Angel U. Ortega
Computer Vision – Lab 1
Version 1.0
9/7/2016

Table of Contents

1. Introduction

1.1. Lab Overview

The lab is composed of three problems, P1, P2, and P3, respectively. P1 is made up of four exercises from chapter 1 of the book (E1, E2, E3, and E5). The exercises are as follows:

P1E1: Take an image and apply Gaussian blur like in Figure 1.9. Plot the image contours for increasing values of theta. What happens? Can you explain why?

P1E2: Implement an unsharp masking operation (http:en.wikipedia.org/wiki/Unsharp_masking)[1] by blurring an image and the subtracting the blurred version from the original. this gives a sharpening effect to the image. Try this on both color and grayscale images.

P1E3: An alternative image normalization to histogram equalization is a quotient image. A quotient image is obtained by diving the image with a blurred version I/(I*G(sub-theta)). Implement this and try it on some sample images.

P1E5: Use gradient direction and magnitude to detect lines in an image. Estimate the extent of the lines and their parameters. Plot the lines overlaid on the image.

P2: Run the histogram equalization algorithm described in the textbook. Experiment with images with various kinds of lighting and report your results.

P3: Run the Rudin-Osher-Fatemi de-noising algorithm described in the textbook. Experiment with various parameter choices and images with different levels of noise and report your results.

1.2. References

[1] http:en.wikipedia.org/wiki/Unsharp_masking
[2] https://drive.google.com/open?id=0B_kWRxLZdmeJWk5QR19PYWtXNDA

2. Proposed Solution Design and Implementation

2.1. P1E1

For this exercise, I utilized the Gaussian filter, then used the contour function to plot the image contours. I used n=5, 10, and 15 as values for theta.

2.2. P1E2

I utilized Wikipedia's algorithm for this exercise: [Sharpened = Original + (Original - Blurred) * Amount]. I first blurred the original image, then created a new image (A) with the result of subtracting the blurred image from the original image. I then followed the formula: Sharpened = Original + A * Amount. I played with different thetas for blurring and different ns for amount ranging from 0 to 1.

2.3. P1E3

For this exercise, I followed the directions in the problem statement. I began by creating a quotient image by diving the image by a blurred version of itself. Then I subtracted this from the original image to obtain an alternate image normalization.

2.4. P1E5

This was the exercise that took me the longest to complete. I began by deriving two images from the original. The first was a cropping from (n,n) to (len(im[0]), len(im)). The second was a cropping from (0,0) to (len(im[0]-n), len(im)-n). Then I subtracted the difference between this two images from 255. I saved this image as im4. Im4, at this point, was an edge detector. I then used cv2 to convert im4 to a binary image and saved as im5. Finally, I added im2 and img5 to get im6. Im6 is the original image and the lines on top. I was unable to estimate the extent of the lines and their parameters.

2.5. P2

For this exercise I utilized the histogram equalization algorithm from the book. I used different images to identify patterns and differences.

2.6. P3

For this last exercise I followed the Rudin-Osher-Fatemi de-noising algorithm from the book. I experimented with different images. The resuts can be seen in the next section.

3. Experimental Results and Conclusions

All images created for this lab can be accessed at:
https://drive.google.com/open?id=0B_kWRxLZdmeJWk5QR19PYWtXNDA [2]
Below are just a few examples from the images derived.

3.1. P1E1

Figure 1: Above images show original image, and contour images with theta values 5, 10, and 15 respectively

As it can be seen in Figure 1, the contour is more detailed, the lower the value of theta. This happens because a greater theta blurs the image too much, removing edges, and thus distorting the contour to a higher degree.

3.2. P1E2

A

B

C

D

Figure 2: From top to bottom: A – Original; B – theta = 1, n = 0.1; C – theta = 300, n = 0.1; D – theta = 300, n = .5

As it can be seen from figure 2, the larger the theta for blurring, and the smaller the amount for sharpening, the better the result.

3.3. P1E3

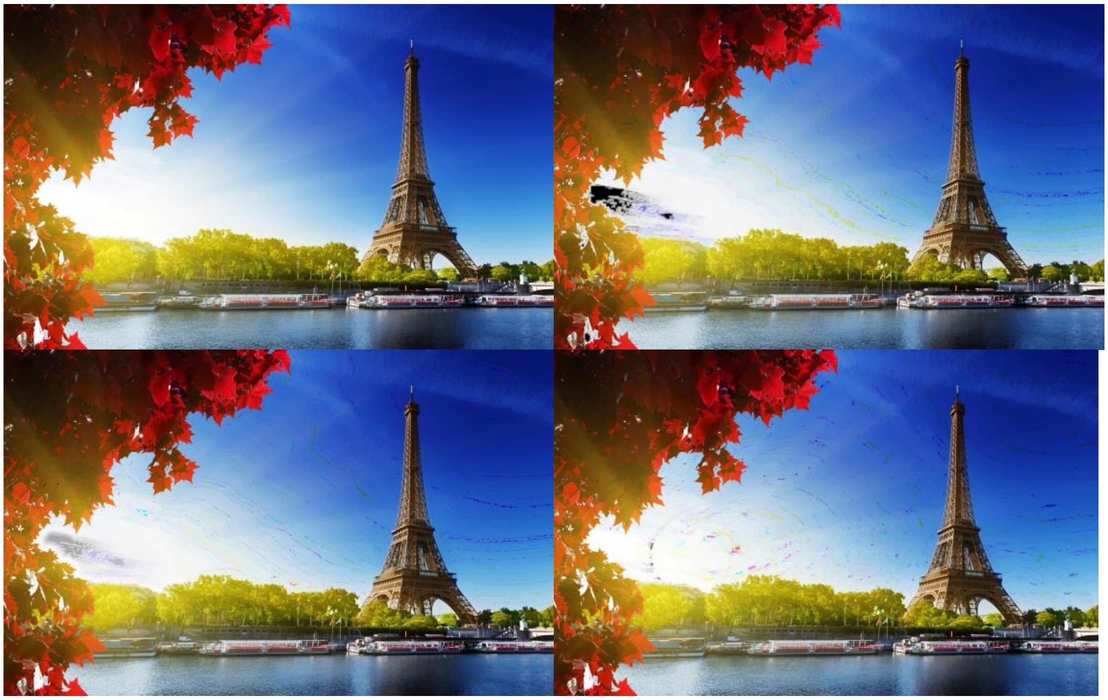


Figure 3: Original, theta = 1, theta = 100, theta = 300

As it can be seen in Figure 3, the greater the theta, the better the results. A small theta caused the image to appear distorter, while a larger theta picked up on the small details.

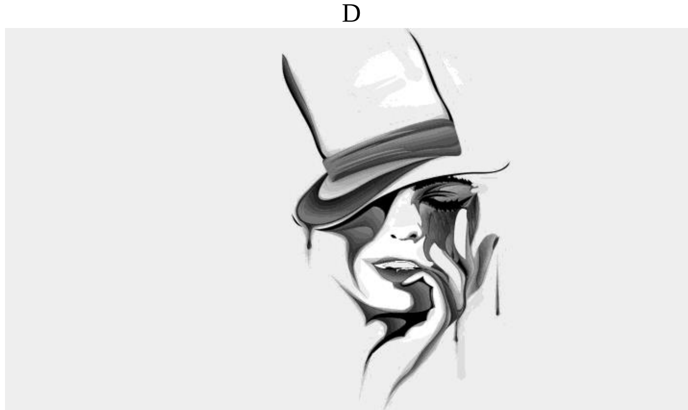
3.4. P1E5

A

B

C

D



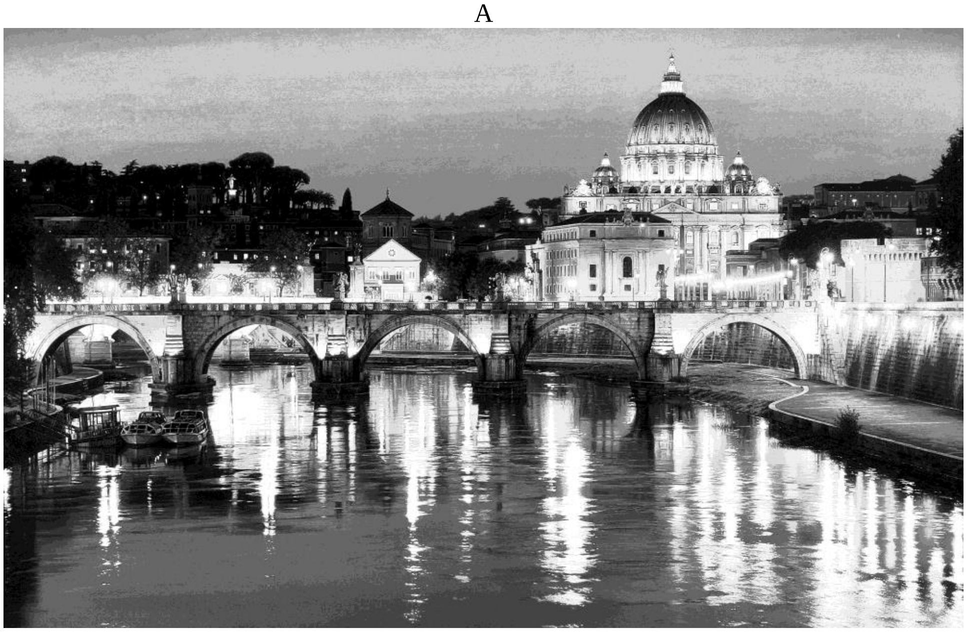
E

Figure 3: A – Original, B – Binary n = 1, C – Result n =1, D – Binary n = 20, Result n = 20

As it can be seen in Figure 3, the bigger the n, the pixel overlap, the thicker more defined lines in the binary picture. The algorithm I came up with does this line detection without the use of for loops.

3.5. P2

A



B

C

D

Figure 5: A – Original 1, B – Histogram Equalization of A, C – Original 2, D – Histogram Equalization of C

As it can be seen from Figure 5, the amount of lighting in the original images did affect the results. The higher amount of lighting of 5-C rendered its histogram equalization cleaner than that of 5-A, which was significantly darker.

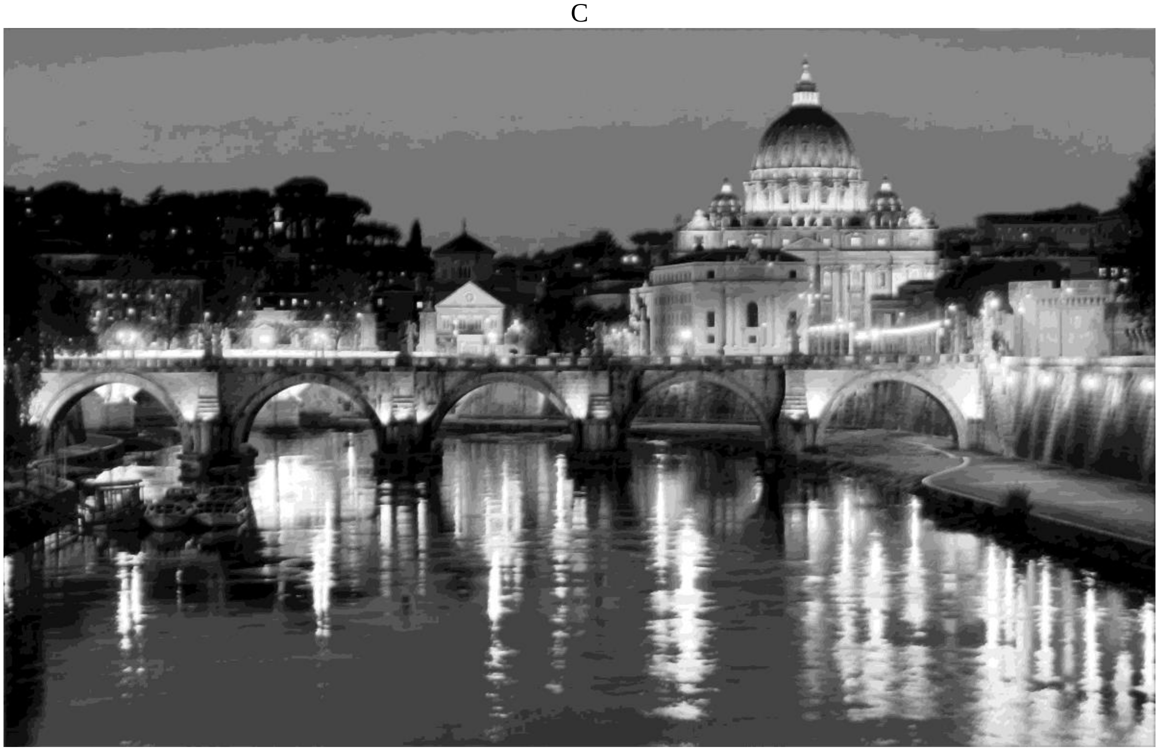
3.6. P3

A

B

As it can be seen in Figure 6, a darker image (C), with more noise has a better de noising than a brighter, less noisy one (A).

C



D

Figure 6: A – Original 1, B – De-noised A, C – Original 2, D – De-noised C

As it can be seen in Figure 6, a darker image (C), with more noise has a better de noising than a brighter, less noisy one (A).

4. Appendix

4.1. Code

4.1.1. Problem 1, Exercise 1

```
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 06 11:54:50 2016

@author: auort
"""

from PIL import Image
from pylab import *
from numpy import *
from numpy import random
from scipy.ndimage import filters
import scipy.misc

"""
Take an image and apply Gaussian blur like in Figure 1.9.
Plot the image contours for increasing values of theta. What
happens? Can you explain why?
"""

directory = 'C:\\Users\\auort\\Desktop\\CV_L1\\'

def construct (identifier, theta, degree):
    im  = array(Image.open(directory + identifier +'_Grayscale.jpg'))
    G = filters.gaussian_filter(im,theta)
    figure()
    gray()
    contour(G, origin='image')
    axis('equal')
    axis('off')
    figure()
    hist(G.flatten(),128)
    show()

construct('1',5,'5')
construct('1',10,'10')
construct('1',15,'15')
construct('2',5,'5')
construct('2',10,'10')
construct('2',15,'15')
construct('3',5,'5')
construct('3',10,'10')
construct('3',15,'15')
construct('4',5,'5')
construct('4',10,'10')
construct('4',15,'15')
construct('5',5,'5')
construct('5',10,'10')
construct('5',15,'15')
```

4.1.2. Problem 1, Exercise 2

```
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 06 11:58:14 2016

@author: auort
"""

from PIL import Image
from numpy import *
from numpy import random
from scipy.ndimage import filters
import scipy.misc

"""
Implement an unsharp masking operation (http:en.wikipedia.org/wiki/Unsharp_
masking) by blurring an image and then subtracting the blurred version from the
original. this gives a sharpening effect to the image. Try this on both color
and grayscale images.

"""

directory = 'C:\\Users\\auort\\Desktop\\CV_L1\\'

def construct (identifier, theta, degree, amount, tag):
    #Original images
    im1  = array(Image.open(directory + identifier +'.jpg'))
    im2  = array(Image.open(directory + identifier +'_Grayscale.jpg'))

    #Blurred images
    G1 = filters.gaussian_filter(im1,theta)
    G2 = filters.gaussian_filter(im2,theta)
    """
    #Blurred and inverted images
    C1 = 255 - B1
    C2 = 255 - B2
    #Blurred, inverted, and clamped images
    D1 = (1.0/255) * C1 + 100
    D2 = (1.0/255) * C1 + 100
    #Blurred, inverted, and clamped + Original
    E1 = D1 + A1
    E2 = D2 + A2
    scipy.misc.imsave(directory + 'P1\\E2\\' + identifier + '\\Sharpened_Color_' +
degree + '.jpg', E1)
    scipy.misc.imsave(directory      +      'P1\\E2\\'      +      identifier      +
'\\Sharpened_Grayscale_' + degree + '.jpg', E2)
    """

    M1 = (im1-G1)
    M2 = (im2-G2)
    S1 = im1 + M1 * amount
    S2 = im2 + M2 * amount
    I1 = 255 - M1
    I2 = 255 - M2
    scipy.misc.imsave(directory + 'P1\\E2\\' + identifier + '\\Color_' + degree +
'_' + tag +'.jpg', S1)
    scipy.misc.imsave(directory + 'P1\\E2\\' + identifier + '\\Grayscale_' + degree
+ '_' + tag + '.jpg', S2)
    """
    scipy.misc.imsave(directory + 'P1\\E2\\' + identifier + '\\Sharpening_Color_' +
degree + '.jpg', S1)
    scipy.misc.imsave(directory      +      'P1\\E2\\'      +      identifier      +
```

```
'\\Sharpening_Grayscale_' + degree + '.jpg', S2)
    scipy.misc.imsave(directory      +      'P1\\E2\\'      +      identifier      +
'\\Sharpening_Color_Invert_' + degree + '.jpg', S3)
    scipy.misc.imsave(directory      +      'P1\\E2\\'      +      identifier      +
'\\Sharpening_Grayscale_Invert_' + degree + '.jpg', S4)
    """

def run (n):
    construct(n, 1, '1', .1, '.1')
    construct(n, 10, '10', .1, '.1')
    construct(n, 20, '20', .1, '.1')
    construct(n, 50, '50', .1, '.1')
    construct(n, 100, '100', .1, '.1')
    construct(n, 200, '200', .1, '.1')
    construct(n, 300, '300', .1, '.1')

    construct(n, 1, '1', .5, '.5')
    construct(n, 10, '10', .5, '.5')
    construct(n, 20, '20', .5, '.5')
    construct(n, 50, '50', .5, '.5')
    construct(n, 100, '100', .5, '.5')
    construct(n, 200, '200', .5, '.5')
    construct(n, 300, '300', .5, '.5')

    construct(n, 1, '1', .9, '.9')
    construct(n, 10, '10', .9, '.9')
    construct(n, 20, '20', .9, '.9')
    construct(n, 50, '50', .9, '.9')
    construct(n, 100, '100', .9, '.9')
    construct(n, 200, '200', .9, '.9')
    construct(n, 300, '300', .9, '.9')


run('1')
run('2')
run('3')
run('4')
run('5')
```

4.1.3. Problem 1, Exercise 3

```
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 06 12:01:49 2016

@author: auort
"""
from PIL import Image
from numpy import *
from numpy import random
from scipy.ndimage import filters
import scipy.misc

"""
An alternative image normalization to histogram equalization is a quotient
image. A quotient image is obtained by diving the image with a blurred version
I/(I*G(sub-theta)). Implement this and try it on some sample images.
"""

directory = 'C:\\Users\\auort\\Desktop\\CV_L1\\'
```

```
def construct (identifier, theta, degree):
    #Original images
    im1  = array(Image.open(directory + identifier +'.jpg'))
    im2  = array(Image.open(directory + identifier +'_Grayscale.jpg'))

    #Blurred images
    G1 = filters.gaussian_filter(im1,theta)
    G2 = filters.gaussian_filter(im2,theta)

    D1 = im1*G1
    D2 = im2*G2

    S1 = im1 / D1
    S2 = im2 / D2

    R1 = im1 - S1
    R2 = im2 - S2

    scipy.misc.imsave(directory + 'P1\\E3\\' + identifier + '\\Color_' + degree +
'.jpg', R1)
    scipy.misc.imsave(directory + 'P1\\E3\\' + identifier + '\\Grayscale_' + degree
+ '.jpg', R2)


def run (n):
    construct(n, 1, '1')
    construct(n, 10, '10')
    construct(n, 20, '20')
    construct(n, 50, '50')
    construct(n, 100, '100')
    construct(n, 200, '200')
    construct(n, 300, '300')

"""
run('1')
"""
run('2')
run('3')
run('4')
run('5')
```

4.1.4. Problem 1, Exercise 5

4.1.5. Problem 2

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 02 13:16:43 2016

@author: auort
"""

from PIL import Image
from numpy import *
import scipy.misc

directory = 'C:\\Users\\auort\\Desktop\\CV_L1\\'
```

```
def histeq(im,nbr_bins=256):
    """ Histogram equalization of a grayscale image. """

    # get image histogram
    imhist,bins = histogram(im.flatten(),nbr_bins,normed=True)
    cdf = imhist.cumsum() # cumulative distribution function
    cdf = 255 * cdf / cdf[-1] # normalize

    # use linear interpolation of cdf to find new pixel values
    im2 = interp(im.flatten(),bins[:-1],cdf)

    return im2.reshape(im.shape), cdf

def construct (identifier):
    im  = array(Image.open(directory + identifier +'.jpg').convert('L'))
    scipy.misc.imsave(directory + identifier + '_Grayscale.jpg', im)
    im2,cdf = histeq(im)
    scipy.misc.imsave(directory +'P2\\'+identifier + '\\HistogramEqualization.jpg',
im2)

construct('1')
construct('2')
construct('3')
construct('4')
construct('5')
```

4.1.6. Problem 3

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 02 14:57:37 2016

@author: auort
"""

from numpy import *
from numpy import random
from scipy.ndimage import filters
import scipy.misc

directory = 'C:\\Users\\auort\\Desktop\\CV_L1\\P3\\'

def denoise(im,U_init,tolerance=0.1,tau=0.125,tv_weight=100):
    """ An implementation of the Rudin-Osher-Fatemi (ROF) denoising model
        using the numerical procedure presented in eq (11) A. Chambolle (2005).

        Input: noisy input image (grayscale), initial guess for U, weight of
        the TV-regularizing term, steplength, tolerance for stop criterion.

        Output: denoised and detextured image, texture residual. """

    m,n = im.shape #size of noisy image

    # initialize
    U = U_init
    Px = im #x-component to the dual field
    Py = im #y-component of the dual field
    error = 1

    while (error > tolerance):
        Uold = U
```

```
        # gradient of primal variable
        GradUx = roll(U,-1,axis=1)-U # x-component of U’s gradient
        GradUy = roll(U,-1,axis=0)-U # y-component of U’s gradient

        # update the dual varible
        PxNew = Px + (tau/tv_weight)*GradUx
        PyNew = Py + (tau/tv_weight)*GradUy
        NormNew = maximum(1,sqrt(PxNew**2+PyNew**2))

        Px = PxNew/NormNew # update of x-component (dual)
        Py = PyNew/NormNew # update of y-component (dual)

        # update the primal variable
        RxPx = roll(Px,1,axis=1) # right x-translation of x-component
        RyPy = roll(Py,1,axis=0) # right y-translation of y-component

        DivP = (Px-RxPx)+(Py-RyPy) # divergence of the dual field.
        U = im + tv_weight*DivP # update of the primal variable

        # update of error
        error = linalg.norm(U-Uold)/sqrt(n*m);

    return U,im-U # denoised image and texture residual

def construct(identifier):
    im  = array(Image.open(directory + identifier +'.jpg').convert('L'))
    U,T = denoise(im,im)
    G = filters.gaussian_filter(im,10)
    scipy.misc.imsave(directory + identifier + '\\Denoised_A.jpg', U)
    scipy.misc.imsave(directory + identifier + '\\TexturedResidual_A.jpg', T)
    scipy.misc.imsave(directory + identifier + '\\DenoisedGausian_A.jpg', G)

construct('1')
construct('2')
construct('3')
construct('4')
construct('5')
```

End of Document

End of Document