Computer Vision Homework 01

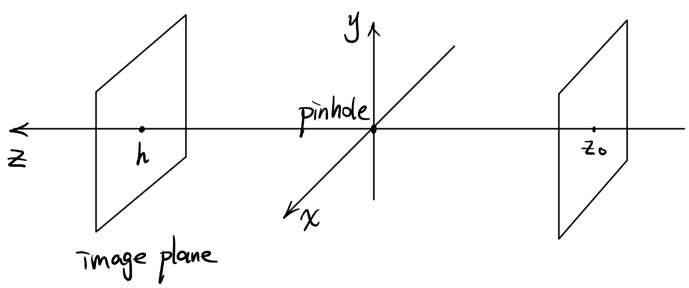
Qiu Yihang

Oct.13-19, 2022

1 Written Assignment

1.1 The Image of Circular Disks

Solution. Let the optical axis be $x = 0, y = 0$. Let the plane where the circular disk is on be $z = z_0$.



Let the circular disk be $C : (x - x_0)^2 + (y - y_0)^2 = r^2$.

Then any point in C can be represented as $(x_0 + r\cos\theta, y_0 + r\sin\theta)$.

Since the camera is a pinhole camera, we know for any point $(x_0 + r\cos\theta, y_0 + r\sin\theta) \in C$,

$$\frac{x_0 + r\cos\theta}{-x_i} = \frac{y_0 + r\sin\theta}{-y_i} = \frac{z_0}{h},$$

where h is the distance of the image plane to the pinhole.

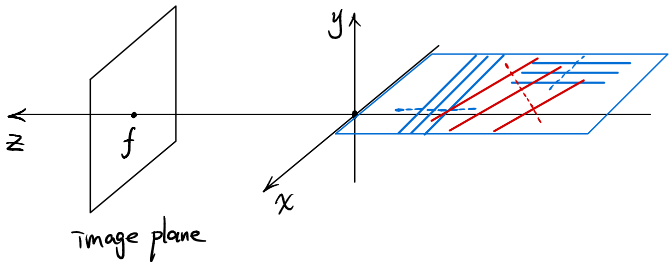
Then we have

$$\begin{aligned} & x_i = -\frac{h}{z_0}\left(x_0 + r\cos\theta\right), \quad y_i = -\frac{h}{z_0}\left(y_0 + r\sin\theta\right) \\ \implies \quad & C' : \left(x_i - \frac{hx_0}{z_0}\right)^2 + \left(y_i - \frac{hy_0}{z_0}\right)^2 = \left(\frac{hr}{z_0}\right)^2 \end{aligned}$$

Thus, the shape of the image is also **a circular disk**. ■

1.2 Vanishing Points in Special Cases

Solution. First we consider the case where $A = C = D = 0, B = 1$, i.e. the plane is $y = 0$.



We consider the following three sets of parallel lines in this plane: $z = \texttt{const}, x = \texttt{const}$, $x - z = \texttt{const}$, whose direction vectors are $l_1 = (0, 0, 1), l_2 = (1, 0, 0), l_3 = (1, 0, -1)$ respectively.

For any (x, y, z) on direction $l = (l_x, l_y, l_z)$, we know

$$\begin{cases} x = x' + tl_x \\ y = y' + tl_y \\ z = z' + tl_z \end{cases}, \qquad \frac{x}{x_i} = \frac{y}{y_i} = \frac{z}{f} \quad \Longrightarrow \quad \begin{cases} x_i = \dfrac{x' + tl_x}{z' + tl_z} f \\ y_i = \dfrac{y' + tl_y}{z' + tl_z} f \\ z_i = f \end{cases}$$

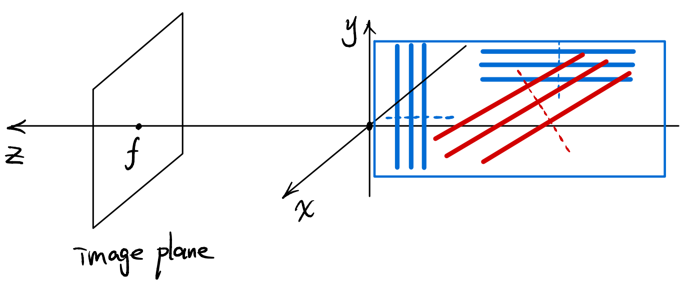
$$t \to \infty, \quad \text{we know Vanishing Point } \left(\frac{fl_x}{l_z}, \frac{fl_y}{l_z}, f \right)$$

CASE 1. $l_1 = (0, 0, 1)$. Thus, the vanishing point is $(0, 0)$. □

CASE 2. $l_2 = (1, 0, 0)$. From the figure above we know there is no vanishing point in this case, since l_2 is parallel to the image plane. The image of all lines is $y = 0$. □

CASE 3. $l_3 = (1, 0, -1)$. Thus, the vanishing point is $(-1, 0)$. □

Now we consider the case where $B = C = D = 0, A = 1$, i.e. the plane is $x = 0$.



We consider the following three sets of parallel lines in this plane: $z = \texttt{const}, y = \texttt{const}$, $y - z = \texttt{const}$, whose direction vectors are $l_1 = (0, 0, 1), l_2 = (0, 1, 0), l_3 = (0, 1, -1)$ respectively.

CASE 1. $l_1 = (0, 0, 1)$. Thus, the vanishing point is $(0, 0)$. □

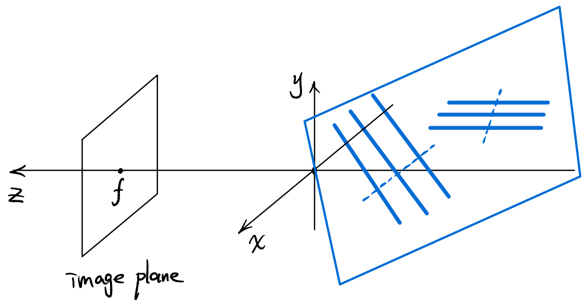
CASE 2. $l_2 = (0, 1, 0)$. From the figure above we know there is no vanishing point in this case, since l_2 is parallel to the image plane. The image of all lines is $x = 0$. □

CASE 3. $l_3 = (0, 1, -1)$. Thus, the vanishing point is $(0, -1)$. ■

1.3 General Relationship Between the Vanishing Point and Lines in a Plane

Solution. Consider the plane $Ax + By + Cz + D = 0$.

Let the vanishing point of lines in the plane $Ax + By + Cz + D = 0$ be (x_p, y_p).



Obvious all directions of lines in the plane are in the plane. Thus, for lines in the plane, we know their direction $l = (l_x, l_y, l_z)$ satisfies $Al_x + Bl_y + Cl_z = 0$.

When $l_z = 0$, l is parallel to the image plane, i.e. the vanishing point does not exist.

When $l_z \neq 0$, we know $A\frac{l_x}{l_z} + B\frac{l_y}{l_z} + C = 0$.

In **1.2**, we have shown the vanishing point of lines with direction (l_x, l_y, l_z) is $\left(\frac{fl_x}{l_z}, \frac{fl_y}{l_z}, f\right)$.

Thus, we know

$$x_p = \frac{fl_x}{l_z},\ y_p = \frac{fl_y}{l_z},\ z_p = f \implies \frac{A}{f}x_p + \frac{B}{f}y_p + C = 0,\ z_p = f.$$

Therefore, the vanishing points of lines in the plane $Ax + By + Cz + D = 0$ are **on the line** $\boldsymbol{\frac{A}{f}x + \frac{B}{f}y + C = 0,\ z = f}$. ■

2 Programming Assignment

2.1 Problem 1: 2D Object Detecting Vision System

The main ideas for the three functions to be completed are listed as follows.

- `binarize`(gray_image, thresh_val): Compare each pixel to the threshold. Here we **choose 128 as the threshold**.
- `label`(binary_image): Implement the **sequential labeling algorithm**. In the second pass, we relabel each connected component and assign a color value $\texttt{color}(C) = \dfrac{\texttt{label}(C)}{N} \times 255$, where N is the number of connected components, $\texttt{label}(C) \in \{1, 2, ..., N\}$.
- `get_attribute`(labeled_image): Find a labeled pixel. Use **BFS** to find all pixels in the same connected component while collecting their coordinates.

  Then we calculate position, orientation and roundness of objects as follows.

$$\bar{x} = \frac{\sum_{(x,y)\in C} x}{\sum_{(x,y)\in C} 1}, \qquad \bar{y} = \frac{\sum_{(x,y)\in C} x}{\sum_{(x,y)\in C} 1}$$

$$a = \sum_{(x,y)\in C} (x - \bar{x})^2, \qquad b = 2 \sum_{(x,y)\in C} (x - \bar{x})(y - \bar{y}), \qquad c = \sum_{(x,y)\in C} (y - \bar{y})^2$$

$$\theta_1 = \frac{1}{2}\arctan\left(\frac{b}{a-c}\right), \qquad \theta_2 = \theta_1 + \frac{\pi}{2}$$

$$\text{second moment} \quad \texttt{E}(\theta) = a\sin^2\theta - b\sin\theta\cos\theta + c\cos^2\theta$$

$$\texttt{position}(C) = (\bar{x}, \bar{y})$$

$$\texttt{orientation}(C) = \underset{\theta\in\{\theta_1,\theta_2\}}{\operatorname{argmin}} \texttt{E}(\theta)$$

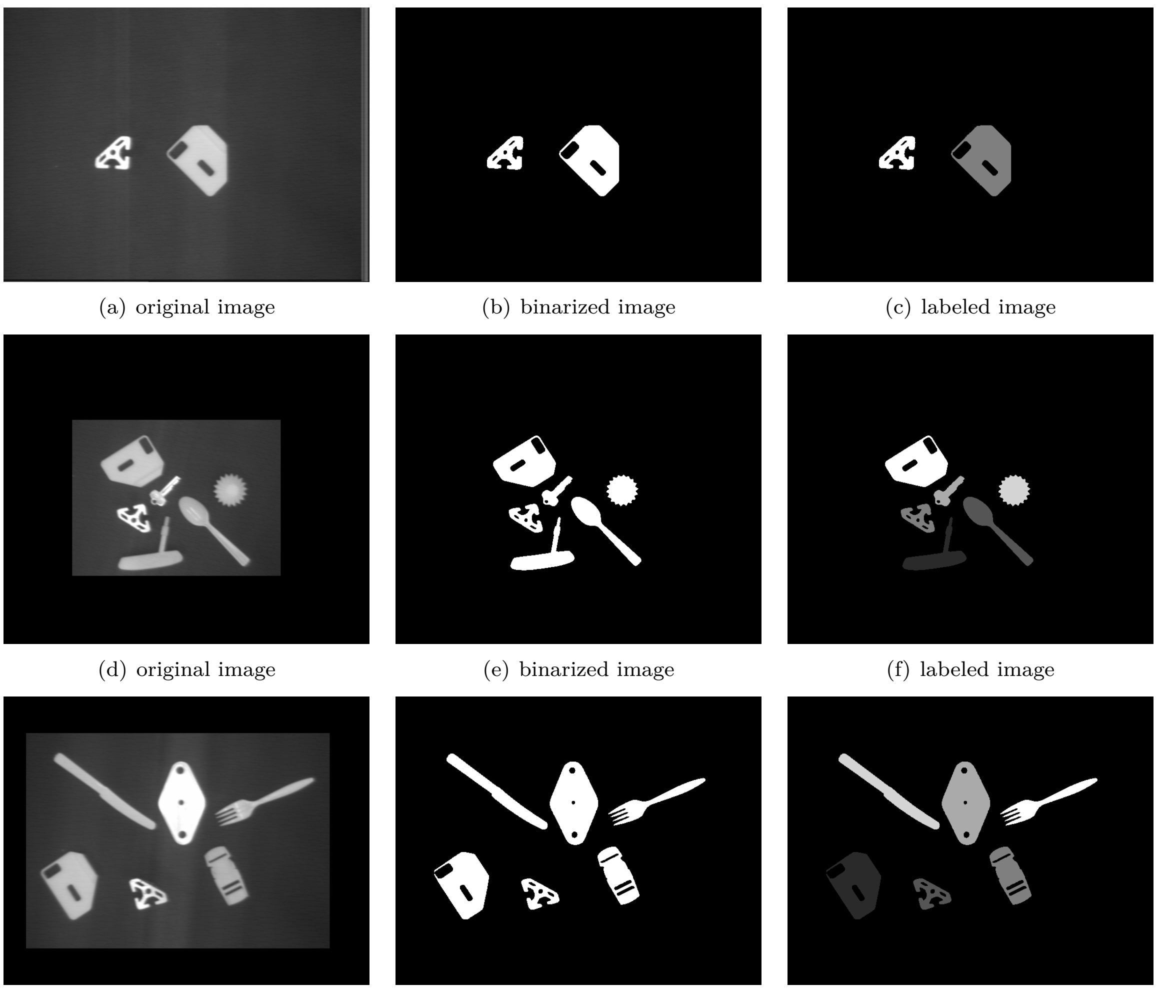
$$\texttt{roundness}(C) = \frac{\min \texttt{E}(\theta)}{\max \texttt{E}(\theta)} = \frac{\texttt{E}(\texttt{orientation}(C))}{\texttt{E}\left(\theta_1(C) + \theta_2(C) - \texttt{orientation}(C)\right)}$$

The object attributes are as follows.

```
# two_objects.png
[{'position': {'x': 349.41329138812426, 'y': 216.40031458906802},
  'orientation': 1.8790893714106605, 'roundedness': 0.5340063842834148},
 {'position': {'x': 195.30663390663392, 'y': 223.4181818181818},
  'orientation': 0.6871951338147986, 'roundedness': 0.4808419004110274}]
# many_objects_1.png
[{'position': {'x': 303.5553171196948, 'y': 178.1833571769194},
  'orientation': 0.4019219108381822, 'roundedness': 0.26867770604637214},
 {'position': {'x': 417.6541193181818, 'y': 241.35700757575756},
  'orientation': -0.7761742875012226, 'roundedness': 0.024360729539394724},
 {'position': {'x': 268.2888660851719, 'y': 257.8768599281683},
```

```
  'orientation': -0.5370733149713873, 'roundedness': 0.48672580916164937},
 ...]
# many_objects_2.png
[{'position': {'x': 130.19451943844493, 'y': 188.1729211663067},
  'orientation': 1.6902675601989725, 'roundedness': 0.5062557354731094},
 {'position': {'x': 265.9125412541254, 'y': 169.65291529152915},
  'orientation': -0.496675233144095, 'roundedness': 0.4806924204306065},
 {'position': {'x': 413.66658366533864, 'y': 204.97684262948206},
  'orientation': 2.022752472102815, 'roundedness': 0.17358105360935486},
 ...]
```

The image results are as follows.



(a) original image (b) binarized image (c) labeled image

(d) original image (e) binarized image (f) labeled image

(g) original image (h) binarized image (i) labeled image

Figure 1: Results of Problem 1

2.2 Problem 2: Circle Detector

The main ideas for the three functions to be completed are as follows.

- `detect_edges`(image): Implement the **Sobel Filter** using convolution. For borders, we use **reflective padding**. The Sobel filters we used are as follows (where I is the image).

$$\texttt{Sobel}_\texttt{x} = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \ \texttt{Sobel}_\texttt{y} = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix},$$

$$G_x(I) = I * \texttt{Sobel}_\texttt{x},\ G_y(I) = I * \texttt{Sobel}_\texttt{y} \implies \text{edge maginitude } G(I) = |G_x(I)| + |G_y(I)|$$

- `hough_circles`(edge_image, edge_thresh, radius_values): Use the edge threshold to determine authentic edge points. For **Hough Transform** implementation, we **choose 200 as the edge threshold, $\{20, 21, ..., 40\}$ as possible radius values.**

- `find_circles`(image, accum_array, radius_values, hough_thresh): Use the Hough threshold to determine authentic circles. Here we **choose 80 as the Hough threshold.**

  Specially, to achieve better results, we **implement NMS (Non-Maximum Suppression)** on the candidates for the radii and positions (r, y, x) of circles, i.e. only preserves circle candidates that are the local maxima in the (r, y, x)-voting accumulator.

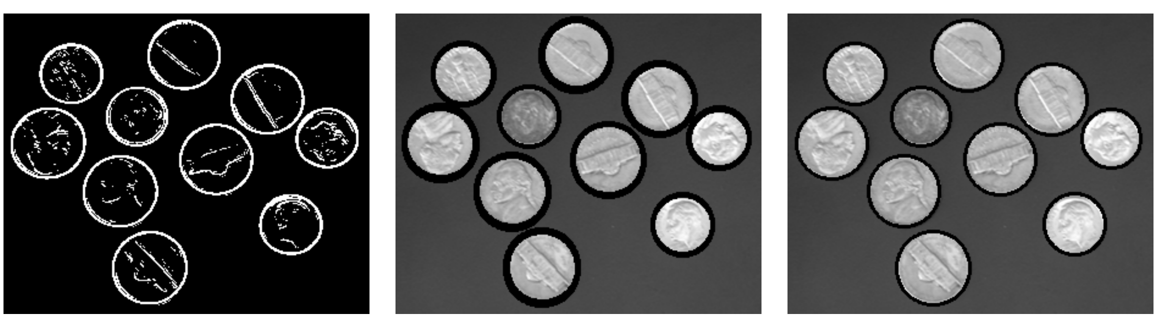
  In the end, we choose to apply **5×5 NMS** since its performance is seemingly the best.

To use my version of `p2_hough_circles.py`, you may use the following instruction.

```
python3 p2_hough_circles.py [name] [edge] [hough] [r_min] [r_max] [nms]
```

where `name`, `edge`, `hough` are respectively the name of the image file, edge threshold and Hough threshold. `r_min` and `r_max` defines possible radius values, i.e. $\{\texttt{r\_min}, \texttt{r\_min} + 1, ..., \texttt{r\_max}\}$. We suppress the non-maxmimum pixels in all $(2 \cdot \texttt{nms} + 1) \times (2 \cdot \texttt{nms} + 1)$ grids. $\texttt{nms} = 2$ is recommended.

The image results are as follows.



(a) edge points (b) circles before NMS (c) circles after NMS **(final result)**

Figure 2: Results of Problem 2

Thus, our program can detect the positions and the radii of circles precisely.