数图第二次大作业

林嘉成 2016011498
(自动化系 自 66)

目录

1 提取过程相关说明

1.1 空洞填补

考虑使用 bwmorph3 中的 fill 操作进行空洞的初步填补。

```
img_bin = bwmorph3(img_bin,'fill');
```

1.2 中心线粗提取

考虑使用 bwskel 进行中心线的提取。

```
img_thin = bwskel(img_bin);
```

1.3 连通域分化及分叉点和端点检测

1.3.1 连通域分化

为了便于后面的操作，故选择在此步骤进行连通域的分化。调用 regionprop3 函数，根据返回的值进行操作，具体代码如下。在该步骤中，会对一些像素值比较小的点进行初步的过滤。对于标定不同连通域的方法为：声明一个和原图一样大小的 Label，在同一连通域处赋上相同的值。

```
VolumeThs = 25;
stats = regionprops3(img_bin,'Volume','VoxelIdxList');
NumConct = length(stats.Volume);
LenVoxel = stats.Volume;
IndVoxel = stats.VoxelIdxList;
NumConctUpd = 0;
IndVoxelUpd = {};
LenVoxelUpd = zeros();
%用于查询连通域的划分
DictImg = zeros(size(img_bin));
for i = 1:NumConct
if(LenVoxel(i,1) < VolumeThs)
img_bin(IndVoxel{i}) = 0;
continue;
end
NumConctUpd = NumConctUpd + 1;
IndVoxelUpd{NumConctUpd} = IndVoxel{i};
LenVoxelUpd(NumConctUpd,1) = LenVoxel(i,1);
DictImg(IndVoxelUpd{NumConctUpd}) = NumConctUpd;
end
```

1.3.2 分叉点和端点检测

调用 bwmorph3 中的'branchpoints','endpoints'。

```
BranchPoint = find(bwmorph3(DictImg, 'branchpoints'));
EndPoint = find(bwmorph3(DictImg, 'endpoints'));
```

1.4 修复细化图像

1.4.1 Prim 最小代价路径算法

修复图像采用 Prim 算法。具体实现为：将端点和分叉点设置为结点，以欧式距离作为代价进行建图，之后再采用 Prim 算法进行连接。

对于代价，没有按照作业说明上所说的可能性进行实现，而是采用了相对比较简单的欧式距离，因为一是考虑到，如果使用可能性作为代价，则可能出现最小支撑树错误率较大的情况, 即若可能性有些出入，则可能会出现连线十分诡异的现象；二是，即使上述情况可以通过一定的操作进行避免，但是根绝我浅显的对中心线提取的用途的了解，我认为使用欧氏距离与使用可能性作为代价值几乎差别不是很大。这里规定一个前提，即输入的增强图像是相对比较良好的图像。使用欧式距离和使用可能性作为代价就好比物理中的近似或者电力电子等工程上的近似计算，即在一定程度上，使用欧氏距离近似和使用可能性 (准确) 差别几乎可以忽略，且运行速度更快。具体代码如下

```
function [Trees] = Prim(Nodes,DictImg)
%% 求解邻接矩阵
NumNodes = length(Nodes);
[cpt_x, cpt_y, cpt_z] = ind2sub(size(DictImg), Nodes);
X1 = repmat(cpt_x,1,NumNodes);
X2 = repmat(cpt_x',NumNodes,1);
Y1 = repmat(cpt_y,1,NumNodes);
Y2 = repmat(cpt_y',NumNodes,1);
Z1 = repmat(cpt_z,1,NumNodes);
Z2 = repmat(cpt_z',NumNodes,1);
Graph = ((X1-X2).^2 + (Y1-Y2).^2 +(Z1-Z2).^2).^(0.5);
%同一连通域，距离设置为0
M1 = repmat(DictImg(Nodes),1,NumNodes);
M2 = repmat(DictImg(Nodes)',NumNodes,1);
Mask = ((M1 - M2)~=0);
Graph = Graph .* Mask;
%% Prim算法
VisitedNode = 1;
NeighborNode = 2:NumNodes;
Trees = [];
while length(VisitedNode) ~= NumNodes
    CostCurr = Graph(VisitedNode,NeighborNode);
    [Costmin,index] = min(CostCurr(:));
    [SrcNode,DstNode] = ind2sub(size(CostCurr), index);
    SrcNode = VisitedNode(SrcNode(1));
    DstNode = NeighborNode(DstNode(1));
    Trees = [Trees;SrcNode,DstNode,Costmin];
    VisitedNode = [VisitedNode;DstNode];
    NeighborNode(find(NeighborNode==DstNode)) = [];
end
end
```

1.4.2 关键点连线操作

用直线将关键点进行连接，即沿着两点之间的连线，每次连接距离为 1，连接之后进行填洞并再次细化。

1.5 细化图像关键点排序

细化图像的关键点的排序算法具体实现为：在分叉点去除的基础上，寻找图像的端点。对于每一个连通域，从一个端点搜索到另一个端点进行排序。具体代码实现如下

```
%对每个分支上的点进行排序
BranchNum = length(stats.Volume);
BranchLen = stats.Volume;
BranchVoxelInd = stats.VoxelIdxList;
BranchEp = find(bwmorph3(coro_tree, 'endpoints'));
CoroTreeSrt = {};
LabelEp = zeros(size(coro_tree));
LabelEp(BranchEp) = 1;
for i = 1:BranchNum
    Ep = find(LabelEp(BranchVoxelInd{i}) == 1);
    if(length(Ep) < 2)
        [subx,suby,subz] = ind2sub(size(coro_tree),stats.VoxelIdxList{i
            });
        CoroTreeSrt{i} = [subx,suby,subz];
        continue;
    end
    BranchMask = zeros(size(coro_tree));
    BranchMask(BranchVoxelInd{i}) = 1;
    %按照端点，每26邻域一搜索
    Region = BranchVoxelInd{i};
    Start = Region(Ep(1,1));
    End = Region(Ep(2,1));
    [XStart,YStart,ZStart] = ind2sub(size(coro_tree),Start);
    [XEnd,YEnd,ZEnd] = ind2sub(size(coro_tree),End);
    %加一个Mask，即每次只对一个分支进行操作
    NewBranch = [XStart,YStart,ZStart];
    BranchMask(XStart,YStart,ZStart) = 0;
    for j = 1:BranchLen(i,1)-1
        MiniVolume = BranchMask(XStart-1:XStart+1,...
        YStart-1:YStart+1,ZStart-1:ZStart+1);
        [dx,dy,dz] = ind2sub(size(MiniVolume),find(MiniVolume == 1));
        Nxyz = [dx,dy,dz] - [2,2,2] + [XStart,YStart,ZStart];
        NewBranch = [NewBranch;Nxyz(1,1),Nxyz(1,2),Nxyz(1,3)];
        BranchMask(Nxyz(1,1),Nxyz(1,2),Nxyz(1,3)) = 0;
        XStart = Nxyz(1,1);
        YStart = Nxyz(1,2);
        ZStart = Nxyz(1,3);
    end
    CoroTreeSrt{i} = NewBranch;
end
CoroTreeSrt = CoroTreeSrt';
coronary_show(CoroTreeSrt);
```

1.6 基于分叉点的分支分化

将分叉点去掉后，再次进行 regionprops 对连通域进行求解，得到不同的分支。在此之后，需要把分叉点补回原位，但是还要按照一定的顺序。具体实现为：首先遍历每一个分叉点，在分叉点周围的 342 邻域范围内进行端点搜索，对于每个端点，判断其连通域标签以及每个连通域的排序走向(即检测到的端点是第一个还是最后一个)，如果为首端点，则将分支点插在最前；如果是末端点，则将分支点插在最后。

2 实现效果

得到的效果图如下。

2.1 ours_066_c1.mha 文件效果图

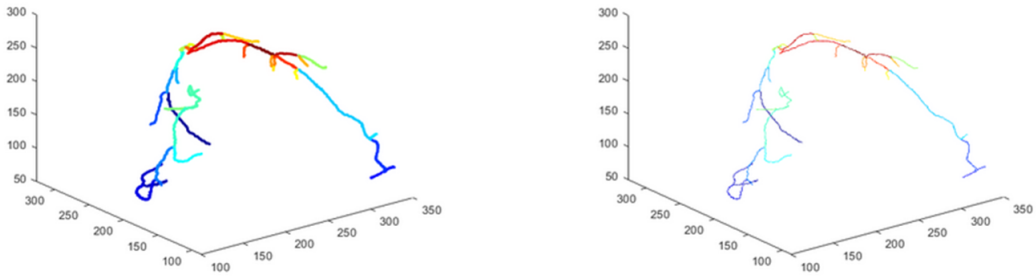

图 1: 左图为未排序散点图，右图为排序后的折线图

2.2 ours_054_c1.mha 文件效果图

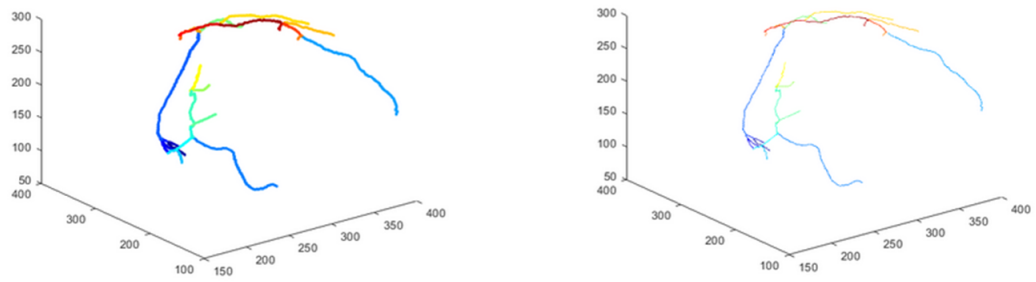

图 2: 左图为未排序散点图，右图为排序后的折线图

3 总结与反思

3.1 问题与解答

3.1.1 regionprops3.VoxelList 返回值问题

在求取连通域时，通过使用 regionprops3.VoxelList 可以直接得到连通域的坐标。但是与得到的坐标与通过 ind2sub 转换得到的坐标，x 方向与 y 方向的坐标竟然互换了。这一奇怪的现象耗费了较久的调试时时间，解决方案是使用 regionprops3.VoxelListIdx，用 ind2sub 进行转换，或者考虑将 regionprops3.VoxelList 返回的坐标的 x 与 y 互换。

3.1.2 电脑配置问题

在布置大作业前期，我发现我的电脑的配置严重不够，无法进行大作业程序的运行与调试，且服务器中的 Matlab 为 2016，有众多工具包无法使用。故尝试购买内存条稍微提升电脑性能。等了 4 天收到内存条，再次运行程序，发现提升内存有助于程序的运行，减少了虚拟内存的使用，且调试起来更加方便。

3.2 总结与反思

本次大作业首先由于自身电脑问题而耽搁了近一个星期，整体实现上所花费的时间并不需要很久。此外，在提高内存之后，我使用了 Matlab 的实时编辑器进行程序的调试，方便且快捷。

此外，本次大作业还是要有一些优化的不够到位的地方，但是由于其他更多的事情的关系，导致没有时间继续优化下去。