ENGG5104 Image Processing and Computer Vision

Assignment 1 – Image Processing Basic

Due Date: 11:59pm on Monday, February 1st, 2016

Part 1: Image Compression

Background:

In this section we focus on compression of digital images. Image compression schemes aim to decrease the number of bits used to represent the image. Using fewer bits allows the image to take up less storage space on a computer, and it can also be transmitted faster (over a network, via satellite, etc.).

The goal of this part is to look at the effect of compression that we do ourselves by implementing a gray-scale JPEG compression.

The gray scale JPEG compression consists of the following steps:

1. *Block splitting*:Splitting the image into 8*8 blocks.
2. *Discrete cosine transform*: Each 8×8 block F is converted to a frequency-domain representation$G$, using a normalized, two-dimensional type-II discrete cosine transform (DCT).
3. *Quantization*: Using a predefined quantization matrix to control compression ratio. Define Q as the quantization matrix. The quantized DCT coefficient are computed as
$$B_{ij} = round(G_{ij}/Q_{ij})$$
4. *Decoding*: decode using inverse discrete cosine transform (inverse DCT) from the quantized DCT coefficients.

Note that we only aim to look at the effect of compression, so we omit the entropy coding step, which is a special form of lossless data compression to further reduce storage. It involves arranging image components in a "zigzag" order employing run-length encoding algorithm, and using Huffman coding on what is left.

Todo List:

Implement the JPEG compression function

```
Def jpegcompress(image, quantmatrix)
```

- TODO #1: Pad the image if the number of blocks is not integer
- TODO #2: Separate the image into blocks, and compress the blocks via quantization DCT coefficients.

Hints:

1. The DCT and inverse DCT can be implemented using python function `cv2.dct2 and cv2.idct2.`
2. The quantization matrix is given.

Reference

1. http://en.wikipedia.org/wiki/JPEG
2. Digital Image Processing, *Rafael C. Gonzalez and Richard E. Woods*, 2nd Edition, Chapter 8.6

Part 2: Richardson-Lucy Deconvolution

Background

The **Richardson–Lucy** algorithm, also known as **Lucy–Richardson** deconvolution, is an iterative procedure for recovering a latent image that has been blurred by a known **point spread function (psf)**. It was named after William Richardson and Leon Lucy, who described it independently. [1][2]

When an image is recorded by a camera, it is generally slightly blurred, due to out of focus effect or camera shake. This process is usually approximated by a simple discrete convolution:

$$B = I \otimes K \Leftrightarrow B_p = \sum_{q \in N(p)} K_{p-q} \cdot I_q$$

Where I is the clear image, B is the observed blurred image and K is the point spread function recording how each pixels in an image relate to neighboring pixels. $(\otimes)$ is discrete variable convolution operator. p, q are 2D pixel locations.$N(p)$ is the neighboring region of pixel p, according to the size of K.

The inverse problem to estimate clear image I_p, given blurred image B and psf K is called deconvolution.

The basic idea for RL deconvolution method is to calculate the most likely I_p, this leads to an equation that can be iteratively solved:

$$I_p^{t+1} = I_p^t \sum_{q \in N(p)} \frac{B_q}{\sum_p K_{q-p} \cdot I_p^t} K_{p-q}$$

It has been shown empirically that if the iteration converges, it will converge to the maximum likelihood solution of I [1]. Equivalently, this can be written in convolution forms:

$$I^{t+1} = I^{t} \cdot \left(\frac{B}{I^{t} \otimes K} \otimes \hat{K}\right)$$

Where $\hat{K}$ is the flipped version of K (e.g. flip K vertically and then horizontally, or rotate K by 180 degree). Other operations are pixel-wise multiplication and division operation.

Sample results: Iteration 1st, Iteration 10th, Iteration 25th

Todo List:

Given the blur kernel PSF, the blurred image B, using RL deconvolution to compute the clear image.

```
def rl_deconv(B, PSF)
```

- TODO: Implement `rl_deconv()` function based on the above equation.

Hints:

1. You can use Python + OpenCV function `cv2.filter2D()` to do correlation between image and PSF. Be careful about the parameters for these functions.
2. You can use `cv2.flip()` to flip a 2D PSF vertically or horizontally. Thus you can then calculate real convolution using `cv2.filter2D()`
3. You may need to process the image's RGB channels separately. Color images are still 2D matrix in Python OpenCV, with each element a tri-value data type indicating 3 channels.
4. Use given PSF images to generate blur and then deblur.
5. Iteration number is important for visual effect. Larger iterations may not lead to better effect. Choose a proper iteration number through experiments.

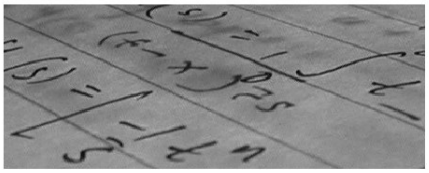
Reference

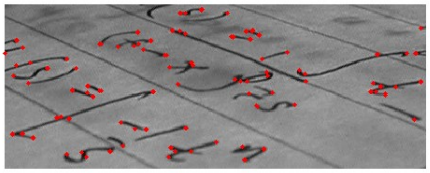
1. http://en.wikipedia.org/wiki/Richardson–Lucy_deconvolution
2. Digital Image Processing, *Rafael C. Gonzalez and Richard E. Woods*, 2nd Edition, Chapter 5.7

Part 3: Harris Corner Detection

Background

Harris corner detection algorithm is used for detecting corner in the image. It considers the corner point with respect to both vertical and horizontal direction. Assume a simple 2D grayscale image is given. By applying Harris corner detection algorithm, we get the detected corners in the image as follows:





Denote a grayscale image as I. Consider a patch centers on coordinate (u, v)in I. It is shifted by (x, y). The weighted sum of squared difference between these two patches, denote as S, is given by:

$$S(x,y) = \sum_u \sum_v w(u,v)\,(I(u+x, v+y) - I(u,v))^2$$

Note that $I(u+x, v+y)$ is approximated by Taylor expansion. Let I_x and I_y be the partial derivatives of I, such that:

$$I(u+x, v+y) = I(u,v) + I_x(u,v)x + I_y(u,v)y$$

Then the approximation $S(x, y)$ of is formulated as:

$$S(x,y) \approx \sum_u \sum_v w(u,v)\,(I_x(u,v)x + I_y(u,v)y)^2$$

which can be further written as matrix form:

$$S(x,y) \approx (x\ y) A \begin{pmatrix} x \\ y \end{pmatrix}$$

where

$$A = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

By analyzing the eigenvalues of A, the corner located in (x, y) is characterized by two "large" eigenvalues. Denote the eigenvalues of A as $\lambda_1$and λ_2. The cases of λ_1 and λ_2 are:

1. $\lambda_1 \approx 0$and $\lambda_2 \approx 0$: The pixel $(x, y)$is not a corner.
2. $\lambda_1 \approx 0$andλ_2 has some large positive value: The pixel $(x, y)$is located in an edge.
3. $\lambda_1$andλ_2 have large positive values: The pixel $(x, y)$is a corner.

Todo List:

Given a grayscale image I, the size k, and the threshold t. The function $w(u, v)$ is simply modeled as a square matrix filled with 1, and its width and height are $2k + 1$. When the eigenvalues of Harris matrix exceed the threshold t, the candidate will be regarded as corner. You are required to figure out the coordinate of the corner in the image I.

```
def harrisdetector(I, k, t)
```

- TODO: Write `harrisdetector()` function based on the above illustration.

Hints:

1. You can use Python function `cv2.filter2D()` to compute the I_x and I_y .
2. The Python function `cv2.eigen` is for computing the eigenvalues of a square matrix.

Reference

1. http://en.wikipedia.org/wiki/Corner_detection#The_Harris_.26_Stephens_.2F_Plessey_.2F_Shi.E2.80.93Tomasi_corner_detection_algorithm
2. Harris, Chris, and Mike Stephens. "A combined corner and edge detector."*Alvey vision conference*. Vol. 15. 1988

Part 4: Unsharp Masking

Background

Unsharp masking is used to sharpen image consists of subtracting a blurred version of an image from the image itself. This process can be expressed as

$$I_s(x, y) = I(x, y) - \hat{I}(x, y),$$

where $I_s(x, y)$is the sharpened image using unsharp masking, and $\hat{I}(x, y)$is a blurred version of the input image $I(x, y)$. $I(x, y)$,$\hat{I}(x, y)$ and $I_s(x, y)$ are shown in the following figure:

As we can see, $I_s(x, y)$maintains the edges which are the details of the image. We can further generalize the unsharp masking as

$$I_{hb}(x, y) = AI(x, y) - \hat{I}(x, y),$$

where A is a constant number and $I_{hb}(x, y)$ is the final result of unsharp masking. We usually set $A \geq 1$, and we can rewrite the generalized unsharp masking as

$$I_{hb}(x, y) = (A - 1)I(x, y) + I(x, y) - \hat{I}(x, y) = (A - 1)I(x, y) + I_s(x, y).$$

We often obtain $I_s(x, y)$ by convoluting a Laplacian operator with the input image $I(x, y)$. The Laplacian operator we use here can be visualized as the following matrix:

−1	−1	−1
−1	8	−1
−1	−1	−1

If $A = 1$, then the result$I_{hb}(x, y)$ is exactly $I_s(x, y)$. We also provide the result $I_{hb}(x, y)$in the case where $A = 2$. We compare the input image $I(x, y)$ and $I_{hb}(x, y)$ as shown in the following figure:

From the above figure, we can see that all the details (edges) of the input image $I(x, y)$ are enhanced as shown in image $I_{hb}(x, y)$.

The unsharp masking algorithm can be summarized to three steps:

1. Compute the image$(A - 1)I(x, y)$. We assume $A \geq 1$ here.
2. Compute the image $I_s(x, y)$ using the Laplacian operator we given.
3. Add the images $(A - 1)I(x, y)$ and $I_s(x, y)$.

Todo List:

Given a grayscale image I, the Laplacian operator k and the constant A, you are required to implement the unsharp masking function and return the new image.

```
def unsharpmask(I, k, A)
```

- TODO: Write `unsharpmask` function based on the above illustration.

Hints:

1. You can use Python function `cv2.filter2D` to compute the convolution between image$I(x, y)$ and the Laplacian operator in order to obtain image $I_s(x, y)$.

Reference

1. http://en.wikipedia.org/wiki/Unsharp_masking
2. Digital Image Processing, *Rafael C. Gonzalez and Richard E. Woods*, 2nd Edition, Chapter 4.4

Handing in

The folder you hand in must contain the following:

- **README.txt** - containing anything about the project that you want to tell the TAs, including brief introduction of the usage of the codes
- **code/** - directory containing all your code for this assignment

Compressed the folder into <*your student ID*>**-Asgn1.zip** or <*your student ID*>**-Asgn1.rar**, and upload it to **e-Learning** system.

More Hints

1. Python + OpenCV + Numpy + Scipy

   You will need Python based libraries:

   - OpenCV: a Python wrapper for the powerful vision library, including almost anything you need for vision and image processing applications.

     You cannot directly use built-in functions in this assignment.

     Installation:

     http://docs.opencv.org/master/d5/de5/tutorial_py_setup_in_windows.html#gsc.tab=0
   - Numpy + Scipy: famous fundamental package for scientific computing. It will be a powerful supplementary for OpenCV. Installation:

     http://www.lfd.uci.edu/~gohlke/pythonlibs/#numpy

     http://www.lfd.uci.edu/~gohlke/pythonlibs/#scipy

     Or pip install
   - matplotlib: a powerful tool for visualization, included in scipy.

     If you still cannot handle installation, an alternative choice:

     http://docs.continuum.io/anaconda/install .

     Anaconda is a set of several science libraries.

2. Listed functions that may be useful
   - cv2.imread() – read image
   - cv2.filter2D() – do correlation filter
   - cv2.flip() – flip an image horizontally or vertically
   - cv2.imwrite() – write result to images
   - cv2.dct / cv2.idct – do DCT and inverse DCT, may require certain data type input
   - cv2.imshow() – show image in a window, should be followed by cv2.waitKey(0)
   - cv2.destroyAllWindows() – close and destroy all windows
   - <numpy array>.shape – output the size of <numpy array>
   - <numpy array>.astype() – change data type
   - <numpy array 2> = <numpy array 1> [1:100, 1:100] – to extract a region from <numpy array 1>
   - pyplot.imshow() – matplotlib function to show image, should be followed by plt.show()
   - scipy.io.loadmat() – to read MATLAB .mat file
   - Refer to documents and Internet resources to find more.