# 缺陷检测

缺陷检测是视觉需求中难度较大的一类需求，主要是难以保证其稳定性和精度。常见缺陷有：凹凸、污点瑕疵、划痕、裂缝、探伤等。 在这里，介绍俩种在halcon中常见的缺陷检测任务：

表面检测任务，有几种方法可用:

通过与参考图像的比较，

通过差分模板比较图像包含相似的对象，并返回它们之间的差异。

通过检测均匀结构表面的缺陷描述

Blob分析，用于包含相似灰度值，颜色或纹理的区域的对象。

常见的纹理检测方法有:

快速傅里叶变换

分类器 通过训练分类器模型，将输入数据分为不同的类别。分类问题的输出是离散值，用来指定其属于哪个类别。

纹理分析

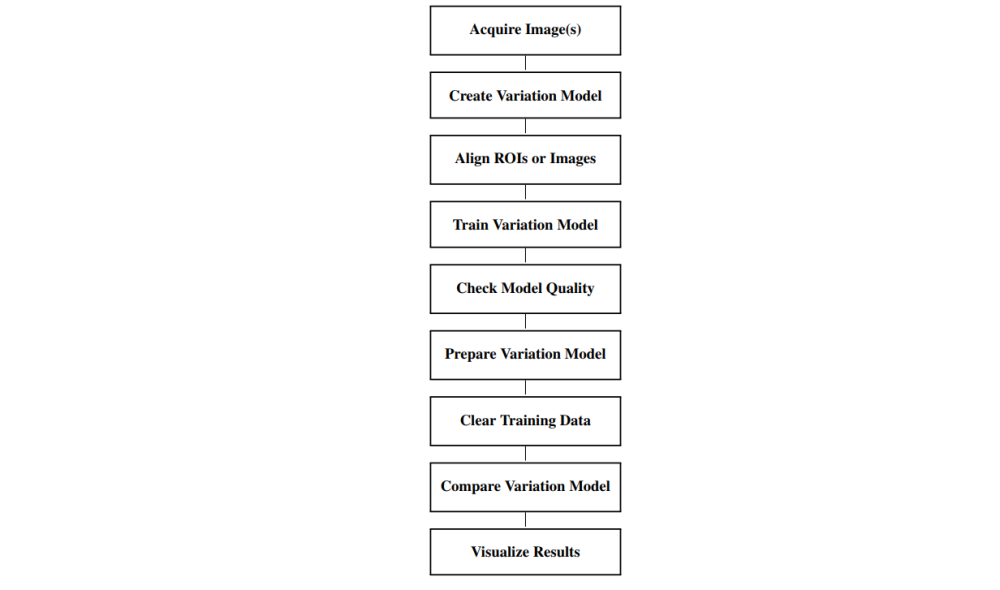
光度立体

频域+空间域结合

## 一、差分模型

差分模型的主要原理是将待检测的图像与一张理想图像作比较，找出待检测图像与理想图像的明显差异。理想图像通常是通过使用多个参考图像进行训练而获得的。除了理想图像之外，训练还派生出图像中每个点允许的灰度值差分信息，这些信息存储在所谓的差分图像中。进而创建出一个差分模型，其他图像可以与它进行比较。差分模型的优点是可以直接通过灰度值对图像进行比较，并通过差分图像进行空间加权。

差分模型主要由以下几个部分组成:



### 1.1 获取图像

训练和比较，都需要获取图像，即训练集和测试集。

### 1.2 创建差分模型

首先，使用create\_variation\_model创建用于图像比较的差分模型，用以存储在以下步骤中依次添加的信息。

### 1.3 对齐ROI或图像

为了训练差分模型，所有训练图像必须放置在相同位置并具有相同方向。因此，在训练模型之前，对象必须对齐。同样地，要与差分模型进行比较的图像也必须对齐。差分模型是用多幅好的物体图像来训练的。因此，训练图像必须显示相同位置和旋转的物体。如果这不能通过外部手段来保证，物体的姿态可以通过使用形狀匹配来确定(find\_generic\_shape\_model)，然后使用affine\_trans\_image将图像转换为参考姿态。

### 1.4训练差分模型

通过提供一组包含良好样本的图像来训练差分模型。算子train\_variation\_model使用训练图像计算出理想图像和相应的差分图像。差分图像表示图像上每一点的灰度值差分量，即容差。 在某些情况下，不可能获得多个训练图像，既而无法从单个训练图像中训练出有用的差分图像。为了解决这一问题，综合生成训练图像的变体，将训练图像在行、列方向上移动±1个像素，或者使用灰度值形态学(gray\_erosion\_shape和gray\_dilation\_shape)，然后训练合成的修改后的图像。从单个图像创建差分模型的另一种可行性方法是使用Mode= ' direct '创建模型。如果无法避免使用一些可能包含错误的物体图像来训练差分模型，则可以将Mode设置为“robust”。在该模式中，将理想目标的图像计算为所有训练图像在各自图像位置的中值，相应的差分图像被计算为训练图像和中值图像，在各自图像位置适当缩放训练图像与中值图像的中位数绝对偏差。这种模式的优点是不会被异常值所干扰，而它的缺点是不能迭代地训练，即必须使用concat\_obj连接所有训练图像，并在一次调用中使用train\_variation\_model进行训练。

### 1.5准备差分模型

为了准备图像比较的差分模型，将理想图像和差分图像转换为两个阈值图像。如果使用train\_variation\_model算子训练多个图像去训练模型，则使用算子prepare\_variation\_model准备模型。如果使用单个图像作为理想图像，并通过过滤理想图像手动创建相应的差分图像（如使用sobel\_amp, edges\_image或gray\_range\_rect），则必须使用prepare\_direct\_variation\_model进行准备，因为理想图像和差分图像尚未连接到差分模型。获得的阈值图像可以使用算子get\_thresh\_images\_variation\_model直接读出。

### 1.6比较差分模型

使用算子compare\_variation\_model将图像与准备好的差分模型进行比较。使用在准备模型步骤中获得的两个阈值图像来确定包含图像中与模型显著不同的所有点的区域。也可以使用compare\_ext\_variation\_model代替，可以设置比较时的其他扩展参数。

### 1.7测试模型精度

用多张图像训练一个差分模型后，可以通过get\_variation\_model算子查询理想对象的图像和对应的差分图像，以检查用于训练的所有图像是否包含相似的对象。如果差分图像在应该没有差分的区域中包含了很大的差分，这就可以得出结论，至少有一个训练图像包含了一个坏物体。

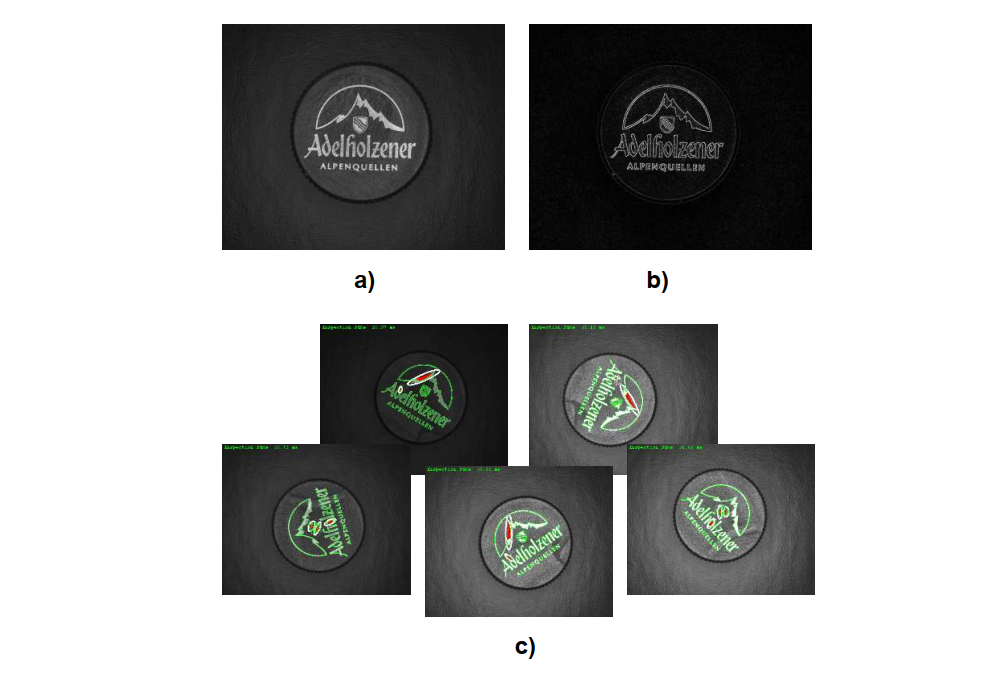
### 1.8清除训练数据

在差分模型准备好之后，当您不再需要使用差分模型进行实际比较时，可以通过使用clear\_train\_data\_variation\_model算子来减少差分模型所需的内存。删除训练数据后，不能进行进一步的训练或应用get\_variation\_model。

### 1.9可视化结果

可视化图像结果，并标记图像中与模型不对应的部分。连接分离的缺陷范围，然后通过select\_shape选择在特定区域范围内的区域来显式提取。

\*这个例子使用差分模型检查瓶盖。这里的困难在于检查过程中光照的变化。  
\*使用单个图像作为理想图像。本文的差分模型是用sobel\_amp算子对理想图像进行预处理得到的。  
\*   
dev\_update\_off ()  
dev\_close\_window ()  
\*   
\* 读取模型图像并初始化可视化  
read\_image (ModelImage, 'cap\_illumination/cap\_illumination\_01')  
get\_image\_size (ModelImage, Width, Height)  
dev\_open\_window (0, 0, Width / 2, Height / 2, 'black', WindowHandle)  
set\_display\_font (WindowHandle, 14, 'mono', 'true', 'false')  
\*   
\* 分割模型区域  
dev\_display (ModelImage)  
binary\_threshold (ModelImage, BrightRegion, 'max\_separability', 'light', UsedThreshold)  
dilation\_circle (BrightRegion, RegionDilation, 1.5)  
connection (RegionDilation, ConnectedRegions)  
select\_shape (ConnectedRegions, SelectedRegions, 'area', 'and', 100, 99999)  
union1 (SelectedRegions, LabelRegion)  
smallest\_circle (LabelRegion, Row, Column, Radius)  
smallest\_rectangle1 (LabelRegion, Row1, Column1, Row2, Column2)  
\* 定义模型区域  
gen\_circle (Region, Row, Column, Radius)  
gen\_rectangle1 (LabelRectangle, Row1, Column1, Row2, Column2)  
intersection (Region, LabelRectangle, RegionIntersection)  
dilation\_circle (RegionIntersection, RegionROI, 5.5)  
reduce\_domain (ModelImage, RegionROI, ImageReduced)  
binary\_threshold (ModelImage, BrightRegion, 'max\_separability', 'light', UsedThreshold)  
dilation\_circle (BrightRegion, RegionDilation, 1.5)  
connection (RegionDilation, ConnectedRegions)  
select\_shape (ConnectedRegions, SelectedRegions, 'area', 'and', 100, 99999)  
union1 (SelectedRegions, LabelRegion)  
smallest\_circle (LabelRegion, Row, Column, Radius)  
smallest\_rectangle1 (LabelRegion, Row1, Column1, Row2, Column2)  
gen\_circle (Region, Row, Column, Radius)  
gen\_rectangle1 (LabelRectangle, Row1, Column1, Row2, Column2)  
intersection (Region, LabelRectangle, RegionIntersection)  
dilation\_circle (RegionIntersection, RegionROI, 5.5)  
reduce\_domain (ModelImage, RegionROI, ImageReduced)  
area\_center (RegionROI, Area, RowModel, ColumnModel)  
dev\_set\_draw ('margin')  
dev\_set\_color ('green')  
dev\_display (RegionROI)  
\*   
\* 创建形状模板匹配模板  
create\_generic\_shape\_model (ModelID)  
set\_generic\_shape\_model\_param (ModelID, 'iso\_scale\_min', 0.98)  
set\_generic\_shape\_model\_param (ModelID, 'iso\_scale\_max', 1.02)  
train\_generic\_shape\_model (ImageReduced, ModelID)  
set\_generic\_shape\_model\_param (ModelID, 'min\_score', 0.55)  
set\_generic\_shape\_model\_param (ModelID, 'num\_matches', 1)  
set\_generic\_shape\_model\_param (ModelID, 'max\_overlap', 0)  
set\_generic\_shape\_model\_param (ModelID, 'pyramid\_level\_lowest', 3)  
\*   
\* 为检测创建一个差分模型.  
create\_variation\_model (Width, Height, 'byte', 'direct', VariationID)  
\* 生成差分图像。  
sobel\_amp (ModelImage, VarImage, 'sum\_abs', 5)  
prepare\_direct\_variation\_model (ModelImage, VarImage, VariationID, [20, 25], [1.6, 1.6])  
\*   
\*   
\* 为了补偿光照的变化，在此过程中确定了变化模型的灰度值范围。  
\* 在该过程中，理想图像被约束到盖子上的感兴趣区域。  
\* 然后，使用binary\_threshold和difference将简化后的图像分割为前景和背景。  
\* 最后，通过灰度强度查询到两个区域灰度值的均值和标准差。  
reduce\_domain (ModelImage, RegionROI, ImageReduced)  
binary\_threshold (ImageReduced, RegionBackground, 'max\_separability', 'dark', UsedThreshold)  
difference (RegionROI, RegionBackground, RegionForeground)  
intensity (RegionForeground, ModelImage, ForegroundGVal, DeviationFG)  
intensity (RegionBackground, ModelImage, BackgroundGVal, DeviationBG)  
\* 执行可视化检测  
\* 图像采集  
list\_files ('cap\_illumination', ['files','follow\_links'], ImageFiles)  
tuple\_regexp\_select (ImageFiles, ['\\.(tif|tiff|gif|bmp|jpg|jpeg|jp2|png|pcx|pgm|ppm|pbm|xwd|ima|hobj)$','ignore\_case'], ImageFiles)  
for Index := 0 to |ImageFiles| - 1 by 1  
 read\_image (rImage, ImageFiles[Index])  
 count\_seconds (StartSeconds)  
 find\_generic\_shape\_model (rImage, ModelID, MatchResultID, NumMatchResult)  
 count\_seconds (EndSeconds)  
 TimeObjSearch := EndSeconds - StartSeconds  
 if (NumMatchResult > 0)  
 \*   
 \* 在这里，在每个图像中搜索对象并按照前面的示例进行对齐。  
 \* 然后，推导出待检测对象的感兴趣区域的灰度值范围，  
 \* 使图像适应差分模型的灰度值范围，再与差分模型进行比较。  
 \* 因此，也可以比较在不同照明下拍摄的图像。  
 \* 使用算子compare\_ext\_variation\_model  
 \* 进行图像比较的模式被设置为light\_dark，  
 \* 因此在RegionDiff中返回太亮和太暗的图像点的单独区域  
 get\_generic\_shape\_model\_result (MatchResultID, 'best', 'row', Row)  
 get\_generic\_shape\_model\_result (MatchResultID, 'best', 'column', Column)  
 get\_generic\_shape\_model\_result (MatchResultID, 'best', 'angle', Angle)  
 vector\_angle\_to\_rigid (Row, Column, Angle, RowModel[0], ColumnModel[0], 0, HomMat2DImage)  
 affine\_trans\_image (rImage, ImageAffineTrans, HomMat2DImage, 'constant', 'false')   
 reduce\_domain (ImageAffineTrans, RegionROI, ImageReduced)  
   
 binary\_threshold (ImageReduced, RegionBackground, 'max\_separability', 'dark', UsedThreshold)  
 difference (RegionROI, RegionBackground, RegionForeground)  
 intensity (RegionForeground, ImageAffineTrans, ForegroundGVal\_2, DeviationFG)  
 intensity (RegionBackground, ImageAffineTrans, BackgroundGVal\_2, DeviationBG)   
   
 \* 将图像缩放到模型的灰度值范围。  
 Mult := (ForegroundGVal - BackgroundGVal) / (ForegroundGVal\_2 - BackgroundGVal\_2)  
 Add := ForegroundGVal - Mult \* ForegroundGVal\_2  
 reduce\_domain (ImageAffineTrans, RegionROI, ImageReduced)  
 scale\_image (ImageReduced, ImageScaled, Mult, Add)  
 \*   
 \* 将差分模型应用于检测。.  
 count\_seconds (StartSeconds)  
 compare\_ext\_variation\_model (ImageScaled, RegionDiff, VariationID, 'light\_dark')  
 count\_seconds (EndSeconds)  
 \* 分析明暗缺陷区域。  
 dev\_display (rImage)  
 count\_obj (RegionDiff, NumberRegionDiff)  
 for I := 1 to NumberRegionDiff by 1  
 select\_obj (RegionDiff, RegionDiffSelected, I)  
 opening\_circle (RegionDiffSelected, RegionOpening, 2.5)  
 connection (RegionOpening, ConnectedRegions)  
 select\_shape (ConnectedRegions, DefectRegions, 'height', 'and', 20, 99999)  
 \*   
 \* 将区域转换到初始位置的输入图像  
 hom\_mat2d\_invert (HomMat2DImage, HomMat2DInvert)  
 affine\_trans\_region (DefectRegions, DefectRegionsTrans, HomMat2DInvert, 'nearest\_neighbor')  
 closing\_circle (DefectRegionsTrans, DefectRegionsClosing, 1.5)  
 \*   
 \* 使用不同颜色显示明暗缺陷  
 dev\_set\_line\_width (2)  
 dev\_set\_draw ('fill')  
 if (I == 1)  
 dev\_set\_color ('red')  
 else  
 dev\_set\_color ('orange')  
 endif  
 dev\_display (DefectRegionsClosing)  
 \*   
 \* 增强缺陷区域。  
 dev\_set\_color ('magenta')  
 dev\_set\_draw ('margin')  
 union1 (DefectRegionsClosing, RegionUnion)  
 closing\_circle (RegionUnion, RegionClosing, 10)  
 connection (RegionClosing, DefectRegionEnlarged)  
 elliptic\_axis (DefectRegionEnlarged, Ra, Rb, Phi)  
 if (|Phi|)  
 area\_center (DefectRegionEnlarged, Area, RowEllipse, ColumnEllipse)  
 gen\_ellipse (Ellipse, RowEllipse, ColumnEllipse, Phi, Ra \* 2, Rb \* 2)  
 dev\_display (Ellipse)  
 endif  
 endfor  
 dev\_set\_line\_width (1)  
 dev\_set\_color ('green')  
 get\_generic\_shape\_model\_result\_object (ModelContoursFound, MatchResultID, 'best', 'contours')  
 dev\_display (ModelContoursFound)  
 \*   
 TimePrintInspect := EndSeconds - StartSeconds  
 disp\_message (WindowHandle, 'Inspection Time: ' + ((TimeObjSearch + TimePrintInspect) \* 1000.0)$'.02' + ' ms', 'window', 12, 12, 'black', 'true')  
 else  
 disp\_message (WindowHandle, 'No shape model found', 'window', 12, 12, 'red', 'true')  
 endif  
\*   
 if (I < 9)  
 disp\_continue\_message (WindowHandle, 'black', 'true')  
 stop ()  
 endif  
endfor

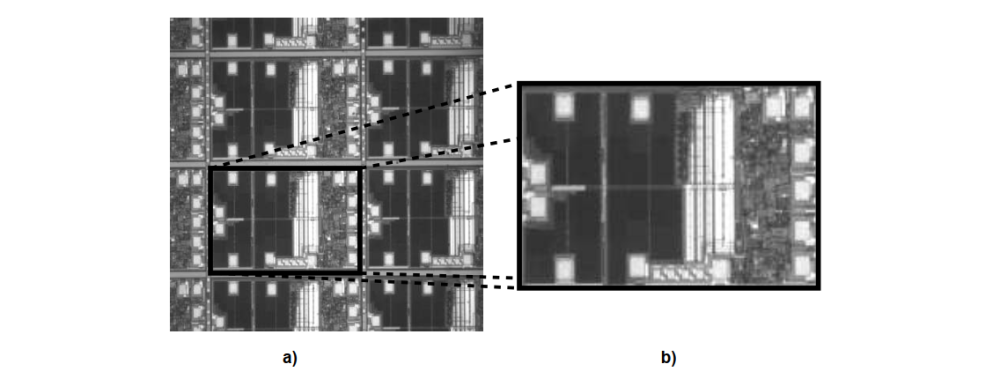


瓶盖检验模型:a)理想图像;B)变差图像;C)不同光照条件下的图像对比结果。

### 二、纹理分析

纹理分析的概念是基于对物体结构特征的利用。深入理解纹理分析的合理应用，首先要了解纹理本身。纹理指的是物体表面的结构特征，这些特征在图像中具体为灰度值的变化。纹理的规律性表现形式从绝对规则到不规则不等，规则纹理的基础是理想的网格结构，而不规则纹理或统计结构的基础是随机函数。

纹理由texel组成，texel是纹理元素的缩写形式，很容易地在规则纹理中识别。不规则纹理也有反复出现的元素，但是，这类texel元素更难识别。当使用不规则纹理时，必要时必须对纹理的度量进行近似。texel是纹理的最小循环结构。



极规则纹理的纹理元素:a)原始图像;b)texel

纹理分析适用于所有由于灰度值结构过于复杂而无法简单求解的任务。它可以解决•某种纹理位于何处?•它属于哪种纹理?

第一个任务通常是通过使用增强或抑制某些纹理的过滤器过滤图像来解决，然后通过标准斑点分析或对每个像素进行分类，将过滤后的图像分割成具有相似纹理的区域。第二个任务通常通过计算整个图像或单个区域的纹理特征来解决。然后根据这些特征对图像或区域进行分类。分类可以使用纹理预测模型来分析纹理，或者通过纹理检测模型来自动提取纹理特征。

一个简单的纹理分析包括以下步骤:

### 2.1获取图像

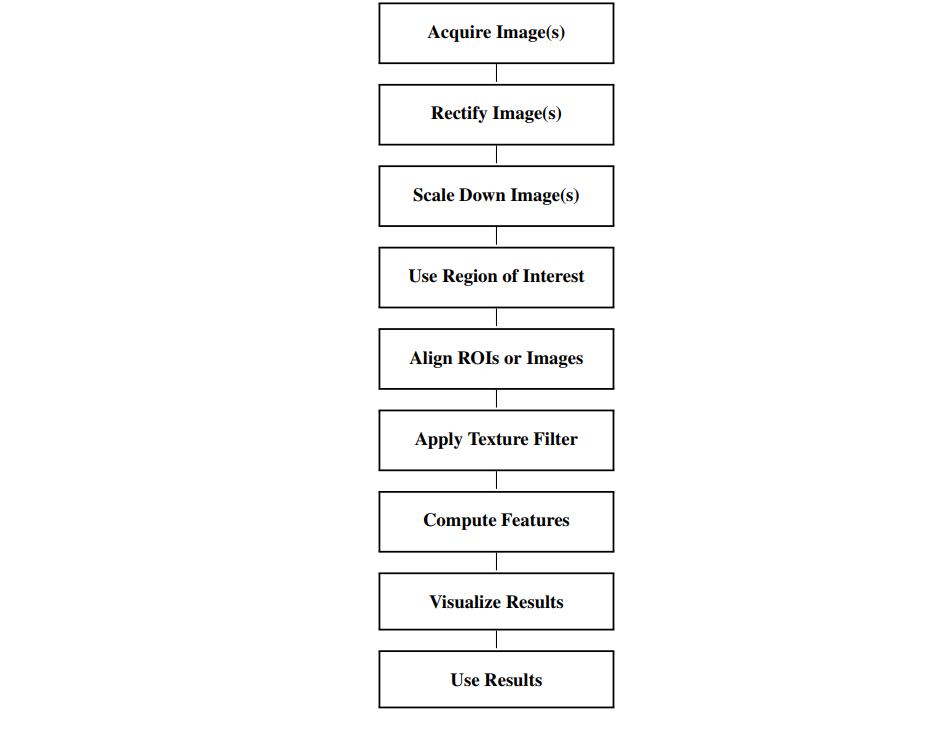
获取源纹理图像

### 2.2缩小图像

当texel和它的最小子结构都比较大时，可以使用zoom\_image\_factor算子缩小图像以加快纹理分析。还可以通过多次调用zoom\_image\_factor来创建缩小图像的金字塔，然后在图像金字塔上应用纹理过滤或特征计算。

### 2.3使用感兴趣区域

创建ROI，缩小检测图像域，再对其进行纹理分析。这将减少处理时间，特别是在计算纹理特征时，ROI可以是先前分割得到的结果。



### 2.4对齐roi或图像

如果相机从一个角度观察物体，则需要校正图像以消除镜头畸变和透视畸变。如果要处理一个规则的纹理，可以应用旋转图像或者仿射变换使纹理与图像轴对齐。这样，纹理过滤和纹理特征都可以更方便地应用或计算。

### 2.5应用纹理滤波

算子texture\_laws提供了一系列纹理过滤器。在调用texture\_laws之后，将mean\_image的核大小设置为texel大小，再平滑图像。对于某些具有简单纹理的图像，算子dyn\_threshold是常规过滤器的一个很好的解决方案，因为它更快。在使用此算子之前，必须将mean\_image的核大小设置为对象大小的两倍。

通过应用Laws Filter，根据图像的空间频率(即texels的大小)增强或抑制纹理结构。通过将输入图像与3x3、5x5或7x7像素的特殊滤镜Mask进行卷积来完成的。对于大多数滤波，产生的灰度值必须通过移位来修改。如果使用合适的滤波，这使得输出图像中的不同纹理更具可比性。滤波的名称由所使用的两个向量的字母组成，其中第一个字母指定列方向的卷积，而第二个字母指定行方向的卷积。可以选择“l”、“e”、“s”、“r”、“w”、“o”，其中“l”表示低频，“o”表示高频。因此，检测低频到高频的第二个字母分别是“l”、“e”、“s”、“r”、“w”和“o”。texture\_laws对图像应用纹理变换。过将输入图像与特殊的Mask进行卷积来完成的。过滤器有:

9个不同的3×3矩阵可从以下三个向量获得:

l = [ 1，2 ，1] e = [−1 ，0 ，1] s = [−1 ，2 ，−1]

25个不同的5×5矩阵可从以下五个向量中获得:

l = [ 1 ，4 ，6， 4 ，1] e = [−1 ，−2 ，0 ，2 ，1] s = [−1 ，0 ，2 ，0 ，−1] w = [−1， 2 ，0 ，−2 1，] r = [ 1 ，−4 ，6 ，−4 ，1] 49个不同的7×7矩阵可从以下七个向量中获得:

l = [ 1 ，6 ，15 ，20 ，15 ，6 ，1] e = [−1， −4， −5， 0， 5， 4， 1]

s = [−1， −2， 1， 4 ，1 ，−2 ，−1] w = [，−1， 0， 3， 0， −3， 0 ，1]

r = [ 1 ，−2， −1 ，4 ，−1 ，−2， −1] u = [ 1， −4， 5 ，0 ，−5 ，4， −1]

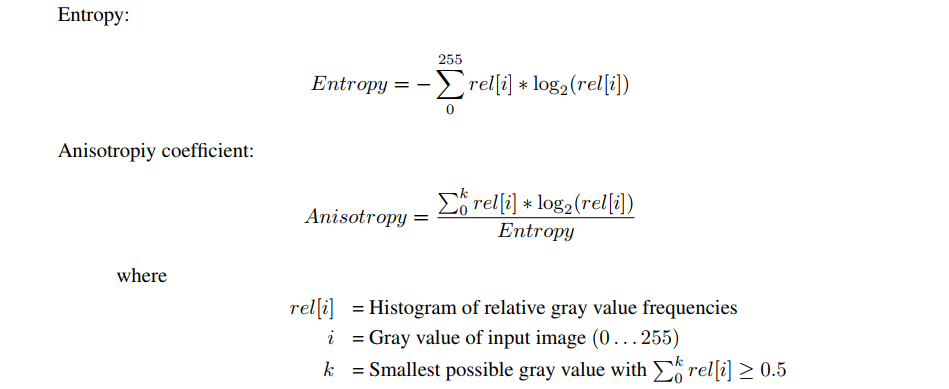
o = [−1, 6, −15, 20, −15, 6, −1]

滤波器的名称有“level,” “edge,” “spot,” “wave,” “ripple,” “undulation,” and“oscillation.”。对于大多数过滤器，产生的灰度值必须通过Shift来修改。

### 2.6计算特征

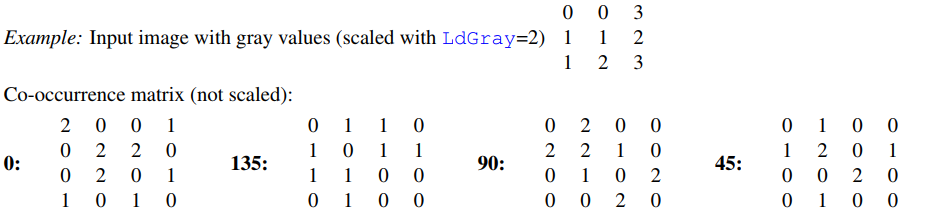
如果使用不同方向对应的基于灰度共生矩阵，使用gen\_cooc\_matrix生成矩阵，再使用cooc\_feature\_matrix，计算特征会更高效。除了基于共生矩阵的纹理特征外，还可以使用图像的熵和各向异性，它们可以由算子entropy\_gray计算。在进行滤波后，使用阈值计算图像中特征的强度。除此之外还可以计算图像梯度的分布，首先使用sobel\_amp过滤图像，然后再使用gray\_histo\_abs计算灰度直方图，显示灰度值的变化情况。

熵和各向异性（Entropy and Anisotropy ）描述了与像素位置无关的图像的灰度值分布。算子entropy\_gray创建输入图像中灰度值相对频率的直方图，并根据这些频率计算区域中每个区域的熵和各向异性系数，公式如下:



基于灰度共生矩阵（Cooccurrence Matrix ）描述了像素的灰度值与其相邻像素的灰度值之间的关系。矩阵元素包含两个灰度值相邻出现的概率，缓慢的灰度值变化由沿主轴对角线的高值表示。

算子gen\_cooc\_matrix从输入区域确定灰度值（i, j)在某一方向(0,45,90,135度)相邻的频率，并将该数字存储在灰度共生矩阵中，最后根据要区分的灰度值数目(LdGray)。对矩阵进行缩放。



cooc\_feature\_image(Regions, Image ,LdGray, Direction ,Energy, Correlation, Homogeneity, Contrast)  
  
参数列表：  
Regions（in） //要检查的区域。  
Image （in） //灰度图像。  
LdGray(in） //要区分的灰度值的数量。（默认6）  
Direction （in） //矩阵的计算方向('0','45','90','130','mean‘）  
Energy（out） //能量  
Correlation（out）//相关性  
Homogeneity（out) //局部均匀性（熵）  
Contrast（out） //对比度（反差）

输出参数详解：

能量（Energy）：是对图像纹理的灰度变化稳定程度的度量,反应了图像灰度分布均匀程度和纹理粗细度。能量越大，表示灰度变化比较稳定，反映了纹理变化的均匀程度。对于灰度图来说，能量低说明灰度值低，对于彩色图来说，能量低说明光强低。

相关性（Correlation）：表示纹理在行或者列方向的相似程度。相关性越大，相似性越高，灰度值变化极端的图像比灰度值非常相似的纹理图像的相关值更低。

（熵）局部均匀性（Homogeneity）：反映图像局部纹理的变化量（即复杂程度），熵值越大图像越复杂，即图像中的大部分区域具有相同的灰度值，则均匀性高。

(反差）对比度（Contrast）：表示矩阵的值的差异程度，也间接表现了图像的局部灰度变化幅度。反差值越大，图像中的纹理深浅越明显，表示图像越清晰；反之，则表示图像越模糊。

### 2.7可视化结果

为了检测纹理分析的结果，显示图像、区域和特征。

### 2.8使用图像

过滤后的图像通常用作分割任务的输入

\* 本例的任务是使用dyn\_threshold算子检测网格中的缺陷。  
\* 通过这种方式，可以使用算子来查找与图像其他部分不同的纹理。  
\* Dyn\_threshold是一个易于处理的算子。与像mean\_image这样的平滑过滤器一起，  
\* 它可以提取局部与其邻域不同的对象。注意，算子dyn\_threshold  
\* 只能用于检测纹理中的缺陷比纹理的其余部分更暗或更亮  
read\_image (Image, 'plastic\_mesh/plastic\_mesh\_01')  
dev\_close\_window ()  
get\_image\_size (Image, Width, Height)  
dev\_open\_window\_fit\_image (Image, 0, 0, Width, Height, WindowHandle)  
set\_display\_font (WindowHandle, 18, 'mono', 'true', 'false')  
dev\_set\_draw ('margin')  
dev\_set\_line\_width (3)  
\* 首先，使用HALCON算子mean\_image对图像进行平滑处理。  
\* mash的大小决定了提取对象的大小:mask大小越大，检测的区域就越大。  
\* mask的大小应该是要提取的物体直径的两倍左右。  
\* 随后，执行dyn\_threshold并查找连接的区域。  
\* 操作符select\_shape的参数area可以找到大小不同的区域。  
\* 最后对发现的错误进行计数和显示。  
list\_files ('plastic\_mesh', ['files','follow\_links'], ImageFiles)  
tuple\_regexp\_select (ImageFiles, ['\\.(tif|tiff|gif|bmp|jpg|jpeg|jp2|png|pcx|pgm|ppm|pbm|xwd|ima|hobj)$','ignore\_case'], ImageFiles)  
for Index := 0 to |ImageFiles| - 1 by 1  
 read\_image (Image, ImageFiles[Index])  
 mean\_image (Image, ImageMean, 49, 49)  
 dyn\_threshold (Image, ImageMean, RegionDynThresh, 5, 'dark')  
 connection (RegionDynThresh, ConnectedRegions)  
 select\_shape (ConnectedRegions, ErrorRegions, 'area', 'and', 500, 99999)  
 count\_obj (ErrorRegions, NumErrors)  
 dev\_display (Image)  
 dev\_set\_color ('red')  
 dev\_display (ErrorRegions)  
 \* 如果错误数超过零，则提示“Mesh not OK”  
 \* 否则，网格是完好无损的  
 \* 显示“Mesh OK”。  
 if (NumErrors > 0)  
 disp\_message (WindowHandle, 'Mesh not OK', 'window', 24, 12, 'black', 'true')  
 else  
 disp\_message (WindowHandle, 'Mesh OK', 'window', 24, 12, 'black', 'true')  
 endif  
 \* 如果要检查的图像的序列号  
 \* 低于14，要求按“Run”继续出现。  
 \* 如果读取最后一个图像，按“Run”将清除SVM。  
 if (Index < 14)  
 disp\_continue\_message (WindowHandle, 'black', 'true')  
 stop ()  
 endif  
endfor99999)  
count\_obj (ErrorRegions, NumErrors)

## 三、分类器

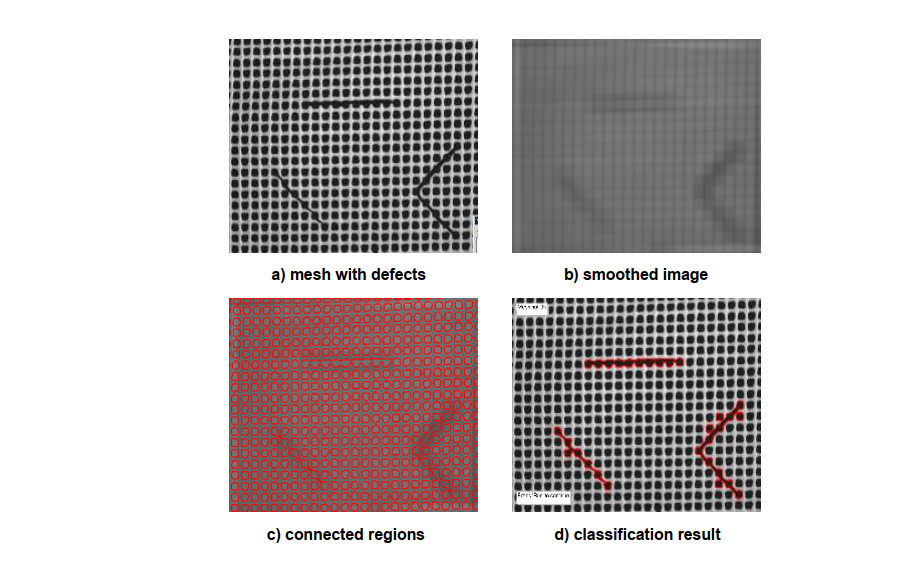
机器视觉中的Classification分类器是一种用于将输入图像或图像的特征向量分配到预定义的类别中的算法或模型。它是机器学习和模式识别领域中常用的一种技术，用于解决图像分类、目标识别和物体检测等问题。

Classification分类器的目标是学习一个决策边界，将不同类别的样本分开。它通过从训练数据中学习类别之间的差异和相似性，来进行分类。分类器可以基于不同的算法，如支持向量机（SVM）、决策树、随机森林、神经网络等。

在机器视觉中，Classification分类器通常使用图像的特征向量作为输入。这些特征向量可以是手工设计的，如颜色直方图、纹理特征、形状特征等。处理非线性可分类的分类方法使用支持向量机。将特征空间转化为高维空间，使特征成为线性可分的。然后，使用线性分类器对特征向量进行分类。

\* 这个示例程序向您展示了如何使用  
\* SVM分类器执行纹理检查任务  
dev\_update\_off ()  
\*   
read\_image (Image, 'plastic\_mesh/plastic\_mesh\_01')  
get\_image\_size (Image, Width, Height)  
dev\_close\_window ()  
dev\_open\_window (0, 0, Width, Height, 'black', WindowHandle)  
dev\_set\_color ('red')  
set\_display\_font (WindowHandle, 16, 'mono', 'true', 'false')  
\* 用于分类的纹理过滤器将返回图像上的工件  
\* 边界，因为要检查的塑料网格的图像没有  
\* 包含一个整数的网格单元。因为这会导致错误  
\* 在图像边缘检测到错误时，必须排除靠近的区域  
\* 从图像边界进行训练和分类。这是通过以下矩形完成的  
\* 该图像随后被按2倍的比例缩小。  
gen\_rectangle1 (Rectangle, 10, 10, Height / 2 - 11, Width / 2 - 11)  
\* 创建SVM分类器  
create\_class\_svm (5, 'rbf', 0.01, 0.0005, 1, 'novelty-detection', 'normalization', 5, SVMHandle)  
\* 训练是基于五张不包含错误的图像.  
for J := 1 to 5 by 1  
 read\_image (Image, 'plastic\_mesh/plastic\_mesh\_' + J$'02')  
 \* 缩小图像，网格的分辨率非常高  
 zoom\_image\_factor (Image, ImageZoomed, 0.5, 0.5, 'constant')  
 dev\_display (ImageZoomed)  
 disp\_message (WindowHandle, 'Adding training samples...', 'window', 12, 12, 'black', 'true')  
 \* 生成纹理图像  
 gen\_texture\_image (ImageZoomed, ImageTexture)  
 \* 将样本添加到分类器中  
 add\_samples\_image\_class\_svm (ImageTexture, Rectangle, SVMHandle)  
endfor  
dev\_display (ImageZoomed)  
disp\_message (WindowHandle, 'Training SVM...', 'window', 12, 12, 'black', 'true')  
\* 训练支持向量机。这将生成相当多的支持向量。  
train\_class\_svm (SVMHandle, 0.001, 'default')  
\* 为了提高分类速度，通过reduce\_class\_svm减少支持向量的数量，得到一个名为svmhandlerreduced的新分类器  
reduce\_class\_svm (SVMHandle, 'bottom\_up', 2, 0.001, SVMHandleReduced)  
\* 检测塑料网格中的错误。  
dev\_set\_draw ('margin')  
dev\_set\_line\_width (3)  
for J := 1 to 14 by 1  
 \* 在循环中读取图像。每个图像再次缩放并转换为纹理图像。  
 \* 对ROI使用算子classify\_image\_class\_svm，将结果存储在SVMHandleReduced  
 read\_image (Image, 'plastic\_mesh/plastic\_mesh\_' + J$'02')  
 zoom\_image\_factor (Image, ImageZoomed, 0.5, 0.5, 'constant')  
 dev\_display (ImageZoomed)  
 dev\_set\_color ('white')  
 dev\_display (Rectangle)  
 gen\_texture\_image (ImageZoomed, ImageTexture)  
 reduce\_domain (ImageTexture, Rectangle, ImageTextureReduced)  
 \* 使用SVM进行检测  
 classify\_image\_class\_svm (ImageTextureReduced, Errors, SVMHandleReduced)  
 \* 由在图像中可视化分类返回的区域错误。应用形态学算子平滑误差区域边界，  
 \* 提取连通分量。选择一定大小的区域，将纹理检测结果可视化为文本。  
 \* 如果所选区域的数量(NumErrors)为0，则网格的纹理被分类为良好。否则，网格被分类为错误。  
 opening\_circle (Errors, ErrorsOpening, 3.5)  
 closing\_circle (ErrorsOpening, ErrorsClosing, 10.5)  
 connection (ErrorsClosing, ErrorsConnected)  
 select\_shape (ErrorsConnected, FinalErrors, 'area', 'and', 300, 1000000)  
 count\_obj (FinalErrors, NumErrors)  
 dev\_set\_color ('red')  
 dev\_display (FinalErrors)  
 if (NumErrors > 0)  
 disp\_message (WindowHandle, 'Mesh not OK', 'window', 12, 12, 'red', 'true')  
 else  
 disp\_message (WindowHandle, 'Mesh OK', 'window', 12, 12, 'forest green', 'true')  
 endif  
 if (J < 14)  
 disp\_continue\_message (WindowHandle, 'black', 'true')  
 endif  
 stop ()  
endfor

无缺陷网格b)平滑图像c)连通区域d)分类:网格OK



b)平滑图像c)连通区域d)分类:网格不正确。

这个例子的目的是根据表面纹理对不同类型的木材进行分类。

\* 根据木材的表面纹理划分木材的种类  
file\_exists ('classify\_wood.gmc', FileExists)  
\* 该程序默认使用存储的分类器。然而，如果你，  
\* 如果要进行训练，将USE\_STORED\_CLASSIFIER设置为0。  
\* 如果无法找到分类器，则使用USE\_STORED\_CLASSIFIER  
\* 自动设置为0。  
if (FileExists)  
 USE\_STORED\_CLASSIFIER := 1  
else  
 USE\_STORED\_CLASSIFIER := 0  
endif  
ImagePath := '/wood/'  
ReadOK := false  
ReadPath := 'D:/desk/缺陷检测/wood/'  
read\_image (Image, ImagePath + 'apple/apple\_01')  
get\_image\_size (Image, Width, Height)  
dev\_close\_window ()  
dev\_open\_window\_fit\_image (Image, 0, 0, Width, Height, WindowID)  
set\_display\_font (WindowID, 16, 'mono', 'true', 'false')  
dev\_display (Image)  
dev\_update\_off ()  
\* 现在指定了不同的木材类别。  
Classes := ['apple', 'beech', 'cherry', 'maple', 'oak', 'walnut']  
\* 然后，执行训练，这意味着为每种类型的木材读取和训练几个图像。  
\* 因此，SVM学习不同的木材纹理，并将每个新的木材图像与现有图像进行比较。它检查哪个类最相似，并显示分配的类。  
\* 对于每个图像，在一个过程中计算特征，然后传递给操作符classify\_image\_class\_mlp  
if (USE\_STORED\_CLASSIFIER == 1)  
 read\_class\_mlp ('classify\_wood.gmc', MLPHandle)  
 NumClasses := |Classes|  
else  
 \* 函数gen\_features调用实际的特征提取过程，然后对图像进行下采样，并再次使用较小的图像调用第二个过程  
 gen\_features (Image, FeatureVector)  
 \* 过程gen\_sobel\_features计算多个纹理特征。首先计算0度方向和90度方向的相干矩阵  
 NumFeatures := |FeatureVector|  
 NumClasses := |Classes|  
 NumHidden := 15  
 create\_class\_mlp (NumFeatures, NumHidden, NumClasses, 'softmax', 'normalization', 10, 42, MLPHandle)  
 for CorrectClassID := 0 to NumClasses - 1 by 1  
 list\_files (ReadPath + Classes[CorrectClassID], 'files', Files)  
 for k := 0 to |Files| - 1 by 2  
 read\_image (Image, Files[k])  
 dev\_display (Image)  
 gen\_features (Image, FeatureVector)  
 add\_sample\_class\_mlp (MLPHandle, FeatureVector, CorrectClassID)  
 endfor  
 endfor  
 train\_class\_mlp (MLPHandle, 200, 1, 0.0001, Error, ErrorLog)  
 write\_class\_mlp (MLPHandle, 'classify\_wood.gmc')  
 disp\_message (WindowID, 'Training of wood textures completed\nPress \'Run\' to continue', 'window', 12, 12, 'black', 'true')  
 stop ()  
endif  
Errors := 0  
Count := 0  
for CorrectClassID := 0 to NumClasses - 1 by 1  
 list\_files (ReadPath + Classes[CorrectClassID], 'files', Files)  
 for k := 0 to |Files| - 1 by 1  
 Count := Count + 1  
 read\_image (Image, Files[k])  
 gen\_features (Image, FeatureVector)  
 classify\_class\_mlp (MLPHandle, FeatureVector, 2, FoundClassIDs, Confidence)  
 dev\_display (Image)  
 dev\_set\_color ('blue')  
 disp\_message (WindowID, 'correct class: ' + Classes[CorrectClassID], 'window', 12, 12, 'black', 'true')  
 if (CorrectClassID == FoundClassIDs[0])  
 disp\_message (WindowID, 'found class: ' + Classes[FoundClassIDs[0]], 'window', 42, 12, 'forest green', 'true')  
 else  
 Errors := Errors + 1  
 disp\_message (WindowID, 'found class: ' + Classes[FoundClassIDs[0]], 'window', 42, 12, 'red', 'true')  
 disp\_continue\_message (WindowID, 'black', 'true')  
 stop ()  
 endif  
 wait\_seconds (0.1)  
 endfor  
endfor  
ErrorRate := real(Errors) / Count \* 100.0  
disp\_message (WindowID, 'ErrorRate = ' + ErrorRate + '%', 'window', 72, 12, 'black', 'true')  
disp\_end\_of\_program\_message (WindowID, 'black', 'true')  
clear\_class\_mlp (MLPHandle)

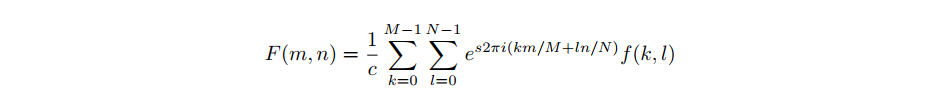
## 四、快速傅里叶变换(FFT)

快速傅里叶变换，简称FFT，是纹理分析的另一种选择。这种方法利用了纹理在频域中有不同频率的表现形式。HALCON的FFT精确地确定了图像中每种纹理频率的数量。它能够构建任何类型的线性滤波器。使用HALCON的FFT可以增强特殊纹理的不同结构。卷积可以通过将图像和滤波器转换到频域，将两个结果相乘，并将结果转换回空间域来执行。因此，使用FFT的两个原因是:

•在傅里叶域中可以应用较大的滤波器尺寸滤波器，因为FFT对于任何Mask大小都需要相同的时间。

•可以定制滤波，从图像消除或增强特定的频率，这是一个很大的优势。

傅里叶变换的一般定义如下:



对于前向变换(即进入频域的变换)，指数中的符号s应该设为1还是-1，意见不一。对于归一化因子c的大小也存在分歧。对于前向变换，有时将其设置为1，有时设置为MN，有时(在单一FFT的情况下)设置为√MN。

Fft\_generic允许单独选择这些选项。通过“Direction ”可以选择FFT的逻辑方向，可能值为' to\_freq '和' from\_freq '。参数Exponent用于确定指数的符号，可以设置为1或-1。规范化因子可以用Norm设置，并且可以采用值' none '， ' sqrt '和' n '。Mode参数决定FFT的DC term位置，它可以设置为' dc\_center '或' dc\_edge '。在任何情况下，都必须保证参数使用的一致性。

这意味着用于前向和后向变换的归一化因子在相乘时必须产生MN，指数必须具有相反的符号，并且两个变换的Mode必须相等。

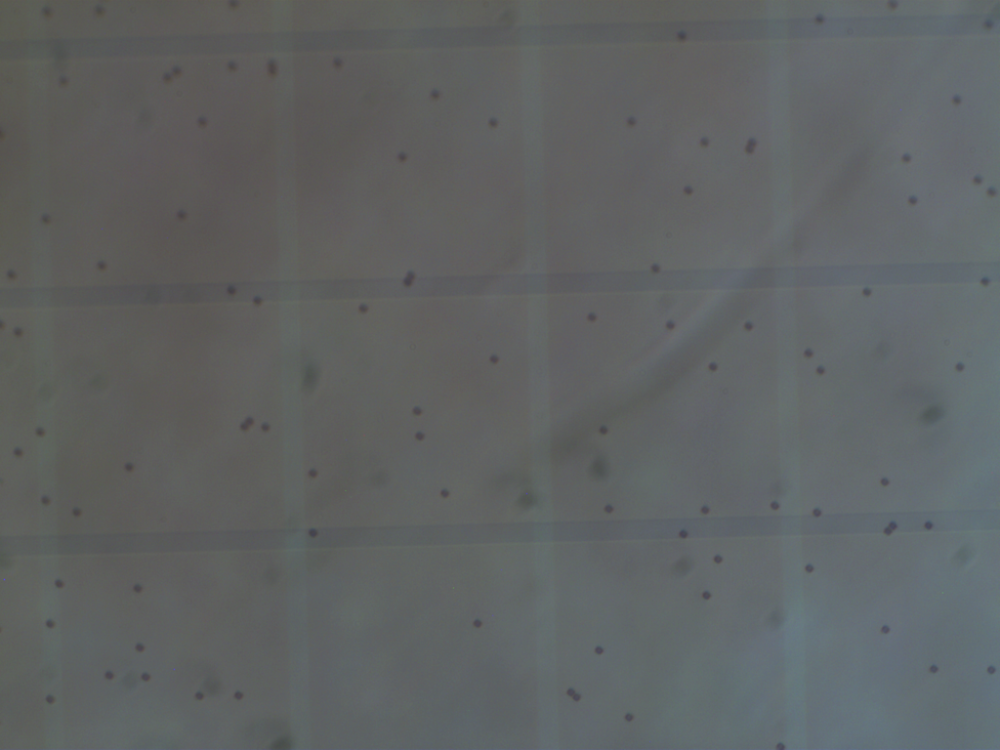
例如，' (to\_freq，-1, n, dc\_edge) '用于正向变换，' (from\_freq, 1, none, dc\_edge) '用于反向变换。在这种情况下，FFT可以被解释为具有三角基函数的插值。另一种可能的组合是' (to\_freq，-1, sqrt, dc\_center) '和' (from\_freq, 1, sqrt, dc\_center) '。ResultType参数可用于指定反向变换的结果图像类型(Direction = ' from\_freq ')。在正向变换(Direction = ' to\_freq ')中，ResultType必须设置为' complex '。

1，脏污检测（低通滤波，差分，线提取）

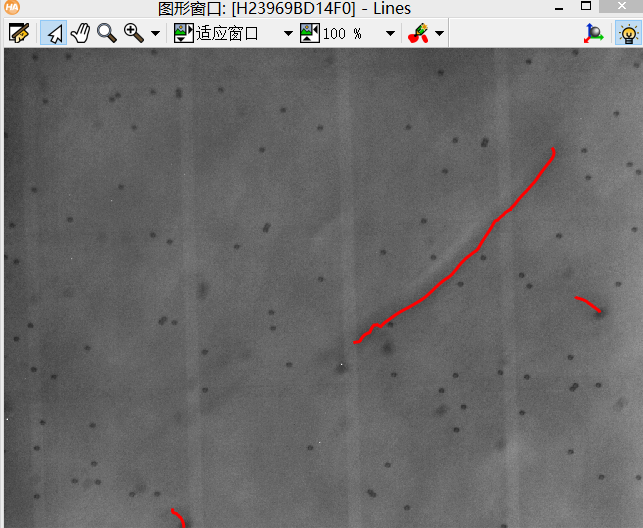
如下图，在塑料薄膜上有一些线条型的脏污，在空间域中向提取脏污的区域是比较困难的(方格会影响空间域的二值化），所以我们就想到了在频域中去处理它。

思路：

* 用gen\_gauss\_filter算子在频域中生成高斯滤波器
* 读入图片，将B空间转到频域
* 用高斯滤波器（低通滤波)进行滤波（即得到背景图像）
* 差分（原图——背景图），锐化图像
* 用lines\_gauss提取脏污线痕



\* 这个例子展示了如何检测mura缺陷  
\*   
dev\_close\_window ()  
dev\_update\_off ()  
read\_image (Image, 'lcd/mura\_defects\_blur\_01')  
get\_image\_size (Image, Width, Height)  
dev\_open\_window\_fit\_size (0, 0, Width, Height, 640, 480, WindowHandle)  
set\_display\_font (WindowHandle, 14, 'mono', 'true', 'false')  
dev\_set\_draw ('margin')  
dev\_set\_line\_width (3)  
dev\_set\_color ('red')  
ScaleFactor := 0.4  
calculate\_lines\_gauss\_parameters (17, [25, 3], Sigma, Low, High)  
\* Image Acquisition 01: Code generated by Image Acquisition 01  
list\_files ('D:/desk/缺陷检测/lcd', ['files','follow\_links'], ImageFiles)  
tuple\_regexp\_select (ImageFiles, ['\\.(tif|tiff|gif|bmp|jpg|jpeg|jp2|png|pcx|pgm|ppm|pbm|xwd|ima|hobj)$','ignore\_case'], ImageFiles)  
for Index := 0 to |ImageFiles| - 1 by 1  
 read\_image (Image, ImageFiles[Index])  
 decompose3 (Image, R, G, B)  
 \* 计算图像的实值快速傅里叶变换。  
 rft\_generic (B, ImageFFT, 'to\_freq', 'none', 'complex', Width)  
 gen\_gauss\_filter (ImageGauss, 100, 100, 0, 'n', 'rft', Width, Height)  
 convol\_fft (ImageFFT, ImageGauss, ImageConvol)  
 rft\_generic (ImageConvol, ImageFFT1, 'from\_freq', 'none', 'byte', Width)  
 sub\_image (B, ImageFFT1, ImageSub, 2, 100)  
 \* perform the actual inspection  
 zoom\_image\_factor (ImageSub, ImageZoomed, ScaleFactor, ScaleFactor, 'constant')  
 \* 提取图像上的线条，提取的结果属于亚像素精度的XLD轮廓  
 get\_domain (ImageZoomed, Domain)  
 erosion\_rectangle1 (Domain, RegionErosion, 7, 7)  
 reduce\_domain (ImageZoomed, RegionErosion, ImageReduced)  
 lines\_gauss (ImageReduced, Lines, Sigma, Low, High, 'dark', 'true', 'gaussian', 'true')  
 hom\_mat2d\_identity (HomMat2DIdentity)  
 hom\_mat2d\_scale\_local (HomMat2DIdentity, 1 / ScaleFactor, 1 / ScaleFactor, HomMat2DScale)  
 affine\_trans\_contour\_xld (Lines, Defects, HomMat2DScale)  
 \*   
 dev\_display (Image)  
 dev\_display (Defects)  
 if (Index < 3)  
 disp\_continue\_message (WindowHandle, 'black', 'true')  
 stop ()  
 endif  
endfor



相关API参数：

lines\_gauss（提取图像上的线条，提取的结果属于亚像素精度的XLD轮廓）

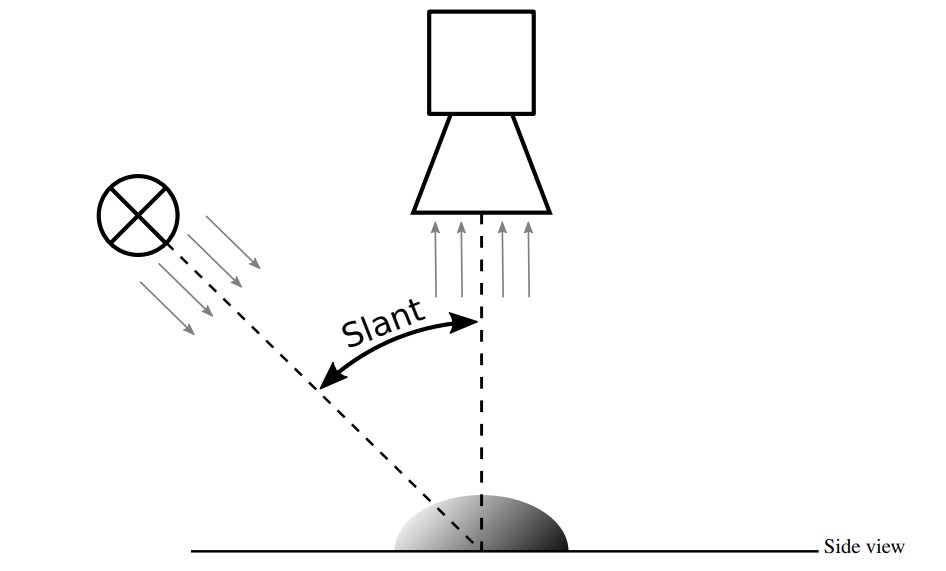
lines\_gauss(Image , Lines ,Sigma, Low, High, LightDark, ExtractWidth, LineModel, CompleteJunctions )  
Image (input\_object) //输入图像  
Lines (output\_object) //检测线条（XLD）  
Sigma (input\_control) //高斯滤波值  
Low (input\_control) //滞后阈值分割的低阈值  
High (input\_control) //滞后阈值分割的高阈值  
LightDark (input\_control)//提取线条的类型，暗色还是亮色，(’dark’, ‘light’)  
ExtractWidth (input\_control) //是否提取线宽（‘false’,‘true’）  
LineModel (input\_control) //用来调整线条位置和宽度的线模型（‘bar-shaped’, ‘gaussian’, ‘none’, ‘parabolic’）  
CompleteJunctions (input\_control) //在断连的部分是否添加节点使线条连续（‘false’, ‘true’）

## 五、光度立体

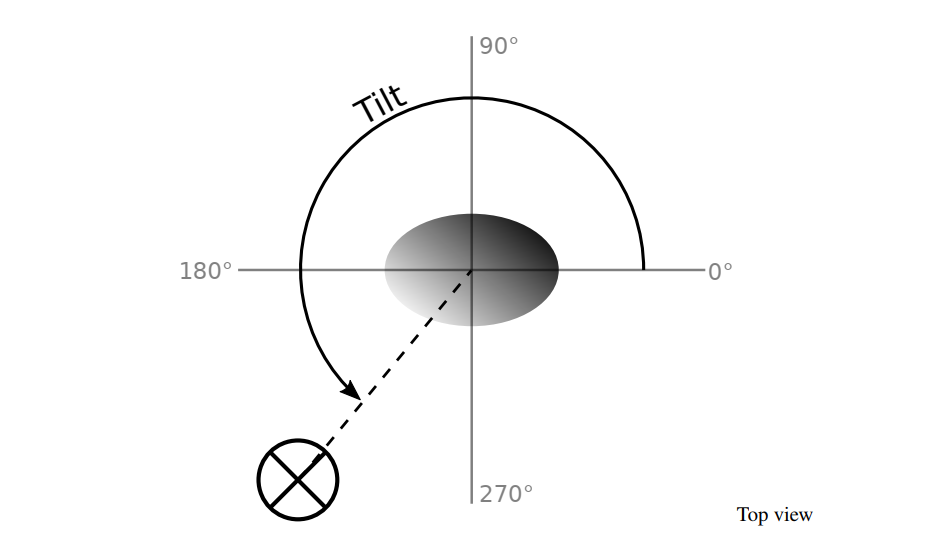
在HALCON中，使用增强的光度立体视觉方法，加强检测三维表面。利用阴影可方便快速的检测物体表面的缺口或凹痕。 使用光度立体视觉方法可在复杂图像中轻松找到表面缺陷 。

* 适用场景：适用于检测金属物料上面的凹凸特征。
* 函数原理：1.通过photometric\_stereo算子获得表面梯度图像，该算子可以得到表面梯度图像和反照率图像。需要输入多张从不同角度照明所得到的图像。 2.通过derivate\_vector\_field算子获得高斯（平均）曲率图像，该算子中需要输入表面梯度图像。

photometric\_stereo基于Woodham算法，因此一方面假设相机执行正交投影。也就是说，必须使用远心镜头或长焦距镜头。另一方面，它假设每个光源发出平行且均匀的光束。使用强度均匀的远心照明光源，或者使用远点光源。此外，物体必须具有朗伯反射特性，即它必须以漫反射的方式反射入射光。具有镜面反射特性的物体或物体的区域(即镜像或光滑表面)不能正确处理，从而导致错误的结果

带有远心镜头的相机必须垂直放置，即垂直于要重建的场景。在获取图像的过程中，相机相对于场景的方向不能改变。每张图像要指定照明方向，照明方向必须指定参数Slants和Tilts中的角度，它们描述了与场景相关的照明方向。

倾斜角度（Slant）在物体平面或与其平行的任何平面内测量。它描述了从图像中心向右指向的方向和投射到平面上的光的方向之间的夹角。也就是说，在观看图像(或相应场景)时，倾斜角度为0表示光线来自右侧，倾斜角度为90表示光线来自顶部，倾斜角度为180表示光线来自左侧，以此类推。



这个角度是以图像为准的，比如光从图像右侧打过来，角度就是0°，从上面打过来，角度是90°，从左面打过来，角度是180°，下边打过来是270°。

正常情况下一般都是至少采集三张不同方向打光的图。物体的三维几何形状通常会导致阴影投射，三个方向打光不能很好的表征缺陷特征，造成重建的图像特征不明显，所以通常使用三个以上不同方向的光源，避免死角。随着打光方向的增加采集图像也会跟着增加，那么算法处理的时间也会变长。根据经验：

1. 4-6个不同方向打光能满足大部分应用；
2. Slant角度一般选择30度-60度；
3. Tilt角度通常都是均匀分布在被测物体周围，比如3个方向打光，Tilt角度应该是[0,120,240]OR[0,120,-120]，4个方向打光是[0,90,180,-90]。需要注意的是，打光方向不能相同，否则重构的图像结果达不到预期效果。

输入图像：

输入图像是一个图像数组，其中每张图像都是在不同打光方向下采集的。如果采集的是多通道图像，可以通过算子 image\_to\_channels转换成单通道图像，采集的多张图像可以通过算子concat\_obj合并成一个数组图像。

光度立体法依托于对光度信息的评估，也就是图像中的灰度值。因此，图像质量的好坏决定了结果。要保证好的图像质量，首先要确保相机采集的图像具有线性特征，可以使用算子radiometric\_self\_calibration确认相机特性，如果相机采集的图像是非线性的，可以利用算子 lut\_trans 矫正灰度信息。此外，如果需要更高精度，可以从以下两点着手：（1）、使用相机的全部动态范围；（2）、使用高于8位深度的图像（灰度范围0-65535而不是0-255类型的图像）

输出图像：算子输出重建后的梯度、反射率、以及高度场图像。1、梯度图（矢量场）是根据对图像求偏导数获取，它可以作为算子reconstruct\_height\_field\_from\_gradient的输入。为了视觉观看更直观，将表面梯度进行归一化处理。因此ResultType类型需要设置成“normalized\_gradient”，而不是“gradient”。如果ResultType设置成默认模式“all”，处理方式是“gradient”，而不是“normalized\_gradient”，所以在参数设置时要根据需要设置。2、Albedo 图像描述的是物体的反射率，其值介于0（黑色）-1（白色）之间。因此，Albedo反应了物体表面特性。比如对于印刷表面表面，Albedo反应的是表面明暗程度的特性。

3，HeightField 图像中每个像素值以某种关系与其高度一一对应。

ResultType参数：

默认情况下，ResultType设置成“all”。假如在应用中仅仅需要部分结果，可以通过数组的形式在‘gradient’, ‘albedo’, and ‘height\_field’中选择设置ResultType参数，例如ResultType := [‘gradient’,‘albedo’]。对于特定的表面检测应用，如果只需要‘gradient’, ‘albedo’，那么将ResultType设置成‘gradient’, 'albedo’不进行三维重构（‘height\_field’），处理速度将会有效提升。

photometric\_stereo 函数：

photometric\_stereo算子首先会计算出梯度矢量场，如果需要高度场，光度立体法内部会采用reconstruct\_height\_field\_from\_gradient算子进行整合处理，通过 ReconstructionMethod, GenParamName, and GenParamValue这三个参数控制效果。如果参数ResultType参数中没有设置‘height\_field’，可以忽略这三个参数。

derivate\_vector\_field（处理photometric\_stereo 函数输出的重建后的梯度、反射率、以及高度场信息图）

derivate\_vector\_field(VectorField ,Result , Sigma, Component )  
参数列表：  
VectorField（in）// 梯度场图像  
Result（out） // 返回平均曲率场图像  
Sigma（in） // 高斯系数  
Component（in） //组件计算

derivate\_vector\_field函数详解：

将向量场的分量与高斯函数的导数进行卷积，并计算由此得到的各种特征。在光度立体项目中，专门用于处理photometric\_stereo 函数输出的重建后的梯度、反射率、以及高度场图像。

Sigma参数：

如果在Sigma中传递一个值，那么在列和行方向上的平滑量是相同的。

如果在Sigma中传递两个值，第一个值指定列方向的平滑量，第二个值指定行方向的平滑量。

Component参数：（有四个值可选，后两个值专用于光度立体）

1. curl，向量场的旋度。旋度的一个应用是分析光流场。旋度是如果向量场是流体，小船会旋转多少。
2. divergence，向量场的散度。“divergence”的一个应用是分析光流场。打个比方，如果向量场是流体，散度就是源和汇的位置。
3. mean\_curvature，当输入向量场 VectorField为梯度场时，下垫面的平均曲率H。用于处理photometric\_stereo返回的向量场。
4. gauss\_curvature，当输入向量场 VectorField 为梯度场时，下垫面的高斯曲率K。用于处理photometric\_stereo返回的向量场。

1，皮革表面缺陷检测

在实际应用中，有些产品缺陷对光源角度有要求，且方向不固定（比如：带方向的缺陷，需要多角度打光才能凸显缺陷的产品）那么就可以考虑光度立体法。

利用反射率图像和梯度图像检测皮革表面缺陷

下面是两种皮革表面拍摄图像：

dev\_close\_window ()  
dev\_open\_window (0, 0, 640, 480, 'black', WindowHandle)  
set\_display\_font (WindowHandle, 14, 'mono', 'true', 'false')  
\* Part 1利用反射率图像检测皮革表面缺陷  
read\_image (Images, 'photometric\_stereo/leather\_1\_0' + [1:4])  
write\_image (Images, 'tiff', 0, 'D:/1.tiff')  
\*\* 展示不同方向光源成像图像  
for I := 1 to 4 by 1  
 Message := 'Sample 1: Acquire image ' + I + ' of 4'  
 select\_obj (Images, ObjectSelected, I)  
 dev\_display (ObjectSelected)  
 disp\_message (WindowHandle, Message, 'window', 12, 12, 'black', 'true')  
 wait\_seconds (0.5)  
endfor  
\* 应用光度立体法生成的反射率图进行缺陷检测  
Tilts := [6.1,95.0,-176.1,-86.8]  
Slants := [41.4,42.6,41.7,40.9]  
ResultType := ['gradient','albedo']  
photometric\_stereo (Images, HeightField, Gradient, Albedo, Slants, Tilts, ResultType, 'poisson', [], [])  
\* 显示反射率图  
dev\_display (Albedo)  
\*检测缺陷  
var\_threshold (Albedo, Region, 15, 15, 0.4, 0.4, 'light')  
connection (Region, ConnectedRegions)  
select\_shape (ConnectedRegions, SelectedRegions, 'area', 'and', 10, 99999)  
union1 (SelectedRegions, RegionUnion)  
closing\_circle (RegionUnion, RegionClosing, 3.5)  
connection (RegionClosing, Defects)  
area\_center (Defects, Area, Row, Column)  
gen\_circle (Circle, Row, Column, gen\_tuple\_const(|Row|,sqrt(Area) + 30))  
\*显示缺陷  
dev\_display (Albedo)  
dev\_set\_color ('red')  
dev\_set\_draw ('margin')  
dev\_set\_line\_width (4)  
dev\_display (Circle)  
  
\* Part 2 利用梯度图像检测皮革表面缺陷  
read\_image (Images, 'photometric\_stereo/leather\_2\_0' + [1:4])  
for I := 1 to 4 by 1  
 Message := 'Sample 2: Acquire image ' + I + ' of 4'  
 select\_obj (Images, ObjectSelected, I)  
 dev\_display (ObjectSelected)  
 disp\_message (WindowHandle, Message, 'window', 12, 12, 'black', 'true')  
 wait\_seconds (0.5)  
endfor  
  
\* 应用光度立体法生成的反射率图  
photometric\_stereo (Images, HeightField, Gradient, Albedo, Slants, Tilts, ResultType, 'poisson', [], [])  
\*对反射率图二值化（发现无法二值化）  
threshold (Albedo, Region1, 128, 255)  
\* 显示反射率图  
dev\_display (Albedo)  
derivate\_vector\_field (Gradient, Curl, 1, 'curl')  
derivate\_gauss (Curl, CurlGradient, 1, 'gradient')  
\* 显示梯度图  
dev\_display (CurlGradient)  
Message := 'Changes in the gradient curl'  
disp\_message (WindowHandle, Message, 'window', 12, 12, 'black', 'true')  
disp\_continue\_message (WindowHandle, 'black', 'true')  
stop ()  
  
\* 用梯度图寻找缺陷  
threshold (CurlGradient, Region, 0, 0.01)  
rank\_region (Region, RegionCount, 10, 10, 30)//归类区域  
connection (RegionCount, ConnectedRegions)  
select\_shape (ConnectedRegions, SelectedRegions, 'area', 'and', 2000, 99999)  
union1 (SelectedRegions, RegionUnion)  
rank\_region (RegionUnion, RegionCount1, 25, 25, 170)  
connection (RegionCount1, NoTextured)  
  
\* 显示  
dev\_display (Albedo)  
dev\_set\_draw ('margin')  
dev\_set\_color ('red')  
dev\_set\_line\_width (3)  
dev\_display (NoTextured)  
disp\_message (WindowHandle, 'Non-textured areas on leather', 'window', 12, 12, 'black', 'true')  
stop ()