# Introduction to Image Processing

## International Days 2022

Romuald Jolivot

# Outlines

- **General introduction**
  - Bangkok University - BU-CROCCS
  - What is image processing?
  - Applications
- Python + OpenCV
  - Requirements
- Basic image processing implementations
- Instagram-like filter implementations
- Conclusions

# Before we start

- Presentation, Python codes and test images are available to download on the following link:
  - https://shorturl.at/ltLS1

# General Introduction
# Bangkok University - BU-CROCCS

- BU: Private university established in 1962
- BU-CROCCS: Center of Research in Optoelectronics, Communications and Computational Systems established in 2012

# BU-CROCCS

- Affiliated with School of Engineering
- Brings together expertise of faculty members with various research background
- Offers internship opportunities (20-30 international interns/year)

Examples of student projects

Link for internship applications

# BU-CROCCS
# Full-time researchers

**Dr. Karel Sterckx**
- **Belgium**
- **PhD Swansea University (UK)**
- **Optical Wireless Communication / Software Defined Communication Systems**

**Dr. Waleed Mohammed**
- **Egypt**
- **PhD University of Central Florida (USA)**
- **Optics / Photonics / Sensors / Nanotechnology / Fiber Optics**

**Dr. Poompat Saengudomlert**
- **Thailand**
- **PhD MIT (USA)**
- **Digital Communications / Multi-Carrier Modulation / Optical Communications / Optimization**

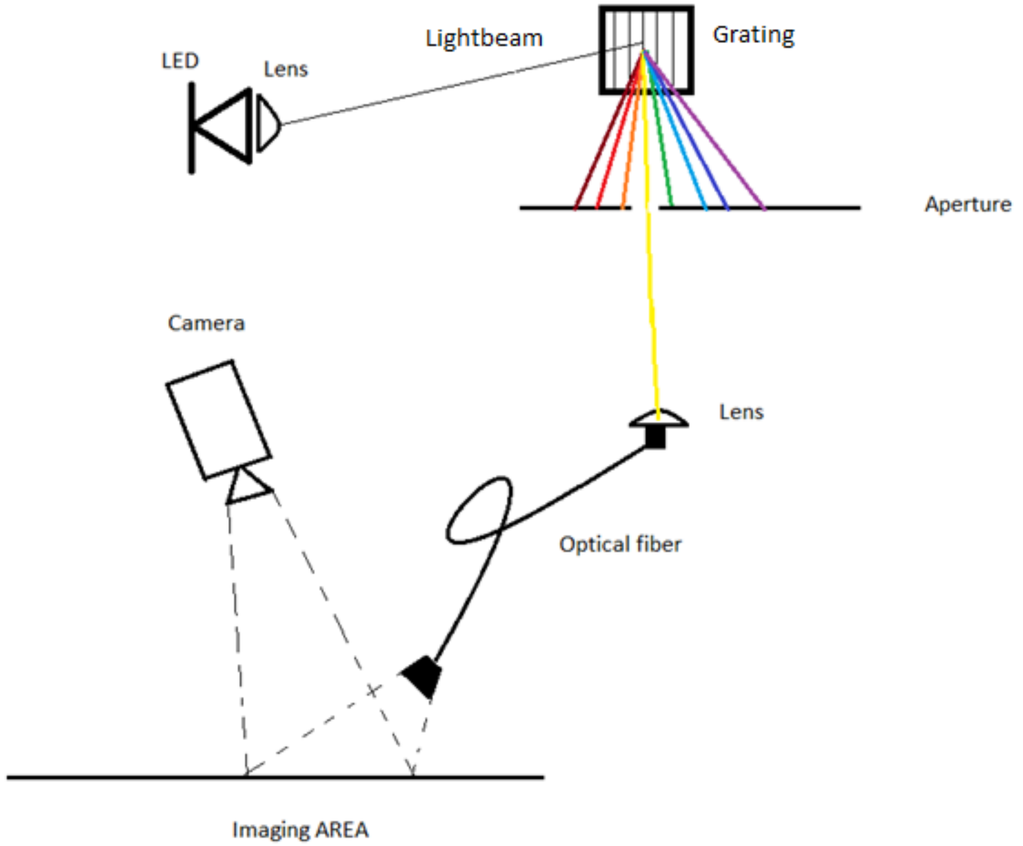
**Dr. Romuald Jolivot**
- **France**
- **PhD University of Burgundy (France)**
- **Image Processing**

# BU-CROCCS
# Research projects

- Image processing based:
  - Skin analysis using multispectral acquisition system
  - Innovative Museum Acquisition and Display Systems
  - Digital Make-up system
  - Low-cost phenotypic system for plant growth analysis
- IOT sensor platform
- Optical Wireless Communication (OWC) - Museum and Hospital applications
- Reflection spectrometry

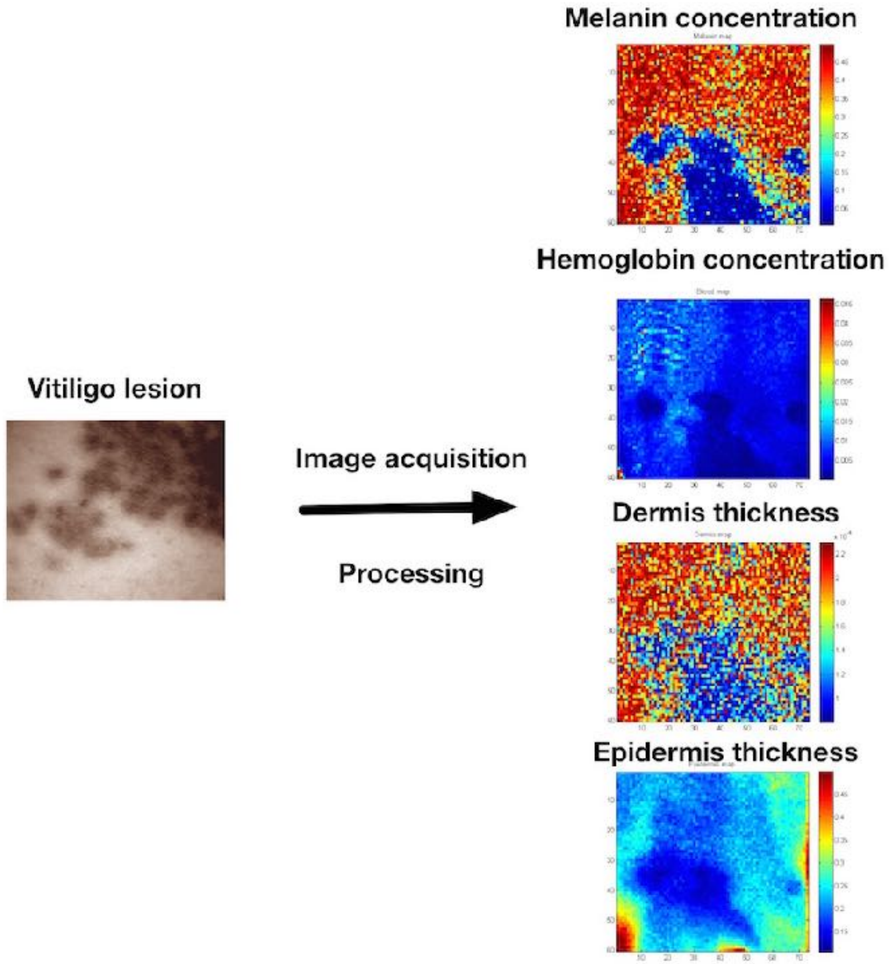
# BU-CROCCS
# Research projects - image processing
## Skin analysis using multispectral acquisition system

- Aim: Collect objective and reproducible quantification of skin optical properties
- System composition:
  - LED light (visible + near infrared)
  - Motor
  - Grating
  - Optical fiber
  - Camera
  - Acquisition time < 5s



# BU-CROCCS
# Research projects - image processing
## Skin analysis using multispectral acquisition system

- Multispectral data processing:
  - Skin component extraction using AI
    - Melanin concentration
    - Hemoglobin concentration
    - Dermis thickness
    - Epidermis thickness
    - etc.



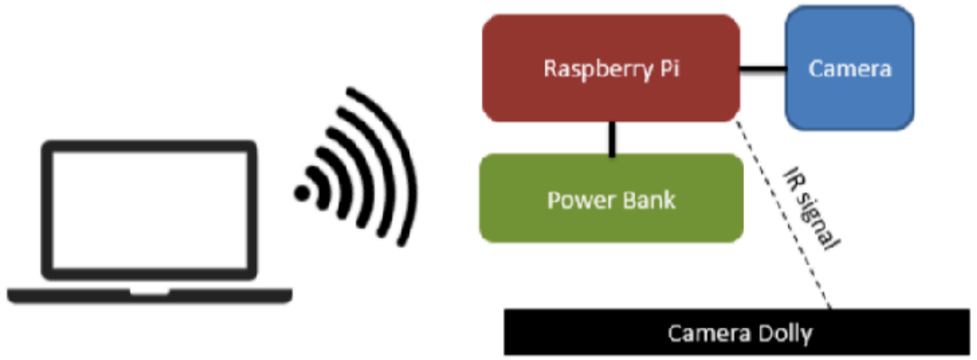
# BU-CROCCS
## Research projects - image processing
### Innovative Museum Acquisition and Display system

- Aims:
  - Automatic digitization of ceramic collections
  - Enhanced display and virtual interaction
- Collaborator:
  - South Asian Ceramics Museum

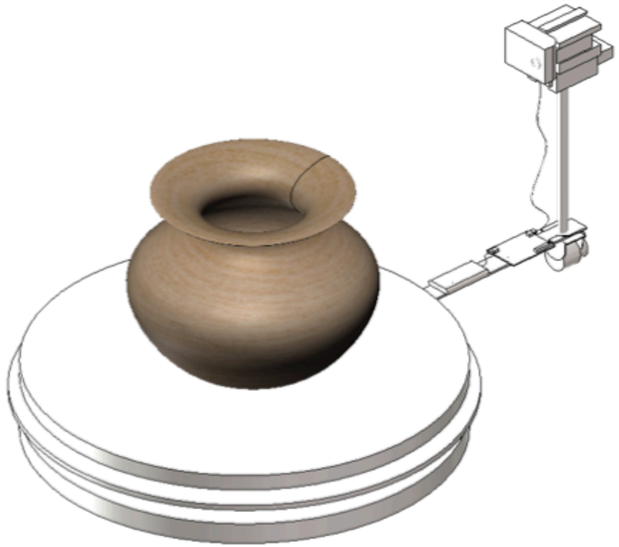
SOUTHEAST ASIAN CERAMICS MUSEUM
BANGKOK UNIVERSITY

# BU-CROCCS
# Research projects - image processing
## Innovative Museum Acquisition and Display system

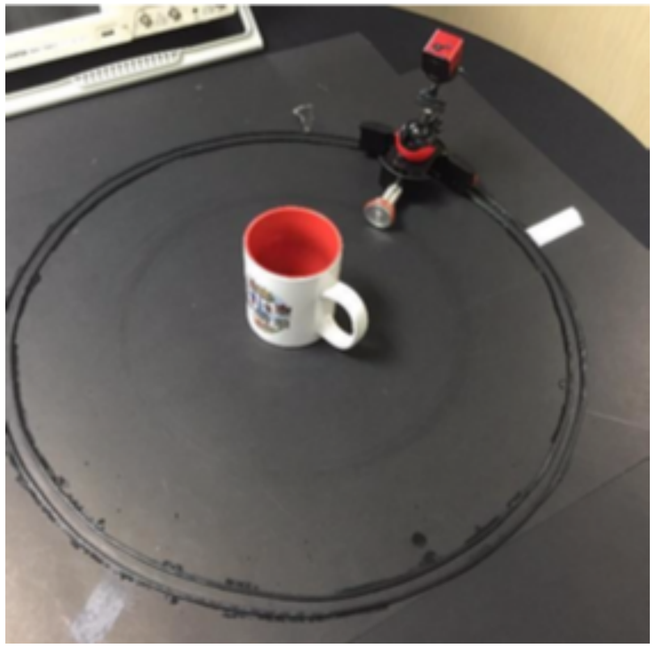
- Acquisition set-up:
  - Based on Raspberry Pi
  - Pi Camera
  - Camera Dolly




Working principle



Schematic set-up

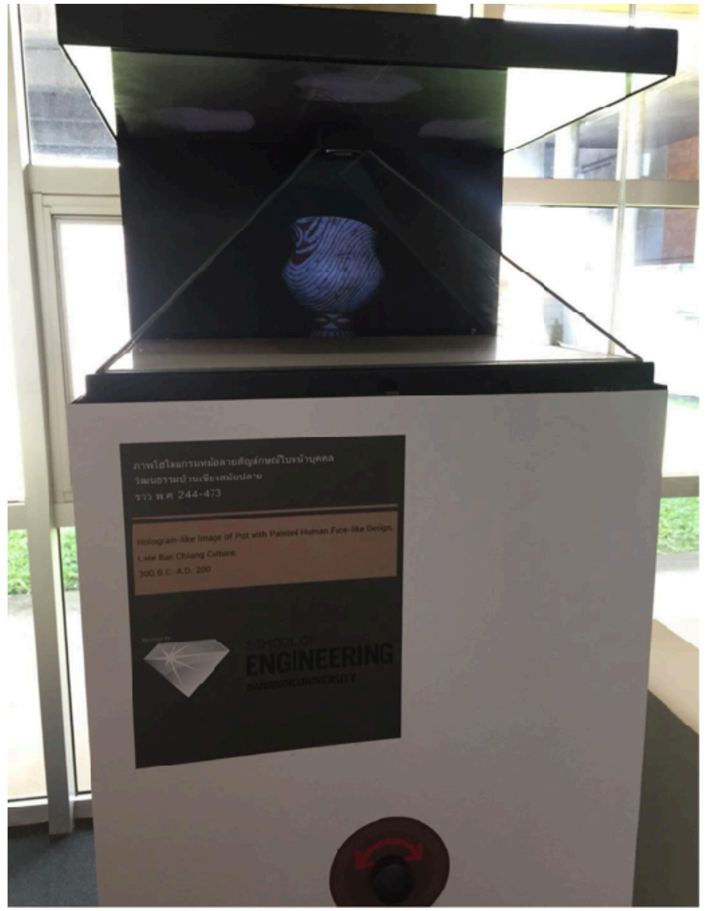


Current set-up

# BU-CROCCS
# Research projects - image processing
## Innovative Museum Acquisition and Display system

- Hologram-like Display Unit:
  - 3 views
  - 4 views

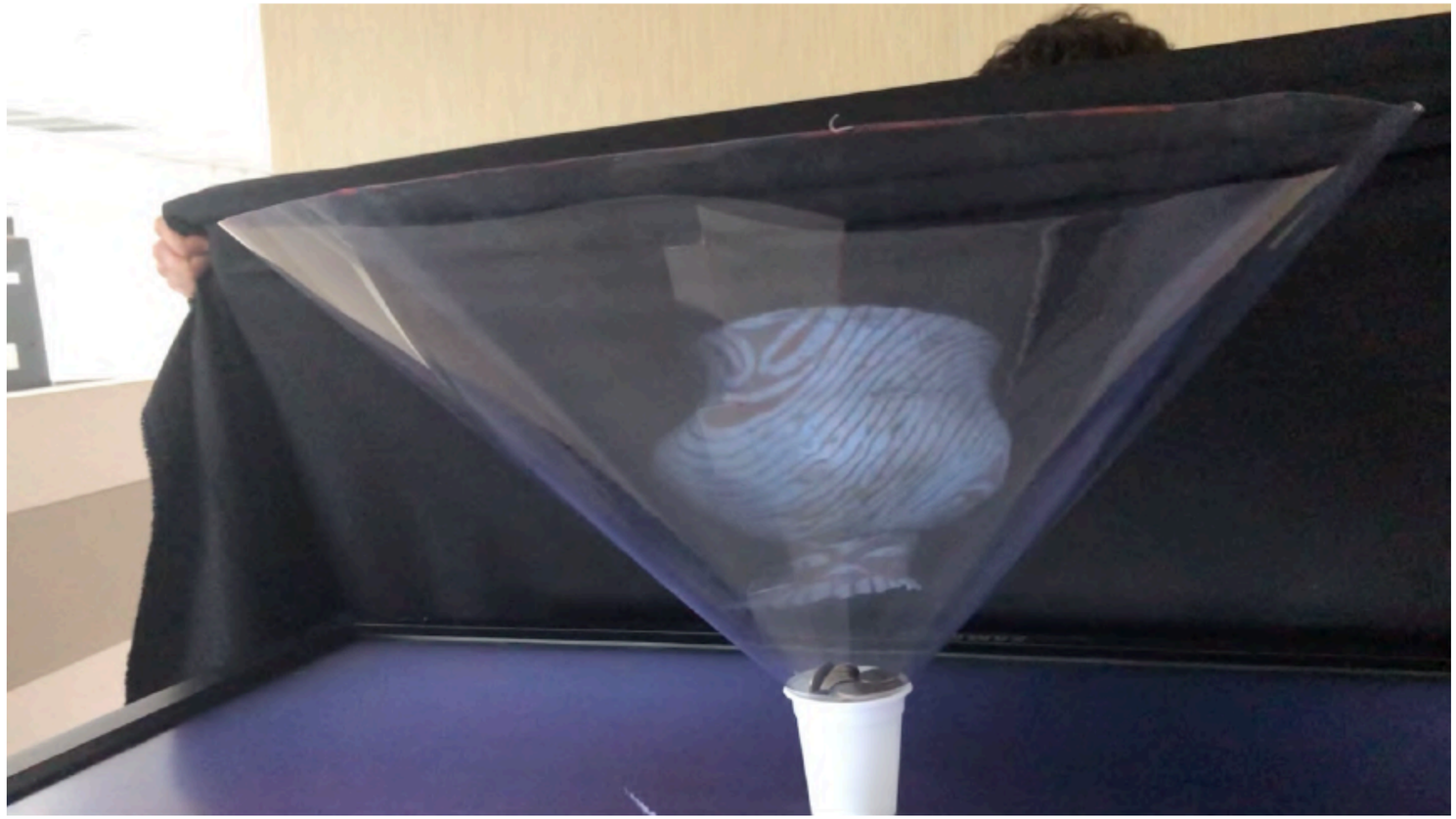
4 views display




3 views display

# BU-CROCCS
# Research projects - image processing
## Innovative Museum Acquisition and Display system

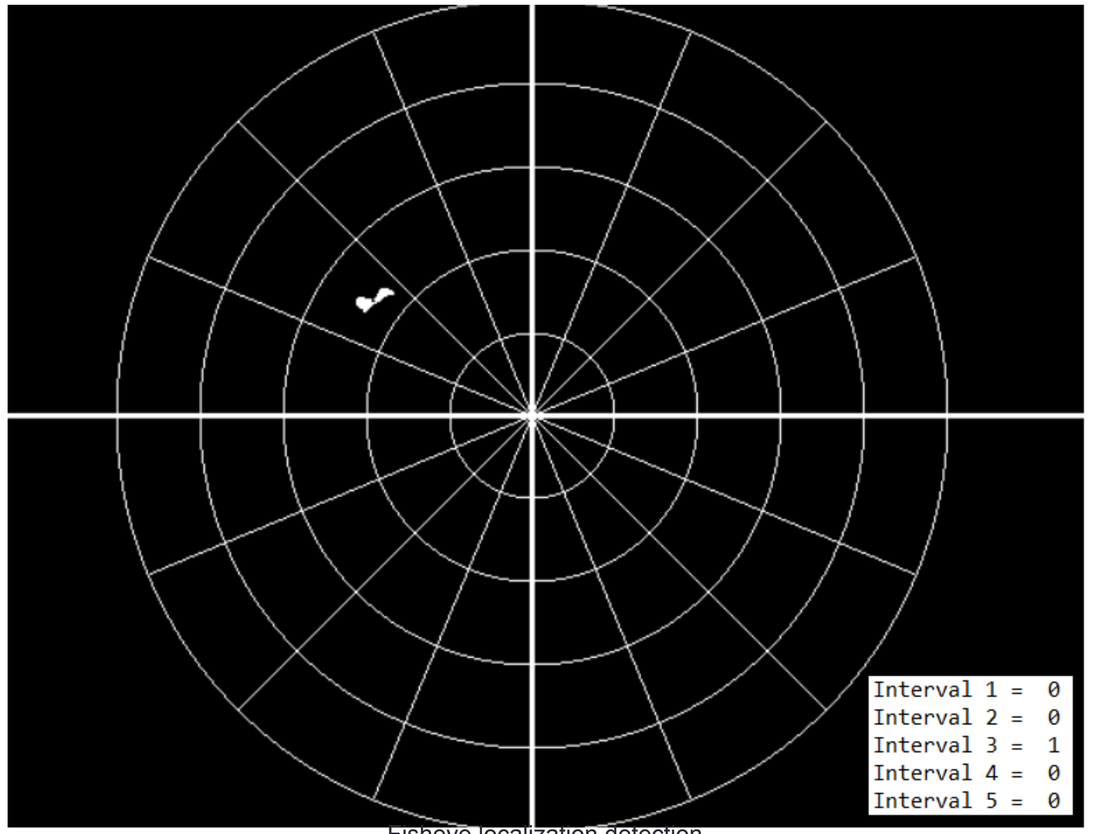
- Hologram-like Display Unit:
  - Cone



Cone display testing setup

# BU-CROCCS
# Research projects - image processing
## Innovative Museum Acquisition and Display system

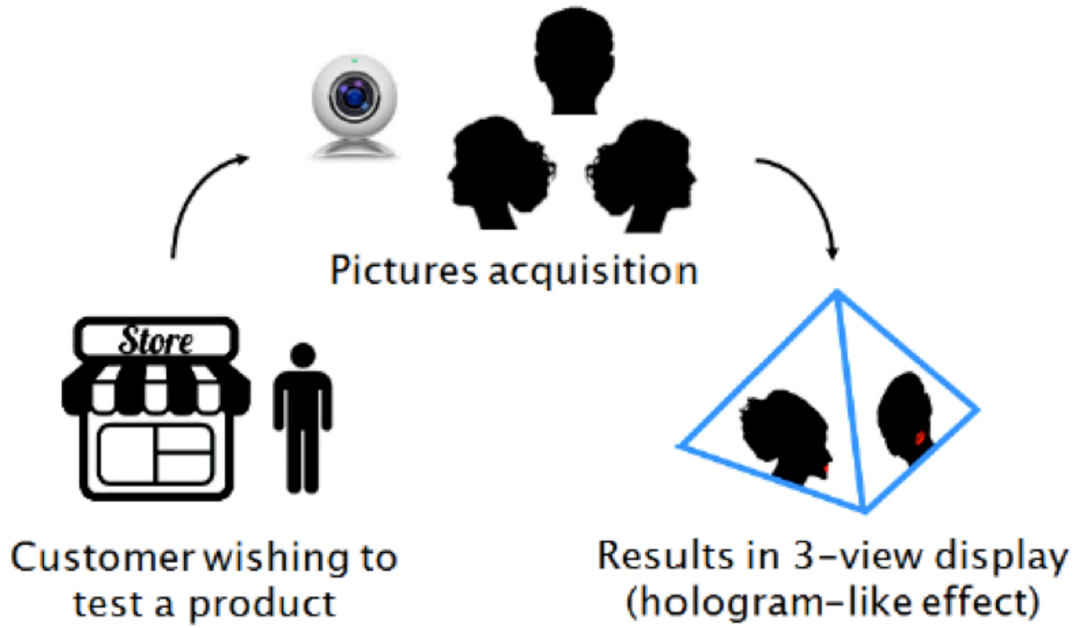
- Interactive Unit:
  - Kinect interaction - control display by hand
  - Fisheye localisation - detect user position



Fisheye localization detection

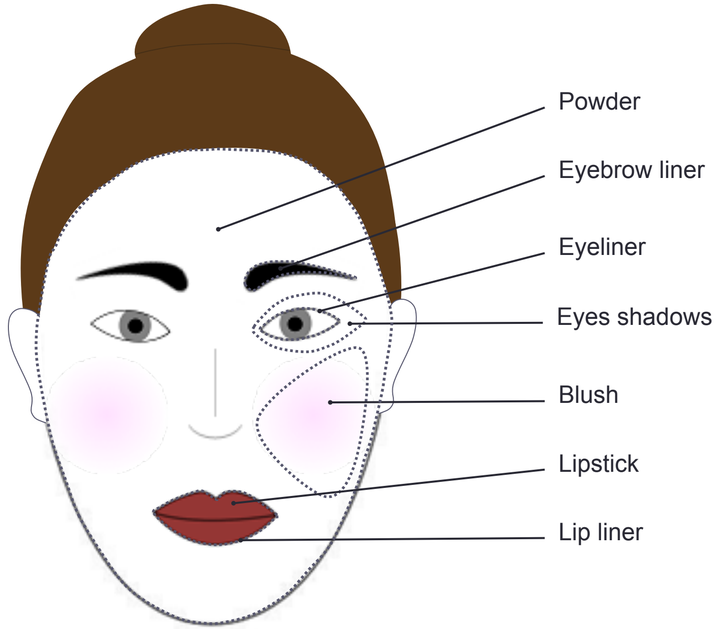
# BU-CROCCS
# Research projects - image processing
## Digital Make-up system

- Aim: Automatic digital make-up application with 3D-like display

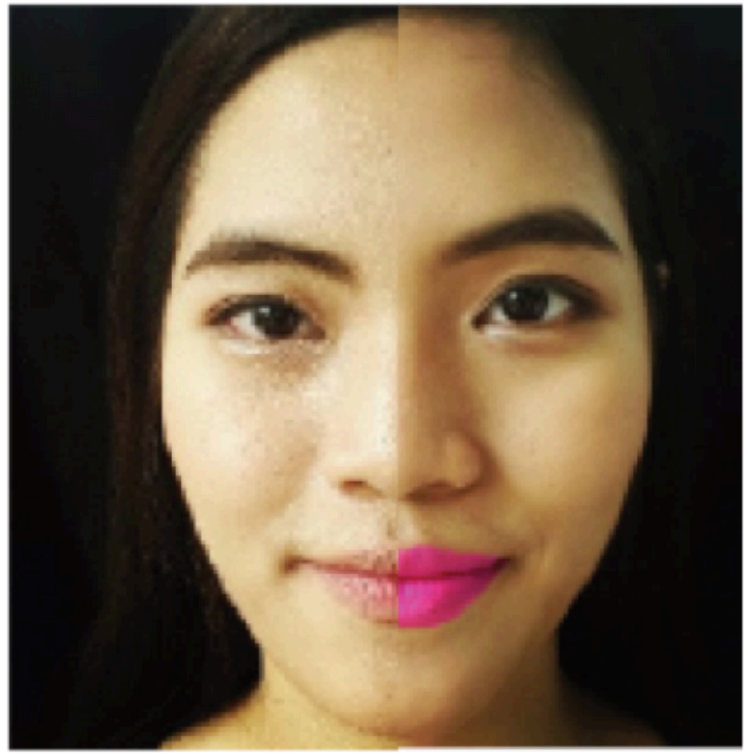


# BU-CROCCS
# Research projects - image processing
## Digital Make-up system

- Automatic make-up area detection




Multiple masks for different make-up applications



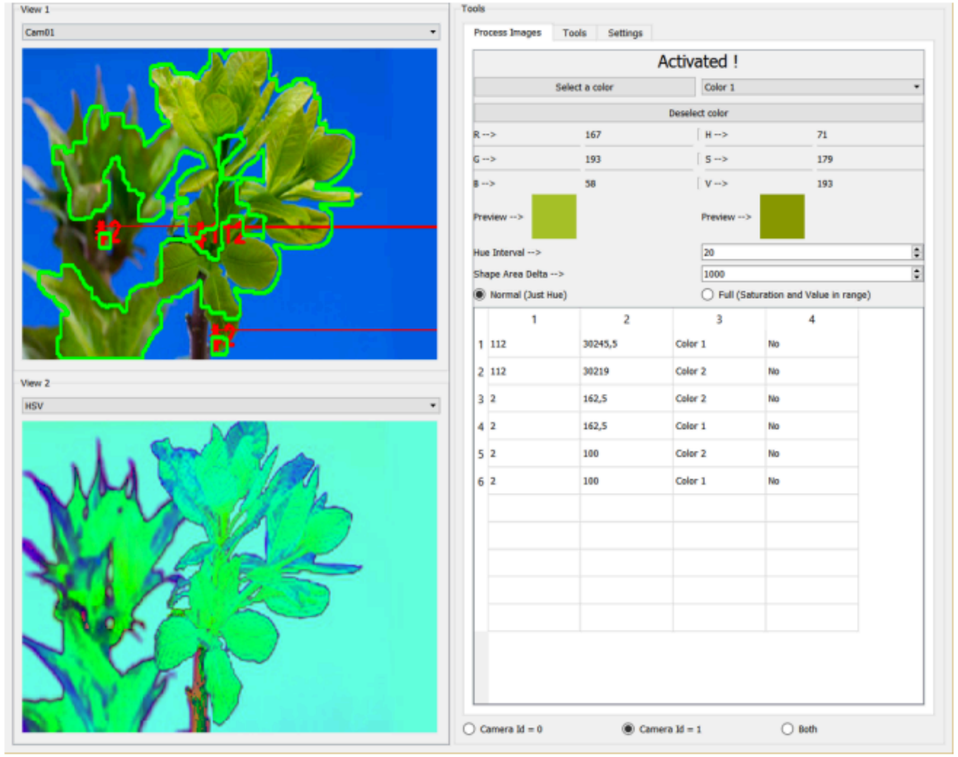
Before (left) and after makeup application (right)

# BU-CROCCS
# Research projects - image processing
## Low-cost phenotyping system for plant growth analysis

- Aim: develop a system for quantitative analysis of plant growth

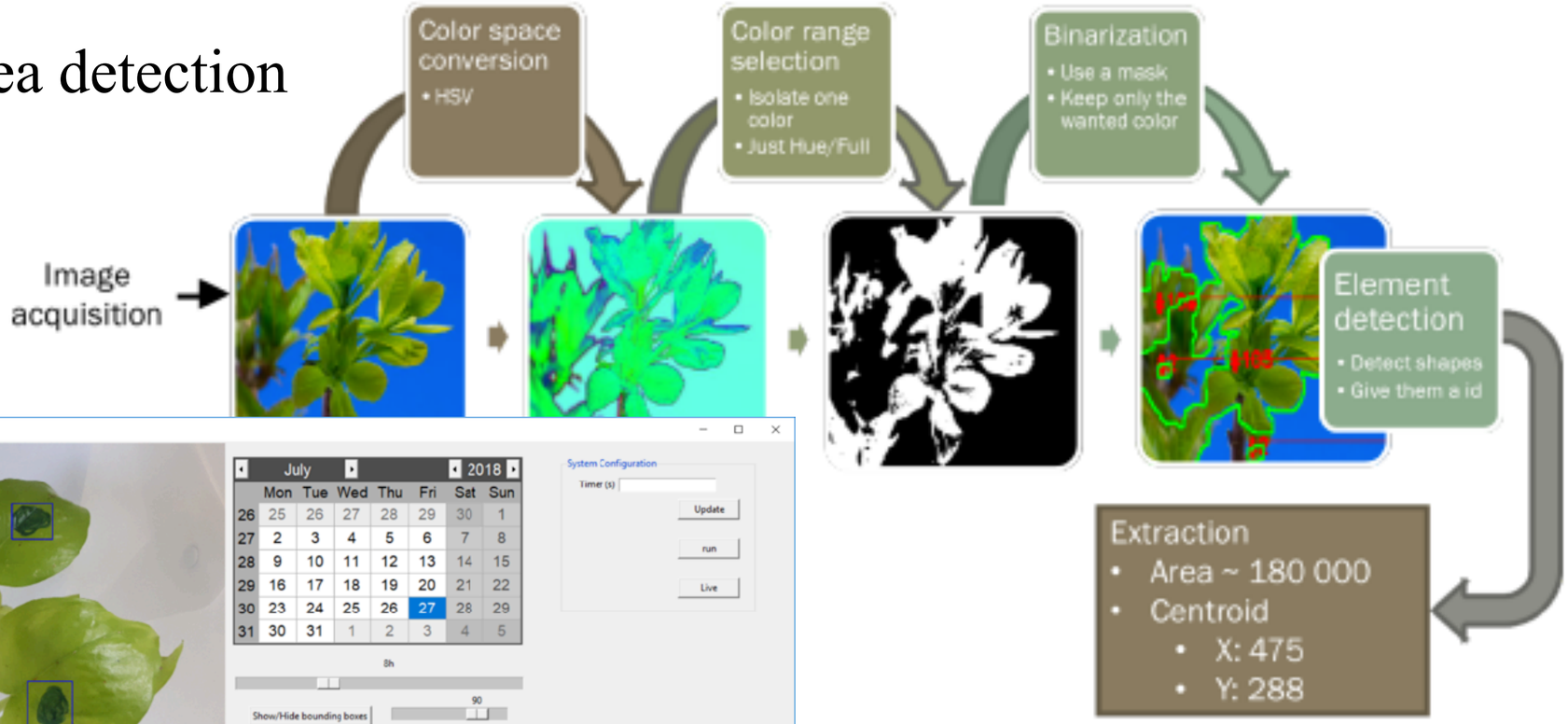
- System:
  - Raspberry Pi + camera
  - Image processing implementation

- Output:
  - Time-lapse recording
  - Plant growth speed
  - Biological features quantification

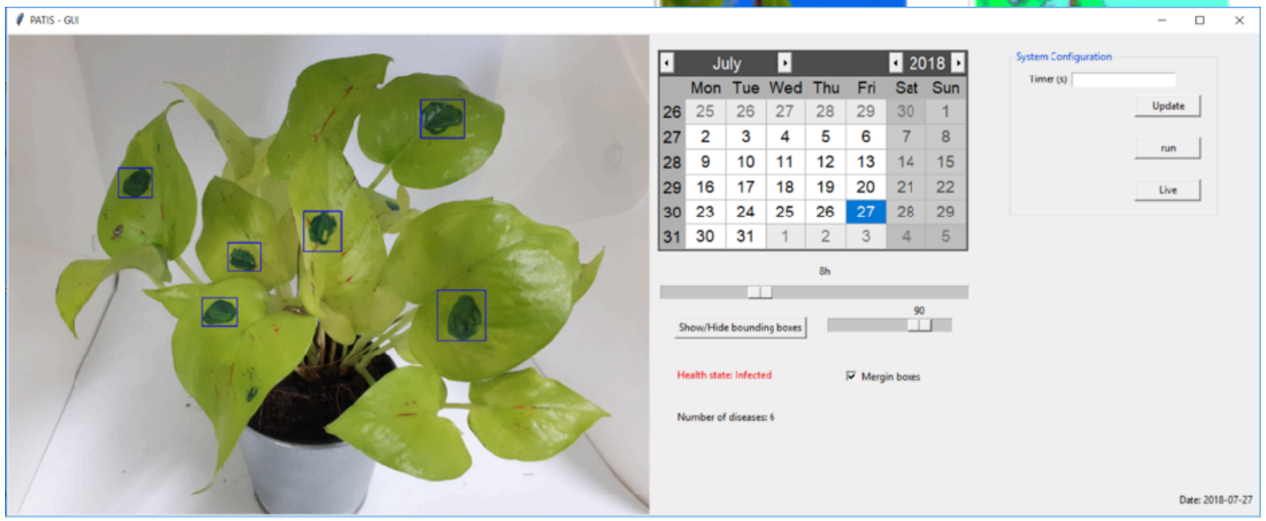


# BU-CROCCS
# Research projects - image processing
## Low-cost phenotyping system for plant growth analysis

- Automatic make-up area detection




- AI



# Outlines

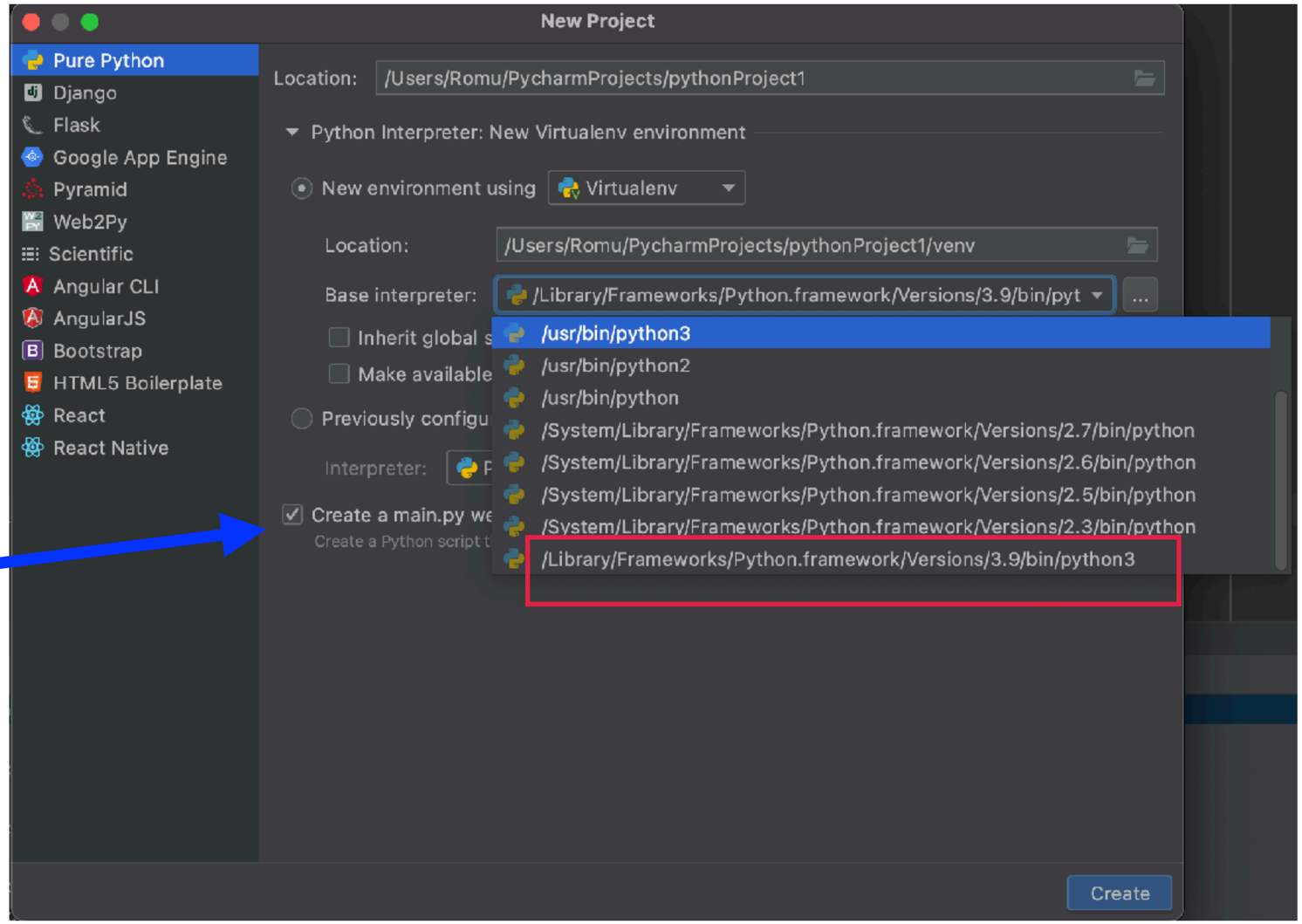
- General introduction
  - Bangkok University - BU-CROCCS
  - What is image processing?
  - Applications
- **Python + OpenCV**
  - Requirements
  - Setup on PyCharm
- Basic image processing implementations
- Advances implementations
- Conclusions

# PyCharm installation

1. Get a student license
2. Download PyCharm Professional IDE
3. Install PyCharm
4. Create New Project (with choice of interpreter)

# PyCharm - Create New Project

1. Allows selection of interpreter
2. Select version 3.9 (latest)
3. For demo, keep “Create a main.py” ticked
4. Create a virtual environment

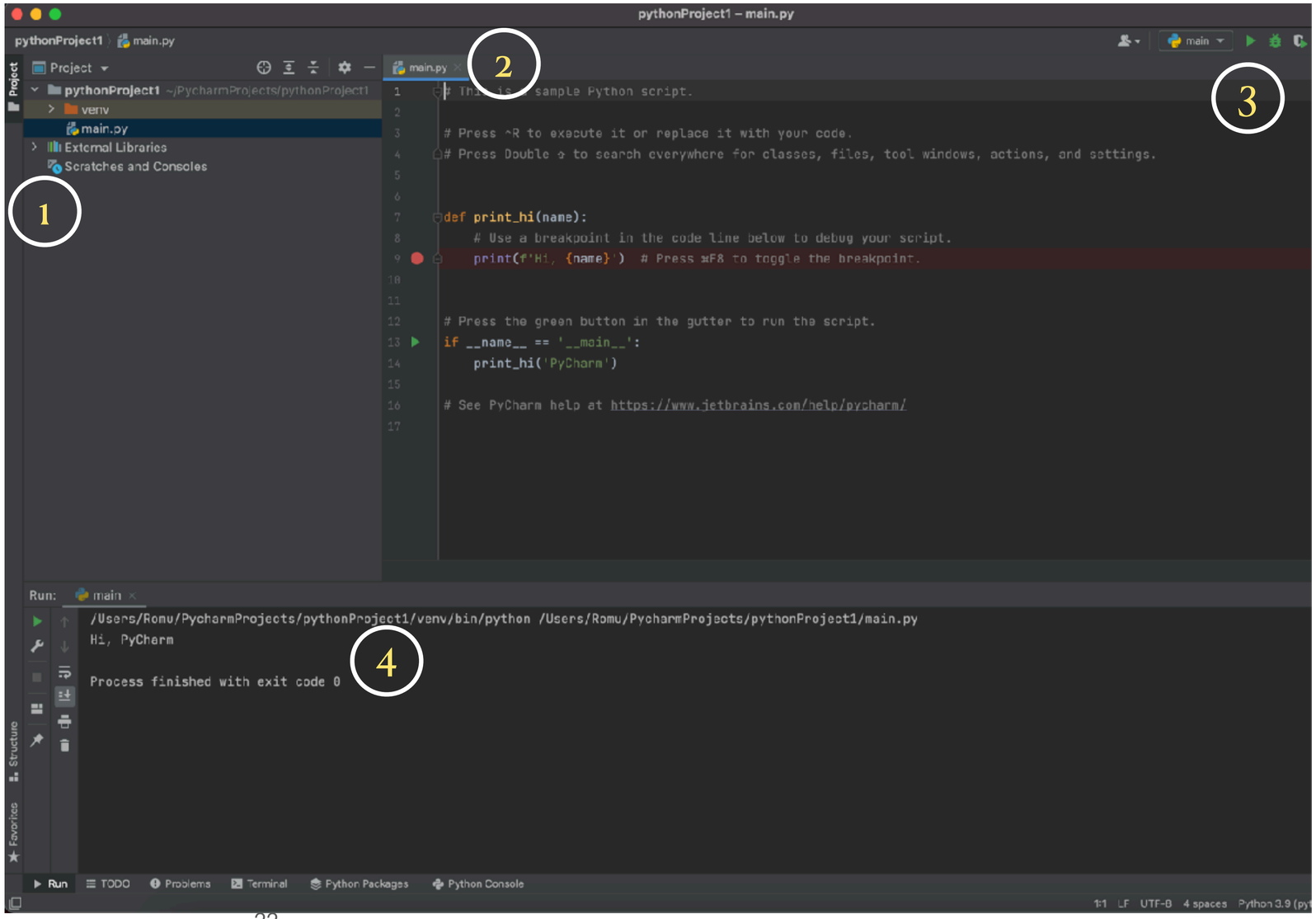


# PyCharm - New Project

PyCharm layout:

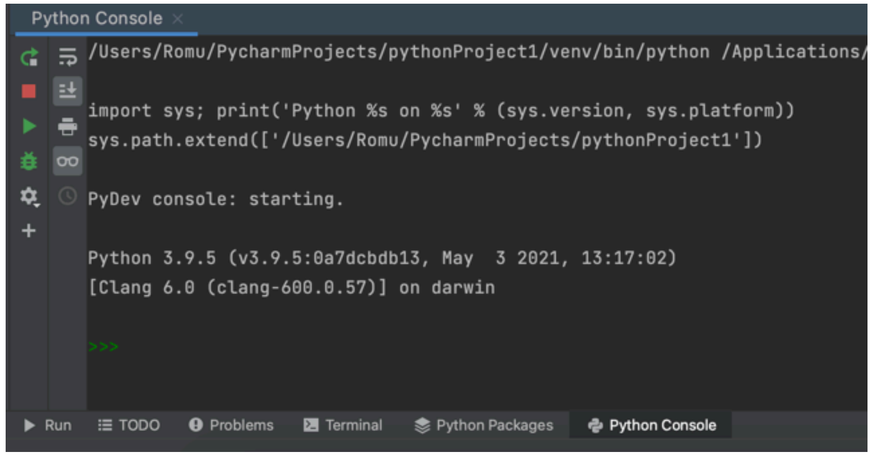
- Test example “main.py”

1. Folder
2. Name of current file
3. Running button
4. Run display

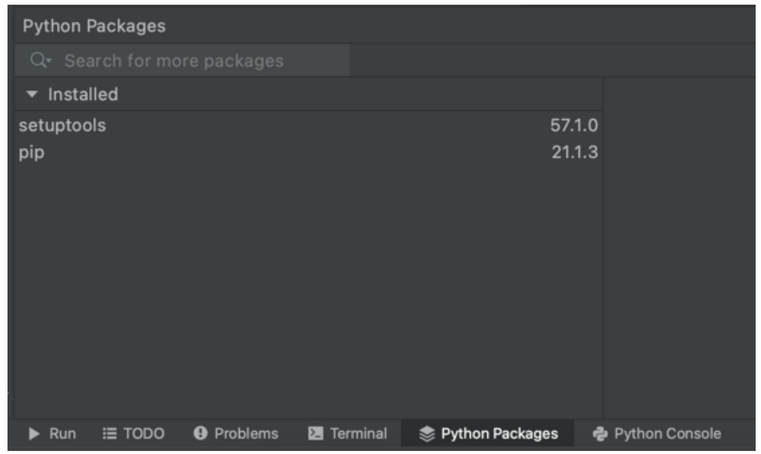


# PyCharm - New Project

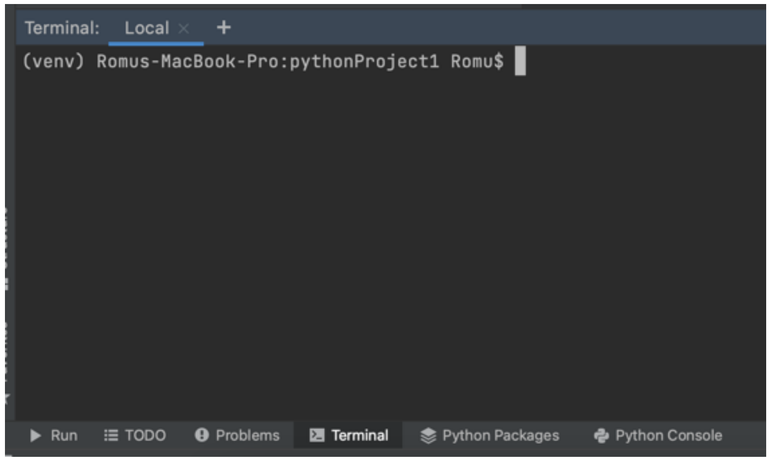
## Useful windows



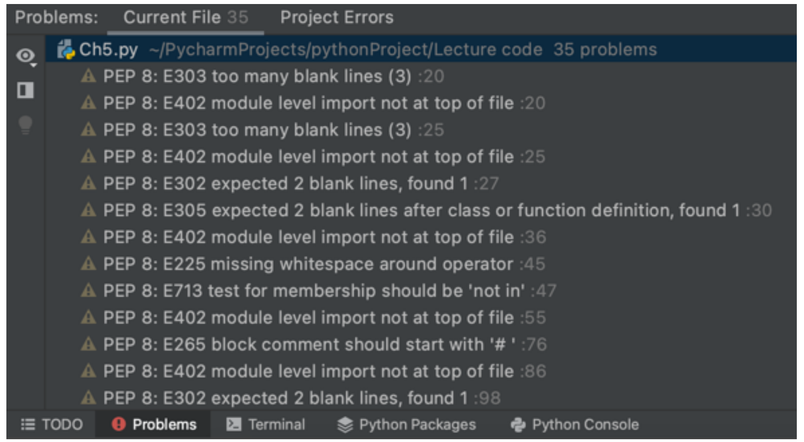

Python Console




Python Package installed

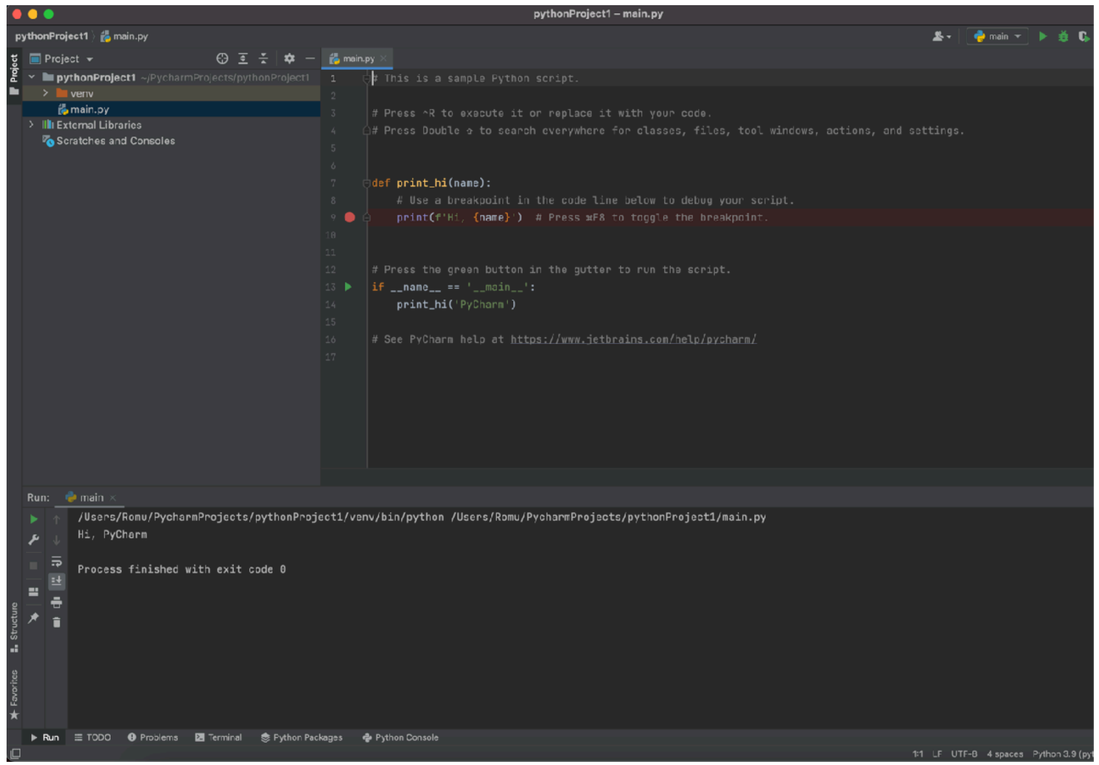



Terminal window

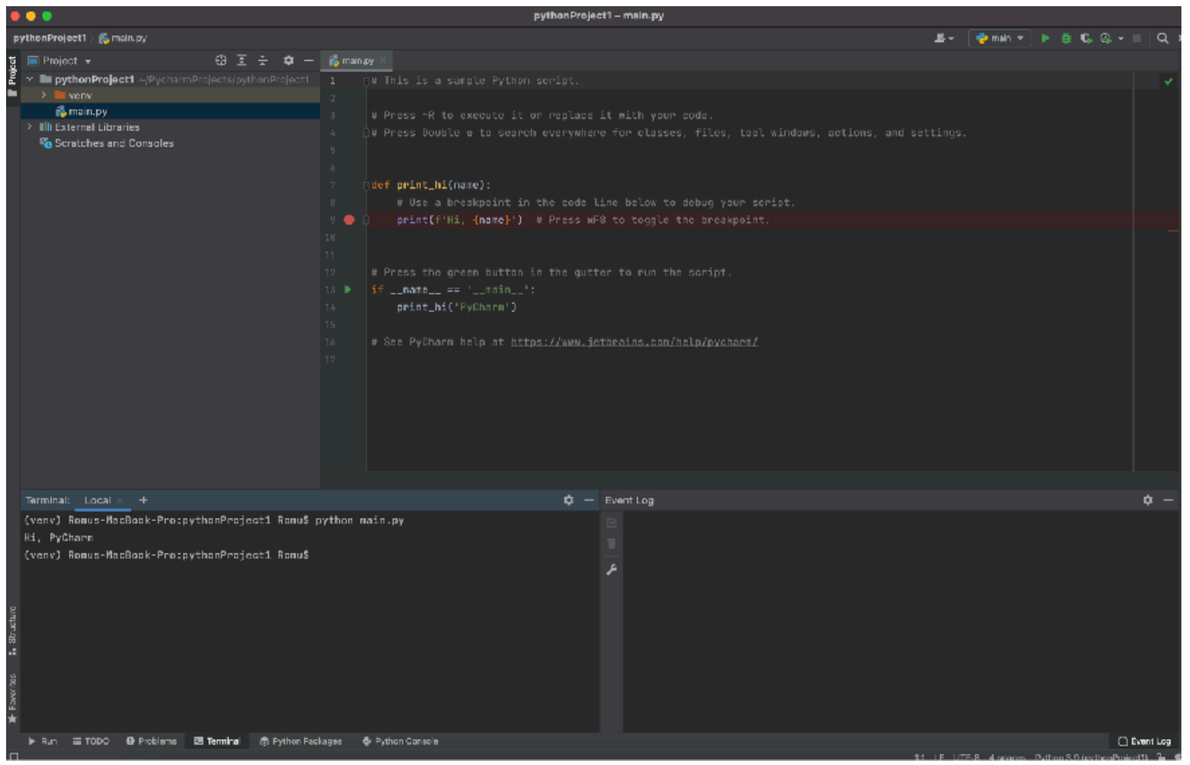



Problems window

# PyCharm - Run New Project



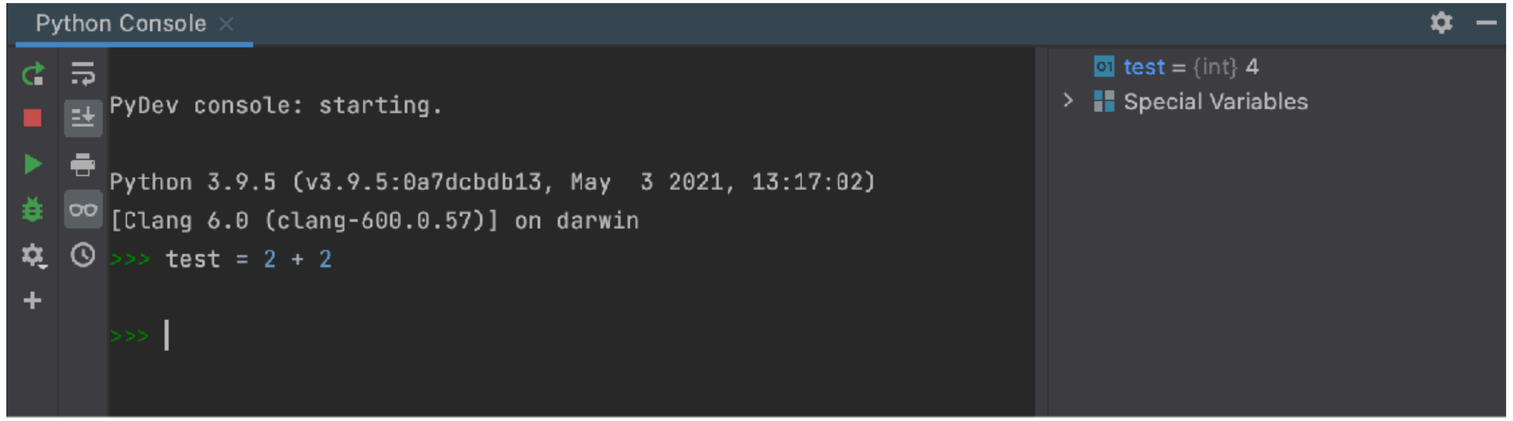

Run using “Run” short cut

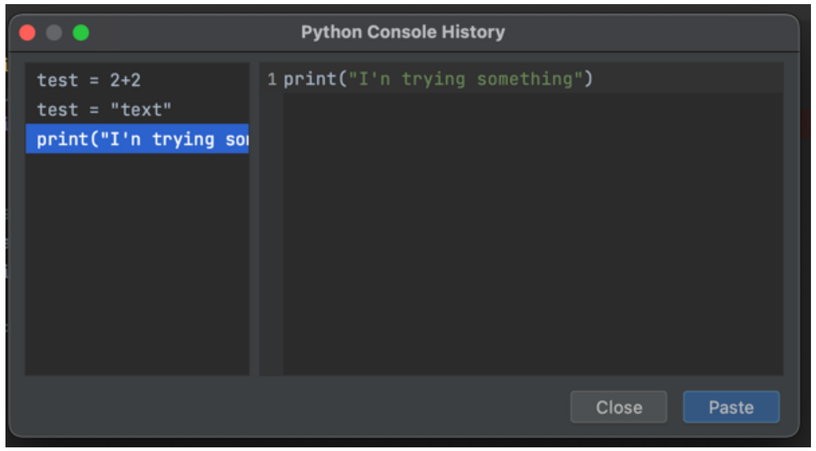



Run using terminal with “python main.py” command

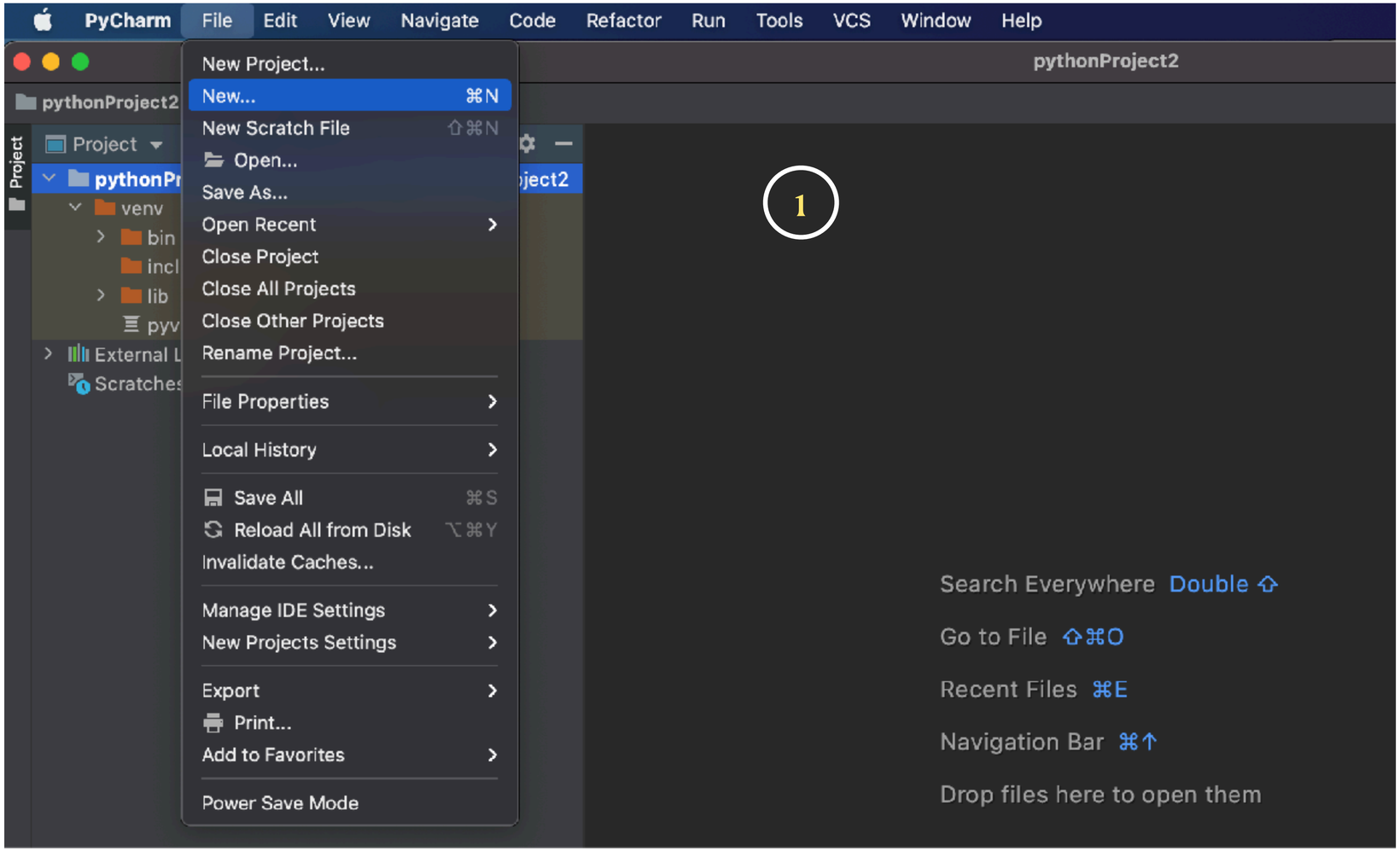
# PyCharm - Python Console

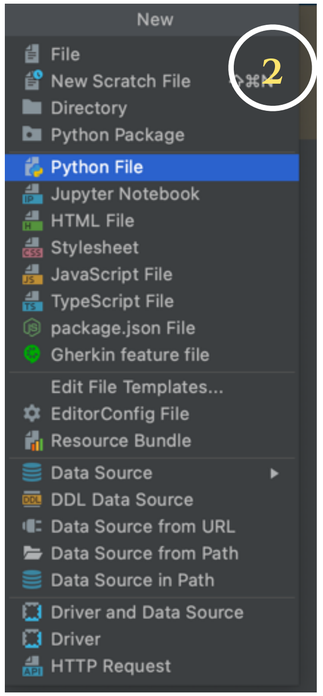
- Allow you to run python code directly
- IDE shows variable information
- Browse Queue history

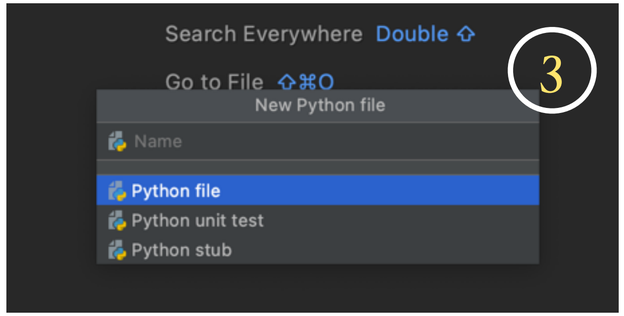


# PyCharm - Create new Python file



# Python + OpenCV

- Python is widely used
- Computer Vision
  - Everything related to processing image and video
- OpenCV
  - Most complete toolbox for image processing
    - Available on Windows, MacOS, Linux
    - C++, Python, Java and more
    - Python is very popular in the scientific community (similar to Matlab…but free)

# Python + OpenCV

- OpenCV
  - Written in C++
  - Free
  - Weak official documentation but improving (now dedicated module for python)
  - Loads of example online

# Python + OpenCV

## Requirements

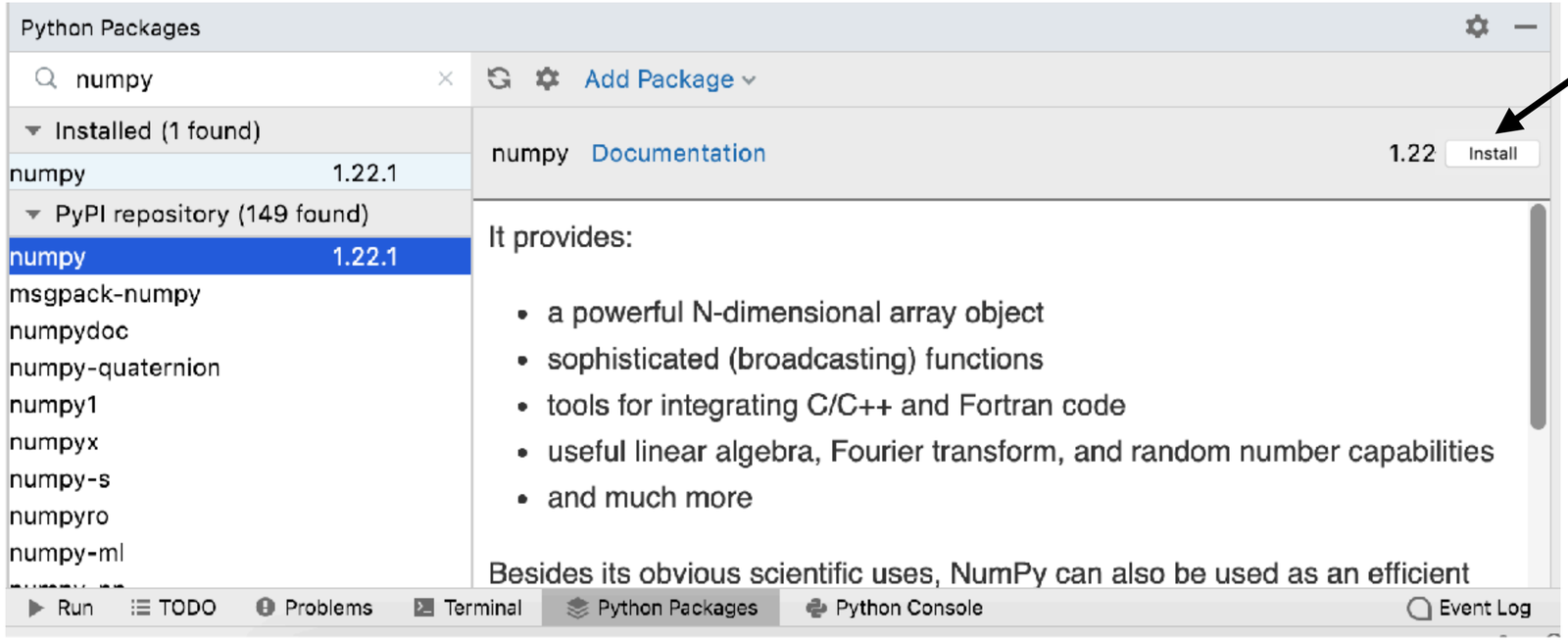
- Python requires packages
- Development on PyCharm (easy to setup) (pip or Python Packages)
- Image processing needs at least packages: 3

| Package name | name on Python package library | version |
|---|---|---|
| Numpy | numpy | 1.22.1 |
| Matplotlib | matplotlib | 3.5.1 |
| OpenCV | opencv-python | 4.5.5.62 |

# Python + OpenCV

## Requirements - Package installation

- Getting Numpy
  - Using Python Packages from PyCharm




Install button

# Python + OpenCV

## Requirements - Package installation

- Check if Numpy is properly installed

```
try:
    import numpy as np
except ImportError as e:
    print(f'Install numpy using "pip install numpy" command’)
else
    print(f’Numpy installed version: {np.__version__}’)
```

- If Numpy is not installed, it prints a message inviting you to install it

```
pip install numpy
```

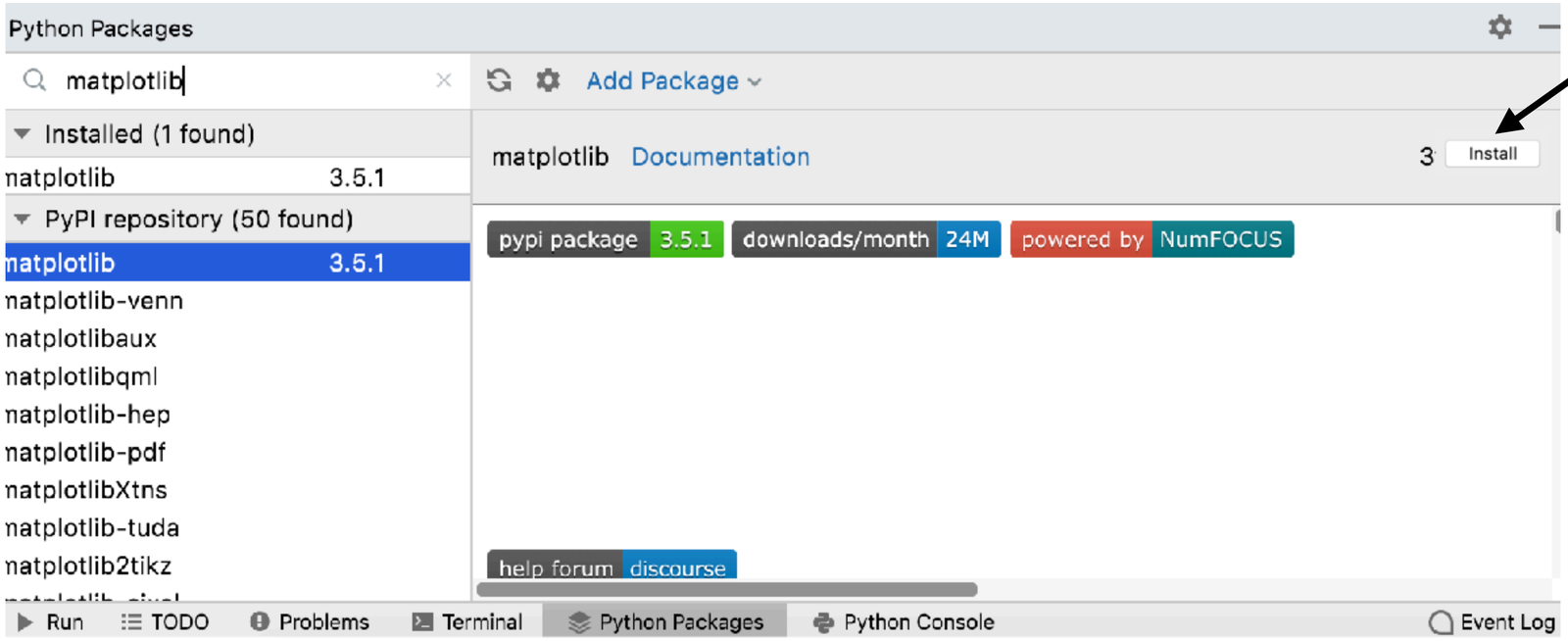
- Else it print the version on the system

```
Numpy installed version: 1.22.1
```

# Python + OpenCV

## Requirements - Package installation

- Getting Matplotlib
  - Using Python Packages from PyCharm




Install button

# Python + OpenCV

## Requirements - Package installation

- Check if Matplotlib is properly installed

```
try:
    import matplotlib as plt
except ImportError as e:
    print(f'Install Matplotlib using "pip install matplotlib" command’)
else
    print(f’Matplotlib installed version: {plt.__version__}’)
```

- If Matplotlib is not installed, it prints a message inviting you to install it

```
pip install matplotlib
```

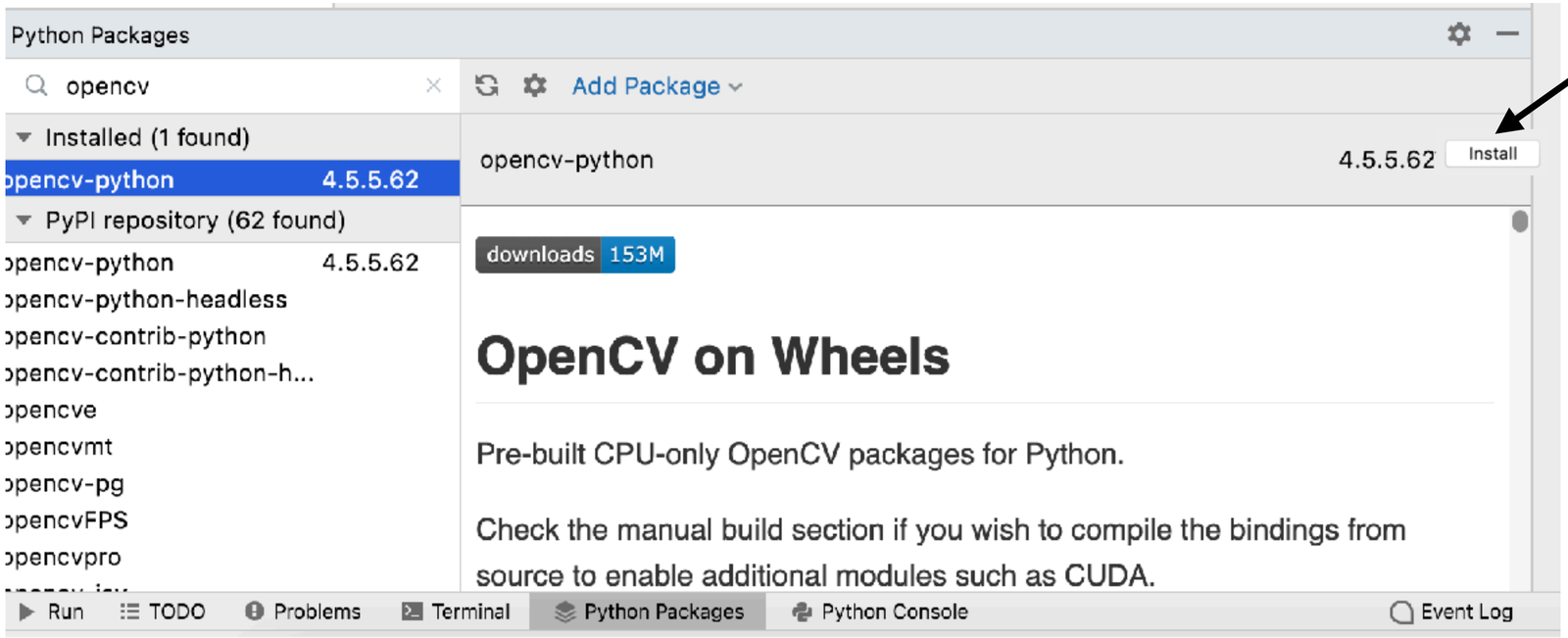
- Else it print the version on the system

```
Matplotlib installed version: 3.5.1
```

# Python + OpenCV

## Requirements - Package installation

- Getting OpenCV
  - Using Python Packages from PyCharm




Install button

# Python + OpenCV

## Requirements - Package installation

- Check if OpenCV is properly installed

```
try:
    import cv2
except ImportError as e:
    print(f'Install OpenCV using "pip install opencv-python“ command’)
else
    print(f’OpenCV installed version: {cv2.__version__}’)
```

- If Matplotlib is not installed, it prints a message inviting you to install it

```
pip install opencv-python
```

- Else it print the version on the system

```
OpenCV installed version: 4.5.5
```

# Python + OpenCV

## Requirements - Package installation

- You can check if all the three packages are installed using
  - “Check_installation.py”

# Structure of the lecture

- One - two slides explaining a concept
- One - two slides showing examples (provided in the link too)
- Highly recommend you to implement the examples yourself so you understand how it works.

# Advices

- This course brings you image processing basics using Python
- Highly recommend to test the codes and examples in the lectures
- Use other sources to improve your knowledge
- Mastering a language (spoken/programming) takes time
- Be curious
- Don’t be frustrated when it doesn’t work
- Don’t hesitate to ask questions

# Outlines

- General introduction
  - Bangkok University - BU-CROCCS
  - What is image processing?
  - Applications
- Python + OpenCV
  - Requirements
- **Basic image processing implementations**
- Instagram-like filter implementations
- Conclusions

# Basic image processing implementations

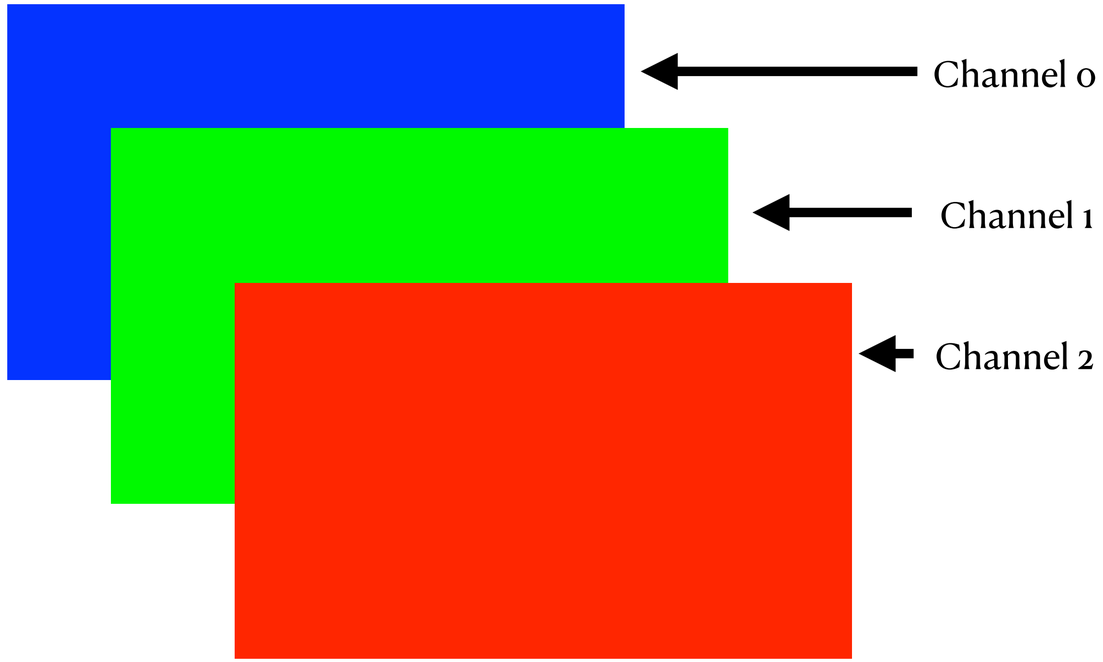
## Basic image processing outline

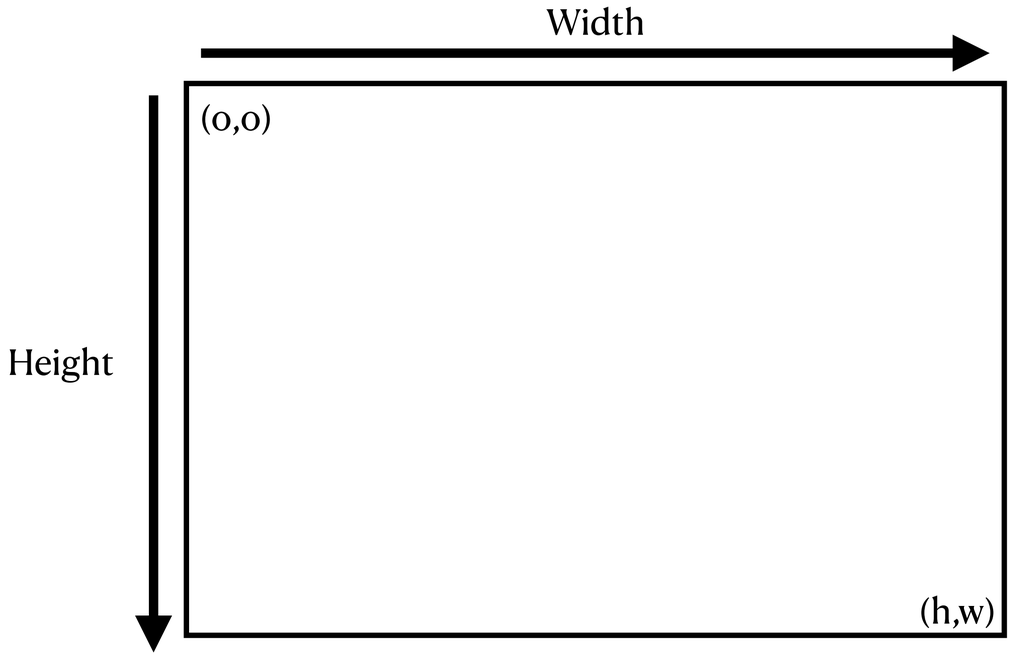
- **Basic**
  - Image reading
  - Image plotting
  - Image properties
  - Image writing
  - Plotting webcam
  - Recording video
  - Playing video
- Histogram
  - What is it?
  - Histogram equalization
  - CLAHE
- Binarization
  - Basic methods
  - Otsu
- Filtering
  - Smoothing filters
    - Average
    - Median
    - Gaussian
  - Sharpening filter
  - Edge Detection
    - Sobel
    - Canny

# Basic image processing implementations

## Basic OpenCV interaction

- Image formation:



# Basic image processing implementations

## Reading image

- Reading image using *imread()* function
  - Syntax
    - loaded_image = cv2.imread('filename.ext', flag)
- *imread* loads image in BGR color format (even for grayscale image)
- *filename.ext:* name of the file (can include path)
- *flag*: specific reading mode
  - cv2.IMREAD_COLOR (default option, not required), can be replaced by "1"
  - cv2.IMREAD_GRAYSCALE (8-bit grayscale image), can be replaced by "0"
  - more options (check online)

# Basic image processing implementations

## Plotting image

- Displaying image using *imshow()* function
  - Syntax
    - cv2.imshow('title text', variable)
- '*title text*' loads text to be displayed on the window
- *variable*: name of the variable you want to display
    - cv2.waitKey(time)
- *time*: time in millisecond for the windows to stay open, "0" to manually close it
    - cv2.destroyAllWindows()
- *variable*: name of the image variable you want to display

# Basic image processing implementations

## Reading and plotting image

- Example of displaying "*logo.png*" using "*loading_image.py*"

```
import cv2

# Read image using imread function
image = cv2.imread("logo.png", cv2.IMREAD_COLOR)

# Display image using imshow('title', variable)
cv2.imshow("logo BU-TM", image)

# Need to use waitKey(0) to keep the image displayed on
# waitKey(time_in_millisecond) to keep it on for a specified
time or 0 for the user to manually close it (pressing keyboard)
cv2.waitKey(0)

# clearing window and memory using (destroyAllWindows)
cv2.destroyAllWindows()
```


logo displayed in a new window

# Basic image processing implementations

## Reading image in grayscale and plotting it

- Example of displaying "*logo.png*" using "*loading_image.py*"
- Load in grayscale using "0" flag

```
import cv2

# Read image using imread function
image_gray = cv2.imread("logo.png", 0)


# Display image using imshow('title', variable)
cv2.imshow("logo BU-TM", image)

# Need to use waitKey(0) to keep the image displayed on
# waitKey(time_in_millisecond) to keep it on for a specified
time or 0 for the user to manually close it (pressing keyboard)
cv2.waitKey(0)

# clearing window and memory using (destroyAllWindows)
cv2.destroyAllWindows()
```


Grayscale logo displayed in a new window

# Basic image processing implementations

## Converting image in grayscale and plotting it

- Converting an image to grayscale using *cv2.cvtColor()* function
  - Syntax
    - grayim = cv2.cvtColor(colorim, cv2.COLOR_BGR2GRAY)
- '*colorim*' color image
- *cv2.COLOR_BGR2GRAY*: flag conversion from color to grayscale
  - other conversion available (check online)
- *grayim*: image converted into grayscale

# Basic image processing implementations

## Converting image in grayscale and plotting it

- Example of displaying "*logo.png*" using "*convert2gray.py*"

```
import cv2

image = cv2.imread("logo.png", cv2.IMREAD_COLOR)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow("logo BU-TM", image)
cv2.imshow("Gray converted logo BU-TM", gray)
cv2.waitKey(0)
cv2.destroyAllWindows()
```


logo displayed in a new window


Grayscale logo displayed in a new window

# Basic image processing implementations

## Extracting image properties

- Extracting properties of image using *shape*, *size*, and *dtype* functions:
- Shape: returns height, width(, channel)
  - Syntax
    - print(image.shape)
    - h, w = image.shape # for grayscale image
    - h, w, c = image.shape # for color image
- Size: returns number of elements in the array (h*w(*channel))
  - Syntax
    - image_size = image.size
- Dtype: Returns the datatype of the array
  - Syntax
    - image_type = image.dtype

# Basic image processing implementations

## Extracting image properties

- Example of properties extraction using "*image_properties.py*"

```
import cv2

image = cv2.imread('logo.png',1)

# Shape returns 3 values: height, width and number of channel in the image
h, w, c = image.shape
image_size = image.size
image_type = image.dtype

print(f'The image height is {h}, the image width is {w}, the color channel {c}')
print(f'The image size is {image_size} and image type {image_type}')
# Display image
cv2.imshow('Original Image', image)
cv2.waitKey(0)
cv2.destroyAllWindows()

>> The image height is 284, the image width is 748, the color channel 3
>> The image size is 637296 and image type uint8
```

# Basic image processing implementations

## Extracting color plans (BGR) in-built function and basic method

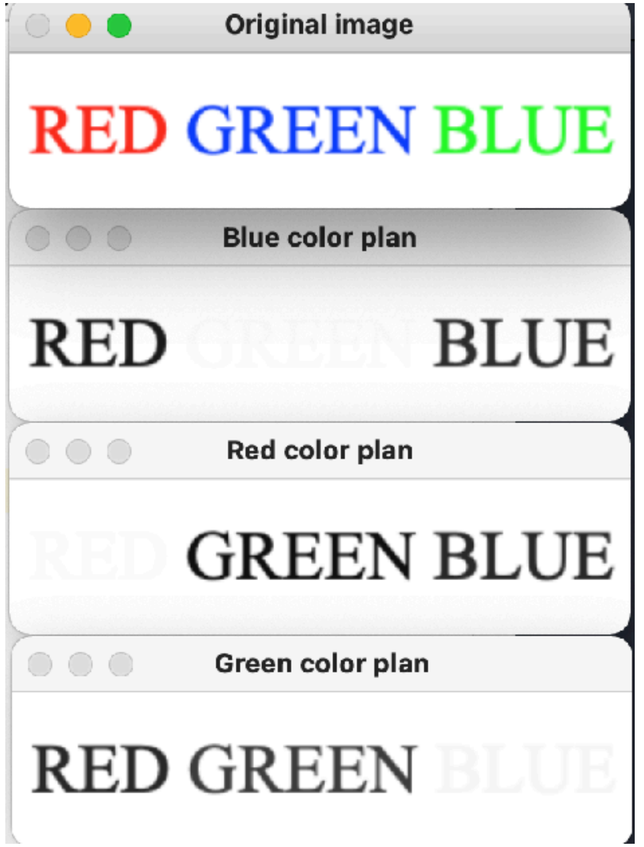
- Extracting color plans
- Method one - using *split*() function:
  - Syntax
    - planB, planG, planR = cv2.split(image)
- Method two - extracting plan directly:
  - Syntax
    - planB = image[:, :, 0]
    - planG = image[:, :, 1]
    - planR = image[:, :, 2]

# Basic image processing implementations

## Extracting color plans (BGR) in-built function and plotting them

- Example of separating "*color.png*" using "*colorseparation.py*"
- Method 1:
  - Display windows one by one

```
import cv2
image = cv2.imread("color.png", cv2.IMREAD_COLOR)
B, G, R = cv2.split(image)
# Display original image
cv2.imshow("Original image", image)
cv2.waitKey(0)
cv2.imshow("Blue color plan", B)
cv2.waitKey(0)
cv2.imshow("Green color plan", G)
cv2.waitKey(0)
cv2.imshow("Red color plan", R)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



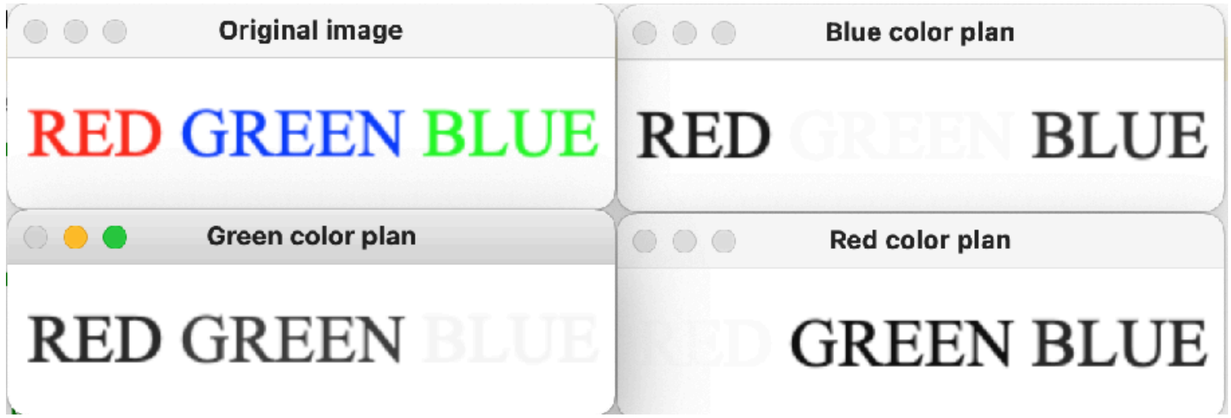

Each images is shown one after the other

# Basic image processing implementations

## Extracting color plans (BGR) basic method and plotting them

- Example of separating "*color.png*" using "*colorseparation2.py*"
- Method 2:
  - Display windows in one go

```
import cv2

# Read image using imread function
image = cv2.imread("color.png", cv2.IMREAD_COLOR)
B = image[:,:,0]
G = image[:,:,1]
R = image[:,:,2]
# Display original image
cv2.imshow("Original image", image)
cv2.imshow("Blue color plan", B)
cv2.imshow("Green color plan", G)
cv2.imshow("Red color plan", R)
cv2.waitKey(0)

cv2.destroyAllWindows()
```




All images are displayed at the same time

# Basic image processing implementations

## Writing image

- Writing an image using "*imwrite*()" function:
  - Syntax:
    - [status = ]cv2.imwrite('image_name.ext', img_var)
- [status = ]: optional, returns *True* if successful, *False* else
- 'image_name.ext': name of the file to save and specify extension
- img_var: variable to save
- Check if image is saved:
  - Syntax:

```
if status == True:
    print(f'Written status: {status}')
else:
    print(f'Image not saved')
```

# Basic image processing implementations

## Writing image

- Example of image saving using "*writing_image.py*"

```
import cv2

# Read image using imread function
image = cv2.imread("color.png", cv2.IMREAD_COLOR)

B, G, R = cv2.split(image)

# To get the status of writing the image, save the value into a variable
statusB = cv2.imwrite('colorB.png',B)
if statusB == True:
  print(f'Written B image status: {statusB}')
else:
  print(f'Image B not saved')

>> Written B image stature: True
```

# Basic image processing implementations

## Plotting live webcam

- Getting camera feed using *VideoCapture*() function:
  - Syntax:

```
cap = cv2.VideoCapture(0) # device (0) is usually the main webcam
while True:
    status, frame = cap.read()
    frame = cv2.flip(frame, 1)
    cv2.imshow('Webcam', frame)
    if cv2.waitKey(1) == 27: #esc to stop
        break
cap.release() # release the hardware (webcam)
cv2.destroyAllWindows() # close all windows
```

- status returns *True* if successful, else *False*
- frame the extracted image from the feed
- cv2.flip(image, flag) flips the image vertically if flag = 0, horizontally if flag =1 #this is optional
- cv2.waitKey(1) == 27(image, flag) flips 27 corresponds to "esc" key
- Check code using "*camerafeed.py*"

# Basic image processing implementations

## Recording video

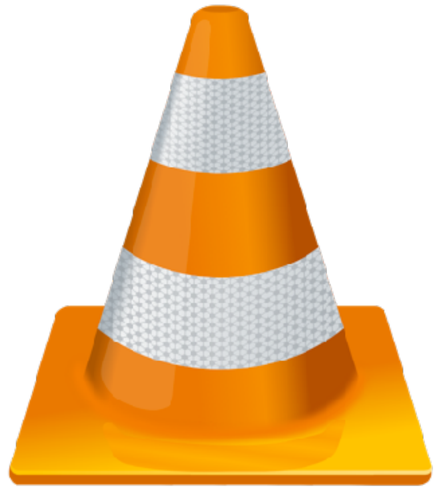
- Record a webcam feed using *VideoWriter()* function
- Need to run the webcam feed using *VideoCapture()*
  - Syntax:
    - codec = cv2.VideoWriter_fourcc(*'DIVX') # Other codecs available
    - out = cv2.VideoWriter('mywebcamvideo.avi', codec, frame_rate, (width, height))
    - out.write(frame)
    - out.release()
- codec: is a 4-byte code used to specify the codec (also referred to as fourcc)
- out: define the video information (name of the file, fourcc codec, number of frame, image size)
- out.write(frame): save the image "frame" using the parameter of out in the specified file
- out.release(): release the video

# Basic image processing implementations

## Recording video

- Try example using "*record_webcam_video.py*":

```
import cv2
# Capture video from webcam
cap = cv2.VideoCapture(0)
# frame counter
currentFrame = 0
# Get webcam width
width = cap.get(cv2.CAP_PROP_FRAME_WIDTH)
# Get webcam height
height = cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
# Set codec
codec = cv2.VideoWriter_fourcc(*"DIVX")
out = cv2.VideoWriter('myrecordedvideo.avi', codec, 24.0, (int(width), int(height)))

while cap.isOpened(): # method checking if the capture is properly initialized
    ret_val, frame = cap.read()
    frame = cv2.flip(frame, 1)     # flip the video
    out.write(frame) # save the frame
    cv2.imshow('Live webcam', frame)
    if cv2.waitKey(1) == 27:
        break  # esc to quit and stop the video
    currentFrame += 1
cap.release() # release camera
out.release() # release video
cv2.destroyAllWindows()
```



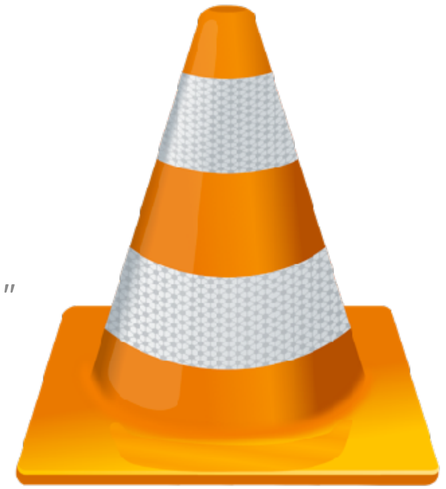
my_webcamv_ideo.avi

# Basic image processing implementations

## Playing video

- Try example using "*play_video.py*":

```
import cv2
cap = cv2.VideoCapture('myrecordedvideo.avi')
while cap.isOpened(): # Check that the frame are correctly loaded
    ret, frame = cap.read() # read each frame
    if not ret: # if a frame is defective, print a message
        print(f"problem with frame")
        break
    cv2.imshow('frame', frame)
    if cv2.waitKey(25) == ord('q'): # print a frame every 25 ms or quit with "q"
        break
cap.release()
cv2.destroyAllWindows()
```



my_webcamv_ideo.avi

# Basic image processing implementations

## Basic image processing outline

- Basic
  - Image reading
  - Image plotting
  - Image properties
  - Image writing
  - Plotting webcam
  - Recording video
  - Playing video
- **Histogram**
  - What is it?
  - Histogram equalization
  - CLAHE
- Binarization
  - Basic methods
  - Otsu
- Filtering
  - Smoothing filters
    - Average
    - Median
    - Gaussian
  - Sharpening filter
  - Edge Detection
    - Sobel
    - Canny

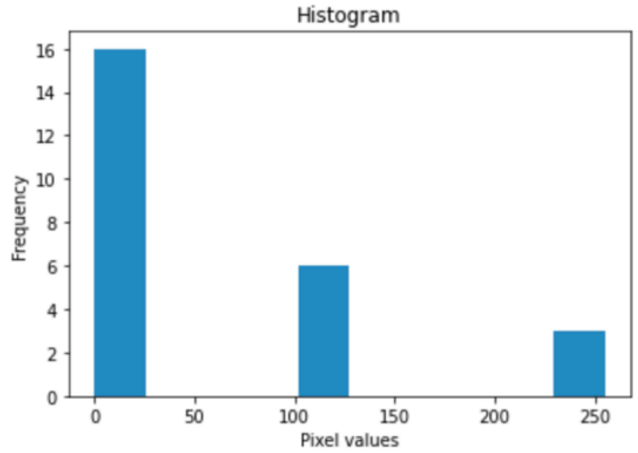
# Basic image processing implementations

## Image histogram

- What is an histogram?
  - Distribution of the pixel intensities in an image
  - Statistical information

- Useful transformation:
  - Equalization
    - Exploit the dynamic range of the image
    - Increase the contrast

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 255 | 122 | 122 | 0 |
| 0 | 255 | 122 | 122 | 0 |
| 0 | 255 | 122 | 122 | 0 |
| 0 | 0 | 0 | 0 | 0 |



# Basic image processing implementations

## Plotting histogram using OpenCV of grayscale image

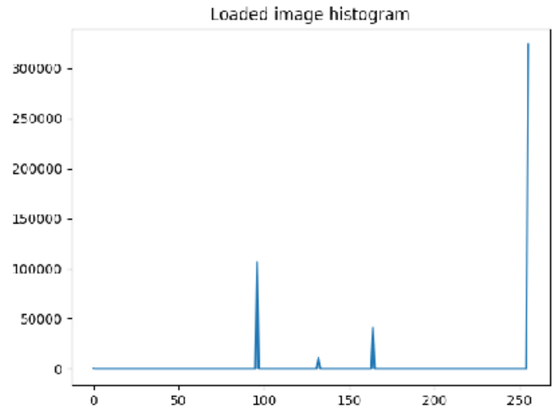
- Extract image histogram using *cv2.calcHist*() function:
  - Syntax:
    - cv2.calcHist([image], channel, mask, histSize, ranges[, hist[, accumulate]])
- [image]: image source
- channel: index of channel, [0] for grayscale, [0], [1], [2] for blue, green, red channel
- mask: to work on full image, mask set as “None”
- histSize: number of bin count (full scale 256)
- ranges: define range [0, 256]

# Basic image processing implementations

## Plotting histogram using OpenCV of grayscale image

- Try example using "*histogramOpenCV.py*":

```
import cv2
from matplotlib import pyplot as plt

# read image
image= cv2.imread('test.png', 0) # 0 to read in grayscale

# Get histogram from 0 to 255
hist = cv2.calcHist([image], [0], None, [256], [0, 256])
cv2.imshow('Loaded image',image)
cv2.waitKey(0)
# show the plotting graph of an image
plt.plot(hist)
plt.title('Loaded image histogram')
plt.show()
cv2.waitKey(0)
```




Histogram

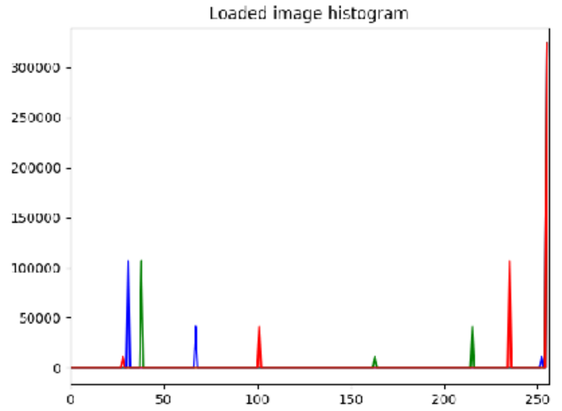
# Basic image processing implementations

## Plotting histogram using OpenCV of color image

- Try example using "*histogramcolorimage.py*":

```
import cv2
from matplotlib import pyplot as plt

image = cv2.imread('imgcolor.png')
color = ('Blue','Green','Red')
for i,col in enumerate(color):
    hist = cv2.calcHist([image],[i],None,[256],[0,256])
    plt.plot(hist,color = col)
    plt.xlim([0,256])
plt.title('Loaded image histogram')
plt.show()
cv2.imshow('Loaded image',image)
cv2.waitKey(0)
```

Original image




Histogram

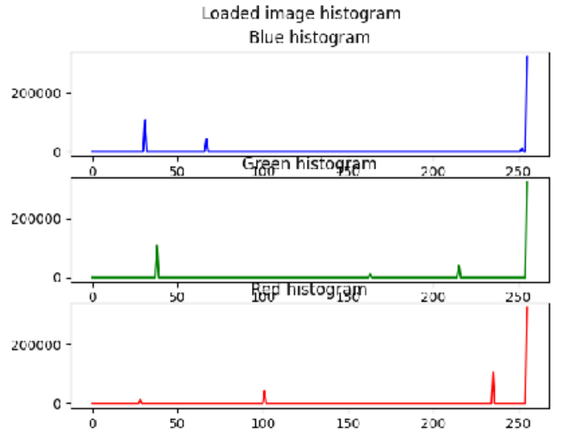
# Basic image processing implementations

## Plotting histogram using OpenCV of color image

- Try example using "*histogramcolorimage2.py*":

```
import cv2
from matplotlib import pyplot as plt
image = cv2.imread('imgcolor.png')
color = ('Blue','Green','Red')
histb = cv2.calcHist([image],[0],None,[256],[0,256]) # extract hist of b
histg = cv2.calcHist([image],[1],None,[256],[0,256]) # extract hist of g
histr = cv2.calcHist([image],[2],None,[256],[0,256]) # extract hist of r
plt.subplot(3,1,1) # use subplot
plt.plot(histb,color = 'Blue')
plt.title('Blue histogram')
plt.subplot(3,1,2)
plt.plot(histg,color = 'Green')
plt.title('Green histogram')
plt.subplot(3,1,3)
plt.plot(histr,color = 'Red')
plt.title('Red histogram')
plt.suptitle('Loaded image histogram')
plt.show()
cv2.imshow('Loaded image',image)
cv2.waitKey(0)
```

Original image




Histogram

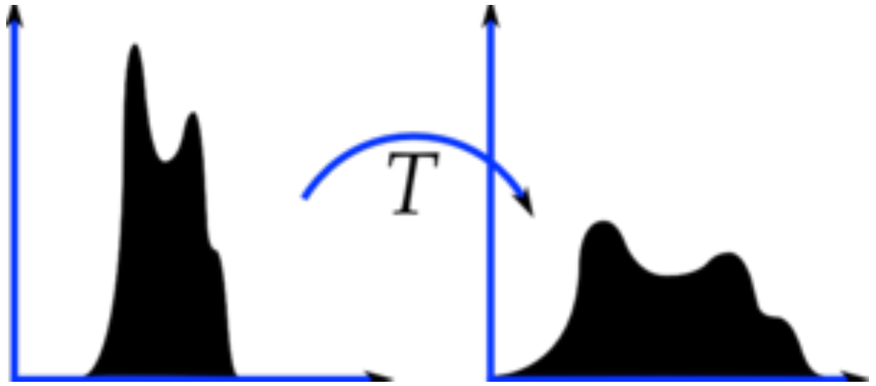
# Basic image processing implementations

## Image histogram equalization

- What is an histogram equalization?
  - Aims to increase global contrast by extending the dynamic range distribution

$$f_{new}[x, y] = (2^D - 1).\frac{CH(f[x, y])}{wh}$$

- were D: the dynamic range (8 bits), (w, h): image dimension (width and height respectively, CH: the cumulated histogram




Histogram equalization principle*



*Histograms - 2: Histogram Equalization, Open Source Computer Vision, accessed online at https://docs.opencv.org/3.4/d5/daf/tutorial_py_histogram_equalization.html December 18th, 2021

# Basic image processing implementations

## Image histogram equalization

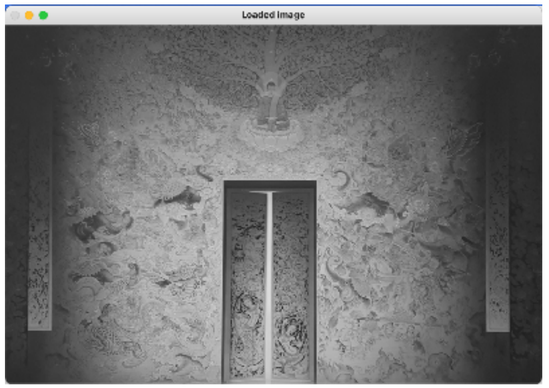
- Perform thresholding using *cv2.equalizeHist*() function:
  - Syntax:
    - eqimage = cv2.equalizeHist(image)
- image: image source (grayscale)
- eqimage: histogram equalized image

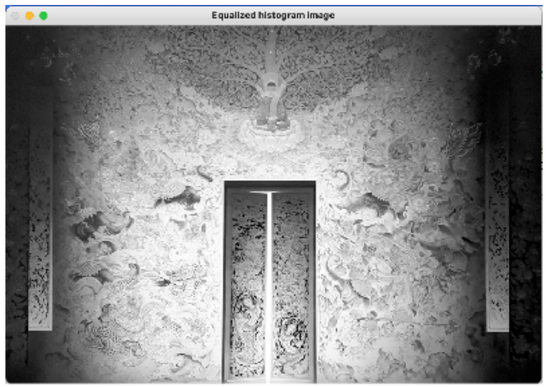
# Basic image processing implementations

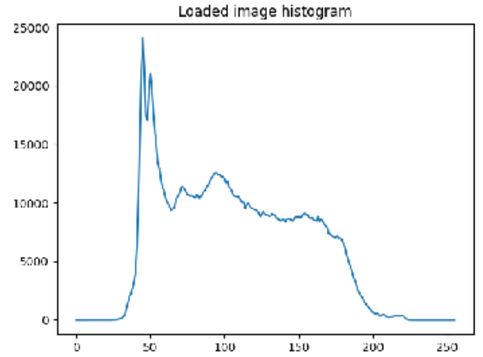
## Image histogram equalization

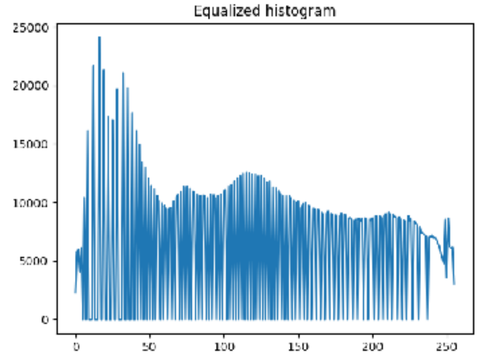
- Try example using "*histogramequalize.py*":

```
import cv2
from matplotlib import pyplot as plt
image= cv2.imread('image.png', 0) # 0 to read in grayscale
# Get histogram from 0 to 255
hist = cv2.calcHist([image], [0], None, [256], [0, 256])
histeqimage = cv2.equalizeHist(image) # apply histogram
equalization
histeq = cv2.calcHist([histeqimage], [0], None, [256], [0,
256]) # Get the new histogram
cv2.imshow('Loaded image',image) # Display original image
cv2.imshow('Equalized histogram image',histeqimage) # Display
equalized histogram image
# Plot each histogram
plt.plot(hist)
plt.title('Loaded image histogram')
plt.show()
plt.plot(histeq)
plt.title('Equalized histogram')
plt.show()
cv2.waitKey(0)
```



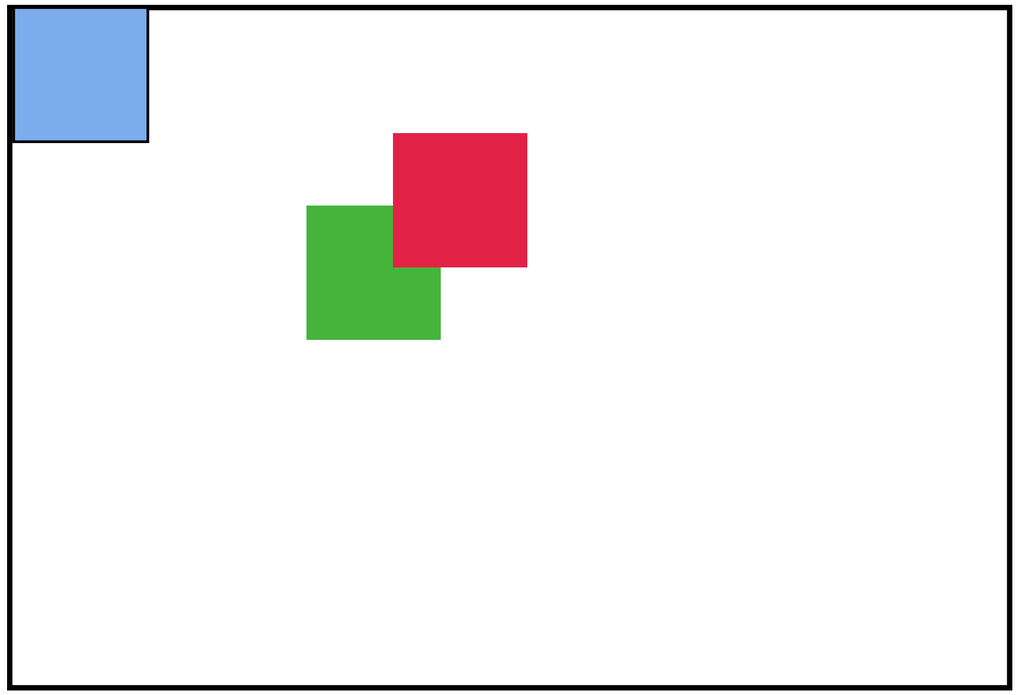



# Basic image processing implementations

## Image histogram equalization - CLAHE

- What is CLAHE?
  - Contrast Limited Adaptive Histogram Equalization
  - Parameter 1: contrast limit
  - Parameter 2: tile size

- Equivalent to histogram equalization
- Limited to small region
- Avoid noise amplification



Example of CLAHE moving tile

# Basic image processing implementations

## Image histogram equalization

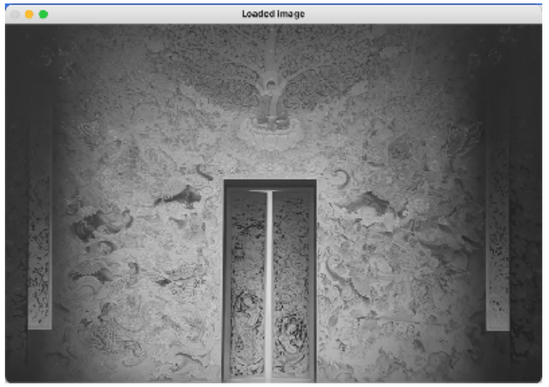
- Perform thresholding using "*cv2.createCLAHE*()" function:
  - Syntax:
  - 1: Initiate CLAHE algorithm
    - clahe = cv2.createCLAHE([clipLimit = 40], [titleGridSize = (8,8)])
  - 2: Apply method on CLAHE object
    - claheimage = clahe.apply(image)
- clipLimit: contrast limit (optional) default value 40.0
- titleGridSize: tiles grid size (optional) default value 8x8
- clahe: CLAHE object to be applied on the image
- claheimage: histogram equalized image using CLAHE

# Basic image processing implementations

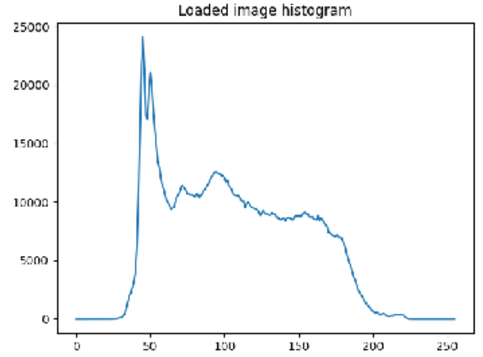
## Image histogram equalization - CLAHE (8x8)

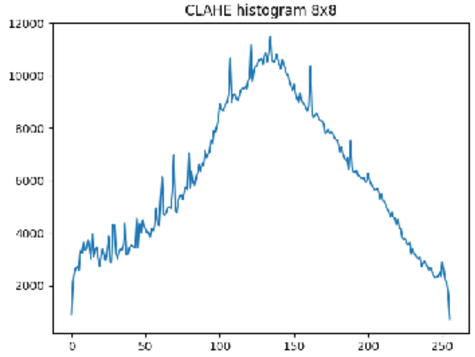
- Try example using "*histogramCLAHE.py*":

```
import cv2
from matplotlib import pyplot as plt
image= cv2.imread('image.png', 0) # 0 to read in grayscale
# Get histogram from 0 to 255
hist = cv2.calcHist([image], [0], None, [256], [0, 256])
# create a CLAHE object (Arguments are optional).
clahe = cv2.createCLAHE() #clipLimit=40.0, tileGridSize=(8,8)
claheimage = clahe.apply(image)
CLAHEhisteq = cv2.calcHist([claheimage], [0], None, [256],
[0, 256])
cv2.imshow('Loaded image',image) # Plot original image
cv2.imshow('CLAHE image 8x8',claheimage) # Plot CLAHE image
# show the plotting graph of an image
plt.plot(hist)
plt.title('Loaded image histogram')
plt.show()
plt.plot(CLAHEhisteq)
plt.title('CLAHE histogram 8x8')
plt.show()
cv2.waitKey(0) # Close the windows
```







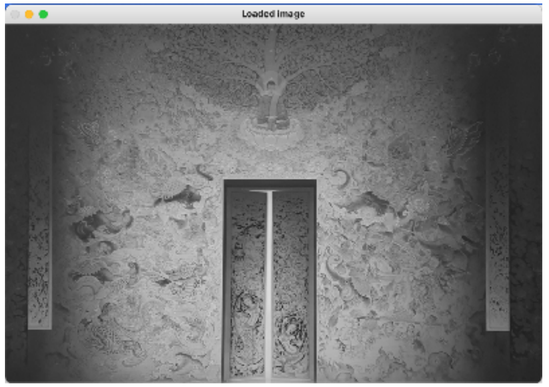



# Basic image processing implementations

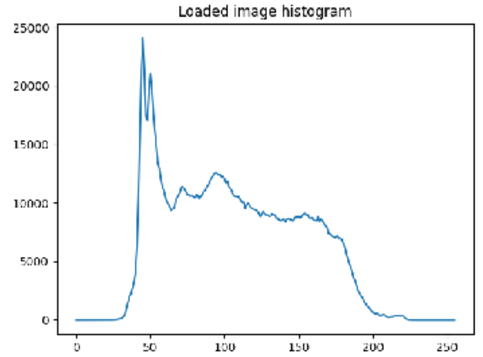
## Image histogram equalization - CLAHE (16x16)

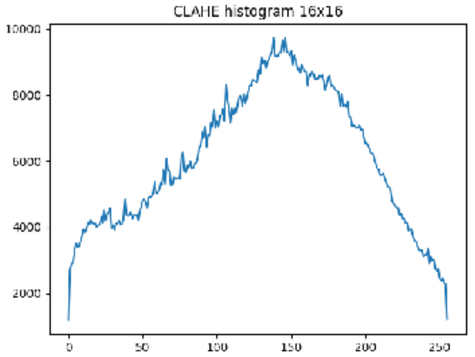
- Try example using "*histogramCLAHE.py*":

```
import cv2
from matplotlib import pyplot as plt
image= cv2.imread('image.png', 0) # 0 to read in grayscale
# Get histogram from 0 to 255
hist = cv2.calcHist([image], [0], None, [256], [0, 256])
# create a CLAHE object (Arguments are optional).
clahe = cv2.createCLAHE(clipLimit=40.0, tileGridSize=(16,16))
claheimage = clahe.apply(image)
CLAHEhisteq = cv2.calcHist([claheimage], [0], None, [256],
[0, 256])
cv2.imshow('Loaded image',image) # Plot original image
cv2.imshow('CLAHE image 16x16',claheimage) # Plot CLAHE image
# show the plotting graph of an image
plt.plot(hist)
plt.title('Loaded image histogram')
plt.show()
plt.plot(CLAHEhisteq)
plt.title('CLAHE histogram 16x16')
plt.show()
cv2.waitKey(0) # Close the windows
```



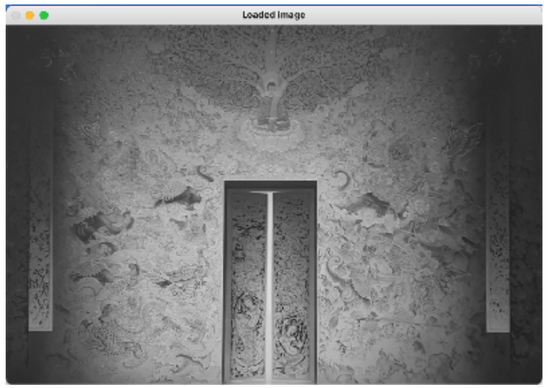


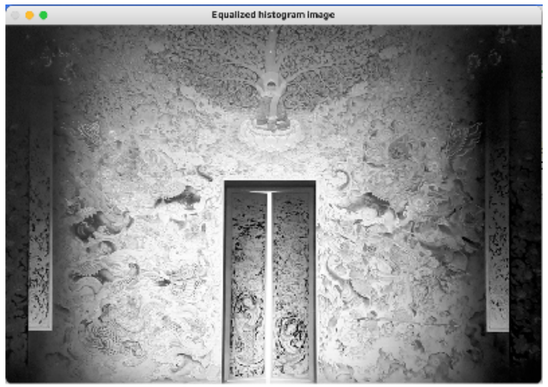


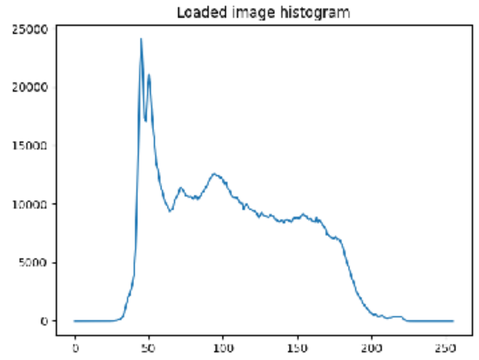
# Basic image processing implementations

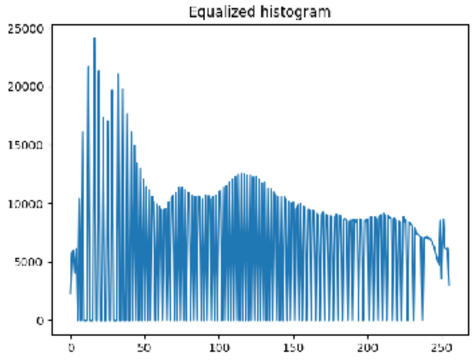
## Image - histogram equalization - CLAHE (8x8) - CLAHE (16x16)

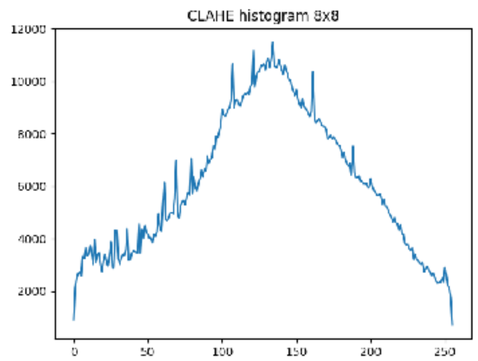


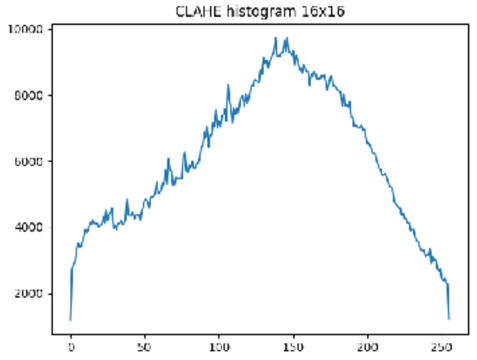





# Basic image processing implementations

## Basic image processing outline

- Basic
  - Image reading
  - Image plotting
  - Image properties
  - Image writing
  - Plotting webcam
  - Recording video
  - Playing video
- Histogram
  - What is it?
  - Histogram equalization
  - CLAHE
- **Binarization**
  - Basic methods
  - Otsu
- Filtering
  - Smoothing filters
    - Average
    - Median
    - Gaussian
  - Sharpening filter
  - Edge Detection
    - Sobel
    - Canny

# Basic image processing implementations

## Thresholding

- Binarization of image:
- Most basic method to segment an image
  - Convert grayscale image into black and white image
- Compare pixel intensity value with a set threshold
  - $f_{thresholded}(x,y) = \begin{cases} f_o & for \quad f(x,y) < V_{tresh} \\ f_{255} & for \quad f(x,y) > V_{tresh} \end{cases}$
- were $f_{thresholded}(x,y)$ is the new pixel value after binarization at location x and y
- $f_o$ is the new pixel value set to **zero** if the original pixel value is lower or equal to $V_{tresh}$
- $f_{255}$ is the new pixel value set to **255** if the original pixel value is higher than $V_{tresh}$
- $V_{thresh}$ is the threshold value

# Basic image processing implementations

## Thresholding

- Perform thresholding using *cv2.threshold*() function:
  - Syntax:
    - ret, threshim = cv2.threshold(image, thresholdValue, maxValue, thresholdingTechnique)
- image: image source (grayscale)
- thresholdValue: threshold value
- maxValue: Maximum value assigned to a pixel
- thresholdTechnique: type of thresholding applied
  - cv2.THRESH_BINARY (basic binarization)
  - cv2.THRESH_BINARY_INV (inverted of basic binarization)
  - cv2.THRESH_TRUNC (pixel value set to threshold value if higher)
  - cv2.THRESH_TOZERO (pixel set to zero for all intensity lower than threshold)
  - cv2.THRESH_TOZERO_INV (inverted TOZERO threshold method)
- ret: threshold value
- threshim: binarized image

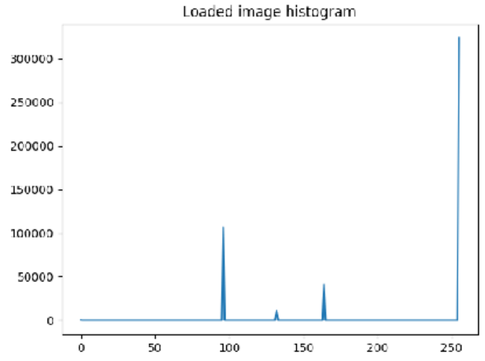
# Basic image processing implementations

## Thresholding

- Try example using "*simplebinarization.py*":
  - Threshold value: 120

```
import cv2

image = cv2.imread('imgcolor.png',0) #Read image in grayscale
cv2.imshow('Loaded image',image) # Plot grayscale image
ret, thresh1 = cv2.threshold(image, 120, 255, cv2.THRESH_BINARY) # Apply binarization with threshold of 120
cv2.imshow('Thresholded image',thresh1) # Plot the binarized image
cv2.waitKey(0) # Close windows
cv2.destroyAllWindows()
```

Grayscale original image




Binarized image with threshold value of 120

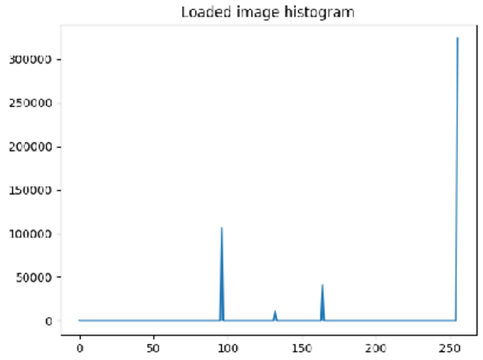
# Basic image processing implementations

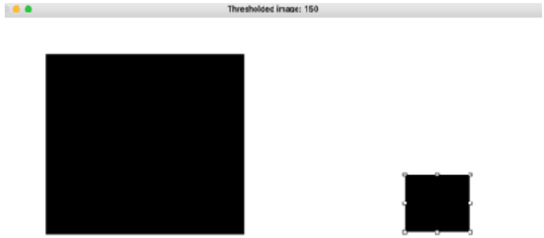
## Thresholding

- Try example using "*simplebinarization.py*":
  - Threshold value: 150

```
import cv2

image = cv2.imread('imgcolor.png',0) #Read image in grayscale
cv2.imshow('Loaded image',image) # Plot grayscale image
ret, thresh1 = cv2.threshold(image, 150, 255, cv2.THRESH_BINARY) # Apply binarization with threshold of 120
cv2.imshow('Thresholded image',thresh1) # Plot the binarized image
cv2.waitKey(0) # Close windows
cv2.destroyAllWindows()
```

Grayscale original image



Binarized image with threshold value of 150

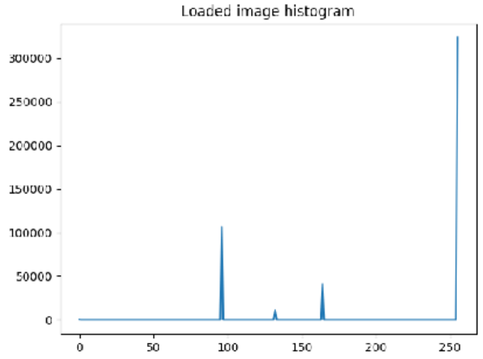
# Basic image processing implementations

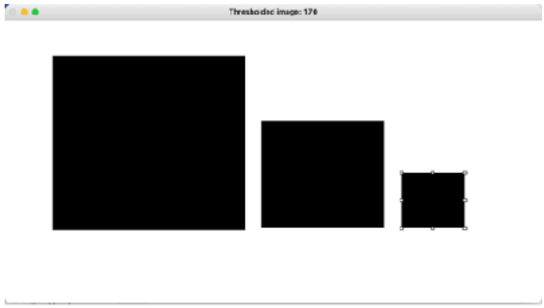
## Thresholding

- Try example using "*simplebinarization.py*":
  - Threshold value: 170

```
import cv2

image = cv2.imread('imgcolor.png',0) #Read image in grayscale
cv2.imshow('Loaded image',image) # Plot grayscale image
ret, thresh1 = cv2.threshold(image, 170, 255, cv2.THRESH_BINARY) # Apply binarization with threshold of 120
cv2.imshow('Thresholded image',thresh1) # Plot the binarized image
cv2.waitKey(0) # Close windows
cv2.destroyAllWindows()
```

Grayscale original image



Binarized image with threshold value of 170

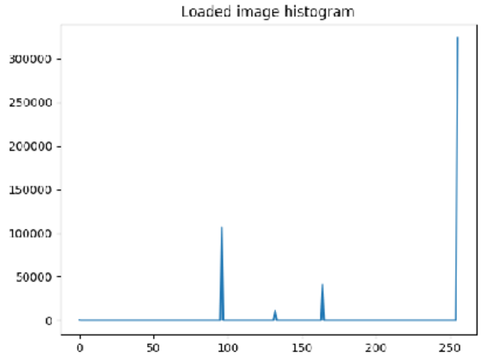
# Basic image processing implementations

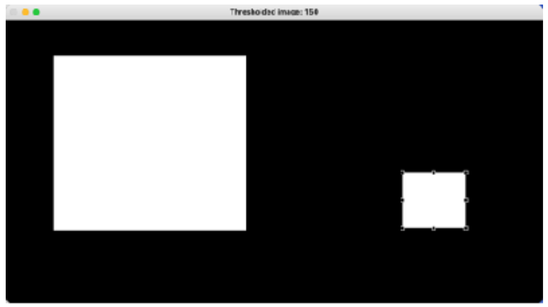
## Thresholding - Inverted Binarization

- Try example using "*simpleinvertedbinarization.py*":

```
import cv2

image = cv2.imread('imgcolor.png',0) #Read image in grayscale
cv2.imshow('Loaded image',image) # Plot grayscale image
ret, thresh1 = cv2.threshold(image, 150, 255, cv2.THRESH_BINARY_INV) # Inverted
cv2.imshow('Thresholded image',thresh1) # Plot the binarized image
cv2.waitKey(0) # Close windows
cv2.destroyAllWindows()
```

Grayscale original image



Inverted binarized image with threshold value of 150

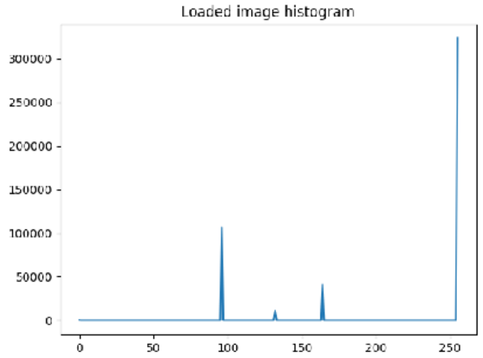
# Basic image processing implementations

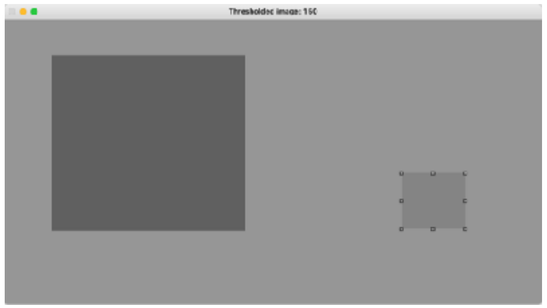
## Thresholding - Inverted Binarization

- Try example using “simpletruncbinarization.py”:

```
import cv2

image = cv2.imread('imgcolor.png',0) #Read image in grayscale
cv2.imshow('Loaded image',image) # Plot grayscale image
ret, thresh1 = cv2.threshold(image, 120, 255, cv2.THRESH_TRUNC) # Inverted
cv2.imshow('Thresholded image',thresh1) # Plot the binarized image
cv2.waitKey(0) # Close windows
cv2.destroyAllWindows()
```

Grayscale original image



Truncanated binarized image with threshold value of 150

# Basic image processing implementations

## Thresholding - Otsu method

- Otsu threshold: Automatic threshold value determination (best used for 2 classes)
- Perform thresholding using *cv2.threshold*() function + *cv2.THRESH_OTSU* flag:
  - Syntax:
    - retotsu, threshimotsu = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)
- thresholdTechnique: type of thresholding applied
  - cv2.THRESH_BINARY+cv2.THRESH_OTSU
- retotsu: threshold value determined by Otsu method
- threshimotsu: binarized image using Otsu method

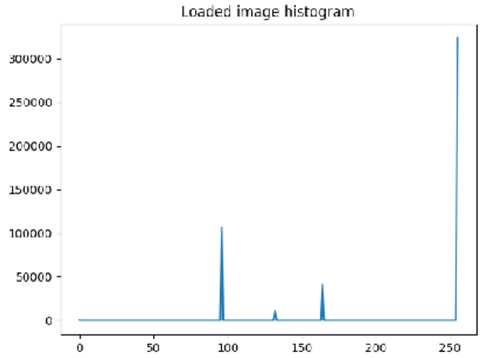
# Basic image processing implementations

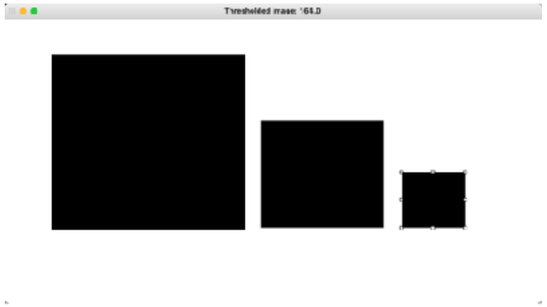
## Thresholding - Otsu method

- Try example using "*otsubinarization.py*":

```
import cv2
image = cv2.imread('imgcolor.png',0) #Read image in grayscale
cv2.imshow('Loaded image',image) # Plot grayscale image
retotsu, threshotsu = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU) # Apply binarization
with Otsu
cv2.imshow(f'Thresholded image: {retotsu}',threshotsu) # Plot the binarized image
print(f'Otsu threshold: {retotsu}')
cv2.waitKey(0) # Close windows
cv2.destroyAllWindows()
```

Grayscale original image



Binarized image using Otsu method

# Basic image processing implementations

## Thresholding - Otsu method

- Try example using “cameraOtsu.py”:

```
import cv2

cap = cv2.VideoCapture(0)
cv2.namedWindow('Webcam frame',cv2.WINDOW_AUTOSIZE)
while True:
    ret_val, frame = cap.read()
    frame = cv2.flip(frame, 1)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    retotsu, threshotsu = cv2.threshold(gray, 0,
255,cv2.THRESH_BINARY + cv2.THRESH_OTSU)  # Apply binarization with
Otsu
    cv2.imshow('Otsu threshold', threshotsu)
    print(f'Otsu threshold: {retotsu}')
    if cv2.waitKey(1) == 27:
        break  # esc to quit
cap.release()
cv2.destroyAllWindows()
```


Otsu method applied directly to the camera feed

# Basic image processing implementations

## Basic image processing outline

- Basic
  - Image reading
  - Image plotting
  - Image properties
  - Image writing
  - Plotting webcam
  - Recording video
  - Playing video
- Histogram
  - What is it?
  - Histogram equalization
  - CLAHE
- Binarization
  - Basic methods
  - Otsu
- **Filtering**
  - Smoothing filters
    - Average
    - Median
    - Gaussian
  - Sharpening filter
  - Edge Detection
    - Sobel
    - Canny

# Basic image processing implementations

## Image filtering

- Filters have multiple applications
  - Smoothing (removing noise)
  - Sharpening
  - Edge detection
  - etc.
- Use kernel to determine new pixel value
  - Kernel can have different size and values (based on the application):

| | | |
|---|---|---|
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |

3x3 window

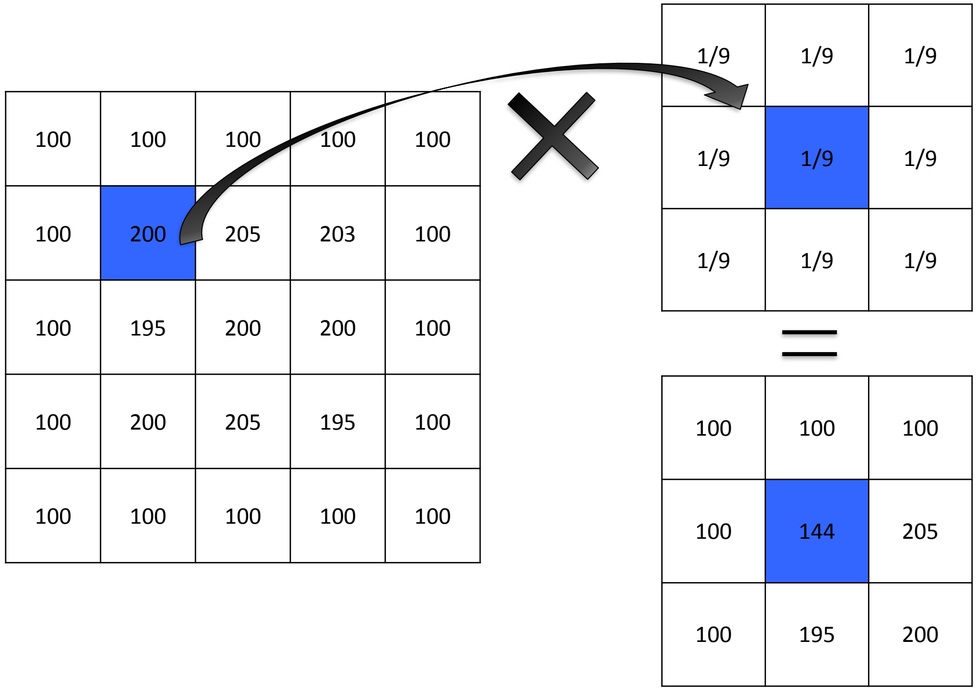
| | | | | |
|---|---|---|---|---|
| 1/15 | 1/15 | 1/15 | 1/15 | 1/15 |
| 1/15 | 1/15 | 1/15 | 1/15 | 1/15 |
| 1/15 | 1/15 | 1/15 | 1/15 | 1/15 |
| 1/15 | 1/15 | 1/15 | 1/15 | 1/15 |
| 1/15 | 1/15 | 1/15 | 1/15 | 1/15 |

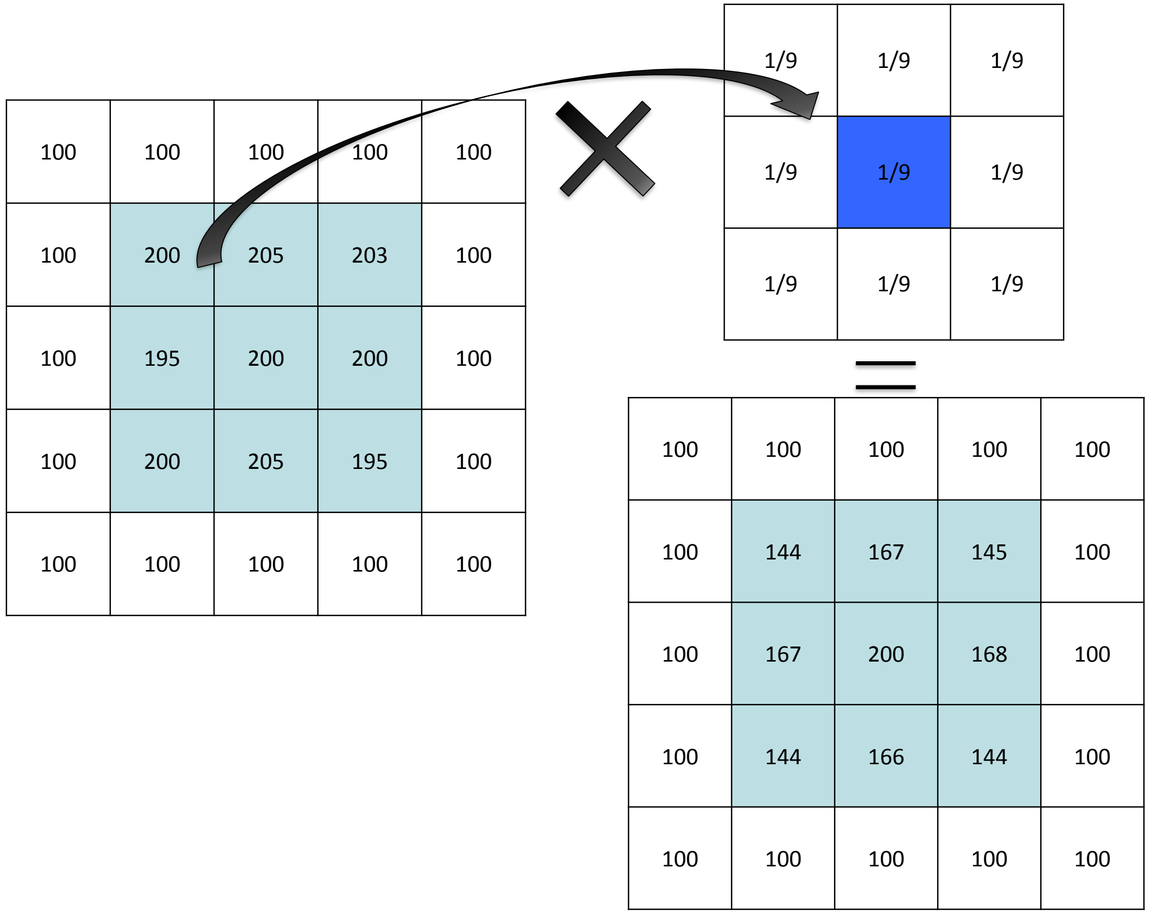
5x5 window

# Basic image processing implementations

## Image filtering

- Using kernel on an image
  - Sliding the kernel on every pixel



# Basic image processing implementations

## Smoothing filter - Average

- Apply average filter using *cv2.blur*() function:
  - Syntax:
    - blurim = cv2.blur(image, kernelsize)
- image: input image
- kernelsize: size of the kernel to apply (3,3), (5,5)…
- blurim:blurred image using average technique and defined kernel

| | | |
|---|---|---|
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |

3x3 window

| | | | | |
|---|---|---|---|---|
| 1/15 | 1/15 | 1/15 | 1/15 | 1/15 |
| 1/15 | 1/15 | 1/15 | 1/15 | 1/15 |
| 1/15 | 1/15 | 1/15 | 1/15 | 1/15 |
| 1/15 | 1/15 | 1/15 | 1/15 | 1/15 |
| 1/15 | 1/15 | 1/15 | 1/15 | 1/15 |

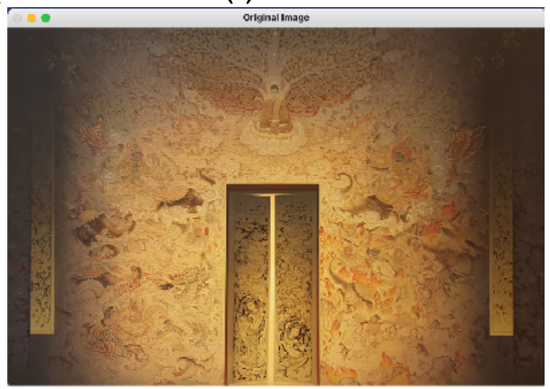
5x5 window

# Basic image processing implementations

## Smoothing filter - Average

- Try example using "*blurimage.py*":

```
import cv2

image = cv2.imread("image.png", cv2.IMREAD_COLOR)
blurim9 = cv2.blur(image, (9,9))
blurim15 = cv2.blur(image, (15,15))
cv2.imshow("Original Image", image)
cv2.imshow("Original Image blurred with 9x9 kernel", blurim9)
cv2.imshow("Original Image blurred with 15x15 kernel", blurim15)
cv2.waitKey(0)
cv2.destroyAllWindows()
```




Original image

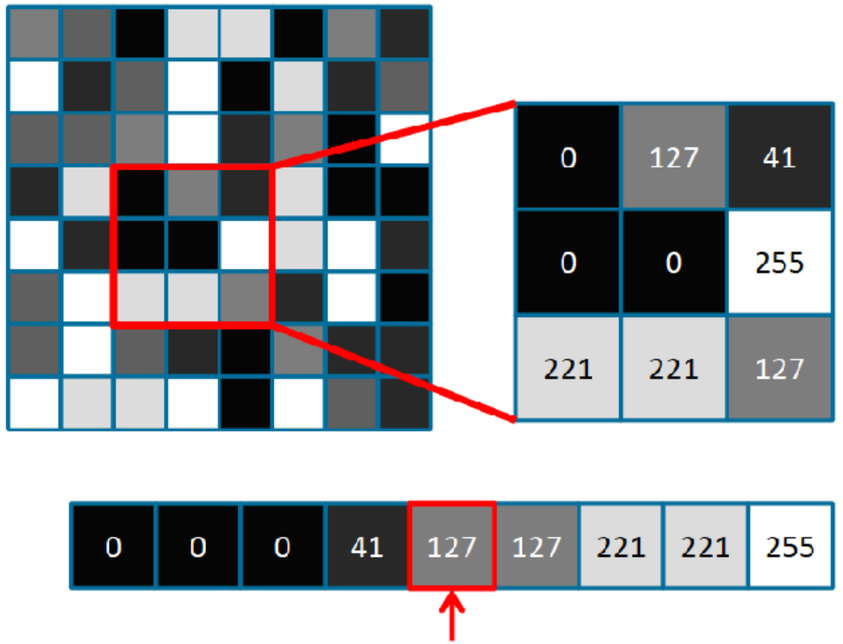

9x9 average filter


15x15 average filter

# Basic image processing implementations

## Smoothing filter - Median

- Median filter: Select the median (middle of sorted windows)
- Apply average filter using *cv2.medianBlur*() function:
  - Syntax:
    - blurim = cv2.medianBlur(image, kernelsize)
- image: input image
- kernelsize: Must be an odd number (3, 5, etc.)
- blurim:blurred image using median technique and defined kernel

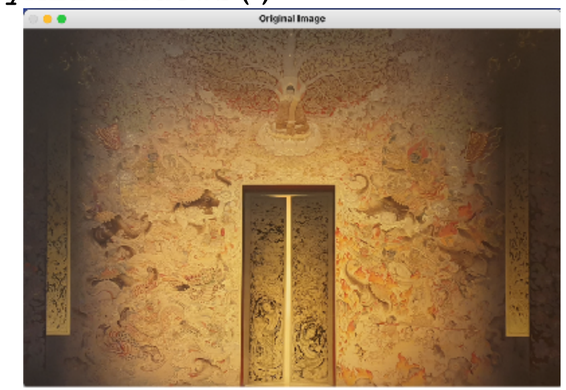


# Basic image processing implementations

## Smoothing filter - Median

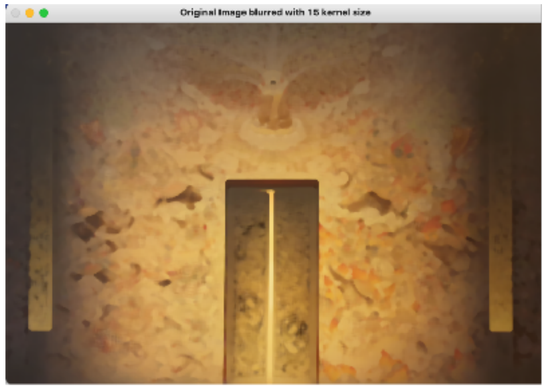
- Try example using "*medianfilter.py*":

```
import cv2

image = cv2.imread("image.png", cv2.IMREAD_COLOR)
blurim9 = cv2.medianblur(image, 5)
blurim15 = cv2.blur(image, 9)
cv2.imshow("Original Image", image)
cv2.imshow("Original Image blurred with 9x9 kernel", blurim9)
cv2.imshow("Original Image blurred with 15x15 kernel", blurim15)
cv2.waitKey(0)
cv2.destroyAllWindows()
```




Original image


Size 9 kernel filter




Size 15 kernel filter

# Basic image processing implementations

## Smoothing filter - Gaussian

- Gaussian filter: Select the median (middle of sorted windows)
- Apply average filter using "*cv2.GaussianBlur*()" function:
  - Syntax:
    - blurim = cv2.GaussianBlur(image, kernelsize[, sigmaX[, dst[, sigmaY[, borderType]]]])
- image: input image
- kernelsize: Gaussian kernel size (width, height) can be different, must be odd
- sigmaX and SigmaY: Standard deviation
- borderType: see border types option in OpenCV documentation
- blurim:blurred image using gaussian technique and defined kernel

1/16

| | | |
|---|---|---|
| 1 | 2 | 1 |
| 2 | 4 | 2 |
| 1 | 2 | 1 |

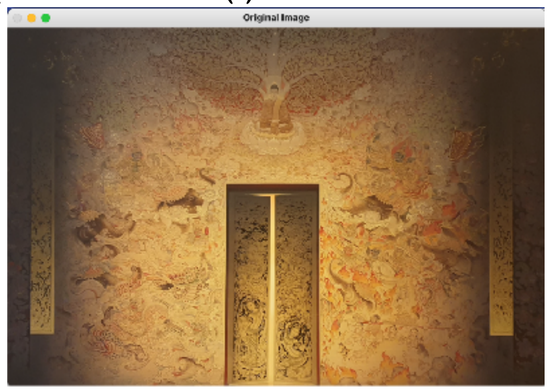
3x3 window

# Basic image processing implementations

## Smoothing filter - Gaussian

- Try example using "*gaussianblur.py*":

```
import cv2

image = cv2.imread("image.png", cv2.IMREAD_COLOR)
gaussian9 = cv2.GaussianBlur(image, (9,9),cv2.BORDER_DEFAULT)
gaussian15 = cv2.GaussianBlur(image, (15,15),cv2.BORDER_DEFAULT)
cv2.imshow("Original Image", image)
cv2.imshow("Original Image blurred with 9x9 kernel size", gaussian9)
cv2.imshow("Original Image blurred with 15x15 kernel size", gaussian15)
cv2.waitKey(0)
cv2.destroyAllWindows()
```




Original image


Size 9x9 kernel filter


Size 15x15 kernel filter

# Basic image processing implementations

## Sharpening filter

- Image sharpening:
  - Goal is to enhance details in a image
  - Highlight change of intensity
  - Different kernels available
  - Applied through convolution

| | | |
|---|---|---|
| 0 | -1 | 0 |
| -1 | 5 | -1 |
| 0 | -1 | 0 |

Laplacian filter invariant to 90° rotation

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 9 | -1 |
| -1 | -1 | -1 |

Laplacian filter invariant to 45° rotations

# Basic image processing implementations

## Sharpening filter

- To apply sharpening filter
- Create a kernel using Numpy:
  - Syntax:
    - kernel = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])
- Apply kernel to image using *filter2D()* function:
  - Syntax:
    - filteredim = cv2.filter2D(image, ddepth, kernel)
- image: original image
- ddepth: desirable depth of the destination image (-1) output depth same as input depth
- kernel: kernel to be applied on the image

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 9 | -1 |
| -1 | -1 | -1 |

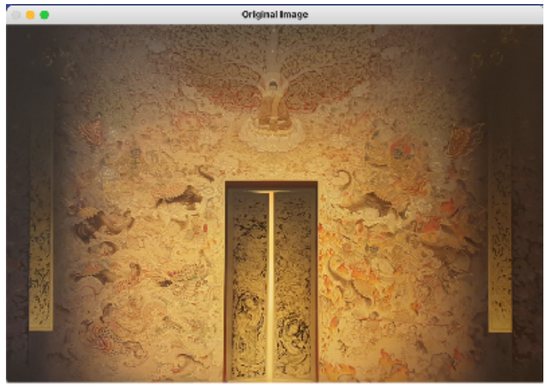
Laplacian filter invariant to 45° rotations

# Basic image processing implementation

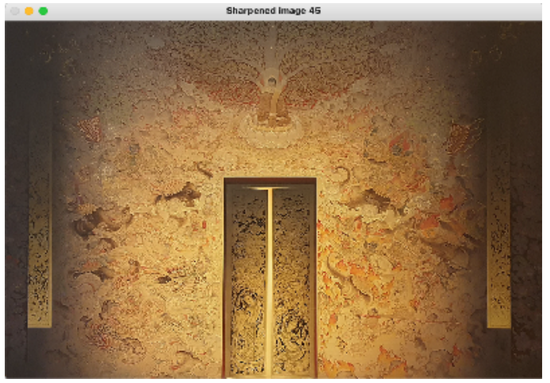
## Sharpening filter

- Try example using "*sharpenfilter.py*":

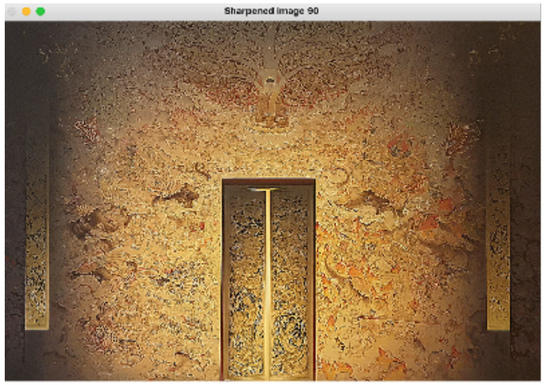
```
import cv2
import numpy as np

image = cv2.imread('image.png')
# Create kernel
kernel45 = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
kernel90 = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])
# Apply  filter
filterimg45 = cv2.filter2D(image, -1, kernel45)
filterimg90 = cv2.filter2D(image, -1, kernel90)
cv2.imshow('Original', image)
cv2.imshow('Sharpened image 45', filterimg45)
cv2.imshow('Sharpened image 90', filterimg90)
cv2.waitKey(0)
cv2.destroyAllWindows()
```




Original image




Sharpen 5




Sharpen 9

# Basic image processing implementations

## Edge detection

- Edge detection:
  - Boundaries are important features in IP
  - Sudden change of pixel intensity
  - Most common methods:
    - Sobel
    - Canny
    - Laplacian

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

Sobel vertical direction kernel

| | | |
|---|---|---|
| 1 | 2 | 1 |
| 0 | 0 | 0 |
| -1 | -2 | -1 |

Sobel horizontal direction kernel

# Basic image processing implementations

## Edge detection - Sobel

- Sobel perform edge detection on X and Y direction
- Merge both using bitwise operation
- $G_x$ and $G_y$ the intensity gradient in x and y direction, $A$ and $B$ the horizontal and vertical kernels

$$G_x = A * I$$

$$G_y = B * I$$

were * is the convolution operator and $I$ the image

- The gradient magnitude G is determined using:

$$G = \sqrt{G_x^2 * G_y^2}$$

- The orientation gradient is determined using:

$$\theta = arctan(G_y/G_x)$$

# Basic image processing implementations

## Edge detection - Sobel

- Detect edges using sobel method with *cv2.Sobel()* function:
  - Syntax:
    - sobelim = cv2.Sobel(image, ddepth, dx, dy, ksize)
- image: grayscale image
- ddepth: desirable depth of the destination image (-1) output depth same as input depth, can use cv2.CV_64F (corresponding to 64bit floating point)
- dx: order of the derivative x
- dy: order of the derivative y
- ksize: kernel size (1, 3, 5 or 7)
- sobelim: output image after sobel method

# Basic image processing implementations

## Edge detection - Sobel

- Try example using "*edgesobel.py*":

```
import cv2

image = cv2.imread('cameraman')
grayim = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
sobelx = cv2.Sobel(grayim, cv2.CV_64F, 1, 0)
sobely = cv2.Sobel(grayim, cv2.CV_64F, 0, 1)
sobelx = cv2.convertScaleAbs(sobelx)
sobely = cv2.convertScaleAbs(sobely)
sobelxy = cv2.bitwise_or(sobelx, sobely)
sobelxy = cv2.convertScaleAbs(sobelxy)
cv2.imshow('Sobel X', sobelx)
cv2.imshow('Sobel Y', sobely)
cv2.imshow('Sobel X Y using Sobel() function', sobelxy)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Original image

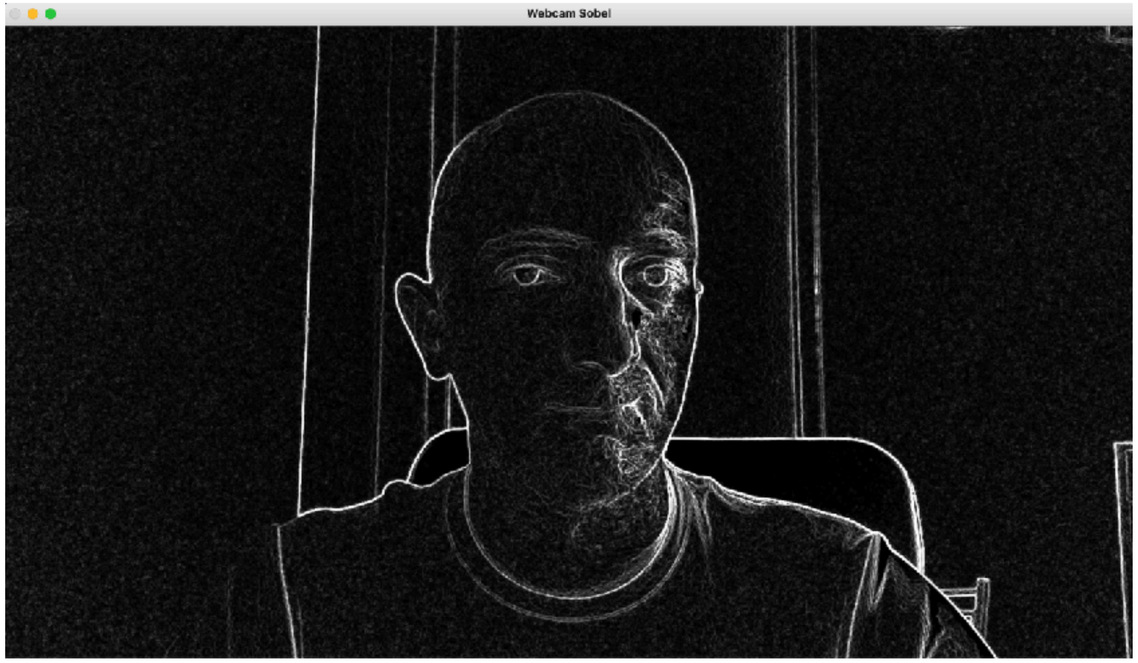

X sobel, Y Sobel, combined sobel

# Basic image processing implementations

## Edge detection - Sobel

- Try example using "*sobelcamera.py*":

```
import cv2
cap = cv2.VideoCapture(0)
cv2.namedWindow('Webcam frame',cv2.WINDOW_AUTOSIZE)
while True:
    ret_val, frame = cap.read()
    frame = cv2.flip(frame, 1)
    grayim = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    sobelx = cv2.Sobel(grayim, cv2.CV_64F, 1, 0)
    sobely = cv2.Sobel(grayim, cv2.CV_64F, 0, 1)
    sobelx = cv2.convertScaleAbs(sobelx)
    sobely = cv2.convertScaleAbs(sobely)
    sobelxy = cv2.bitwise_or(sobelx, sobely)
    sobelxy = cv2.convertScaleAbs(sobelxy)
    cv2.imshow('Webcam Sobel', sobelxy)
    if cv2.waitKey(1) == 27:
        break  # esc to quit
cap.release()
cv2.destroyAllWindows()
```




Sobel applied to camera feed

# Basic image processing implementations

## Edge detection - Canny

- Canny perform edge detection based on multi-stage process:

1. Noise reduction
2. Gradient calculation (Sobel)
3. Non-maximum suppression
4. Double threshold
5. Hysteresis

# Basic image processing implementations

## Edge detection - Canny

- Detect edges using canny method with *cv2.Canny()* function:
  - Syntax:
    - cannyim = cv2.Canny(image, lowerT, upperT[, sobelsize[, L2Gradient]])
- image: grayscale image
- lowerT: Lower threshold value for hysteresis
- upperT: Upper threshold value for hysteresis
- sobelsize: Size of sobel kernel
- L2Gradient: Boolean parameter for more precision
- cannyim: output image after canny method

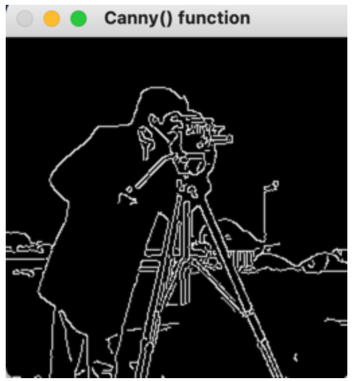
# Basic image processing implementations

## Edge detection - Canny

- Try example using "*edgecanny.py*":

```
import cv2

image = cv2.imread('cameraman.png')
grayim = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blurim = cv2.GaussianBlur(grayim, (3,3), cv2.BORDER_DEFAULT)
lowerT = 75
upperT  =150
cannyim = cv2.Canny(blurim, lowerT, upperT)
cv2.imshow('Canny() function', cannyim)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Original image



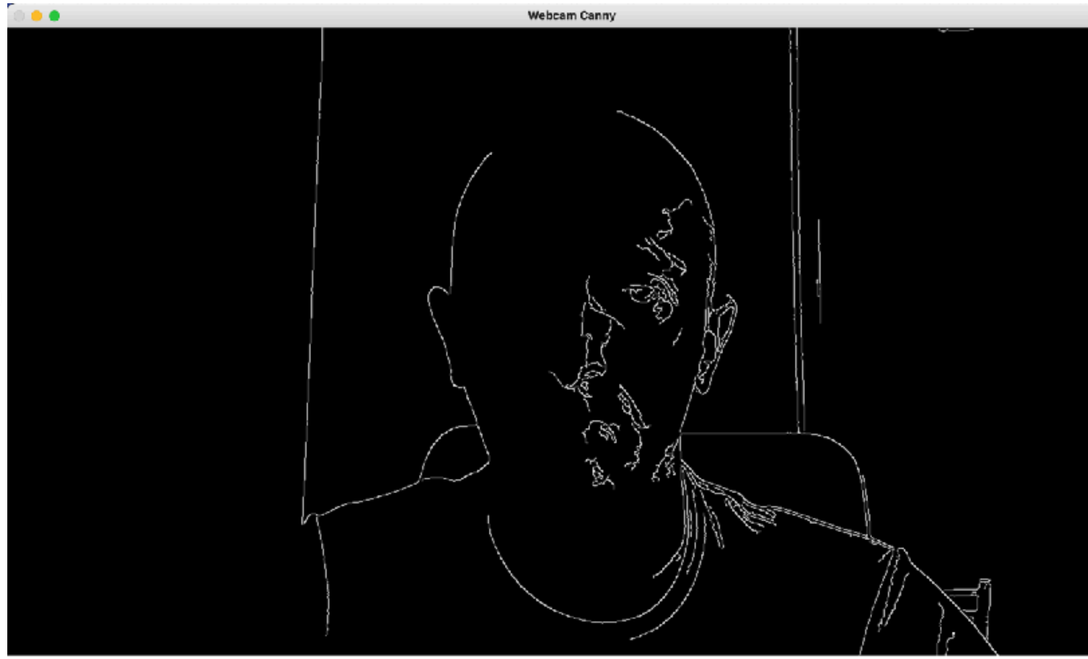

Canny edge

# Basic image processing implementations

## Edge detection - Canny

- Try example using "*cannycamera.py*":

```
import cv2
cap = cv2.VideoCapture(0)
cv2.namedWindow('Webcam frame',cv2.WINDOW_AUTOSIZE)
while True:
    ret_val, frame = cap.read()
    frame = cv2.flip(frame, 1)
    grayim = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blurim = cv2.GaussianBlur(grayim, (3, 3), cv2.BORDER_DEFAULT)
    lowerT = 50
    upperT = 150
    cannyim = cv2.Canny(blurim, lowerT, upperT)
    cv2.imshow('Webcam Canny', cannyim)
    if cv2.waitKey(1) == 27:
        break  # esc to quit
cap.release()
cv2.destroyAllWindows()
```




Canny applied to camera feed

# Outlines

- General introduction
  - Bangkok University - BU-CROCCS
  - What is image processing?
  - Applications
- Python + OpenCV
  - Requirements
- Basic image processing implementations
- **Instagram-like filter implementations**
- Conclusions

# Instagram-like filter implementations

## Why?

- Make your own filters
- Play on different parameters:
  - Saturation
  - Brightness
  - Colour
  - etc.
- Use new OpenCV functions
- Apply directly on video (but easy switch to image)
- Implemented filters:
  - Inverted filter
  - Brightness filter
  - Sepia filter
  - Pencil effect filter

# Instagram-like filter implementations

## Inverted filter

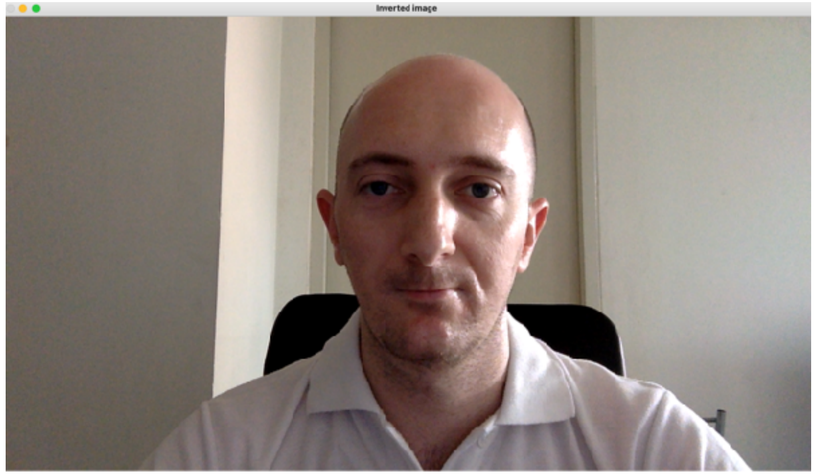
- Inverted filter…invert the pixel values
  - Subtract 255 and get the absolute value of the pixel
- Done using *cv2.bitwise_not()* function:
  - Syntax
    - invertedimage = cv2.bitwise_not(image)
- image: input image
- invertedimage: as the name suggest

# Instagram-like filter implementations

## Inverted filter

- Try example using "*invertedfiltercamera.py*":

```
import cv2
cap = cv2.VideoCapture(0)
cv2.namedWindow('Webcam frame',cv2.WINDOW_AUTOSIZE)
while True:
    ret_val, frame = cap.read()
    frame = cv2.flip(frame, 1)
    invertedimage = cv2.bitwise_not(frame)
    cv2.imshow('Inverted image', invertedimage)
    if cv2.waitKey(1) == 27:
        break  # esc to quit
cap.release()
cv2.destroyAllWindows()
```




Original image


Image after inverted filter

# Instagram-like filter implementations

## Brightness filter

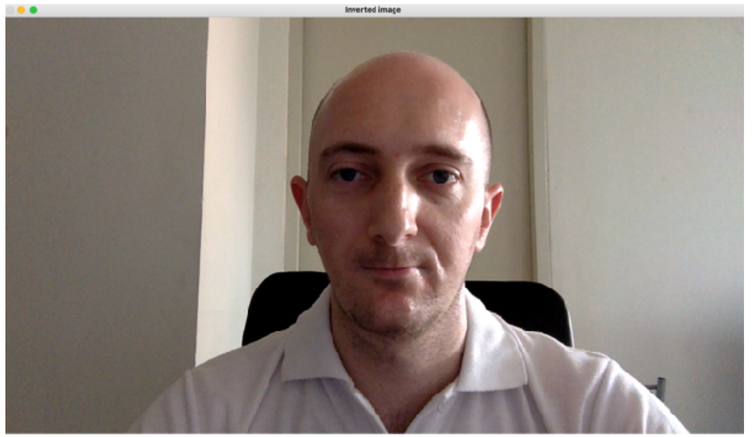
- Brightness filter…changes the brightness values
  - Use *cv2.convertScaleAbs()* function and play with the beta value:
  - Syntax
    - newimage = cv2.convertScaleAbs(image[, alpha= 1[, beta = 0]])
- image: input image
- alpha: (optional: 1),  0<alpha<1 lower contrast, alpha >1 higher contrast
- beta: (optional: 0), -127< beta <127 range of brightness

# Instagram-like filter implementations

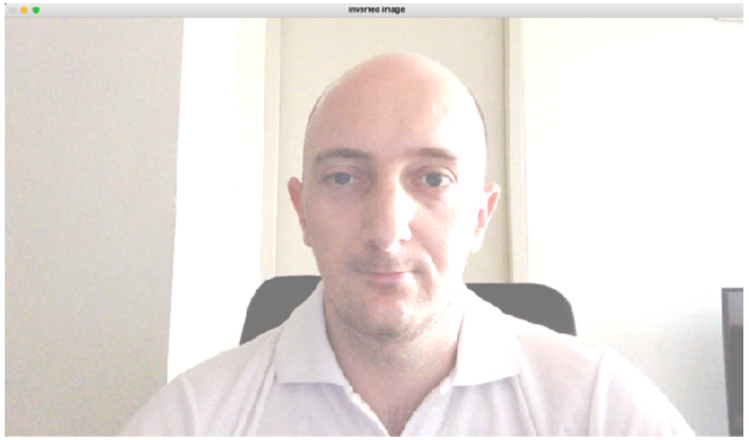
## Brightness filter

- Try example using "*brightnessfiltercamera.py*":

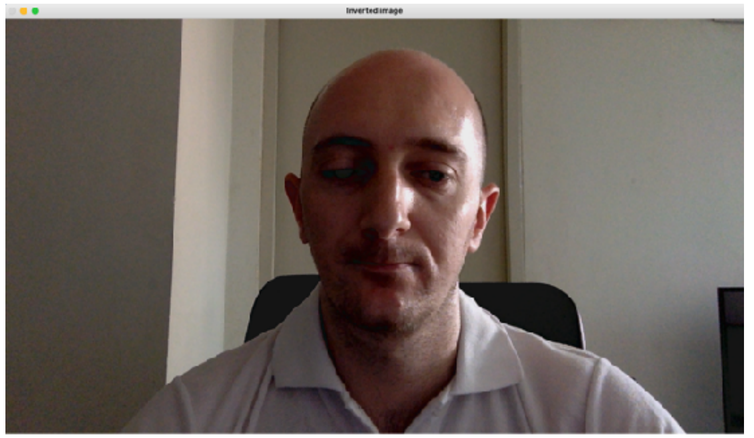
```
import cv2
cap = cv2.VideoCapture(0)
cv2.namedWindow('Webcam frame',cv2.WINDOW_AUTOSIZE)
while True:
    ret_val, frame = cap.read()
    frame = cv2.flip(frame, 1)
    brightimage = cv2.convertScaleAbs(frame, beta = 120)
    cv2.imshow('Inverted image', brightimage)
    if cv2.waitKey(1) == 27:
        break  # esc to quit
cap.release()
cv2.destroyAllWindows()
```




Original image




Beta value: 120




Beta value: -20

# Instagram-like filter implementations

## Sepia filter

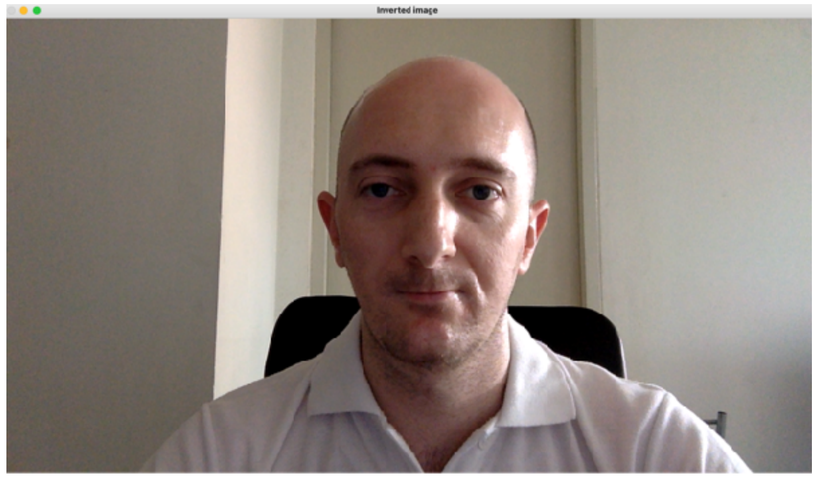
- Sepia filter provides a vintage effect
  - Use *cv2.transform()* function and apply a sepia matrix transformation
  - Sepia Matrix
    - TB = 0.272R + 0.534G + 0.131B
    - TG = 0.349R + 0.686G + 0.168B
    - TR = 0.393R + 0.769G + 0.189B
  - Syntax
    - newimage = cv2.transform(image, matrix)
- image: input image
- matrix: sepia matrix generated using numpy
-

# Instagram-like filter implementations

## Sepia filter

- Try example using "*sepiafiltercamera.py*":

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)
cv2.namedWindow('Webcam frame',cv2.WINDOW_AUTOSIZE)
while True:
    ret_val, frame = cap.read()
    frame = cv2.flip(frame, 1)
    frame64 = np.array(frame, dtype=np.float64)
    sepiamatrix = np.matrix([[0.272, 0.534, 0.131],
                             [0.349, 0.686, 0.168],
                             [0.393, 0.769, 0.189]])
    framesepia = cv2.transform(frame64, sepiamatrix)
    framesepia[np.where(framesepia > 255)] = 255
    framesepia = np.array(framesepia, dtype=np.uint8)
    cv2.imshow('Sepia image', framesepia)
    if cv2.waitKey(1) == 27:
        break  # esc to quit
cap.release()
cv2.destroyAllWindows()
```




Original image


Applied Sepia filter

# Instagram-like filter implementations

## Pencil effect filter

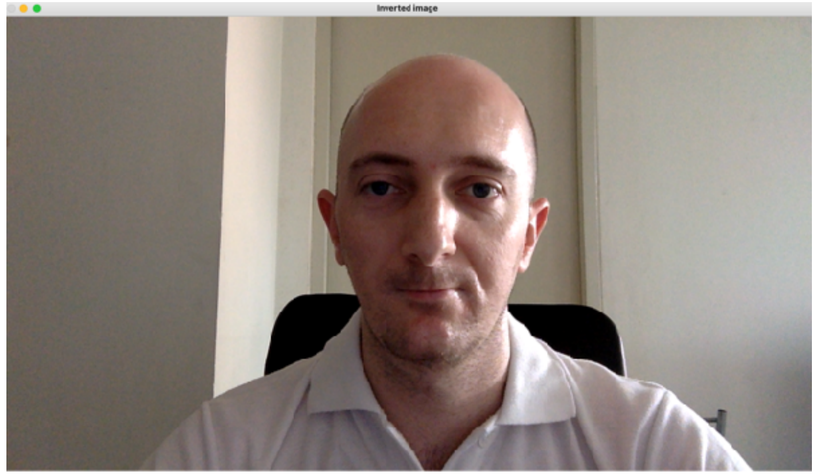
- Convert original image into a pencil drawn one
- Existing function in OpenCV to directly obtain both grayscale and color version
- Use *cv2.pencilSketch()* function:
  - Syntax
    - graypsim, colorpsim = cv2.pencilSketch(image, sigma_s, sigma_r, shade_factor)
- image: input image
- sigma_s: range between 0 to 200 (control the size of the neighborhood for smoothing)
- sigma_r: range between 0 to 1 (control how dissimilar colors are)
- shade_factor: range between 0 to 0.1 (scaling the intensity, the higher the brighter)
- graypsim: grayscale image of pencil-like non-photorealistic line drawing
- colorpsim: grayscale image of pencil-like non-photorealistic line drawing

# Instagram-like filter implementations

## Pencil effect filter

- Try example using "*pencilfiltercamera.py*":

```
import cv2

cap = cv2.VideoCapture(0)
cv2.namedWindow('Webcam frame',cv2.WINDOW_AUTOSIZE)
while True:
    ret_val, frame = cap.read()
    frame = cv2.flip(frame, 1)
    grayspim, colorspim = cv2.pencilSketch(frame, sigma_s=100, sigma_r=0.05, shade_factor=0.05)
    cv2.imshow('Pencil-like image', colorspim)
    if cv2.waitKey(1) == 27:
        break  # esc to quit
cap.release()
cv2.destroyAllWindows()
```




Original image


Pencil-like filter

# Outlines

- General introduction
  - Bangkok University - BU-CROCCS
  - What is image processing?
  - Applications
- Python + OpenCV
  - Requirements
- Basic image processing implementations
- Instagram-like filter implementations
- **Conclusions**

# Conclusion

Quick introduction to basic of image processing

Play and modify the codes as your like

Come to BU for internship !!!

Thank you