Image Processing - 67829

Exercise 2: Fourier Transform & Convolution

Due date: 22/11/2015

The purpose of this exercise is to help you understand the concept of the frequency domain by performing simple manipulations on images. This exercise covers:

- Implementing Discrete Fourier Transform (DFT) on 1D and 2D signals
- Performing image derivative
- Convolution theory

1 Discrete Fourier Transform - DFT

1.1 1D DFT

Write two functions that converts a 1D discrete signal to its fourier representation and vice versa. The two functions should have the following interfaces:

```
fourierSignal = DFT(signal)
signal = IDFT(fourierSignal)
```

where: `signal` is a row vector of type double, and `fourierSignal` is a row complex vector of type double. **note: "IDFT" stands for inverse DFT**. You should use the following formulas:

For DFT transform:

$$F(u) = \sum_{x=0}^{N-1} f(x) e^{-\frac{2\pi i u x}{N}}$$

And for IDFT transform:

$$f(x) = \frac{1}{N} \sum_{u=0}^{N-1} F(u) e^{\frac{2\pi i u x}{N}}$$

Both functions should be implemented without loops.

1.2 2D DFT

Write two functions that converts a 2D discrete signal to its fourier representation and vice versa. The two functions should have the following interfaces:

```
fourierImage = DFT2(image)
image = IDFT2(fourierImage)
```

where: `image` is a grayscale image of type double, and `fourierImage` is 2D array with complex numbers of type double.

You should use the following formulas:

For DFT2:

$$F(u,v) = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} f(x,y) e^{-2\pi i(\frac{ux}{N} + \frac{vy}{M})}$$

And for IDFT2:

$$f(x,y) = \frac{1}{NM} \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} F(u,v) e^{2\pi i(\frac{ux}{N} + \frac{vy}{M})}$$

Where M is the height of the 2D array, and N is the width of the 2D array.

<u>Note</u>: You should implement these 2D transformation using subsequent 1D transformations (section 1.1). You can loop over one dimension and call 1D transformation each time on one vector, or you can make your 1D transformations to be compatible with an input of a matrix so you will be able to do make it without loops.

Pay attention that you should not implement the Fast Fourier Transform.

2 Image derivatives

2.1 Image derivatives in image space

Write a function that computes the magnitude of image derivatives. You should derive the image in each direction separately (rows and column) using simple convolution with $[1; 0; -1]$ and $[1, 0, -1]$ to get the two image derivatives. Next, use these derivative images to compute the magnitude image. The function should display the result. The function should have the following interface:

```
magnitude = convDerivative(inImage)
```

Where the input and the output are grayscale images of type double.

2.2 Image derivatives in fourier space

Write a function that computes the magnitude of image derivatives using fourier transform. Slides 27-28 from Tirgul 3 will guide you how to derive in the x and y directions. You are recommended to use fftshift in the frequency domain so the (0,0)frequency will be at the center of the image. In this function you should also return the magnitude and display the result. The function should have the following interface:

```
magnitude = fourierDerivative(inImage)
```

Where the input and the output are double grayscale images.
The magnitude should be calculated in the following way:

```
magnitude = sqrt(abs(dx).^2+abs(dy).^2)
```

In both sections (2.1 and 2.2) you should <u>not</u> normalize the magnitude values to be in the range of [0,1], just return the values you get.
Q: Why did you get two different magnitude images?

3 Convolution theory

In this section we will get familiar with the convolution theory. You should blur an image f in two ways:

- using a convolution with gaussian kernel g in image space
- using a point-wise operation between the fourier image F with the same gaussian kernel, but in fourier space G.

3.1 Blurring in image space

You should write a function that performs image blurring using 2D convolution between the image f and a gaussian kernel g. The function should also display the result. The function should have the following interface:

```
blurImage = blurInImageSpace(inImage,kernelSize)
```

where:
`inImage` - is the input image to be blurred (grayscale double image).
`kernelSize` - is the size of the gaussian in each dimension (one odd integer).
`blurImage` - is the output blurry image (grayscale double image).

Comments:

- The gaussian kernel g should contain approximation of the gaussian distribution using the binomial coefficients. A consequent 1D convolutions of [1 1] with itself is an elegant way for deriving a row of the binomial coefficients. Explore how you can get a 2D gaussian approximation using the 1D binomial coefficients. Pay attention that the kernel elements should be summed to 1.
- Once you have a 2D gaussian kernel, you can convolve the image with the kernel using the function `conv2`.
- You can handle the border issue as you like. (zero padding, cyclic image...).

3.2 Blurring in fourier space

You should write a function that performs image blurring with gaussian kernel in fourier space. This function should also display the result image. The function's interface:

```
blurImage = blurInFourierSpace(inImage,kernelSize)
```

where:
`inImage` - is the input image to be blurred (grayscale double image).
`kernelSize` - is the size of the gaussian in each dimension (one odd integer).
`blurImage` - is the output blurry image (grayscale double image).

In this function you should create the 2D gaussian kernel g in the same way you created in section 3.1 and transform it to its fourier representation G. You should also transform the image f to it fourier representation F and multiply point-wise $F \cdot G$. Then, you should perform the inverse transform on the result in order to get back to image space.
Comments:

- The kernel g has $kernelSize$ pixels in each direction, hence, its fourier representation will be also with $kernelSize$ pixels in each direction and can not be point-wise multiply with F (which have the same size of the image). Therefore, you should pad the gaussian kernel, g, with zeros to bring it to the same size as the image while preserving the center of the gaussian at (0,0).
- To do so, you are recommended to build a 2D-gaussian kernel in the center of an image. in case that the resolution of the image is $[m \;\; n]$, you should locate the center of the gaussian kernel at $[floor(m/2)+1 \quad floor(n/2)+1]$. The convention for even number of pixels is the same. Then you can use `ifftshift` so the center of the gaussian will be at (0,0) of the image.

Q: What happens if the center of the gaussian (in the space domain) will not be at the (0,0) of the image? Why does it happen?
Q: What is the difference between the two results (Blurring in image space and blurring in fourier space)?

4 Important Comments

- Use `conv2` with the `'same'` option when you perform the convolution operation (in section 2.1 and 3.1).
- In order to compute the fourier transform in sections 2.2 and 3.2 you should use **your** 2D implementation from section 1.2.
- When you want to transpose a matrix with complex numbers use the operator `.'` instead of just `'` . The reason is that the operator `'` computes the complex conjugate transpose of a matrix so it changes the sign of the imaginary part of each element.

5 Some tips

- **Fourier centering:** The output of your `DFT2` implementation is a matrix which contains the Fourier coefficients. This matrix is organized s.t. $F(0,0)$ is located in the (1,1) entry (top left corner). However, visualizing the Fourier coefficients may be easier to do with $F(0,0)$ shifted to the center of the matrix. For this purpose use `fftshift` which performs a *cyclic translation* of a matrix in both axes (try to shift a grayscale image in order to understand the behavior of this function). In case that you have shifted the frequency domain, and now you want to perform the inverse transform, you should shift the coefficient image back beforehand using `ifftshift`.
- **Fourier display:** To best visualize a Fourier coefficient image `im` you should first apply the intensity transformation `log(1+abs(im))` discussed in class. This operation reduces the dynamic range of the coefficient magnitude image and will therefore cause more values to become visible (try to display `im` with and without this transformation and compare what you see).
- **Useful functions:** `conv2` (2D convolution – use the `'same'` option); `meshgrid` (used to create index maps, you can also use 'repmat' which replicates a matrix); `complex` (used to create complex numbers). `fft2` (2D discrete Fourier transform); `ifft2` (inverse 2D DFT) - Matlab implementation

for FFT (the fast version of DFT) so you can use these functions to check your results from section 1.1 and 1.2

- When you return from the frequency domain to the image domain, using your implementation (IDFT2) or matlab implementation (ifft2), there might be some very small imaginary part in the matrix elements because of numerical errors. You can ignore them and take only the real part of the matrix.

6 Submission

Submission instructions may be found in the "Exercise Guideline's" document published on the course web page. Please read and follow them carefully. Your README should also include the answers for the questions in section 2 and 3.

Good luck and enjoy!