1. Hough transform

1.1 Circular Hough Transform (75%)

The circular Hough Transform can be used to detect circles in edge images. It is very similar to the linear Hough Transform, with the difference that there is one parameter more to take into account: circle radius.

Assume we are given a pair of an original gray-scale image $I[y, x]$ and a corresponding edge image `edge_I`$[y, x]$, of size $H \times W$, such that $1 \leq x \leq W$, and $1 \leq y \leq H$. Every pixel of the edge image has logical value and is equal to 1 if the corresponding pixel of the original image belongs to an edge, and 0 otherwise.

In this exercise you are asked to implement a version of Hough transform for detecting circles. A circle can be represented in the following way:

$$(x - x_c)^2 + (y - y_c)^2 = r^2, \quad 1 \leq x_c \leq W, \quad 1 \leq y_c \leq H, \quad r_{min} \leq r \leq r_{max}, \tag{1.1}$$

where (x, y) are pixel coordinates in the edge image `edge_I`, and (x_c, y_c, r) are the parameters of the model.

If the radius of the circle to be found is fixed, then the parameter space is reduced to 2-d. Each point (x, y) on the original circle (in the image space) defines a circle centered at (x, y) with radius r according to 1.1. The intersection point of all such circles in the parameter space corresponds to the center point (x_c, y_c) of the original circle on the image space.

An accumulator matrix is used to find the intersection point in the parameter space. The parameter space is divided into cells using a grid and the accumulator matrix is constructed based on this grid. Each element, or "vote", in the accumulator matrix denotes the number of circles in the parameter space that pass through the corresponding cell. This matrix is filled by passing through every edge point, formulating a circle in the parameter space and increasing the voting number of the cell through which the circle passes. For a fixed radius, the accumulator array is a 2-dimensional matrix, whose rows and columns correspond to different values of parameters x_c and y_c respectively. For a range of radii, the accumulator becomes 3-dimensional, where the third dimension corresponds to the grid of investigated radii.

After voting, we compute local maxima of the accumulator array. The positions of these maxima correspond to the circle centers (x_c, y_c) in the original image space.

Exercise 1.1 Implement a function
`[accum] = CircleHough(edge_I,r)`
that takes the edge image and a vector of radius ranges as input, and returns the 3-d accumulator array as output.

- To compute the edge image from the gray-scale image you can use the MATLAB function `edge`.
- Find the set of coordinates $\{(x_e, y_e)\}$ of the edge points from `edge_I` $\in \mathbb{R}^{H \times W}$.
- Initialize the accumulator array with zeros. The size of the array is $H \times W \times |r|$.

- Implement a loop that for each radius r in the range does the following:
  - $\forall x \in \{x_e\}$
    Computes left and right boundaries (x_l, x_r) of the circle
    to be investigated from x and r
  - $\forall x_c \in [x_l, x_r]$
    Computes (y_{c1}, y_{c2}) from 1.1
  - $accum(\lceil y_{c1} \rceil, x_c, r) = accum(\lceil y_{c1} \rceil, x_c, r) + 1$
  - $accum(\lceil y_{c2} \rceil, x_c, r) = accum(\lceil y_{c2} \rceil, x_c, r) + 1$
- For each radius value in r, visualize the accumulator array using the `imagesc` function.

■

After voting we need to detect circles by finding local maxima in the accumulator array. This is the subject of the next exercise.

Exercise 1.2 Implement a method
`[x_c,y_c] = DetectCircles(accum,r,thres)`
that takes the accumulator array and the range of radii as input and a threshold. The output will be the coordinates of the detected circles for each radius value.

- Implement a loop that for each radius value r in the range does the following:
  - Computes the locations of the local maxima of the current accumulator array. You can use the function `imregionalmax` for this task.
  - If the computed local maxima are larger than the threshold, then the corresponding locations of these maxima are the centers (x_c, y_c) of the detected circles.

Hint: Using cell arrays for storing `x_c,y_c` maybe helpful. ■

To evaluate the approach we want to visualize the results. We need to plot the circles that got high number of votes in the accumulator array.

Exercise 1.3 Implement a function `ShowCircles(I,r,x_c,y_c)` that takes the original image, the radius range and the detected circle centers from `DetectCircles` as input and displays the circles found.

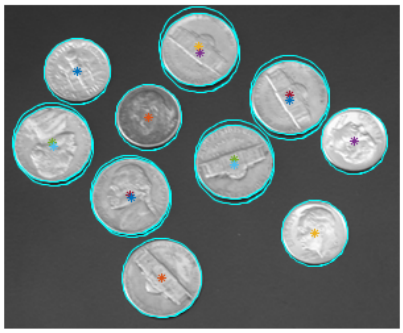
- For each radius r in the range, plot in a separate figure the circles found with this radius.
- Plot all found circles of different radii in the same figure.
  Hint: The set of points on the circle can be written as $(x_c + r * \cos(\theta), y_c + r * \sin(\theta))$

■

Now apply your implementation to the image in Figure 1.1a. In Figure 1.1b we show a representative result of applying the circular Hough transform. Run experiments with different ranges of radius and different thresholds. What are your findings? How does the algorithm's accuracy change with respect to the input parameters?

Answer the questions above as comments in `main.m`.

(a) Coins image.



(b) Coins image with detected circles.

Figure 1.1: Result of applying the circular Hough transform.

1.2 Red Eye Correction (25%)

We will now use the Hough Transform to correct for the red eye effect on the photograph shown below.

We will assume that the Hough Transform has already been applied to this image. The two circles with the highest accumulator values are already loaded in `main.m`, so you can do this exercise without needing the previous one.

The circles contain the red-eye effect. Your task is to attenuate the red-eye effect as much as possible.

Exercise 1.4 For each eye or circle, do the following. Code this in `main.m`.

1. Detect the pixels that lie within the circle.
2. For those pixels, attenuate the red color by dividing its value by a factor α. For example $\alpha = 3$.

Display the resulting image, and answer the following questions as comments in `main.m`:

- How does the result change with the value of α? Why wouldn't you just set the

red value of those pixels to zero?

- How does the Hough Transform radius estimate affect your red eye correction algorithm? What would happen if it was a few pixels off? What about the circle center?

■