2. Hough transform

2.1 Goals

The goal of this exercise is to

- Compute edge images from the real-world images
- Understand and apply a Hough Transform for detecting objects of a specific shape

2.2 Linear Hough Transform

Assume we are given a pair of an original gray-scale image $I[x, y]$ and a corresponding edge image `edge_I`$[x, y]$, of size $W \times H$, 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.

We are interested in detecting the presence of a specific shape in the original image. This is usually a first step in various Computer Vision tasks (detection of road signs, eye detection, etc.). A well known approach to detecting shapes such as lines, circles and ellipses is the Hough transform.

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

$$xcos(\theta) + ysin(\theta) = r, \qquad 0 \leq \theta < \pi \tag{2.1}$$

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

The Hough transform consists of checking every non-zero element of the edge image, computing its vote and storing it in an accumulator array. This array in our case is a 2-dimensional matrix, whose rows and columns correspond to different values of parameters r and θ respectively.

Exercise 2.1 Implement a function `accum = LinearHoughAccum(edge_I)` that takes the edge image as an input and returns the accumulator array as an output. For now you can assume that the image is square, that is, $W = H$.

- To compute the edge image from the gray-scale image you can use the results of the previous exercise or the MATLAB function `edge`.
- Implement a loop that for every non-zero element of the edge image and every value of parameter θ calculates the value of parameter r and increases the corresponding position of the accumulator array by 1.
  - Create an array of $\theta_k : \theta_0 = 0, \;\; \theta_k = \theta_{k-1} + 0.01, \;\; \theta_k, \forall k < \pi$
  - Allocate memory for the accumulator array with sizes: $(\lceil \sqrt{2}W \rceil, N)$, where W is the horizontal dimension of the image I, N is the number of elements in the θ_i and operator $\lceil \cdot \rceil : \mathbb{R} \to \mathbb{Z} : \lceil x \rceil - 1 < x \leq \lceil x \rceil, \forall x \in \mathbb{R}$
  - Fill the accumulator array with zeros
  - Implement a loop that does the following:

$\forall i, j, k$
$r = j * cos(\theta_k) + i * sin(\theta_k)$
$accum(\lceil r \rceil, k) = accum(\lceil r \rceil, k) + 1$

- Visualize the accumulator array using `imagesc` function
- How the dimensions of the accumulator array change if I is not square ($W \neq H$)?
- How can this approach be extended to detecting circles? How many parameters will there be for such model?

■

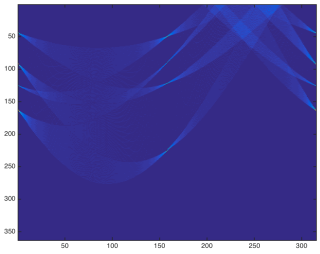
To evaluate the approach we want to visualize the results. We need to plot the line that got the higher number of votes in the accumulator array.

Exercise 2.2 Implement a function `show_lines(I,accum)` that takes the original image and the accumulator array as an input, and displays the line that has the maximum number of votes in the accumulator array. This consists of the following operations:

- Find the maximum value of the accumulator array
  Hint: you can use MATLAB function `max`
- Plot the line on top of the original image using Eq.2.1.

■

Original image



Sample output of the accumulator array from Exercise 2.1

Sample output of the Exercise 2.2