Image Processing - 67829

Exercise 2: Fourier Transform & Convolution

Due date: 1/12/2014

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` and `fourierSignal` are row vectors.
note: "IDFT" stands for inverse DFT. You should use the transformation formulas from Lecture 3 slide no. 13.

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: `signal` and `fourierSignal` are 2D arrays
You should implement this 2D transformation using subsequent 1D transformations (section 1.1). 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 an image derivative. 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 also display the result. The function should have the following interface:

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

2.2 Image derivatives in fourier space

Write a function that computes the magnitude of image derivative using fourier transform. Slides 27-28 from Tirgul 3 will guide you how to derive in the x and y directions. In this function you should also return the magnitude and display the result. The function should have the following interface:

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

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 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.

`kernelSize` - is the size of the gaussian in each dimension (squared kernel).

`blurImage` - is the output blurry 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.
- You can handle the border issue as you like. (zero padding, cyclic image...)
- The size of the gaussian, $kernelSize$, will always be an odd number.

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)
```

In this function you should create the 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 you preserve the gaussian in the middle of the image.

- To make it clear, in case that the resolution of the image is $[m \ \ n]$, you should locate the gaussian kernel at $[floor(m/2)+1 \ \ \ floor(n/2)+1]$. The convention for even number of pixels is the same.

Q: What is the difference between the two results?

4 Important Comments

- Use `conv2` with the `'same'` option when you perform the convolution operation (in section 2.1 and 3.1).
- The input of all the above functions will be grayscale images.
- In order to compute the fourier transform in sections 2.2 and 3.2 you should use **your** 2D implementation from section 1.2.

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). In addition, shift the frequency coefficient image using `fftshift` to center $F(0,0)$.
- **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

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!