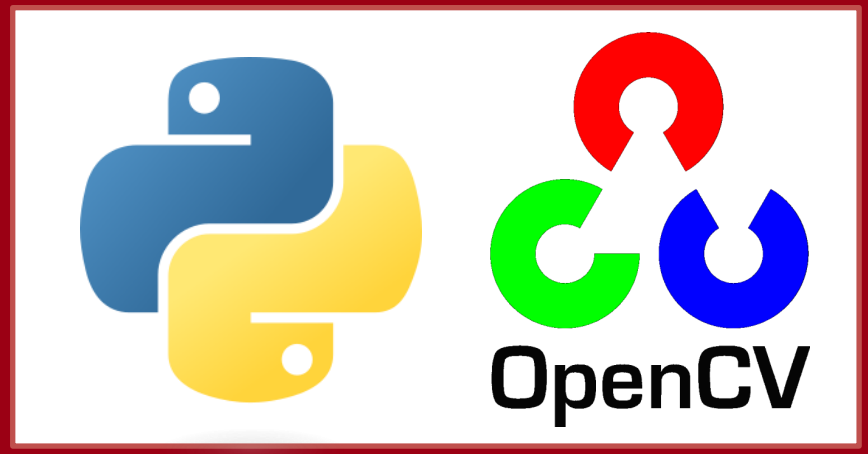
DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

UNIVERSITÀ
DEGLI STUDI
DI PADOVA



LAB 1: Python and OpenCV Intro

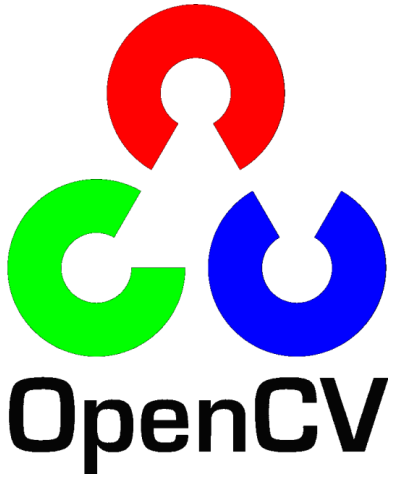
Computer Vision 2023-24

F. Lincetto, P. Zanuttigh

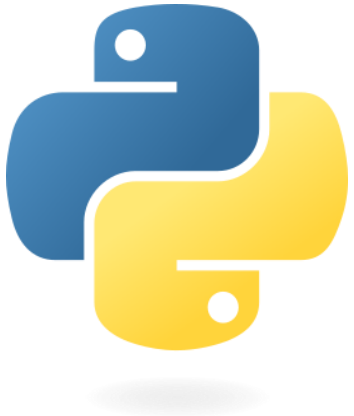
DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Agenda for LAB1

1. Installation tips
2. Introduction to Python
3. The OpenCV library
4. Assigment for LAB1
5. Q&A and support



What is Python?



- Python is a high-level interpreted programming language
- It was created in 1991 by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands
- It's known for its clear syntax and readability that make it a powerful language used by many of the world's leading tech companies

Setup

- To install Python, you are warmly encouraged to use the open-source Anaconda distribution
  - It allows to create and activate different environments
  - This is convenient in order to separate dependencies of different projects
  - *Suggestion:* use the default settings for the installation

- A compact version (miniconda) can be downloaded from https://docs.conda.io/projects/miniconda/en/latest/#latest-miniconda-installer-links
- Follow the official installation instructions specific for your OS https://docs.conda.io/projects/miniconda/en/latest/miniconda-install.html

Create an environment

- Once the installation is complete, let's create our environment
- Open the terminal (Anaconda prompt on Windows) and type the following commands:

```
conda create --name computervision
conda activate computervision
conda install -c conda-forge opencv
conda install -c conda-forge notebook
```

- Every time you open a new terminal you must repeat the `activate` command to enable this environment
- Installing packets on the `base` environment is discouraged

Lab Computers

Some notes for the computers in the labs

Linux

- Python is pre-installed (use the command "`python3`")
- You can use pip to install opencv
- Otherwise install and use miniconda (and then OpenCV through it)

Windows: two options

1. Use the Cygwin64 Terminal (Linux-like environment)
2. Install Miniconda and use Python from the Anaconda prompt
   - Use conda or pip to install Opencv

Jupyter notebook

- **Jupyter Notebook** is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and text
- For the first lab and for the mid-term homework, we will use it
- To launch it, type this command:

```
conda activate computervision
jupyter notebook
```

- After this, a web page should be opened
- You can move to your folder and create or load a new notebook (.ipynb extension)
- Organize your code in different cells and don't forget to comment it
- When finished, remember to save your file!

Producing Software in Python

- Write the source code in a .py file
- Run it

That's it!

- Python is an interpreted language and not a compiled one
  - Anyway, compilation is sometimes automatically performed
  - Python compiles .py code to bytecode stored in .pyc or .pyo
  - Then the interpreter executes the bytecode on a Virtual Machine
- Every step is on-the-fly, the user just needs to run `python myfile.py`

Simple Sum Program

```
1   # Program to display the sum of two integers
2
3   print("Enter the first integer: ")
4   number1 = int(input())
5
6   print("Enter the second integer: ")
7   number2 = int(input())
8
9   sum = number1 + number2
10
11  print(f"The sum is {sum}")
12
```

```
Enter the first integer:
2
Enter the second integer:
4
The sum is 6
```

Some key points about Python

- **Interpreted Language:** the written code is translated to a computer-readable format at runtime
- **Dynamically Typed:** the Python interpreter does automatic data type assignment and management
- **Indentation Syntax:** Python uses whitespace indentation, rather than curly braces or keywords, to delimit blocks
- **Rich Libraries:** Python has a large standard library that is available for use
  - In case you need more packages, **pip** and **anaconda** package managers allows you to install third-party libraries.
  - To import a package, you just need to use the `import` function

```
1    import numpy as np
```

Convenient alias for numpy

Functions in Python

```
13  # Function definition
14  def myfunction(arg1: int, arg2: float = 0.5) -> float:
15
16      sum = arg1 + arg2
17      return sum
18
19
20
21  # Function usage
22  n1 = 2
23  n2 = 0.3
24  mysum = myfunction(n1, n2)
25  mysum = myfunction(arg1 = n1, arg2 = n2)
26  mysum = myfunction(n1)
```

Argument type (not required)

Output type (not required)

Argument default value (not required)

Argument name

Return statement

Standard function call

Verbose function call

Function call using the default value for arg2

- Functions can be defined everywhere in Python
- Functions that don't return anything may not have the return statement

Classes: a Quick Look

- A class declaration is a logical abstraction that defines a new type
- It determines what an object of that type will look like
- An object declaration creates a physical entity of that type
  - An object occupies memory space, a type definition does not
- Each object of a class
  - Shares the same copy of member functions
  - Has its own copy of every variable declared within the class (almost always)

```
33  # Class declaration
34  class MyClass:
35
36      # Constructor and method definitions
37
```

Constructors

- Every object we create will require some sort of initialization
- A class constructor is automatically called each time an object of that class is created
- A constructor function has always the ***same name*** *(__init__)* and has ***no return type***

```
33  # Class declaration
34  class Rectangle:
35
36      # Constructor definition
37      def __init__(self, a: int, b: int):
38
39          self.width = a
40          self.height = b
41
42      # method definitions
43      def area(self) -> int:
44
45          area = self.width * self.height
46          return area
47
```

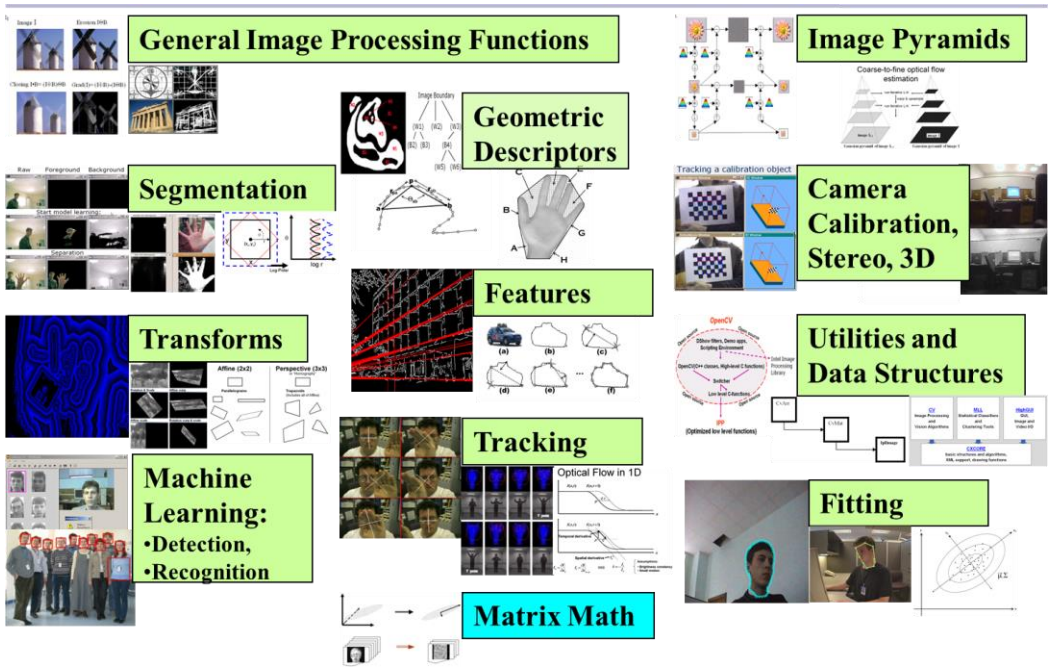
NumPy Array

- NumPy, which stands for Numerical Python, is a library adding support for large, multi-dimensional arrays and matrices
- Numpy supports a variety of data types with various number of bit depth (e.s.: uint8, int8, float16...)
- An array or matrix can be initialized passing a standard python list to the constructor

```
60  import numpy as np
61
62  # Basic constructor
63  array = np.array([0, 1, 2, 3])
64  print("array: ", array)
65  print("array shape", array.shape)
66
67  matrix = np.array([[0, 1, 2], [3, 4, 5]])
68  print("matrix :", matrix)
69  print("matrix shape", matrix.shape)
70
```

```
array:  [0 1 2 3]
array shape (4,)
matrix : [[0 1 2]
 [3 4 5]]
matrix shape (2, 3)
```

The OpenCV Library



- OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision
- The library is cross-platform and free for use under the open-source BSD license
- See the online documentation!

Image as numpy Array

- The primary data structure in OpenCV is the **numpy array**
- It stores images and their components
- Image attributes
  - shape – length, width and number of num_channels (data type: tuple(int, int, int))
    - Possible num_channels → 1: grayscale, 3: BGR, 4: BGR+Alpha (notice order: opencv loads as BGR, not as RGB)
  - dtype: <data_type><bit_depth>
    - data_type can be Unsigned, Signed or Floating points values

```
49  import cv2 as cv
50
51  image = cv.imread('image_path.png')
52  print("Image type: ", type(image))
53
54  height, width, channels = image.shape
55  print("Image shape: ", width, height)
56  print("Image channels: ", channels)
57
58  print("Data type: ", image.dtype)
59
```

```
Image type:  <class 'numpy.ndarray'>
Image shape:  1917 972
Image channels:  3
Data type:  uint8
```

Image as numpy array

```
74  import numpy as np
75
76  height = 1080
77  width = 1920
78
79  # Empty grayscale image
80  image = np.empty(shape=(height, width, 1))
81
82  # Grayscale image initialized to black (16 bit floating point)
83  image = np.zeros(shape=(height, width, 1), dtype=np.float16)
84  # Grayscale image initialized to white (32 bit floating point)
85  image = np.ones(shape=(height, width, 1), dtype=np.float32)
86
87  # RGB image initialized to black (8 bit unsigned int)
88  image = np.zeros(shape=(height, width, 3), dtype=np.uint8)
89  # RGB image initialized to white (16 bit signed int)
90  image = np.ones(shape=(height, width, 3), dtype=np.int16)
91
92  # RGB yellow image (8 bit unsigned)
93  color = np.array([0, 255, 255])
94  image = np.full(shape=(height, width, 3), fill_value=color, dtype=np.uint8)
95
```

N.B.: Images can have pixel values either in range [0, 1] or [0, 255]

Numpy Image Access

```
 99  # Image data access
100  # <array_name>[row_index, column_index, channel_index]
101
102  pixel_i = image[100, 200]
103  channel_green_pixel_i = image[100, 200, 1]
104  print("pixel_i", pixel_i)
105  print("channel_green_pixel_i", channel_green_pixel_i)
106
107  # Image patch access
108  # Patch defined by rows from 100 and 149 included
109  # and columns from 200 and 269 included
110  patch = image[100:150, 200:270]
111  print(patch.shape)
112
```

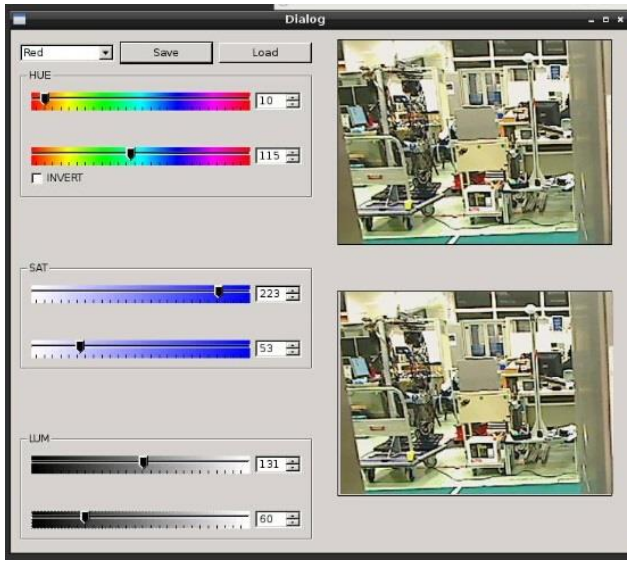
```
pixel_i [  0 255 255]
channel_green_pixel_i 255
patch_shape (50, 70, 3)
```

Numpy Functions

```
160  import numpy as np
161
162  mat_1 = np.ones((4,5))
163  mat_2 = np.ones((4,5))
164  mat_3 = np.ones((5,3))
165
166  # Matrix sum of a scalar value
167  mat_1 = mat_1 + 2
168
169  # Matrix sum element wise
170  out = mat_1 + mat_2
171
172  # Prodcut for a scalar values
173  mat_1 = mat_1 * 3
174
175  # Hadamard product (element-wise)
176  out = mat_1 * mat_2
177
178  # Dot product
179  out = np.dot(mat_1, mat_2)
180
181  # Matrix product
182  out = mat_1 @ mat_3
```

HighGUI Module

- Image I/O, rendering
- Processing keyboard and other events, timeouts
- Trackbars
- Mouse callbacks
- Video I/O




```
114  import cv2 as cv
115
116  # Create new window
117  cv.namedWindow(winname: str, flags: int = cv.WINDOW_AUTOSIZE)
118  # Destroy a window
119  cv.destroyWindows(winname: str)
120  # Show a numpy image
121  cv.imshow(winname: str, image: np.array)
122  # Read an image from file
123  cv.imread(filepath: str)
124  # Write a numpy image to file
125  cv.imwrite(filepath: str, image: np.array)
126
```

Image I/O Example

- OpenCV provides simple and useful ways to read and write images
- Note that there are many extra options to these commands which are available on the documentation
- waitKey(x: int) has two main features
  - if x > 0, then waitKey will wait x milliseconds
  - if x = 0, then waitKey will not move until key is pressed

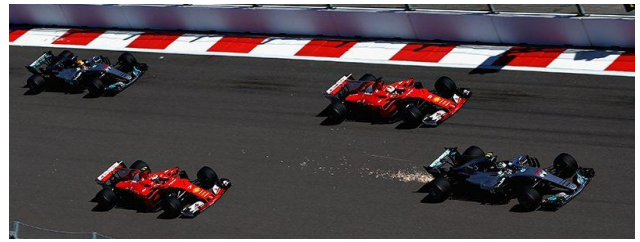
```
127  import cv2 as cv
128
129  # Read an image from a file
130  image = cv.imread("myimage.png")
131
132  # Write an image to a file
133  cv.imwrite("./my_folder/new_image.png", image)
134
135  # Create window for output
136  cv.namedWindow("My Image")
137
138  # Output image to window
139  cv.imshow("My Image", image)
140
141  # Pause program
142  key = cv.waitKey(0)
```

Mouse Callback

```
146  import cv2 as cv
147
148  # Set the callback function for any mouse event
149  # The function my_function will be called when some mouse event happens
150  # You can pass data to the function (e.g., the image), use cast to recover the data
151  cv.setMouseCallback("My window", my_function, function_params)
152
153  # This function is automatically called when a mouse event happens
154  # x,y: coordinates of mouse position, event: type of event, flags: get buttons status
155  def my_function(event: int, x: int, y: int, flags: int, function_params):
156
157      if event == cv.EVENT_LBUTTONDOWN:
158          print(f"Left click-position: {x}, {y}")
```

Assignment

Goal: Change the soccer shirt (or Ferrari cars or unipd flags) color in the image



Write a program that:

Loads one of the images stored inside the data folder (you can use the "*roma.jpg*", "*f1.jpg*" or "*unipd.jpg*" images)

1. Shows the image on a window
2. Captures the left click of the mouse and computes the mean RGB color over a 9x9 neighborhood of the clicked point
3. Segment the target regions by applying a static threshold to the three channels R, G and B (e.g., $\Delta R < 50$, $\Delta G < 50$, $\Delta B < 50$, but try to change the value)
4. Apply a new color to the selected regions (let's use `BGR = (37,201,92)`)

Is the result satisfying?

5. Repeat the task but working with the CIE LAB color space
   - Hint: The luminance depends on illumination, try to segment using only the A and B components

Can you find a smarter way to better segment only the desired the regions?

Example of the Results

Input

RGB

CIELAB
(preserve L)