**README FILE:**

***Yogev hassid – 205836927 , Ron azu – 205966781***

The purpose of this exercise is to help you understand the concept of the convolution and edge

detection by performing simple manipulations on images.

This exercise covers:

\_ Implementing convolution on 1D and 2D arrays

\_ Performing image derivative and image blurring

\_ Edge detection

*Submitted files:*

**Ex2\_main.py:**

All the program will run in this file with test functions that we wrote.

This file runs all the functions we need to execute, present each image and compare our results to cv2 results.

**Ex2\_utils.py:**

All the code should be in this file.

Ex2\_main.py import this file for execute the functions in it.

I added side functions for some main functions for simplifying the exercise.

In addition, each function has notes that indicate what I performed at each step.

*Functions:*

**Convolution**

conv1D **:** implementation convolution of 1D discrete signal.

Conv2D : implementation convolution of 2D discrete signal

**Note:** The result of conv1D should match np.convolve(signal, kernel, 'full') and conv2D should match cv2.\_lter2D with option 'borderType'=cv2.BORDER REPLICATE.

**Image derivatives & blurring**

convDerivative: function that computes the magnitude and the direction of an image gradient. You should derive the image in each direction separately (rows and column) using simple convolution with [1; 0;-1]T and [1; 0;-1] to get the two image derivatives. Next, use these derivative images to compute the magnitude and direction matrix and also the x and y derivatives.

blurImage1: blurImage1 should be fully implemented by our self, using your our implementation of convolution and Gaussian kernel.

blurImage2: blurImage2 should be implemented by using pythons internal functions:

flter2D and getGaussianKernel.

**Edge detection**

Each function implements edge detections according to a different method.

edgeDetectionSobel: blurring on one direction and derivative on the second direction.

edgeDetectionZeroCrossingSimple/LOG: convolution with laplacian of Gaussian kernel and look for patterns.

edgeDetectionCanny: smooth the image with a Gaussian kernel, compute the partial derivatives Ix,Iy , compute magnitude and cirection of the gradient, quantize the gradient directions, perform non-maximum suppression, for each pixel compare to pixels along its gradient direction.

If the magnitude of the pixel is not a maximum, set it to zero , Define two thresholds T1 > T2

- Every pixel with |G(x,y)| greater than T1 is presumed

to be an edge pixel.

- Every pixel which is both

(1) connected to an edge pixel, and

(2) has |G(x,y)| greater than T2

is also selected as an edge pixel.

**Hough Circles**

houghCircle**:**

1. For each A[a,b,r] = 0;
2. Process the filtering algorithm on image Gaussian Blurring, convert the image to grayscale ( grayScaling), make Canny operator, The Canny operator gives the edges on image.
3. Vote the all possible circles in accumulator.
4. The local maximum voted circles of Accumulator A gives the circle Hough space.
5. The maximum voted circle of Accumulator gives the circle.