

基于物体轮廓检测和特征提取的木材图像计数模型

摘要

木材在运输前后都需要进行计数，传统的人工计数的方法既耗时耗力而且难免出现由于计数员的疏忽大意而数错的现象。随着社会的发展和人工成本的逐年上升，迫切需要利用新技术采用新方法来改进传统的计数方式。基于此，本文利用计算机图像识别针对两类图像实现木材计数。

我们将实际获得的图像分为两种类型，一种为受背景影响小的木材堆图像，另一种为受背景影响大的木材堆图像，围绕这两种问题，我们对木材图像计数进行了深入研究。

对于问题一，针对图像受背景影响小的情形，我们选取了相应的图像，建立了基于分水岭算法受背景影响小的图像处理模型。首先对图像进行数字化预处理，在这一过程中我们进行了灰度化处理、图像降噪、模糊增强、二值化处理、双边滤波、分水岭算法处理过程。对于模糊增强处理过程，利用了数学形态学理论，构建了迭代模型，调试迭代参数来增强边界信息。采用分水岭算法，我们提取了图像中木材的边界信息，从而可以解决实际中木材存在的粘连现象对计数结果造成的影响。经过以上预处理过程，我们得到了边界处理效果较好的图像，然后分别尝试 Hough 变换圆算法、连通面积法来实现图像的计数，比较计数结果，当边界存在粘连时，利用连通面积法处理图像，把人工计数结果作为标准值，以图像 1 为例，木材树木标准数目为 46 根，计算机识别为 45 根，相对误差仅为 2.1%，结果表明我们的方法对于处理背景影响小、存在边界粘连且大小不一的木材图像处理的适用性较好，准确性较高。

对于问题二，针对图像受背景影响大的图片，我们以例 1 为例对预处理的步骤进行了改进，并建立了基于 SIFT 算法的计数模型。我们经过一系列的对比尝试后选择利用灰度处理、图像降噪、边缘增强、HSV 变换、边缘检测、图像二值化处理以及图像分割的方法对原始图像进行预处理，预处理图像显示该处理步骤效果较好。接着利用 SIFT 算法，选取木材堆截面的特征图与预处理的图片进行比对，进而计算得到图片中的木材数为 652，与实际的木材数 634 相比，相对误差为 2.8%。通过随机选取相应的图片进一步检验模型的适应度，结果显示三张图片计数的相对误差分别为 2.2%、1.7% 和 1.8%。因此，我们的方法对于处理背景影响大、存在边界重合且大小一致的木材图像具有较好的适用性。

关键词：轮廓检测 特征提取 分水岭算法 SIFT 算法

一、问题重述

木材作为重要的建筑和工业原料，在社会生产生活中具有重要的地位。木材的长途运输一般按木材根数统计以校验中途是否丢失。因此一般在起运和到货时一般由双方人员进行木材根数的清点。目前木材商人是通过人工数数的方式来统计木材数量。如果每堆木材清点之后两个人的数量差距在千分之一以内，则认为该数量是准确的。此种方式虽然可靠，但是效率比较低下，且人工成本逐年上升。现在需要我们利用计算机算法来处理木材照片，从而达到从照片中直接获得木材的数量的目的。

二、问题分析

问题要求我们通过木材堆的照片获得木材的数量信息。我们将通常可以获得的木材堆图片分为两类分别讨论：一种为受背景影响小的木材堆图像，另一种为受背景影响大的木材堆图像。

针对问题一背景值影响较小的图像，木材边缘的增强处理和粘合处理是预处理的关键。我们可以利用数学形态学理论，构建迭代模型，调试迭代参数可以增强边界信息。采用分水岭算法，提取图像中木材的边界信息，从而解决实际中木材存在的粘连现象对计数造成影响。整个预处理包括：灰度化处理、图像降噪、模糊增强、二值化处理、双边滤波、分水岭算法处理过程。在计数方面，分别尝试 Hough 变换圆算法、连通面积法来实现图像的计数。

针对问题二背景值影响较大的图像，除了需要与问题一相似的灰度化处理、图像降噪和边缘增强以外，前期需要首先去除背景值的影响，考虑到可以利用 HSV 变换显示背景信息，接着利用边缘检测结合图像分割去除背景值信息。这样我们就可以突出显示每根木材的边界，接着提取图像内的特征木材端面，进而利用 SIFT 算法通过端面比对获得照片中木材的数量。

三、模型假设与约定

- 1、假设正常情况下人工所数木材数目即为准确数目。
- 2、假设拍摄的图像像素足够高，不影响图像处理。
- 3、假设同堆木材面积可能不同，但形状完全相同。
- 4、假设同堆方形木材的木材截面相同（方形木材为处理过的木材）。
- 5、假设木材堆积的地面无与木材截面形状相似的物体。

四、符号说明

符号	意义
C_M	图像经过处理后的对比度
G	增强后的图像梯度场
U_0	未梯度处理增强的图像
U	梯度处理增强的结果
$I(x, y)$	原始图像
$N(x, y)$	经双边滤波去噪后的图像
$L(x, y, \sigma)$	图像的尺度空间
$D(x, y, \sigma)$	尺度空间函数

五、模型建立与求解

5.1 基于分水岭法的低噪声背景木材计数

5.1.1 图片数字化预处理

首先需要对图片进行预处理，提高图像的对比度，从而提取出每个木材端面的特征，为下一步的计数做准备。图像预处理的具体流程如图 1 所示。



图 1 预处理流程图

5.1.1.1 灰度处理

木材端面图像一般是由数码相机、摄像机或手机等拍摄设备获取，因此木材端面图像一般都是 RGB 彩色图像。由于彩色图像包含的信息量一般都很大，不仅会占用较大的存储空间，而在一定情况下还会影响图片处理的速度。因此，在进行图像处理时，需要先把彩色图像进行灰度化处理，将其转换为只包含亮度信息而不包含色彩信息的灰度图像。

我们加权平均值法进行灰度化处理，满足公式：

$$R=G=B=W_R \times R+W_G \times G+W_B \times B$$

其中 W_R 、 W_G 、 W_B 分别是 R、G、B 三颜色分量的权重。实验表明当 $W_R=0.299$ ， $W_G=0.587$ ， $W_B=0.114$ 时，因为人类的视觉对绿色敏感度最高，而相应地对蓝色敏感度最低，这样取值后可以将图像转换为最适合人眼观察且数据量更易处理的灰度图像。

5.1.1.2 图像降噪

木材图像在拍摄过程中，受光线强度等因素的影响，难免会产生噪声，使图像的质量下降，为了优化图片质量，需要进行去噪处理，常用的有邻域平均法、掩模消噪法等。邻域平均法虽能有效抑制噪声，但同时也引起了模糊，模糊程度与邻域半径成正比。掩模消噪法只能降低噪声的强度，并不能有效去除噪声。这里选用中值滤波法，不仅可有效去噪，还不会使模糊程度加深。

中值滤波的基本原理是把数字图像中某一点的值用该点的一个邻域的各点值的中值交换。在输入图像 $x(n_1, n_2)$ 中，以任一像素为中心设置一个确定的邻域 A，A 的边长为 $2N+1$ ，($N=0,1,2,\dots$)。将邻域内各像素的强度值按大小顺序排列，取位于中间位置的值（中值）作为该像素点的输出值，遍历整幅图像就可完成整个滤波过程：

$$A=x(i,j), y=Med\{n_1, n_2, n_3, \dots, n_{2N+1}\}$$

其中， i 、 j 表示图片中相元的横纵坐标， Med 表示取中值。

5.1.1.3 图像模糊处理

在对图像进行模糊增强处理时，我们运用了数学形态学方法，其中数学形态学涉及到腐蚀、膨胀、开运算、闭运算等图像处理过程。

腐蚀是消除目标所有边界点的一个过程，是结果是目标沿周边比原目标少一个像素。我们使用腐蚀对木材边缘的点进行处理以去除图像中二值化之后边缘附近的小而无意义的目标。把结构元素 B 平移 a 后得到 Ba ，若 Ba 包含于 X，我们记下这个 a 点，所有满足上述条件的 a 点组成的集合称做 X 被 B 腐蚀的结果。用公式表示为：

$$E(X)=\{a | Ba \in X\}=X \ominus B$$

膨胀可以看做是腐蚀的对偶运算，其定义是：把结构元素 B 平移 a 后得到 Ba ，若 Ba 击中 X，则记下这个 a 点。所有满足上述条件的 a 点组成的集合称做 X 被 B 膨胀的结果。用公式表示为：

$$D(X)=\{a | Ba \uparrow X\}=X \oplus B,$$

基于腐蚀与膨胀，我们可以实现开运算、闭运算、顶帽运算、黑帽运算，具体原

理如下表所示：

表 1 模糊增强方法汇总表

开运算	先腐蚀再膨胀
闭运算	先膨胀再腐蚀
顶帽运算	开运算结果图与原图像之差
黑帽运算	闭运算结果图与原图像之差

为了对图像进行模糊增强我们采取了如图所示的迭代运算模型

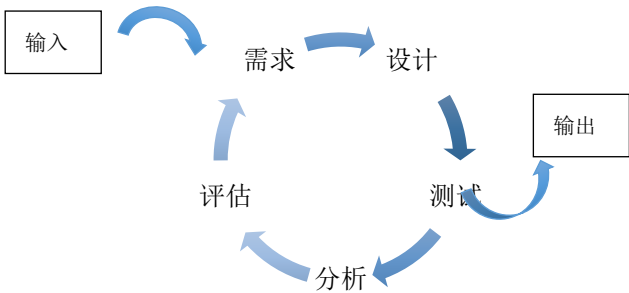


图 2 迭代运算模型示意图

我们在迭代运算模型下，依次进行开运算与腐蚀，不断调节迭代次数，选取模糊增强效果最好的图像处理方式。在这里我们对图像 1 进行灰度处理，图像降噪后进行模糊处理，我们对图一进行依次进行开运算、腐蚀，对比不同迭代次数，其中图像 1 中实际存在的木材数目为 46 根，迭代次数比较结果如表 2 所示。



图 3 背景影响小的木材图像 1

表 2 不同迭代次数的计数结果

开运算	腐蚀	计数结果	相对误差
0	7	44	4.3%
1	7	45	2.1%
1	5	48	4.3%
3	7	40	13.0%

对表 2 得到的相对误差进行比较，在迭代模型下，进行开运算 1 次，腐蚀 7 次，此时的计数准确性最高。

5.1.1.4 二值化处理

灰度阈值法是一种基于像素的分割方法，利用不同的阈值实现图像的分割，适用于目标和背景之间对比强烈的图像。在进行阈值分割时，大致分为 3 步：

- 第一、画出图像的灰度直方图，
- 第二、根据直方图的波谷，确定阈值
- 第三、用确定的阈值与图像中所有像素点的灰度值进行比较，若大于该阈值，则把像素

点归为目标, 反之, 则为背景, 完成分割。
灰度阈值法的处理过程如图 4 所示:



图 4 灰度阈值法处理过程

为了对图像进行分割, 确定木材边界, 我们采用了分水岭算法, 分水岭变换是一种著名的空间域的图像分割算法, 主要用于分割面对对象的图像轮廓。它是通过不断迭代进而对图像中的像素进行标记来达到分割的目的。在整个分割的过程中包括排序和淹没两个过程。先对图像中的灰度值进行由低到高进行排序, 然后由低到高进行淹没, 最后采用循环队列进行标注和判断。分水岭原理模拟示意图如图 5 所示。

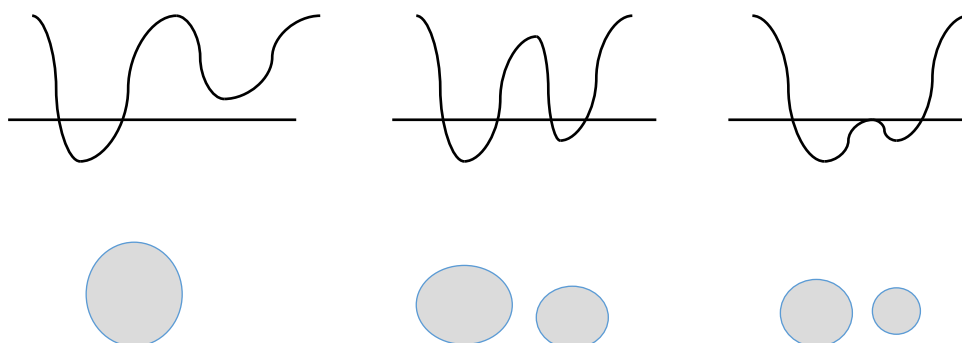


图 5 分水岭模型原理模拟图

对于一幅二维图像 $f(x, y)$, 令其梯度图像中所有局部极小值点坐标的集合分别为 $N_1, N_2 \dots N_i$, $C(N_i)$ 以表示与局部最小值批有关的集水盆, 令 $T[n]$ 表示坐标 (s, t) 的集合, 其中 $f(s, t) < n$, 即^[1]

$$T[n] = \{(s, t) | f(s, t) < n\}$$

其中 $T[n]$ 表示 $f(x, y)$ 中点的坐标集合, 这些集合中的点都在 $f(x, y) = n$ 的下方。

令 $C(N_i)$ 表示所有集水盆坐标点的集合, 集水盆的区域与第 n 阶段被覆盖的最小值有关:

$$C(N_i) = C(N_i) \cap T(n)$$

如果对于图像中的任意点 $(x, y) \in C(N_i)$, 且 $(x, y) \in T(n)$, 则有 $C(N_i) = 1$, 反之 $C(N_i) = 0$ 。

令 $C(n)$ 表示阶段 n 的集水盆地:

$$C(n) = \bigcup_{i=1}^R C(M_i)$$

那么,

$$C(\max+1) = \bigcup_{i=1}^R C(M_i)$$

经过以上分水岭算法, 我们得到的图像处理结果如下图6所示:

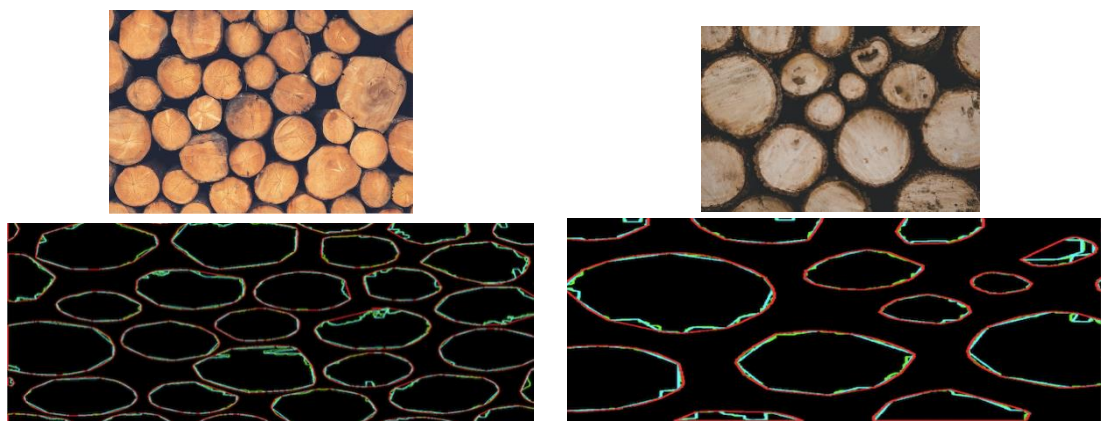


图 6 图像 1（左）、2（右）的分水岭边界处理效果

5.1.1.5 双边滤波

对于数字图像信号，噪声为或大或小的极值，这些极值通过加减作用于图像像素的真实灰度值上，对图像造成亮、暗点干扰，极大降低了图像质量，影响图像复原、分割、特征提取、图像识别等后继工作的进行^[2]。要构造一种有效抑制噪声的滤波器必须考虑两个基本问题：能有效地去除目标和背景中的噪声；同时，能很好地保护图像目标的形状、大小及特定的几何和拓扑结构特征，因此为了对图像中对木材进行噪声处理，我们比较了不同种类的滤波器，如下表 3 所示：

表 3 不同滤波器的比较

	原理	优点	缺点
领域平均滤波器	判断图像中的各像素点是否含有噪声	算法简单、使用灵活、计算速度快	会造成图像一定程度的模糊,特别是在边缘和细节处。
高斯滤波器	采用像素周围的邻域并找到其高斯加权平均值。	减少平滑处理中的模糊,得到更自然的平滑效果	没有考虑像素是否具有几乎相同的强度,不考虑像素是否是边缘像素,模糊了边缘。
中值滤波器	以滤波器包围的图像区域中所包含的像素的排序为基础,然后使用统计排序结果决定的值代替中心像素的值。	降噪时引起的模糊效应较低。	易造成图像的不连续性
双边滤波器	结合图像的空间邻近度和像素值相似度的一种折中处理,同时考虑空间与信息相似性	(1) 双边滤波技术注重图像像素之间的关系；(2) 双边滤波技术将图像的空间信息考虑在内	只能够对于低频信息进行较好的滤波

这里，我们针对图像 1 分别进行了以上滤波处理，效果图如图 7-8 所示：

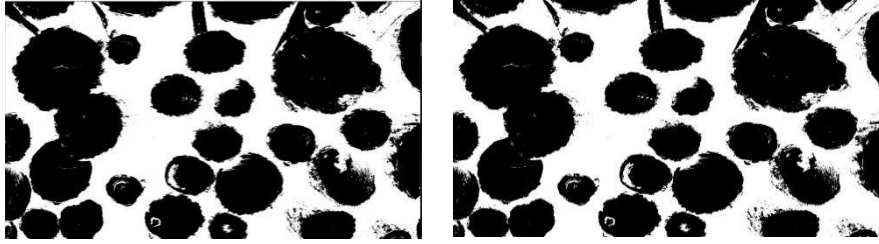


图 7 领域平均滤波器（左） 高斯滤波器处理（右）

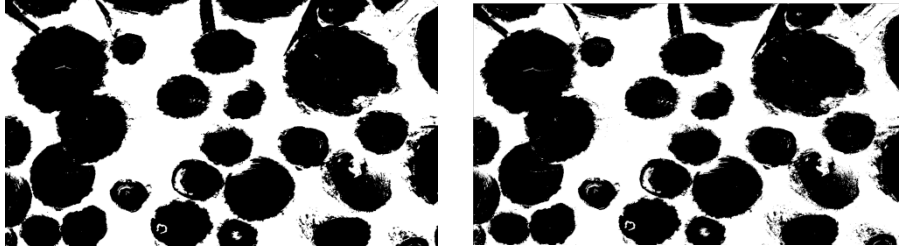


图 8 中值滤波器处理（左） 双边滤波器处理（右）

对比以上滤波器处理效果图，经过双边滤波处理后显然能够较好的消除噪声影响，因此我们在对图像进行去噪处理时，选择双边滤波的方法。双边滤波是一种非线性的滤波方法，能够达到保边去噪的目的，具有简单，非迭代，局部处理的特点之所以能够达到保边去噪的滤波效果是因为滤波器由两个函数构成：一个函数是由几何空间距离决定滤波器系数，另一个是由像素差值决定滤波器系数。

双边滤波器在空间中也采用高斯滤波器，但是还有一个高斯滤波器是像素差的函数。空间的高斯函数确保仅考虑附近的像素用于模糊，而强度差的高斯函数确保仅考虑具有与中心像素相似的强度的像素用于模糊，因此它保留了边缘，与其他滤波器相比在降低噪音方面非常有效，同时保持边缘清晰。在本文中，我们在进行木材图像处理时采用双边滤波法。

对于一个图像 $N(x)$ ，滤波函数表示如下^[3]：

$$N(x, y) = \frac{1}{\omega_p} \sum_{i,j \in \Omega} \omega_s(i, j) \omega_r(i, j) I(i, j) \quad (1)$$

$$\omega_s(\xi, c) = \exp \left[-\frac{d^2(\xi-c)}{2\sigma_s^2} \right] \quad (2)$$

$$\omega_r(\xi, c) = \exp \left[-\frac{\delta^2(\xi-c)}{2\sigma_r^2} \right] \quad (3)$$

$$\omega_p = \sum_{i,j \in \Omega} \omega_s(i, j) \omega_r(i, j) \quad (4)$$

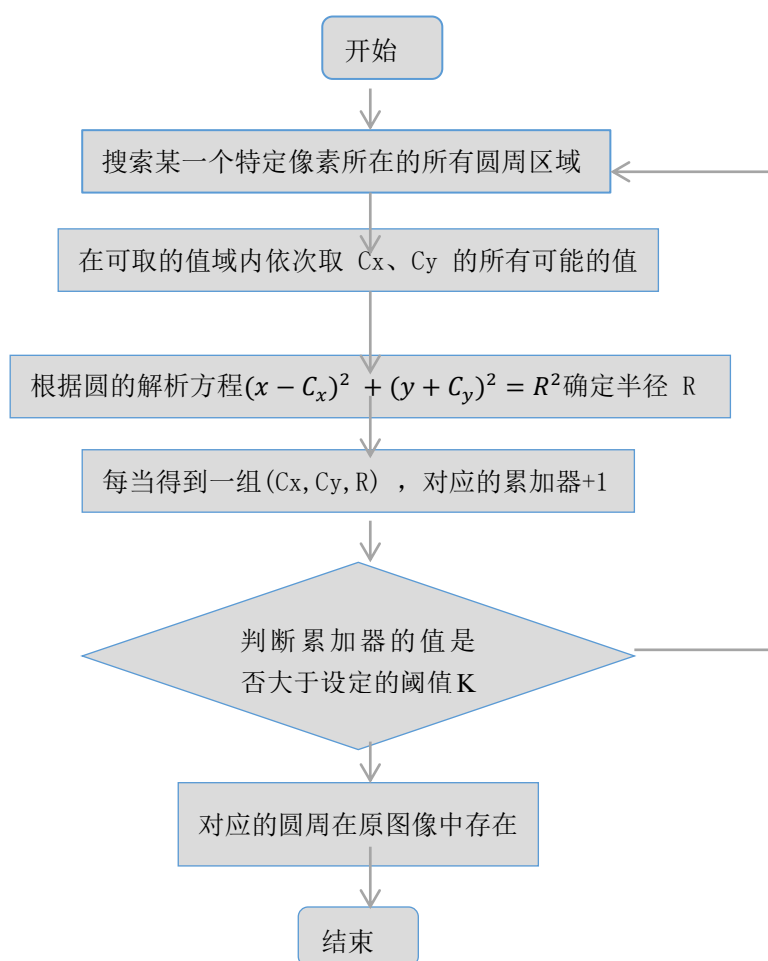
式中： $N(x,y)$ 表示去噪后图像； ω_p 为归一化参数； $I(i,j)$ 表示原始图像； Ω 表示滤波的窗口范围； $\omega_s(i,j)$ 表示高斯距离权值； $\omega_r(i,j)$ 表示高斯像素相似度权值； d 和 δ 分别表示两个像素 ξ 和 c 的空间距离差和像素差； σ_s 和 σ_r 分别表示相似度方差和空间方差。(1)式主要通过处理目标点位置像素与窗口内其余像素的距离和像素亮度信息作为权重进行加权平均之后，得到目标位置处的像素。

5.1.2 图像的计数处理

在对图像 1、2 进行木材识别计数时，我们尝试了两种计数算法，分别是 Hough 变换圆检测与连通面积法检测。

Hough 变换^[4]利用图像的全局特征将特定形状的像素连接起来，形成平滑边缘的一种方法。它将原图像上的点映射到用于累加的参数空间，实现对已知解析式曲线的识别。圆的解析方程为： $(x - C_x)^2 + (y - C_y)^2 = R^2$ ，从式中可以看出，圆的解析方程中含有 C_x 、 C_y 和 R 参数，因此圆的 Hough 变换需要使用三维的参数空间。定义三维参数空间 $C(C_x, C_y, R)$ ，其中 C_x 、 C_y 为原图像中圆周区域的圆心坐标， R 为圆周区域的半径，每一组 (C_x, C_y, R) 唯一确定一个圆周区域。

为每个圆周区域对应的参数空间指定一个累加器，那么圆的 Hough 变换的算法流程图如下图所示。



采用连通面积法进行检测，首先提取连通分量，算法实现的步骤如下^[5]：

- (1) 确定一个对应于4连通和8连通的 3×3 的模板结构元素，并初始化连通分量的标号为1。
- (2) 备份要标注连通分量的原图像，将图像的四周边界置为黑色(算法不处理边界上的点)，分配与原图像大小相同的目标图像空间，并初始化为全0(黑色为背景色，白色为前景色)。
- (3) 找到一个前景点(即像素值为 255)，初始化目标图像为只有连通区中的一点，从这一个点开始用 3×3 模板结构元素进行膨胀，计算和原图像的交集，限制膨胀不会超出区域。不断执行上述过程，直到与上一次膨胀后的结果相同，说明该连通区域已经提取完毕。

- (4) 用标号标注刚刚找到的连通区（即将唯一的标号赋给该连通区内的所有像素），记录当前连通区所有点的坐标，并将连通区标号加1以标注下一个连通区。
- (5) 循环执行步骤（4）步骤（5），直到找到所有的连通区域。

在背景影响小的情形下，为了处理木材面积大小不同的情况,我们采用了将每个连通域的面积与木材端面平均面积进行比较,从而利用面积法来进行计数。首先制定算法来确定木材的平均面积，然后基于以上连通提取分量过程，实现木材的计数过程，具体步骤如下^[2]：

- (1) 噪声区域的面积一般较小,小于0.1倍的棒材的平均面积Avg。所以计数时检测到的连通域面积 $A < 0.1 \text{ Avg}$ 时不予计数。
- (2) 由于某些棒材颜色较深、有划痕凹陷或曝光不足,二值化时棒材端面会有一部分被归为背景而导致一根棒材被分割成几块小连通域的情况。此时,连通区域的面积A一般满足 $0.1 \text{ Avg} < A < 0.5 \text{ Avg}$ 。搜索到一块满足条件的连通区域后,应在该连通区域的棒材平均半径范围内继续搜索,若还存在其他满足同样条件的连通区域则判定这是被割裂的棒材,并将它们合并起来计为一根棒材。否则被判为噪声,将其忽略。
- (3) 一个连通区域为一根或多根棒材,此时将连通区域的面积A应满足 $A > 0.5 \text{ Avg}$,棒材数量k按下式计算。

$$\frac{A}{\text{Avg}} - 0.5 < k \leq \frac{A}{\text{Avg}} + 0.5$$

分别利用Hough变换与连通面积法对图像1、2实现计数，计数结果如下表4所示：

表 4 Hough 变换与连通面积法计数结果比较

		Hough变换	连通面积法
图像1	计数结果	59	45
	相对误差	28. 2%	2. 1%
图像2	计数结果	23	18
	相对误差	35. 2%	5. 8%

从上表可知，采用连通面积法进行计数，计数的准确性较高，因此我们基于分水岭算法采用连通面积法对图像进行计数，得到的效果图，如图9所示：

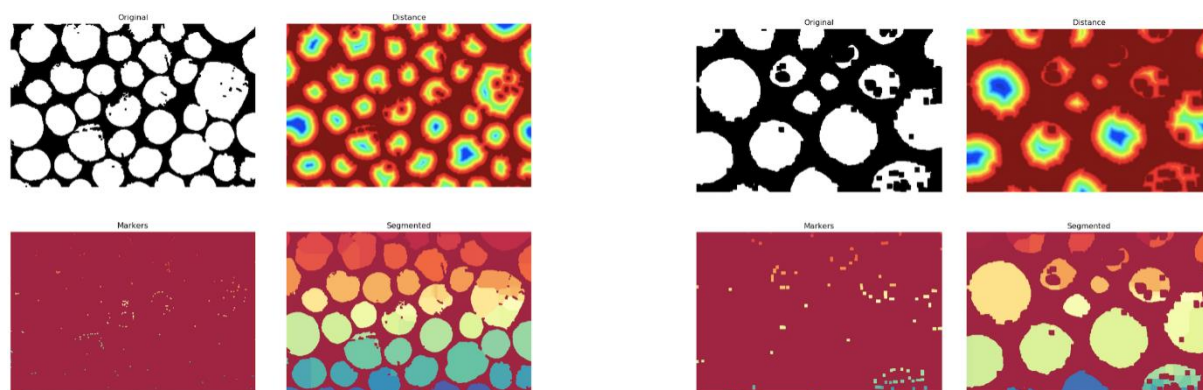


图 9 图像 1（左）、2（右）的计数效果图

5.2 基于SIFT算法的高噪声背景木材堆计数

在一个小节中我们分析了摄影机进行正面平摄时所获得的木材堆图片的计数问题，然而在实际的操作中，这种正面平摄由于工地条件的限制很难完全保证，因此实际能够拍摄的图片多为一定角度的仰视图，这就造成背景值会对目标值产生较大的影响。下面我们将以附件 1 所给图片为例，分析在背景值影响较大的情况下，如何处理木材照片并通过算法实现对木材堆进行有效计数。

通过将此图片与上一小节中木材端面图片的对比我们可以得知，在这张图片中有两个突出的特点：

- (1) 背景值影响大，其中，背景包括天空、地面阴影和地面沙子；
- (2) 由于木材已经过初步加工成方形，因此边缘重合度较高，不易区分。

这些因素会对预处理和计数结果产生较大影响，因此这将是我们在处理时需要着重解决的两个问题。

5.2.1 图片预处理

首先需要对图片进行预处理，提取出每个木材端面的边界，为下一步的计数做准备。图像预处理的具体流程如图 10 所示。其中，灰度化处理、图像降噪的模型建立与 5.1 节中相同，在此不再赘述。



图 10 预处理流程图

5.2.1.1 边缘增强处理

由于此类木材经过初步的加工，已形成较为规则的边界，这种边界的重合度较高，因此在边缘增强方面处理尤为重要。我们先尝试使用开运算、闭运算、梯度运算、顶帽运算和黑帽运算从期提高图像边缘的对比度，各种方法处理的结果如下图 11-13 所示。



图 11 灰度化图（左）、闭运算预处理（右）



图 12 开运算预处理（左）梯度运算预处理（右）

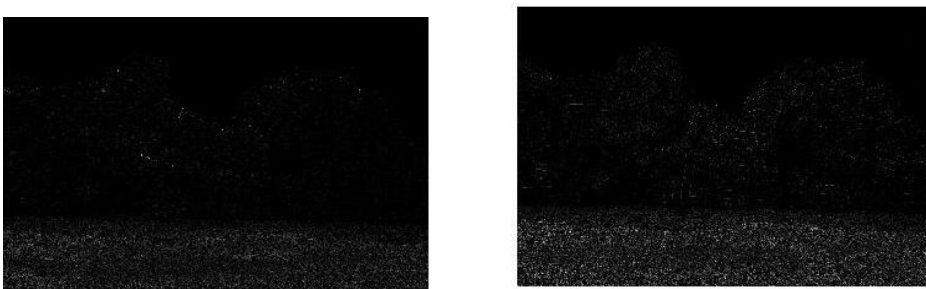


图 13 顶帽运算（左）黑帽运算（右）

从初步的边缘增强处理中，我们发现梯度处理所得到的边缘增强效果最好。因此选用梯度处理的方法来增强边界。

对比度又被称为“能见度”，定义为：

$$C_M = \frac{I_{max} - I_{min}}{I_{max} + I_{min}}$$

其中的 I_{max} 和 I_{min} 分别表示最亮的亮度和最暗的亮度。

直方图均衡化是指对原图像中的像素灰度进行某种映射变换，使得变换后图像的灰度级能够均匀分布,从而扩大动态范围并提高对比度。我们采用了一种基于梯度场直方图均衡化来增强图像对比度的方法，同时考虑到了直方图均衡化方法对于增强图像的梯度场具有相当好的鲁棒性。通过对图像的梯度场进行直方图均衡化，实现阴影或高光成分中的细节信息的增强化，而且在理论上直方图均衡化和熵最大化理论有密切的联系，它是熵最大化过程的近似实现。熵最大化过程会使得梯度场中各成分的出现概率趋向一致，在这种情况下图像的各种细节信息显然都能够被充分地表达出来，图像对比度达到最佳效果^[6]。

为了在增强图像梯度场的同时保持图像梯度的变化方向，使算法满足对比不变性要求，我们在处理图像过程中采用：

$$G = \frac{\Delta u_0}{\|\Delta u_0\|} \cdot L(\|\Delta u_0\|)$$

其中, u_0 为待增强的图像, $L(\|\Delta u_0\|)$ 表示原图像的梯度模 $\|\Delta u_0\|$ 经过直方图均衡化以后的取值, Δu_0 表示梯度场的方向信息。由于梯度场的均衡化可能导致图像梯度模的取值范围和原图像不一致, 而这会导致最后重建结果不理想, 因此以上的直方图均衡化过程限制在区间 $(0, \|\Delta u_0\|_{max})$ 内进行, $\|\Delta u_0\|_{max}$ 表示图像梯度模的最大值。

在得到增强后的梯度场以后，通过最小二乘原理得到目标函数

$$E(u) = \int_{\Omega} (\|\Delta u - G\|) dx dy \quad (1)$$

重建出增强结果。其中, u 表示增强结果, G 表示增强后的图像梯度场. 泛函式的 Euler-Lagrange 方程为

$$\Delta u = \text{div}(G)$$

上式即为 Poisson 方程，其中 Δ 是 Laplacian 算子， $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$

将图像表示为一个大小为 $m \times n$ 二维矩阵 U ，那么二维的 Laplacian 运算为：

$$\Delta U = A_{mm}U + UA_{nn} \quad (2)$$

其中 A_{mm} 和 A_{nn} 分别表示大小为 $m \times m$ 和 $n \times n$ 一维 Laplacian 算子。

用矩阵表示 Poisson 方程 $\Delta u = \text{div}(G)$ ：

$$A_{mm}U + UA_{nn} = V$$

其中 $V = \text{div}(G)$ ，考虑 Laplacian 算子矩阵的特性，可得到 $S_1^T \Lambda_1 S_1 U + U S_2^T \Lambda_2 S_2 = V$ ， Λ_1 和 Λ_2 分别表示对角元素为 A_{mm} 和 A_{nn} 特征值的对角阵。(2) 两边同乘以正弦矩阵 S_1 和 S_2 得到表达式：

$$S_1 S_1^T \Lambda_1 S_1 U S_2^T + S_1 U S_2^T \Lambda_2 S_2 S_2^T = S_1 V S_2^T$$

化简上式得到：

$$\Lambda_1 S_1 U S_2^T + S_1 U S_2^T \Lambda_2 = S_1 V S_2^T$$

令 $W = S_1 U S_2^T$ ，则有：

$$\Lambda_1 W + W \Lambda_2 = S_1 V S_2^T$$

通过 Kronecker 积方法将二维的矩阵运算向量化为：

$$[(I_{nn} \otimes \Lambda_1) + (\Lambda_2 \otimes I_{mm})] \text{vet}(W) = \text{vet}(S_1 V S_2^T)$$

其中 Λ_1 和 Λ_2 分别表示对角元素为 Λ_{mm} 和 Λ_{nn} 的特征值的对角阵， $\text{vet}(\cdot)$ 表示将二维矩阵按列排成一维向量。

由于 Laplacian 算子矩阵是负定矩阵，所以 $[(I_{nn} \otimes \Lambda_1) + (\Lambda_2 \otimes I_{mm})]$ 是一个以负值为对角元素的对角矩阵，其逆矩阵对角元素为矩阵 $[(I_{nn} \otimes \Lambda_1) + (\Lambda_2 \otimes I_{mm})]$ 对角元素的倒数。这样，矩阵 W 可以通过直接求逆矩阵的方法来实现，最后求取增强图像：

$$U = S_1^T W S_2$$

利用 Python 编程实现以上模型，结果如下图所示。

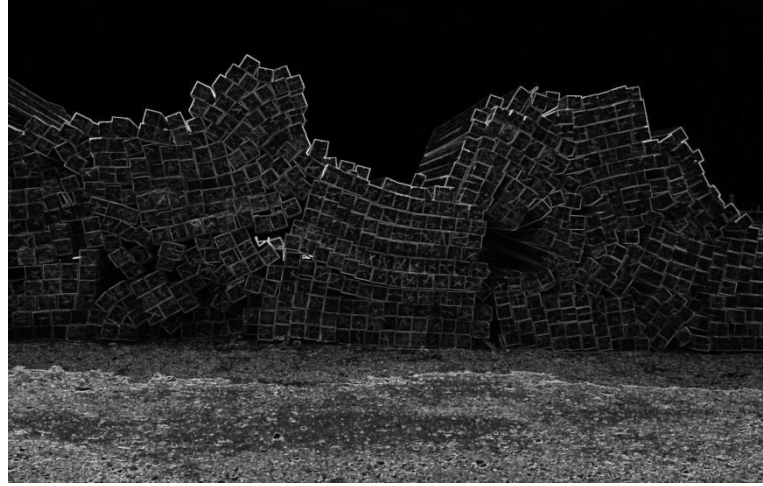


图 14 边缘增强处理结果

5.2.1.2 HSV 变换

由于方形端面木块的边界重合度较高，且地面沙子的灰度值与木块的灰度值接近，在计数时可能对木块的计数产生影响，因此需要将地面上沙子所在的区域去除。针对原始木材端面的 RGB 图像进行 HSV 变换，可显示出木材阴影与沙子较明显的区分线，为背景与目标的图片分割做基础。HSV 是代表亮度、明度和饱和度

(hue, saturation, value) 的色彩模型。它可以用近似的颜色立体来定量化。如图 15 所示。颜色立体曲线锥形改成上下两个六面金字塔状。环绕垂直轴的圆周代表色调

(H)，以红色为 0° ，逆时针旋转，每隔 60° 改变一种颜色并且数值增加 1，一周 360° 刚好六种颜色，顺序为红、黄、绿、青、蓝、品红。垂直轴代表明度 (V)，取黑色为 0，白色为 1。从垂直轴向外沿水平面的发散半径代表饱和度 (S)，与垂直轴相交处为 0，最大饱和度为 1。

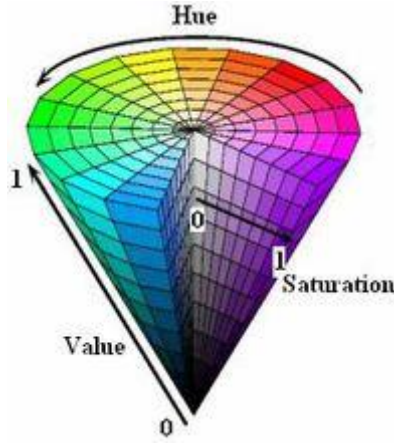


图 15 HSV 颜色模型

模型构建:

$$H = \begin{cases} \frac{(G - B) \times \pi}{3 \times [\max(R, G, B) - \min(R, G, B)]} & , R = \max(R, G, B) \\ \frac{(B - R) \times \pi}{3 \times [\max(R, G, B) - \min(R, G, B)]} & , G = \max(R, G, B) \\ \frac{(R - G) \times \pi}{3 \times [\max(R, G, B) - \min(R, G, B)]} & , B = \max(R, G, B) \end{cases}$$

$$S = \begin{cases} 0 & , \max(R, G, B) = 0 \\ \frac{\max(R, G, B) - \min(R, G, B)}{\max(R, G, B)} & , \max(R, G, B) \neq 0 \end{cases}$$

$$V = \max(R, G, B)$$

式中，H 表示经过 HSV 变换后的色调，S 表示 HSV 变换后的饱和度，V 表示经过 HSV 变换后的明度。R 表示 RGB 彩色空间中 (i, j) 点的红通道的值，G 代表绿通道的值，B 代表蓝通道的值。利用 Python 实现以上模型。

经过 HSV 变换后的图像如图 16 所示，从图片中我们可以明显地看出木材阴影与沙子的分界线。

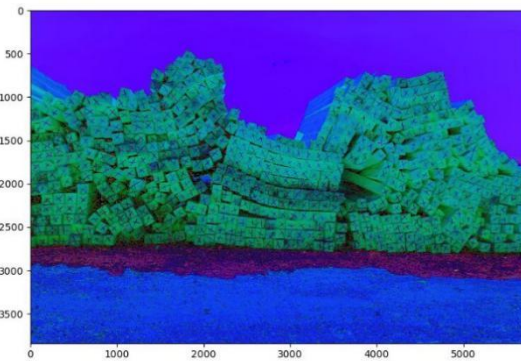


图 16 HSV 变换后结果

5.2.1.3 边缘检测

经过上一步 HSV 变换可以明显地看出地面沙子和木块堆的分界线，现在需要检验出分界线。分别采用连通法、傅里叶变换法、canny 边缘检测法进行边缘检测。其中，连通法模型的建立方法在上一小节已经说明，这里不再赘述，下面介绍傅里叶变换法

和 canny 边缘检测的模型建立。

(1) 傅立叶变换法

傅立叶变换提供一条从空域到频率自由转换的途径来观察图像，可以将图像从灰度分布转化到频率分布上来观察图像的特征。傅里叶频谱图上我们看到的明暗不一的亮点，图像上某一点与邻域点差异的强弱，即梯度的大小，也即该点的频率的大小一般来讲，梯度大则该点的亮度强，否则该点亮度弱。通过观察傅里叶变换后的频谱图可以看出，图像的能量分布，如果频谱图中暗的点数更多，那么实际图像是比较柔和的，反之，如果频谱图中亮的点数多，那么实际图像一定是尖锐的，边界分明且边界两边像素差异较大的。二维傅里叶变换公式如下所示^[7]：

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(\frac{ux}{M} + \frac{vy}{N})}$$

式中 $f(x, y)$ 代表一幅大小为 $M \times N$ 的矩阵，其中 $x = 0, 1, 2, \dots, M-1$ 和 $y = 0, 1, 2, \dots, N-1$ ， $F(u, v)$ 表示 $f(x, y)$ 的傅里叶变换。可以转换为三角函数表示方法，其中 u 和 v 可用于确定正余弦的频率。 $F(u, v)$ 所在坐标系被称为频域，由 $u = 0, 1, 2, \dots, M-1$ 和 $v = 0, 1, 2, \dots, N-1$ 定义的 $M \times N$ 矩阵常称为频域矩阵。 $f(x, y)$ 所在坐标系被称为空间域，由 $x = 0, 1, 2, \dots, M-1$ 和 $y = 0, 1, 2, \dots, N-1$ 所定义的 $M \times N$ 矩阵常被称为空间域矩阵。

(2) canny 边缘检测

在将木块端面图经过过去噪和增强的基处理后，构建 Canny 边缘检测模型检验每根木块的边界。构建模型如下：

1) 非极大值抑制。通常灰度变化的地方都比较集中，将局部范围内的梯度方向上，灰度变化最大的保留下来，其它的不保留，这样可以剔除掉一大部分的点。将多个像素宽的边缘变成一个单像素宽的边缘，即“胖边缘”变成“瘦边缘”。对梯度图像中每个像素进行非极大值抑制的算法是：

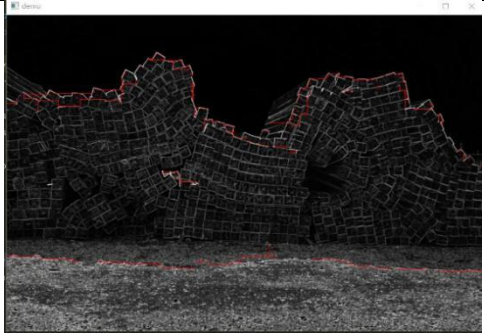
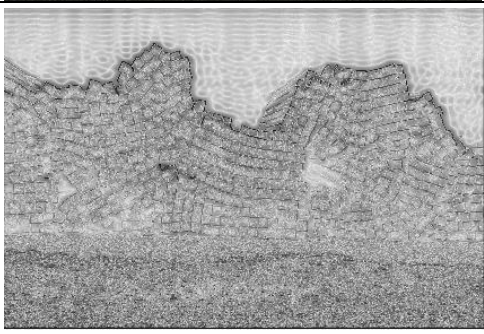
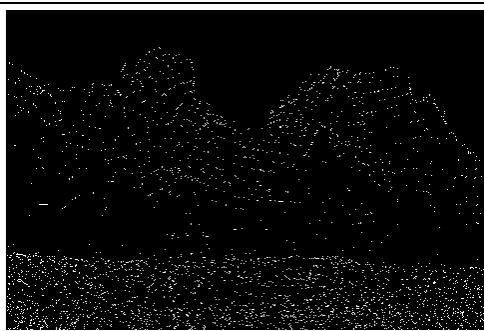
- i. 将当前像素的梯度强度与沿正负梯度方向上的两个像素进行比较。
- ii. 如果当前像素的梯度强度与另外两个像素相比最大，则该像素点保留为边缘点，否则该像素点将被抑制。

2) 双阈值筛选。通过非极大值抑制后，仍然有很多的可能边缘点，进一步的设置一个双阈值，即低阈值 (low)，高阈值 (high)。灰度变化大于 high 的，设置为强边缘像素，低于 low 的，剔除。在 low 和 high 之间的设置为弱边缘。进一步判断，如果其领域内有强边缘像素，保留，如果没有，剔除。这样做的目的是只保留强边缘轮廓的话，有些边缘可能不闭合，需要从满足 low 和 high 之间的点进行补充，使得边缘尽可能的闭合^[8]。

(3) 结果对比

表 5 边缘检测结果对比

方法	结果	特点
----	----	----

连通法		用时短 (1s), 效果好
傅里叶变换法		时间长
Canny 边缘检测		时间短 (1s), 但结果过于分散

从图上可以看出，利用连通法进行边缘检测的效果最好。在灰度图中以红线标出，接着利用工具裁剪掉灰度图中红色部分，结果如图 17 所示。

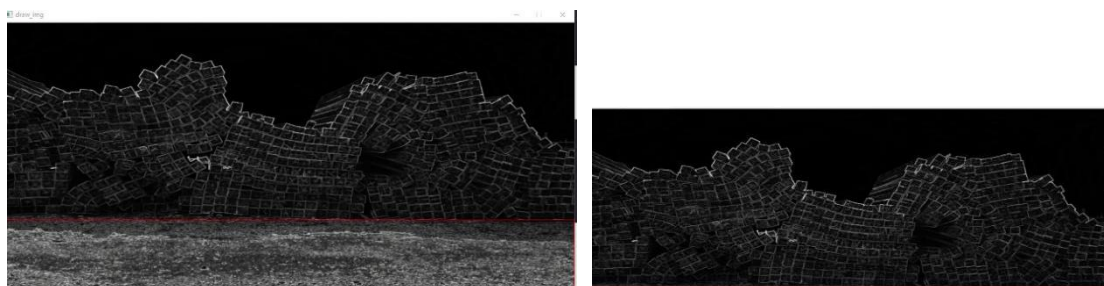


图 17 沙子边缘检测结果（左）去除沙子影响的灰度图（右）

5.2.1.4 二值化处理

二值化的建模过程同 5.1 中所述，我们分别采取全局阈值、局部阈值、自适应阈值进行处理，处理结果如图 18-19 所示：

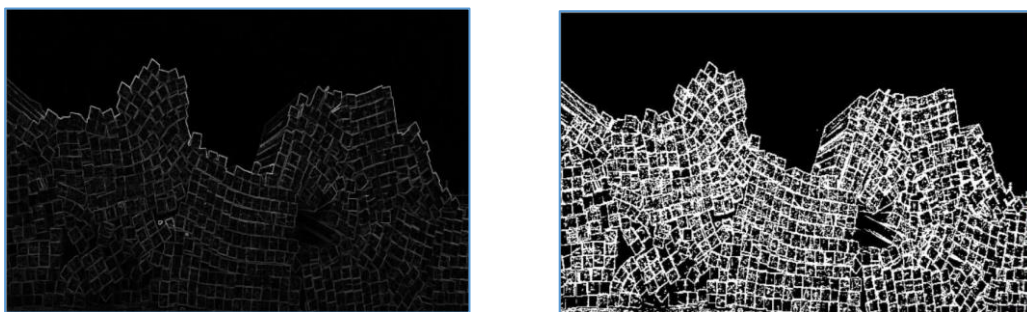


图 18 未处理图（左）自适应二值化图（右）

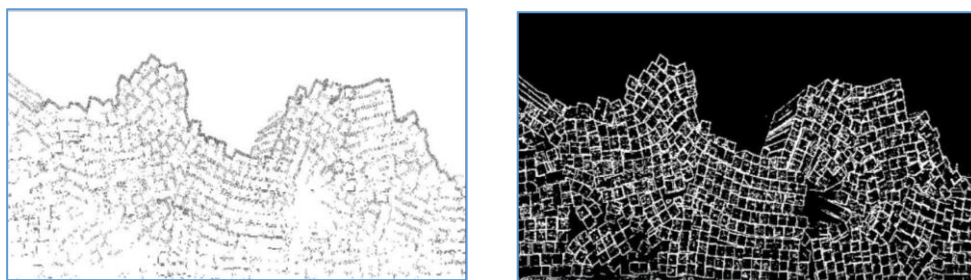


图 19 局部（左）、全局（右）阈值二值化

由结果可知，自适应二值化的方法处理效果最好，因此选用自适应二值化图（图 18 右）。

5.2.1.5 图像分割—灰度阈值法

灰度阈值法是一种基于像素的分割方法，利用不同的阈值实现图像的风格，适用于目标和背景间对比强烈的图像。在本问题中，我们假设目标和背景间虽然相邻像素的灰度值相近，但目标和背景的像素值总体上有差异。在图像的直方图上，目标和背景对应于不同的峰，分割阈值位于两峰之间的谷底。由于木材截面和背景填充具有较大的对比度，因此使用阈值分割法效果较好。

首先利用灰度直方图判断选择阈值。根据灰度阈值法的原理，结合图 20，我们可以看出图像具有双峰的特点，其中在高亮度峰的右侧具有一个弱峰，要将木材边缘分离出来，我们需要设定阈值。阈值应该位于两峰的谷底，因此最佳阈值 T 应该选择 5。

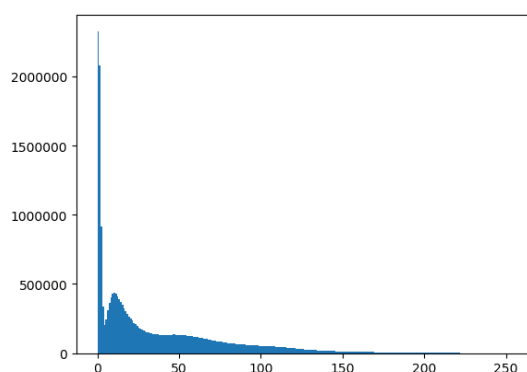


图 20 灰度直方图

接着利用以下述公式处理图像得到分割结果：

$$g(x,y) = \begin{cases} 1, & f(x,y) \geq T \\ 0, & f(x,y) < T \end{cases}$$

其中， T 指根据灰度直方图所得到的阈值， x,y 指在木材端面图片中像素点的坐标，

$f(x,y)$ 指分割处理前该 (x,y) 点的灰度值， $g(x,y)$ 表示分割处理后 (x,y) 点的灰度值。
图片处理结果如图 21 所示。



图 21 图像分割处理结果

5.2.2 基于SIFT算法对木材端面进行计数

SIFT 的全称是 Scale Invariant Feature Transform，尺度不变特征变换，由加拿大教授 David G.Lowe 提出的。SIFT 特征对旋转、尺度缩放、亮度变化等保持不变性，是一种非常稳定的局部特征。

5.2.2.1 尺度空间的极值检测

在一定的范围内，无论木材的端面是大还是小，人眼都可以分辨出来。然而计算机却不具备这样的能力，在未知的场景中，计算机视觉并不能提供木材端面的尺度大小。因此可以将把木材端面在不同尺度下的图像都提供给机器，让机器能够对木材端面在不同的尺度下有一个统一的认知。在建立统一认知的过程中，要考虑的就是在图像在不同的尺度下都存在的特征点。

尺度空间理论的主要思想是利用高斯核对原始图像进行尺度变换，获得图像多尺度下的尺度空间表示序列，对这些序列进行尺度空间特征提取。二维高斯函数定义如下：

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

一幅二维图像，在不同尺度下的尺度空间表示可由图像与高斯核卷积得到：

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

其中 (x,y) 为图像点的像素坐标， $I(x,y)$ 为图像数据， L 代表了图像的尺度空间。 σ 称为尺度空间因子，它也是高斯正态分布的方差，其反映了图像被平滑的程度，其值越小表征图像被平滑程度越小，相应尺度越小。大尺度对应于图像的概貌特征，小尺度对应于图像的细节特征。因此，选择合适的尺度因子平滑是建立尺度空间的关键。

在这一步里面，主要是建立高斯金字塔和 DOG(Difference of Gaussian)金字塔，然后在 DOG 金字塔里面进行极值检测，以初步确定特征点的位置和所在尺度。

(1) 建立高斯金字塔

为了得到在不同尺度空间下的稳定特征点，将图像 $I(x,y)$ 与不同尺度因子下的高斯核 $G(x,y,\sigma)$ 进行卷积操作，构成高斯金字塔。

高斯金字塔有 o 阶，一般选择 4 阶，每一阶有 s 层尺度图像， s 一般选择 5 层。第

1 阶的第 1 层是放大 2 倍的原始图像，其目的是为了得到更多的特征点；在同一阶中相邻两层的尺度因子比例系数是 k ，则第 1 阶第 2 层的尺度因子是 $k\sigma$ ，然后其它层以此类推则可；第 2 阶的第 1 层由第一阶的中间层尺度图像进行子抽样获得，其尺度因子是 $k^2\sigma$ ，然后第 2 阶的第 2 层的尺度因子是第 1 层的 k 倍即 $k^3\sigma$ 。第 3 阶的第 1 层由第 2 阶的中间层尺度图像进行子抽样获得。其它阶的构成以此类推。

(2)建立 DOG 金字塔

DOG 即相邻两尺度空间函数之差，用 $D(x, y, \sigma)$ 来表示，如下述公式所示：

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma)$$

DOG 金字塔通过高斯金字塔中相邻尺度空间函数相减即可，如图 22 所示。在图中，DOG 金字塔的第 1 层的尺度因子与高斯金字塔的第 1 层是一致的，其它阶也一样。

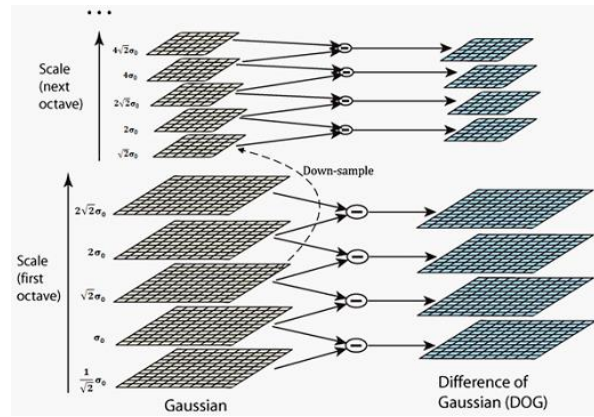


图 22 高斯图像金字塔 (S=2) 与 DOG 金字塔

(3)DOG 空间的极值检测

在上面建立的 DOG 尺度空间金字塔中，为了检测到 DOG 空间的最大值和最小值，DOG 尺度空间中中间层(最底层和最顶层除外)的每个像素点需要跟同一层的相邻 8 个像素点以及它上一层和下一层的 9 个相邻像素点总共 26 个相邻像素点进行比较，以确保在尺度空间和二维图像空间都检测到局部极值，如图 23 所示。

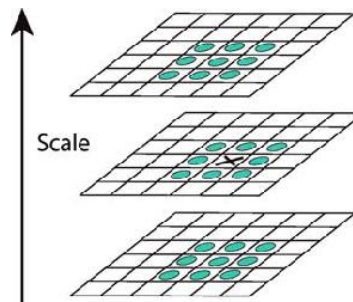


图 23 DOG 空间局部极值检测

在图 3 中，标记为叉号的像素若比相邻 26 个像素的 DOG 值都大或都小，则该点将作为一个局部极值点，记下它的位置和对应尺度。

5.2.1.2精确定位特征点位置

由于 DOG 值对噪声和边缘较敏感，因此，在上面 DOG 尺度空间中检测到局部极

值点还要经过进一步的检验才能精确定位为特征点。下面对局部极值点进行三维二次函数拟和以精确确定特征点的位置和尺度，尺度空间函数 $D(x, y, \sigma)$ 在局部极值点 (x_0, y_0, σ) 处的泰勒展开式如公式 () 所示。

$$D(x, y, \sigma) = D(x_0, y_0, \sigma) + \frac{\partial D^T}{\partial X} X + \frac{1}{2} X^T \frac{\partial^2 D}{\partial X^2} X \quad (8)$$

$$\text{其中 } X = (x, y, \sigma)^T, \quad \frac{\partial D}{\partial X} = \begin{bmatrix} \frac{\partial D}{\partial x} \\ \frac{\partial D}{\partial y} \\ \frac{\partial D}{\partial \sigma} \end{bmatrix}, \quad \frac{\partial^2 D}{\partial X^2} = \begin{bmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial x \partial y} & \frac{\partial^2 D}{\partial x \partial \sigma} \\ \frac{\partial^2 D}{\partial y x} & \frac{\partial^2 D}{\partial y^2} & \frac{\partial^2 D}{\partial y \partial \sigma} \\ \frac{\partial^2 D}{\partial \sigma x} & \frac{\partial^2 D}{\partial \sigma y} & \frac{\partial^2 D}{\partial \sigma^2} \end{bmatrix}。 \text{公式(4)中的一阶和二阶}$$

导数是通过附近区域的差分来近似求出的，列出其中的几个，其它的二阶导数以此类推。

通过对公式(8)求导，并令其为 0，得出精确的极值位置 X_{\max} ，如公式(9)所示：

$$X_{\max} = - \left(\frac{\partial^2 D}{\partial X^2} \right)^{-1} \frac{\partial D}{\partial X} \quad (9)$$

在上面精确确定的特征点中，同时要去掉低对比度的特征点和不稳定的边缘响应点，以增强匹配稳定性、提高抗噪声能力。

去除低对比度的特征点：把公式(9)代到公式(8)中，只要前两项，得到公式(10)：

$$D(X_{\max}) = D + \frac{1}{2} \frac{\partial D^T}{\partial X} \quad (10)$$

通过式(6)计算出 $D(X_{\max})$ ，若 $|D(X_{\max})| \geq 0.03$ ，则该特征点就保留下来，否则就丢弃。

去除不稳定的边缘响应点：海森矩阵如公式(11)所示，其中的偏导数是上面确定的特征点处的偏导数，它也是通过附近区域的差分来近似估计的。

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \quad (11)$$

通过 2×2 的海森矩阵来计算主曲率，由于 D 的主曲率与 H 矩阵的特征值成比例，根据文献[5]，不具体求特征值，求其比例 ratio。设 α 是最大幅值特征， $r = \frac{\alpha}{\beta}$ β 是次小的，则 $radio$ 如公式(12)所示。

$$\begin{aligned}
tr(H) &= D_{xx} + D_{yy} = \alpha + \beta \\
Det(H) &= D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta \\
radio &= \frac{tr(H)^2}{Det(H)} = \frac{(\alpha + \beta)^2}{\alpha\beta} = \frac{(\gamma\beta + \beta)^2}{\gamma\beta^2} = \frac{(\gamma + 1)^2}{\gamma}
\end{aligned} \tag{12}$$

由公式 (12) 求出 $radio$ ，常取 $r=10$ ，若 $radio \leq \frac{(r+1)^2}{r}$ 则保留该特征点，否则就丢弃。

5.2.1.3 确定特征点主方向

利用特征点邻域像素的梯度方向分布特性为每个特征点指定方向参数，使算子具备旋转不变性。

$$\begin{aligned}
m(x, y) &= \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2} \\
\theta(x, y) &= \arctan \frac{L(x+1, y) - L(x-1, y)}{L(x, y+1) - L(x, y-1)}
\end{aligned} \tag{13}$$

公式(13)为 (x, y) 处的梯度值和方向。 L 为所用的尺度为每个特征点各自所在的尺度， (x, y) 要确定是哪一阶的哪一层。在实际计算过程中，在以特征点为中心的邻域窗口内采样，并用梯度方向直方图统计邻域像素的梯度方向。梯度直方图的范围是 $0^\circ \sim 360^\circ$ ，其中每 10° 一个柱，总共 36 个柱。梯度方向直方图的峰值则代表了该特征点处邻域梯度的主方向，即作为该特征点的方向。在梯度方向直方图中，当存在另一个相当于主峰值 80% 能量的峰值时，则将这个方向认为是该特征点的辅方向。一个特征点可能会被指定具有多个方向(一个主方向，一个以上辅方向)，这可以增强匹配的鲁棒性。

通过上面的 3 步，图像的特征点已检测完毕，每个特征点有 3 个信息：位置、对应尺度、方向。

5.2.1.5 生成 SIFT 特征向量

首先将坐标轴旋转为特征点的方向，以确保旋转不变性。接下来以特征点为中心取 8×8 的窗口(特征点所在的行和列不取)。在图 24 左边，中央黑点为当前特征点的位置，每个小格代表特征点邻域所在尺度空间的一个像素，箭头方向代表该像素的梯度方向，箭头长度代表梯度模值，图中圈内代表高斯加权的范围(越靠近特征点的像素，梯度方向信息贡献越大)。然后在每 4×4 的图像小块上计算 8 个方向的梯度方向直方图，绘制每个梯度方向的累加值，形成一个种子点，如图 3 右边图所示。此图中一个特征点由 2×2 共 4 个种子点组成，每个种子点有 8 个方向向量信息，可产生 $2 \times 2 \times 8$ 共 32 个数据，形成 32 维的 SIFT 特征向量，即特征点描述器，所需的图像数据块为 8×8 。这种邻域方向性信息联合的思想增强了算法抗噪声的能力，同时对于含有定位误差的特征匹配也提供了较好的容错性。实际计算过程中，为了增强匹配的稳健性，文献[5]建议对每个特征点使用 4×4 共 16 个种子点来描述，每个种子点有 8 个方向向量信息，这样对于一个特征点就可以产生 $4 \times 4 \times 8$ 共 128 个数据，最终形成 128 维的 SIFT 特征向量，所需的图

像数据块为 16×16 。此时 SIFF 特征向量已经去除了尺度变化、旋转等几何变形因素的影响，再继续将特征向量的长度归一化，则可以进一步去除光照变化的影响。

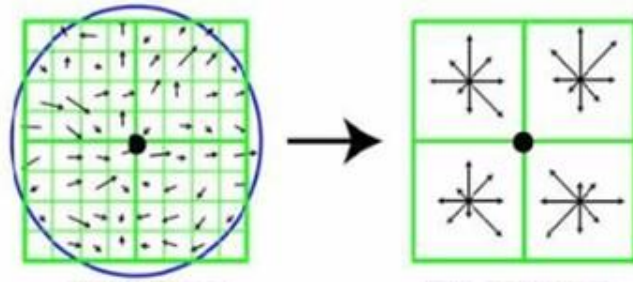


图 24 像梯度（左）及特征点描述器（右）

当两幅图像的 SIFT 特征向量，即特征描述器生成后，下一步就是进行特征向量的匹配。

5.2.1.6 SIFT特征向量的匹配

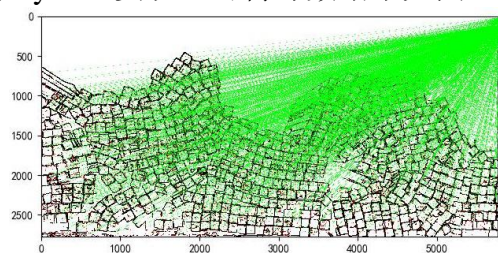
首先，进行相似性度量。一般采用各种距离函数作为特征的相似性度量，如欧氏距离、马氏距离等。

通过相似性度量得到图像间的潜在匹配。本文中采用欧氏距离作为两幅图像间的相似性度量。获取 SIFT 特征向量后，采用优先 k—d 树进行优先搜索来查找每个特征点的 2 近似最近邻特征点。在这两个特征点中，如果最近的距离除以次近的距离少于某个比例阈值，则接受这一对匹配点。降低这个比例阈值，SIFT 匹配点数目会减少，但更加稳定。

其次，消除错配。通过相似性度量得到潜在匹配对，其中不可避免会产生一些错误匹配，因此需要根据几何限制和其它附加约束消除错误匹配，提高鲁棒性。常用的去外点方法是 RANSAC 随机抽样一致性算法，常用的几何约束是极线约束关系^[9]。

5.2.2 计数结果

结合上述模型，利用 Python 实现，可得计数结果如图 25 所示。



the number of sticks:652

图 25 计数结果

图中共分为左右两个部分，左边为被匹配的图像或为原图转换的二值化图，右上角为匹配的图像（匹配的图像为被匹配图像中的单位木块），图中的红色圆点代表的是被匹配图像的特征，图中绿色的直线代表的是被匹配图像和匹配图像的特征匹配。

5.2.2 计数结果

人工计数可得图原图中的木材数为 634，相对误差为 2%。通过分析图像，我们认为

产生误差的原因主要在于预处理，可分为两个方面：

- (1) 未处理图片左上角的木块（如图 26 所示）。没有考虑到该木块不只有端面图像，还有侧面图像，导致重复计数，影响计数结果。
- (2) 图片左下角的木块由于在未处理前的初始状态时处于阴影中，灰度值过大在梯度处理时被当成背景去除，造成计数过小。
- (3) 匹配图像的选择存在误差。

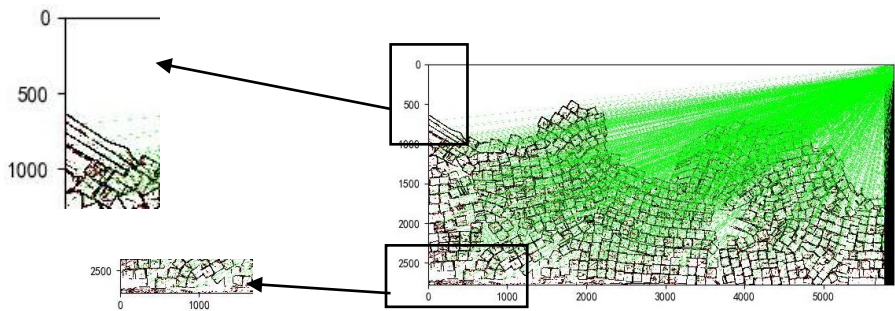


图 26 产生计数误差的局部图

六、模型检验

6.1 木材计数对于处理背景影响大木材堆图片的算法检验

为了检验我们的模型是否具有较强的实用性，我们选取了不同角度即变换不同位置的图片进行处理，这些图片中的木材间隙较小、颜色较浅，大小基本一致，并且受光照条件影响程度较大，我们基于SIFT算法对其进行计数识别，依次来检验算法。其步骤如一下流程图所示：

图 27 模型检验流程图



由于我们获得两张图像木材处于不同位置，这些图片中的木材的端面与正面端面相比有所倾斜，而且受背景影响条件比较大，经过以上处理步骤，以此来说明算法的准确性以及适用条件。

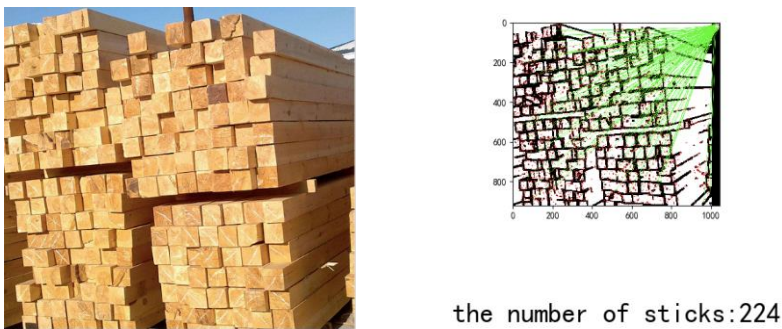


图 28 向左倾斜木材原图（左）、计数效果（右）

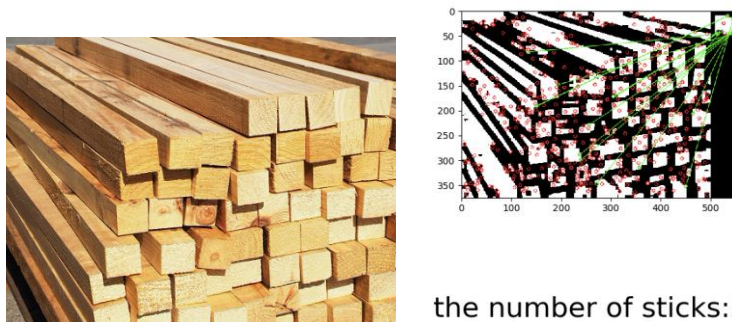


图 29 向右倾斜木材原图（左）、计数效果（右）

数字化预处理结合SIFT算法，我们能够处理光照不均匀、木材堆积程度密等背景影响较大的图像并对其进行计数，与此同时我们变换位置角度，选取向左倾斜和向右倾斜的图片来进行处理，从而扩展了处理木材计数问题时算法的适用范围。

6.2 结果讨论

利用以上预处理与计数的算法我们可以解决木材计数的问题，为了检验我们算法的适用性，并进行误差分析，我们将计算机处理结果与人工计数结果进行对比，结果如下表所示。

表 6 模型误差汇总表

	人工计数	机器计数	相对误差
向左倾斜	219	224	2.2%
向右倾斜	58	57	1.7%
正面	640	652	1.8%

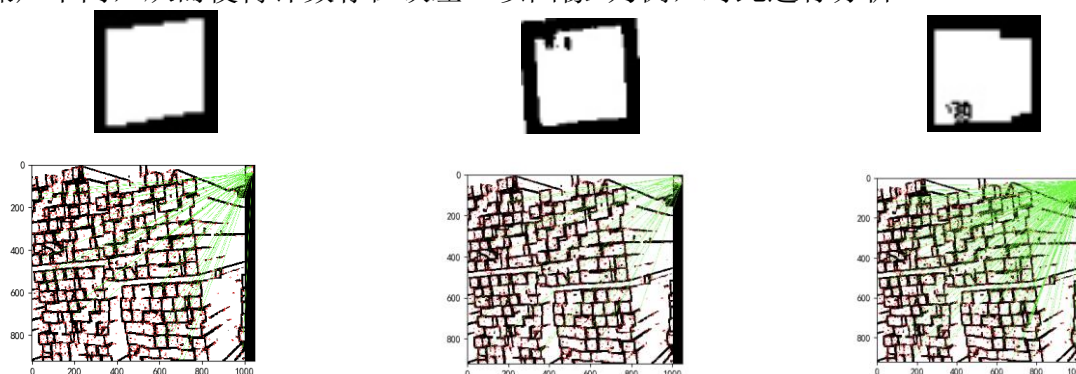
从上表可知SIFT算法处理角度不同的木材图像时，准确性较高。

6.3 误差分析

引起误差的原因分为多个方面，一方面图像由所处的环境即背景的复杂度造成，即：

- 1、光照对其影响较大，使其经过预处理后在计数过程仍存在一定的误差；
- 2、由于地面不平所导致的在图形切割方面存在误差等。

另一方面在计数处理过程中，选取不同的小方块来进行特征匹配时 由于不同方块表面噪声不同，从而使得计数存在误差。以图像3为例，对此进行分析。



the number of sticks:136

the number of sticks:184

the number of sticks:204

图 30 选取不同小方块来进行特征匹配的计数结果

考虑到以上存在引起误差到原因，结合相对误差结果，可以得出结论：我们的模型算法对于解决图像的角度问题、光照问题、密集问题、颜色问题等有较好的适用性，并且具有较高的准确性，这也说明了基于以上模型算法进行图像特征识别具有较好的鲁

棒性。

八、模型评价

	优点	缺点
模型一	针对目标信息受背景影响小的图像处理情形,优化的算法准确实现了木材计数	算法的计算准确率易受迭代次数的影响,需要不断调节参数来提高准确率
	可处理于边界粘连,大小不一的木材图像,适用条件较广	算法对于处理噪声过大的图像,存在较大的误差
模型二	针对目标信息受背景影响较大的图像处理,设计的算法解决了计数问题,具有很高的准确性	对于木材之间端面相差较大的情形,该算法的准确性降低
	可应用于受光照影响较大的、角度有所倾斜、边界密集集中的木材图像处理,适用条件广泛	在实现算法计数时,需要不断的选取不同被匹配对象进行比较,知道获得最优化结果

九、模型推广

基于背景影响大与背景影响小的木材图像模型,我们可以实现利用计算机算法来处理木材图片,从照片中计算得到木材的数量。这种处理方法可推广到光照不均匀、角度倾斜、边缘密集、形状不同的木材图像处理,可以通过增加环境参数进一步提高计数的准确度。

此外,在实际生活中,通常会涉及对木材堆的管理和计数工作。目前,我们主要还是以实现功能为主,下一步必须优化程序结构,改善用户界面,提高程序的交互能力。真正实现快速有效的计算机辅助木材计数系统。

十、参考文献

- [1]党文静. 图像区域分割算法的研究与应用[D]. 安徽理工大学, 2018.
- [2]陈基伟. 基于数字图像处理的棒材计数方法研究[D]. 山东大学, 2012.
- [3]于海平, 林晓丽. 基于增强双边滤波的图像分割模型及应用[J]. 计算机工程与设计, 2019, 40(04): 1064-1069.
- [4]钟新秀, 景林, 林耀海, 孙蕾. 结合k-means聚类和Hough变换的原木根数统计方法[J]. 宜宾学院学报, 2016, 16(12): 40-43.
- [5]周文欢, 郑力新. 提取连通分量算法在棒材自动计数中的应用[J]. 微型机与应用, 2011, 30(18): 38-41.
- [6]吴丽琼. 基于梯度的图像插值放大算法研究[D]. 山东大学, 2017.
- [7]康世英, 姚斌. 快速连通域标记算法在堆叠棒材计数中的应用研究[J]. 机械与电子, 2018, 36(11): 29-33.
- [8]梅安新, 彭望琚. 遥感导论. 北京: 高等教育出版社, 2001
- [9]傅卫平, 秦川, 刘佳, 杨世强, 王雯. 基于SIFT算法的图像目标匹配与定位[J]. 仪器仪表学报, 2011, 32(01): 163-169.

十一、附录

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy import ndimage as ndi
import skimage as sm
from skimage import morphology
from skimage.feature import peak_local_max
from skimage.io import imshow
from skimage.color import rgb2gray
from skimage.filters.rank import median
from skimage.measure import find_contours

# image = cv2.imread("./raw_data/1.jpg")
# dst = cv2.fastNlMeansDenoisingColored(image, None, 10, 10, 7, 21)
# img = cv2.pyrDown(dst, cv2.IMREAD_UNCHANGED)
## ret, thresh = cv2.threshold(cv2.cvtColor(img.copy(), cv2.COLOR_BGR2GRAY), 127,
255, cv2.THRESH_BINARY)
# thresh = cv2.adaptiveThreshold(cv2.cvtColor(img.copy(), cv2.COLOR_BGR2GRAY),
255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 25, 10)
# thresh = median(thresh, sm.morphology.disk(5))
# cv2.namedWindow("thresh", cv2.WINDOW_NORMAL)
# cv2.imshow("thresh", thresh)
# cv2.waitKey()
# cv2.destroyAllWindows()
#####

#####
# kernel = np.ones((3,3), np.uint8)
# opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations = 1)

# threshold, imgOtsu = cv2.threshold(thresh, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
# cv2.namedWindow("hull", cv2.WINDOW_NORMAL)
# cv2.imshow("hull", imgOtsu)
# cv2.waitKey()
# cv2.destroyAllWindows()
# cv2.namedWindow("hull", cv2.WINDOW_NORMAL)
# cv2.imshow("hull", thresh)
# cv2.waitKey()
# cv2.destroyAllWindows()
# findContours 函数查找图像里的图形轮廓
```

```

# 函数参数 thresh 是图像对象
# 层次类型, 参数 cv2.RETR_EXTERNAL 是获取最外层轮廓, cv2.RETR_TREE 是
获取轮廓的整体结构
# 轮廓逼近方法
# 输出的返回值, image 是原图像、contours 是图像的轮廓、hier 是层次类型
#####
#####
image = cv2.imread("./raw_data/4.jpg")
gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

ret,                                     thresh                                     =
cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
# thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY, 25, 10)

# noise removal
kernel = np.ones((3,3),np.uint8)
opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 1)
opening = cv2.bilateralFilter(opening,9,80,80)
opening = median(opening, sm.morphology.disk(3))
# opening = cv2.morphologyEx(opening,cv2.MORPH_GRADIENT,kernel, iterations = 1)
#####
#####
th, im_th = cv2.threshold(opening, 220, 255, cv2.THRESH_BINARY_INV)
# sure_bg = cv2.dilate(opening,kernel,iterations=2)
# Copy the thresholded image.
im_floodfill = im_th.copy()

# Mask used to flood filling.
# Notice the size needs to be 2 pixels than the image.
h, w = im_th.shape[:2]
mask = np.zeros((h + 2, w + 2), np.uint8)

# Floodfill from point (0, 0)
cv2.floodFill(im_floodfill, mask, (0, 0), 255)

opening = im_floodfill
opening = cv2.erode(opening,kernel,iterations=7)
#####
#####
image, contours, hier = cv2.findContours(opening, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
# 创建新的图像 black

```



```
black = cv2.cvtColor(np.zeros((image.shape[1], image.shape[0]), dtype=np.uint8),
cv2.COLOR_GRAY2BGR)
```

```
counter = 0
```

```
for p,cnt in enumerate(contours):
```

```
    area = cv2.contourArea(contours[p])
```

```
    if area < 30:
```

```
        print("$$$$")
```

```
        continue
```

轮廓周长也被称为弧长。可以使用函数 `cv2.arcLength()` 计算得到。这个函数的第二参数可以用来指定对象的形状是闭合的（`True`），还是打开的（一条曲线）

```
    epsilon = 0.01 * cv2.arcLength(cnt, True)
```

函数 `approxPolyDP` 来对指定的点集进行逼近，`cnt` 是图像轮廓，`epsilon` 表示的是精度，越小精度越高，因为表示的意思是原始曲线与近似曲线之间的最大距离。

第三个函数参数若为 `true`，则说明近似曲线是闭合的，它的首位都是相连，反之，若为 `false`，则断开。

```
    approx = cv2.approxPolyDP(cnt, epsilon, True)
```

`convexHull` 检查一个曲线的凸性缺陷并进行修正，参数 `cnt` 是图像轮廓。

```
    hull = cv2.convexHull(cnt)
```

勾画图像原始的轮廓

```
    cv2.drawContours(black, [cnt], -1, (0, 255, 0), 2)
```

用多边形勾画轮廓区域

```
    cv2.drawContours(black, [approx], -1, (255, 255, 0), 2)
```

修正凸性缺陷的轮廓区域

```
    cv2.drawContours(black, [hull], -1, (0, 0, 255), 2)
```

```
    counter+=1
```

显示图像

```
print(counter)
```

```
plt.imshow(black)
```

```
cv2.namedWindow("hull", cv2.WINDOW_NORMAL)
```

```
cv2.imshow("hull", black)
```

```
cv2.waitKey()
```

```
cv2.destroyAllWindows()
```

```
from scipy import ndimage as ndi
```

```
# labels = dst
```

```
distance = ndi.distance_transform_edt(opening) #距离变换
```

`min_distance`: 最小的像素在 $2 \times \text{min_distance} + 1$ 区分离（即峰峰数至少 `min_distance` 分隔）。找到峰值的最大数量，使用 `min_distance = 1`。

`exclude_border`: 不排除峰值在图像的边界

`indices`: `False` 会返回和数组相同大小的布尔数组，为 `True` 时，会返回峰值的坐标

```
    local_maxi = peak_local_max(distance, exclude_border = 0, min_distance = 12, indices=False,
```

```

footprint=np.ones((10, 10)),labels=opening)

#寻找峰值
markers = ndi.label(local_maxi)[0] #初始标记点
label_ =morphology.watershed(-distance, markers, mask=opening) #基于距离变换的分
水岭算法

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 12))
axes = axes.ravel()
ax0, ax1, ax2, ax3 = axes

ax0.imshow(opening, cmap=plt.cm.gray)#, interpolation='nearest')
ax0.set_title("Original")
ax1.imshow(-distance, cmap=plt.cm.jet, interpolation='nearest')
ax1.set_title("Distance")
ax2.imshow(sm.morphology.dilation(markers,sm.morphology.square(10)),
cmap=plt.cm.Spectral, interpolation='nearest')
ax2.set_title("Markers")
ax3.imshow(label_, cmap=plt.cm.Spectral, interpolation='nearest')
ax3.set_title("Segmented")
for ax in axes:
    ax.axis('off')

fig.tight_layout()
plt.show()

print('Load Image')

imgFile = './raw_data/1.jpg'

# load an original image
img = cv2.imread(imgFile)
#####

#####

# color value range
cRange = 256

rows, cols, channels = img.shape

# convert color space from bgr to gray
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#####

#####

```

```

thresh = median(imgGray, sm.morphology.disk(5))
kernel = np.ones((3, 3), np.uint8)
thresh = cv2.erode(thresh, kernel, iterations=3)#膨胀
# laplacian edge
imgLap = cv2.Laplacian(thresh, cv2.CV_8U)

# otsu method
threshold, imgOtsu = cv2.threshold(thresh, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)

# adaptive gaussian threshold
imgAdapt = cv2.adaptiveThreshold(thresh, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)
# imgAdapt = cv2.medianBlur(imgAdapt, 3)
#####
#####

canny = cv2.Canny(imgAdapt, 10, 20)
# 霍夫变换圆检测
circles = cv2.HoughCircles(canny, cv2.HOUGH_GRADIENT, 1, 50, param1=80,
param2=30, minRadius=0, maxRadius=50)
# 输出返回值，方便查看类型
# print(circles)

## 输出检测到圆的个数
print(len(circles[0]))
#
# print('-----我是条分割线-----')
# 根据检测到圆的信息，画出每一个圆
for circle in circles[0]:
    # 圆的基本信息
    print(circle[2])
    # 坐标行列(就是圆心)
    x = int(circle[0])
    y = int(circle[1])
    # 半径
    r = int(circle[2])
    # 在原图用指定颜色圈出圆，参数设定为 int 所以圈画存在误差
    img = cv2.circle(img, (x, y), r, (255, 255, 255), 1, 8, 0)
# 显示新图像
cv2.namedWindow("binary2", cv2.WINDOW_NORMAL)
cv2.imshow('binary2', img)

# 按任意键退出

```

```

cv2.waitKey(0)
cv2.destroyAllWindows()

image = cv2.imread("./raw_data/2.jpg")
gray = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)

ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
thresh = median(thresh, sm.morphology.disk(5))
kernel = np.ones((3, 3), np.uint8)
opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations=1)
sure_bg = cv2.dilate(opening, kernel, iterations=3) #膨胀
# binary = cv2.morphologyEx(binary, op= cv2.MORPH_GRADIENT, kernel=kernel)
dist_transform = cv2.distanceTransform(opening, 1, 5)
rets, sure_fg = cv2.threshold(dist_transform, 0.2*dist_transform.max(), 255, 0) #参数改小了，出现不确定区域
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg, sure_fg) #减去前景

# cv2.namedWindow("binary2", cv2.WINDOW_NORMAL)
# cv2.imshow('binary2', unknown)
# cv2.waitKey(0)
# cv2.destroyAllWindows()

#####
#####3
distance = ndi.distance_transform_edt(binary) #距离变换
# min_distance: 最小的像素在  $2 \times \text{min\_distance} + 1$  区分离（即峰峰数至少 min_distance 分隔）。找到峰值的最大数量，使用 min_distance = 1。
# exclude_border: 不排除峰值在图像的边界
# indices: False 会返回和数组相同大小的布尔数组，为 True 时，会返回峰值的坐标
local_maxi = peak_local_max(distance, exclude_border = 0, min_distance = 12, indices=False,
                                footprint=np.ones((10, 10)), labels=binary) #
寻找峰值
markers = ndi.label(local_maxi)[0] #初始标记点
label_ = morphology.watershed(-distance, markers, mask=binary) #基于距离变换的分水岭算法
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 12))
axes = axes.ravel()
ax0, ax1, ax2, ax3 = axes

ax0.imshow(binary, cmap=plt.cm.gray)#, interpolation='nearest')

```

```

ax0.set_title("Original")
ax1.imshow(-distance, cmap=plt.cm.jet, interpolation='nearest')
ax1.set_title("Distance")
ax2.imshow(sm.morphology.dilation(markers,sm.morphology.square(5)),      cmap=
plt.cm.Spectral, interpolation='nearest')
ax2.set_title("Markers")
ax3.imshow(label_, cmap= plt.cm.Spectral, interpolation='nearest')
ax3.set_title("Segmented")

for ax in axes:
    ax.axis('off')

fig.tight_layout()
plt.show()
#####

#####

#####
####
contours = find_contours(label_, 0.5)

#绘制轮廓
fig, (ax0,ax1) = plt.subplots(1,2,figsize=(16,16))
ax0.imshow(sure_fg,plt.cm.gray)
ax1.imshow(sure_fg,plt.cm.gray)

for n, contour in enumerate(contours):
    ax1.plot(contour[:, 1], contour[:, 0], linewidth=2)
ax1.axis('image')
ax1.set_xticks([])
ax1.set_yticks([])
plt.show()

print('总共有多少个●: ',len(contours))

def water_shed(image):
    #1 去噪，灰度，二值化
    blurred = cv.pyrMeanShiftFiltering(image,10,30)
    blurred=cv.bilateralFilter(image,0,50,5)
    # cv.imshow('blurred',blurred)
    gray=cv.cvtColor(blurred,cv.COLOR_BGR2GRAY)
    ret,binary=cv.threshold(gray,0,255,cv.THRESH_BINARY|cv.THRESH_OTSU)
    # cv.imshow('binary',binary)

```

```

#2. mophology 开操作去除噪点
kernel=cv.getStructuringElement(cv.MORPH_RECT,(3,3))
open_binary=cv.morphologyEx(binary,cv.MORPH_OPEN,kernel,iterations=2)
#mophology binary,2 次开操作
# cv.imshow('1-open-op',open_binary)
dilate_bg=cv.dilate(open_binary,kernel,iterations=3) #3 次膨胀
# cv.imshow('2-dilate-op',dilate_bg)
#3.distance transform
# DIST_L1:曼哈顿距离, DIST_L2: 欧氏距离,masksize:跟卷积一样
dist=cv.distanceTransform(open_binary,cv.DIST_L2,3) #? ?
dist_norm=cv.normalize(dist,0,1.0,cv.NORM_MINMAX)# 0-1 之间标准化
# cv.imshow('3-distance-t',dist_norm*50)

ret,surface=cv.threshold(dist,dist.max()*0.65,255,cv.THRESH_BINARY)
# cv.imshow('4-surface',surface)

#4 计算 marker
surface_fg=np.uint8(surface) #计算前景
unknown=cv.subtract(dilate_bg,surface_fg) #计算未知区域
# cv.imshow('5-unknown',unknown)
ret,markers=cv.connectedComponents(surface_fg) #通过计算 cc, 计算 markers
print(ret)
# cv.imshow('6-markers',markers)

#5 watershed 分水岭变换
markers=markers+1 #用 label 进行控制
markers[unknown==255]=0
markers=cv.watershed(image,markers) #分水岭的地方就编程-1
image[markers==-1]=[0,0,255]
cv.imshow('7-result',image)

src = cv.imread("./raw_data/4.jpg")
# cv.imshow("gray_img", src)
cv.namedWindow("result", cv.WINDOW_NORMAL)
water_shed(src)
cv.waitKey(0)
cv.destroyAllWindows()

mpl.rcParams['font.sans-serif'] = ['SimHei']

class processing_image:
    def __init__(self, filename="./raw_data/timg.jpg",

```



```

output="./out_data",icon="./raw_data/icon_timg.jpg"):
    self.filename = filename
    self.output = output
    self.icon = icon

def op_gray_to_four_type(self, kernel=(9, 9), erode_iter=5, dilate_iter=5):

    img = cv2.imread(self.filename)
    # gray
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # erode dilate
    closed = cv2.erode(img, None, iterations=erode_iter)
    img = cv2.dilate(closed, None, iterations=dilate_iter)

    kernel = np.ones(kernel, np.uint8)
    # open operation
    img_open = cv2.morphologyEx(img, op=cv2.MORPH_OPEN, kernel=kernel)
    # close operation
    img_close = cv2.morphologyEx(img, op=cv2.MORPH_CLOSE, kernel=kernel)
    # gradient operation
    img_grad = cv2.morphologyEx(img, op=cv2.MORPH_GRADIENT,
kernel=kernel)
    # tophat operation
    img_tophat = cv2.morphologyEx(img, op=cv2.MORPH_TOPHAT,
kernel=kernel)
    # blackhat operation
    img_blackhat = cv2.morphologyEx(img, op=cv2.MORPH_BLACKHAT,
kernel=kernel)
    # Plot the images
    images = [img, img_open, img_close, img_grad,
              img_tophat, img_blackhat]
    names = ["raw_img", "img_open", "img_close", "img_grad", "img_tophat",
"img_blackhat"]
    cv2.imwrite(self.output+"/gradient_image1.jpg",img_grad)
    fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(15, 15))
    for ind, p in enumerate(images):
        ax = axs[ind // 3, ind % 3]
        ax.imshow(p, cmap='gray')
        ax.set_title(names[ind])
        ax.axis('off')
    plt.show()

def op_first_to_three_type(self, flag=False):
    # 全局閾值

```

```

def threshold_demo(image):
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY) # 把输入图像
灰度化
    # 直接阈值化是对输入的单通道矩阵逐像素进行阈值分割。
    ret, binary = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY |
cv2.THRESH_TRIANGLE)
    if flag:
        cv2.imwrite(self.output + "/global_binary_first1.jpg", binary)
    return binary

# 局部阈值
def local_threshold(image):
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY) # 把输入图像
灰度化
    # 自适应阈值化能够根据图像不同区域亮度分布, 改变阈值
    binary = cv2.adaptiveThreshold(gray, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 25, 10)
    if flag:
        cv2.imwrite(self.output + "/local_binary_first1.jpg", binary)
    return binary

# 用户自己计算阈值
def custom_threshold(image):
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY) # 把输入图像
灰度化
    h, w = gray.shape[:2]
    m = np.reshape(gray, [1, w * h])
    mean = m.sum() / (w * h)
    ret, binary = cv2.threshold(gray, mean, 255, cv2.THRESH_BINARY)
    if flag:
        cv2.imwrite(self.output + "/custom_binary_first1.jpg", binary)
    return binary

if flag:
    src = cv2.imread("./out_data/gradient_image1.jpg")
else:
    src = cv2.imread(self.filename)
src = cv2.cvtColor(src, cv2.COLOR_BGR2RGB)
global_scr = threshold_demo(src)
local_scr = local_threshold(src)
custom_scr = custom_threshold(src)
images = [src, global_scr, local_scr,
          custom_scr]
names = ["src", "global_scr", "local_scr", "custom_scr"]

```

```

fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(10, 10))
for ind, p in enumerate(images):
    ax = axs[ind // 2, ind % 2]
    ax.imshow(p, cmap='gray')
    ax.set_title(names[ind])
    ax.axis('off')
plt.show()

def op_cutting_image(self):
    raw_img = cv2.imread(self.filename)
    img = cv2.imread("./out_data/gradient_image1.jpg")
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    blurred = cv2.bilateralFilter(gray, 7, sigmaSpace=75, sigmaColor=75)
    ret, binary = cv2.threshold(blurred, 127, 255, cv2.THRESH_BINARY)
    closed = cv2.dilate(binary, None, iterations=130)
    closed = cv2.erode(closed, None, iterations=127)

    _, contours, hierarchy = cv2.findContours(closed, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    c = sorted(contours, key=cv2.contourArea, reverse=True)[0]

    # compute the rotated bounding box of the largest contour
    rect = cv2.minAreaRect(c)
    box = np.int0(cv2.boxPoints(rect))
    # draw a bounding box around the detected barcode and display the image
    draw_img = cv2.drawContours(raw_img.copy(), [box], -1, (0, 0, 255), 3)

    h, w, _ = img.shape
    Xs = [i[0] for i in box]
    Ys = [i[1] for i in box]
    x1 = min(Xs)
    x2 = max(Xs)
    y1 = min(Ys)
    y2 = max(Ys)
    hight = y2 - y1
    width = x2 - x1
    crop_img = img[0:h - hight, x1:x1 + width]
    raw_img = raw_img[0:h - hight, x1:x1 + width]
    cv2.imwrite(self.output + "/raw_draw_image1.jpg", draw_img)
    cv2.imwrite(self.output + "/raw_cutting_image1.jpg", raw_img)
    cv2.imwrite(self.output + "/gray_cutting_image1.jpg", crop_img)

def op_edge_test(self):
    def gray_dege_test():

```

```

img = cv2.imread("./out_data/gradient_image1.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray, (9, 9), 0)
ret, binary = cv2.threshold(blurred, 127, 255, cv2.THRESH_BINARY)
closed = cv2.dilate(binary, None, iterations=110)
closed = cv2.erode(closed, None, iterations=120)

_, contours, hierarchy = cv2.findContours(closed, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

cv2.drawContours(img, contours, -1, (0, 0, 255), 3)
plt.imshow(img)
plt.show()
cv2.imwrite(self.output + "/gray_edge_test.jpg", img)

def fourier_edge_test():
    img = cv2.imread('./out_data/gradient_image1.jpg', 0)
    f = np.fft.fft2(img)
    fshift = np.fft.fftshift(f)

    rows, cols = img.shape
    crow, ccol = int(rows / 2), int(cols / 2)
    for i in range(crow - 30, crow + 30):
        for j in range(ccol - 30, ccol + 30):
            fshift[i][j] = 0.0
    f_ishift = np.fft.ifftshift(fshift)
    img_back = np.fft.ifft2(f_ishift) # 进行高通滤波
    # 取绝对值
    img_back = np.abs(img_back)
    plt.subplot(121), plt.imshow(img, cmap='gray') # 因图像格式问题，暂
已灰度输出

    plt.title('Input Image'), plt.xticks([]), plt.yticks([])
    # 先对灰度图像进行伽马变换，以提升暗部细节
    rows, cols = img_back.shape
    gamma = copy.deepcopy(img_back)
    rows = img.shape[0]
    cols = img.shape[1]
    for i in range(rows):
        for j in range(cols):
            gamma[i][j] = 5.0 * pow(gamma[i][j], 0.34) # 0.34 这个参数
是我手动调出来的，根据不同的图片，可以选择不同的数值
    # 对灰度图像进行反转

    for i in range(rows):

```

```

        for j in range(cols):
            gamma[i][j] = 255 - gamma[i][j]

plt.subplot(122), plt.imshow(gamma, cmap='gray')
plt.title('Result in HPF'), plt.xticks([]), plt.yticks([])
cv2.imwrite(self.output + "/fourier_edge_test_image1.jpg", gamma)
plt.show()

def canny_edge_test():
    img = cv2.imread('./out_data/gradient_image1.jpg', 0)
    edges = cv2.Canny(img, 100, 200)

    plt.subplot(121), plt.imshow(img, cmap='gray')
    plt.title('original'), plt.xticks([]), plt.yticks([])
    plt.subplot(122), plt.imshow(edges, cmap='gray')
    plt.title('edge'), plt.xticks([]), plt.yticks([])
    cv2.imwrite(self.output + "/canny_edge_test_image1.jpg", edges)
    plt.show()

gray_dege_test()
fourier_edge_test()
canny_edge_test()

def op_trans_plot(self):
    im_in = cv2.imread("./out_data/global_binary_first1.jpg",
cv2.IMREAD_GRAYSCALE)
    th, im_th = cv2.threshold(im_in, 220, 255, cv2.THRESH_BINARY_INV)

    # Copy the thresholded image.
    im_floodfill = im_th.copy()

    # Mask used to flood filling.
    # Notice the size needs to be 2 pixels than the image.
    h, w = im_th.shape[:2]
    mask = np.zeros((h + 2, w + 2), np.uint8)

    # Floodfill from point (0, 0)
    cv2.floodFill(im_floodfill, mask, (0, 0), 255)
    cv2.imwrite(self.output + "/edge_processing1.jpg", im_floodfill)

def op_counter(self):
    ob1 = cv2.imread("./out_data/edge_processing1.jpg",
cv2.IMREAD_GRAYSCALE)
    # ob1 = cv2.dilate(ob1, None, iterations=2)

```

```

ob1 = cv2.bilateralFilter(ob1, 7, sigmaSpace=70, sigmaColor=70)
ob1 = cv2.erode(ob1, None, iterations=2) # 1 # 2
ob1 = cv2.dilate(ob1, None, iterations=2)
ob2 = cv2.imread(self.icon, cv2.IMREAD_GRAYSCALE)
# ob2 = cv2.bilateralFilter(ob2, 7, sigmaSpace=60, sigmaColor=60)
ob2 = cv2.erode(ob2, None, iterations=1)
# ob2 = cv2.dilate(ob2, None, iterations=1)
# orb = cv2.xfeatures2d.SURF_create()
orb = cv2.xfeatures2d.SIFT_create()
keyp1, desp1 = orb.detectAndCompute(ob1, None)
keyp2, desp2 = orb.detectAndCompute(ob2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
search_params = dict(checks=50)
flann = cv2.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(desp1, desp2, k=2)
matchesMask = [[0, 0] for i in range(len(matches))]
for i, (m, n) in enumerate(matches):
    if m.distance < 0.7 * n.distance:
        matchesMask[i] = [1, 0]
# 如果第一个邻近距离比第二个邻近距离的 0.7 倍小，则保留
draw_params = dict(matchColor=(0, 255, 0), singlePointColor=(255, 0, 0),
matchesMask=matchesMask, flags=0)
img3 = cv2.drawMatchesKnn(ob1, keyp1, ob2, keyp2, matches, None,
**draw_params)
a = len(keyp1) // len(keyp2)
plt.figure(figsize=(8, 8))
plt.subplot(211)
plt.imshow(img3)
plt.subplot(212)
plt.text(0.5, 0.6, "the number of sticks:" + str(a), size=30, ha="center",
va="center")
plt.axis('off')
plt.show()
cv2.imwrite(self.output+"/counter_sticks_image1.jpg", img3)

if __name__ == '__main__':
    ob =
processing_image(filename="/raw_data/timber4.jpg", icon="/raw_data/icon_timber4.png")
    ob.op_gray_to_four_type()
    ob.op_first_to_three_type()
    # ob.op_cutting_image()
    ob.op_edge_test()
    ob.op_first_to_three_type(flag=True)

```



```
ob.op_trans_plot()  
ob.op_counter()
```
