CSC 482
Assignment 2
Hithesh Shanmugam

Implement a Hough detector for circles.

1. **Edge Detection**

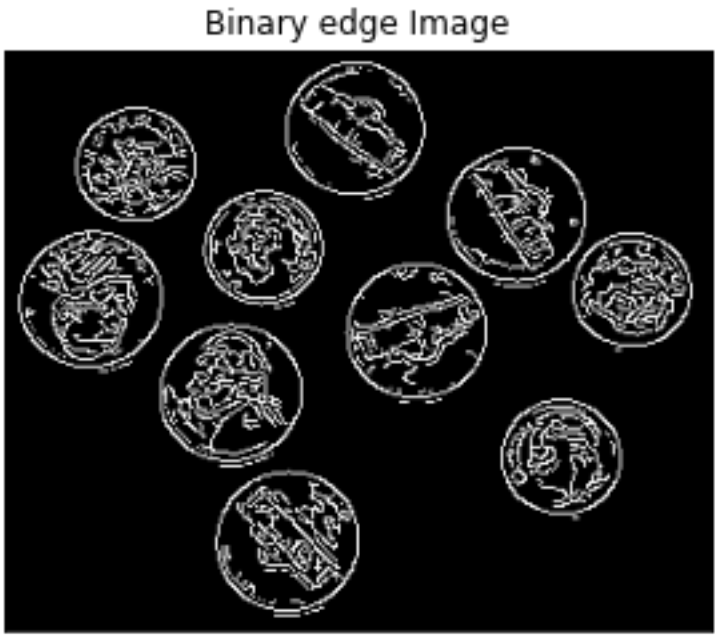
Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
image=cv2.imread("C:/Users/sures/OneDrive - DePaul University/Desktop/coins.png")
def edge_detection(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(gray, 50, 150)
    _, binary = cv2.threshold(edges, 127, 255, cv2.THRESH_BINARY)
    return binary
binary=edge_detection(image)
```

```
plt.imshow(binary,cmap='gray')
plt.xticks([]),plt.yticks([])
plt.title('Binary edge Image')
plt.show()
```

Output:



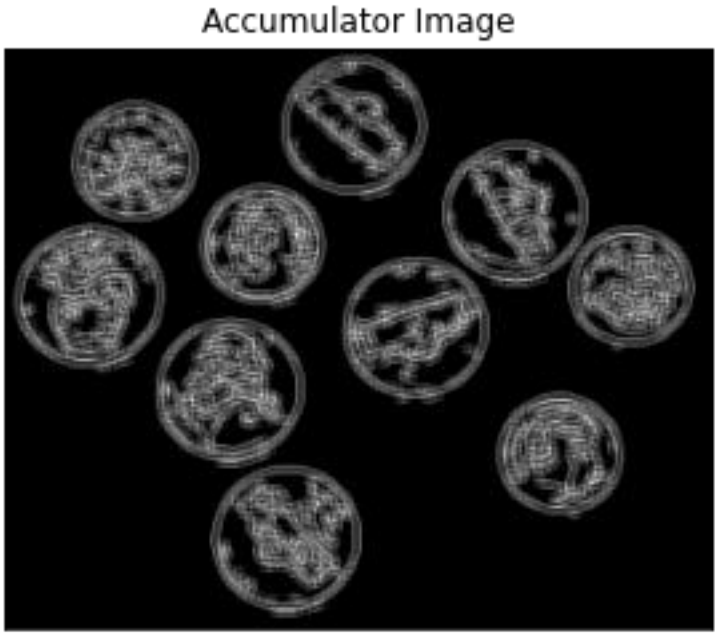
2. **Create and populate the accumulator**

Code:

```
def hough_circles_accumulator(binary_image, radius):
    # Create accumulator with dimensions (image height, image width, radius)
    height, width = binary_image.shape[:2]
    accumulator = np.zeros((height, width, radius))

    # Find all edge points in the binary image
    edge_points = np.column_stack(np.where(binary_image > 0))

    # Iterate over edge points and vote in the accumulator
    for x, y in edge_points:
        for r in range(radius):
            for theta in range(360):
                a = int(x - r*np.cos(np.deg2rad(theta)))
                b = int(y - r*np.sin(np.deg2rad(theta)))
                if a >= 0 and a < height and b >= 0 and b < width:
                    accumulator[a, b, r] += 1

    # Return the accumulator
    return accumulator
```

```
accumulator=hough_circles_accumulator(binary,10)
# Choose a radius to visualize
fixed_radius = 2

# Get the accumulator at the fixed radius
accumulator_slice = accumulator[:, :, fixed_radius]

# Display the accumulator slice
plt.imshow(accumulator_slice, cmap='gray')
plt.xticks([]),plt.yticks([])
plt.title('Accumulator Image')
plt.show()
```

Output:



3. **Plot circles on your original image**

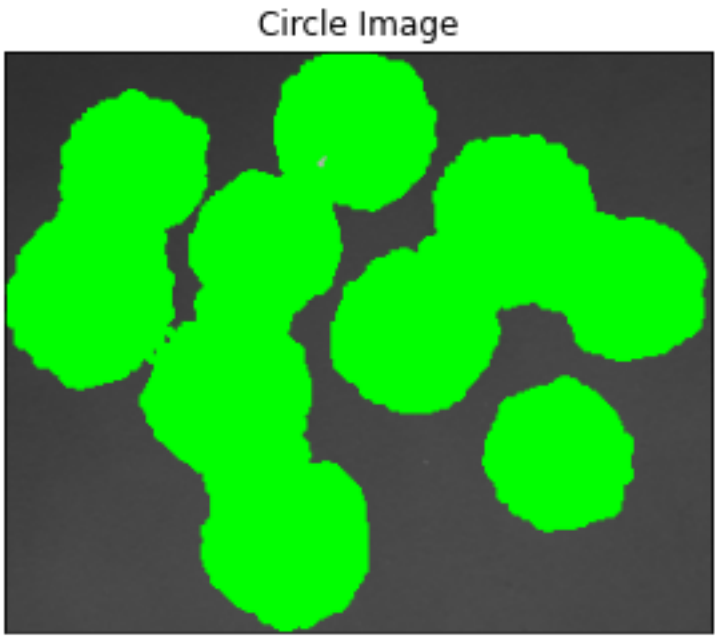
Code:

```
def draw_circles(original_image, accumulator, threshold):
    # Find maxima in the accumulator
    idx = np.where(accumulator >= threshold)
    circles = set(zip(idx[0], idx[1], idx[2]))
    # Draw circles on the original image
    for x, y, r in circles:
        cv2.circle(original_image, (y, x), r, (0, 255, 0), 2)
    # Return the original image with circles drawn on it
    return original_image
```

```
# Draw circles on the original image
threshold = 100
circled_image = draw_circles(image, accumulator, threshold)
```

```
plt.imshow(circled_image,cmap='gray')
plt.xticks([]),plt.yticks([])
plt.title('Circle Image')
plt.show()
```

Output:



4. **Non-integral accumulator**

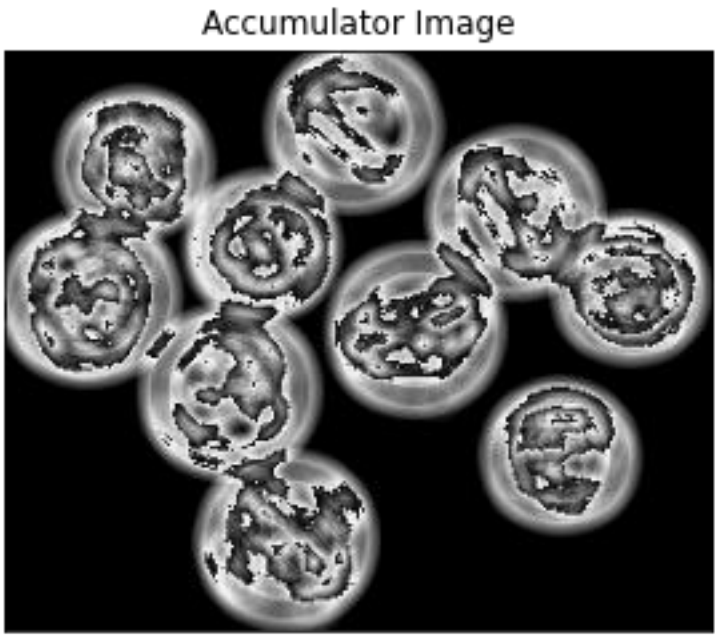
Code:

```
import math
def detect_circles(image, min_radius, max_radius):
    # Apply edge detection
    edges = cv2.Canny(image, 100, 200)

    # Create the accumulator with the same size as the image
    accumulator = np.zeros_like(image)

    # Loop over all pixels in the edge image
    for x in range(image.shape[0]):
        for y in range(image.shape[1]):
            # If the pixel is part of an edge
            if edges[x, y] > 0:
                # Loop over all possible radii
                for r in range(min_radius, max_radius + 1):
                    # Loop over all angles
                    for theta in range(360):
                        a = int(x + r * math.cos(math.radians(theta)))
                        b = int(y + r * math.sin(math.radians(theta)))
                        if a >= 0 and a < image.shape[0] and b >= 0 and b < image.shape[1]:
                            accumulator[a, b] += 1

    return accumulator
```

```
image=cv2.imread("C:/Users/sures/OneDrive - DePaul University/Desktop/coins.png")
accumulator_non_integral=detect_circles(image,5,10)
```

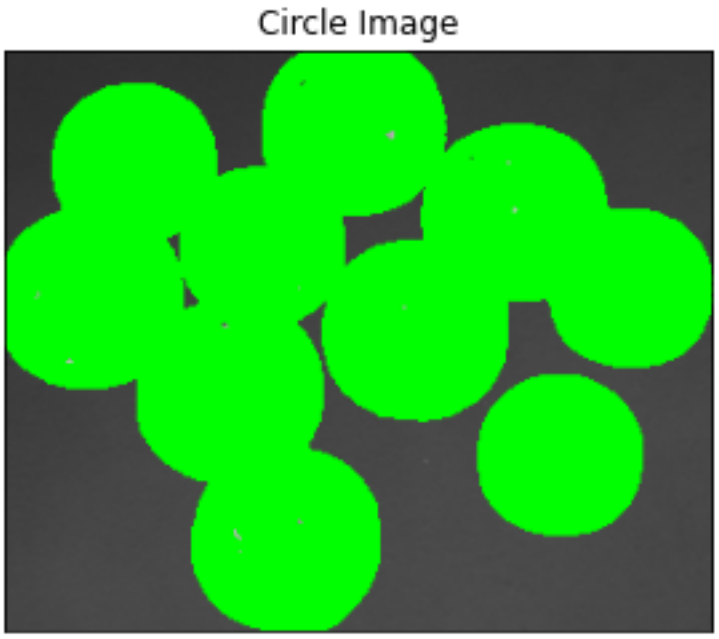
```
# Choose a radius to visualize
fixed_radius = 2

# Get the accumulator at the fixed radius
accumulator_slice = accumulator_non_integral[:, :, fixed_radius]

# Display the accumulator slice
plt.imshow(accumulator_slice, cmap='gray')
plt.xticks([]),plt.yticks([])
plt.title('Accumulator Image')
plt.show()
```

Output:



```
circled_image = draw_circles(image, accumulator_non_integral, threshold)
plt.imshow(circled_image,cmap='gray')
plt.xticks([]),plt.yticks([])
plt.title('Circle Image')
plt.show()
```

Output:



Explanation:

The major difference between binary edges and real-valued edges what I learnt is that binary edges just take into account an edge's presence or absence, whereas real-valued edges also take into account an edge's strength. This implies that the real-valued edges may distinguish between strong and weak edges, which may have an impact on how well the edge detector works.

This is clearly seen in the images too we can see more clear circles in the real valued edges are taken into the accumulator and also the accumulator image is both the same slice (radius=2, threshold=100). In the real-valued edge the accumulator image coins look like nucleus.

However, it also makes the run-time longer, as the algorithm needs to consider all pixels in the image, rather than just the pixels with an edge in the binary case.

5. **Polar coordinates:**

   Explanation:
   The Hough Transform is a technique for identifying shapes in images that searches for high peaks in a parameter space representation of the potential shape instances that correlate to real shape instances in the image. With regard to circle detection, two parameters—the circle's center (x, y) and radius—are utilized to represent circles in the parameter space using the usual form of a circle.

   The polar form of a circle, (r cos θ−a) ^2 + (r sin θ−b) ^2 = r^2, is a trigonometric parameterization of a circle where (a, b) is the center of the circle and r is its radius. However, this form of a circle still requires a search over two parameters: the radius and the center of the circle. Therefore, using the polar form of a circle in the Hough Transform for circle detection doesn't change the fundamental computational issues of the method, such as the need for a two-dimensional search over the radius and center parameters, the large number of possible circle instances, and the difficulty of dealing with degenerate cases such as small circles or overlapping circles.

   Additionally, converting from image coordinates to polar coordinates incurs additional computing costs and offers little advantages over the traditional form of a circle in terms of robustness or accuracy.

   Conclusion: The standard form of a circle is still preferable in this method, despite the fact that the polar form of a circle is sometimes a beneficial parameterization for circle detection in the Hough Transform.