University of Amsterdam

Anton Pannekoek instituut

Basic Linux and Coding for AA (BLAC) Exercise 6 (second set called 6) (week 4)

Author:
Timo Halbesma, 6126561
Version 1.2

Supervisor:
Dr. T. Coenen

Manipulating images

Step 3

shape: (375, 500, 3)
dtype: float32
type: <type 'numpy.ndarray'>

Step 4

Listing 1: TLRH's solution for the BLAC homework 6 (week 4).

```
#!/usr/bin/python
# -* coding: utf-8 -*

# BLAC_ex6_Friday_6126561.py

# Basic Linux and Coding for AA homework 6 (Friday week 4)
# Usage: python BLAC_ex6_Friday_6126561.py
# TLR Halbesma, 6126561, september 26, 2014. Version 1.1; added usm
# todo: change functions such that not all separate steps require
#       creating the grayscale, separate channels, gaussian blur, etc again.

import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np

from BLAC_ex6_Friday_6126561_sobel import sobel_filtered
from BLAC_ex6_Friday_6126561_gaussian_blur import gaussian_blur
from BLAC_ex6_Friday_6126561_unsharp_mask import unsharp_mask


# http://matplotlib.org/users/image_tutorial.html
def plot_individual_channels(pngimage):
    # Step 2
    img = mpimg.imread(pngimage)

    # Step 3: three dimensional array (x, y, N), where x and y is the number
    # of pixels and N the number of channels (either 3 RGB, or 4 RGB alpha).
    print 'shape: ', img.shape, '\ndtype: ', img.dtype, '\ntype: ', type(img)

    fig = plt.figure()
    ax1 = fig.add_subplot(3, 2, 3)
    ax2 = fig.add_subplot(3, 2, 2)
    ax3 = fig.add_subplot(3, 2, 4)
    ax4 = fig.add_subplot(3, 2, 6)

    ax1.imshow(img)
    ax1.set_xticklabels([])
    ax1.set_yticklabels([])
    ax1.set_title('Original')

    red_img = img[:, :, 0]
    ax2.imshow(red_img, cmap='Reds')
    ax2.set_xticklabels([])
    ax2.set_yticklabels([])
    ax2.set_title('Red')

    green_img = img[:, :, 1]
    ax3.imshow(green_img, cmap='Greens')
    ax3.set_xticklabels([])
    ax3.set_yticklabels([])
    ax3.set_title('Green')

    blue_img = img[:, :, 2]
    ax4.imshow(blue_img, cmap='Blues')
    ax4.set_xticklabels([])
    ax4.set_yticklabels([])
    ax4.set_title('Blue')

    try:
        alpha = img[:, :, 3]
        # 0 is transparant, 255 is totally saturated.
    except IndexError:
```

```
        print 'No alpha channel present'
    else:
        ax5 = fig.add_subplot(3, 2, 5)
        ax5.imshow(alpha, cmap='binary')
        ax5.set_xticklabels([])
        ax5.set_yticklabels([])
        ax5.set_title('Alpha')

    fig.suptitle('RGB image and its separate channels')

    plt.savefig('BLAC_hw6_TLRH_6126561_separate_channels.pdf')


def lightness(img):
    R, G, B = img[:, :, 0], img[:, :, 1], img[:, :, 2]
    return (np.fmax(np.fmax(R, G), B) + np.minimum(np.minimum(R, G), B)) / 2.


def average(img):
    R, G, B = img[:, :, 0], img[:, :, 1], img[:, :, 2]
    return (R + G + B) / 3


def luminosity(img):
    R, G, B = img[:, :, 0], img[:, :, 1], img[:, :, 2]
    return 0.21*R + 0.72*G + 0.07*B


def plot_greyscale_images(pngimage):
    fig = plt.figure()

    ax0 = fig.add_subplot(3, 2, 3)
    ax1 = fig.add_subplot(3, 2, 2)
    ax2 = fig.add_subplot(3, 2, 4)
    ax3 = fig.add_subplot(3, 2, 6)

    ax0.imshow(mpimg.imread(pngimage))
    ax0.set_xticklabels([])
    ax0.set_yticklabels([])
    ax0.set_title('Orignal')

    ax1.imshow(lightness(mpimg.imread(pngimage)), cmap='binary')
    ax1.set_xticklabels([])
    ax1.set_yticklabels([])
    ax1.set_title('Lightness')

    ax2.imshow(average(mpimg.imread(pngimage)), cmap='binary')
    ax2.set_xticklabels([])
    ax2.set_yticklabels([])
    ax2.set_title('Average')

    ax3.imshow(luminosity(mpimg.imread(pngimage)), cmap='binary')
    ax3.set_xticklabels([])
    ax3.set_yticklabels([])
    ax3.set_title('Luminosity')

    fig.suptitle('RGB image and three greyscale methods')
    # fig.subplots_adjust(hspace=.5)

    plt.savefig('BLAC_hw6_TLRH_6126561_greyscale.pdf')


def plot_sobel_filtered(pngimage):
    fig = plt.figure()
    ax0 = fig.add_subplot(2, 1, 1)
    ax1 = fig.add_subplot(2, 1, 2)

    ax0.imshow(mpimg.imread(pngimage))
    ax0.set_xticklabels([])
    ax0.set_yticklabels([])
    ax0.set_title('Original')

    # Edges are also visible in the greyscale image.
    # sobel_filtered is in a different file, as requested.
    edges = sobel_filtered(luminosity(mpimg.imread(pngimage)))
    ax1.imshow(edges, cmap='binary')
    ax1.set_xticklabels([])
    ax1.set_yticklabels([])
    ax1.set_title('Sobel Filtered')
```

```
    fig.suptitle('Edge Detection: Sobel Method')
    plt.savefig('BLAC_hw6_TLRH_6126561_edges.pdf')


def plot_gaussian_blur(pngimage):
    red = mpimg.imread(pngimage)[:, :, 0]
    green = mpimg.imread(pngimage)[:, :, 1]
    blue = mpimg.imread(pngimage)[:, :, 2]

    fig = plt.figure()
    ax0 = fig.add_subplot(2, 1, 1)
    ax1 = fig.add_subplot(2, 1, 2)

    ax0.imshow(mpimg.imread(pngimage))
    ax0.set_xticklabels([])
    ax0.set_yticklabels([])
    ax0.set_title('Original')

    # gaussian_blur is in a different file, as requested.
    radius, sigma = 7, 0.84089642
    blurred_img = gaussian_blur(red, green, blue, radius, sigma)
    print type(blurred_img), blurred_img.dtype, blurred_img.shape
    ax1.imshow(blurred_img)
    ax1.set_xticklabels([])
    ax1.set_yticklabels([])
    ax1.set_title('Gaussian Blurred with kernel size {0} and sigma {1}'
                  .format(radius, sigma))

    fig.suptitle('Gaussian blur')
    plt.savefig('BLAC_hw6_TLRH_6126561_gaussian_blur.pdf')


def plot_unsharp_mask(pngimage):
    threshold, amount, radius, sigma = 0.02, 0.5, 2, 0.84089642
    red = mpimg.imread(pngimage)[:, :, 0]
    green = mpimg.imread(pngimage)[:, :, 1]
    blue = mpimg.imread(pngimage)[:, :, 2]

    rgb_image = np.dstack((red, green, blue))
    edges = sobel_filtered(luminosity(mpimg.imread(pngimage)))
    blurred_img = gaussian_blur(red, green, blue, radius, sigma)

    unsharp_mask(rgb_image, edges, blurred_img, threshold, amount)

    fig = plt.figure()
    ax0 = fig.add_subplot(2, 1, 1)
    ax1 = fig.add_subplot(2, 1, 2)

    ax0.imshow(mpimg.imread(pngimage))
    ax0.set_xticklabels([])
    ax0.set_yticklabels([])
    ax0.set_title('Original')

    ax1.imshow(mpimg.imread(pngimage))
    ax1.set_xticklabels([])
    ax1.set_yticklabels([])
    ax1.set_title('Unsharp Mask with threshold {0}, amount {1}'
                  .format(threshold, amount) +
                  ', radius {0} and sigma {1}'
                  .format(radius, sigma))

    fig.suptitle('Unsharp Mask')
    plt.savefig('BLAC_hw6_TLRH_6126561_unsharp_mask.pdf')


def main():
    # Step 1
    inputfile = './thunderbird_logo-only_RGB.png'

    # Step 4
    plot_individual_channels(inputfile)

    # Step 5
    plot_greyscale_images(inputfile)

    # Step 6
    plot_sobel_filtered(inputfile)

    # Step 7
    plot_gaussian_blur(inputfile)
```

```
    # Step 8
    plot_unsharp_mask(inputfile)

if __name__ == "__main__":
    main()
```

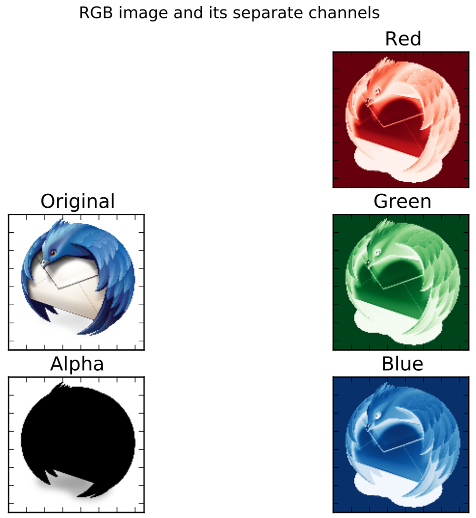

Figure 1: Different channels of the RGB png image.

Step 5

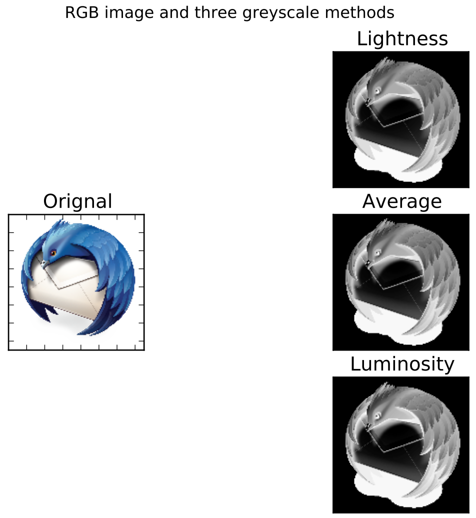

Figure 2: Grayscale applied to png image.

Step 6

Listing 2: TLRH's solution for the BLAC homework 6 (week 4) step 6.

```
#!/usr/bin/python
# -* coding: utf-8 -*

# BLAC_ex6_Friday_6126561_sobel.py

# Basic Linux and Coding for AA homework 6 (Friday week 4)
# Usage: import into BLAC_ex6_Friday_6126561.py
# TLR Halbesma, 6126561, september 29, 2014. Version 1.0; implemented

from scipy import signal as sg
import numpy as np


def sobel_filtered(gray_luminosity):
    # https://en.wikipedia.org/wiki/Edge_detection
    # First order Sobel method
    sobel_operator_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    sobel_operator_y = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])

    # From Scipy convolve2d documentation.
    l_x = sg.convolve2d(gray_luminosity, sobel_operator_x, 'same')
    l_y = sg.convolve2d(gray_luminosity, sobel_operator_y, 'same')

    # Gradient magnutide according to Wikipedia.
    magnitude_gradient = np.sqrt(l_x**2, l_y**2)

# https://stackoverflow.com/questions/7185655/applying-the-sobel-filter-using-scipy
    # One might have to normalize according to this stack overflow answer.
    # magnitude_gradient *= 255. / np.max(magnitude_gradient)

    return magnitude_gradient
```

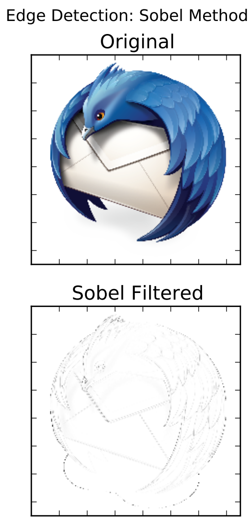

Figure 3: Sobel filter applied to png image to detect the edges.

Step 7

Listing 3: TLRH's solution for the BLAC homework 6 (week 4) step 7.

```
#!/usr/bin/python
# -* coding: utf-8 -*

# BLAC_ex6_Friday_6126561_gaussian_blur.py

# Basic Linux and Coding for AA homework 6 (Friday week 4)
# Usage: import into BLAC_ex6_Friday_6126561.py
# TLR Halbesma, 6126561, september 29, 2014. Version 1.0; implemented

from scipy import signal as sg
import numpy as np


def two_dim_gauss(x, y, sigma):
    return 1. / (2*np.pi*sigma**2) * np.exp((x**2 + y**2) / -2*sigma**2)


def gaussian_matrix(radius, sigma):
    kernel = np.array([[two_dim_gauss(x, y, sigma) for x in range(radius)]
                      for y in range(radius)])

    normalization_cst = 1. / np.sum(kernel)
    kernel *= normalization_cst

    return kernel


# https://en.wikipedia.org/wiki/Gaussian_blur
def gaussian_blur(red, green, blue, radius, sigma):
    kernel = gaussian_matrix(radius, sigma)

    # From Scipy convolve2d documentation.
    blurred_red = sg.convolve2d(red, kernel, 'same')
    blurred_green = sg.convolve2d(green, kernel, 'same')
    blurred_blue = sg.convolve2d(blue, kernel, 'same')

    return np.dstack((blurred_red, blurred_green, blurred_blue))
```

Figure 4: Gaussian blur applied to png image to detect the edges.

Listing 4: TLRH's solution for the BLAC homework 6 (week 4) step 8.

```
#!/usr/bin/python
# -* coding: utf-8 -*

# BLAC_ex6_Friday_6126561_unsharp_mask.py

# Basic Linux and Coding for AA homework 6 (Friday week 4)
# Usage: import into BLAC_ex6_Friday_6126561.py
# TLR Halbesma, 6126561, september 29, 2015. Version 1.0; implemented

import numpy as np


# https://en.wikipedia.org/wiki/Unsharp_masking
def unsharp_mask(rgb, edges, blur, threshold, amount):
    edges_rgb = np.dstack((edges, edges, edges))
    return rgb - blur * threshold + edges_rgb * amount
```

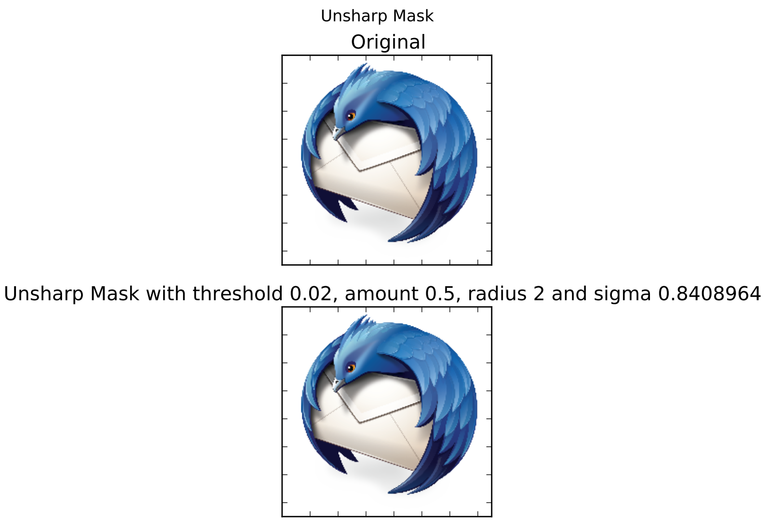

Figure 5: Unsharp Mask Technique applied to png image to detect the edges.