Edge Detection with OpenCV

In this exercise, we will look at OpenCV using its Python bindings.

Exercises

1. Test for Python 3.6 by running the following from the command prompt:

```
C:\python36\python.exe
```

2. Create a python file on your C drive. Navigate to its parent folder and run it by typing:

```
C:\python36\python.exe filename.py
```

3. Import OpenCV, numpy (for numerical calculations) and matplotlib (for plotting) as follows:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
```

4. Download the "GMIT.jpeg" image from Moodle into the same folder as your python file. Load the image into OpenCV as follows:

```
img = cv2.imread('GMIT.jpg',)
```

5. Using OpenCV, convert the GMIT image to grayscale (look up online how to achieve this). Plot the output image to verify that it is indeed grayscale as described in next step:

6. **Plotting images with Python and Matplotlib:**

   You can use: "cv2.imshow('image',img)" to plot a single image. Try this out. However, in order to plot multiple images in a window, we will use Matplotlib's "plt.subplot" function. This method can plot a number of images in a figure as follows. The docs for the function are here: http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.subplot

   For now specify nrows to be 2 and ncols to be 1. "plot_number" should correspond to the number of the plot in this grid. imgOrig should correspond to the original image loaded in and imgGray should correspond to the grayscale version

```
plt.subplot(nrows, ncols,1),plt.imshow(imgOrig, cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(nrows, ncols,2),plt.imshow(imgGray, cmap = 'gray')
plt.title('GrayScale'), plt.xticks([]), plt.yticks([])
plt.show()
```

Note: In order to fix subplot colour issues, plot "`cv2.cvtColor(imgOrig, cv2.COLOR_BGR2RGB)`" rather than "`imgOrig`" in above when dealing with colour images.

7. Images can be filtered with various low-pass filters (LPF), high-pass filters (HPF), etc. A LPF helps in removing noise, or blurring the image. A HPF filters helps in finding edges in an image. OpenCV provides a function, `cv2.GaussianBlur()`, to convolve a kernel with an image. As an example, we will try a simple averaging filter on an image. A 5x5 averaging filter kernel can be defined as follows:

$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Filtering with the above kernel results in the following being performed: for each pixel, a 5x5 window is centered on this pixel, all pixels falling within this window are summed up, and the result is then divided by 25. This equates to computing the average of the pixel values inside that window. This operation is performed for all the pixels in the image to produce the output filtered image. Try this code and check the result. Experiment with different kernel sizes – 3x3, 5x5, 9x9, 13x13.

```
imgOut = cv2.GaussianBlur(imgIn,(KernelSizeWidth, KernelSizeHeight),0)
```

8. Plot the result of the above filtering using a 2 row and 2 column subplot. The top row should contain the original image and the grayscale version. The bottom row should contain the image filtered with two different sized kernels.

9. Next, we'll perform edge detection using the Sobel operator. The Sobel operator detects horizontal and vertical edges by multiplying each pixel by the following two kernels:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

In code this is achieved as follows:

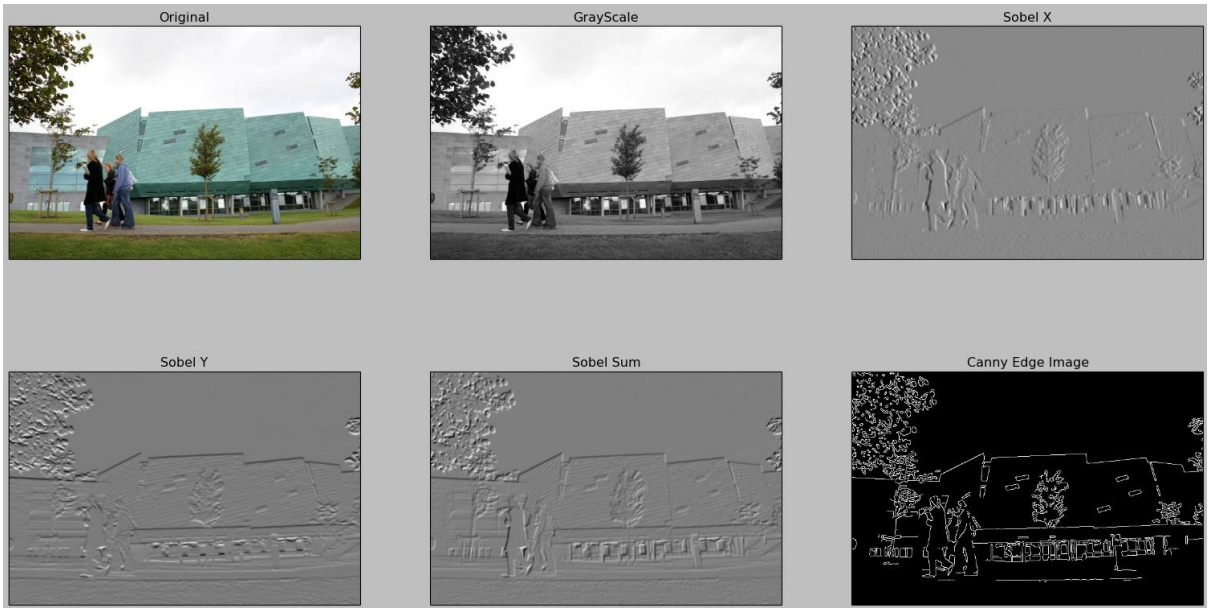
```
sobelHorizontal = cv2.Sobel(imgIn,cv2.CV_64F,1,0,ksize=5)  # x dir
sobelVertical   = cv2.Sobel(imgIn,cv2.CV_64F,0,1,ksize=5)  # y dir
```

10. Plot the two Sobel output images in a new figure

11. Add the two Sobel outputs together to create a new figure that combines horizontal and vertical edges. Plot this new figure

12. Perform Canny edge detection as follows and plot the result. Set the cannyThreshold to 100 initially. Set cannyParam2 to be 200 initially. cannyParam2 should be roughly 2 to 3 times the magnitude of cannyThreshold.

```
canny = cv2.Canny(imgIn,cannyThreshold,cannyParam2)
```



13. Trial all the above with another image of your choice. Experiment with different CannyThresholds to get an acceptable edge description of your image. Include this image in your submission.

Advanced exercises

1. Select a threshold for the Sobel sum image. Set all the values below the threshold to 0 and all above the threshold to 1 (Use a for loop to achieve this). Visualise the result for different thresholds

2. Manually write your own edge detector using a first derivative. Use a for loop to iterated over the image row by row and column by column and calculate the first derivative (difference) across all neighbouring pixels.