Exercises 3 (Week 3) - Point processing

All exercises are relevant for the course but none are mandatory. You are highly advised to do as many exercises as you have time for. We have made enough exercises so that there is enough material to do within class hours as well as extra exercises (marked as *"Extra"*) for those who want to do more outside class. None of the exercises should take very long to solve so don't spend too much time not making progress. In case you need help don't hesitate to ask as we can probably give you a hint or two to guide you :). You are welcome to send questions by email.

Learning goals In these exercises you will be experimenting with point-wise processing of images (thresholds, contrast stretching etc. in both grayscale and color images) using OpenCV and pylab libraries in python. The exercises are meant to exemplify the contents of the lecture on low-level processing of images, but they simultaneously serve to expand your experience with python. Several of the exercises contain elements that will be useful later in the course and for your assignment. Notice point processing one of the main learning goals of the course and essential for the exam.

Get Exercise Data

Download *"Exercises_3.zip"* from LearnIT and extract the zip file and set that folder as the source folder of your project.

1 Loading And Displaying Images

In this exercise you will get more experience with loading, showing and saving images in python. Some of the functions for loading and displaying images used in these exercises were actually used in the exercises week 2.

1. Open the file *Test1.py* of your source folder. The file contains functions for showing images using OpenCV or pylab. The module loads an image(*children.tif*). Load two more images *Eye1.jpg* and *Marker1.jpg* and call them *I2* and *I3*. Use the function *cvtColor()* and convert the images to grayscale:

```
I1=cv2.cvtColor(I1, cv2.COLOR_RGB2GRAY)
```

Showing images using OpenCV

2. Call the functions *show1_OpenCV* and *show2_OpenCV* to show one and two images (examples are given below). Examine the functions and see how images are displayed using OpenCV.

```
#Examples of how to call the functions
show1_OpenCV(I1)# showing one image using OpenCV
show2_OpenCV(I1,I2)# showing two images using OpenCV
show1_pylab(I1)# showing one image using pylab
show2_pylab(I1,I2)# showing two images using pylab
```

3. Define a function called *showAll_OpenCV(**image)* that takes multiple images ("Collecting Parameters" is mentioned in the first week lecture) and display them. Call your own function as below:

```
showAll_OpenCV(image1=I1,image2=I2, image3=I3)
```

Showing images using pylab

4. Call *show1_pylab* and *show2_pylab* and see how you can display one image or two images using the *subplot()* function in pylab. Inspire by *show2_pylab*.

5. Define a function called *showAll_pylab(**image)* that takes multiple images and display them in one window.Call your own function as below:

```
showAll_pylab(image1=I1,image2=I2, image3=I3)
```

Saving the images using OpenCV

6. Use *cv2.imwrite()* to save the grayscale images in the source folder.

2 Graylevel Mappings

In this exercise you will experiment with gray level mappings.

1. Define a function, *grayLevelMap2(I,vector)* in the *Test1.py* that given a grayscale image *I* and a *vector*, does a gray mapping on the image using the values in the vector for the mapping. The function *vector* is actually a list that defines the 255 values needed for the mapping, for example *vector=[255,254,253,...,0]* which has been shown in figure(1) can be used fro inverting an image (for instance the second element of the vector defines the value (254) that a gray value=2 should be mapped to it).

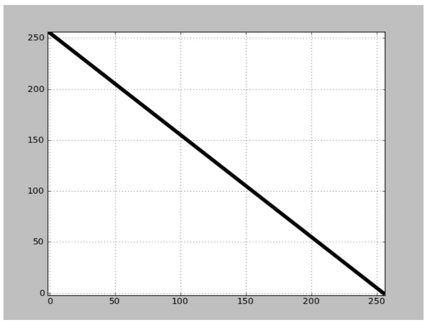
figure 1

Define different vectors that have been illustrated in figure(2), and test the *grayLevelMap2* with these vectors on e.g. *flag-dk.jpg* and *children.tif*: (*hint:* use the $min()$ and $max()$ function together with lambda for controlling the $underflow$ and $overflow$):

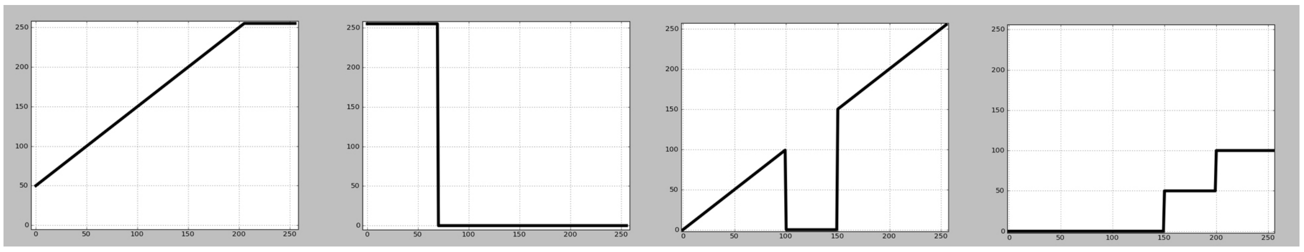
figure 2

You can display a defined vector in a figure using the function below:

```
def displayFunc(vector):
    points=[]
    for i in range(len(vector)):
        points.append((i,vector[i]))

    bins=range(256)
    n=vector
    plt.grid(None, 'major', 'both')
    plot(bins, n, 'k-', linewidth = 5)
    axis([-2, 256, -2, 256])
    plt.show()
```

2. Make a function *grayLevelMap(I,a,b)* that given a grayscale image *I*, changes the brightness and the contrast of *I* according to the mathematical function

$$g(x, y) = a * f(x, y) + b$$

   Test the function on *GreenTest.jpg* image with different values for a and b.

3. Copy the function below in to the *Test1.py*:

```
def f(x):
    func=(255-x)*sqrt(100/x+1)
    return func
```

   Change the *grayLevelMap(I,f)* function so that given a grayscale image *I*, and a function *f* returns an image where the function *f* has been applied to each pixel in *I*. In this way you can define any function *f* and apply it to an image.

4. $^{(Extra)}$ Change the function *grayLevelMap* to the following signature *grayLevelMap(I,func)* that given a grayscale image *I* and a function *func* written as a string, applies the function *func* to each element in *I*. For example `grayLevelMap(I1,'a*x+b')` does the same linear mapping as before. Changes the brightness and the contrast of the image. (hint: use the *eval(func)* for converting the string expression to an expression)

5. $^{(Extra)}$ Test the function *grayLevelMap* on *GreenTest.jpg* image and a Gamma correction function like $0.5x^{1.3}$. It should returns an image like below:

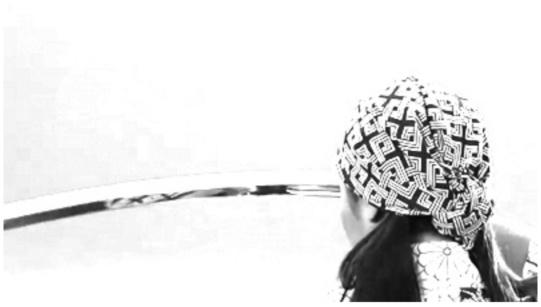

figure 3: GreenTest.jpg mapped with the $0.5x^{1.3}$ function

6. $^{(Extra)}$ Load the *Eye1.jpg* image and apply the gamma correction on this image using different values of gamma. For which value it is mostly the pupil that is displayed.

7. $^{(Extra)}$ Test the *grayLevelMapp* function on *GreenTest.jpg* with some other functions, e.g. , $30 * log(x + 1), 5 * *log(x + 1)and(255 - x) * sqrt(100/x + 1)$., and play with the values.

8. $^{(Extra)}$ In this exercise you will be experimenting with different thresholding techniques in opencv (thresholding, automatic and adaptive thresholding). The main difference of adaptive thresholding is that , adaptive thresholding adapts the threshold value on each pixel to the local image characteristics while global thresholding methods dont. As a result, in some conditions, such as bad lighting conditions, adaptive thresholding gives better results.

   Open the *threshold.py* file, which contains several examples of thresholding methods.

   (a) Apply different methods of thresholding to the following images: *marker1.jpg, marker2.jpg, marker3.jpg and marker4.jpg*

(b) Which method separates the markers the best?
(c) Why does the method perform better on some images and not others
(d) What happens if you set the last parameter of the adaptiveThreshold function to 0 when testing on *marker4.jpg*)?

3 Image Histograms

In this exercise you will experiment with image histograms and histogram equalization using OpenCV.
The *histogram.py* file contains an example of loading an image, creating the histogram using the *numpy.histogram()* function and displaying the histogram.

1. Load the *GreenTest.jpg* image and display it. Now extract the RGB channels and display the histogram of each channel. You can get individual color channels either through slicing (e.g. I[:,:,0]) or by using *cv2.split()*. You will see a peak in each histogram. What is the connection between these peaks and the background area of the image? Which values in the RGB channels correspond to the green background?
2. There is an example in *histogram.py* for histogram equalization using the *equalize* function. Use this example and show the histogram before and after histogram equalization of the following images and compare the results with the original images: *Eye1.jpg*,*Eye2.jpg*,*Eye3.jpg* and *fprint.jpg* Notice how histogram equalization increases the contrast and enhance the image.
3. Open the *colorProblem.jpg* image in the *histogram.py*. Extract the RGB channels. Display the histogram of each channel. In which channels do you have a low contrast?
4. $^{(Extra)}$ Equalize the blue channel of the *colorProblem.jpg* image and merge the channels again. Display the resulting image and compare it with the original image. You will be surprise by the result! :) (hint: *cv2.merge()*)

4 Image Arithmetic

This exercise is about image arithmetic and specifically background subtraction. Here we want to implement the background subtraction method for extracting the Swordswoman from the green screen video in the *GreenScreen(lowMP4).mp4* file.

1. Open the *backgroundSubtraction.py* file and run the program. When running the file the *GreenScreen(lowMP4).mp4* file will be displayed.
2. Change the program so that it converts each frame to grayscale (cv2.cvtColor()) and displays the grayscale video.
3. There is a variable *backGround* defined in the nwbackgroundSubtraction.py file. Subtract each grayscale image in the image sequence with the background image *backGround* (converted to gray). Run and test the program. Can simple subtraction extract the moving Swordswoman from the image?
4. $^{(Extra)}$ What happens if you calculate the absolute difference of each pixel (Hint: use *np.abs()*)
5. Replace subtraction with the *cv2.absdiff()* function. Does it improve the result? What is the problem of using the " $-$ " operator?
6. Apply a binary thresholding on the difference image. Which threshold values gives the best results (start with values around 30-50)? (*threshold.py* file, contains several examples of thresholding methods in cv2)
7. $^{(Extra)}$ Extract the RGB channels of each frame and the background image. Subtract the corresponding channels of each frame and the background image using *absdiff*. Add the results of the three channels using the *cv2.addWeighted()* function with values $alpha, Beta = 0.5$. Run the program. Does it make any difference? (Hint: $\alpha_2(\alpha_1 R_{added} + \beta_1 G_{added}) + \beta_2 B_{added}$)
8. Load the *GreenScreen(lowMP4)_Background.jpg* image. We want to use this image instead of green background in the Swordswoman video. Modify the program so that in each frame it replaces the green background with this image.(Hint: cv2.bitwise_and(colored image1, colored image2))

5 Image Downsampling

This exercise is about downsampling (reduce the size) of images.

1. Load the image *zone.jpg* in the *Test1.py*. Within *Test1.py* define a function, *resizeImgOdd(I)*, that given an image I, uses slicing to remove all odd rows and columns in the image and returns the resulting image.

   Test the method using *zone.jpg* and show the original image and downsampled image using *show2_OpenCV()*. Are there any differences between the images?

2. In the previous question (1) the images were actually resized to half of their original size (downsampling by sampleFactor=2). Make a function *resizeImg(I,sampleFactor)* that takes an integer *sampleFactor* and downsamples the image *I* by sampleFactor. Resize the *zone.jpg* image with the values: 2,3,4,5. Display the resulting images. Which effects do you observe when changing sampleFactor?

3. $^{(Extra)}$ Load *circular zone plate.png* image and resize it with sampleFactor 2,3,4 and 5. You see that the pattern changes when you resize it! This effect is called *"Aliasing"* and is an important concept in graphics too.

4. In OpenCV image resizing can be done using the *cv2.resize(image,dsize)* function (*dsize* defines the size of the destination image). For example the following snippet loads and downsamples the image *'someImage.jpg'* to an image of size 200×100:

   ```
   I = cv2.imread('someImage.jpg')
   cv2.resize(I,(200,100))
   ```

   Make a function *resizeImgOpenCV(I,s)* that resizes an image I with scalefactor,s using *cv2.resize*.

5. $^{(Extra)}$ Resize the *zone.jpg* image with the given scalefactors: 0.3,0.5,2,3, and show the results. Do you get the same results as before?

6 Croma Keying

The purpose of this exercise is experiment with color-based pixel processing and are illustrated through blue/green screening (chroma keying). The exercises also introduces how to read and write video files.

1. Create a new python file called *GreenScreen.py*. Load and display the image *GreenTest.jpg*.

2. Define a function *ChangeGreenToWhite(I,GThr)* which given an image *I* and a threshold value *Gthr* returns an image where potential pixels are replaced with white ($(255,255,255)$). Perform thresholding on the green channel only. Replace each pixel in *I* with white when the corresponding pixel in the green channel is are greater than *GThr*. You may use the built-in function *nonzero(I)* to find the indices of the pixels which are different from 0/False.

3. Use *GreenTest.jpg* to experiment with *ChangeGreenToWhite(I,GThr)* by using different values of *Gthr* to get the best results.

4. Load the *GreenTest_Background.jpg* in *GreenScreen.py*. Make a function *ChangeGreenRGB(I,backI,GThr)* that given images, *I* and *backI* (background) and a threshold *GThr*, that replace the green pixels in *I* with the pixel with the same coordinates in *backI*. Test the modified function with the *GreenTest_Background.jpg* image as the background.

5. The method as you have implemented so far may fail (for example when the person is wearing white). This problem can be solved when using all the color channels (R,G,B). Extend the *ChangeGreenRGB* function so that it uses all channels to detect the green back ground.

   That's all folks, that is green screening!

6. $^{(Extra)}$ You can make green screening independent on the intensity by using the HSV color space. Make a function *ChangeGreenHSV(I,backI,(Hthr,Vthr))* that detects the green color in HSV space using thresholding and replaces the background with backI.

7. The following example shows how to load the an image sequence using opencv and saves it again in another file

```
from pylab import *
import cv2
import cv

cap = cv2.VideoCapture("GreenScreen(lowMP4).mp4")
writer = cv2.VideoWriter('GreenScreen_Output.avi', cv.CV_FOURCC('D','I','V','3'), 15.0,
    (640,360), True) #Make a video writer for
cv2.namedWindow("input")
running=True
while(running):
    running,I = cap.read()
    if (running):
        #Do your code here resulting -> imgNew
        writer.write(I)#Write the image
        cv2.imshow("input", array(I))
        ch = cv2.waitKey(1)
        if ch == 32:#  Break by SPACE key
            break
writer.release()
```

Use this example to do chroma keying on each image in the *GreenScreen(lowMP4).mp4* file. In this video you have a sequence of the green screen images like the *GreenTest.jpg*. Use the *GreenScreen(lowMP4)_Background.jpg* image as background and apply the *ChangeGreenRGB* function to every frame in the video. It will be probably too slow when display the processed video in real time (Compare the result video (speed and accuracy) with the result of the exercise(4-8)).