Image Processing Software

Mikołaj Leszczuk, Zbigniew Hulicki, Andrzej Matiolański

# Introduction

Determining object position in the image, recognizing an object or automatic number plate detection - these are all image processing tasks. The aim of this exercise is to get acquainted with the basics of programming using image processing software.

The exercise will be performed in parallel in various programming environments, to learn about them and compare them.

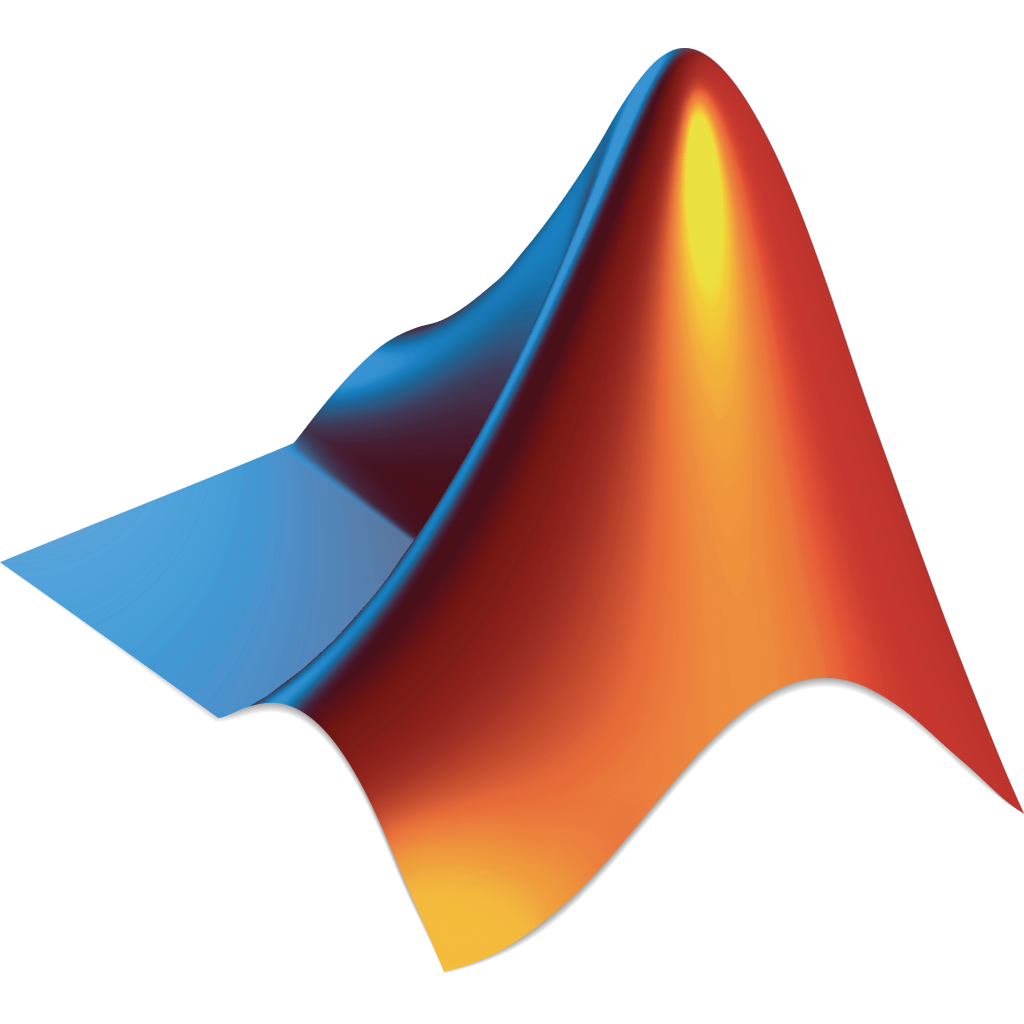
# Setup

Please boot the computer and log in. Choose the OS you are most familiar with. Below are instructions for starting work in the desired environments.

## MATLAB with Image Processing Toolbox

MATLAB is a computer program that is an interactive environment for performing scientific and engineering calculations, and for creating computer simulations. Image Processing Toolbox is a software tool for image processing. Together, they create a very powerful, yet easy to learn environment, which is however relatively resource-consuming and does not allow for easy implementation of the resulting solutions in practice, usually requiring their reimplementation in a different programming language.

MATLAB together with the Image Processing Toolbox should already be configured by default on computers in the laboratory. To start the environment, just click the MATLAB icon:



## Python with OpenCV

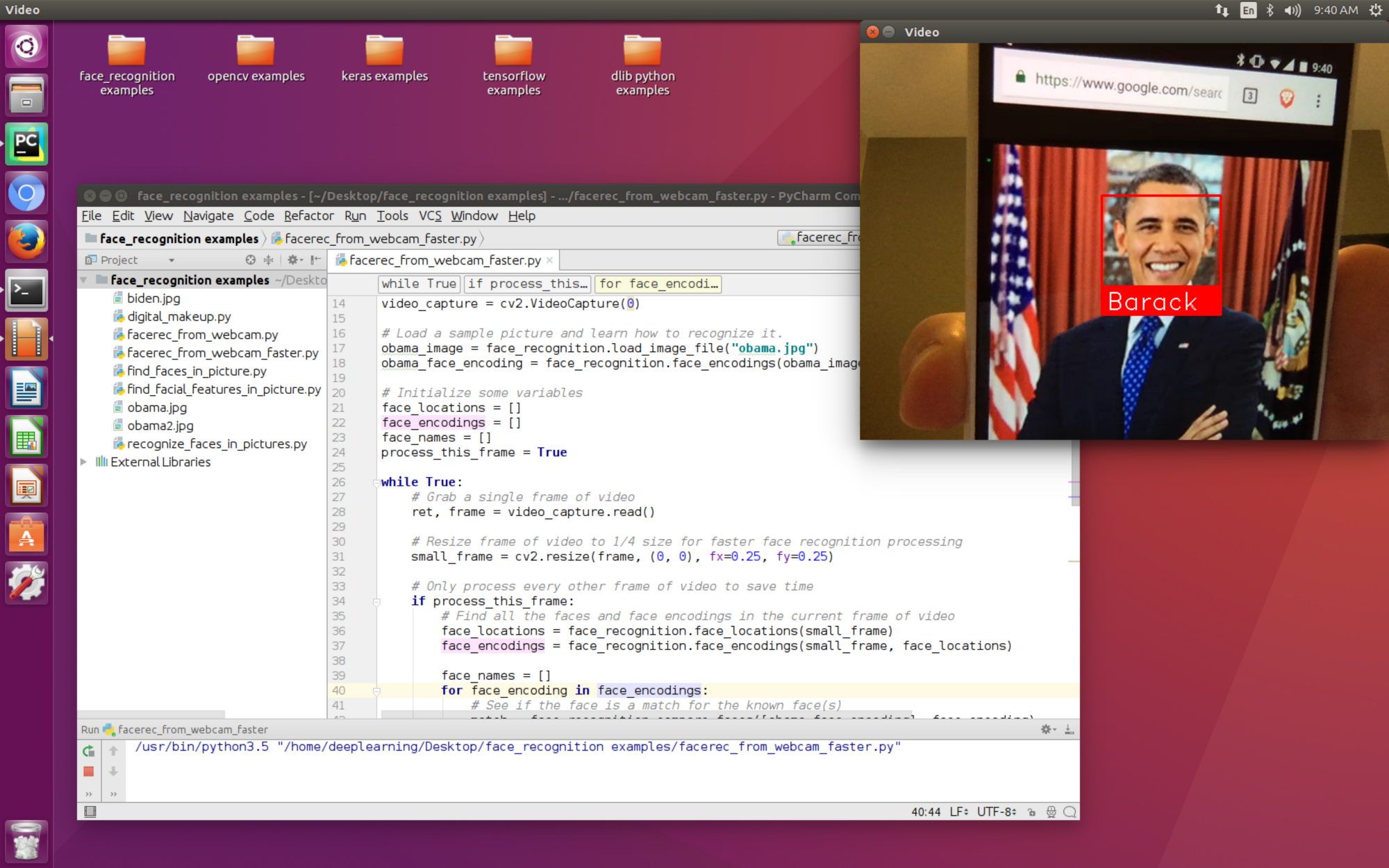
Python is a general-purpose high-level programming language with an extensive standard library package, whose guiding principle is the readability and clarity of the source code. Its syntax is characterized by transparency and brevity. OpenCV is a library of functions used in image processing, based on open-source code and initiated by Intel. This library is cross-platform, it can be used in macOS, Windows and Linux. Its authors focus on real-time image processing. The distinguishing features of this environment from the MATLAB-based environment are, with a little more complex code, less resourcefulness and the possibility of directly implementing the created solutions.

The Python programming language already includes NumPy - a library adding support for large, multi-dimensional image arrays and matrices, along with a large collection of high-level mathematical functions to operate on these image arrays. Nevertheless, the Python environment, along with the OpenCV library, is unfortunately not configured by default on the computers in the lab **(except when it was previously configured as part of some previous exercises)**. Therefore, the easiest way to start working in this environment is to download a VirtualBox computer program, serving as a second type of hypervisor (<https://www.virtualbox.org/wiki/Downloads>), and then, a file with a Deep Learning Virtual Machine (VM) disk image containing preconfigured Python environment with the OpenCV library (<https://archive.org/details/DeepLearningVMForVirtualBox.tar>, faster local copy at the AGH Department of Telecommunications - <http://www.kt.agh.edu.pl/~miklesz/DeepLearningVM_for_VirtualBox.tar.gz>).

Then, just to start the environment, just click the VirtualBox icon, and after running this application, start the virtual machine:



Detailed instructions for launching the Deep Learning VM can be found at: <https://medium.com/@ageitgey/try-deep-learning-in-python-now-with-a-fully-pre-configured-vm-1d97d4c3e9b>. After correct configuration, the screen should look like a pre-configured Ubuntu virtual machine:



Tip 1: If you feel that the virtual machine graphics are working too slowly, enable: Settings/Display/3D Acceleration.

Tip 2: Please note that if you use Python, there might be two independently configured versions installed on the machine: Python 2 (**python**) and Python 3 (**python3**).

Tip 3: To start using OpenCV in a Python script, you must make the following imports:

**import numpy as np**

**import cv2**

# Tasks

Please do the following tasks, trying to compliment the code independently in **each** programming environment. Probably for each of the environments, it is enough to create one file with the source code, which will be gradually expanded.

## Reading Images

Read an image (any image). The image should be in the working directory, or a full path of the image should be given.

## Displaying Images

Display an image in a window, without creating the display window in advance (thus, allowing the environment to create the window automatically, on-demand).

## Waiting for Key Press

Waits for any keyboard event. Continue the program, if one presses any key.

## Closing/Destroying Windows

Destroy the window you created.

## Creating Windows

Modify your source code in order to create a window **before** you display an image in it. You may want to wait for a key button press just after creating the window, in order to see it.

## Converting to Grayscale

Modify your source code in order to convert your image into grayscale before displaying it.

## Playing Videos from File

Capture a video from a file and display it in the loop. Check the end of the video by checking if frames are read correctly. Also, while displaying the frames, use appropriate pause/wait time between frames (if it is too less, the video will be very fast and if it is too high, the video will be slow).

**Please be aware that not all video formats and codecs are readable in all environments. This exercise was tested at least with the MPEG-4 Part 14 (MP4) format, using the programming environments as described in the introduction to this exercise.**

## Saving Videos

Save the captured video frame-by-frame, using the Motion JPEG (MJPG) codec and AVI format.

## Accessing and Modifying Pixel Values

Access a true-color image pixel value by its row **100** and column **100** coordinates. Access the pixel but only blue value. Modify the same pixel values to **[255, 255, 255]**. Modify the only red value of **[10, 10]** to **100**.

## Accessing Image Properties

Image properties include several rows, columns and channels, type of image data, number of pixels etc.

Access and return a shape of the image being a tuple of several rows, columns and channels (if the image is color).

Access and return the total number of pixels.

Obtain and return the image datatype.

## Obtaining Image Regions of Interests (ROI)

Obtain image Region of Interest (ROI) using indexing. Selecting an object and copy it to another region in the image.

**Please note that starting from this exercise, you will not find the appropriate functions in the talk slides. Independent studying of the documentation will be necessary.**

## Splitting and Merging Image Channels

Split the color image to single channels. Join these individual channels to a color image.

Set all the red channel pixels to zero.

## Making Borders for Images (Padding)

Create a border around the image, something like a photo frame.

## Mirroring Images

Perform a mirror image reflection.

# Additional References

* “MATLAB Documentation”, <https://www.mathworks.com/help/matlab>
* “OpenCV documentation index”, <https://docs.opencv.org>