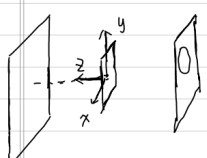


一. Written Assignment

a. 以下图所示建立空间坐标系



假设 disk 的圆心为 (x_0, y_0, z_0)
则边缘的方程为 $(x-x_0)^2 + (y-y_0)^2 = R^2, z=f$
根据相似三角形, 边缘上某点 (x, y, z) 的像点 (x', y', z') 满足 $x = \frac{z_0}{z'}x', y = \frac{z_0}{z'}y', z = f$
 \therefore 代入上述方程
 $(\frac{z_0}{z'}x' - x_0)^2 + (\frac{z_0}{z'}y' - y_0)^2 = R^2$
 \therefore 成像仍然是圆

b.

① 由题意 $A=C=D=0, B=1 \Rightarrow y=0$
选取如下 3 个方向向量:
 $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$
由 vanishing point 公式, 对应 vanishing point 为 $(2f, 0), (\frac{2}{3}f, 0), (\frac{4}{3}f, 0)$
由于在该平面上 $\frac{y}{z}$ 可以为任意值
故整个平面的投影为 $y=0, z=f$

② $B=C=D=0, A=1$
同理, 选取 $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$
对应 vanishing point 为 $(0, \frac{1}{2}f), (0, \frac{2}{3}f), (0, \frac{4}{3}f)$

故整个平面的投影为 $x'=0, z'=f$

c.

设相平面中任意的一条直线的法向量为 (a, b, c)
则 $Aa + Bb + Cc = 0 \quad \dots ①$
该直线的 vanishing point 为 $\begin{cases} x' = f \cdot \frac{a}{c} \\ y' = f \cdot \frac{b}{c} \end{cases} \quad ②$
 $\Rightarrow \begin{cases} \frac{cx'}{f} = a \\ \frac{cy'}{f} = b \end{cases} \quad \text{代入 } ①$
 $A \cdot \frac{cx'}{f} + B \cdot \frac{cy'}{f} + Cc = 0$
 $\Rightarrow Ax' + By' + Cf = 0$

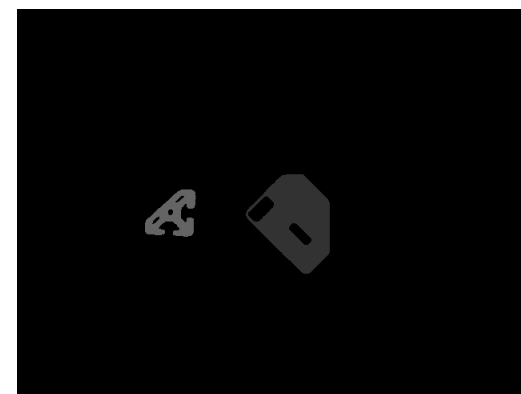
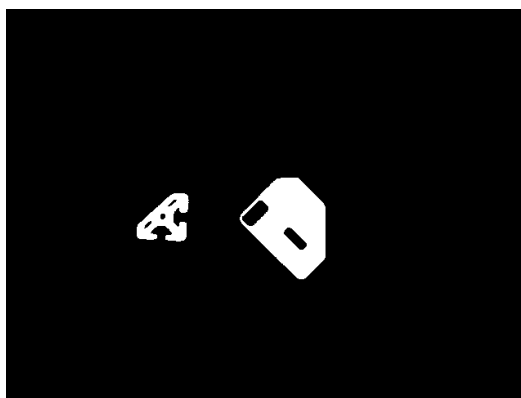
二. Programming Assignment

Problem1:

a: 主要思路为遍历灰度图的每一个像素, 如果该像素值超过阈值 (如 128) 则将二值图像的该位置设置为 255. 否则设置为 0. 处理后的 two_objects.png 如右图所示:

b: 主要思路为根据搜索附近像素的递归算法实现 sequence labeling, 对二值图像中的每一个连通分量打上标签, 算法伪代码参考 <https://courses.cs.washington.edu/courses/cse373/00au/chcon.pdf>. 实现过程中使用 8 邻居的搜索范围, 同时将 python 的最大迭代次数设置为 10000. 打完标签之后的示意图如右图所示:

c: 首先遍历 labeled image 记录下除了背景之外的所有点的 xy 坐标, 并进行坐标转换 (原点转换), 然后计算出每一个连通分量的平均 xy 坐标. 接下来, 为了计



算每一个连通分量的 **orientation**，再对之前的 **xy** 坐标进行原点迁移。接下来遍历新坐标中的每一个点，按照公式

where:
$$\begin{aligned} a &= \iint_I (x')^2 b(x, y) dx' dy' \\ c &= \iint_I (y')^2 b(x, y) dx' dy' \end{aligned}$$

$$b = 2 \iint_I (x' y') b(x, y) dx' dy'$$

25 October 2021 Computer Vision, SJTU, Wei Shen (a, b, c are easy to compute)

Therefore,
$$\text{Orientation: } \theta = \theta_1 = \frac{\text{atan2}\left(\frac{b}{a-c}\right)}{2}$$

Oct 2021 Computer Vision, SJTU, Wei Shen

计算出每一个连通分量的 **orientation**，代入 E 函数计算 **roundedness**。

attribute_list 如下：

```
two_objects: [{'position': (349.33298470388286, 216.45365407242778), 'roundedness': 0.5336319534756405, 'orientation': -1.259956523462201}, {'position': (195.3160469667319, 223.3820939334638), 'roundedness': 0.4799636466920349, 'orientation': 0.6875462936637149}]

Many_objects_1: [{'position': (265.97616566814276, 365.13401927585306), 'roundedness': 0.5217196889211334, 'orientation': 0.08042727460237048}, {'position': (461.6430812129662, 313.7504356918787), 'roundedness': 0.9902664427338181, 'orientation': 1.2635628997735306}, {'position': (326.0154385964912, 309.29473684210524), 'roundedness': 0.13319471993926776, 'orientation': 0.7788385087054038}, {'position': (417.71620665251237, 241.29181410710072), 'roundedness': 0.024421609826590758, 'orientation': -0.7760238443266907}, {'position': (268.30828220858893, 257.85327198364007), 'roundedness': 0.4860732206012444, 'orientation': -0.5388371734983242}, {'position': (303.571394686907, 178.27300759013283), 'roundedness': 0.2702711841586357, 'orientation': 0.405201992726549}]

Many_objects_2: [{'position': (188.3515625, 357.90033143939394), 'roundedness': 0.007633528961638986, 'orientation': -0.6431420831724858}, {'position': (331.9617982504706, 338.21769460746316), 'roundedness': 0.3072674402498919, 'orientation': -1.5309195723290279}, {'position': (475.3399815894446, 339.9671678428966), 'roundedness': 0.020855451285962178, 'orientation': 0.4032474194877969}, {'position': (413.6556685685934, 204.95137682957082), 'roundedness': 0.17394416151886238, 'orientation': -1.1179094173122177}, {'position': (130.16157675232074, 188.1522938248352), 'roundedness': 0.5078766943974373, 'orientation': -1.4483813438029263}, {'position': (265.9671412924425, 169.6462212486309), 'roundedness': 0.48091224785679243, 'orientation': -0.49296932904138474}]
```

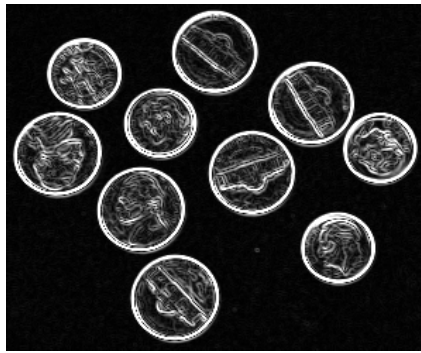
Problem2:

a: 首先构造 Sobel 算子如下图所示：

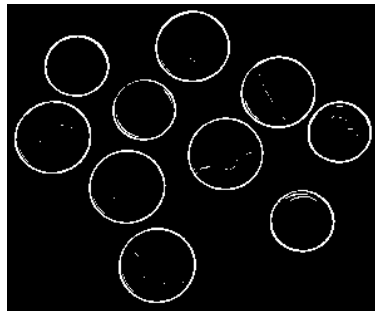
x 方向:
$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$
 y 方向:
$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

然后使用 **cv2.filter2D** 函数对原灰度图进行卷积实现得到 2 个梯度矩阵，之后对这两个矩阵的每一个元素进行平方和再开根号，得到最终的边缘图像，如下图所示：

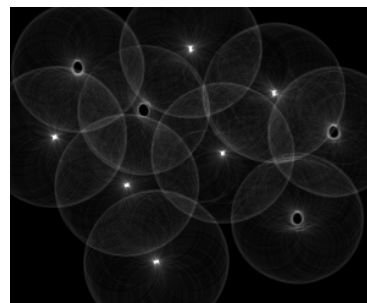
b: 由于 a 中得到的边缘并不全是圆的边缘，故为了检测出图中的圆，需要将边缘图像中的梯



度小于阈值的边缘去掉。经过多次尝试，阈值设置为 300 效果较好，边缘图像如下：



然后，遍历所有边缘点，遍历所有可能的半径取值，对每一个边缘点画圆绘制 hough transform 图像，得到的 hough transform 域的图像为：



半径 29 为例

c: 通过 b 中得到的 accumulator array，遍历每一个 r 和 x, y ，选出超过阈值的部分点，作为圆心的候选（即上图中较亮的点）。经过多次尝试，阈值选取为 100。但是筛选之后的点较多，主要原因是存在一些局部区域，该区域内部有多个半径接近，圆心也接近的圆。接下来将进行非极大值抑制的方法来消除这一现象，主要思路为距离为 10 个像素点之内的半径之差小于 2 的多个圆中只取一个 vote 最多的圆。最后经过处理后得到 10 个圆，与图片中的硬币个数相同他们的半径和坐标为：[(24, 49, 55), (24, 84, 109), (24, 102, 265), (25, 173, 235), (28, 33, 148), (28, 70, 217), (29, 106, 36), (29, 119, 174), (29, 145, 95), (30, 208, 119)] 将其画在原图上的效果为：

