cs512 Assignment 1

Jiranun Jiratrakanvong
Department of Computer Science
Illinois Institute of Technology

October 1, 2015

Abstract

The objective of this assignment is to learn about basic image operations with OpenCV by conducting a program using python as a programming language. The program will perform basic image processing corresponding to the key pressing such as making grayscale image, getting color channels, smoothing the image, etc.

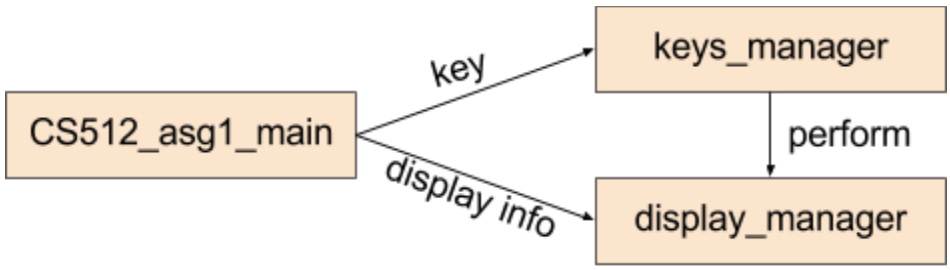
Problem Statement

The program from this assignment need to perform simple image manipulation using OpenCV**.** The program will load an image by either reading it from a file which is specified in the command line or capturing it directly from the camera. When the user presses a key perform the operation corresponding to the key on original image. The special keys on the keyboard are used to modify the displayed image as follows:

- <ESC>: quit
- 'i' - reload image
- 'x' - save the current image into the file out.jpg
- 'g' - convert the image to grayscale by openCV function
- 'G' - convert the image to grayscale by manual convolution
- 'c' - cycle through the color channels
- 's' - smooth the grayscale image by openCV function
- 'S' - smooth the grayscale image by manual convolution
- 'x' - convolve the grayscale image with an x derivative filter
- 'y' - convolve the grayscale image with an y derivative filter
- 'm' - show magnitude of the gradient
- 'p' - plot the gradient vectors of the image every N pixels
- 'r' - rotate the grayscale image
- 'h' - show help

Purpose Solution

Program was implemented into 3 files including CS512_asg1_main.py, keys_manager.py, and display_manager.py. First, the CS512_asg1_main will be place where the program begins. The implementation in this file will include loading image or capturing the image from the camera, image size calculation, setting window's name and getting key pressed. The display information, including original image, image size scale factor, window's name, will be sent to display_manager, and the key pressed will send to the keys_manager. Second, the keys_manager will call the function of display_manager corresponding to the key pressed as well. The file does not contain a lot of implementation here, but it make the codes easier to read, understand, and fix. Third, the display_manager will use the image information from CS512_asg1_main to display the image in the suitable size, which can be set. Moreover, all of the image operations will be performed here.



Overall process of the program

Start with the global parameter, if the program can neither read the specified image file or capture an image from the camera, the program will get the original image from the DEFAULT_IMG_FILE instead (please make sure that the DEFAULT_IMG_FILE is exist)

```
# check if image name is specified in command line
if len(sys.argv) == 2:
    original_image = cv2.imread(sys.argv[1])
    window_name = sys.argv[1]

    # Check if the image can be read
    if original_image is None:
        print "ERROR :", sys.argv[1], "can not be read. Use \"" + DEFAULT_IMG_FILE + "\" instead!"
        # get default image instead
        original_image = cv2.imread(DEFAULT_IMG_FILE)
        window_name = DEFAULT_IMG_FILE

# capture an image from the camera if image name is not specified
elif len(sys.argv) < 2:
    cap = cv2.VideoCapture(0)
    retval, original_image = cap.read()
    window_name = "From Your Camera"

# otherwise, get default image
else:
    original_image = cv2.imread(DEFAULT_IMG_FILE)
```

Another global parameter is MAXIMUM_SIDE_WIDTH. This parameter will be specified in order to control the display size of the window in case the original image is bigger than user's monitors. It will not effect to any process of the image operation. The original image and modified image will have the original size. From the code below, both width and height of display image will not bigger than MAXIMUM_SIDE_WIDTH, so the factor will be calculated and used in the display_manager. Also, the window size will be *[name of the image] - [width] x [height]* (i.e. myfile.jpg 630x630)

```
# Calculate resize factor in case image is too big to show
# resize factor will depend on MAXIMUM_SIDE_WIDTH
max_side = max(original_image.shape[0], original_image.shape[1])
if max_side > MAXIMUM_SIDE_WIDTH:
    dpm.setWindowSizeFactor(1.0 / (float(max_side) / float(MAXIMUM_SIDE_WIDTH)))

# Add image dimention to the window name
window_name = window_name + ' - ' + str(original_image.shape[0]) + 'x' + str(original_image.shape[1])
```

Every image operation will be performed at display_manager.py. These are the description of the algorithm implemented

```
def setWindowName(strWinName):...
def setWindowSizeFactor(factor):...
def showResizedImage(img):...
def setOriginalImage(img):...
def setCurrentImage(img = None):
    global current_image, original_image
    if img is None:
        current_image = original_image
    else:
        current_image = img
    showResizedImage(current_image)
```

After image modified by any operation, the program will call setCurrentImage() in order to save the modified image to the current image and show this current image.

- 'G' key (grayscale by conversion function) : the program uses NTSC formula as the conversion function as below.

```
def makeGrayScaleByOpenCV():
    global original_image
    setCurrentImage(cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY))

def makeGrayScaleByManual():
    global original_image
    gImg = np.ones((original_image.shape[0], original_image.shape[1]), np.float32)
    for i in range(0, original_image.shape[0], 1):
        for j in range(0, original_image.shape[1], 1):

            # NTSC conversion formula
            gImg[i][j] = 0.587 * float(original_image[i][j][0]) \
                       + 0.144 * float(original_image[i][j][1]) \
                       + 0.299 * float(original_image[i][j][2])

    normalized_image = np.uint8((gImg[:,:] - gImg.min()) * 255.0 / (gImg.max() - gImg.min()))
    setCurrentImage(normalized_image)
```

- ‘c’ key : the color channels of the image will be splitted by cv2.split()

```
def makeRGBChannelImage(ch):
    global original_image
    output_img = np.ones((original_image.shape[0], original_image.shape[1], 3), np.uint8)
    b, g, r = cv2.split(original_image)

    if ch == 0:
        output_img[:,:,0] = original_image[:,:,0]
    elif ch == 1:
        output_img[:,:,1] = original_image[:,:,1]
    else:
        output_img[:,:,2] = original_image[:,:,2]

    setCurrentImage(output_img)
```

- ‘s’ and ‘S’ key : First, the gaussian filter will be created. The gaussian filter size will depend on sigma which users can control by the trackbar. The value will be odd number only. This is the code which creating the filter

```
def calGaussian(x,y,sigma):
    return np.exp( (-1.0) * (np.power(x, 2.0)+np.power(y, 2.0)) / (2.0 * np.power(sigma, 2.0)))

def getGuessienFilter(sigma):
    filter_size = sigma * 5
    middle_position = filter_size / 2
    kernel = array([[calGaussian( i-middle_position, j-middle_position, sigma )
                     for i in range(0, filter_size)] for j in range(filter_size - 1 , -1, -1)])
    return kernel[:,:]/kernel.sum()
```

For the ‘s’ key, the filter will be convolved to the image by OpenCV function

```
cv2.createTrackbar('Sigma (OpenCV)', window_name, 0, 5, sliderHandlerForOpenCVSmoothing)
```

For the ‘S’ key, the filter will be manual convolved to every pixel of image. The boundaries of the image will be ignored.

```
middle_position = (sigma * 5) / 2
kernel = getGuessienFilter(sigma)
outputImg = np.ones((bw_image.shape[0], bw_image.shape[1]), np.uint8)
# convolve the image - use middle_position to ignore the boundaries
for i in range(middle_position, bw_image.shape[0] - middle_position, 1):
    for j in range(middle_position, bw_image.shape[1] - middle_position, 1):
        convolved_array = bw_image[i - middle_position: i + middle_position + 1,
                                   j - middle_position: j + middle_position + 1] * kernel
        outputImg[i][j] = convolved_array.sum()
setCurrentImage(outputImg)
```

- ‘x’ and ‘y’ key : first, the image will be convolved with the derivative filter. In this program, the sobel filter will be used. After the convolution, the image will be normalized and shown.

```
def getDerivativesofImage(isX):
    global original_image
    float_of_image = np.array(cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY), np.float32)
    if isX:
        kernel = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], np.float32)
    else:
        kernel = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]], np.float32)
    return cv2.filter2D(float_of_image, -1, kernel)

def showDerivativesofImage(isX):
    result = getDerivativesofImage(isX)
    normalized_image = np.uint8((result[:,:] - result.min()) * 255.0 / (result.max() - result.min()))
    setCurrentImage(normalized_image)
```

- ‘m’ key : the magnitude will be calculated based on x and y derivatives of images as the formula $\sqrt{\left(\frac{\delta I}{\delta x}\right)^2+\left(\frac{\delta I}{\delta y}\right)^2}$

```
def showMagnitudeofGradient():
    Xderv_img = getDerivativesofImage(True)
    Yderv_img = getDerivativesofImage(False)

    gradient_magnitude = np.sqrt((np.power(Xderv_img[:,:], 2.0) + np.power(Yderv_img[:,:], 2.0)))
    normalized_image = np.uint8((gradient_magnitude[:,:] - gradient_magnitude.min()) * 255.0
                                / (gradient_magnitude.max() - gradient_magnitude.min()))
    setCurrentImage(normalized_image)
```

- ‘p’ key : the pace of plotted vector can be controlled by users via trackbar. The length of vector will be 5% of the minimum length of image sides.

```
def sliderHandlerForVector(vector_pace):
    global original_image
    output_img = original_image.copy()
    Xderv_img = getDerivativesofImage(True)
    Yderv_img = getDerivativesofImage(False)
    vector_pace = (vector_pace + 2)
    vector_max_length = min(original_image.shape[0], original_image.shape[1]) * 0.05
    vector_length_factor = vector_max_length / np.max([np.sqrt((np.power(Xderv_img[:,:], 2.0) + np.power(Yderv_img[:,:], 2.0)))])
    for i in range(0, original_image.shape[1], vector_pace):
        for j in range(0, original_image.shape[0], vector_pace):
            vector = ( int(i + Xderv_img[j][i] * vector_length_factor), int(j + Yderv_img[j][i] * vector_length_factor) )
            cv2.line(output_img, (i,j), vector, (0,0,0))
            cv2.line(output_img, (i,j), (i,j), (0,0,255))
    setCurrentImage(output_img)

def addVectors():
    cv2.createTrackbar('Pace', window_name, 0, 15, sliderHandlerForVector)
```

- ‘r’ key : the degree can be controlled by users via trackbar. The rotation matrix will be created first, and the apply to the image

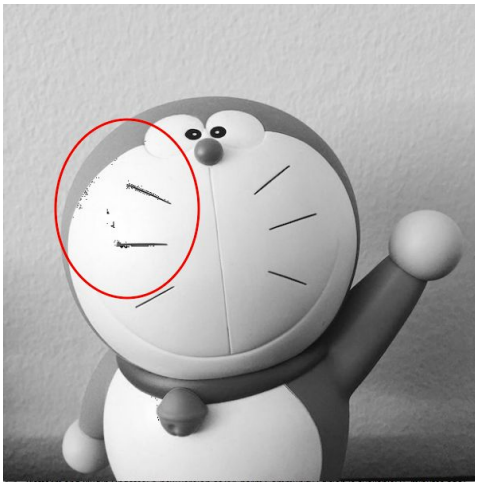
```
def rotateHandler(degree):
    global bw_image
    angle = degree * np.pi
    rows = bw_image.shape[0]
    cols = bw_image.shape[1]
    matrix = cv2.getRotationMatrix2D((cols / 2, rows / 2), angle, 1)
    setCurrentImage(cv2.warpAffine(bw_image, matrix, (cols, rows)))

def rotateImage():
    BW_ORIGINAL_IMG = cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY)
    setCurrentImage(BW_ORIGINAL_IMG)
    cv2.createTrackbar('Degree', window_name, 0, 125, rotateHandler)
```

Implementation Details

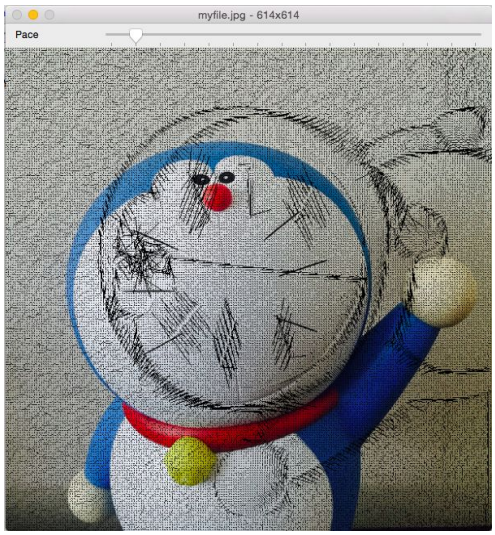
Problem and Solution

- Python - I was new to python. I used to implement the program by using C++, but python is not too hard to learn so I can learn it from the document teacher gave, and on the internet. Keeping practice makes me better now.

- Data type - Because the image is uint8, when the program calculated something, the overflowing occurred. It made some pixel of the display image become black. At first, I did not know that it was because of the data type, so I tried to print the value of the calculation result which was more than 255. Finally, I found some result was 255.34. That made the black dot because the the image can be only 255. The issue was fixed by calculate the value by float and then normalize it to int



- Assignment specification - some of them were not so clear for me. I solved this problem by asking the professor, and sent emails to TA.



- Incorrect Vectors - after I finished the 'p' key function, I tested it with an image, and I found that the vectors were rotated as below.Because the image is an array which is lists of the lists of column. I solved it by swap the iterator from image[i][j] to image[j][i].

Instructions for using the program

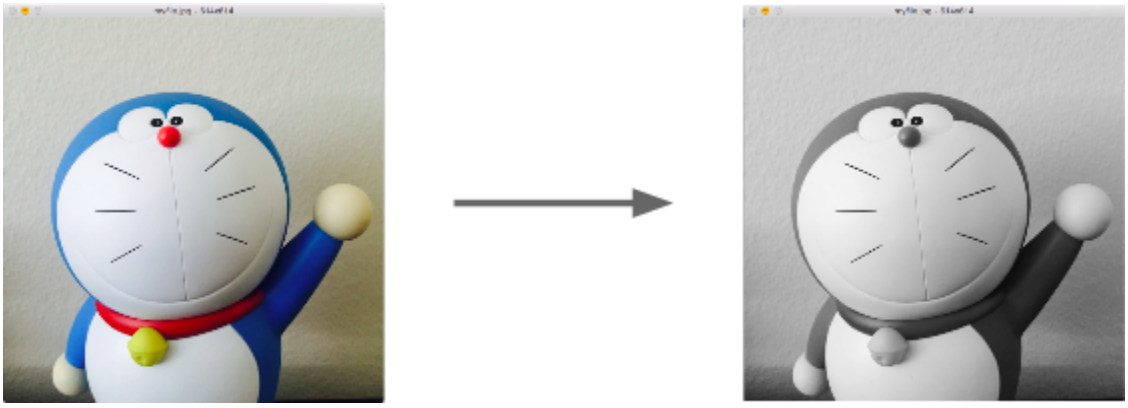
- Global parameter :
    a. DEFAULT_IMG_FILE : the default image file will be read when the user's specified file does not exist.
    b. MAXIMUM_SIDE_WIDTH : when the image is too big to show on the user's screen, the maximum length of image sides will be reduced to MAXIMUM_SIDE_WIDTH. This parameter will not effect to any logic of the image operation.
- Start by command line : *python CS512asg1_main.py "myfile.jpg"*

Results and discussion

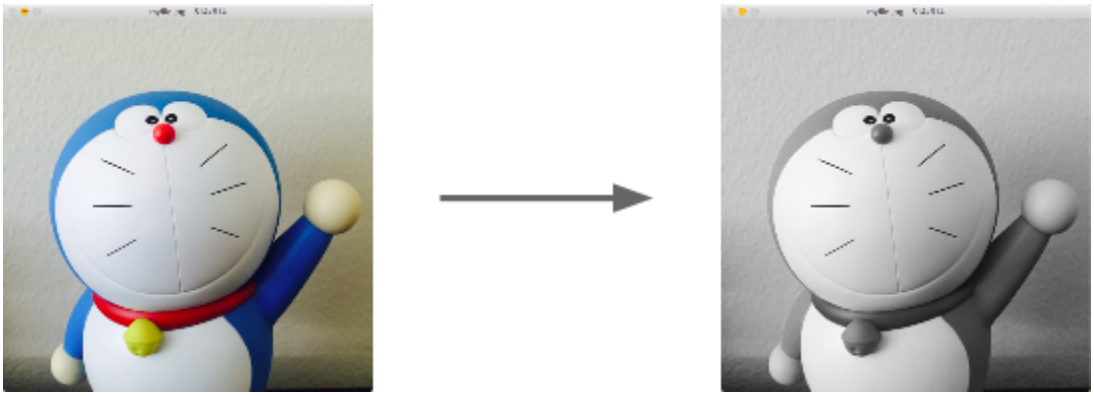
First, when the program started via command line, the original image will be read and shown in a new window.

Then, the result of the image will be performed corresponding to the key pressed.

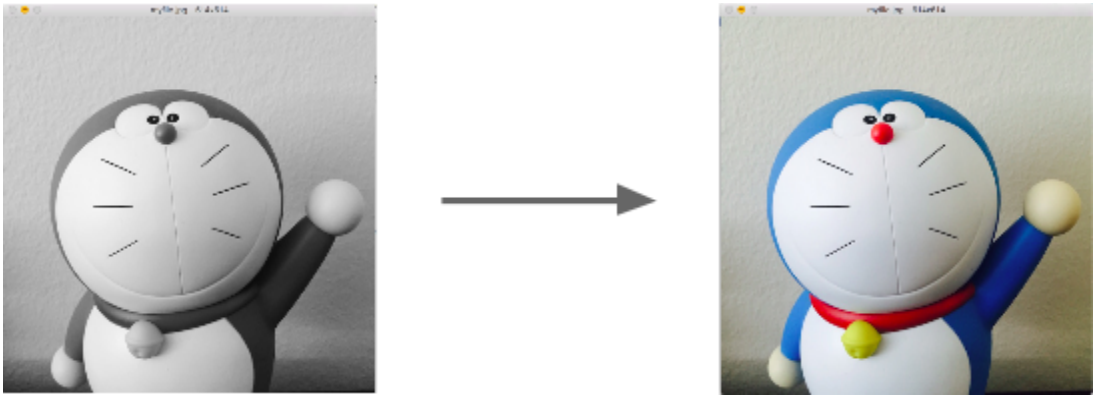
- ’g’ - convert the image to grayscale using the openCV conversion function.



- ’G’ - convert the image to grayscale using your implementation of conversion function (the program will take more time if the image has higher resolution)



- ’i’ - reload the original image (i.e. cancel any previous processing)

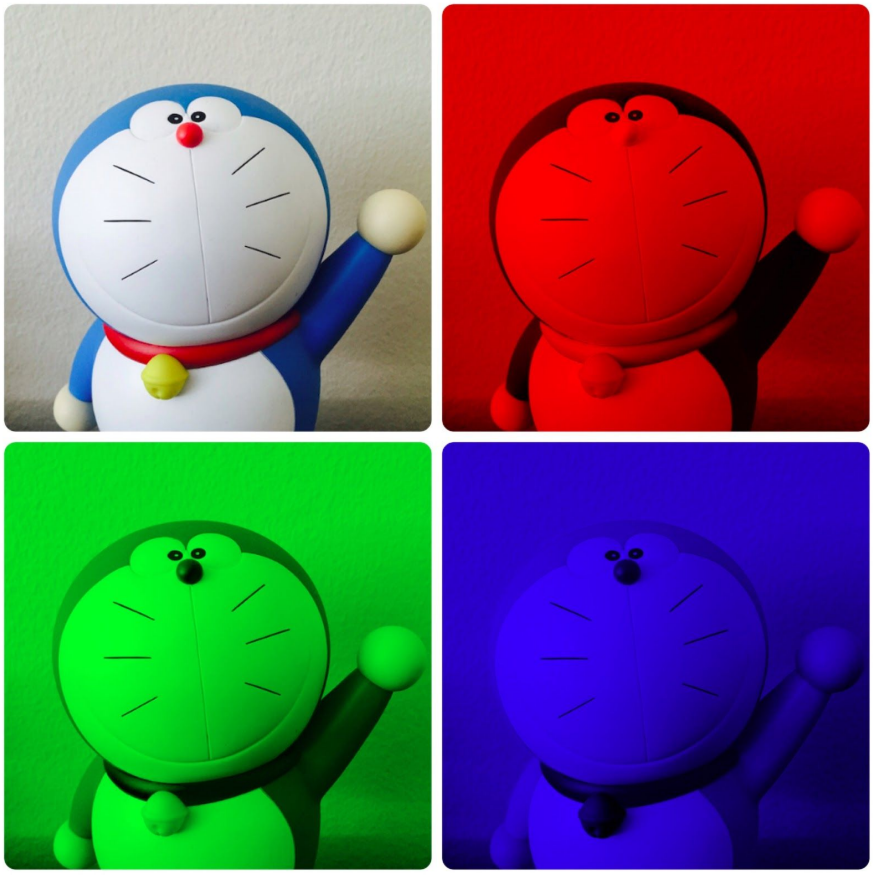


- ’w’ - save the current (possibly processed) image into the file ’out.jpg’

myfile.jpg

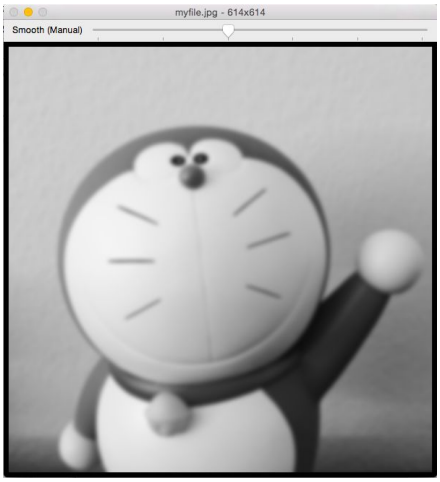
out.jpg

- ’c’ - cycle through the color channels of the image showing a different channel every time the key is pressed.

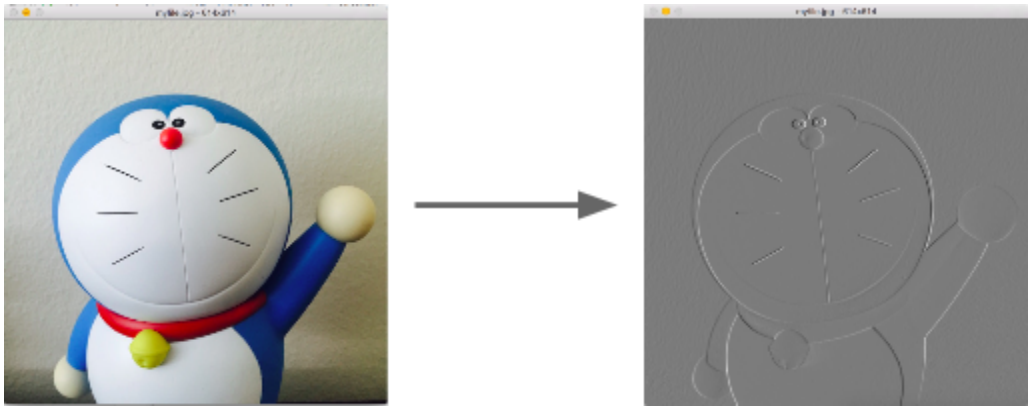


- ’s’ - convert the image to grayscale and smooth it using the openCV function. Use a track bar to control the amount of smoothing (sigma)

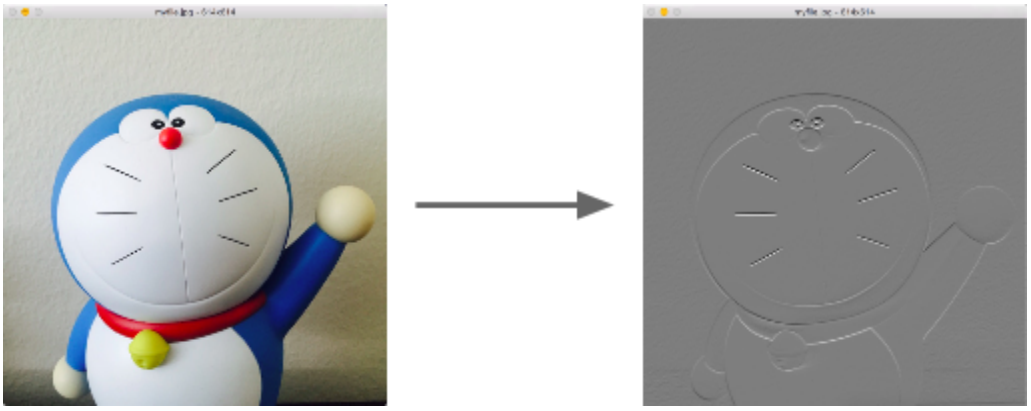
- ’S’ - convert the image to grayscale and smooth it using manual convolution. Use a track bar to control the amount of smoothing (sigma) (boundaries ignored)



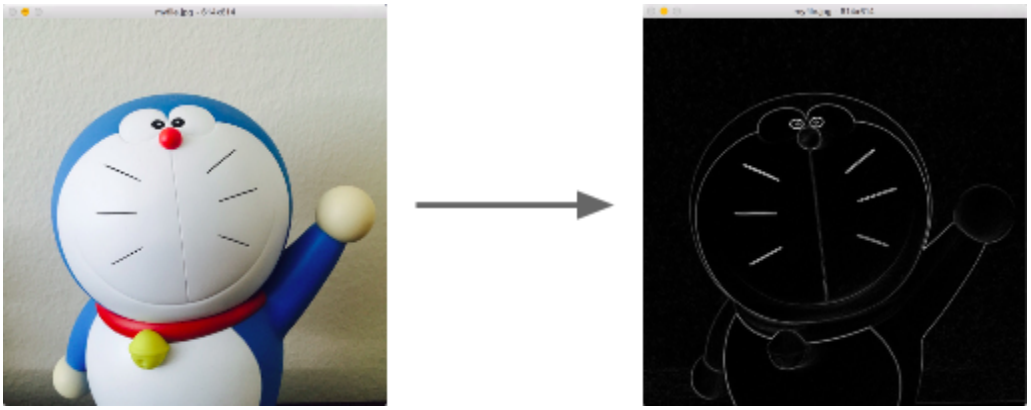
- ’x’ - convert the image to grayscale and perform convolution with an x derivative filter. Normalize the obtained values to the range [0,255].



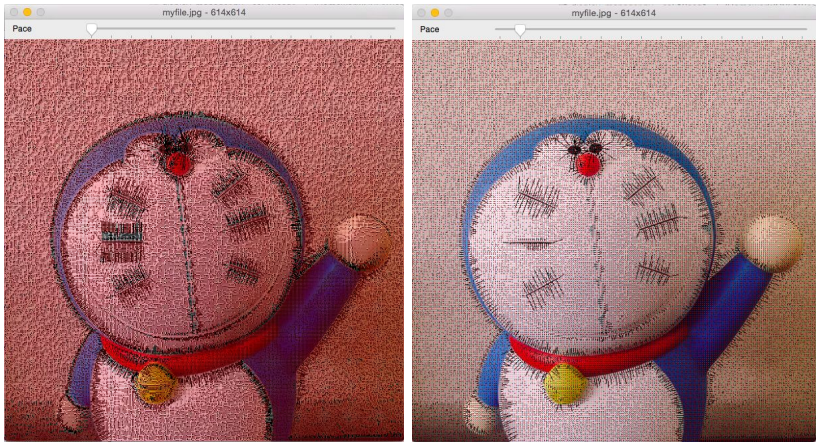
- ’y’ - convert the image to grayscale and perform convolution with a y derivative filter. Normalize the obtained values to the range [0,255].

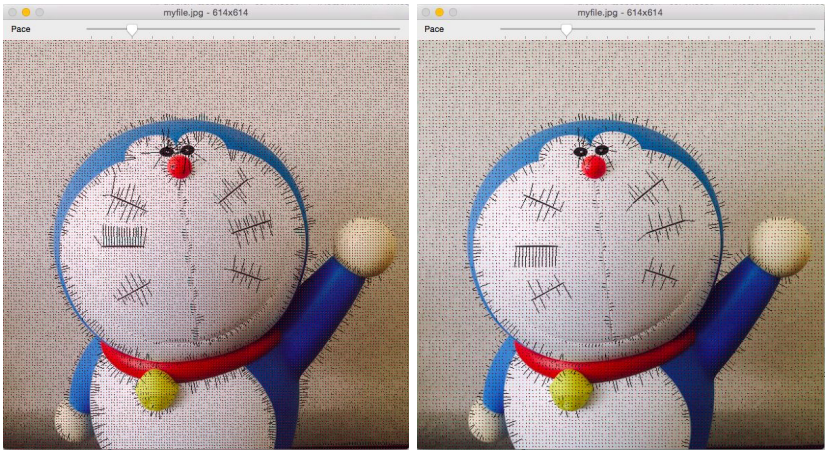


- ’m’ - show the magnitude of the gradient normalized to the range [0,255]. The gradient is computed based on the x and y derivatives of the image.



- ’p’ - plot the gradient vectors of the image every [Pace] pixels and let the plotted gradient vectors have a length 5% of the minimum length of image sides. The vector will be black line, and the start point will be red.





- ’r’ - convert the image to grayscale and rotate it using an angle of Q degrees. Use a track bar to control the rotation angle. The rotation of the image should be performed using an inverse map so there are no holes in it.

- ’h’ - Display a short description of the program, its command line arguments, and the keys it supports.

```
<ESC>: quit
'i' - reload image
'x' - save the current image into the file out.jpg
'g' - convert the image to grayscale by openCV function
'G' - convert the image to grayscale by manual convolution
'c' - cycle through the color channels
's' - smooth the grayscale image by openCV function
'S' - smooth the grayscale image by manual convolution
'x' - convolve the grayscale image with an x derivative filter
'y' - convolve the grayscale image with an y derivative filter
'm' - show magnitude of the gradient
'p' - plot the gradient vectors of the image every N pixels
'r' - rotate the grayscale image
'h' - help
```

Strength and weaknesses

- Program can process any size image, and the image will be shown in the appropriate size.
- The gaussian filter for smoothing is created by gaussian formula, and the size of filter is equal to sigma x 5. Sigma will be set by trackbar. It will be odd number only in order to calculate easier.
- The sobel filter for ‘x’, ‘y’, ‘m’, and ‘p’ is hard code, so it can not change the sigma or filter size.

Evaluation

Every operation was evaluated, and tested by a various kind of images. i.e..

References

- Gady Agam - *CS512: Computer Vision, Introduction to Python, and Using Python with OpenCV*, Department of Computer Science, Illinois Institute of Technology
- OpenCV, *Splitting and Merging Image Channels,* http://docs.opencv.org/master/d3/df2/tutorial_py_basic_ops.html

$$W_i = \sqrt{\frac{TM(P_i)}{TC(P_i)}}$$