Hough Transform Implementation for Circle Detection

Contents

1 Introduction

This document provides a comprehensive overview and detailed explanation of a Hough Transform implementation designed for detecting circles in images. The documentation is divided into two main parts: first, a discussion of the theoretical architecture of the Hough Transform for circle detection; and second, a step-by-step code breakdown that explains each component of the implementation.

2 Architecture of the Hough Transform for Circle Detection

The Hough Transform is a technique used for detecting geometric shapes within images by transforming points in image space into a parameter space. For circle detection, the essential concepts are:

- **Parameter Space:** A circle in a two-dimensional image can be defined by its center coordinates (a, b) and its radius r. In this implementation, the radius is assumed to be known, so the focus is on determining (a, b).
- **Accumulator Array:** An accumulator (a 2D array) is used to count votes for each potential circle center. Each edge pixel votes for all possible centers that could form a circle with the given radius.
- **Quantization:** The angle θ is discretized from 0° to 360° using a fixed step size (θ_{step}). This quantization reduces the continuous parameter space into discrete bins for practical computation.
- **Thresholding:** After voting, a dynamic threshold is applied (based on a fraction of the maximum vote count). Candidate centers with vote counts exceeding this threshold are returned as detected circle centers.

3 Detailed Code Breakdown

3.1 Function: `my_convolve2d`

Purpose: Performs a 2D convolution between a grayscale image and a kernel, with zero padding.
Steps:

1. **Template Flipping:** The kernel is flipped horizontally and vertically.
2. **Zero Padding:** The image is padded based on the kernel size.
3. **Convolution:** The kernel is slid over every pixel of the image, and the weighted sum is computed.
4. **Output:** Returns the convolved image.

```
def my_convolve2d(image, temp):
    """
    Perform a 2D convolution between an image and a template.
    Implements convolution with zero padding.
    """
    # Flip the template (convolution operation)
    temp = np.flipud(np.fliplr(temp))

    # Get image and template dimensions
```

```
    iH, iW = image.shape
    kH, kW = temp.shape

    # Calculate padding size
    pad_h = kH // 2
    pad_w = kW // 2

    # Pad the image with zeros
    padded_image = np.pad(image, ((pad_h, pad_h), (pad_w, pad_w)),
        mode="constant", constant_values=0)

    # Initialize the output image
    output = np.zeros((iH, iW), dtype=np.float64)

    # Perform convolution
    for i in range(iH):
        for j in range(iW):
            region = padded_image[i:i+kH, j:j+kW]
            output[i, j] = np.sum(region * temp)

    return output
```

Listing 1: Function: my_convolve2d

3.2 Function: `hough_circle`

Purpose: Detects circle centers by applying the Hough Transform to an edge-detected image.
Steps:

1. **Initialization:**
   - Create an accumulator array with the same dimensions as the input image.
   - Discretize the angle space from 0° to 360° with a given step size (θ_{step}); convert these angles to radians.
2. **Voting:**
   - Iterate over each edge pixel.
   - For each pixel and for each quantized angle θ, compute the candidate center coordinates using:
   $$a = x - r\cos(\theta), \quad b = y - r\sin(\theta)$$
   - Update the accumulator if the computed center falls within the image bounds.
3. **Thresholding:** Determine the maximum vote count, compute a dynamic threshold (`vote_threshold` = `vote_threshold_ratio` × maximum votes), and select the centers where votes exceed this threshold.

```
def hough_circle(canvas, r, vote_threshold_ratio, theta_step):
    H, W = canvas.shape
    accum = np.zeros((H, W), dtype=np.float32)  % Accumulator for
        circle centers
    theta = np.deg2rad(np.arange(0, 360, theta_step))

    # Iterate over every pixel in the edge-detected image
    for y in range(H):
        for x in range(W):
```

```
            if canvas[y, x] > 0:  % Only consider edge points
                for t in theta:
                    a = int(x - r * np.cos(t))
                    b = int(y - r * np.sin(t))
                    if 0 <= a < W and 0 <= b < H:
                        accum[b, a] += 1

    % Determine threshold and detect centers
    max_votes = np.max(accum)
    vote_threshold = vote_threshold_ratio * max_votes
    centers = np.argwhere(accum >= vote_threshold)

    return centers, accum
```

Listing 2: Function: hough_circle

3.3 Function: `imshow`

Purpose: Displays the image using Matplotlib, ensuring that data is properly formatted for visualization.

Steps:

1. Verify that the image is not `None`; if it is, raise an error.
2. Convert and clip the image to an 8-bit unsigned integer type if it is not already.
3. Display the image with an optional title while removing axis ticks.

```
def imshow(image, title=None):
    if image is None:
        raise ValueError("Error: The image is None. Please check the
            input image.")

    # Convert image to uint8 if necessary
    if image.dtype != np.uint8:
        image = np.clip(image, 0, 255).astype(np.uint8)
    plt.imshow(image, cmap='gray')
    if title:
        plt.title(title)
    plt.axis('off')
    plt.show()
```

Listing 3: Function: imshow

3.4 Main Workflow

Description: The main script ties together the individual functions to perform circle detection:

1. **Image Loading:** The grayscale image is loaded from a specified path.
2. **Edge Detection:** The Canny edge detector is applied to the loaded image to generate an edge map.
3. **Parameter Setup:** Parameters such as the known circle radius (r), vote threshold ratio, and angle step (θ_{step}) are defined.
4. **Hough Transform Application:** The `hough_circle` function is called with the edge map and parameters. It returns the detected centers and the accumulator array.

5. **Visualization:**
    - The detected circle centers are printed and overlaid on the original image.
    - The accumulator is displayed using a heatmap (hot colormap) to illustrate the vote distribution.

4 Results

4.1 Input Image

The input image used in this implementation is shown in Figure 1.

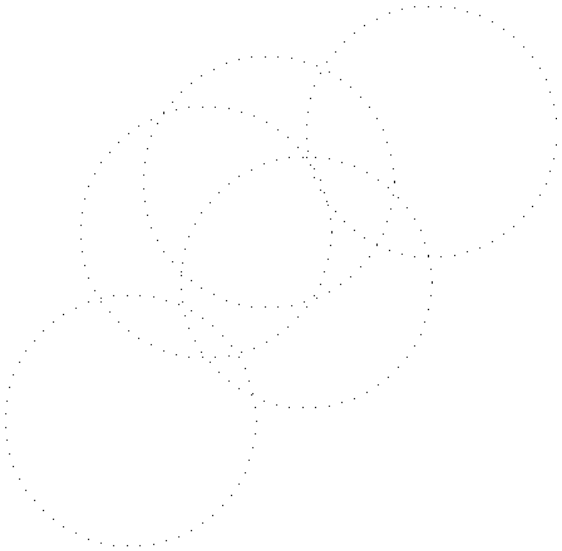

Figure 1: Input Circle Image

4.2 Output Image

The output image, displaying the detected circle centers overlaid on the original image, is shown in Figure 3.

4.3 Accumulator

Figure 3.

5 Conclusion

This document has presented a detailed overview of a Hough Transform-based circle detection algorithm. It explained the architectural concepts, including parameter quantization and accumulator voting, and provided a thorough step-by-step explanation of the code. The results section includes visual examples of both the input image and the output image with the detected circle centers.

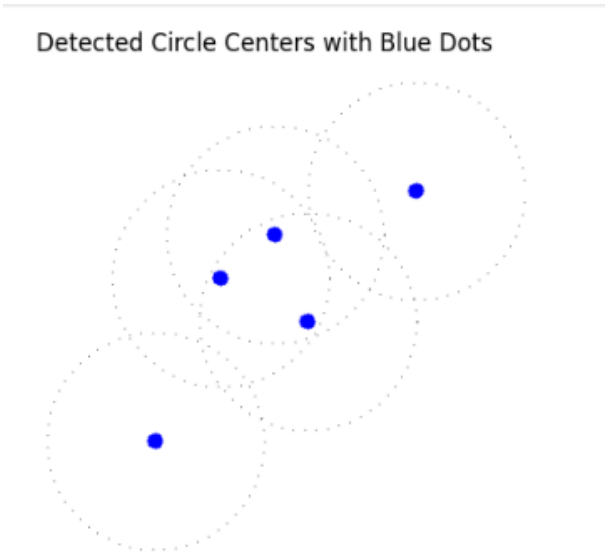

Figure 2: Detected Circle Centers on the Original Image

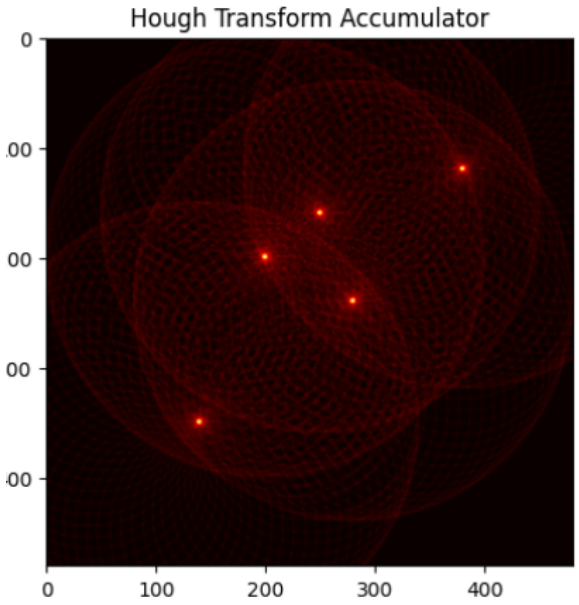

Figure 3: Hough Transform Accumulator