INTRODUCTION TO DIGITAL IMAGE PROCESSING
361.1.4751

EXERCISE 4 - Edges and Feature Descriptors

Submission Date: 24.12.18

Introduction

In this assignment we will learn about edge detection and feature extraction. Although MATLAB has many built in functions, our goal is to learn image processing through doing things ourselves. Please avoid using built-in MATLAB functions unless clearly stated otherwise. However, feel free to compare your own functions to the built-in functions.
Before you start, please read the submission instructions at the end of the exercise and follow them during your work. For any questions regarding this assignment please refer to the course forum on the moodle web site, for personal questions **only** please email hgazit@post.bgu.ac.il , davidit@post.bgu.ac.il , royshau@post.bgu.ac.il or arbellea@post.bgu.ac.il .

1 Edge Detection

In this section we will play with some famous edge detectors.

1.1 Reading the Image

Read the image named *tire.tif*. Normalize the image between $[0, 1]$. This image is built-in into MATLAB, you can simply read it by writing the line *imread('tire.tif')*. For those of you who will not be able to do it, the image is added to the exercise file.

1.2 Roberts Edge Detector

The Roberts edge detector is one of the first edge detectors invented. Its derivative kernels are defined as:

$$G_{Rx} = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, G_{Ry} = \frac{1}{2}\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

1. Write your own function named *dip_roberts_edge(img,thresh)* that will apply the Roberts edge detector on *'img'* and output the edge image with the same size as the input image. The *'thresh'* parameter will determine the gradient magnitude cutoff threshold. Note: You don't have to deal with the image boundaries, you can use the *conv2()* function with the parameter *'same'*.

2. Display 3 edge images generated using *dip_roberts_edge(img,thresh)* with 3 different thresholds. Which of the edge images do you think represents the edges in the original image in the best manner? Why?

1.3 Prewitt Edge Detector

The Prewitt edge detector was developed to solve some of the problems of the Roberts detector. Its derivative kernels are defined as:

$$G_{Px} = \frac{1}{6}\begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}, G_{Py} = \frac{1}{6}\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}$$

1. Write your own function named *dip_prewitt_edge(img,thresh)* that will apply the Prewitt edge detector on *'img'* and output the edge image with the same size as the input image. The *'thresh'* parameter will determine the gradient magnitude cutoff threshold. Note: You don't have to deal with the image boundaries, you can use the *conv2()* function with the parameter *'same'*.

2. Display 2 edge images generated using *dip_prewitt_edge(img,thresh)* with 2 different thresholds.

3. Add *'gaussian'* noise to the image using *imnoise()*.

4. Apply both *dip_roberts_edge(img,thresh)* and *dip_prewitt_edge(img,thresh)* on the noisy image (using $thresh = 0.1$) and show the results.

5. What is the main disadvantage of the Roberts edge detector and how does the Prewitt edge detector solve it?

1.4 Modified Prewitt Edge Detector

We define a modified Prewitt edge detector. Its derivative kernels are defined as:

$$G_{MPx} = \frac{1}{9}\begin{pmatrix} -1 & 0 & 0 & 1 \\ -1 & 0 & 0 & 1 \\ -1 & 0 & 0 & 1 \end{pmatrix}, G_{MPy} = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}$$

1. Write your own function named *dip_prewitt_edge_mod(img,thresh)* that will apply the modified Prewitt edge detector on *'img'* and output the edge image with the same size as the input image. The *'thresh'* parameter will determine the gradient magnitude cutoff threshold. Note: You don't have to deal with the image boundaries, you can use the *conv2()* function with the parameter *'same'*.

2. Compare the performance of the modified Prewitt detector to the original Prewitt detector on the noisy image from the previous section. Display the images and explain the reason for the difference.

3. What is the disadvantage of using the modified Prewitt edge detector?

1.5 Canny Edge Detector

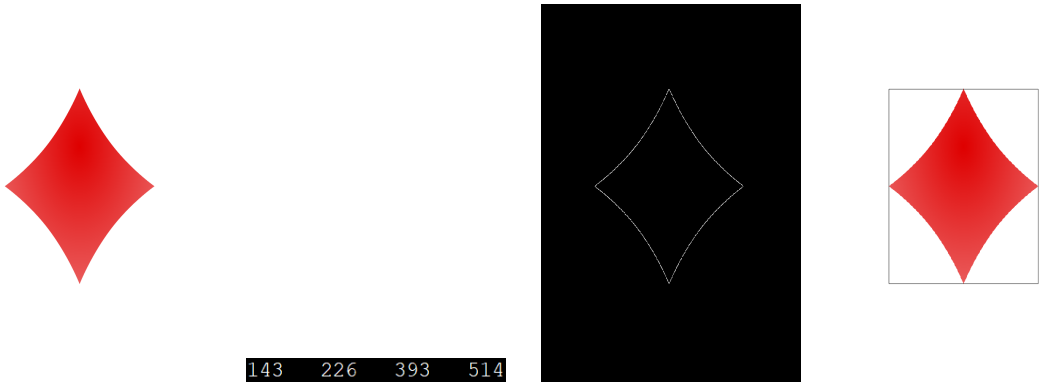
The Canny edge detector is the most commonly used edge detector.

1. Write a short summery about the Canny edge detector algorithm.
2. Use the MATLAB functions *edge(img,'Canny')* and *edge(img,'Prewitt')* on the image and show the results. Note: if a specific threshold is not mentioned, the MATLAB function will choose the threshold by itself.
3. Explain the differences between the two edge detectors as it can be seen from the images. Which edge detector will you use?

1.6 Diamonds are forever

In this subsection you will write a function named *dip_edge_detect(img)* that detects the outer bounds of the diamond in the attached diamond image, according to the following criteria. You may use built-in MATLAB functions. Special attention will be given to the criteria in your code, be very strict.

1. The function receives an image of an unknown size, meaning you cannot assume a predefined size. The code will be checked with a different sized image.
2. The function detects the diamond's edges.
3. The function has two outputs: The first is a row vector with a length of 4, the second output is a one channeled logical edge image of the same size as the input image.
4. The row vector represents the diamond's bounding box. A bounding box is the box with the smallest volume (surface in this case) that bounds the desired object. The first and second coordinates of the vector are the bounding box's upper left corner's column and row number respectively. The third and fourth coordinates are the bounding box's width and height respectively. The vector has to be of type uint16. Make sure the bounding box fits in the image. It is recommended to use a built-in MATLAB function to find the bounding box.
5. The edge image has to be of type logical, of the same size as the input image but must have one channel (i.e. even if the input has 3 channels because it is colored, the edge image still has to have one channel).
6. Use the logical edge image to create the vector representing the bounding box.
7. In case no edges are detected, the output vector is the zero vector of type uint16 and the output image is the zero matrix of type logical.
8. It is okay if the bounding box misses a few pixels of the diamond.
9. Display the output of the function. **Do not use figures, imshow, etc. within the function**. No explanation is needed.
10. An example output is demonstrated in Figure 1 . You do not have to display the result image composed of the diamond together with the bounding box.



(a) example diamond (b) example vector (c) example edge (d) example result

Figure 1: *The output of dip_edge_detect(img). (a) The input image (b) The output vector (c) The output edge image (d) The result*

2 Mona Surfaliza

In this exercise we will use SURF features in order to fix the orientation of the image and detect objects in an image. You may use all available MATLAB functions but a special emphasis will be made to your written explanations.

2.1 Make Mona Straight Again

1. Explain the SIFT algorithm.
2. Shortly explain how the SURF algorithm improves the SIFT algorithm and what these improvements are.
3. Read the images '*mona_org.jpg*' and *'crooked_mona.jpg*'.
4. Extract the SURF feature points of each of the images and display 10 of them for each of the images correspondingly.
5. Use the extracted feature points to straighten the *'crooked_mona.jpg*' image so that Mona's face will be straight. Display the images. Explain your algorithm using a block diagram and elaborate on each of the steps.

2.2 Fake or Real

1. Read the image 'straight_mona.jpg' and use the SURF features to automatically detect images in which the real Mona Liza face exists.
   (a) The 10 test images are in the zip file named 'Monas'.
   (b) The last letter in the image name 'Y' or 'N' suggests if we consider that Mona's face is in the image or not.

(c) The code should print out the names of the images you detected Mona's face inside.

(d) The algorithm should be totally automatic and run on all the images in the same manner.

(e) Try taking under consideration the trade-off between finding a lot of correct images and falsely detecting wrong images.

(f) Try to do your (and your algorithms) best, if you can't detect an image, just try your best and explain why it was impossible.

2. Display the images you found.

3. Explain your algorithm using a block diagram and elaborate on each of the steps.

4. For each image show the matching features. See example in Figure 2 .

5. Analyze the results. Why did you succeed in the image you did? Why did you get it wrong in the images you did? Where did the SURF features have good performance and where did they not?

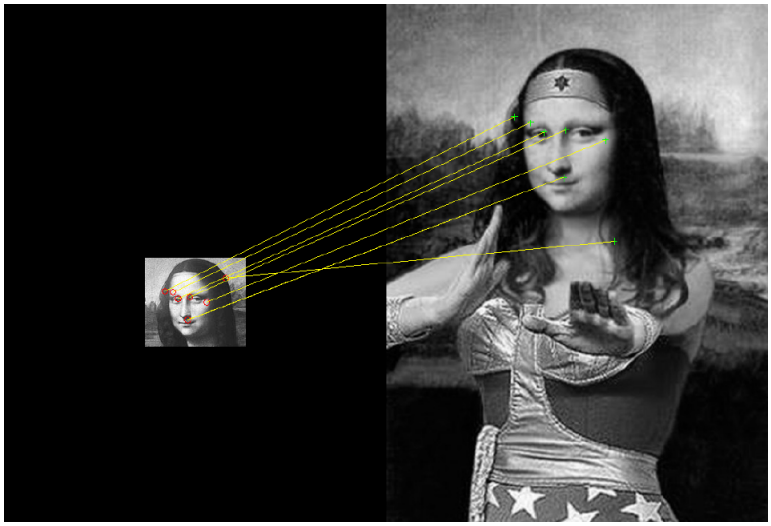


Figure 2: Example for matching features image

3 Bonus

Take two images and use the extracted features methods that you learned in class and in this exercise to create a new trashed image. **Be creative! One of the images will be chosen by the course staff and it's authors will receive one bonus point to the final grade**. The staff will judge by the visual result, originality and the code.
In order to implement your algorithm, you can use any code you find online.

Submission Instructions

The assignment is to be done in MATLAB and submitted to the course moodle page in the form of a *.zip (**not RAR**) containing ***Ex4.m***, *.m files and images along with a report in the form of a PDF file (**not .doc**). The file name should be the initials and ID of both of the team members ex. `'HG-1234567_ AA-7654321.pdf'`. The use of built-in MATLAB image processing functions is strictly forbidden unless the instructions say differently.

Document Instructions

The following instructions are mandatory and will be checked and graded by the course staff. Failing to follow these instructions **will** reduce points from you grade.

- Each section should have the relevant title as is in this document.
- Every explanation should be accompanied with the relevant image and every image should be explained in the text.
- The displayed images should be large enough for us to see them.
- The document should be organized and readable.

Code Instructions

The following instructions are mandatory and will be checked and graded by the course staff. Failing to follow these instructions **will** reduce points from you grade.

- Use MATLAB version 2014b or later. If you don't have one on your computer, you can work from the computer laboratories in building 33.
- A **main** function should call all the section functions in the correct order and should be named *Ex4.m*.
- The first line of *Ex4.m* should print the full names and IDs of all team members. Use MATLAB's *disp()* function.
- Write modular functions for the subsections and reuse those functions throughout your code whenever possible.
- Every **.m* file should start with a comment containing the full names and IDs of all team members.
- The code should be clearly written with correct indentations (you could use automatic indentation Ctrl+A followed by Ctrl+I).
- Use meaningful names for all functions and variables.
- Try to avoid overriding variables.
- Write comments for every line of code that is not completely self explanatory.

- For every image displayed give a meaningful title using MATLAB's title() function.
- Use subplots whenever possible.
- All paths to files should be relative paths. If you are using subfolders use MATLAB's *fullfile()* function to construct the path to the file. Do not hard code '/' or '\' in the paths.
- The code should run completely without errors. A project with errors **will not be checked!**