INTRODUCTION TO DIGITAL IMAGE PROCESSING

361.1.4751

EXERCISE 3 - Transformations

Submission Date: 11.12.18

Introduction

In this assignment we will practice several transformations on the image domain: Fourier, Wavelets and Hough. 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 results to the built-in functions for sanity check. Before you start the exercise, please read through the submission instructions at the end of the exercise and follow them. For any question regarding this assignment please post a question in the course forum on Moodle. for private questions only please email Itamar David at davidit@post.bgu.ac.il or hgazit@post.bgu.ac.il, royshau@post.bgu.ac.il, arbellea@post.bgu.ac.il.

1 2D-Fourier Transform

In this section we will exercise the 2D Fourier Transform on images and learn about some of its properties.

1.1 Writing your own functions

In this section you **may NOT** use the following MATLAB functions: *fft(), ifft(), fft2(), ifft2(), fftn(), ifftn(), ifftw() fftshift(), ifftshift().*

1. Write your own 2D-FFT function named *dip_fft2(I)* **and** an inverse-FFT function called *dip_ifft2(FFT).*
2. Write your own shift zero-frequency component to center of spectrum function named *dip_fftshift(FFT).*
3. Read the *beatles.png* image accompanied to this exercise, and convert it to grayscale. Make sure you also convert it into type *double* (if needed) and normalize it to $[0, 1]$ values before proceeding. Note that the image is not a built-in MATLAB file, so please download it from the course website.
4. Compute the 2D-FFT of the image using your function, and dispaly the resulting image. Make sure you shift the output image before displaying it using *dip_fftshift().* Use *imshow()* and **colorbar** functions to display the results.

5. Reconstruct the original image by using your inverse-FFT function. Is it identical to the original image? Note that the output of the iFFT are complex numbers - you should display only the **real** part of the image using *real()*.

The equations for the FFT and iFFT for an image I of size $M \times N$:

$$F(u+1, v+1) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(m+1, n+1) \cdot e^{-2\pi i\left(\frac{pm}{M} + \frac{pn}{N}\right)}$$

$$I(m+1, n+1) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u+1, v+1) \cdot e^{2\pi i\left(\frac{pm}{M} + \frac{pn}{N}\right)}$$

1.2 Transformation properties

In this section **you MAY** use the built-in Matlab *fft2(), ifft2() and fftshift()* functions.

1. **Linearity (Free Willy)**

   (a) Load the *freewilly.mat* file accompanied to the exercise. Display it as an image using *imshow()*. **don't** normalize this file.

   (b) As you can see, Willy the whale is imprisoned. Your job is to free Willy! To do that, you are told that the prison bars are made of a sinudoidal signal in the X axis that was **added** to the original image: $0.5 \cdot sin\left(\frac{2\pi f_x}{N}x\right)$, where N is the number columns in the image. Given this image, find the spatial frequency of the prison bars f_x. Explain your answer! (*Hint:* how many bars are there in the image? You may also plot the first row of the image if you find it helpful).

   (c) Based on your answer in section (b), create the image of the prison bars and display it. MAKE SURE it has the same dimensions as *frewilly.mat.*

   (d) Compute the 2D-FFT of the prison bars image, and display its **amplitude** (use the function *abs()*). Explain this result - give a mathematical proof that this is the Fourier transform that is expected here.

   (e) Explain how can you free Willy (i.e. filter out the prison bars) through the Fourier domain. Based on your answer, write a function *Free_ Willy(Willy)* that returns **and displays** Willy without the prison bars.

2. **Scaling, translation and seperability**

   (a) Initialize a 128×128 all-zeros matrix. At the center of it, place a 40×40 all-ones square (in pixels 44:84 in each dimension). Display the image and its 2D-FFT. Explain why it looks the way it does.

   (b) Initialize a 128×128 all-zeros matrix. At rows 64:104 and columns 64:104, place a 40×40 all-ones square. Display the new image and its 2D-FFT. Does the FFT looks the same as in section (a)? Now Display only the FFT **amplitude** of the previuos and the new image. Are they they identical? Explain why. Base your answer on the **mathematical relationship** between the two images and their 2D-Fourier transforms.

(c) Initialize a 128×128 all-zeros matrix. At the center of it, place a 80×20 all-ones rectangle. Display the new image and its 2D-FFT. Does the FFT looks the same as in section (a)? Explain why. Base your answer on the **mathematical relationship** between the two images and their 2D-Fourier transforms.

(d) Can you represent the image from section (c) using two 1D vectors? Explain how.

(e) Explain how can you compute the 2D-FFT of an image using 1D-FFTs if the image is separable into two 1D vectors. Write a function *sep_fft2(v1,v2)* that receives a pair of 1D vectors (of lengths N1 and N2 respectively), and returns the 2D-FFT (a N1*N2 matrix) based on the 1D-FFTs of the vectors (DO NOT use *fft2()* here). Apply this function on the two vectors you described in section (d), and display the resulting 2D-FFT. Is it identical to the 2D-FFT of the image from section (c)?

2 Wavelet Transform

In this exercise we will learn the tools for 2D Wavelet Decompostion its applications to images.

1. **2D wavelet tools**

   (a) Read the *beetle.jpg* image (accompanied to this exercise), convert it to grayscale and normalize to $[0, 1]$.

   (b) Extract the wavelet decomposition using MATLAB's *wavedec2()* function: use the *'haar'* wavelet, you may choose any level of decomposition between 3-5.

   (c) Use Matlab's *detcoef2* and *appcoef2* functions to find the detail and approximation coefficients for each level.

   (d) Display the results for each level.

   (e) Explain the results. What are the differences between each level? What can you learn from each level?

2. **Denoising**

   (a) Read the *beetle.jpg* image, convert it to grayscale and normalize to $[0, 1]$.

   (b) Create noisy image by adding **gaussian** noise to the original image using *imnoise()* function.

   (c) Remove the noise- use the functions *wthrmngr()* and *wdencmp()* with the wavelet decomposition as input, you may choose the other parameters.

   (d) Display and explain the result, use the function *psnr()* before and after the denoising to measure your result.

3 Hough Transform

In this exercise we will learn the Hough transform both for line and circles detection. You **may NOT** use the following MATLAB functions: *hough(), houghlines(), imfindcircles().*

1. Read the *circuit.tif* image and mormalize it to $[0, 1]$. This is a Matlab built-in image that you don't need to download (read it directly using *imread('circuit.tif ')*).

2. Extract the edges using MATLAB's *BW=edge(I)* and rotate it by 15 degrees using *BW=imrotate(BW,15)*.
3. Write your own *dip_hough_lines(BW)* function that calculates the Hough Matrix for finding **lines**.
4. Display the Hough Matrix (using *imshow(M,[])*) and explain the result.
5. Find the 5 most significant lines in the rotated BW image, and plot them on the grayscale circuit image (you will need to rotate it by 15 degrees as well). Use MATLAB?s *houghpeaks* function to find the peaks of your Hough matrix.
6. Read Matlab's built-in *coins.png* image and normalize it to [0; 1]. Write your own *dip_hough_circles(BW)* function that calculates the Hough Matrix for finding **circles**. Repeat steps 1-5, but this time find and plot the 5 most significant circles in the image. **Note:** This time your Hough matrix is 3D, so you can display one slice (2D image).
   Hint: you should bound your search for radius values 15<R<35.

In order to construct the Hough matrix follow these steps:

```
1. Create an empty matrix of size $\left[2\sqrt{M^2+N^2}+1\right]\times 181$ corresponding to values for $r \in \left[-\sqrt{M^2+N^2}, \sqrt{M^2+N^2}\right]$ and $\theta \in [-90, 90]$
    (a) For every white pixel in the image, $x, y \in BW$:
        i. For every value of $\theta$:
            A. $r = x\cos(\theta) + y\sin(\theta)$
            B. Add 1 to the Hough matrix at the location corresponding to $r, \theta$
```

In order to construct the Hough matrix **for circles** follow these steps:

```
1. Create an empty matrix of size $M \times N \times 21$ corresponding to the size of $BW$
    (a) For every pixel in the image, $x, y \in BW$:
        i. For every value of $R \in [15, 35]$:
            A. For every value of $\theta \in [-90, 90]$:
               $a = x - R\cos(\theta)$
               $b = y - R\sin(\theta)$
               Add 1 to the Hough matrix at the location corresponding to $(a, b, R)$. (Don't forget to round the values)
```

Note: This time your Hough matrix is 3D, so *houghpeaks* may not work. To find the 5 peaks you may use this piece of code instead:

```
peaks = zeros(5,3);
for i = 1:5
        [val,idx] = max(H(:));
        [idx1,idx2,idx3] = ind2sub(size(H),idx);
        peaks(i,:) = [idx1,idx2,idx3];
        H(idx1,idx2,idx3) = 0;
end
```

4 Bonus

Use the Generalized Hough Transform (GHT) is a creative way. Take two images, one is a binary image of an interesting shape and the other is an image of your liking. Use the GHT to find the shape of the first image in the second image and show the results. **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 the GHT, you can use the added *Generalized_hough_transform_standalone.m* function or any other implementation 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 *.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 says 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 *Ex3.m.*
- All the other function should named *dip_ *.m*
- The first line of *Ex3.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!**