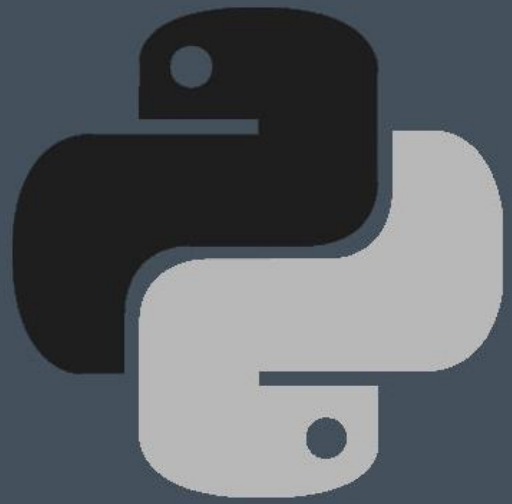
AIE
ACADEMY OF INNOVATIVE EDUCATION

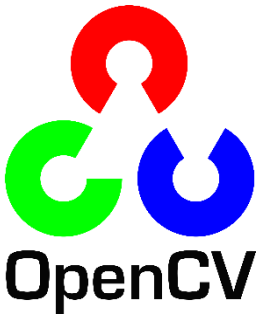


DAY 02

IMAGE PROCESSING with OpenCV

Thakshila Dasun
BSc. Eng in Mechatronics Eng
CIMA (UK)

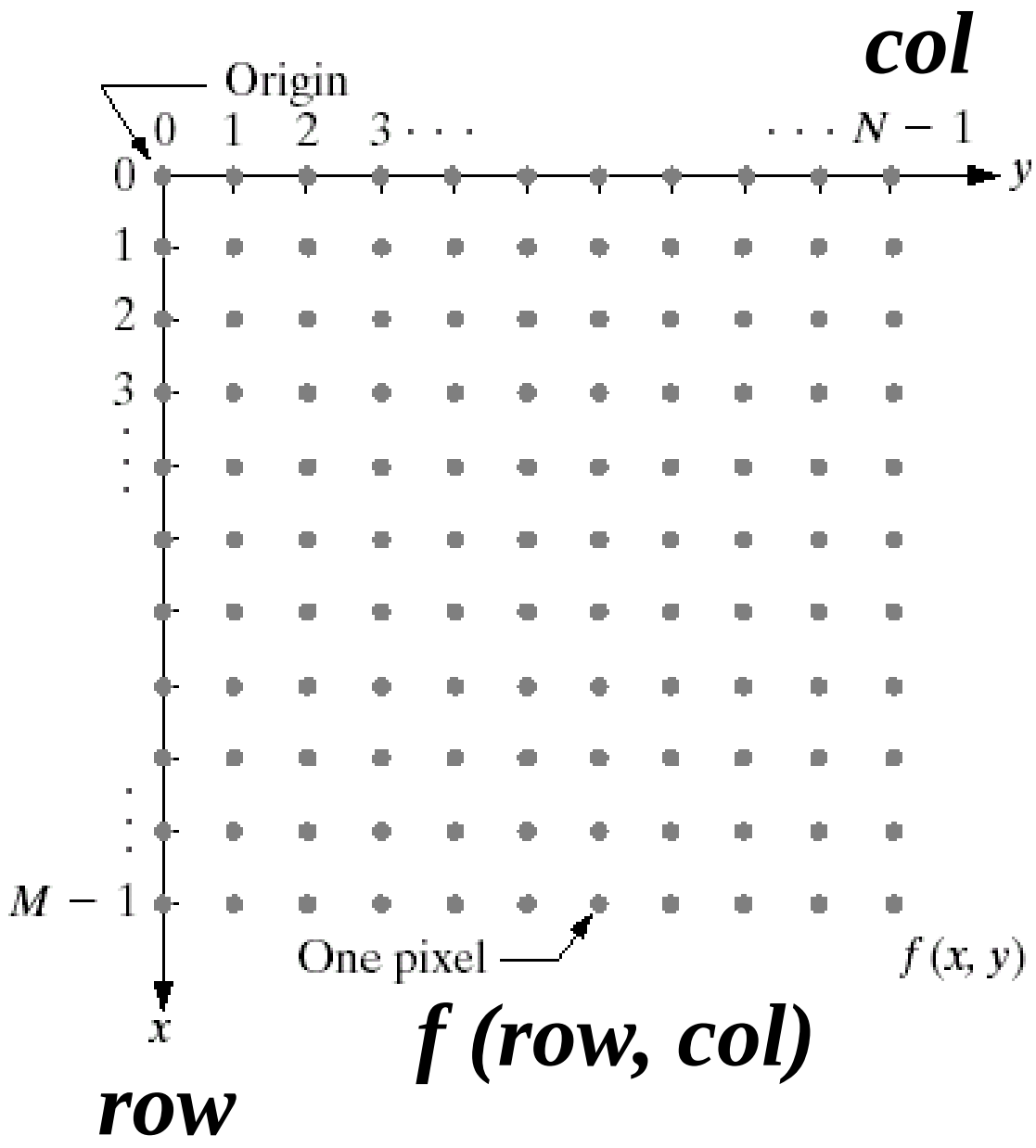
INTRODUCTION TO OPENCV



- OpenCV, which is an image and video processing library with bindings in C++, C, Python, and Java
- Used for all sorts of image and video analysis, like facial recognition and detection, license plate reading, photo editing, advanced robotic vision, optical character recognition, and a whole lot more.

Image Representation

- A digital image is composed of M rows and N columns of pixels each storing a value
- Pixel values are most often grey levels in the range 0-255(black-white) or R,G,B values
- images can easily be represented as matrices



loading Images

- Img to be cv2.read(image file, parms). The default is going to be IMREAD_COLOR, which is color without any alpha channel.
- For IMREAD_COLOR simple numbers can also be used, as -1, 0, or 1. Color is 1, grayscale is 0, and the unchanged is -1. Thus, for grayscale

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('dog.jpg',cv2.IMREAD_GRAYSCALE)
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Draw on Images

- You can simply use pyplot module to draw on images.
- There are advanced methods for drawing on images provided by OpenCv, they will be discussed later.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('watch.jpg',cv2.IMREAD_GRAYSCALE)
plt.imshow(img, cmap = 'gray')
plt.xticks([]), plt.yticks([]) # to hide tick values on X and Y axis
plt.plot([200,300,400],[100,200,300],'c', linewidth=5)
plt.show()
```

Loading Video Source (1)

- Handling frames from a video is identical to handling for images

```
cap = cv2.VideoCapture(0)
```

- This will return video from the first webcam on your computer. 0 is the default Camera (webcam), you can use 1,2 etc if you have webcams connected.

```
while(True):
    ret, frame = cap.read()
```

- This code initiates an infinite loop (to be broken later by a break statement), where we have ret and frame being defined as the cap.read(). Basically, ret is a boolean regarding whether or not there was a return at all, at the frame is each frame that is returned.

Loading Video Source (2)

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

- Defining a new variable, gray, as the frame, converted to gray. OpenCV use Blue Green Red where other applications use RGB

```
cv2.imshow('frame',gray)
```

- Despite being a video stream, we still use imshow. One frame at a time in a high frame rate.

```
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
```

- if we get a key, and that key is a q, we will exit the while loop with a break, which then runs:

```
cap.release()
cv2.destroyAllWindows()
```

Recording Video

```
import numpy as np
import cv2
cap = cv2.VideoCapture(1)
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi',fourcc, 20.0, (640,480))
while(True):
        ret, frame = cap.read()
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        out.write(frame)
        cv2.imshow('frame',gray)
        if cv2.waitKey(1) & 0xFF == ord('q'):
                break
cap.release()
out.release()
cv2.destroyAllWindows()
```

NOTE: A **video codec** is an electronic circuit or software that compresses or decompresses digital **video**. It converts uncompressed **video** to a compressed format or vice versa

Drawing and Writing on Image (1)

```
cv2.line(img,(0,0),(150,150),(255,255,255),15)
```

- The cv2.line() takes : **where, start coordinates, end coordinates, color (bgr), line thickness.**

```
cv2.rectangle(img,(15,25),(200,150),(0,0,255),15)
```

- The parameters : **image, the top left coordinate, bottom right coordinate, color, and line thickness.**

```
cv2.circle(img,(100,63), 55, (0,255,0), -1)
```

- The parameters: **image/frame, the center of the circle, the radius, color, and then thickness.**

NOTE: When we put -1 for thickness, object will actually be filled in

Drawing and Writing on Image (2)

```
pts = np.array([[10,5],[20,30],[70,20],[50,10]], np.int32
cv2.polylines(img, [pts], True, (0,255,255), 3)
```

- pts, short for points, as a numpy array of coordinates. Then, we use cv2.polylines to draw the lines. The parameters are as follows: where is the object being drawn to, the coordinates, should we "connect" the final and starting dot, the color, and again the thickness.

```
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(img,'OpenCV Tuts!',(0,130), font, 1, (200,255,155), 2, cv2.LINE_AA)
```

•**img** – Image.
•**text** – Text string to be drawn.
•**org** – Bottom-left corner of the text string in the image.
•**font** – CvFont structure initialized using InitFont().
•**fontFace** – Font type. **fontScale** – Font scale factor that is multiplied by the font-specific base size.
•**color** – Text color.
•**thickness** – Thickness of the lines used to draw a text.
•**lineType** – Line type. See the line for details.
•**bottomLeftOrigin** – When true, the image data origin is at the bottom-left corner. Otherwise, it is at the top-left corner.

Open CV supported Fonts

- **CV_FONT_HERSHEY_SIMPLEX** normal size sans-serif font
- **CV_FONT_HERSHEY_PLAIN** small size sans-serif font
- **CV_FONT_HERSHEY_DUPLEX** normal size sans-serif font (more complex than CV_FONT_HERSHEY_SIMPLEX)
- **CV_FONT_HERSHEY_COMPLEX** normal size serif font
- **CV_FONT_HERSHEY_TRIPLEX** normal size serif font (more complex than CV_FONT_HERSHEY_COMPLEX)
- **CV_FONT_HERSHEY_COMPLEX_SMALL** smaller version of CV_FONT_HERSHEY_COMPLEX
- **CV_FONT_HERSHEY_SCRIPT_SIMPLEX** hand-writing style font
- **CV_FONT_HERSHEY_SCRIPT_COMPLEX** more complex variant of CV_FONT_HERSHEY_SCRIPT_SIMPLEX

Resize Images

cv.**Resize(src, dst, interpolation=CV_INTER_LINEAR)**

```
small = cv2.resize(image, (0,0), fx=0.5, fy=0.5)
```

```
resized_image = cv2.resize(image, (100, 50))
```

```
small = scipy.misc.imresize(image, 0.5)
```

- **src** – input image.
- **dst** – output image; it has the size dsize (when it is non-zero) or the size computed from src.size(), fx, and fy; the type of dst is the same as of src.
- **dsize** –output image size; if it equals zero, it is computed as:
- Either dsize or both fx and fy must be non-zero.
- **fx** –scale factor along the horizontal axis; when it equals 0, it is computed as
- **fy** –scale factor along the vertical axis; when it equals 0, it is computed as
- **interpolation** –interpolation method:
  - **INTER_NEAREST** - a nearest-neighbor interpolation
  - **INTER_LINEAR** - a bilinear interpolation (used by default)
  - **INTER_AREA** - resampling using pixel area relation. It may be a preferred method for image decimation, as it gives moire'-free results. But when the image is zoomed, it is similar to the INTER_NEAREST method.
  - **INTER_CUBIC** - a bicubic interpolation over 4x4 pixel neighborhood
  - **INTER_LANCZOS4** - a Lanczos interpolation over 8x8 pixel neighborhood

Color-spaces

- There are more than 150 color-space conversion methods available in OpenCV. But we will look into only two which are most widely used ones, BGR to Gray and BGR to HSV.
- For color conversion, we use the function cv2.cvtColor(input_image, flag) where flag determines the type of conversion.
- For BGR right to Gray conversion we use the flags cv2.COLOR_BGR2GRAY. Similarly for BGR to HSV, we use the flag cv2.COLOR_BGR2HSV.

HSV Colors

- **HSV** (**Hue, Saturation, Value**) alternative representations of the RGB color mode