钱币定位系统

一、定位原理概述

1. **图像预处理：** 首先，观察原图的大小来判断是否需要 `resize` ；然后，将图像转换为灰度图。
2. **边缘检测：** 利用 `Canny` 算法进行边缘检测。
3. **圆检测：** 使用 `HoughCircle` 算法进行圆检测。
4. **结果输出：** 将检测到的圆形区域的圆心坐标和半径输出，并可视化标记在原始图像上。

二、模块介绍

（一）核心模块

1. Canny模块

（1）整体流程：

1. **高斯滤波（Gaussian Blur）**：首先对输入的图像进行高斯滤波，以平滑图像并降低噪声。
2. **计算梯度（Get_gradient_img）**：使用一阶差分算子计算图像的梯度幅值和梯度方向。
3. **非极大值抑制（Non_maximum_suppression）**：在梯度图上执行非极大值抑制，将局部最大值之外的所有梯度值抑制为0。
4. **滞后阈值法（Hysteresis Thresholding）**：使用双阈值法来识别强边缘和弱边缘，并连接边缘。在这一步中，像素强度超过高阈值被标记为强边缘，低于低阈值的像素被抑制，而介于两者之间的像素则根据其是否与强边缘相连来确定是否被标记为弱边缘。

（2）各个函数的功能说明：

1. `__init__(self, image: np.ndarray, Guassian_kernel_size: int = 7, sigma: float = 1.0, high_threshold: int = 12, low_threshold: int = 4, sign :bool = False, sobel: bool = False)`

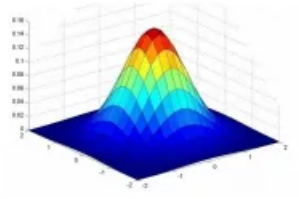
- 功能：初始化Canny算法类，对输入图像进行规范化和参数设置，并调用Canny算法的主要步骤。
- 输入参数：
  - `image`：输入的原始图像，要求为灰度图像。
  - `Guassian_kernel_size`：高斯核尺寸，默认为7。
  - `sigma`：高斯核标准差，默认为1.0。
  - `high_threshold`：滞后阈值法中的高阈值，默认为12。
  - `low_threshold`：滞后阈值法中的低阈值，默认为4。
  - `sign`：判断是否使用较复杂的算子，默认为False。
  - `sobel`：判断是否选择sobel，否则选prewitt，默认为False。

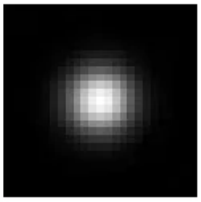
2. `Gaussian_kernel(self) -> np.ndarray`

- 功能：创建指定大小的高斯核。
- 返回值：指定大小的高斯核。

高斯卷积核模板，中心的格子为（0,0），左上角的格子为（-1,-1），将卷积核每个格子的坐标代入高斯函数中，就可以得到每个格子的权值，再进行归一化处理，就得到了我们需要的高斯核。

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





0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

5 x 5, σ = 1

3. `Gaussian_Blur(self) -> np.ndarray`

- 功能：对输入图像进行高斯滤波。
- 返回值：高斯滤波后的图像。

高斯滤波就是用高斯核对图像进行卷积操作，起到去除噪声的效果。

4. `Get_gradient_img(self) -> np.ndarray`

- 功能：选择算子，计算图像的梯度幅值和梯度方向。
- 返回值：梯度图像。

图像的边缘是由于像素值的突变引起的，因此我们需要通过求梯度强度和方向来获取到突变发生的位置，从而检测到边缘特征点。梯度表示了信号变化的大小和方向，梯度值越强，表明该点越有可能是边，梯度方向与边的方向垂直。

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1,y) - f(x,y)}{1}$$

a. 由上面这个公式就可以自然地得出一个简单的算子

`x_kernel = np.array([[-1, 1]])`

`y_kernel = np.array([[-1], [1]])`

b. 再引入两个近似求导的算子

Prewitt: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

Sobel: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

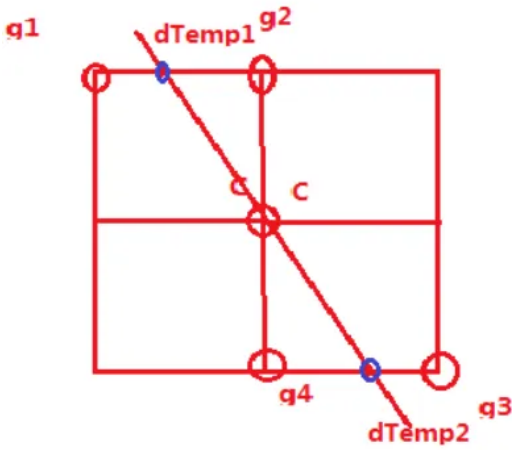
用算子分别求得水平和垂直方向的梯度图后，再用公式 $\|\nabla f\| = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$ 得到梯度图。

5. `Non_maximum_suppression(self) -> np.ndarray`

- 功能：对梯度图进行非极大值抑制。

- 返回值：非极大值抑制后的图像。

我们希望图像中的每个像素点都是在梯度方向上的最大值，因此我们找到梯度方向上离该点最近的两个点，比较他们的值，如果该点的梯度强度不是三个点中最大的，那么就舍弃该点。



但我们发现最近的两个点可能实际上并不存在，因为像素点是离散分布的。为了求出不存在的点的梯度值，我们采用线性插值的方法， `weight` 的值与梯度方向有关。

```
dTemp1 = weight*g1 + (1-weight)*g2;
dTemp2 = weight*g3 + (1-weight)*g4;
```

6. `Hysteresis_thresholding(self) -> np.ndarray`

- 功能：对非极大值抑制后的图像进行滞后阈值法。
- 返回值：滞后阈值法处理后的图像。

像素点梯度值超过高阈值的，认定为强边缘，赋值为255；低于低阈值的，认为不是边缘，赋值为0；其余的像素点为弱边缘，对于每个弱边缘，找到该点八邻域像素点，若周围像素点存在强边缘，则认定该点也为强边缘，否则不是边缘。

7. `canny_algorithm(self) -> np.ndarray`

- 功能：按照顺序和步骤调用以上所有成员函数，完成Canny算法的全部流程

（3）参数对结果影响的分析

1. `kernel_size & sigma`

- 核大小指的是高斯核的尺寸，通常用奇数尺寸来保证中心有一个明确的像素点。核大小决定了滤波器覆盖的区域范围，核越大，考虑的邻近像素越多，平滑效果也越明显。但是，如果核过大，可能会导致图像过度模糊，丧失重要的细节信息
- 标准差σ决定了高斯函数的形状，即权重分布的集中程度。σ值越大，权重分布越平坦，边缘像素对中心像素的影响越大，从而使得滤波后的图像更加平滑。相反，σ值越小，权重集中在中心像素，滤波效果越轻微。
- **对最终结果的影响**：在处理具有高频细节的图像时，可能需要选择较小的核和σ值以避免过度模糊；而在处理噪声较多的图像时，则可能需要较大的核和σ值以更好地消除噪声。

2. `high_threshold & low_threshold`

- 高阈值用于确定潜在的边缘像素，即那些具有足够强度被认为是边缘的像素。这些像素通常被称为强边

缘。高阈值的选择决定了边缘检测的灵敏度。如果高阈值设置得太高，可能会导致很多真实的边缘被忽略，从而降低边缘检测的准确性。相反，如果高阈值设置得太低，可能会将一些噪声误判为边缘。

- 低阈值用于确定那些可能与强边缘相连的弱边缘像素。这些弱边缘像素在没有强边缘连接的情况下会被抑制。低阈值的选择影响了边缘的连续性和完整性。如果低阈值设置得太高，可能会导致边缘断裂，失去连续性。如果低阈值设置得太低，可能会导致噪声被错误地连接到边缘上，从而产生不完整的边缘。
- **对最终结果的影响**：a)当高阈值和低阈值都设置得较高时，只有那些强度非常高的边缘会被检测出来，有时候甚至会得到一张全黑的图片。b)当高阈值设置得较低而低阈值设置得较高时，可以检测到更多的边缘，包括一些较弱的边缘。c)当高阈值和低阈值都设置得较低时，边缘检测可能会非常敏感，不仅会检测到所有的边缘，还可能将噪声误判为边缘，导致边缘检测的结果非常杂乱。
- **Tips**：由于算子选择和原始图像不同，所谓的高阈值和低阈值并不是指绝对的数值，而是相对于特定图像的梯度值分布而言的。这些阈值是为了从图像中提取出真正的边缘而设置的相对标准，可能对于这张图12就是较高的高阈值，而对另一张图100才算较高。

3. 算子的选择

a) Sobel算子：

- Sobel算子是一种近似高斯函数的梯度算子，它通过两个3x3的核分别检测水平和垂直方向的边缘。
- 它对边缘的方向响应较为敏感，能够较好地区分水平和垂直边缘。
- 它对噪声有一定的抑制作用，因为Sobel算子考虑了像素周围的亮度变化，从而减少了噪声引起的误检。
- Sobel算子提取的边缘通常较为清晰和连续，适用于多种类型的图像。

b) Prewitt算子：

- Prewitt算子与Sobel算子类似，也是使用两个3x3的核分别检测水平和垂直方向的边缘。
- Prewitt算子的核是固定的，不考虑像素值的权重，也就是说这些算子对所有像素的影响是相同的，因此对噪声的抑制作用不如Sobel算子。
- 它对边缘的检测结果较为尖锐，但可能会产生更多的噪声点和边缘断裂。
- Prewitt算子适用于边缘对比度较高的图像，但在噪声较多的图像中可能会产生较多的误检。

c) 最简单的[-1,1]水平和垂直算子：

- 最简单的水平和垂直算子是一阶差分算子，它们分别使用[-1,1]的核来检测水平和垂直方向的边缘。
- 这些算子对边缘的检测非常敏感，能够捕捉到细微的亮度变化，但同时也非常容易受到噪声的影响。
- 由于没有考虑像素周围的权重，这些算子提供的边缘信息可能不够连续，且边缘可能会显得较为粗糙。
- 简单的[-1,1]算子适用于边缘非常清晰且噪声较少的图像，但在实际应用中，通常需要结合其他算法来改善边缘检测的效果。

4. `image`

- 分辨率高的图像，算法的时长会比较长

2. HoughCircle模块

(1) 整体流程

1. **霍夫变换投票（Hough_Vote）**：遍历图像中的每个边缘点，在每个点沿着梯度方向上进行投票，并在投票矩阵中累加相应的投票数。
2. **圆的选择（Select_Circle）**：根据设定的阈值，从投票矩阵中筛选出可能代表圆的候选点，并将其转换为圆心坐标和半径。
3. **非极大值抑制（Non_maximum_suppression）**：针对可能重叠的圆，进行非极大值抑制，通过计算圆心

之间的距离来判断是否重叠，若重叠则取平均值作为最终的圆。

（2）各个函数的功能说明：

1. `__init__(self, image: np.ndarray, angle: np.ndarray, step: int = 4, threshold: int = 50)`

- 功能：初始化Hough圆形检测算法类，设置参数空间大小和阈值，并调用Hough算法的主要步骤。
- 输入参数：
  - `image`：Canny算法得到的边缘图。
  - `angle`：Canny算法中计算得到的梯度方向矩阵。
  - `step`：变换步长大小，默认为4。
  - `threshold`：筛选单元的阈值，默认为50。

2. `Hough_Vote(self) -> np.ndarray`

- 功能：沿梯度方向对参数空间中的所有单元进行投票，注意投票过程分为正向和负向。
- 返回值：投票矩阵。

针对每一个边缘点，沿着其梯度的正向和反向，进行圆心的投票，对于每一个点都会得到一组圆心的集合。由于我们设置了步长，因此这个投票实际上是对立方体空间的投票。

3. `Select_Circle(self) -> list`

- 功能：从投票矩阵中筛选出大于阈值的圆的候选点。
- 返回值：候选圆列表。

选择票数大于阈值的所有圆，作为候选圆。

4. `Non_maximum_suppression(self) -> list`

- 功能：对可能重叠的圆进行非极大值抑制，将重叠的圆的圆心和半径取平均值作为最终的圆。
- 返回值：结果圆列表。

对于同一个圆，我们可能得到多个圆心结果，这些圆心相近，我们取平均值合并它们。

5. `hough_algorithm(self) -> list`

- 功能：按照算法顺序调用以上成员函数，完成Hough圆形检测算法的全部流程。
- 返回值：圆形拟合结果，即圆的坐标及半径集合。

通过以上函数，实现了Hough圆形检测算法的各个步骤，从而得到最终的圆形拟合结果。

（3）参数分析

1. `step`

a)较大的步长

- 可以降低参数空间的大小，从而减少需要计算和存储的单元格数量。这可以降低算法的计算复杂度和内存使用，但可能会牺牲检测的准确性。
- 可以通过降低参数空间的分辨率来提高算法对噪声的抗性。较小的噪声变化不太可能影响到不同参数空间单元的投票结果，从而减少了误检的可能性。

b）较小的步长

- 较小的步长可以提高检测结果的精度，因为它允许算法更细致地搜索可能的圆。然而，这也可能导致过拟合，即算法可能会检测到噪声或非圆形结构中的假圆。

`step` 参数影响算法能够检测到的圆的最小尺寸。较小的 `step` 值意味着算法可以检测到更小的圆，而较大的 `step` 值可能会导致一些小圆无法被检测到，需要根据具体的情况来选择值。如果图像中的圆非常小且噪声较多，可能需要选择较小的 `step` 值；而如果图像中的圆较大且计算资源有限，可能需要选择较大的 `step` 值。

2. `threshold`

- 如果阈值设置得过高，只有那些在投票矩阵中获得大量投票的圆会被选为候选圆。这可能会导致一些小的或弱的圆被忽略，因为它们可能没有积累足够的投票数。这样的设置可以减少噪声引起的误检，但也可能漏掉一些真实的圆。
- 如果阈值设置得过低，那么几乎所有在投票矩阵中获得投票的圆都会被选为候选圆，包括那些由于噪声或图像细节而偶然获得投票的圆。这会导致大量的误检，使得圆检测的结果变得不可靠。低阈值可能无法区分真实的圆和噪声，从而降低了检测的准确性。
- 适当的阈值选择应该能够平衡检测的灵敏度和特异性。图像的特性，如噪声水平、圆的大小和强度，都会影响阈值的选择。例如，在噪声较多的图像中，可能需要设置较高的阈值来抑制噪声；而在圆较小或较弱的图像中，可能需要设置较低的阈值以避免错过重要的圆。

（二）模块整合

钱币定位模块

函数功能说明

1. `def __init__(self, origin_image: np.ndarray, image: np.ndarray, Gaussian_kernel_size: int = 7, sigma: float = 1.0, high_threshold: int = 12, low_threshold: int = 4, step: int = 4, threshold: int = 50, sign :bool = False, sobel: bool = False)`

- 功能：初始化 `CoinDetectionSystem` 类，定义算法参数
- 参数：

  `origin_image: np.ndarray`： 用于显示拟合结果的resize后的原图。

  `image: np.ndarray`： 用于执行算法的灰度图。

  .....其他参数已经描述过了
- 属性：

  `canny_detector: Canny`： Canny边缘检测器对象。

  `hough_circle_detector: Hough_Circle`： Hough圆检测器对象。

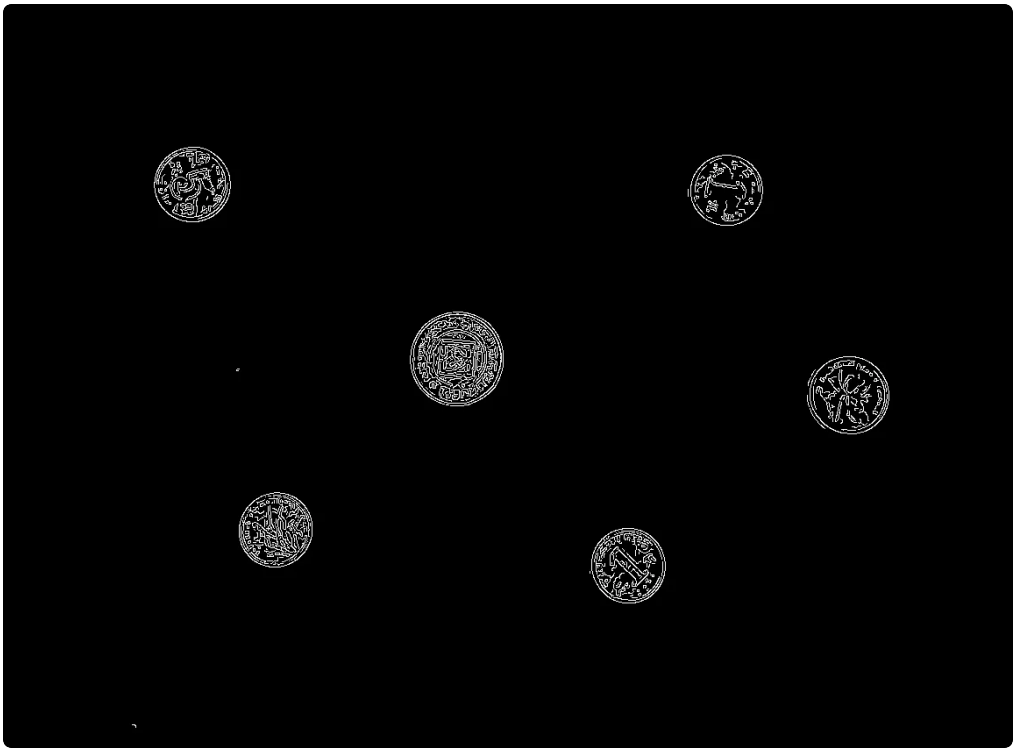
2. `def plot_coins_fitting(self)`

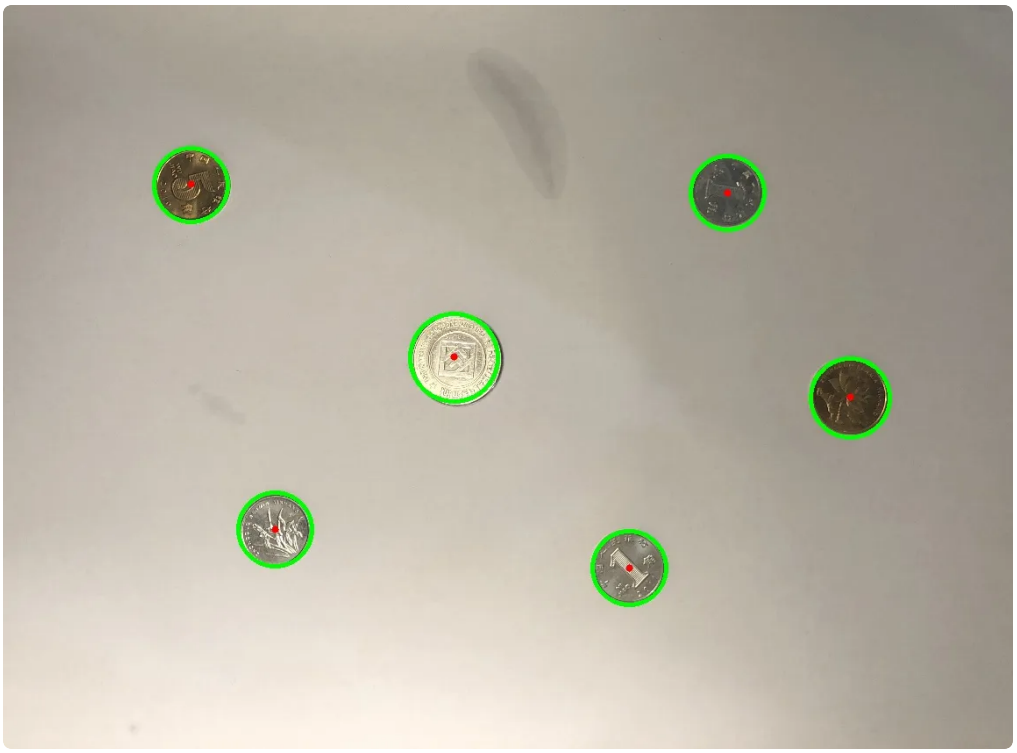
- 功能：将检测到的圆形拟合结果绘制在原图上。

三、实验结果展示

`k=3,sigma=1,使用prewitt算子`

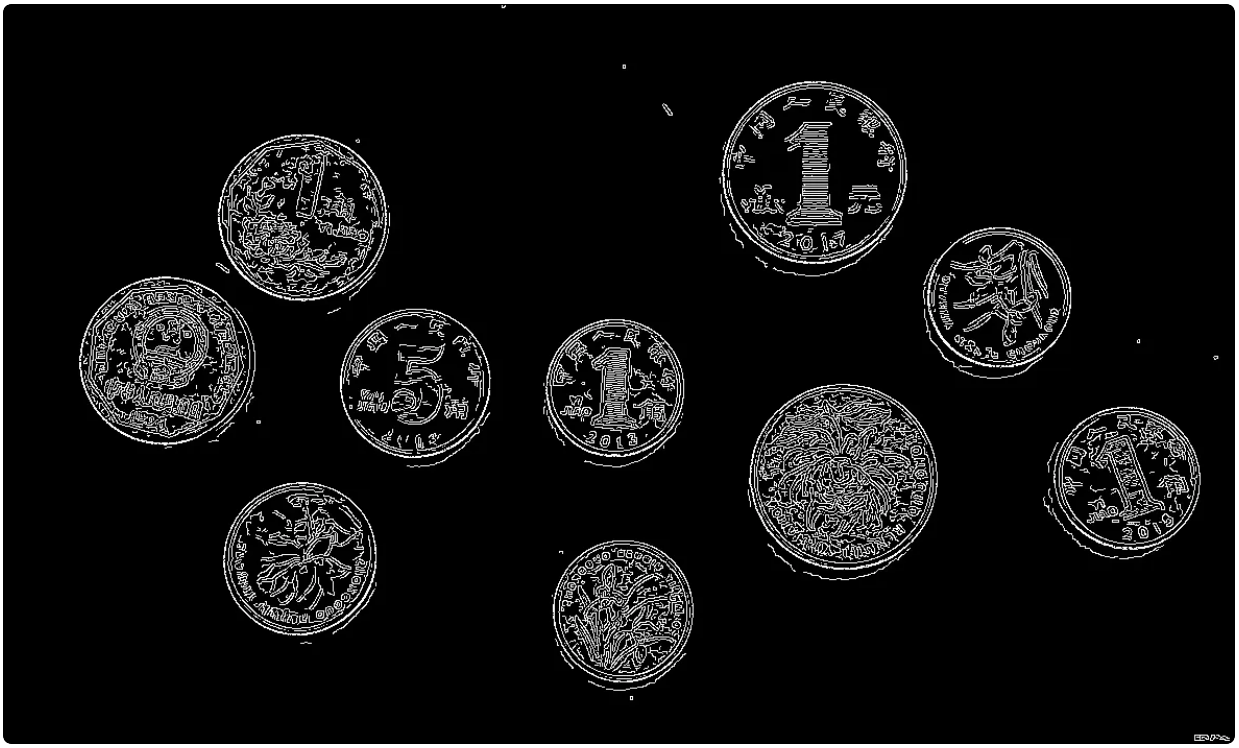
（一）coin1

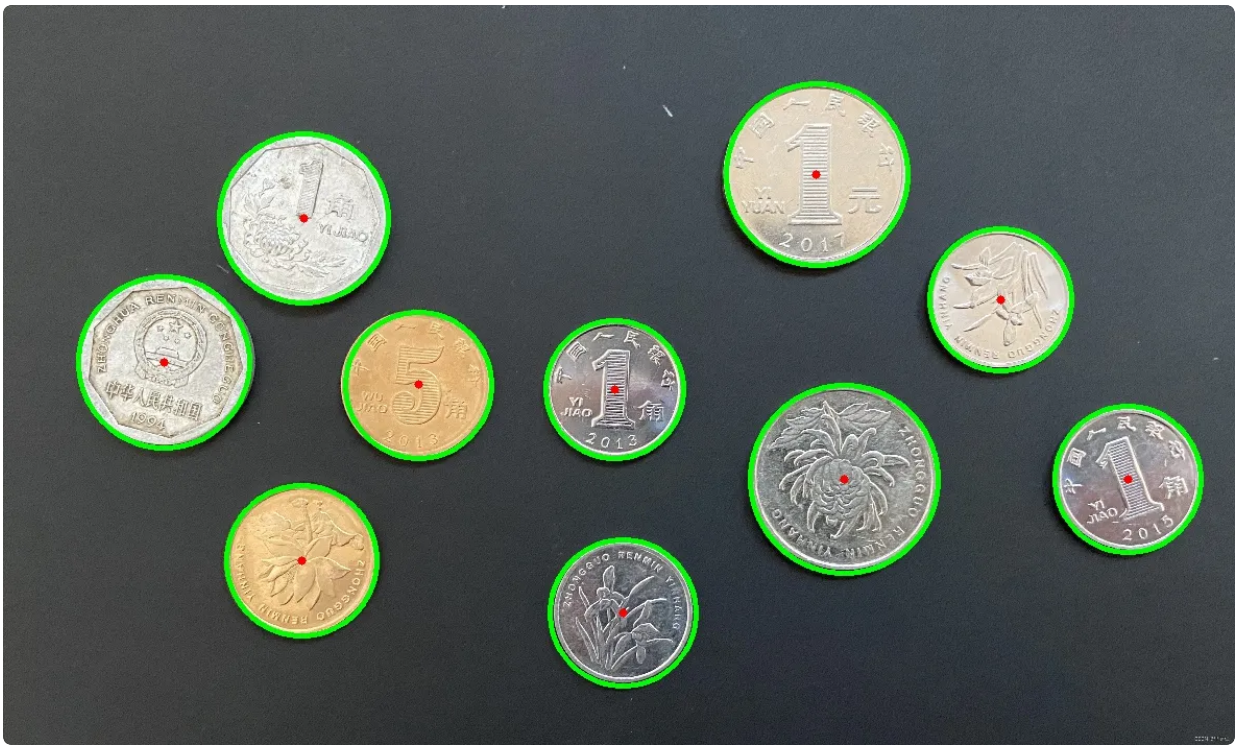




```
圆心坐标: (214.0, 204.4), 半径: 42.0
圆心坐标: (310.0, 598.0), 半径: 42.0
圆心坐标: (514.0, 401.3333333333333), 半径: 50.0
圆心坐标: (714.0, 642.0), 半径: 42.0
圆心坐标: (826.0, 214.0), 半径: 42.0
圆心坐标: (966.0, 447.3333333333333), 半径: 44.666666666666664
```

（二）coin2





```
圆心坐标: (154.0, 342.0), 半径: 82.0
圆心坐标: (286.0, 532.0), 半径: 72.0
圆心坐标: (288.0, 204.0), 半径: 81.0
圆心坐标: (398.0, 363.3333333333333), 半径: 70.0
圆心坐标: (586.0, 368.0), 半径: 66.0
圆心坐标: (594.0, 582.0), 半径: 70.0
圆心坐标: (779.6, 162.0), 半径: 86.8
圆心坐标: (806.0, 454.0), 半径: 90.0
圆心坐标: (956.0, 282.0), 半径: 68.0
圆心坐标: (1078.0, 454.0), 半径: 70.0
```