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) Make a list of at least 5 different applications of robot (machine) vision.

1. Food Automation

3D machine vision systems can be deployed in consumer industries like food automation, which involves food processing, segmentation, and packing using an automated machine. Up to 94% of food packaging operations are already using robotics.

2. Depalletization/Depalletization

The process of depalletizing is as arduous as monotonous. Using autonomous machines increases productivity by completing the tasks with high speed. This tasks have to involve different shapes, weights and materials, among the main challenges for these solutions.

2. Bin Picking

This is an automated process of detecting, classifying, picking, and placing a target object to achieve the desired result. It is one of the most common automation applications in smart factories and warehouses. 3D machine vision cameras play an essential role in increasing the accuracy of the bin-picking operation, especially for tiny objects and randomly distributed items.

3. Military Drones

First world countries such as United States have invested significant capital to develop and give AI capabilities to war drones. Such objects are surpasing pilots vision capabilities to detect, filter and choose right targets by diminishing casualties. With the help of deep and expensive vision sensors drones can now detect potential targets from several kilometers away and save resources.

4. Computer vision for Forest fire detection

Artificial intelligence and agrotechnological companies are joining forces to create systems capable of creating wildfire detection systems with the help of deep learning. This way the algorithms can coordinate robots via drone surveilance and alarme systems that help send signals before a wildfire becomes a major issue, with the possibility of diminishing significal damage for the region.

5. Precision Agriculture techniques are creating greater efficiencies and profitability in agriculture:

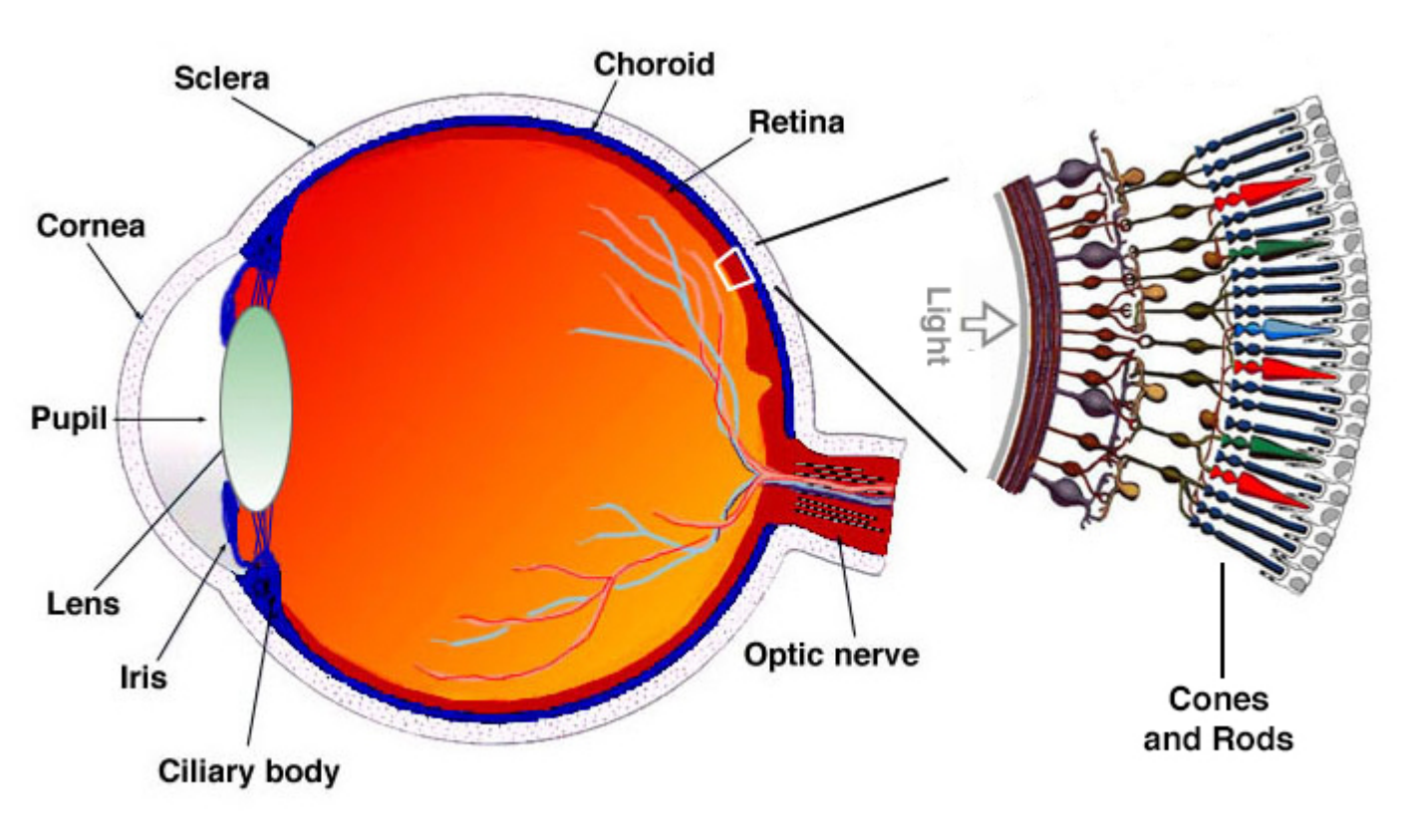
1. **Field Robots** automate processes such as harvesting, planting, weeding and more. Machine vision systems are used to identify and categorize crops, providing essential visual input to automate such tasks.
2. **Phenotyping** is crucial for ensuring only the best crop breeds are grown. Machine vision helps identify the best breeds by monitoring growth and identifying phenotypic traits that signify a robust genotype as the rapidly expanding human population needs greater volumes of food.
3. **Grading and Sorting** helps to separate good crops from bad crops and determine which will be stable for longer shipments and which will go bad first and should be shipped to local markets. It combines deep learning techniques with robot arms.
4. **Livestock Identification** monitors Livestock's growth over the course of their lifetime to provide important information about their progress towards harvesting.
5. **Machine Guidance** helps autonomous tractors and other vehicles in the agriculture industry to guide them in variable outdoor conditions for full autonomy.

b) What is the resolution of the tightly spaced cones in the fovea, and how is this compared to the spacing between pixels in a typical digital camera?

- Cones are sensitive to the visible light. They are tighlty packed as a regular sampling array, spaced approximately at

$$2.5\mu m$$

  . Such distribution is aproximately the same spacing as in the pixels on a typical camera sensor.
  - Cones are specialized in three different kinds, being able to Filter and detect Red, Green and blue Respectively.
- Rods are sensitive to low levels of light
  - Rods exist in just a specific kind, therefore the inability to distinguis from colors in the darkness.



c) How much storage is needed for a one hour digital video (colour) with no compression if we assume a frame rate of 50 frames per second (fps) and that each image frame is 3840×2160 pixels.

1. Total number of seconds

$$t = 1hour * 60minutes * 60seconds = 3600seconds$$

2. Total number of frames

$$f = t * 50 = 3600 * 50 = 180000$$

3. Number of channels [R, G, B]

$$c = 3$$

4. Image size = width *height* Number of channels

$$image_{size} = 3840 * 2160 * 3 = 24883200 \ bytes$$

5. Total size

$$total_{size} = image_{size} * f = 24883200 * 180000 = 4478976000000 \ bytes \approx 4.479 \ Terabytes$$

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.

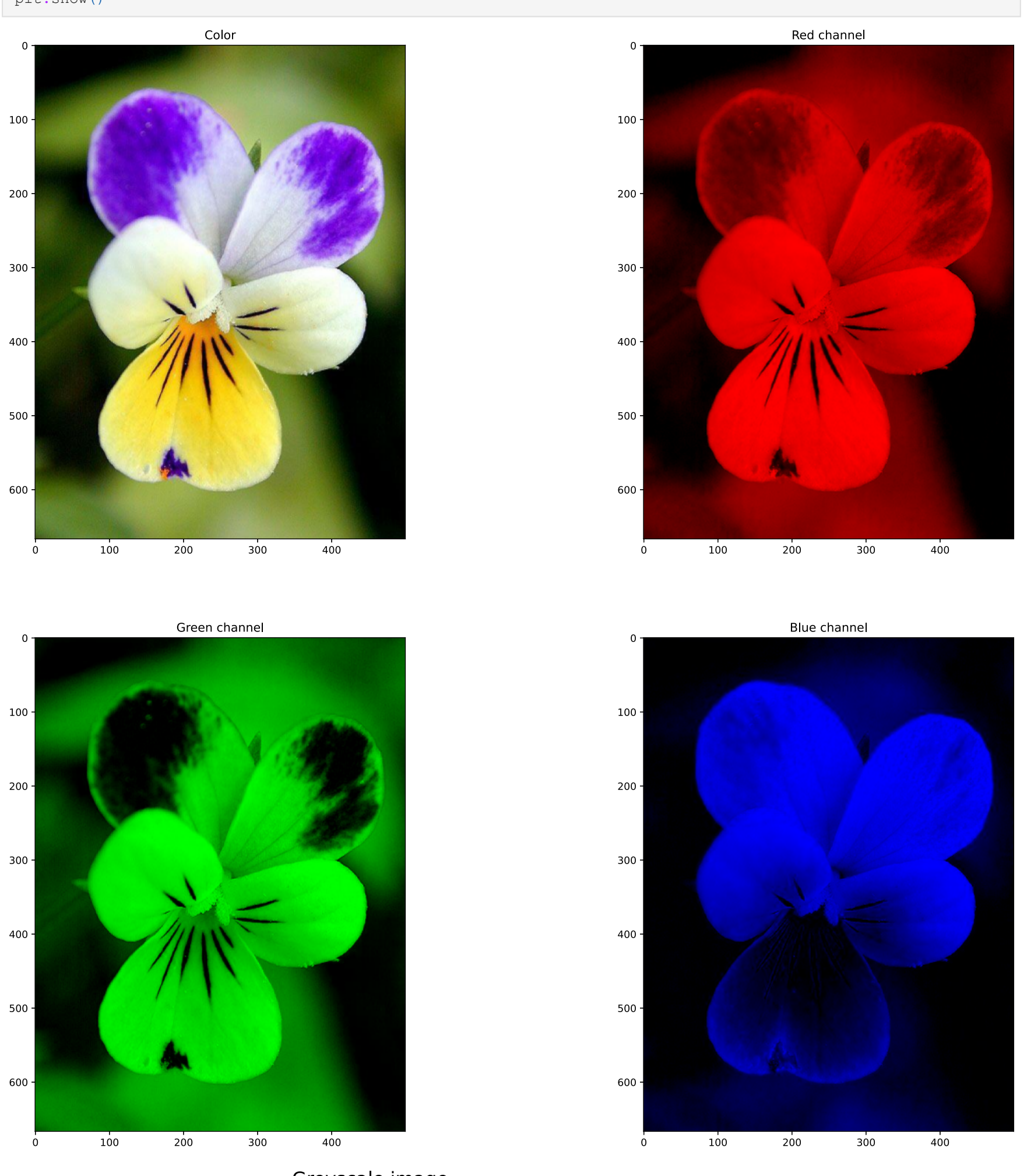
Describe how the different colour components contributes to different parts of the image (the petals and the background). Show the gray level image in a separate figure. Describe this image in relation to the colour components.

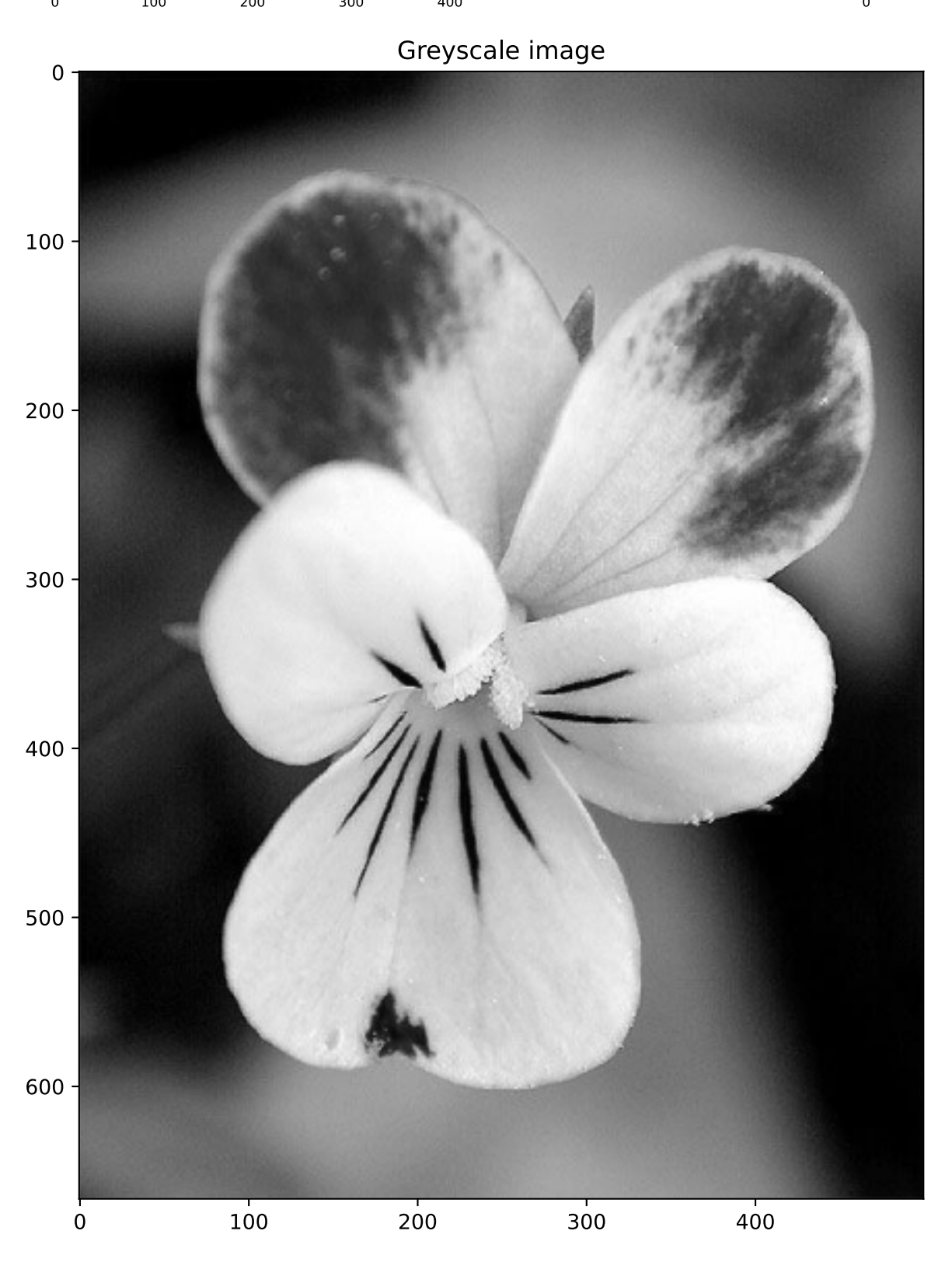
The filesize can be checked in **bytes** using the following commands: $pythonimp \text{ or } tosfi \leq size = os.path. \geq tsize(my_path)$

In [34]:
```python
# Import useful packages
import os # useful for the filesize
import cv2
import matplotlib.pyplot as plt

# Complete the parts with "..."

########################################################
######## a)
# Import the image, which is located in the folder images/ (you can download it from CANVAS)
A_path = '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)
print('File size          : ', filesize)
## The results should be:
# Image Dimension    :  (667, 500, 3)
# Image Height       :  667
# Image Width        :  500
# Number of Channels :  3
```

```
Image Dimension    :  (667, 500, 3)
Image Height       :  667
Image Width        :  500
Number of Channels :  3
File size          :  49232
```

In [35]:
```python
########################################################
######## b).
# Extract 2D images (the various channels + grayscale)
R = A * (1,0,0)
G = A * (0,1,0)
B = A * (0,0,1)

plt.figure(figsize=(20,20))
plt.subplot(221)
plt.imshow(A)
plt.title('Color')
plt.subplot(222)
plt.imshow(R)
plt.title('Red channel')
plt.subplot(223)
plt.imshow(G)
plt.title('Green channel')
plt.subplot(224)
plt.imshow(B)
plt.title('Blue channel')
plt.show()

# Greyscale image
Gr = cv2.imread(A_path,flags=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):

1. Describe how the different colour components contributes to different parts of the image (the petals and the background).

The normal image shows a colorful flower with 4 petals and a dominance of white, yellow, blach stripes and purple stains. In the three channels is possible to see that black is always represented as black, but in those whose dominance is a specific color i.e. Yellow, where the dominant colors are green and red, the blue channel shows black shadows; whereas in the purple stains the blue channel has a greater dominance.

2. Describe the gray image in relation to the colour components.

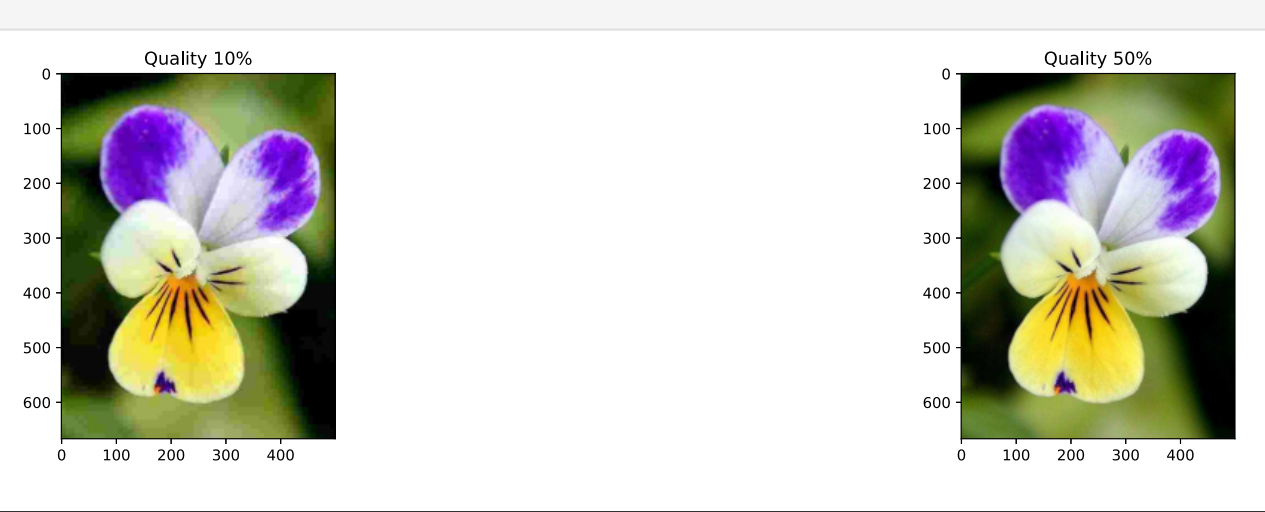
The gray image could be described as the average of the three colors on each pixel, sown in a single channel.

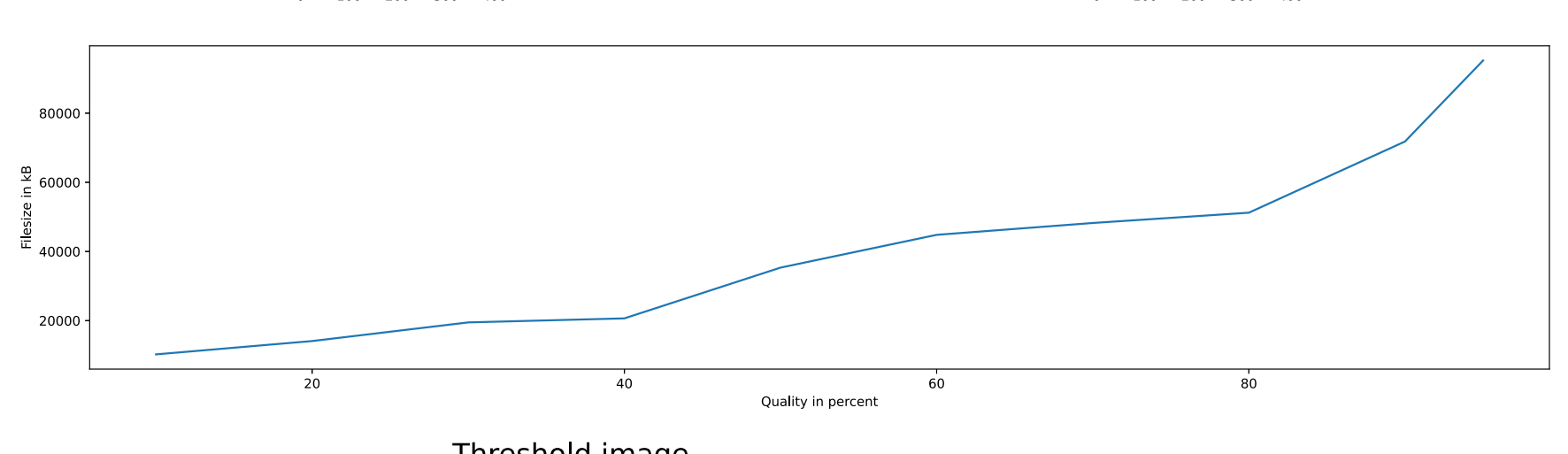
Problem 2 continues

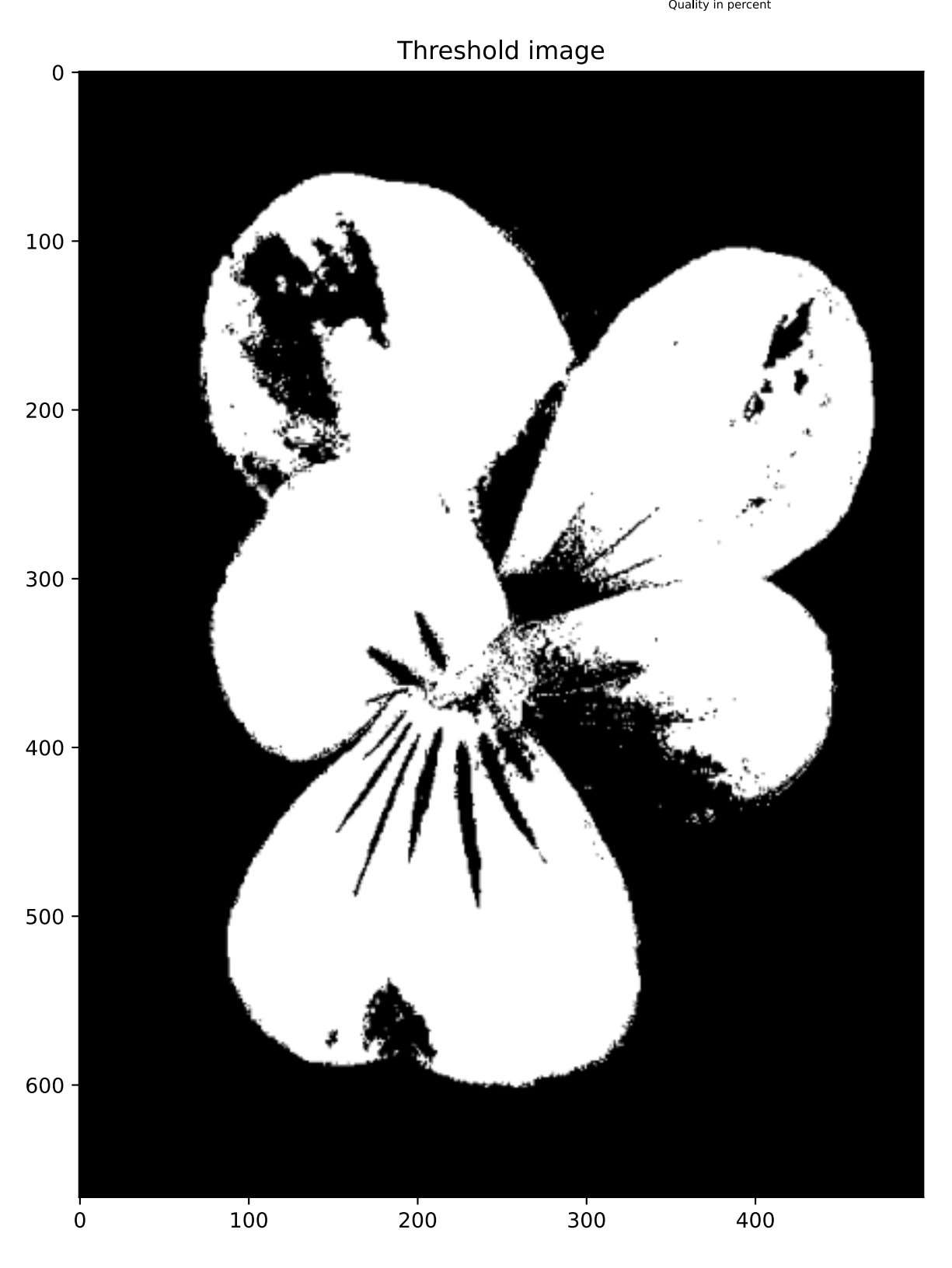
c) The image data can be written to new files with a chosen format. Use `cv2.imwrite` and JPG. We want to study different degrees of compression by using `[cv2.IMWRITE_JPEG_QUALITY, jpg_quality]` as option in the `cv2.imwrite` function, where `cv2.IMWRITE_JPEG_QUALITY` is the quality flag, and `jpg_quality` is the selected quality for saving the image. Let `jpg_qualities` be `[10,20,30,40,50,60,70,80,90,95]` 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=10` and `jpg_qualities=50` (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 red component and the blue component, `Fmask = Bmask or Rmask`. `Bmask` is the output from thresholding the blue component with a level of approximately (160/255) and `Rmask` is the result from thresholding the red component with level (200/255) approximately. Execute these operations and adjust the two levels for the best result. Display the final logical image `Fmask` and describe the result.

In [150…
```python
########################################################
######## c)
# Image compression
jpg_qualities = [10,20,30,40,50,60,70,80,90,95]
size = []

# write the images with a specific quality
for jpg_quality in jpg_qualities:
    filename = "./images/flower{}.jpg".format(str(jpg_quality))
    cv2.imwrite(filename,cv2.cvtColor(A,cv2.COLOR_RGB2BGR),[cv2.IMWRITE_JPEG_QUALITY, jpg_quality])
    size.append( os.path.getsize(filename))

# Read the two images
img10 = cv2.cvtColor(cv2.imread("./images/flower10.jpg")  ,cv2.COLOR_BGR2RGB)
img50 = cv2.cvtColor(cv2.imread("./images/flower50.jpg")  ,cv2.COLOR_BGR2RGB)

plt.figure(figsize=(20,10))
plt.subplot(212)
plt.plot(jpg_qualities,size) # Show the plot for the filesize in kB of the images generated
plt.xlabel("Quality in percent")
plt.ylabel("Filesize in kB")
plt.subplot(221)
plt.imshow(img10)
plt.title("Quality 10%")
plt.subplot(222)
plt.imshow(img50)
plt.title("Quality 50%")
plt.show()

########################################################
######## d)
# Thresholding: Black and White (binary) images

# _, means that we are skipping the first output.
# look in OpenCV documentation to find out what the first output of threshold is
t_b, Bmask = cv2.threshold(A[:,:,0],200,255,cv2.THRESH_BINARY)
t_r, Rmask = cv2.threshold(A[:,:,2],200,255,cv2.THRESH_BINARY)
Fmask = Bmask | Rmask

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







Problem 2 c)

After observing the generated images with loss of quality, for the flower 50% is still a pretty good result. For the 10 persent squares and sharp edges star to be noticeable in the background, hohwever the flower still remains nitid, which means is still useful for data manipulation and transformation.

Problem 2 d)

Final results for a good thresold were 200/255 on both R and B channels. This combination specially allowed the noise on the background to be properly filtered with binary algorithm. There is however a loss in one of the petals as the color of the background is brighter than in most other parts.

Problem 3

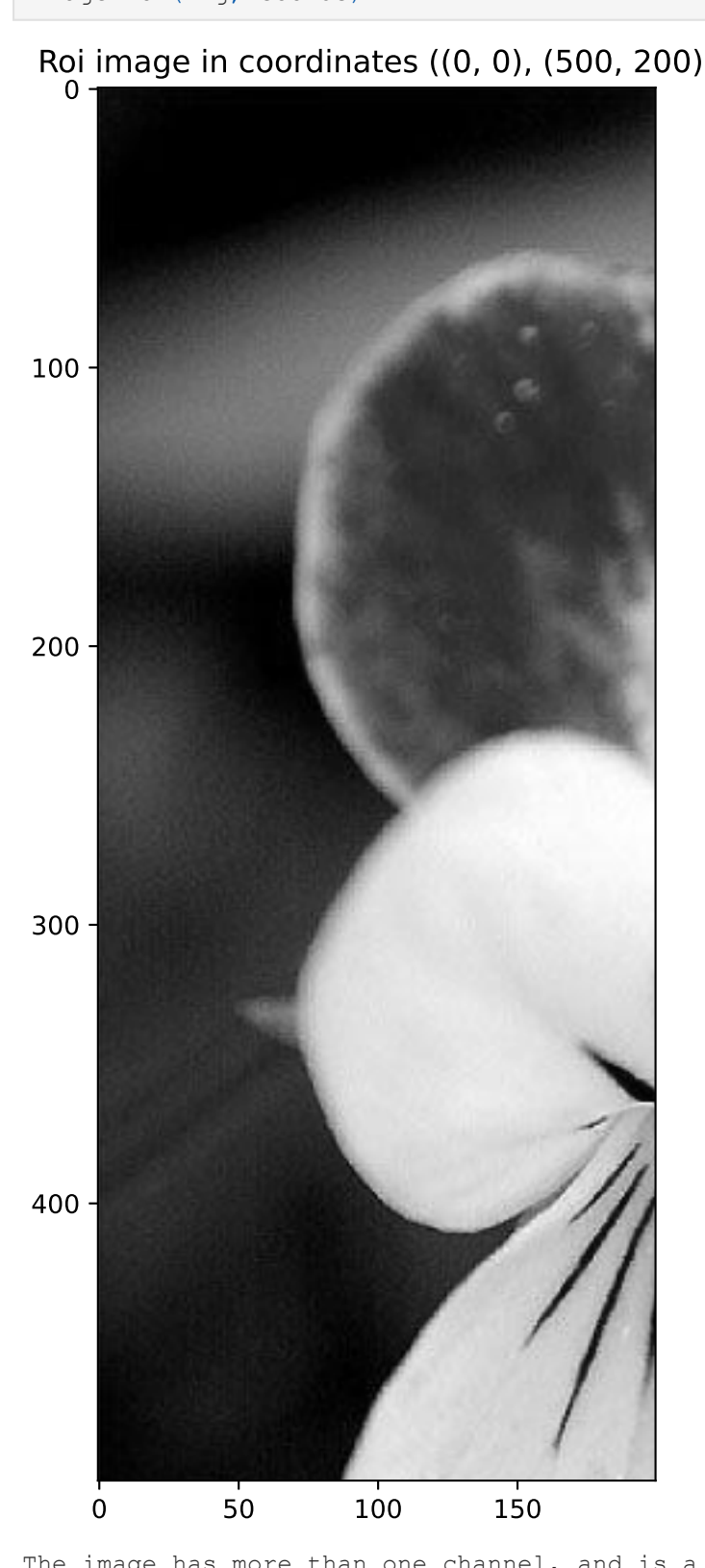
Write a function that extracts a rectangular region from an input image, commonly known as cropping. Give the function the name **image2roi** (roi = region of interest). Let this function work as follows:

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 [166…
```python
'''
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 has more than one channel, and is a color image. Process will skip')
        return

    # Plot the greyscale image and the ROI based on the coords values
    img_roi = img[coords[0][0]:coords[1][0],coords[0][1]:coords[1][1]]
    plt.figure(figsize=(10,10))
    plt.imshow(img_roi, cmap='gray', vmin=0, vmax=255)
    plt.title(f'Roi image in coordinates {coords}')
    plt.show()
```

In [175…
```python
##  To test your function, complete the following lines:

coords = ((0,0),(500,200))
img = cv2.imread(A_path,flags=cv2.IMREAD_GRAYSCALE)  # use the flower image or something else you want.

image2roi(img, coords)
img = cv2.imread(A_path)
# Test a color image
image2roi(img, coords)
```




```
The image has more than one channel, and is a color image. Process will skip
```

Problem 4

The representation of a digital image as a column vector is very useful in some occasions. We therefore include this here, from a practical view, using `python`. To explore this we start with a tiny test image. Let the image be

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

To produce this image with `numpy`, use:

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

a) Use the `numpy` function `f = F.flatten()` What is the resulting f?

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

c) What happens using the following operation `fr1 = F[:]`?

d) Array and matrix operations are very efficient with `numpy`. Check how the following operation work:

```python
fr2 = F[2,:]
fr3 = F[:,3]
```

In [4]:
```python
# 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 = F.flatten()
print("f: ")
pprint(f)

######## b)
F1 = np.reshape(f,(4,4))
print("F1: ")
pprint(F1)

######## c)
fr1 = F[:]
print("fr1: ")
pprint(fr1)

######## d)
fr2 = F[2,:]
fr3 = F[:,3]
print("fr2: ")
pprint(fr2)
print("fr3: ")
pprint(fr3)
```

```
F:
matrix([[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12],
        [13, 14, 15, 16]])
f:
matrix([[ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16]])
F1:
matrix([[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12],
        [13, 14, 15, 16]])
fr1:
matrix([[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12],
        [13, 14, 15, 16]])
fr2:
matrix([[ 9, 10, 11, 12]])
fr3:
matrix([[ 4],
        [ 8],
        [12],
        [16]])
```

Problem 4, answers:

a)

$$f = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \end{bmatrix} \quad (2)$$

b)

$$F_1 = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \quad (3)$$

c)

$$F_{r1} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \quad (4)$$

d)

$$F_{r2} = \begin{bmatrix} 9 & 10 & 11 & 12 \end{bmatrix} \quad (5)$$

$$F_{r3} = \begin{bmatrix} 4 \\ 8 \\ 12 \\ 16 \end{bmatrix} \quad (6)$$

Delivery (dead line) on CANVAS: 05-09-2021 at 23:59

Contact

Course teacher

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

Teaching assistant

Tomasetti Luca, room E-401 E-mail: luca.tomasetti@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.