ELE510 Image Processing with robot vision: LAB, Exercise 1, Fundamentals.

Purpose: *To learn some basic operations on images using Python, OpenCV and other packages. The emphasis is on the fundamentals of digital images.*

The theory for this exercise can be found in chapter 1 and 2 of the text book [1]. Supplementary information can found in chapter 1, 2 and 3 in the compendium [2]. See also the following documentations for help:

- OpenCV
- numpy
- matplotlib

IMPORTANT: Read the text carefully before starting the work. In many cases it is necessary to do some preparations before you start the work on the computer. Read necessary theory and answer the theoretical part frst. The theoretical and experimental part should be solved individually. The notebook must be approved by the lecturer or his assistant.

Approval:

The current notebook should be submitted on CANVAS as a single pdf file.

To export the notebook in a pdf format, goes to File -> Download as -> PDF via LaTeX (.pdf).

Note regarding the notebook: The theoretical questions can be answered directly on the notebook using a *Markdown* cell and LaTex commands (if relevant). In alternative, you can attach a scan (or an image) of the answer directly in the cell.

Possible ways to insert an image in the markdown cell:

```
![image name]("image_path")
```

```
<img src="image_path" alt="Alt text" title="Title text" />
```

Under you will find parts of the solution that is already programmed.

You have to fill out code everywhere it is indicated with `...`

The code section under `######## a)` is answering subproblem a) etc.

Problem 1

a) What does a pixel value represent? What is the standard bit depth and why is it the common practice?

Proposed answer (a):

The pixel value represent the amount of light captured. In a RGB color image, the amount is split among the three color channels. These values are quantized into a finite discrite number. The range of this quantized value is defined by the bit depth b, which is represented as 2^b. The standard value for b corresponds to 8, since 8 bits correspond to 1 byte and the byte was originally the smallest number of bits that could hold a single character in standard ASCII.

b) What is the difference between image processing and image analysis? What primary problems are they trying to solve?

Proposed answer (b):

Image processing is the field of study in which algorithms operate on input images to produce output images, whereas image analysis is the field of study in which algorithms operate on images to extract higher level-information. While an image processing algorithm outputs another image, an image analysis algorithm outputs a non-image type of data structure.

The main problems in image processing are: 1) Enhancement: transform an image to improve its visual appereance 2) Restoration: restore an image corrupted by noise 3) Compression: store an image with fewer bits than that by the original image

The main problems in image analysis are: 1) Segmentation: process of determining which pixels in the image belong together 2) Classification: determine which pixels in the image corresponds to a predifined class 3) Shape from X: aims to recover the 3D structure fo the scene

c) Given a storage space of 512 GB, we would like to save a digital video recorded in grayscale that fills all the available memory. The video was recorded with no compression, a frame rate of 100 frames per second, and an image frame of 2048×1024 pixels. Then, how long would the lenght of the video be in time? Please provide an answer in HH:MM:SS format.

Proposed answer (c):

The amount of time is computed in seconds as:

$$\frac{512GB}{1B \cdot 2048pixels \cdot 1024pixels \cdot 100fps} = \frac{512 \cdot 10^9 B}{1B \cdot 2048pixels \cdot 1024pixels \cdot 100fps} = 2,441.40625s$$
$$= 00{:}40{:}41\text{HH:MM:SS}$$

Problem 2

In this problem we use one image, `flower.jpg` (relative path: `./images/flower.jpg`).

a) Import the image; let the name of the flower image be **A**. Find the following properties: height, width, channels, filesize [+]. Be aware tha opencv represents image colar channel in the order BGR (blue, green, red) instead of RGB as is more common. Matplotlib use RGB, so if we are using matplotlib to show images they need to be converted first.

b) Image **A** is represented as a 3D array in Python. With **A** as input we now want to extract 4 different 2D images:

- **R** representing the red colour component,
- **G** representing the green colour component,

- **B** representing the blue colour component, and
- **Gr** representing a grey level version.

The rgb components are found by using `A[:,:,k]` where `k=1,2 and 3`. The grey level image can be imported using a particular flag (`cv2.IMREAD_GRAYSCALE`), or converted from an already imported color-image to grayscale (find the cv2 function yourself in the documentation). Use `matplotlib` to display the colour image and the 3 colour components in the same figure.

Describe how the different colour components contributes to different parts of the image (the petals and the background). Show the gray level image in a separate figure. Describe this image in relation to the colour components.

> The filesize can be checked in **bytes** using the following commands: ```python import os filesize = os.path.getsize(my_path) ```

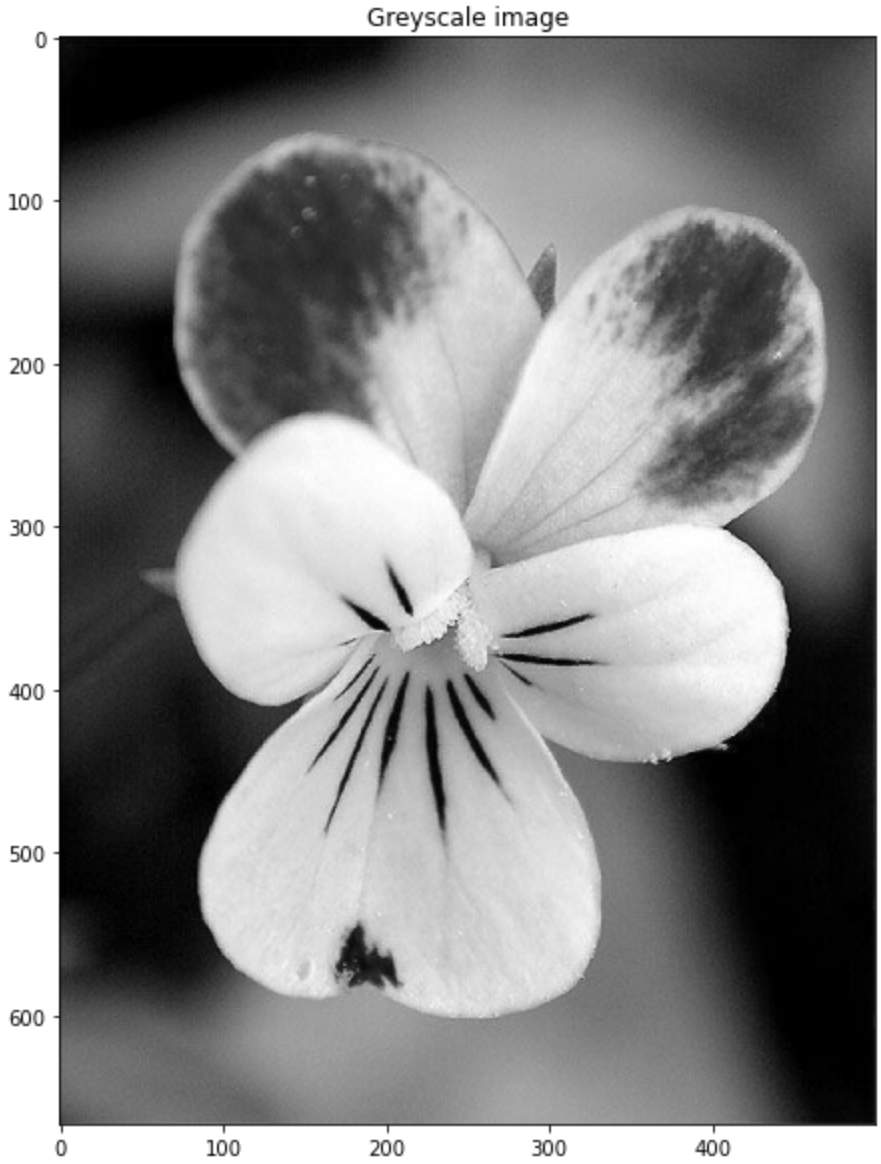
In [1]:
```
# Import useful packages
import os  # useful for the filesize
import cv2
import matplotlib.pyplot as plt

#########################################################
######## a)
# Import the image, which is located in the folder images/ (you can download it from CAN
A_path = os.path.join(os.getcwd(), 'images/flower.jpg')
A = cv2.imread(A_path)
# Convert the image from BGR (OpenCV standard) to RGB (standard)
A = cv2.cvtColor(A, cv2.COLOR_BGR2RGB)

# image properties
height = A.shape[0]
width = A.shape[1]
channels = A.shape[2]
filesize = os.path.getsize(A_path)

print('Image Dimension    : ', A.shape)
print('Image Height       : ', height)
print('Image Width        : ', width)
print('Number of Channels : ', channels)
```

```
Image Dimension    :  (667, 500, 3)
Image Height       :  667
Image Width        :  500
Number of Channels :  3
```

In [2]:
```
#########################################################
######## b)
# Extract 2D images (the various channels + greyscale)
R = A[:,:,0]
G = A[:,:,1]
B = A[:,:,2]

plt.figure(figsize=(20,20))
plt.subplot(221)
plt.imshow(A)
plt.title('Color')
plt.subplot(222)
plt.imshow(R, cmap='gray', vmin=0, vmax=255)
plt.title('Red channel')
plt.subplot(223)
plt.imshow(G, cmap='gray', vmin=0, vmax=255)
```

```
plt.title('Green channel')
plt.subplot(224)
plt.imshow(B, cmap='gray', vmin=0, vmax=255)
plt.title('Blue channel')
plt.show()
```

In [3]:
```
Gr = cv2.imread(A_path, cv2.IMREAD_GRAYSCALE)
plt.figure(figsize=(10,10))
plt.imshow(Gr, cmap='gray', vmin=0, vmax=255)
plt.title('Greyscale image')
plt.show()
```



Answer to question 2 b) (describe):

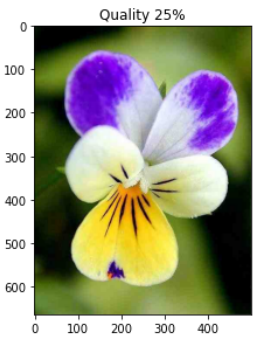
The violet part of the two upper petals contributes mostly to the blue and red component. The yellowish lower petal contributes to red and green, mostly. White parts are represented by all colors and the greenish background by green and also some red and a little blue. The gray level image eliminates the hue and saturation information and extracts the luminance by the following formula:
$0.2989 * R + 0.5870 * G + 0.1140 * B$

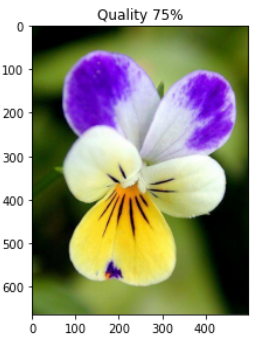
Problem 2 continues

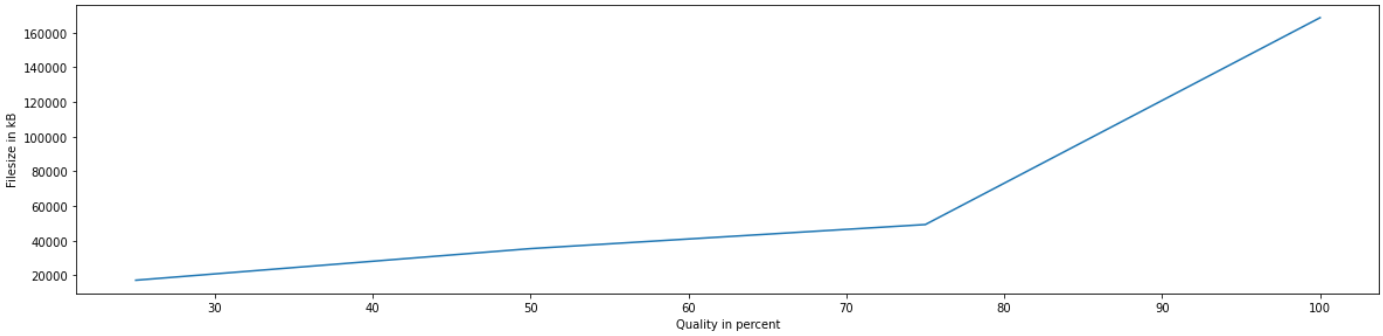
c) The image data can be written to new files with a chosen format. Use `cv2.imwrite` and JPG. We want to study different degrees of compression by using `[cv2.IMWRITE_JPEG_QUALITY, jpg_quality]` as option in the `cv2.imwrite` function, where `cv2.IMWRITE_JPEG_QUALITY` is the quality flag, and `jpg_quality` is the selected quality for saving the image. Let `jpg_qualities` be `[25,50,75,100]` and make a graph that show the filesize in kB as a function of `jpg_qualities` for this image. When a repeated procedure is done, like in this case, it is efficient to make a script or a function for the problem. Display the compressed images for `jpg_qualities=25` and `jpg_qualities=75` (use `plt.imshow`). Study these images and discuss the degradation of the images caused by the compression.

d) A simple way of finding objects in an image is by using thresholding. The OpenCV function `threshold`. performs simple thresholding and ouputs a logical image matrix. We want to find a logical mask identifying the flower (foreground and not the background) in our image. We can do that by combining the result from thresholding the green component and the red component, `Fmask = Gmask or Rmask`. `Gmask` is the output from thresholding the green component with a level of approximately (180/255) and `Rmask` is the result from thresholding the red component with level (150/255) approximately. Execute these operations and adjust the two levels for the best result. Display the final logical image `Fmask` and describe the result.

In [4]:
```
##########################################################
######## c)
# Image compression

A = cv2.imread(A_path)
jpg_qualities = [25,50,75,100]
size = []

for jpg_quality in jpg_qualities:
    filename = "./images/flower{}.jpg".format(str(jpg_quality))
    status = cv2.imwrite(filename, A, [cv2.IMWRITE_JPEG_QUALITY, jpg_quality])
    size.append(os.path.getsize(filename))

img25 = cv2.imread("./images/flower25.jpg")
img25 = cv2.cvtColor(img25, cv2.COLOR_BGR2RGB)
img75 = cv2.imread("./images/flower75.jpg")
img75 = cv2.cvtColor(img75, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(20,10))
plt.subplot(212)
plt.plot(jpg_qualities, size)
plt.xlabel("Quality in percent")
plt.ylabel("Filesize in kB")
plt.subplot(221)
plt.imshow(img25)
plt.title("Quality 25%")
plt.subplot(222)
plt.imshow(img75)
plt.title("Quality 75%")
plt.show()
```

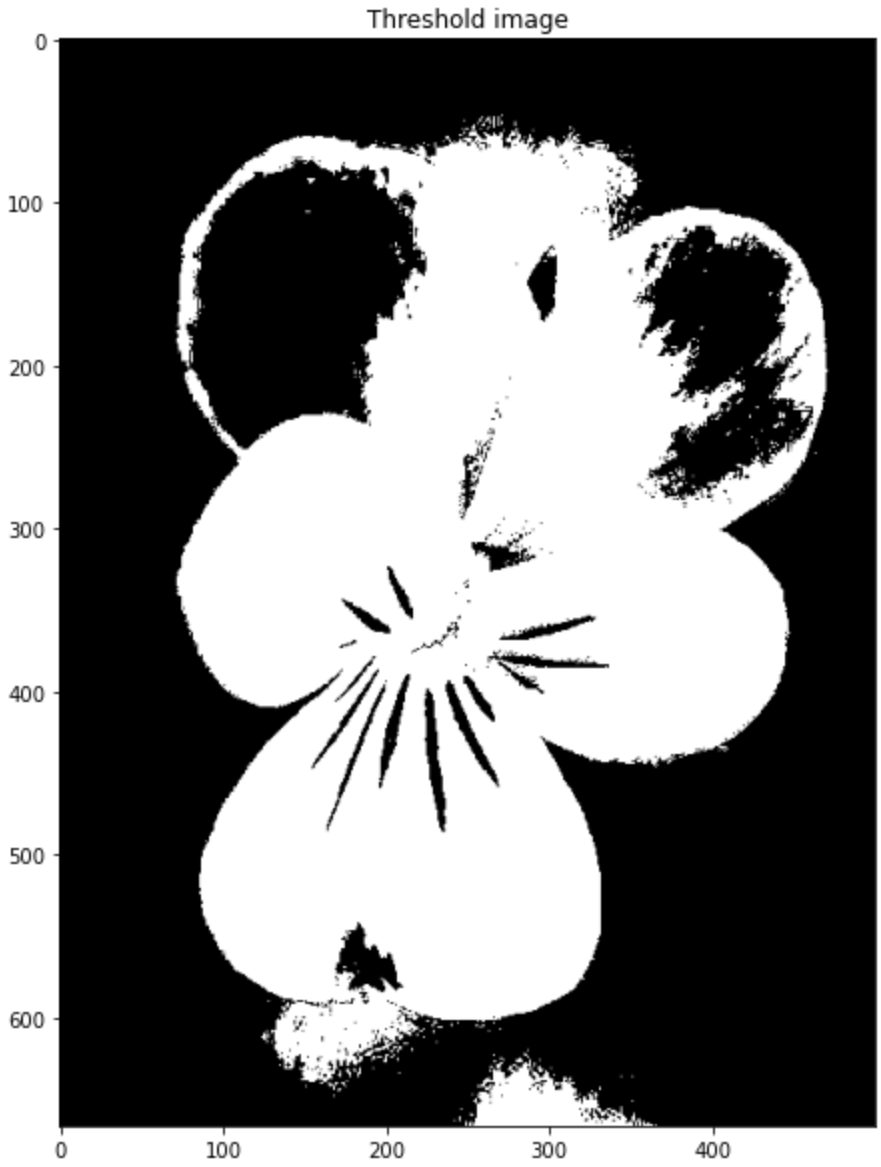






At 25% quality the blocking artifact is annoying, while at 75% blocking artifacts are not visible.

In [5]:
```
#########################################################
######## d)
# Thresholding: Black and White (binary) images
_, Gmask = cv2.threshold(G, 180, 255, cv2.THRESH_BINARY)
_, Rmask = cv2.threshold(R, 150, 255, cv2.THRESH_BINARY)
Fmask = Gmask | Rmask

plt.figure(figsize=(10,10))
plt.imshow(Fmask, cmap='gray', vmin=0, vmax=255)
plt.title('Threshold image')
plt.show()
```



Problem 3

Write a function that extracts a rectangular region from an input image, commonly known as cropping. Give the function the name **image2roi** (roi = region of interest). Let this function work as follows:

a) Input parameters should be an iamge and the coordinates for the roi (fname, coords). First check if the image is colour or grey level. If it is colour a message should be printed out and the function closed (return). If it is a grey level image continue to the next step, **b)**.

b) The size of the image is computed and the image displayed with indexes shown along the axis. Extract the sub image (region of interest) given the coordinates, display it and the function ended.

In [6]:
```
'''
Function that takes in input an image and the coordinates for the ROI

'''
def image2roi(img, coords):
    if len(img.shape)>2:
        print("The image is not in greyscale. Exit.")
        return

    roi = img[coords[0]:coords[1], coords[2]:coords[3]]

    print(roi)
    plt.figure(figsize=(20,20))
    plt.subplot(121)
    plt.imshow(img, cmap='gray', vmin=0, vmax=255)
    plt.title('Original image')
    plt.subplot(122)
    plt.imshow(roi, cmap='gray', vmin=0, vmax=255)
    plt.title('Region of interest')
    plt.show()


coords = [100,300,100,300]
fname = "./images/flower.jpg"
img = cv2.imread(fname, cv2.IMREAD_GRAYSCALE)
image2roi(img, coords)
```

```
[[138 127 111 ... 172 171 170]
 [135 123 103 ... 172 171 171]
 [124 113  93 ... 172 172 173]
 ...
 [223 224 224 ... 192 186 206]
 [224 224 224 ... 199 194 199]
 [225 225 224 ... 191 180 183]]
```

Problem 4

`NumPy` is a `Python` library that's vital for computer vision. It handles arrays and math efficiently. It's used to represent images, apply filters, and prepare data for computer vision tasks. To explore this, let an image be:

$$F(x,y) = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}, \tag{1}$$

To produce this image with `numpy`, use:

```
F = np.matrix('1 2 3 4;5 6 7 8;9 10 11 12;13 14 15 16')
```

a) Use `numpy` to retrieve the first element of the first column, the first column and the first row. Do the same for the last column and row.

b) Use the `numpy` function `reshape` to reconstruct the image matrix into any other shape. Refer to numpy.reshape for full documentation.

c) Use `numpy` to create a boolean array that contains the value `True` for values multiple of 5 and bigger than 6.

In [7]:
```
# Import useful packages
import numpy as np
from pprint import pprint

F = np.matrix('1 2 3 4;5 6 7 8;9 10 11 12;13 14 15 16')
print("F:")
pprint(F)

######## a)
f_ele = F[0,0]
f_row = F[0,:]
f_col = F[:,0]
print("f_ele: ")
pprint(f_ele)
print("f_row: ")
pprint(f_row)
print("f_col: ")
pprint(f_col)

l_row = F[-1,:]
l_col = F[:,-1]
print("l_row: ")
pprint(l_row)
print("l_col: ")
pprint(l_col)

######## b)
f_res = np.reshape(F, (2, 8))
print("f_res: ")
pprint(f_res)

######## c)
f_bool = (F > 6) & (F %5==0)
print("f_bool: ")
pprint(f_bool)
```

```
F:
```

```
matrix([[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12],
        [13, 14, 15, 16]])
f_ele:
1
f_row:
matrix([[1, 2, 3, 4]])
f_col:
matrix([[ 1],
        [ 5],
        [ 9],
        [13]])
l_row:
matrix([[13, 14, 15, 16]])
l_col:
matrix([[ 4],
        [ 8],
        [12],
        [16]])
f_res:
matrix([[ 1,  2,  3,  4,  5,  6,  7,  8],
        [ 9, 10, 11, 12, 13, 14, 15, 16]])
f_bool:
matrix([[False, False, False, False],
        [False, False, False, False],
        [False,  True, False, False],
        [False, False,  True, False]])
```

Contact

Course teacher

Professor Kjersti Engan, room E-431, E-mail: kjersti.engan@uis.no

Teaching assistants

Saul Fuster Navarro, room E-401 E-mail: saul.fusternavarro@uis.no

Jorge Garcia Torres Fernandez, room E-401 E-mail: jorge.garcia-torres@uis.no

References

[1] S. Birchfeld, Image Processing and Analysis. Cengage Learning, 2016.

[2] I. Austvoll, "Machine/robot vision part I," University of Stavanger, 2018. Compendium, CANVAS.