扫描时赋予每个像素位置一个 label，扫描过程中同一个连通区域内的像素集合中可能会被赋予一个或多个不同的 label, 因此需要将这些属于同一个连通区域但具有不同值的 label 合并，也就是记录它们之间的相等关系。

第二遍扫描就是将具有相等关系的 label 所标记的像素归为一个连通区域并赋予一个相同的 label（通常这个 label 是多个相同的 labels 中的最小值），在智能车中，我们需要做的便是找到属于赛道的第一个点，再将图像中与其等价的像素点置 1，与其不等价的像素点置 0 即可。

相应的伪代码如下所示：

```
//（1）第一次扫描：
label = 0;
labelSet[500];
LabelIndex[500];//用于记录哪个label与赛道等价

Init(labelSet);//labelSet为一个数组，记录每一个标记之间的等价关系
Init(Blabel);//Blabel二值化图像的label数组，与图像大小相等
Init(B);//B为二值化图像

for(x=0;x<length;x++){
```

```
for(y=0;y<height;y++){
if(B[x][y] == 1){// ,B(x,y)为当前访问点，B(x,y)==1即此点为白色
        if(与B(x,y)相邻的所有为1的点都无标记 || B(x,y)没有与其相邻的为1的点){
                label += 1;
                Blabel[x][y] = label;
        }
        else if(B(x,y)相邻的点中存在已经标记过的点){
                Blabel[x][y] = min{Neighbors};

                //如果存在多个相邻的有标记的点，且标记不相同，记录Neighbors中各个值之间的等价关系，即这些值（label）同属于同一个连通区域

                if(num(Neighbors) > 1){
                UnionSets(FindSets(label_1),FindSets(label_2))//label_1 ,label_2为两个等价的标记，其中label_1为二者中比较小的那个
                }
        }
}
}
}
//寻找起始点，找到位于赛道上的一个点
StartPoint = FindStart();
//查并集，将等价的label合并到一起
Father = FindSets(Blabel(StartPoint.x,StartPoint.y)); //找到起始点所属的最小的标记
for(i = 0;i<LabelNum;i++){
        if(Father == FindSets(i)) LabelIndex[i] = 1;
        else LabelIndex[i] = 0;
}
```

```
37
38  //第二次扫描，将与赛道主体等价的标记赋值
39  for(x = 0;x<length;x++){
40  for(y = 0;y<height;y++){
41          if(LabelIndex[Blabel[x][y]]) B[x][y] = 1;
42          else B[x][y] = 0;
43  }
44  }
```

1.2 Seed Filling 算法

Seed Filling 的主体思路是选取一个前景像素点作为种子，然后根据连通区域的两个基本条件（像素值相同，位置相邻）将与种子相邻的前景像素合并到同一个像素集合中，最后得到的该像素集合则为一个连通区域。

下面是 Seed Filling 算法的连通区域分析方法的伪代码：

```
1   //寻找起始点，给到位于赛道上的一个点
2   StartPoint = FindStart();
3   PointStack[500];//堆栈
4   StackPointer = 0;//堆栈的指针
5
6
7   Init(Blabel);//Blabel二值化图像的label数组，与图像大小相等
8   Init(B);//B为二值化图像
9
10  Push(StartPoint);
11  Blabel[StartPoint.x][StartPoint.y] = 1;
12  while(StackPointer > 0 ){//当堆栈不为空
13          Point = Pop(PointStack);
14          Blabel[Point.x][Point.y] = 1;
15          Push(Neighbor_Point);//将与之相邻的其他的点压入堆栈
16  }
```

```
17
18  //第二次扫描，将与赛道主体等价的标记赋值
19  for(x = 0;x<length;x++){
20          for(y = 0;y<height;y++){
21                  if(Blabel[x][y] == 1) B[x][y] = 1;
22                  else B[x][y] = 0;
23          }
24  }
```

2 连通域算法的具体实现

Two-Pass 算法以及 Seed Filling 算法二者各有优劣，Two-Pass 算法需要对所有的区域都进行扫描，而 Seed Filling 算法类似于深度优先搜索，但在循环过程中，判断哪个点没有被使用较为困难。因此，我选择了使用 Two-Pass 算法进行连通域的测算与分析。

具体代码请参照 Github 上面的示例程序，在使用程序前，请安装 opencv 的 C++ 库，如果你是用的是 Visual Studio 2019 及以上版本（以下版本可能会出现版本不兼容的情况），理论上可以直接运行.

另外，读取图片的代码如下所示：

```
1  imag = imread("test.png", IMREAD_GRAYSCALE);//读取灰度图
2  threshold(imag, result,0,255,THRESH_OTSU);//使用大津法进行二值化
```

由于最开始进行测试时，我选择的是输入灰度图，再将其转换为二值化的方式。如果你想要直接输入灰度图，请根据情况进行修改。

保存图片的代码如下图所示：

```
1  imshow("out", result);
2  imwrite("out.png", result);
3  waitKey(0);
```

输出的结果如下图所示：

图 2.3: 原图

图 2.4: 二值化结果

图 2.5: 连通域结果

三、其他的图像处理方法

1 二值化

一般来讲，二值化的方式有平均阈值，大津法阈值，Sobel/Roberts 等算子进行边缘提取。对于阈值的方法，我们首先需要计算出一个二值化需要的阈值，接着再对整张图片进行扫描，所有灰度值小于阈值的均置 0，而所有灰度值大于阈值的均置 1。

但由于赛道上光线往往是不确定的，光线的分布也不一定是均匀的，每张图片的亮度都不一样，因此，我们阈值往往是不确定的，或者说是波动的。因此，我们并不能简单的确定一个阈值便不改变了，而是根据每张图片分别计算阈值。平均阈值的方法是我们将所有的像素值加和求平均获得阈值，在加上/减去一个修正量，但经过实验我们发现，使用这种方法计算出来的阈值往往并不是很准确，要么偏大，要么偏小。因此，后来我们便采用了大津法阈值的策略，大津法是一种图像灰度自适应的阈值分割算法，是 1979 年由日本学者大津提出，并由他的名字命名的。大津法按照图像上灰度值的分布，将图像分成背景和前景两部分看待，前景就是我们要按照阈值分割出来的部分。背景和前景的分界值就是我们要求出的阈值。遍历不同的阈值，计算不同阈值下对应的背景和前景之间的类内方差，当类内方差取得极大值时，此时对应的阈值就是大津法（OTSU 算法）所求的阈值。

大津法的具体程序请参照龙邱给出的示例程序。

大津法阈值的方式能在一定情况下解决光照强度不均匀的问题，但无法解决一张图片上光照强度不均匀的情况（如地板反光，强光直射等等）。因此，我们可以进一步尝试动态阈值，即每个区域都有不同的阈值。

2 滤波

滤波主要的目的是去除图像中的噪点，防止其对图像的识别造成干扰，在这里，我们采用了最简单的阈值识别方式,当一个白点周围与之相邻的黑点数目大于 2，那么将其置黑，当一个黑点周围与之相邻的白点数目大于 2，那么将其置白。具体情况请参照龙邱给出的示例程序。

另外，由于连通域的存在，滤波这个东西其实是可有可无的。其主要的目的是对图像进行修正，我们也可以根据自己的理解编写对应的滤波算法。

四、参考文献

[1] 连通域分析：https://blog.csdn.net/tiandijun/article/details/51279643

[2] 大津法阈值 (OSTU)：https://blog.csdn.net/dcrmg/article/details/52216622