

电子科技大学计算机科学与工程学院

实 验 报 告

学 号 2020080903009

姓 名 李皓

(实验) 课程名称 实验二 基元检测

教师 任亚洲

电子科技大学

实验报告

学生姓名：李皓 学号：2020080903009 指导教师：任亚洲

一、实验名称: 图像预处理

二、实验目的:

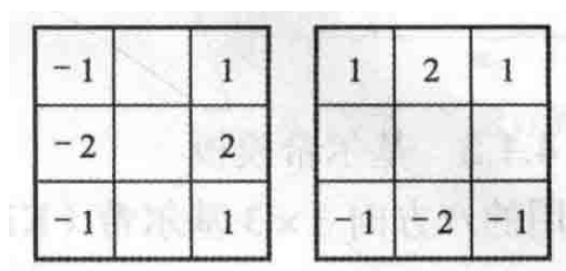
熟练掌握基元检测的各种方法, 包括边缘检测, 以及角点检测。

三、实验原理:

3.1 边缘检测

像素值灰度的变化可以通过计算导数的方法来检测, 我们一般使用一阶或者二阶导数进行检测。由于边缘的本质是灰度值的变化, 边缘处的导数会有不同的相应情况。对于阶梯状边缘而言, 一阶导数会在此处以阶跃形式出现, 而其余地方数值为 0; 对于屋顶状边缘, 一阶导数会出现上升沿和下降沿, 此时二阶导数的极小值点处便是图像的边缘所在。

Sobel 边缘导数算子常常分为两个方向, 要获得边缘响应, 我们需要根据两个偏导分量计算梯度矢量的模长。计算模长常常选择 1 范数、2 范数、以及无穷范数。

The image shows two 3x3 Sobel operator templates. The left template is for the horizontal gradient (Gx) and the right template is for the vertical gradient (Gy).

-1		1
-2		2
-1		1

1	2	1
-1	-2	-1

图 1: Sobel 算子

3.2 角点检测

我们以 SUSAN 算子介绍角点检测的方法。SUSAN 算子只使用一个圆形模板来得到各向同性的响应。它不仅可以检测出图像中目标的边缘点, 而且能较鲁棒地检测出图像中目标上的角点。

SUSAN 算子将模板中各个像素的灰度都与模板中心的核像素的灰度进行比

较，总有一部分模板区域像素的灰度与核像素的灰度相同或相似。这部分区域可称为核同值区（USAN 区），即与核有相同值的区域。USAN 区包含了很多与图像结构有关的信息。利用这种区域的尺寸、重心等统计量可以帮助检测图像中的边缘和角点。

利用上述 USAN 面积的变化可检测边缘或角点。具体说来，USAN 面积较大（超过一半）时表明核像素处在图像中的灰度一致区域，在模板核接近边缘时该面积减少，而在接近角点时减少得更多，即 USAN 面积在角点处取得最小值。

3.3 调研：高斯差分角点检测

张小洪等人提出的 [1] 高斯差分角点检测方法利用了不同 σ 值下的高斯卷积之差进行角点检测。如图（2）所示，利用一组卷积模板处理图像，观察可知角点在不同 σ 值下的变化程度不同，可以根据这一特性进行角点检测。

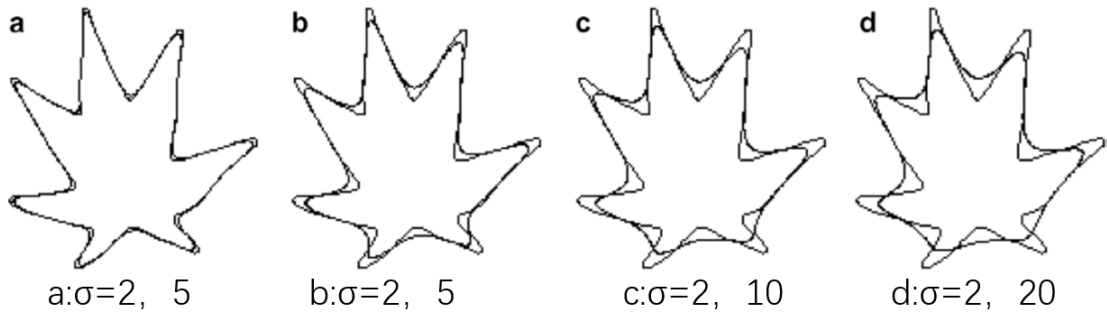


图 2: 不同 σ 值下角点检测的结果

我们定义高斯模板如下：

$$G(u, \sigma) = \frac{1}{\sqrt{2\pi}\sigma^2} \exp(-u^2/2\sigma^2) \quad (1)$$

则卷积后的图像为：

$$C(u, \sigma) = G(u, \sigma) * C(u) = (X(u, \sigma), Y(u, \sigma)) \quad (2)$$

计算不同 σ 数值下的差分图像 D ：

$$D(u, \sigma) = C(u, m\sigma) - C(u, \sigma) \quad (3)$$

$$[DoG * x(u)]^2 + [DoG * y(u)]^2 \quad (4)$$

其中 DoG 为高斯差分算子，计算如下：

$$DoG = G(u, m\sigma) - G(u, \sigma) \quad (5)$$

四、实验内容:

- 1) 边缘检测: 利用一阶导数算子(节 4.1.2)进行边缘检测, 可采用图 4.1.2 提供模板中的任意一个。在真实图像上测试, 对一副真实图像, 需画出图 4.1.3 的六副图像。
- 2) 角点检测: 利用 SUSAN 算子进行角点检测, 并在真实图像上测试, 将角点标识(参照图 4.2.5)。
- 3) 高斯差分角点检测: 实现高斯差分角点检测, 并且与 SUSAN 算子角点检测进行对比。

五、实验设备:

- Ubuntu 个人版 20.04 LTS
- Pycharm 2022.1
- Python 3.9
- PIL numpy matplotlib 等 python 库

六、实验步骤:

6.1 使用 Sobel 算子进行图像边缘检测

该部分使用 Sobel 算子进行角点检测, 首先我们需要使用 python 中的 PIL 库读取图像, 并且将其转化为'int32' 类型的矩阵以避免运行过程中出现溢出。

```
1 from PIL import Image
2 pil_im = Image.open("sky.png").convert('L')
3 im_array = asarray(pil_im)
```

为了提高计算的性能, 所有计算都尽可能使用 ndarray 中的矩阵乘法实现, 为此, 不同方向的 Sobel 偏导算子都使用矩阵存储。

```
1 Sobel_mask_x = np.array([[ -1, 0, 1],
2                           [ -2, 0, 2],
3                           [ -1, 0, 1]])
4
5 Sobel_mask_y = np.array([[ 1, 2, 1],
6                           [ 0, 0, 0],
7                           [-1, -2, -1]])
```

我们定义了模板运算的函数 mask_slid_3x3, 传入图像数组以及模板, 再指定二值化的图像阈值, 便可以得到两个方向上偏导数的响应, 以及 1 范数、2 范数、无穷范数下的梯度响应。

```

1 mask_slid_for_3x3(img_array, Sobel_mask_x, Sobel_mask_y, thershold=110, file_name
  = 'sky_sobel_110_')

```

在该函数中首先创建图像的副本，以存储边缘响应的结果：

```

1 (x_length, y_length) = img.shape
2 img_x = img.astype('int32')
3 img_y = img.astype('int32')
4 img_g_1 = img.astype('int32')
5 img_g_2 = img.astype('int32')
6 img_g_i = img.astype('int32')
7 # 复制备份
8 padding_img = np.pad(img, ((1, 1), (1, 1)), 'edge').astype('int32')
9 # 使用复制填充边缘

```

随后使用传入的模板运算，得到处理结果。该运算过程只使用了两层 for 循环，效率较高。

```

1 for i in range(x_length):
2     for j in range(y_length):
3         block = padding_img[i:i + 3, j:j + 3]
4         temp_x = block * mask_x
5         temp_y = block * mask_y
6         temp_x = temp_x.sum()
7         temp_y = temp_y.sum()
8         img_x[i][j] = temp_x
9         img_y[i][j] = temp_y
10        img_g_1[i][j] = abs(temp_x)+abs(temp_y)
11        img_g_2[i][j] = math.sqrt(temp_x**2+temp_y**2)
12        img_g_i[i][j] = max(temp_y,temp_x)

```

计算所得的结果不在 0-255 的灰度范围内，所以我们还需要对每一张图像进行最大最小拉伸，使之落在 0-255 的范围内。在拉伸之后，我们使用传入的阈值，将图片转化为二值图像。在完成所有工作后，保存图像。

```

1 single_dir_pic = [img_x, img_y]
2 for i, img_copy in enumerate(single_dir_pic):
3     sub_img = img_copy - img
4     sub_img -= sub_img.min()
5     sub_img = sub_img.astype('float64')
6     sub_img *= (255.0 / sub_img.max())
7     sub_img = sub_img.astype('uint8')
8     im = Image.fromarray(sub_img)
9     im.save(fp= file_name + str(i) + '.png',
10            format='png')
11
12 grad_pic = [img_g_1, img_g_2, img_g_i]
13 for i, img_copy in enumerate(grad_pic):
14     sub_img = img_copy - img
15     sub_img -= sub_img.min()
16     sub_img = sub_img.astype('float64')
17     sub_img *= (255.0 / sub_img.max())
18     sub_img = sub_img.astype('uint8')
19     sub_img_b = np.where(sub_img > thershold, 255, 0)
20     im_b = Image.fromarray(sub_img_b.astype('uint8'))
21     im_b.save(fp= file_name + str(i+2) + 'b.png',
22              format='png')

```

6.2 使用 SUSAN 算子进行角点检测

考虑到 SUSAN 算子的众多超参数，在设计过程中，我在函数入口处给出调节的参数，便于对比不同效果。

```
1 def SUSAN(radius = 5, img=None, threshold_T = 27, Adjust_G=3,  
2         top_k_range = [5,10,20,30,40,80,160], filename = 'obj'):
```

为了创建一个圆形的 SUSAN 算子，并且实现可调半径的功能，我使用正误二值数组创建了一个半径为 r 的圆形模板。

```
1 x = np.arange(0, 2 * radius + 1)  
2 y = np.arange(0, 2 * radius + 1)  
3 circle_mask = (x[np.newaxis, :] - radius) ** 2 + (y[:, np.newaxis] -  
4             radius) ** 2 <= radius ** 2  
# 创建可复用的模板
```

随后我们遍历每个像素，将原始图像首先啊切分为两倍半径的 block，再在 block 之上使用圆形模板处理，成功将循环数量减少至 2 层。

```
1 for i in range(x_length):  
2     for j in range(y_length):  
3         logging.info("processing : " + str((i, j)))  
4         block = np.copy(padding_img[i:i + 2 * radius + 1, j:j + 2 * radius + 1])  
5         # 裁剪处理的块  
6         center_expand_block = np.full(block.shape, copy_img[i][j])  
7         # 每个中心的块  
8         block -= center_expand_block  
9         block = abs(block)  
10        block_C = np.where(block > threshold_T, 0, 1)  
11        # 判断是否为和同值区域  
12        block = block_C.astype('bool')  
13        block *= circle_mask  
14        # 掩盖出圆形区域  
15        copy_img[i][j] = block.sum()  
16        # 得到每一个像素的核同值区域
```

为了标记角点位置，我们从大到小的角点响应，记录 top_k 索引，并且在原始图像上标记。由于 top_k 不需要按序排列，我们使用 ndarray 中的 partition，返回索引。这样的好处在于避免循环遍历，加速计算。

```
1 def k_largest_index_argpartition_v2(a, k):  
2     idx = np.argpartition(a.ravel(), a.size-k)[-k:]  
3     return np.column_stack(np.unravel_index(idx, a.shape))  
4     # top_k 函数  
5  
6 for top_k in top_k_range:  
7     coner_list = k_largest_index_argpartition_v2(img_R, top_k)  
8     coner_list_xy = coner_list.T  
9     x = coner_list_xy[0].tolist()  
10    y = coner_list_xy[1].tolist()  
11    plt.figure()  
12    plt.scatter(y, x, color='r')  
13    name_str = dir_name + '_' + str(top_k)  
14    if '.' in name_str:  
15        name_str = name_str.replace('.', '_')  
16    plt.imshow(img_R, cmap=cm.gray)  
17    plt.title(name_str)  
18    plt.savefig(dir_name + '/' + name_str)  
19    plt.show()
```

20 # 选取不同top_k值标记角点

为了方便显示边缘检测的效果，我们还保存了每一个记录为角点的 block 图像：

```
1 for x_,y_ in zip(x,y):
2     plt.figure()
3     plt.imshow(np.copy(padding_img[y_:y_ + 2 * radius + 1, x_:x_ + 2 * radius +
4         1]))
5     plt.scatter(radius, radius, color='r')
6     plt.title(str((y_, x_))+'_'+str(copy_img[x_][y_]))
7     plt.savefig(dir_name + '/edge/' + str((y_, x_))+'_'+str(img_R[x_][y_]))
8     plt.show()
```

6.3 使用 DoG 算法进行角点检测

该算法使用不同的 σ 值对原始图像进行处理。DoG 论文中提出 σ 值选取情况与噪声相关，我们需要维护每次 m 值的变化范围，所以设计以下函数：

```
1 def DoG_Corner_Dection(img=None,thershold = 5.0,sigma = 2**(0.5),
2     num_DoG_images =4,m=1.5,
3     filename = 'obj',top_k_range = [5,10,20,30,40,80,160]
4     ):
5     gaussian_images = []
6     #处理后的高斯卷积图像
7     DoG_images = []
8     #相邻两个高斯卷积图像之差
9     gaussian_images.append(img)
10    num_gaussian_images=num_DoG_images+1
11    # 根据论文中的结论，sigma的大小由噪声情况确定。
```

我们使用 pil 库中的高斯卷积模板卷积处理图像，并且保存在数组中。

```
1 for i in range(num_gaussian_images):
2     r = sigma*(i+1)*m
3     img_temp = pil_im.filter(ImageFilter.GaussianBlur(radius=r))
4     plt.imshow(np.asarray(img_temp),cmap=cm.gray)
5     name_temp = str(i)+"_sigma_"+str(int(r))
6     plt.title(name_temp)
7     plt.savefig(dir_name+'//DoG_pic/'+name_temp)
8     gaussian_images.append(np.asarray(img_temp).astype('int32'))
```

随后对高斯卷积结果求差，得到 DoG 结果。将 DoG 结果之差最大的位置记录为角点位置。

```
1 for k in top_k_range:
2     key_points = []
3     for i in range(num_DoG_images):
4         DoG_images.append(abs(gaussian_images[i]-gaussian_images[i+1]))
5         temp_key_point_list = k_largest_index_argpartition_v2(DoG_images[i], k)
6         for j,key in enumerate(temp_key_point_list):
7             if DoG_images[i][key[0]][key[1]]<thershold:
8                 temp_key_point_list[j]=[0,0]
9         key_points.append(temp_key_point_list)
10
11    key_points = np.asarray(key_points)
12    key_points = np.unique(key_points,axis=0)
13    key_points = key_points.T
14
15    x = key_points[0].tolist()
```

```

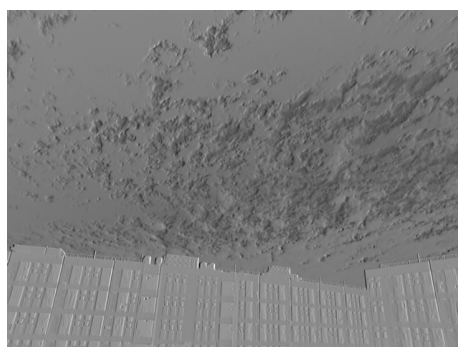
16 y = key_points[1].tolist()
17 plt.figure()
18 plt.scatter(y, x, color='r')
19 plt.imshow(img, cmap=cm.gray)
20 fig_name = str(k)
21 plt.title(fig_name)
22 plt.savefig(dir_name+ '/' +fig_name)
23 plt.show()

```

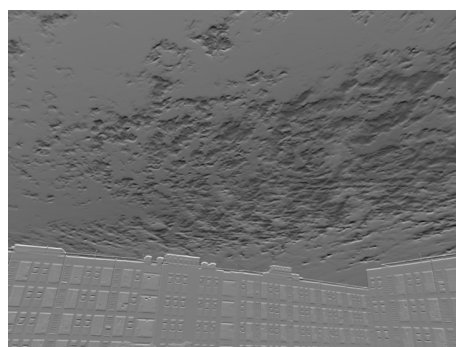
七、实验结果与分析:

7.1 边缘检测

边缘检测结果如下图所示：如图可知，x 方向和 y 方向的边缘被明显检测出来，



(a) x 方向 Sobel 算子



(b) y 方向 Sobel 算子



(c) 原始图像

图 3: 不同方向的 sobel 算子处理结果

为了得到整体响应，我们还需要对偏导算子进行不同范数的计算。使用不同范数的梯度响应图如图（4）所示：

可见二范数下图像梯度响应结果最好，这是因为该范数下各个方向的响应值都被计算到最终的响应中，而无穷范数只计算了两个方向响应的最大值，丢失了较小的方向响应结果。

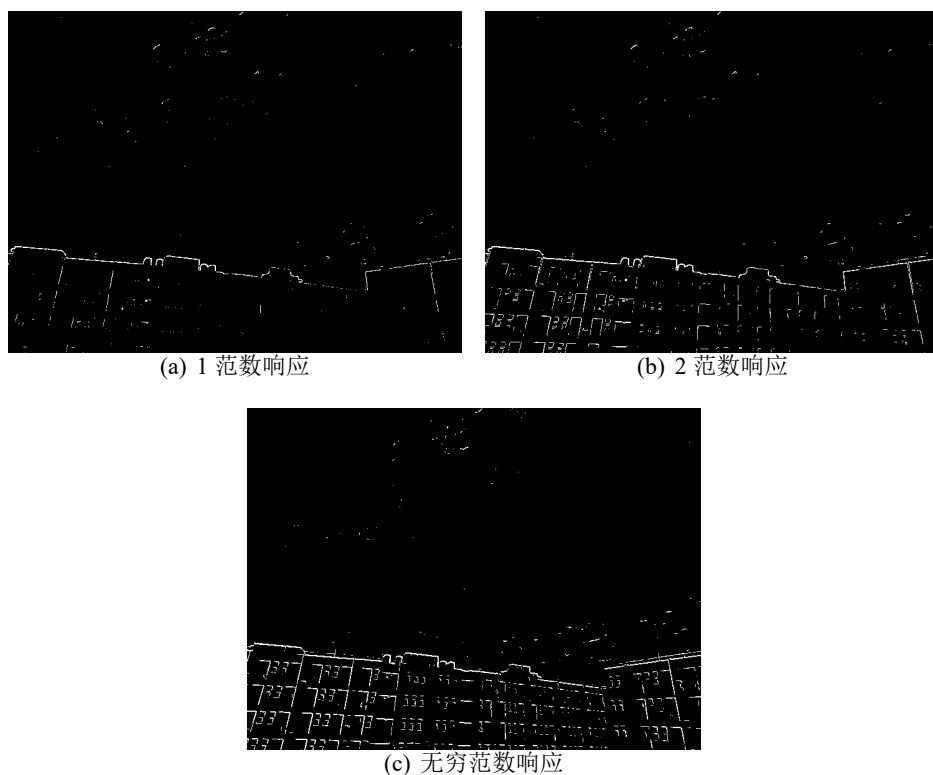


图 4: 不同范数梯度图响应结果

7.2 角点检测

7.2.1 二值图像中的角点检测结果

在二值图像上使用 SUSAN 算子，得到图（5）响应结果：

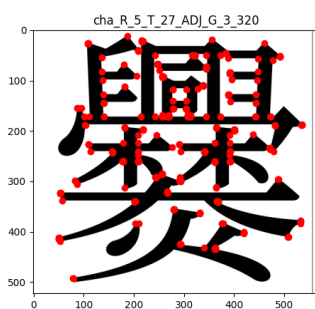


图 5: SUSAN 算子二值图像响应结果

在调整了多次参数之后，设置半径为 5，灰度阈值为 27，最大面积为 0.75 倍模板大小获得了该结果。可见检测效果较好，几乎没有漏检错检的情况。

7.2.2 真实图像上的角点检测结果

在真实图像上使用 SUSAN 算子并没有获得一致的效果，首先添加了 SUSAN 的响应下限，认为核同值区域需要大于 4 倍半径才认为是角点。得到图 (6) 中的结果：可见改进后的方法减少了部分位置错误识别的情况，但是带来了另一些部分

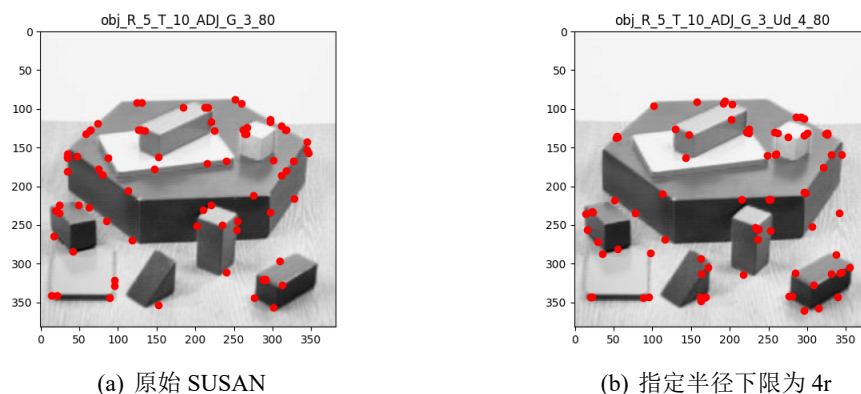


图 6: 指定半径下限的 SUSAN 算子结果

角点的错误检测。为此，我又在添加响应下限的基础上改变半径为 10，再次比对效果 (图 7):

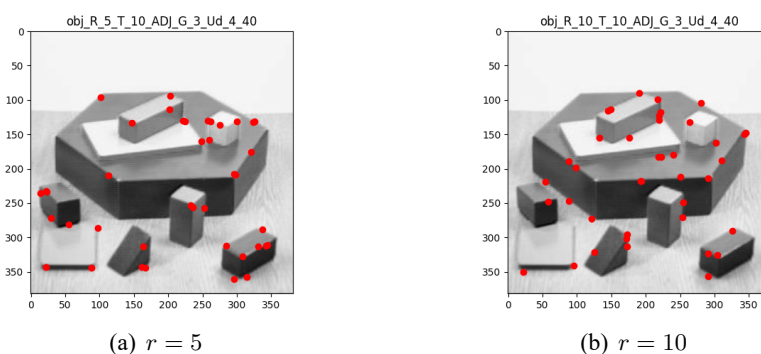


图 7: 不同半径范围下的 SUSAN 算子检测效果

SUSAN 算子半径增加后反而带来更多误分类，说明对于边缘模糊的现实图像，SUSAN 算子表现并不好。我将在后面分析二值图像和现实图像的差异，以分析为何 SUSAN 算子表现不好的原因。

7.2.3 不同图片的对比

SUSAN 算子在二值图像和真实图像的检测效果不同，本质上是因为边缘模糊或者是形状复杂，导致错误识别角点。观察图（8），可以直观对比二者差异。

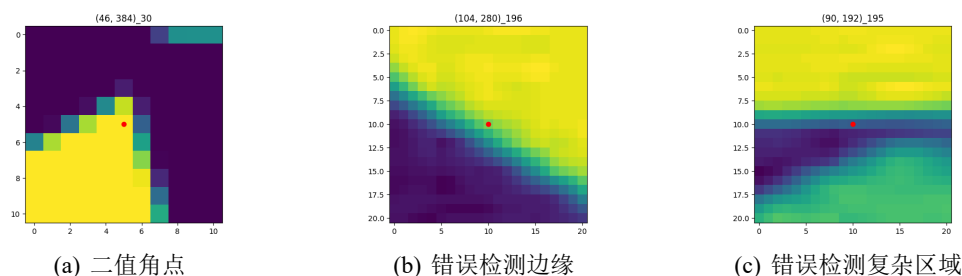


图 8: 角点检测局部细节

在真实图像的检测中，模糊的边缘对角点检测的效果影响很大，而如果设置较小的阈值，又会导致噪声不敏感以及遗漏角点的现象。因此，我们需要探寻新的方法进行真实图像上的角点检测。

7.3 DoG 算法

使用 DoG 算法对原始图像再次检测角点，得到以下结果：图（9）

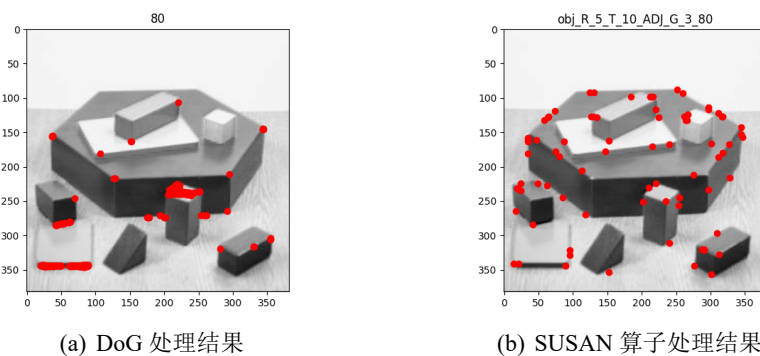


图 9: DoG 与 SUSAN 处理结果对比

DoG 算法几乎没有出现边缘点的误分类，但是出现了图形内部错误识别为角点的问题。这可能是高 σ 数值下的错误响应，可以添加最小响应阈值消除。

八、总结及心得体会:

在本次实验中，我基本掌握和复现了常用的图像基元检测算法。并且通过前后图像对比，直观认识了不同处理方法的效果以及特点。在这一基础上，我在角

点检测这一方面展开研究，并且复现了 DoG 代码。

在本次图像基本操作的实验中，我加深了对一些常见图像处理方法的认知，锻炼了自己的编码能力。在基本要求的基础上，我还寻找了直方图均衡化的创新算法：DoG，对课内知识做了补充。在这一过程中，我锻炼了自己信息检索的能力，以及阅读相关文献资料的能力。

九、对本实验过程及方法、手段的改进建议：

- 1) 边缘检测：可以尝试多种边缘检测的模板，在不同图像上实验模板效果，对比差异。
- 2) 角点检测：尝试查阅相关资料，实现改进的 SUSAN 算子方法。
- 3) **DoG** 角点检测：添加最小响应阈值，以消除区域内部的误检测。

报告评分：

指导教师签字：

参考文献

- [1] Xiaohong Zhang, Honxing Wang, Mingjian Hong, Ling Xu, Dan Yang, and B. Lovell, “Robust image corner detection based on scale evolution difference of planar curves,” *Pattern Recognition Letters*, vol. 30, no. 4, pp. 449–55, Mar. 2009, place: Netherlands Publisher: Elsevier Science B.V.