UNIVERSITY OF Southampton

Introduction to Open CV

Jose Miguel Garro, Shaokang Zhang

Advanced Computational Modelling 2 - Workshop

University of Southampton

13th March 2018

UNIVERSITY OF Southampton

Open CV? What is it?

- Computer vision and machine learning library.
- Contains more than 2500 optimized algorithms.
  - Basic modifications on images and videos.
  - Histograms, rotations, translations, etc.
  - Feature detection, description and tracking.
  - Thousands of more options more.
- Workshop focused to introduce OpenCV computer vision basics to the people.
- Let's kick off the Workshop, we have a lot of things to cover!

Safety checks

Open **ipython** on the terminal and type:

- $ import cv2
- $ print cv2.__version__

If version displayed is equal or bigger than 3.1.0 everything is perfect.

Now, get the last version of the repository files with the following commands:

- $ cd $HOME
- $ cd opencv-feeg6003
- $ hg pull
- $ hg update

UNIVERSITY OF Southampton

Index

- 1st part - Basics
  - Managing Images
  - Managing Videos
  - Basic information and modifications
- 2nd part – Different type of modifications and feature detection
  - Changing colorspaces
  - Masks
  - Transformation matrix
  - Contours
  - Histograms

UNIVERSITY OF Southampton

Managing Images

- Objectives:
  - Read an image from a file.
  - Show the image (or the modificated) to see the results.
  - Write the modified image to sabe the results.

Managing Images - Read

- Action required to get all the image information within the code to apply the necessary algorithms on it.
- OpenCV provides the function **cv2.imread()**
  - Two input arguments
    - File path in single quotes
    - Color flag:
      - 0 – Grayscale
      - 1 – Color
      - -1 – Unchanged
  - One output containing the matrix of pixels

Managing Images - Display

- Optional command to check if the algorithm or modifications performed are working in real time.
- OpenCV contains the function **cv2.imshow()** to do so.
  - Only takes two input arguments
    - Name of the window
    - Matrix of pixels of the desired image to show
  - No output arguments

Managing Images - Write

- Save the final result of the image after applying all the necessary algorithms.
- OpenCV gives the possibility using **cv2.imwrite()**
  - Two input arguments
    - Path and filename
    - Matrix of pixels to be stored
  - No output arguments

Managing Images - Task

Task: Read/Display and Write Images

- $ cd ~/tasks
- $ nano task1.py

Remind: import cv2 – to access to the library (use an alias if preferred)

- Image = cv2.imread('filename',color_flag)
- cv2.imshow('windowname', source)
- cv2.imwrite('filename',source)

Managing Images

```
#Load an color image
colorimg = cv2.imread('atlantis.jpg',1)
# Load an image in grayscale mode.
grayimg = cv2.imread('atlantis.jpg',0)
# Display the color image using the specific function for it.
cv2.imshow('Color', colorimg)
# Display the grayscale image afore loaded
cv2.imshow('Gray', grayimg)
# Save the grayscale image into a file named: 'grayimag.jpg'
cv2.imwrite('grayimage.jpg', grayimg)
```

Managing Videos

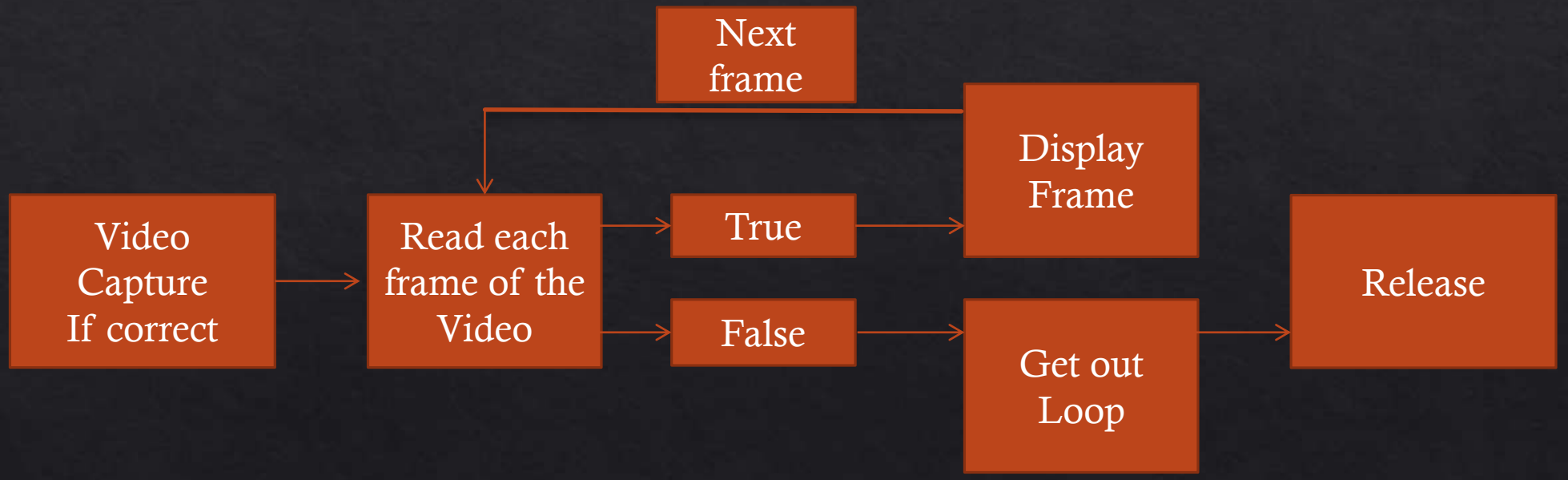
- Objectives:
  - Capture a video from a file.
  - Reproduce and modify videos.
  - Write a modified video in the current directory.

Managing Videos – Capture Video

- Key to obtain the video information within the code and perform the necessary algorithms.
- cv2.**VideoCapture()** allows to get either the video from an external camera (also WebCam) or from a file.
  - One input argument:
    - Integers to grab the video from an external device
    - Filename and its path to obtain the information from a pre-stored file.
  - One output:
    - Contains all the video frames

Managing Videos – Display

- Show the video using OpenCV library, it allows to show results if desired in live.
- Steps to follow:



Managing Videos – Display
Check if the capture is correct

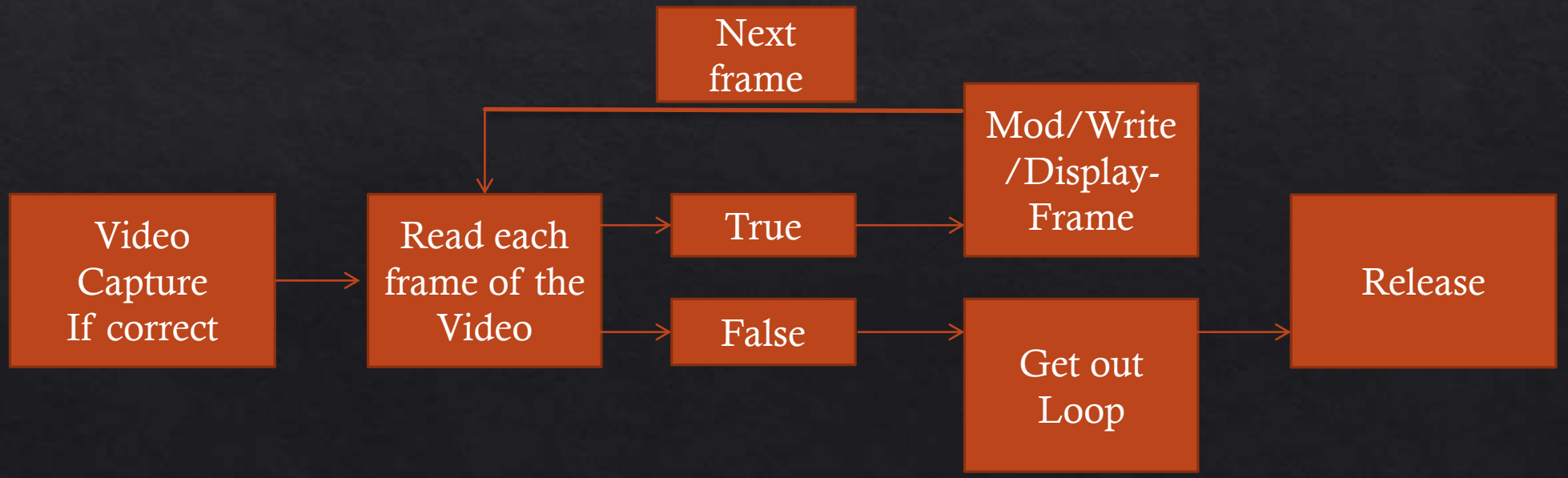
- To check if the capture has been done properly OpenCV provides **isOpened()** for it, this function has to be called from the variable that contains the video information.
- This functions returns a Boolean:
  - True if the video is correct
  - False if the video is corrupt or the code cannot access to it

Managing Videos – Display
Read frames inside a loop

- Once the video has been captured it is time to read frame by frame the video. To do so, the **read()** function is available for the video information variable (the output rom the VideoCapture()).
  - Two outputs:
    - 1st Boolean informing if frame is correct (True) or not (False).
    - 2nd Matrix of pixels of the frame (as it was an image)
- If the frame is True, it means that video has not come to an end, hence the display is performed similarly as it was done for the images.
- If the frame is False, the video has come to an end and the code gets out of the while loop to release the video and close all the windows.

Managing Videos - Write

- Store the results of the modified video as a new video file.
- The procedure is similar to the past section but instead of showing, the main idea is to perform the modifications and then write the new video. It is important to say that modifications, writing and displaying can be done all together.



Managing Videos - Write

- Writing is trickier than it was for images because a couple of more things are needed to do it.
- First, OpenCV requires to set up an object call **cv2.VideoWriter()**.
  - Takes four input arguments:
    - 1st Filename with the corresponding format. Recommended **.avi**
    - 2nd Video code, use recommended by OpenCV **cv2.VideoWriter_fourcc(*'XVID)**
    - 3rd Frame per second (recommended 25ms)
    - 4th Frame size
    - 5th Color flag, If **True** the object writer expects a color video. Otherwise it expects a grayscale video.

Managing Videos - Task

Task: Capture/Display and Write a video.

- $ cd ~/tasks
- $ nano task2.py

Remind:

- video = cv2.VideoCapture('filename')
- cv2.imshow('windowname', source)
- videowriter.write(source) – VideoWriter object provided in the template

Managing Videos

```
vid = cv2.VideoCapture('video.avi')
vwo = cv2.VideoWriter('newvideo.avi', fourcc, 25, (int(vid.get(3)),int(vid.get(4))), False)
# Check if video has been opened properly
while(vid.isOpened()):
        # Read video frames
        ret, frame = vid.read()
        # Check if frame is True
        if ret:
                #Modifications
                # Write and show
                vwo.write(gra)
                cv2.imshow('GenuineVideo', frame)
                cv2.waitKey(25)
        else:
                break
#Release stuff
```

Basic operations

- Objectives:
  - Get shape information of the images/videos.
  - Access to pixels value
  - Access to a specific value
  - Modify pixels value

Basic Operations – Image/Video shape

- Useful to know the number of rows and columns of the matrix provided. In addition it informs about the number of channels of the image.
- OpenCV let us to know this information by means of **shape** on the image/video matrix**.**
  - It returns 3 outputs:
    - First, number of rows of the matrix
    - Second, number of columns of the matrix
    - Third, if the matrix is an color image the third output is going to be 3. Otherwise, if the image is gray scaled, there is not third output.

Basic Operations – Pixel information

- Once, the number of rows and columns is known it is possible to access to the values of an specific pixel. In this case the images are defined with the BGR type (blue, green and red values), so the information obtained from a pixel is the blue, green and red value of the pixel.
- To access to the information the only needed thing is to specify the row and the column desired, as a normal matrix in python.
- Example:
  - Pixel = Matrix[row,column]

Basic Operations – Pixel modification

- Not only it is possible to access to the pixel information but also its value can be modified.
- To do the modification it is only required to select a pixel and specify the value of blue, green and red to set up as a vector.
- Example:
  - For an color image:
    - Matrix[row,column] = [blue,green,red]
  - For a grayscale image:
    - Matrix[row,column] = [intensity]
  - Select a region of a matrix – Image[20:30, 50:60]

Basic Operation - Task

Task: Access to matrix information and modify it

- $ cd ~/tasks
- $ nano task3.py

Remind: BGR type of matrix (Blue, Green and Red color)

- image.shape – 3 outputs if image is in color.
- Access to an specific value of a matrix in Python - Image[row,column]
- Select a region of a matrix – Image[20:30, 50:60]

Basic Operations

```
#Get row/columns/channels from an image or a frame
print colorimg.shape
#Access to a pixel information
pixel = colorimg[50,80]
#Modify a set of pixels
colorimg[20:30, 20:50] = [10,10,10]
#Extra –
b,g,r = cv2.split(colorimg)
New = cv2.merge((b,g,r))
```

Changing Colorspaces

Images information in OpenCV are stored as matrix:

- Pixel position(x,y)
- Color channel(Colorspace)

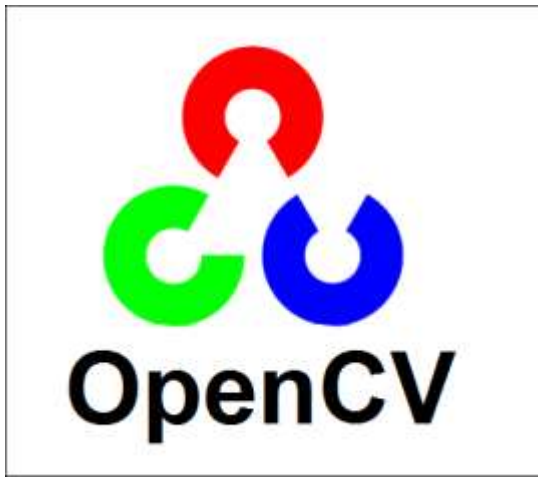
Color mode: BGR, HSV, GRAY

- BGR: mixing blue, green and red
- HSV: Hue(Color), Saturation(Grayscale), Value(Brightness)
- GRAY: Black-and-white mode

Changing Colorspaces

- Function to change colorspaces:
  cv.cvtColor(input_image,flag)
  - input_image: source image
  - flag: **cv.COLOR_BGR2HSV** or **COLOR_BGR2GRAY**
  - get other flags by:

```
import cv2 as cv
flags = [i for i in dir(cv) if i.startswith('COLOR_')]
print(flags)
```



Changing Colorspaces - Task

Task: Convert colorspaces from BGR to GRAY

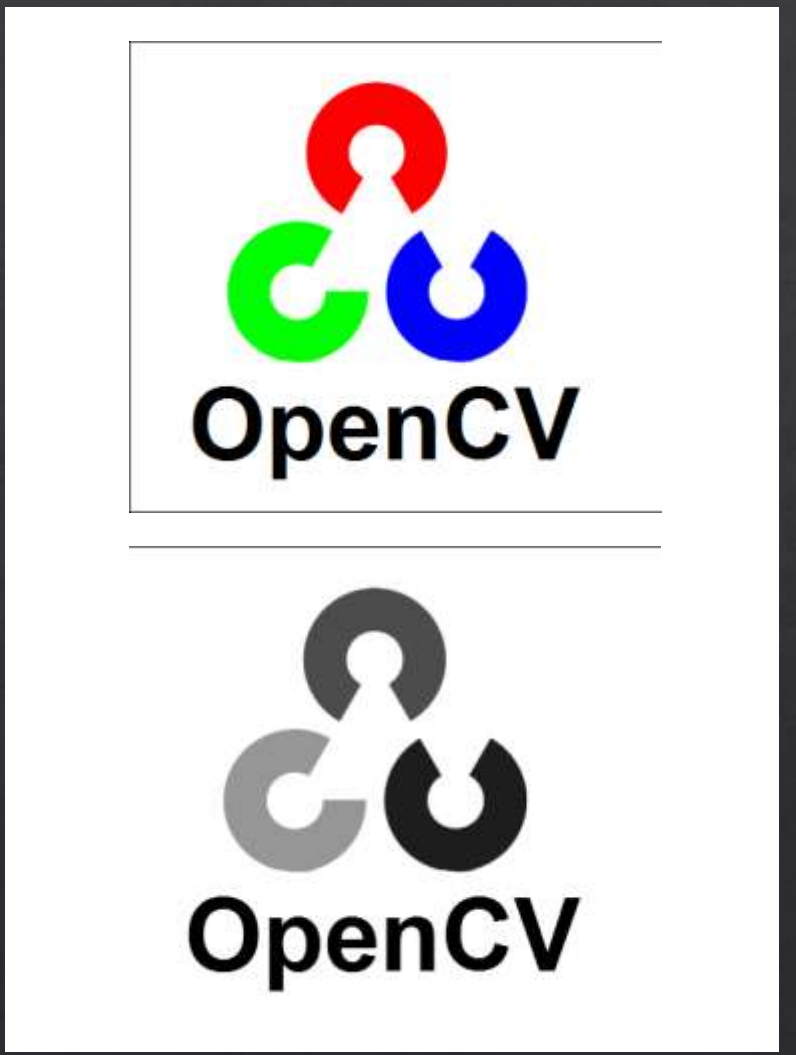
- $ cd ~/tasks
- $ nano task4.py

Remind: : **cv.cvtColor(input_image,flag)**

- input_image: source image
- flag: **cv.COLOR_BGR2HSV** or **COLOR_BGR2GRAY**

Changing Colorspaces

```
import cv2 as cv
image = cv.imread('logo.png')
image_Gray=cv.cvtColor(image,cv.COLOR_BGR2GRAY)
cv.imshow('BGR',image)
cv.imshow('Gray',image_Gray)
cv.waitKey(0)
cv.destroyAllWindows()
```



Extracting Objects by Color

- **Objective:**

  Read a image and detect the object with certain color using HSV mode

- **Why use HSV mode?**
  - Easier to get and modify color in HSV mode (only Hue represents the color)
  - Get HSV value of blue:

```
In:   blue = np.uint8([[[255,0,0]]])
      hsv_blue = cv.cvtColor(blue,cv.COLOR_BRG2HSV)
      print(hsv_blue)

Out:  [[[120, 255, 255]]]
```

Extracting Objects by Color

- Steps:
  - Convert the picture from BGR to HSV
  - Threshold the HSV picture(Get a mask)
  - Extract objects

- Functions used:
  - Threshold: **cv.inRange(image_input, lower_boundary, upper_boundary)**

    Check if every element lies in the range: set it to be 255 if it is and to be 0 if not.
  - Extract: **cv.bitwise_and(img, img, mask = mask)**

    Get product of each element of two images bit-wisely. Here we use it to mask images, so keep the first two parameters the same.

Extracting Objects by Color - Task

Task: Extract blue color object from the OpenCV logo

- $ cd ~/tasks
- $ nano task5.py

Remind:

- Threshold: **cv.inRange(image_input, lower_boundary, upper_boundary) -** Check if every element lies in the range: set it to be 255 if it is and to be 0 if not.
- Extract: **cv.bitwise_and(img, img, mask = mask)**
- Get product of each element of two images bit-wisely. Here we use it to mask images, so keep the first two parameters the same.
- HSV value of **color blue**: [120, 255, 255]

Extracting Objects by Color

```
image = cv.imread('logo.png')
image_hsv = cv.cvtColor(image,cv.COLOR_BGR2HSV)
lower_blue = np.array([110,255,255])
upper_blue = np.array([130,255,255])
mask = cv.inRange(image_hsv,lower_blue,upper_blue)
result = cv.bitwise_and(image,image,mask = mask)
cv.imshow('image', image)
cv.imshow('mask', mask)
cv.imshow('result', result)
cv.waitKey(0)
cv.destroyAllWindows()
```

Source

Mask

Result

Geometric Transformation

- Objectives:
  - Scale, Translate and Rotate

- Algorithm:
  - Position $P = \begin{bmatrix} x \\ y \end{bmatrix}$
  - Transformation matrix $M = \begin{bmatrix} m1 & m2 & m3 \\ m4 & m5 & m6 \end{bmatrix}$
  - Result $Q = \begin{bmatrix} m1 \cdot x + m2 \cdot y + m3 \\ m4 \cdot x + m5 \cdot y + m6 \end{bmatrix}$

Scaling

- Scaling: resizing pictures

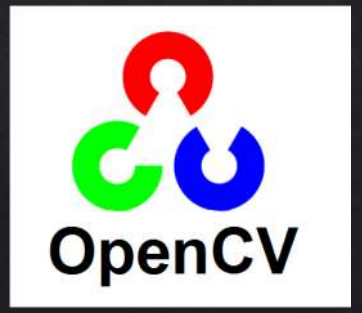
$$M = \begin{bmatrix} fx & 0 & 0 \\ 0 & fy & 0 \end{bmatrix}$$

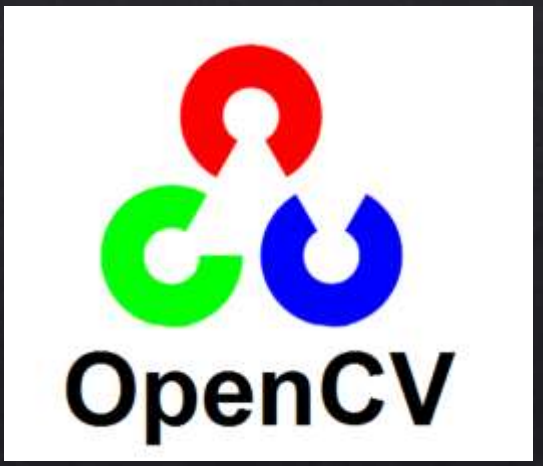
- Function: **cv.resize(img, None, fx, fy, interpolation method)**
- Number of pixels fixed, need interpolation.
  - Methods:

cv.INTER_AREA: for shrinking

cv.INTER_CUBIC: for zooming (slow)

cv.INTER_LINEAR: for zooming (default method)





Scaling - Task

Task: scaling an image with fx = 1.5, fy =1.5

- $ cd ~/tasks
- $ nano task6.py

Remind: **cv.resize(img, None, fx, fy, interpolation method)**

- **Interpolation Methods**:
  - **cv.INTER_AREA**: for shrinking
  - **cv.INTER_CUBIC**: for zooming (slow)
  - **cv.INTER_LINEAR**: for zooming (default method)

Scaling

```
import numpy as np
import cv2 as cv
img = cv.imread('logo.png’)
res = cv.resize(img,None,fx=1.5,fy=1.5,interpolation = cv.INTER_CUBIC)
cv.imshow('Scaling',res)
cv.waitKey(0)
cv.destroyAllwindows()

#Alternative way
height, width = img.shape[:2]
res = cv.resize(img,(1.5*width,1.5*height),interpolation = cv.INTER_CUBIC)
```

Translation

- Translation: move every pixel in the image

$$M = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \end{bmatrix}$$

- To perform algorithm: **cv.warpAffine(img, M, area)**
  - **area:** display area with the form of **(width, height)**

- Steps:
  - Construct transformation matrix M
  - Pass M to **cv.warpAffine()**

Translation - Task

Task: translate an image with tx = 100, ty =50

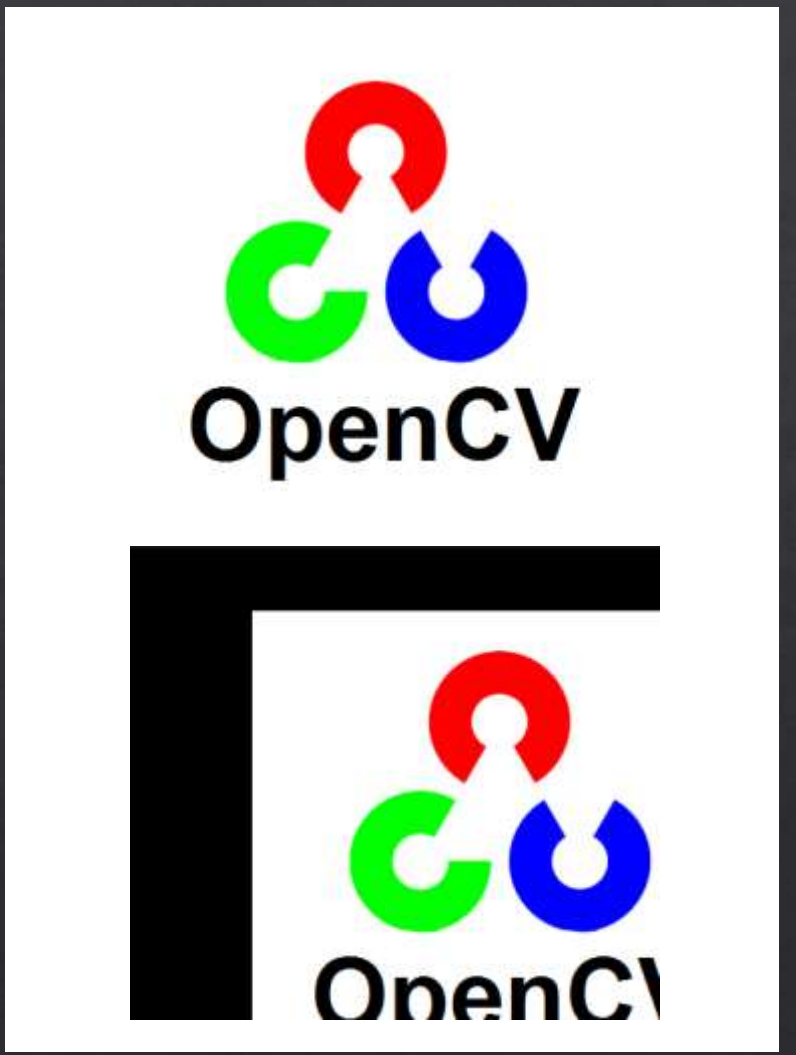
- $ cd ~/tasks
- $ nano task7.py

Steps:

- Construct transformation matrix M
- Pass M to **cv.warpAffine(img, M, area)**
- **area**: display area with the form of **(width, height)**

Translation

```
import numpy as np
import cv2 as cv
img = cv.imread('logo.png')
rows, cols = img.shape[:2]
M = np.float32([[1,0,100],[0,1,50]])
image_new = cv.warpAffine(img,M,(cols,rows))
cv.imshow('Translation',image_new)
cv.waitKey(0)
cv.destroyAllwindows()
```



Rotation

- Objective: Rotate with some certain point

A. Rotate with original center

$$M = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \end{bmatrix}$$

B. Rotate with adjustable center
  - Translate from original point to rotation center
  - Rotate with center
  - Translate backward

- How to get the rotation matrix: **cv.getRotationMatrix2D(rotation center, angle, scale)**

Rotation - Task

Task: rotate an image with original center, angle 90°, scale 1

- $ cd ~/tasks
- $ nano task8.py

Steps:

- Get transformation matrix M using **cv.getRotationMatrix2D(rotation center, angle, scale)**
- Pass M to **cv.warpAffine(img, M, area)**

Rotation

```
import numpy as np
import cv2 as cv
img = cv.imread('logo.png')
rows, cols = img.shape[:2]
M = cv.getRotationMatrix2D((cols/2,rows/2),90,1)
res = cv.warpAffine(img,M,(cols,rows))
cv.imshow('Rotation',res)
cv.waitKey(0)
cv.destroyAllwindows()
```

UNIVERSITY OF Southampton

Contours

- What is contour?
  - Curves along the boundary
  - Given by lists of points in OpenCV
- Main functions:
  - cv.findContours()
  - cv.drawContours()

Find Contours

- **Function: cv.findContours(img,mode,method)**
  - **img**: source image
  - **mode**: contour retrieval mode(use **cv.RETR_TREE** for now, need a binary source image)
  - **method**: contour approximation method
- For the approximation methods, two commonly used methods:
  - **cv.CHAIN_APPROX_NONE**: returns all points
  - **cv.CHAIN_APPROX_SIMPLE**: returns less points
- Use **cv.inRange()** to convert image from GRAY to binary
  - **cv.inRange(image_input, lower_boundary, upper_boundary)**

Find Contours

- **Function: cv.findContours(img,cv.RETR_TREE,method)**
  - Output: **im2, contours, hierarchy = cv.findContours(img_binary,cv.RETR_TREE,method)**
    - **im2:** an binary image
    - **contours:** lists of points
    - **hierarchy:** rank of contours (follow some rules)

Draw Contours

- **Function: cv.drawContours(img,contours,index,(color),thickness)**:
  - **img**: source image
  - **contours**: lists of contour points
  - **index**: index of contour(**-1** to draw all contours or other number to draw certain contour)
  - **color**: color in BGR mode
  - **thickness**: thickness of contours
- Draw contours on the source image

Contours

- Task: find and draw contours
- Steps:
  - Read images and convert to GRAY mode(easier to convert to binary image)
  - Convert to binary image
  - Find contours
  - Draw contours and show

Contours - Task

Task: find and draw contours

- $ cd ~/tasks
- $ nano task9.py

Remind:

- Get a binary image: **cv.inRange(img_gray, lower_boundary, upper_boundary)**
- **im2, contours, hierarchy = cv.findContours(img_binary,cv.RETR_TREE,method)**
  - methods: **cv.CHAIN_APPROX_NONE, cv.CHAIN_APPROX_SIMPLE**
- **cv.drawContours(img,contours,index,(color),thickness)**
- Index = -1 to draw all contours, color given in BGR mode with brackets

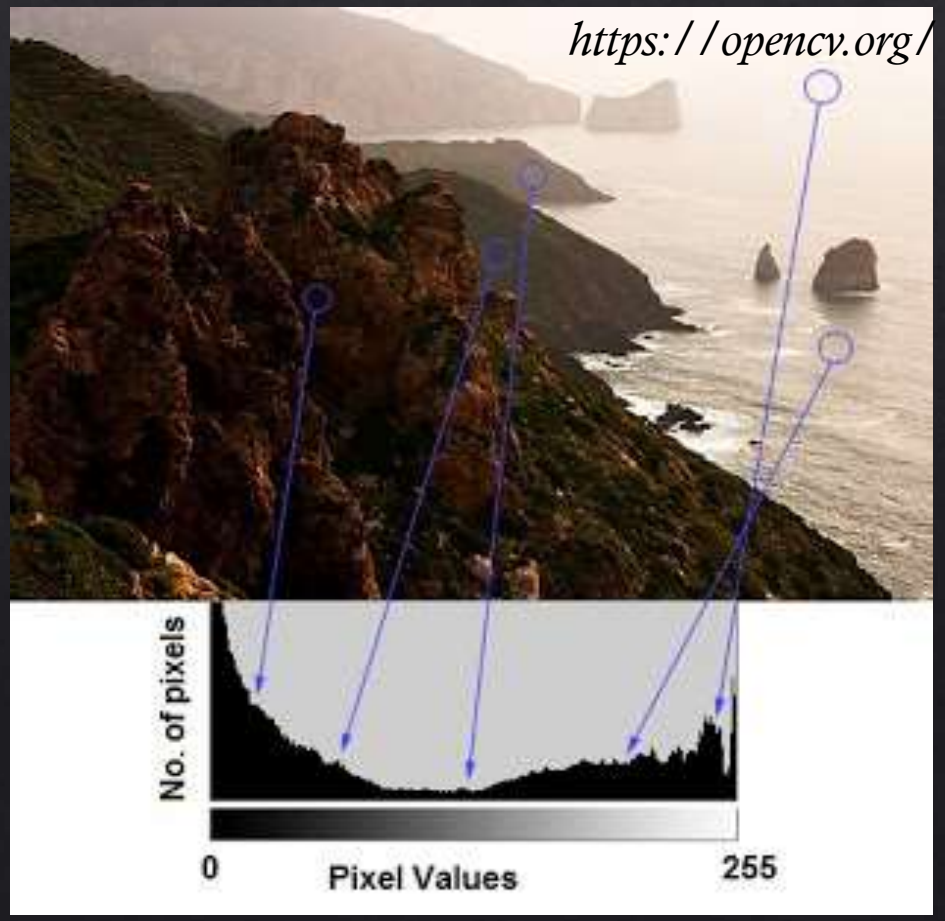
Contours

```
import cv2 as cv
img = cv.imread('logo.png')
imgray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
img_binary = cv.inRange(imgray,170,255)
im2, contours, hierarchy = cv.findContours(img_binary,cv.RETR_TREE,cv.CHAIN_APPROX_SIMPLE)
cv.drawContours(img,contours,-1,(255,0,0),3) #Draw all contours
cv.imshow('img',img)
cv.waitKey(0)
cv.destroyAllWindows()

#Draw certain contour
cv.drawContours(img,contours,1,(0,255,0),3)
#Or use another form
cnt = contours[1]
cv.drawContours(img,[cnt],0,(0,255,0),3)
```

Histogram

- Histogram is a plot which shows the intensity distribution of an image
- Pixel value on X-axis
- Number of pixels on Y-axis



Histogram

- **cv.calcHist([image],[channel],mask,[histSize],[ranges])**
  - **image**: source image(given in square brackets)
  - **channel**: index of channel for calculating histogram(given in square brackets)
  - **mask**: mask image used to restrict the area to calculate('None' for full area)
  - **histSize**: the size of bin(seperate the whole value interval into subintervals with same size)
  - **ranges**: the value range to calculate(normally [0,256])

Histogram - Task

Task: find and draw histogram of channel ‘r’ with histSize = [256] (full scale)

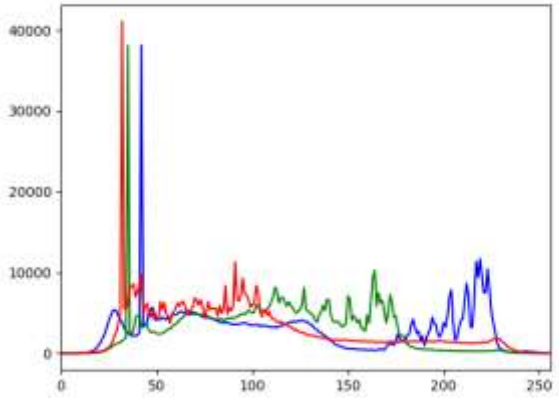
- $ cd ~/tasks
- $ nano task10.py

Remind: **cv.calcHist([image],[channel],mask,[histSize],[ranges])**

- **channel**: index of channel(given in square brackets)
- **mask**: mask image(‘None’ for full area)
- **histSize**: the size of bin([256] for full scale)
- **ranges**: the value range to calculate(normally [0,256])

Histogram

```
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('southampton.jpg')
color = ('b','g','r')
for i, col in enumerate(color):
        hist = cv.calcHist([img],[i],None,[256],[0,256])
        plt.plot(hist, color = col)
        plt.xlim([0,256])
plt.show()
```



UNIVERSITY OF Southampton

Thanks for listening!