ECE 478-578
Intelligent Robotics I

PhD. Husnu Melih Erdogan – Electrical & Computer Engineering

herdogan@pdx.edu - Teaching Assistant

Lab Assistant – Intelligent Robotics Lab

Portland State
UNIVERSITY

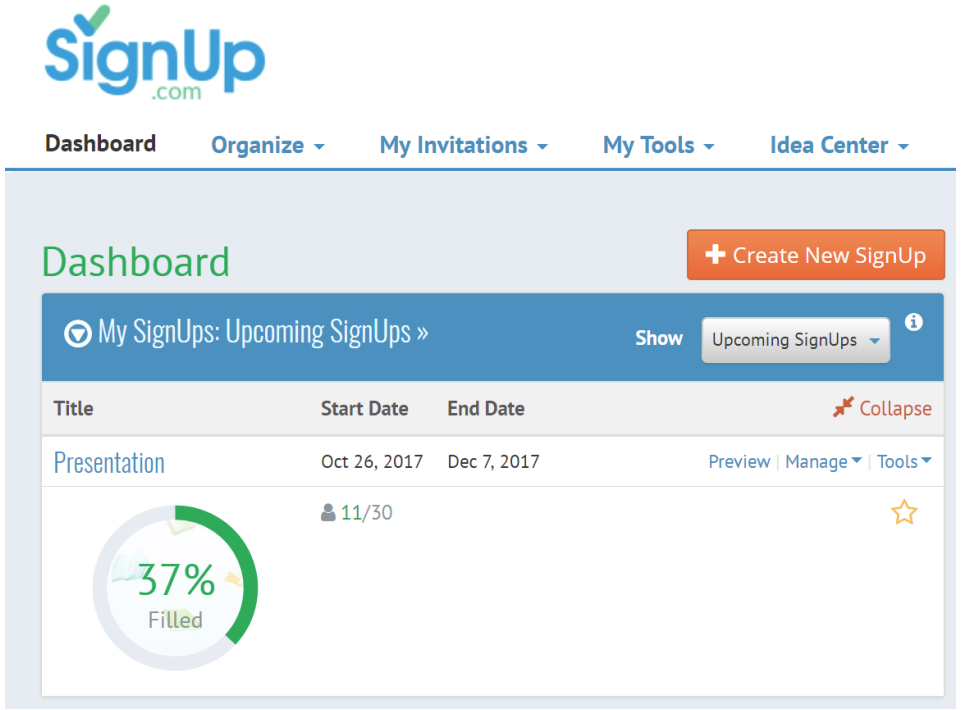
Lecture 12

Introduction to OpenCV 3

Portland State
UNIVERSITY

Announcement

- https://signup.com/client/invitation2/secure/2085477/false#/invitation



- D2L -> Advanced Papers to Read for Graduate Students and Extra Credit
- Example : Deep and Recurrent Neural Networks for Applications in Robotics research paper presentation by Matt Fleetwood

Portland State UNIVERSITY

Open CV

Portland State
UNIVERSITY

Quick Facts – Who is Gary Bradski?

Gary Bradski is an American scientist, engineer, entrepreneur, and author. He co-founded Industrial Perception, a company that developed perception applications for industrial robotic application (since acquired by Google in 2012) and has worked on the OpenCV Computer Vision library, as well as published a book on that library.

Education

Boston University
PhD
AI, machine learning, neuro-modeling
1989 – 1993

University of California, Berkeley
BS
EECS, electrical engineering, computer science
1979 – 1981

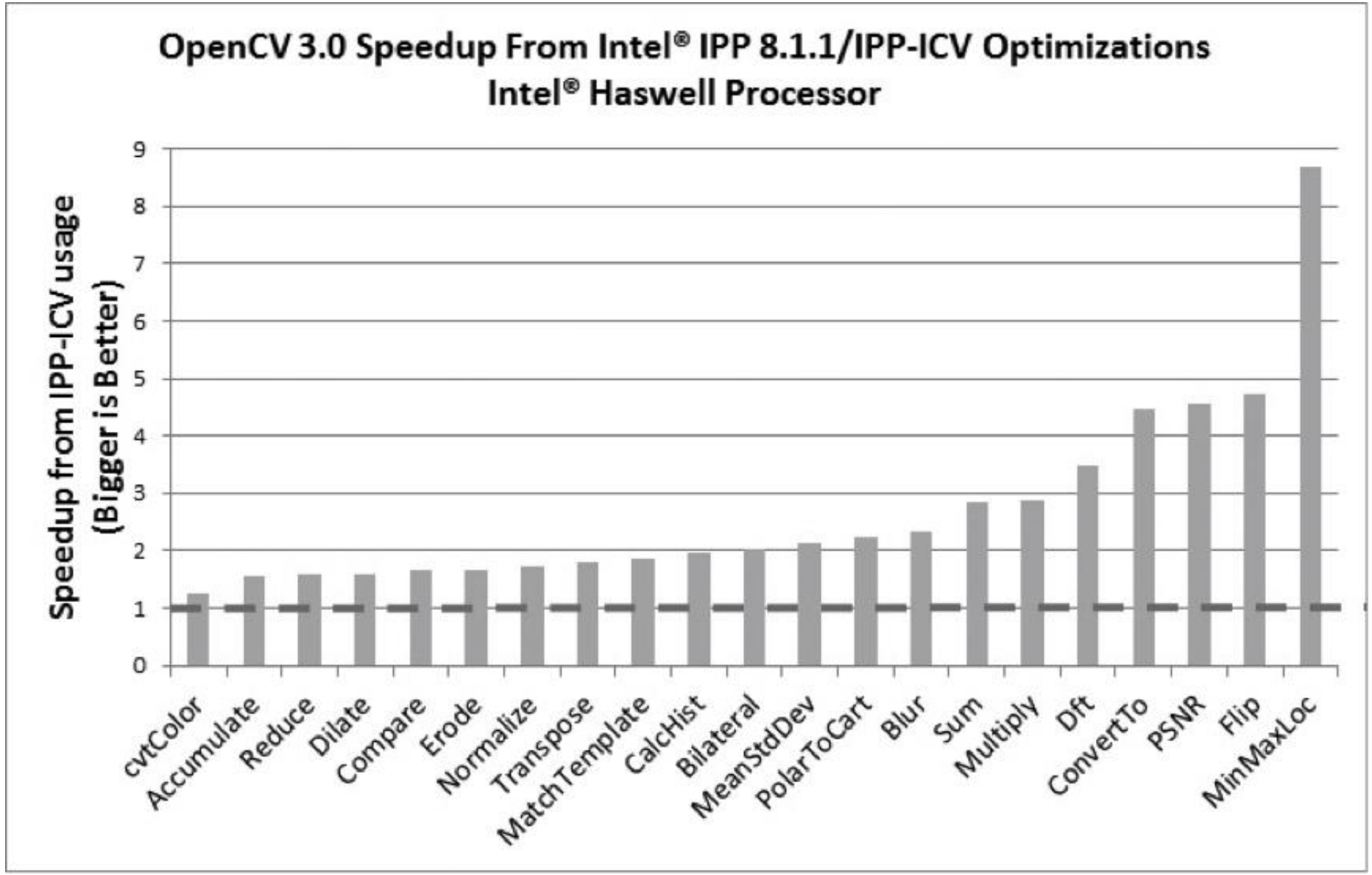
Portland State UNIVERSITY

What is OpenCV?

- OpenCV is an open source computer vision library available from http://opencv.org
- In 1999 Gary Bradski, working at Intel Corporation, launched OpenCV with the hopes of accelerating computer vision and artificial intelligence by providing a solid infrastructure for everyone working in the filed.
- The Library is written in C and C++.
- It runs on Linux, Windows, Mac OS X.
- There are active development interfaces for Python, Java, MATLAB, and other programming languages, including Android and iOS for mobile programming.

What is OpenCV?

- OpenCV is designed for computational efficiency and with a strong focus on real time applications.
- It can take advantage of multicore processors.
- If you want further automatic optimization on Intel architectures, Intel's Integrated Performance Primitives (IPP) libraries can be used.
- It is consist of low level optimized routines in many different algorithmic areas.
- Starting from OpenCV 3 Intel granted the OpenCV team this IPP for free.



Portland State UNIVERSITY

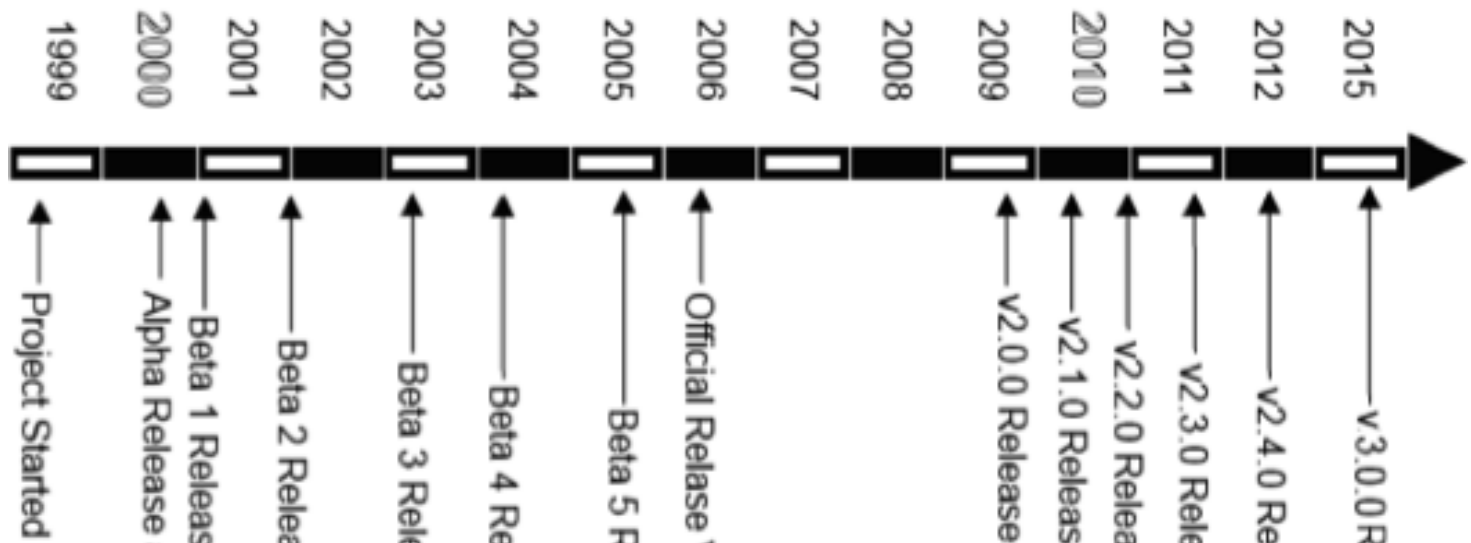
What is OpenCV?

- It is consist of over 500 different functions.
- These functions are for many different areas,
  - Factory product inspections,
  - Medical imaging
  - Security
  - User Interface
  - Camera Calibration
  - Stereo Vision
  - Robotics and more.

What is OpenCV?

- Computer vision and machine learning are BFF.
- That is why OpenCV contains a full, general purpose, Machine Learning Library called ML module.
- It is very useful for the vision tasks that are at the core of OpenCV.
- It is also general enough to be used for many machine learning problems.

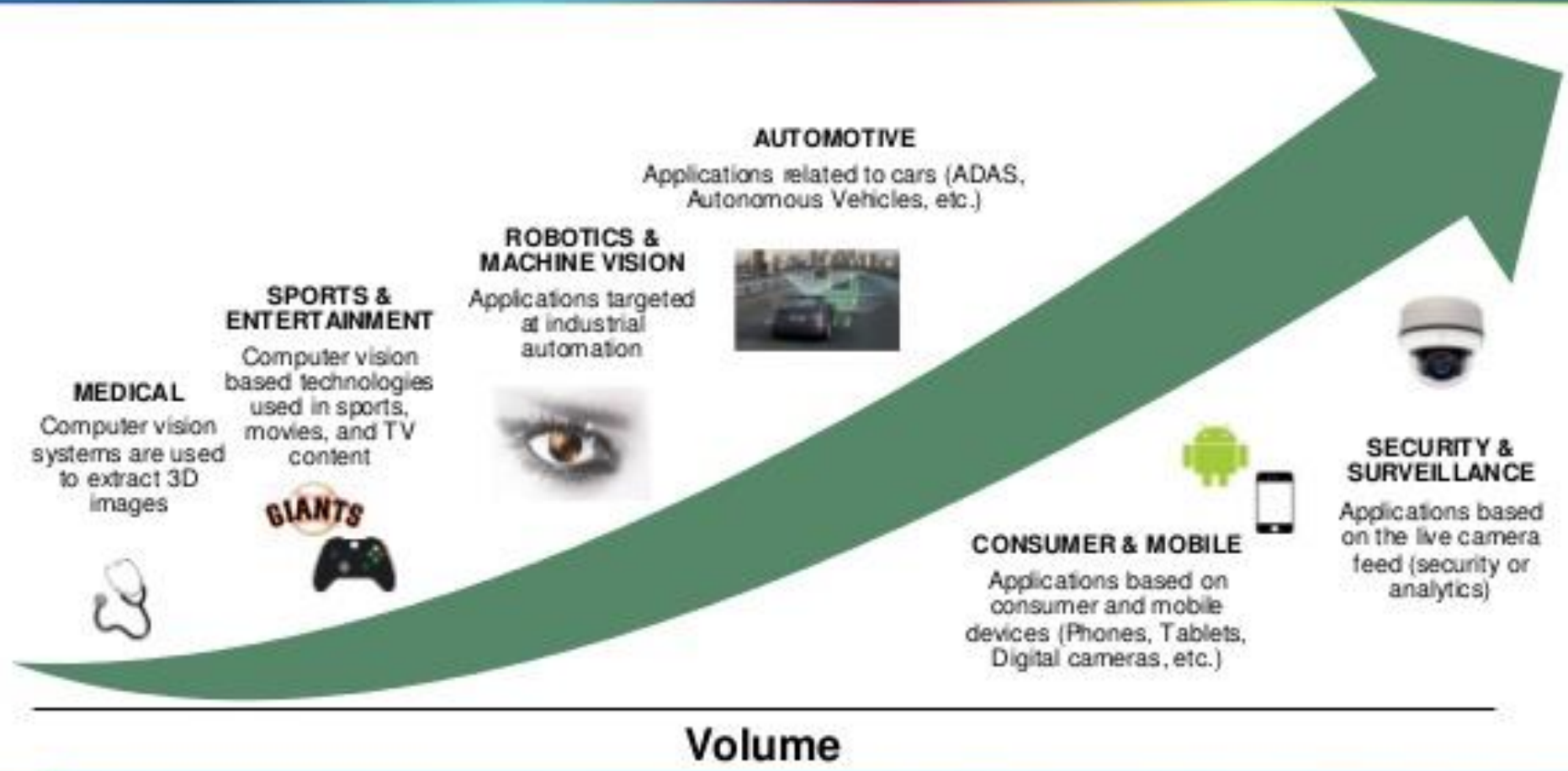
OpenCV Timeline

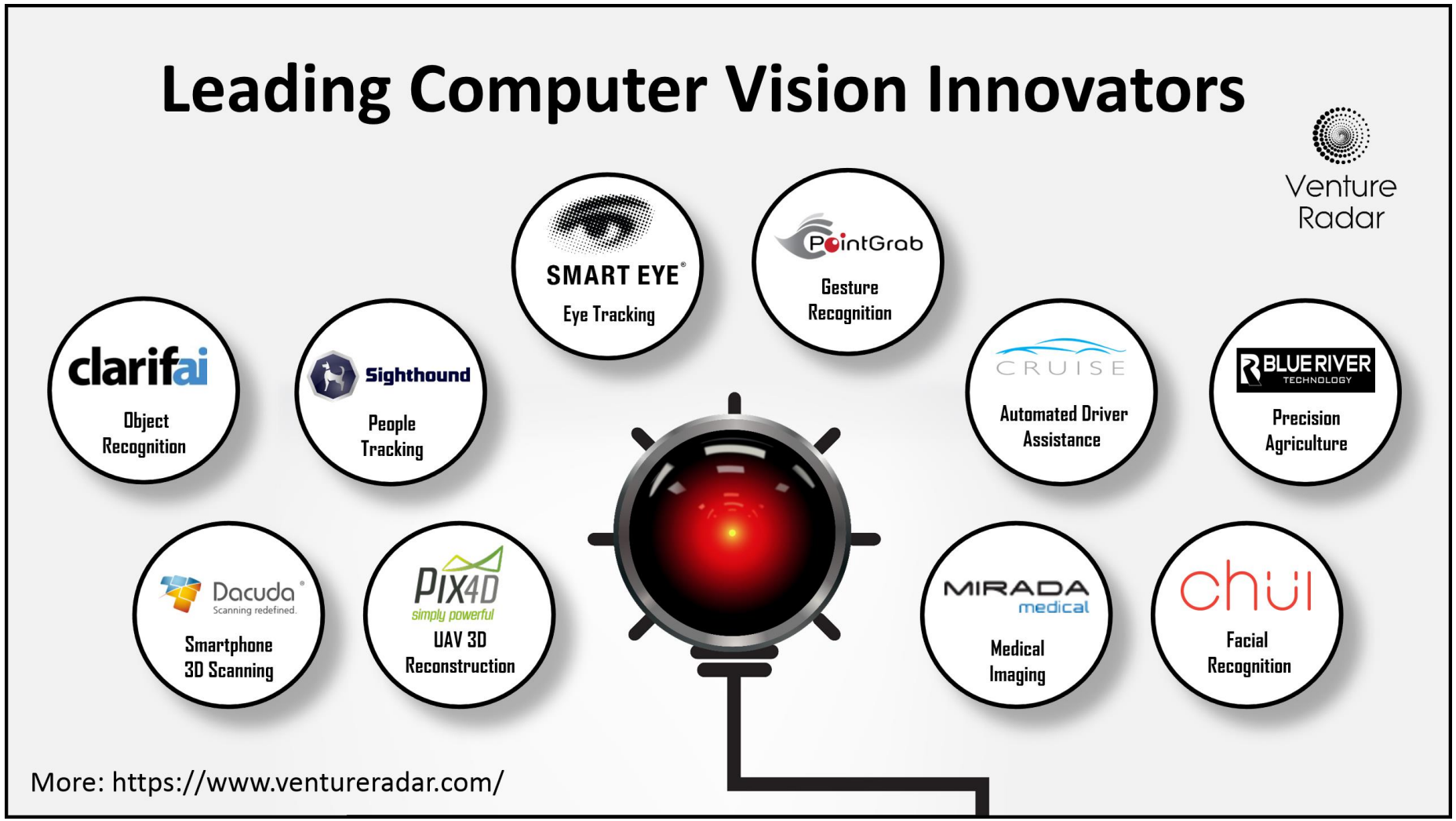


Who uses OpenCV?

- Computer Engineers and Scientist
- Programmers
- Other engineering majors
- Research centers such as Stanford, MIT, CMU, Cambridge.
- People from major companies such as IBM, Microsoft, Intel, SONY, Siemens, and Google.
- OpenCV has a very large user community around the world.

Application Markets for Computer Vision





- http://blog.ventureradar.com/2015/10/21/top-10-innovative-companies-in-computer-vision/

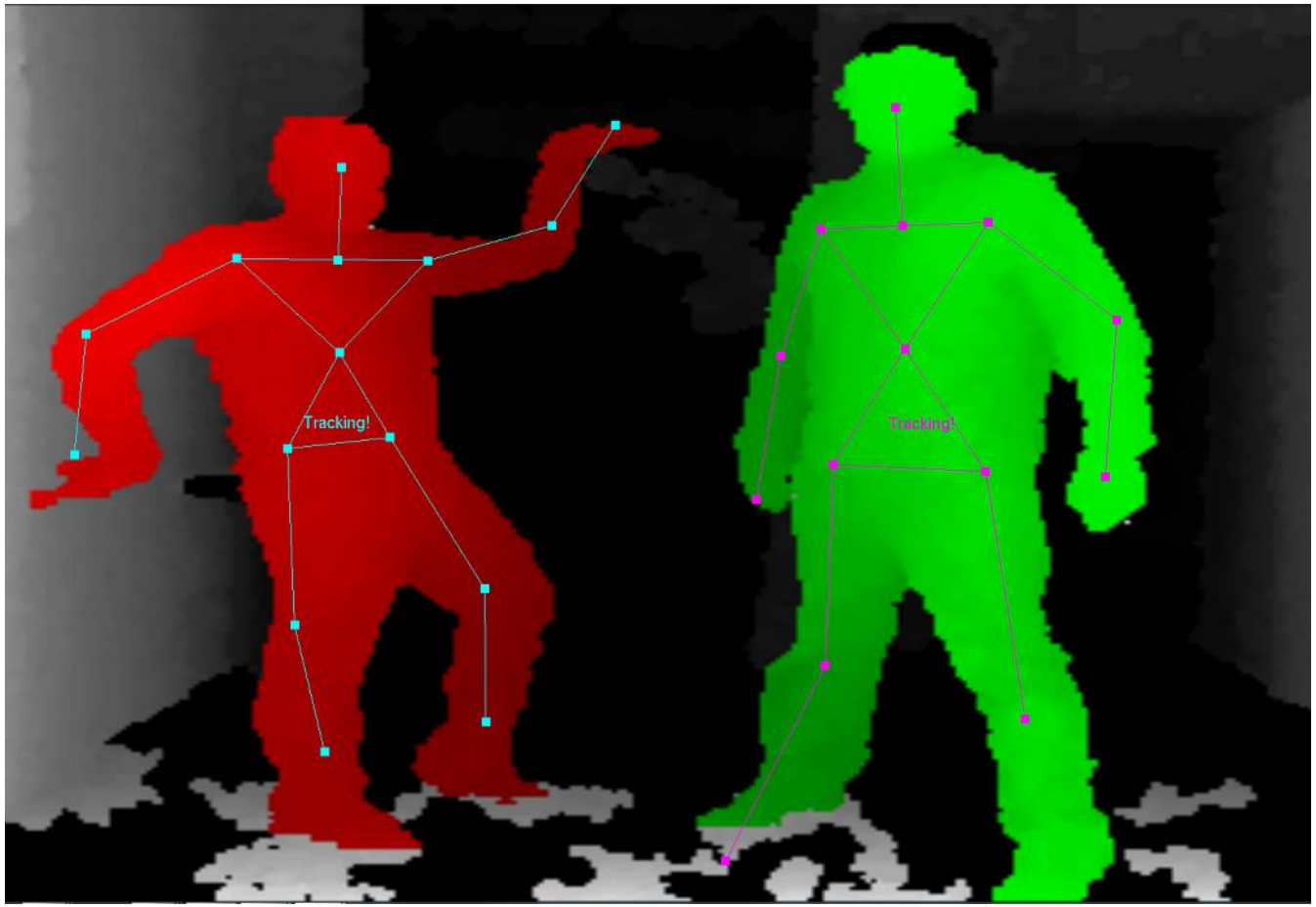
What is Computer Vision?

- Computer vision is concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images. It involves the development of a theoretical and algorithmic basis to achieve automatic visual understanding.
- The applications of computer vision are numerous and include:
- Agriculture, augmented reality, autonomous vehicles, biometrics, character recognition, forensics, industrial quality inspection, face recognition, gesture analysis, geoscience, image restoration medical image analysis, pollution monitoring, process control, remote sensing, **robotics**, security and surveillance, transport

http://www.bmva.org/visionoverview

Applications of Computer Vision

- Face Recognition
- Pose Estimation
- Body Tracking
- Speech Reading
- Palm Recognition
- Car Tracking



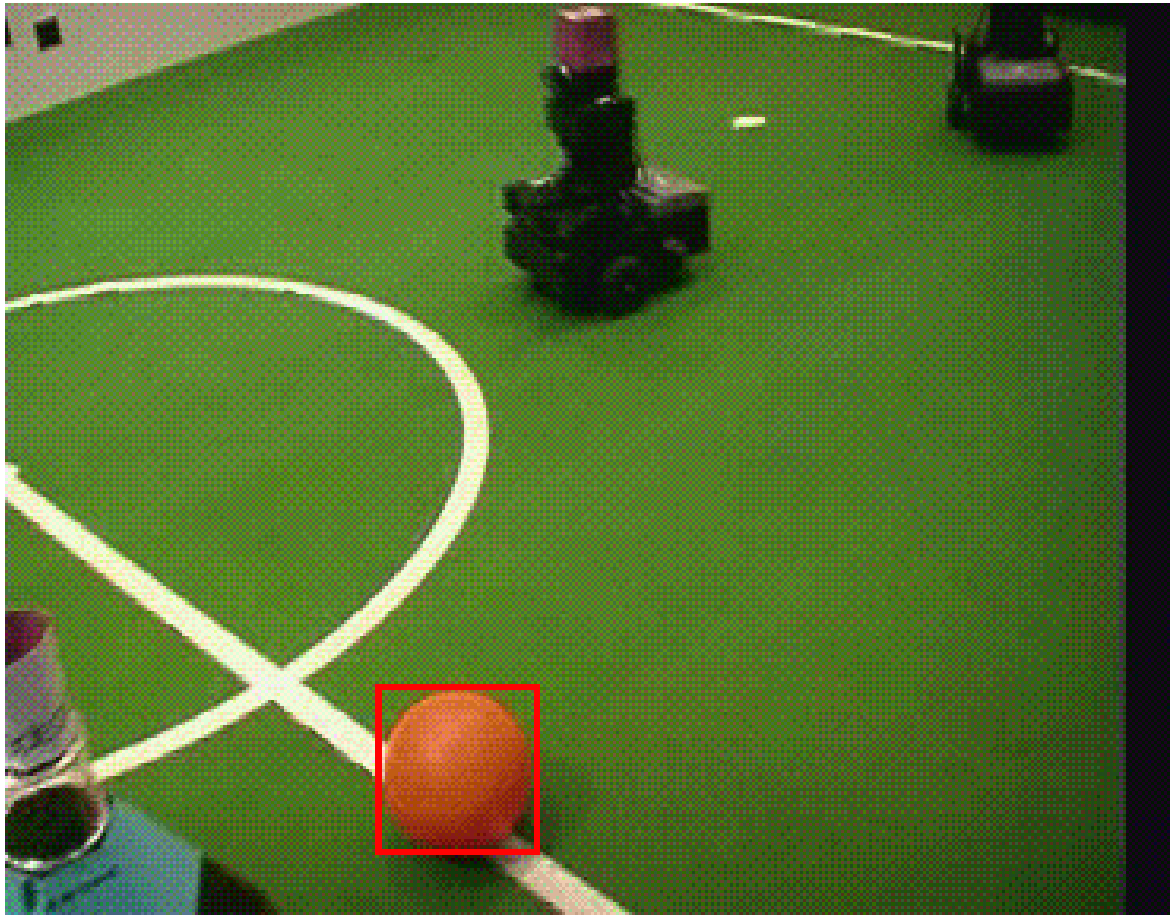
Portland State UNIVERSITY

Applications of Computer Vision

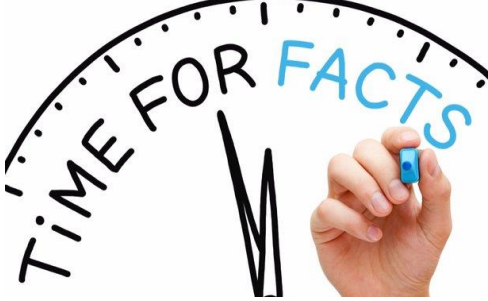
- Face detection
- Face recognition

Applications of Computer Vision

- Object Detection
- Object Recognition



What is Robocup's ultimate goal?



- Robocup's first competition was held in Nagaya, Japan in 1997
- More than 40 countries are represented at Robocup in 2015 including 400 participants ranging from engineers, professors, and students.
- "The ultimate goal of Robocup is to develop humanoid soccer-playing robots that can beat the FIFA world champion team," Gerhard Kraetzschmar, General Chair of last year's RoboCup, previously told Digital Trends. "We hope to reach that goal by 2050."

Portland State UNIVERSITY

Some interesting OpenCV project videos from the Internet

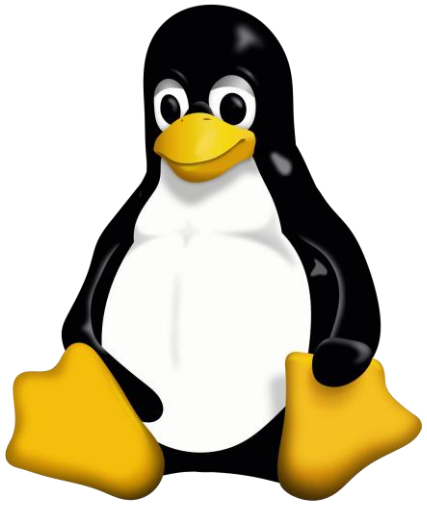
- Finger Drawing
  https://www.youtube.com/watch?v=Z43_hCM74rU
- Traffic Counting
  https://www.youtube.com/watch?v=z1Cvn3_4yGo
- Object Tracking
  https://www.youtube.com/watch?v=CigGvt3DXIw
- Face Recognition and Tracking
  https://www.youtube.com/watch?v=vRHoQVZLvoM
- OpenCV Artificial Vision with Raspberry Pi
  https://www.youtube.com/watch?v=YAu1KlKblR4
- Lane Detection and Steering
  https://www.youtube.com/watch?v=8h9vU1pnNZA
- MIT Dockietown
  https://vimeo.com/152233002

Installing OpenCV - Windows

- https://docs.opencv.org/3.0-beta/doc/tutorials/introduction/windows_install/windows_install.html

Installing OpenCV - Linux

- https://docs.opencv.org/3.0-beta/doc/tutorials/introduction/windows_install/windows_install.html



Portland State UNIVERSITY

Installing OpenCV Mac OSX

- https://www.pyimagesearch.com/2016/12/19/install-opencv-3-on-macos-with-homebrew-the-easy-way/

Installing OpenCV – Other Systems

- https://docs.opencv.org/3.0-beta/doc/tutorials/introduction/table_of_content_introduction/table_of_content_introduction.html

Online Documentation and Tutorials

- https://docs.opencv.org/3.0-beta/doc/tutorials/tutorials.html
- https://www.pyimagesearch.com/
- https://pythonprogramming.net/loading-images-python-opencv-tutorial/
- https://www.learnopencv.com/

OpenCV Contribution Repository

- In OpenCV 3.0 the previously monolithic library has been split into two parts:
  - Mature OpenCV
  - Opencv_contrib - The current state of the art in larger vision functionality

- Opencv_contrib library is maintained and developed by the community. It may have parts under non-OpenCV license and may include patented algorithms.

- https://github.com/opencv/opencv_contrib

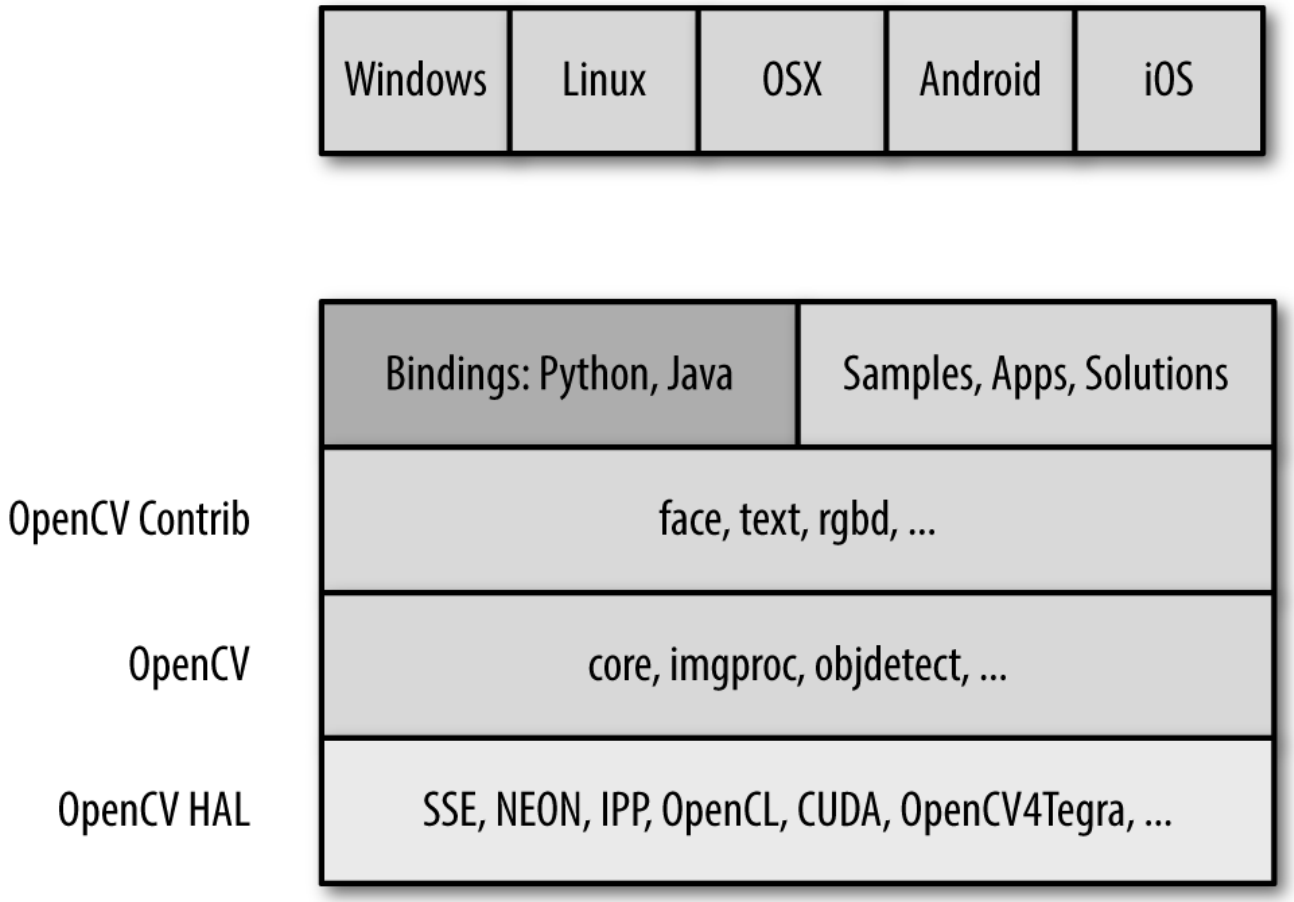
opencv_contrib installation

- Python: https://pypi.python.org/pypi/opencv-contrib-python
- C++: https://github.com/opencv/opencv_contrib
  - Linux : https://docs.opencv.org/trunk/d7/d9f/tutorial_linux_install.html
  - Windows: https://www.learnopencv.com/install-opencv3-on-windows/
  - OSX: https://www.learnopencv.com/install-opencv-3-on-yosemite-osx-10-10-x/

How to contribute to the OpenCV repository

- Details documentation about how to submit your algorithm or contribution to OpenCV repository.
- https://github.com/opencv/opencv/wiki/How_to_contribute

OpenCV and Portability



Lineer Algebra

Portland State
UNIVERSITY

Linear Algebra – Quick Review

- In order to understand how most of the image processing algorithms work you need to know linear algebra.

- Good Review – Numeric Understanding
- http://cs229.stanford.edu/section/cs229-linalg.pdf

- Essence of linear algebra – Geometric intuitions – Geometric Understanding of Linear algebra.
- https://www.youtube.com/watch?v=kjBOesZCoqc

Lineer Algebra – Matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{i1} & a_{i2} & a_{i3} & \cdots & a_{in} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix} \begin{matrix} \\ \\ \\ \\ m \text{ rows} \end{matrix}$$

n columns

$$A = 2 \quad B = \begin{bmatrix} 3 & 4 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & 11 \end{bmatrix} \quad C = \begin{bmatrix} 12 & 13 & 14 \\ 15 & 16 & 17 \\ 18 & 19 & 20 \end{bmatrix} \quad D = \begin{bmatrix} 21 \\ 22 \\ 23 \end{bmatrix} \quad E = \begin{bmatrix} 24 & 25 & 26 \end{bmatrix}$$

Addition/Subtraction

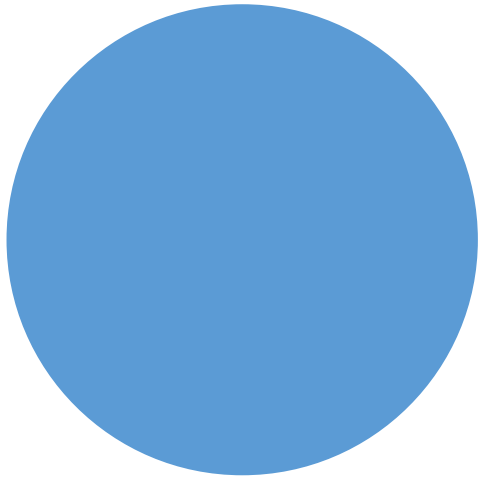
$$A + B = \begin{bmatrix} 3+2 & 4+2 & 5+2 \\ 6+2 & 7+2 & 8+2 \\ 9+2 & 10+2 & 11+2 \end{bmatrix} \qquad B + C = \begin{bmatrix} 3+12 & 4+13 & 5+14 \\ 6+15 & 7+16 & 8+17 \\ 9+18 & 10+19 & 11+20 \end{bmatrix}$$

$B + D$ = not valid

$$B - A = \begin{bmatrix} 3-2 & 4-2 & 5-2 \\ 6-2 & 7-2 & 8-2 \\ 9-2 & 10-2 & 11-2 \end{bmatrix} \qquad B + C = \begin{bmatrix} 3-12 & 4-13 & 5-14 \\ 6-15 & 7-16 & 8-17 \\ 9-18 & 10-19 & 11-20 \end{bmatrix}$$

$B - D$ = not valid

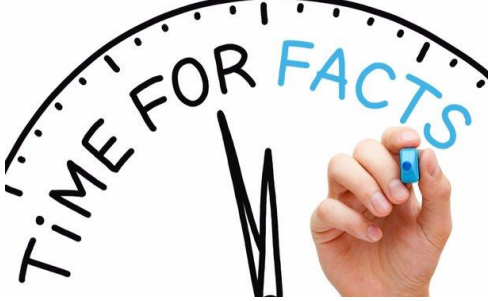
Portland State UNIVERSITY



OpenCV with Python

Portland State
UNIVERSITY

The Very First Digital Image



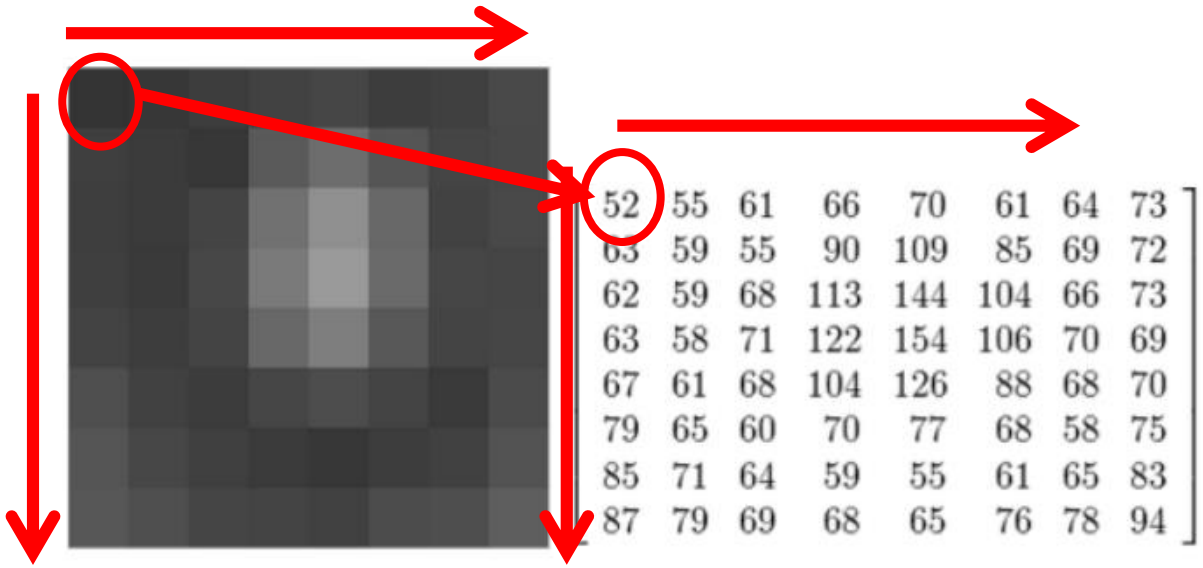
- Russel A. Kirsch took a picture of his infant son and scanned it into a computer. It was the first digital image: a grainy, black-and-white baby picture that literally changed the way we view the world.
- The picture of Kirsch's three-month-old son, was captured as just 30,976 pixels, a 176 × 176 array, in an area measuring 5 cm × 5 cm

Russel A. Kirsch

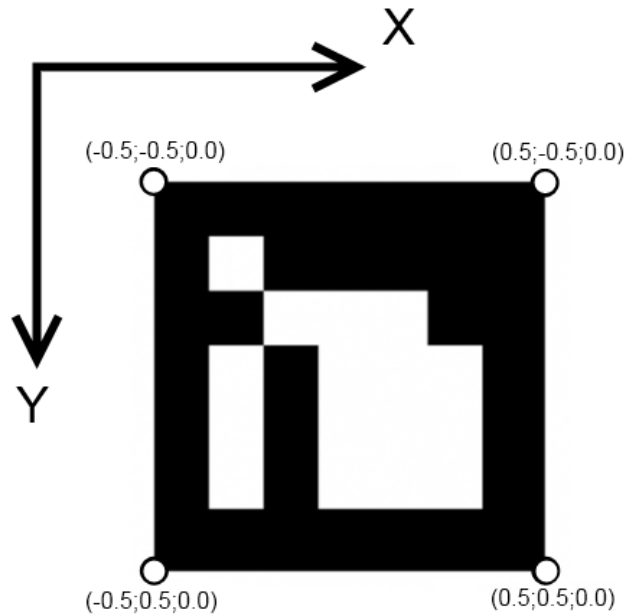
- https://www.wired.com/2010/06/smoothing-square-pixels/

Portland State UNIVERSITY

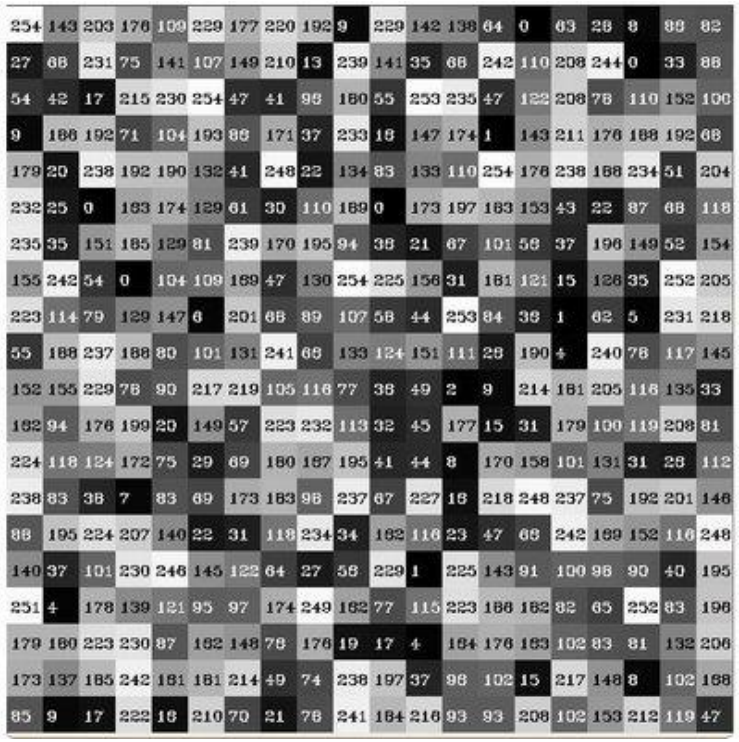
How does an image look like in OpenCV?



$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix}$$



How does an image look like in OpenCV?



Grayscale Image

	Column 0			Column 1			Column ...			Column m		
Row 0	0,0	0,0	0,0	0,1	0,1	0,1	...	...	...	0, m	0, m	0, m
Row 1	1,0	1,0	1,0	1,1	1,1	1,1	...	...	...	1, m	1, m	1, m
Row ...	...,0	...,0	...,0	...,1	...,1	...,1	...	...	...	..., m	..., m	..., m
Row n	n,0	n,0	n,0	n,1	n,1	n,1	n,...	n,...	n,...	n, m	n, m	n, m

Color Image

Portland State UNIVERSITY

Supported Image File Formats

- Currently, the following file formats are supported:
  - Windows bitmaps - *.bmp, *.dib (always supported)
  - JPEG files - *.jpeg, *.jpg, *.jpe (see the *Notes* section)
  - JPEG 2000 files - *.jp2 (see the *Notes* section)
  - Portable Network Graphics - *.png (see the *Notes* section)
  - Portable image format - *.pbm, *.pgm, *.ppm (always supported)
  - Sun rasters - *.sr, *.ras (always supported)
  - TIFF files - *.tiff, *.tif (see the *Notes* section)

How to load and display an image?

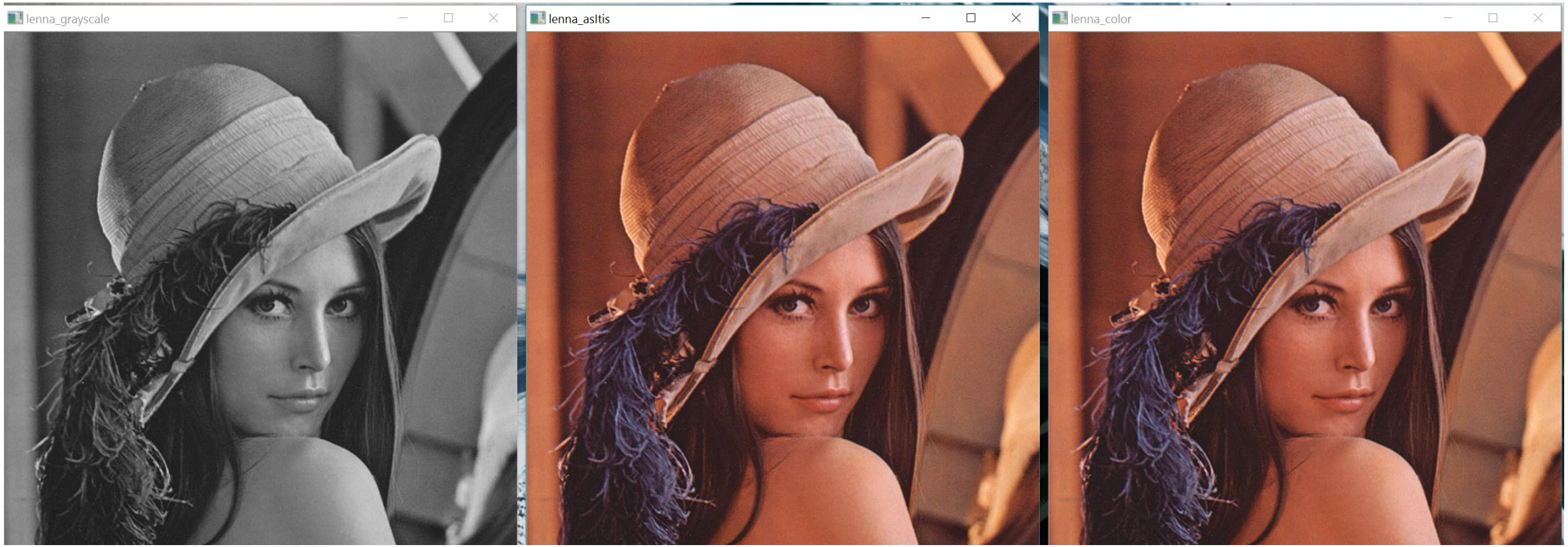
- **Imread(filename, flag)**
- **Python:** cv.**LoadImage**(filename, iscolor=CV_LOAD_IMAGE_COLOR)

- CV_LOAD_IMAGE_UNCHANGED (<0) loads the image as is (including the alpha channel if present)
- CV_LOAD_IMAGE_GRAYSCALE (0) loads the image as an grayscale one
- CV_LOAD_IMAGE_COLOR (>0) loads the image in the BGR format

How to load and display an image?

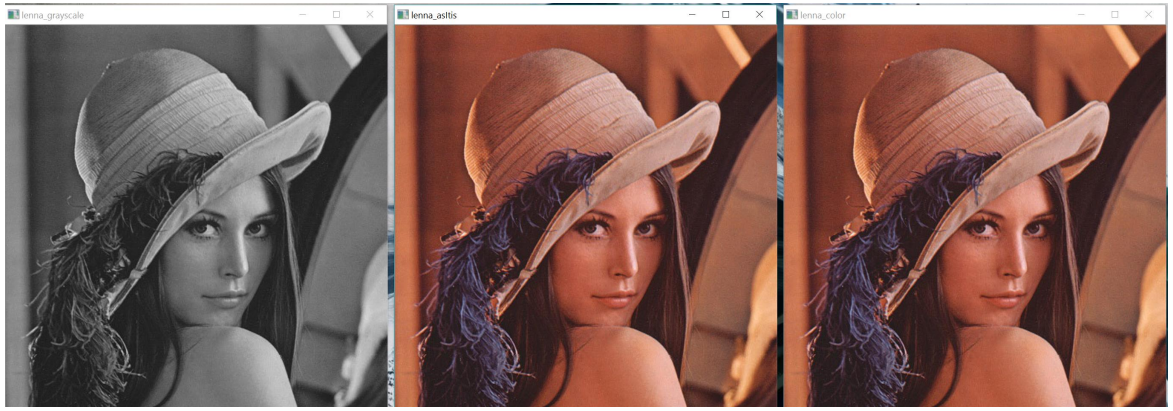
```
import cv2

# = 0: loads the image as a grayscale image
# < 0: loads the image as is
# > 0: loads the image in the BGR format

grayscale = cv2.imread('lenna.jpg',0)
asItis = cv2.imread('lenna.jpg',1)
colorBgr = cv2.imread('lenna.jpg',-1)

cv2.imshow('lenna_grayscale',grayscale)
cv2.imshow('lenna_asItis',asItis)
cv2.imshow('lenna_color',colorBgr)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

How to load and display an image?



Quick Facts – Who is Lenna (Lena)?

Lena Söderberg



Lenna or **Lena** is the name given to a standard test image widely used in the field of image processing since 1973. It is a picture of Lena Söderberg, shot by photographer Dwight Hooker, cropped from the centerfold of the November 1972 issue of Playboy magazine.
Over the years there has been quite a bit of controversy over the use of this image.

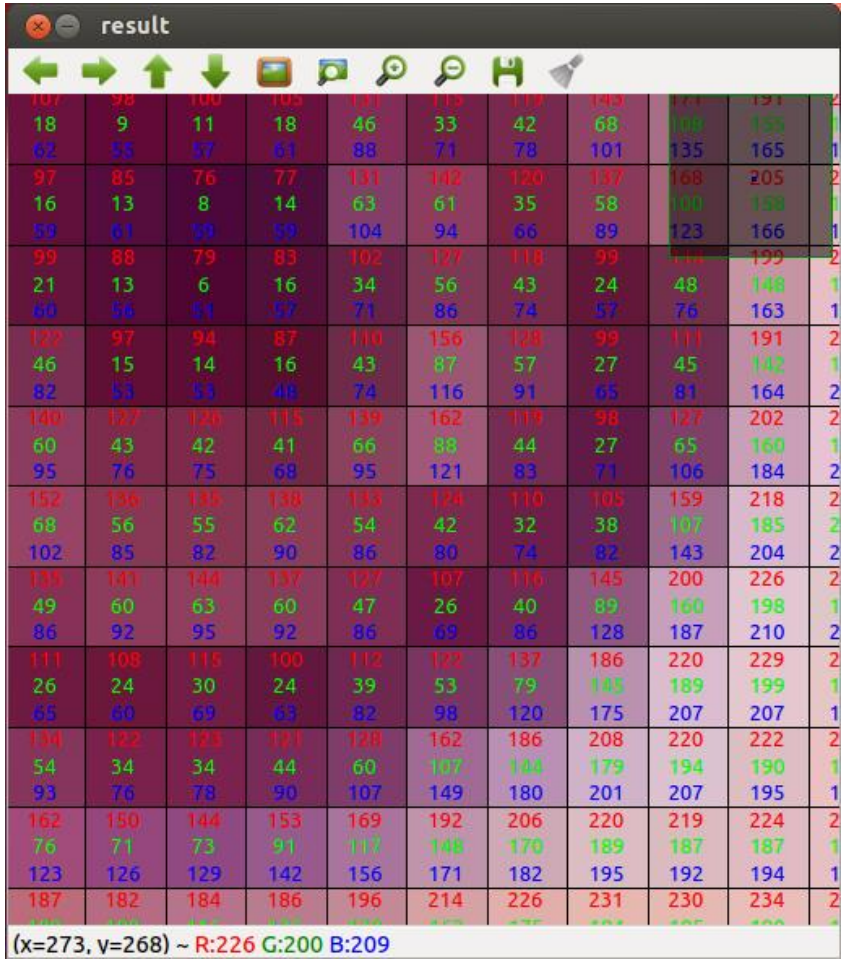
Some people proposed banning the use of this image because of its source. Also, Playboy threatened to prosecute the unauthorized use of the image.

Image of Lena Söderberg used in many image processing experiments.

Portland State UNIVERSITY

RGB (BGR) Image

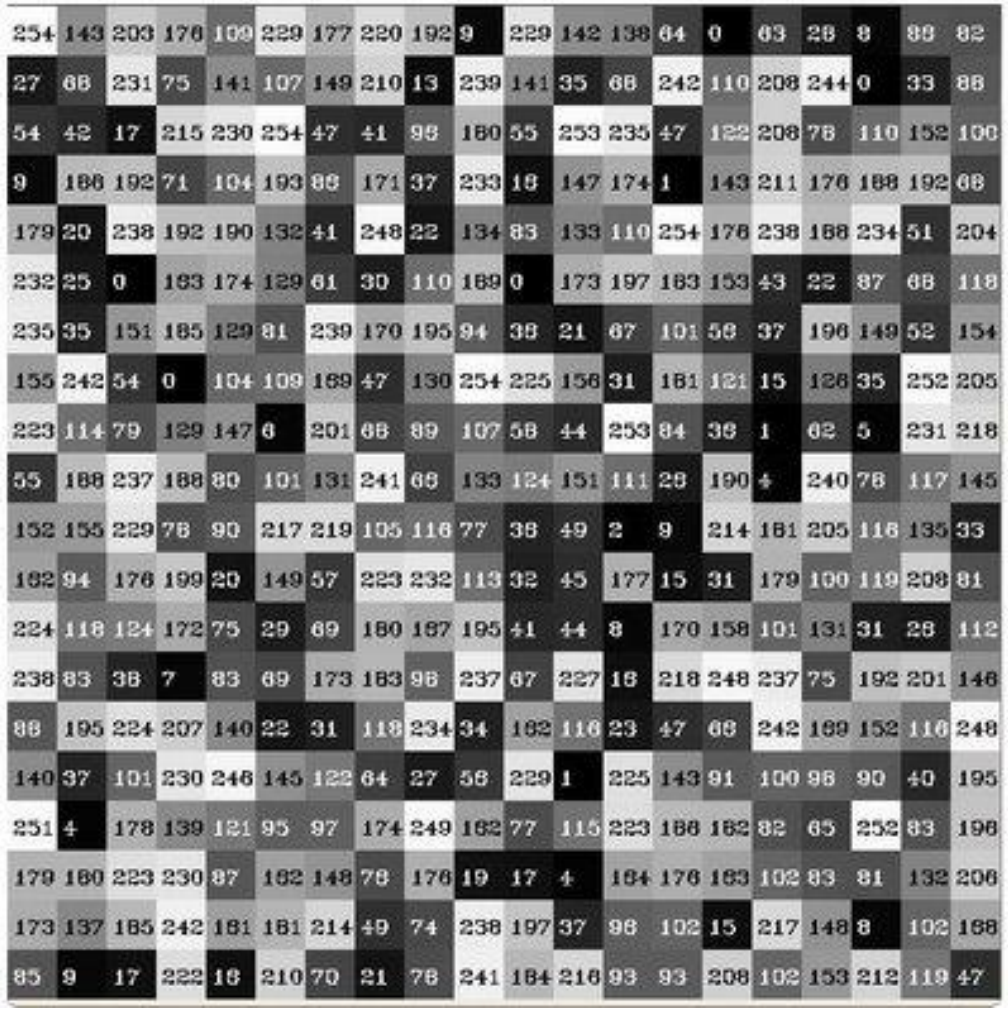
- The **RGB color model** is an additive color model in which red, green and blue light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, red, green, and blue.



Portland State UNIVERSITY

Grayscale Image

- For a grayscale images, the pixel value is a single number that represents the brightness of the pixel. The most common pixel format is the byte image, where this number is stored as an 8-bit integer giving a range of possible values from **0 to 255**. Typically zero is taken to be black, and 255 is taken to be white.



Portland State UNIVERSITY

How to load and play a video – Color Video

cv2.**VideoCapture**(filename) → <VideoCapture object>
cv2.**VideoCapture**(device) → <VideoCapture object>

Parameters:

- **filename** – name of the opened video file (eg. video.avi) or image sequence (eg. img_%02d.jpg, which will read samples like img_00.jpg, img_01.jpg, img_02.jpg, ...)
- **device** – id of the opened video capturing device (i.e. a camera index). If there is a single camera connected, just pass 0.

Supported Video File Formats

- The supported format vary by system but should always include an AVI.

- .avi
- .mp4

How to load and play a video – Color Video

```
import cv2

# Create a VideoCapture object and read from input file
cap = cv2.VideoCapture('Seahawks.avi')

# Read until video is completed
while(cap.isOpened()):
  # Capture frame-by-frame
  ret, frame = cap.read()
  if ret == True:

    # Display the resulting frame
    cv2.imshow('Frame',frame)

    # Press Q on keyboard to  exit
    if cv2.waitKey(25) & 0xFF == ord('q'):
      break

  # Break the loop
  else:
    break

# When everything done, release the video capture object
cap.release()

# Closes all the frames
cv2.destroyAllWindows()
```

How to load and play a video – Color Video

Portland State UNIVERSITY

How to load and play a video – Grayscale Video

```
import numpy as np
import cv2

flag = 1
# Create a VideoCapture object and read from input file
cap = cv2.VideoCapture('Seahawks.avi')

# Read until video is completed
while(cap.isOpened()):
    # Capture frame-by-frame
    ret, frame = cap.read()
    # Display the resulting frame
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Show only the 1st frame
    if flag == 1:
        cv2.imshow('frame1',gray)
        flag = 0

    # Display the resulting frame
    cv2.imshow('frame',gray)

    # Press Q on keyboard to  exit
    if cv2.waitKey(10) & 0xFF == ord('q'):
        break

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

How to load and play a video – Grayscale Video

Portland State UNIVERSITY

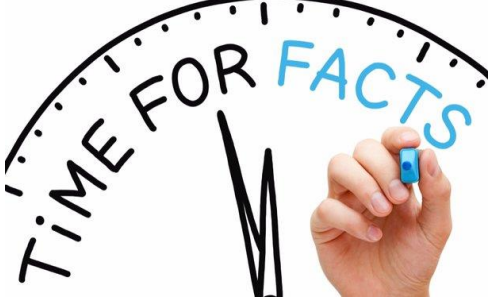
How to load and play a video –Grayscale the 1st Frame

Portland State UNIVERSITY

Input From a Camera

```
import cv2

# use camera as a source
# if you have more than 1 camera change the number 0, 1, 2 etc.
cap = cv2.VideoCapture(0)

while(True):
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Our operations on the frame come here
    # Make the frame grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    #Display the original frame
    cv2.imshow('color image frame',frame)

    # Display the resulting frame
    cv2.imshow('grayscale frame',gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

Quick Facts - Computer Vision in Sport



ELSEVIER

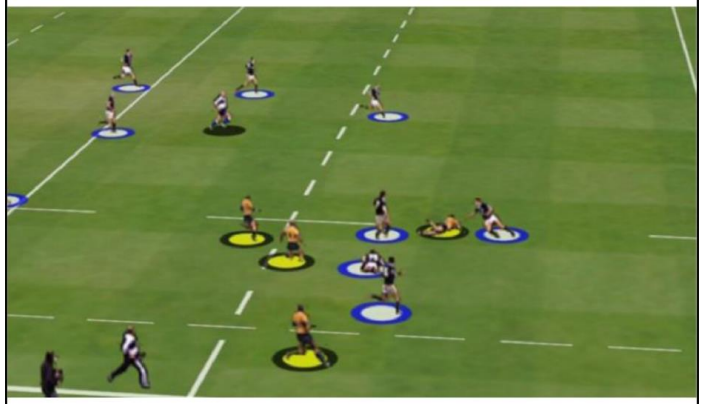
Computer Vision and Image Understanding

Volume 159, June 2017, Pages 3-18

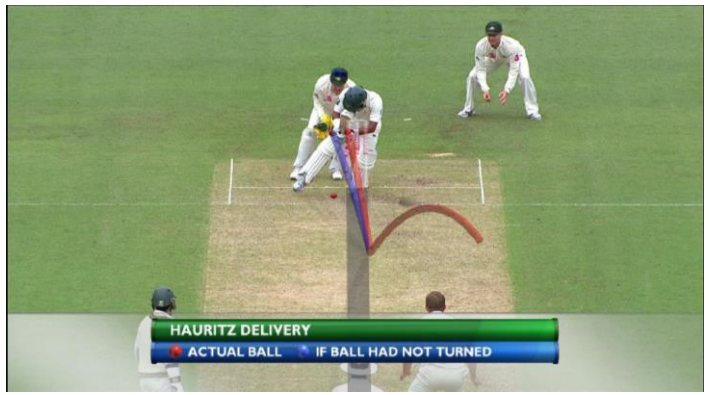
Computer vision for sports: Current applications and research topics

Graham Thomas [a], Rikke Gade [b], Thomas B. Moeslund [b], Peter Carr [c], Adrian Hilton [d]

(a) Original image

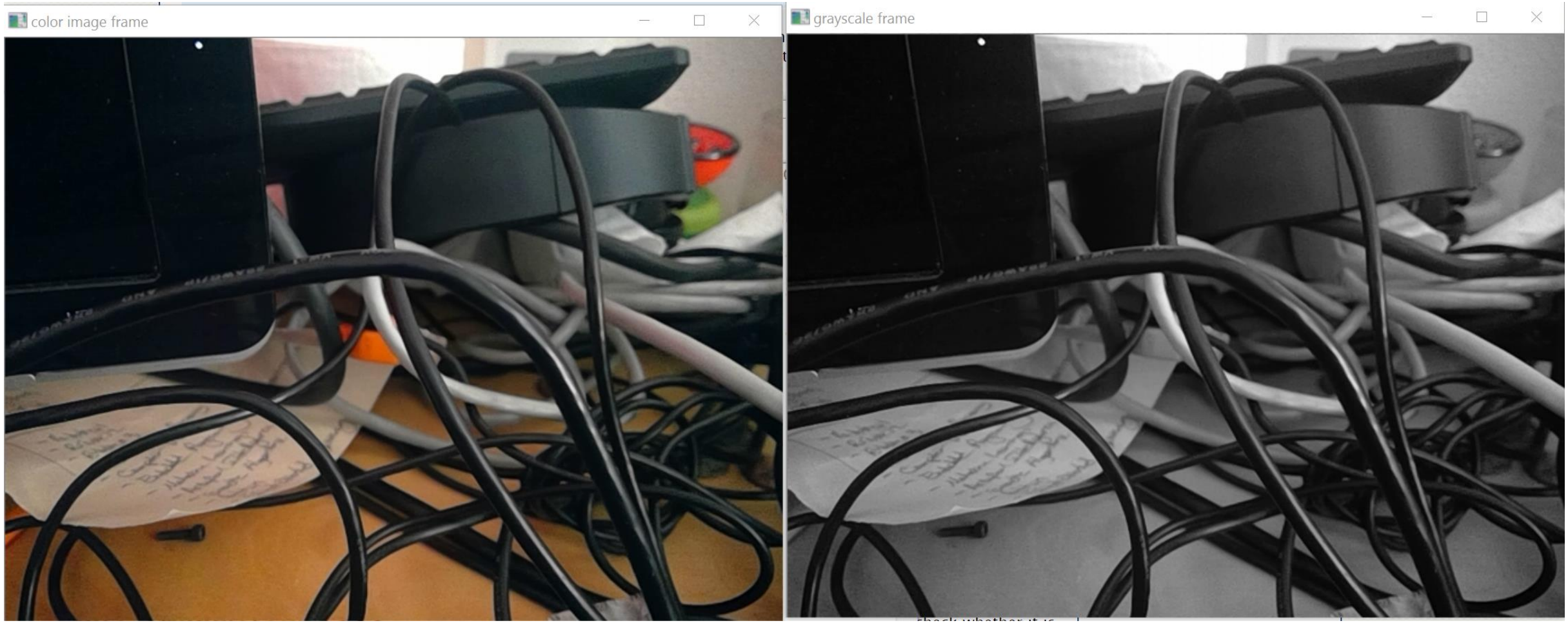


(b) Virtual view



Portland State UNIVERSITY

Input From a Camera



Saving Images

Saves an image to a specified file.

cv2.**imwrite**(filename, img[, params])

•**filename** – Name of the file.
•**image** – Image to be saved.
•**params**

–Format-specific save parameters encoded as pairs paramId_1, paramValue_1, paramId_2, paramValue_2, ... The following parameters are currently supported:

Parameters:

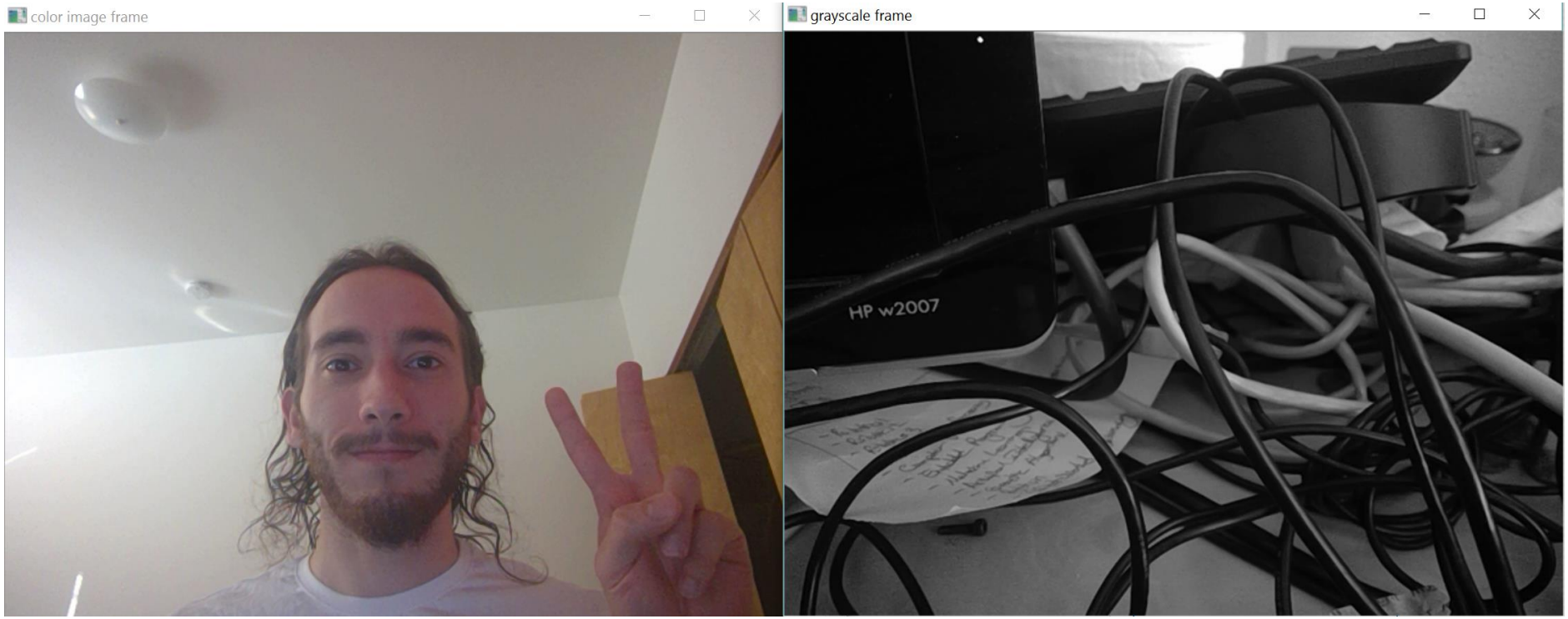
- For JPEG, it can be a quality (CV_IMWRITE_JPEG_QUALITY) from 0 to 100 (the higher is the better). Default value is 95.
- For WEBP, it can be a quality (CV_IMWRITE_WEBP_QUALITY) from 1 to 100 (the higher is the better). By default (without any parameter) and for quality above 100 the lossless compression is used.
- For PNG, it can be the compression level (CV_IMWRITE_PNG_COMPRESSION) from 0 to 9. A higher value means a smaller size and longer compression time. Default value is 3.
- For PPM, PGM, or PBM, it can be a binary format flag (CV_IMWRITE_PXM_BINARY), 0 or 1. Default value is 1.

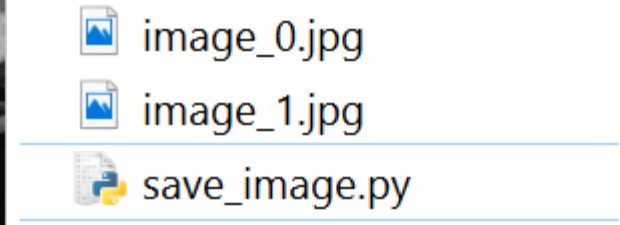
Saving Images

```
import cv2

# use camera as a source
# if you have more than 1 camera change the number 0, 1, 2 etc.
capBack = cv2.VideoCapture(0)
capFront = cv2.VideoCapture(1)

# counter for the image name
counter = 0

while(True):
    # Capture frame-by-frame from the back and the front camera
    ret, frameBack = capBack.read()
    ret, frameFront = capFront.read()

    # Our operations on the frame come here
    # Make the frame grayscale
    gray = cv2.cvtColor(frameBack, cv2.COLOR_BGR2GRAY)

    #Display the original frame from the front camera
    cv2.imshow('color image frame',frameFront)

    # Display the resulting frame
    cv2.imshow('grayscale frame',gray)

    # if c is presssed, save the image. Name the image image_<counter>.jpg.
    if cv2.waitKey(1) & 0xFF == ord('c'):
        name = "image_" + str(counter) + ".jpg"
        cv2.imwrite(name, frameBack)
        counter = counter + 1

    # break the while loop if q is pressed
    elif cv2.waitKey(1) & 0xFF == ord('q'):
            break

# When everything done, release the capture
capBack.release()
capFront.release()
cv2.destroyAllWindows()
```

Portland State UNIVERSITY

Saving Images





Saving Videos

VideoWriter constructors

cv.**CreateVideoWriter**(filename, fourcc, fps, frame_size, is_color=true)

Parameters:

- **filename** – Name of the output video file.
- **fourcc** – 4-character code of codec used to compress the frames. For example, CV_FOURCC('P','I','M','1') is a MPEG-1 codec, CV_FOURCC('M','J','P','G') is a motion-jpeg codec etc. List of codes can be obtained at Video Codecs by FOURCC page.
- **fps** – Framerate of the created video stream.
- **frameSize** – Size of the video frames.
- **isColor** – If it is not zero, the encoder will expect and encode color frames, otherwise it will work with grayscale frames (the flag is currently supported on Windows only).

Saving Videos

The functions/methods write the specified image to video file. It must have the same size as has been specified when opening the video writer.

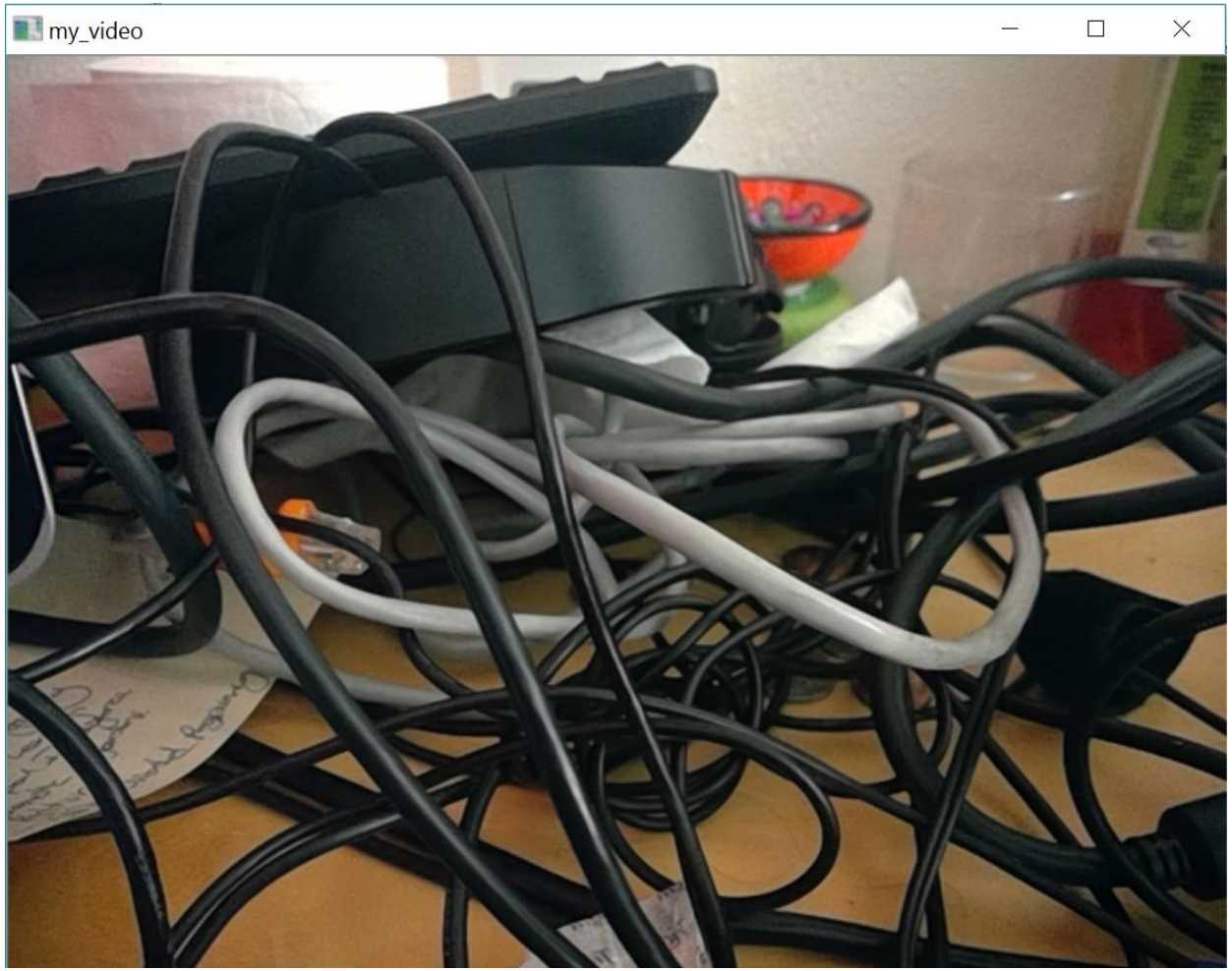
cv2.VideoWriter.**write**(image) → None

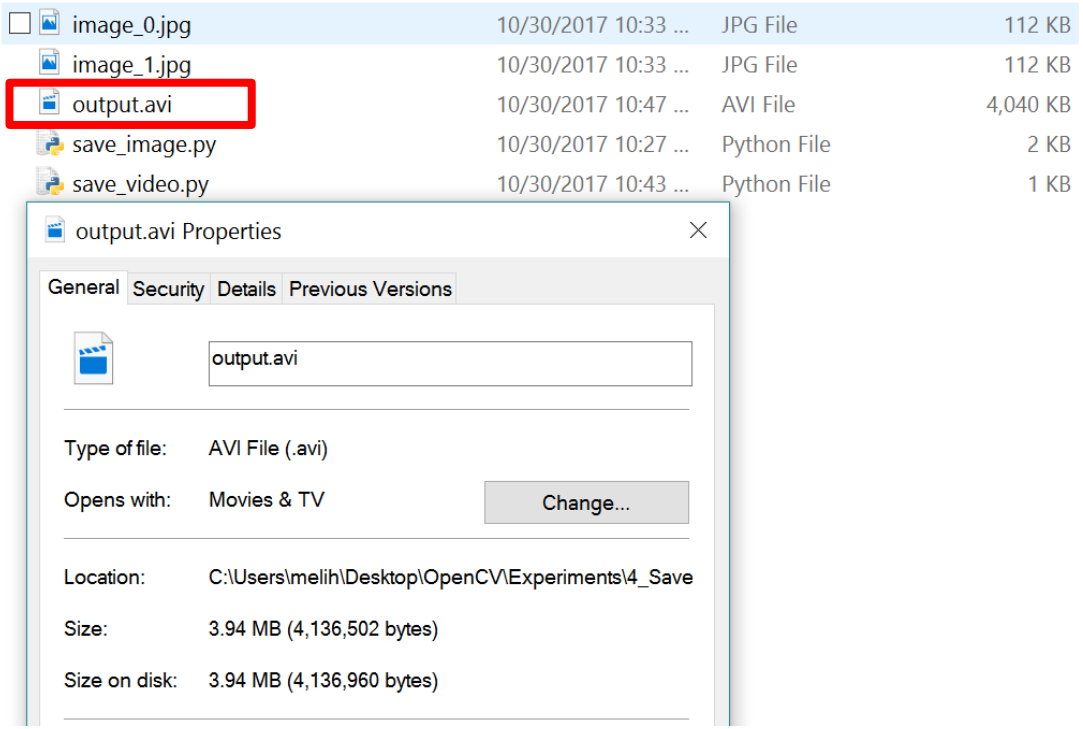
Parameters:
- **writer** – Video writer structure (OpenCV 1.x API)
- **image** – The written frame

Saving Videos

```
import cv2

cap = cv2.VideoCapture(0)

# Define the codec and create VideoWriter object
# fourcc: 4-character code of codec
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi',fourcc, 20.0, (640,480))

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

        # write the frame
        out.write(frame)

        cv2.imshow('my_video',frame)

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

# Release everything if job is finished
cap.release()
out.release()
cv2.destroyAllWindows()
```

Saving Videos





Portland State UNIVERSITY

OpenCV User Interface

- OpenCV has a few but useful user interface tools.
- Especially, they are helpful when we test different values with different functions.

  - Trackbar
  - Mouse
  - Window

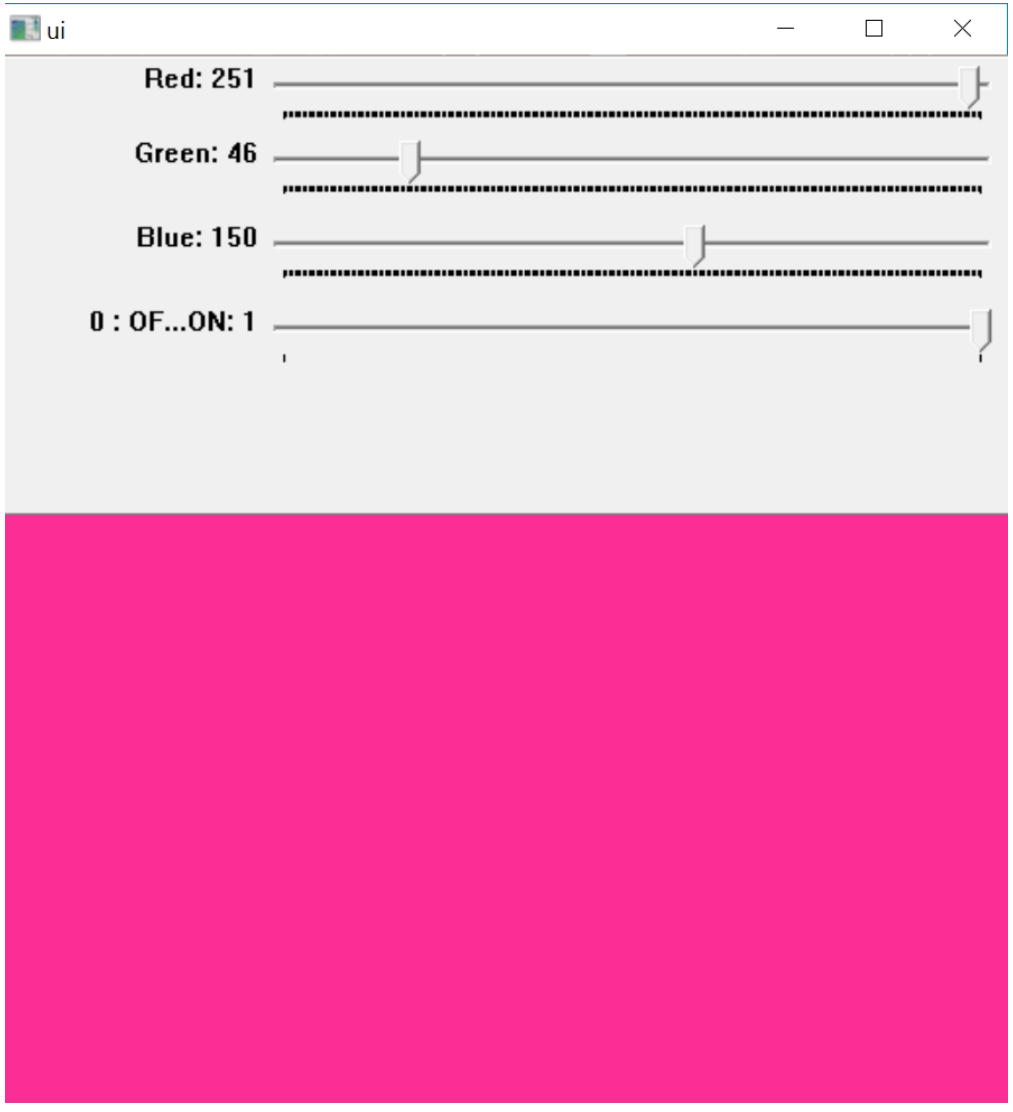
OpenCV User Interface - Trackbar

```
import cv2
import numpy as np

def nothing(x):
    pass

# Create a black image and a window
img = np.zeros((300,512,3), np.uint8)
cv2.namedWindow('ui')

#resize and move the window
cv2.moveWindow('ui', 0, 0)
cv2.resizeWindow('ui', 512, 512)

# create trackbars for color change
cv2.createTrackbar('Red','ui',0,255,nothing)
cv2.createTrackbar('Green','ui',0,255,nothing)
cv2.createTrackbar('Blue','ui',0,255,nothing)

# set trackbar values
cv2.setTrackbarPos('Red','ui', 123)
cv2.setTrackbarPos('Green','ui', 123)
cv2.setTrackbarPos('Blue','ui', 123)
```

```
# create a switch for ON/OFF functionality
switch = '0 : OFF \n1 : ON'
cv2.createTrackbar(switch, 'ui',0,1,nothing)

while(1):
    cv2.imshow('ui',img)
    k = cv2.waitKey(1) & 0xFF

    # Esc key to break
    if k == 27:
        break

    # get current positions of four trackbars
    r = cv2.getTrackbarPos('Red','ui')
    g = cv2.getTrackbarPos('Green','ui')
    b = cv2.getTrackbarPos('Blue','ui')
    s = cv2.getTrackbarPos(switch,'ui')

    if s == 0:
        img[:] = 0
    else:
        img[:] = [b,g,r]

cv2.destroyAllWindows()
```

Portland State UNIVERSITY

OpenCV User Interface - Trackbar



Portland State UNIVERSITY

OpenCV User Interface - Mouse

```
import cv2
import numpy as np

drawing = False # true if mouse is pressed
mode = True # if True, draw rectangle. Press 'm' to toggle to curve
ix,iy = -1,-1

# mouse callback function
def draw_circle(event,x,y,flags,param):
    global ix,iy,drawing,mode

    # if left button is clicked enable drawing
    if event == cv2.EVENT_LBUTTONDOWN:
        drawing = True
        ix,iy = x,y

    # if left button is clicked disable drawing
    if event == cv2.EVENT_RBUTTONDOWN:
        drawing = False
        ix,iy = x,y

    # if drawing is enabled draw black circles
    if event == cv2.EVENT_MOUSEMOVE:
        if drawing == True:
            cv2.circle(img,(x,y),2,(0,0,0),-1)
```

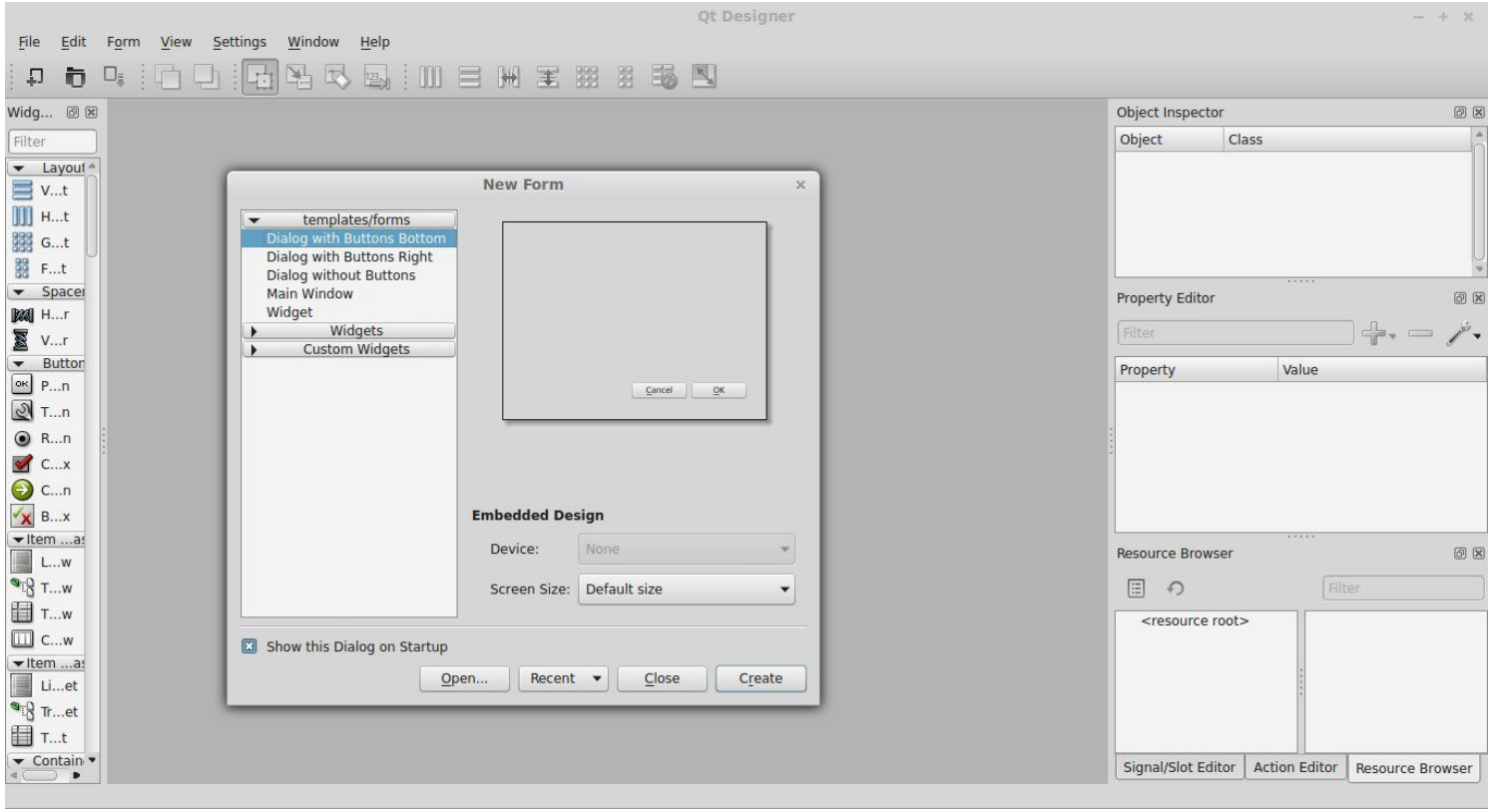
```
cv2.namedWindow('whiteboard')
cv2.setMouseCallback('whiteboard',draw_circle)

while(1):
    cv2.imshow('whiteboard',img)
    k = cv2.waitKey(1) & 0xFF
    if k == ord('m'):
        mode = not mode
    elif k == 27:
        break

cv2.destroyAllWindows()
```

Portland State UNIVERSITY

OpenCV User Interface - Mouse

Python User Interface Libraries

- **Top 7 Python GUI -** https://insights.dice.com/2017/08/07/7-top-python-gui-frameworks-for-2017-2/



PyQt

Portland State UNIVERSITY

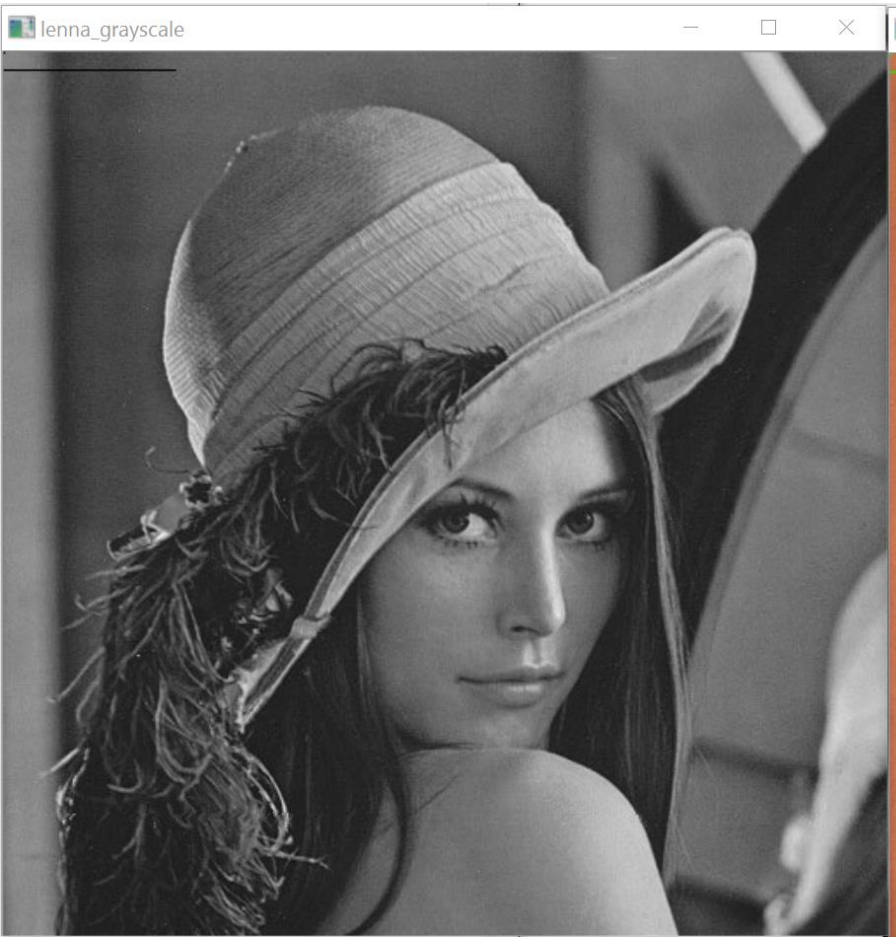
Accessing and Setting a pixel value

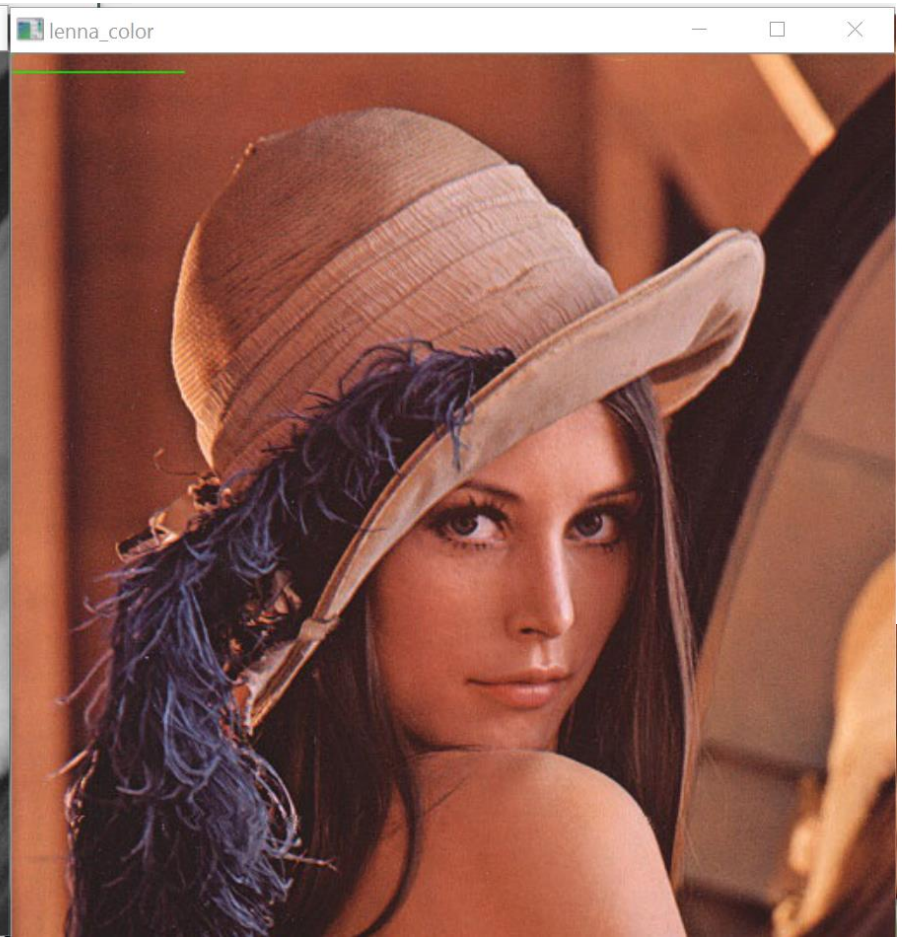
```
import cv2

# = 0: loads the image as a grayscale image
# < 0: loads the image as is
# > 0: loads the image in the BGR format

grayscale = cv2.imread('lenna.jpg',0)
colorBgr = cv2.imread('lenna.jpg',-1)

cv2.imshow('lenna_grayscale',grayscale)
cv2.imshow('lenna_color',colorBgr)

# print images before any changes happen
print ("")
print ("grayscale image matrix: " + str(grayscale))
print ("")
print ("grayscale first pixel: " + str(grayscale[0][0]))

print ("")
print ("colorBgr image matrix: " + str(colorBgr))
print ("")
print ("colorBgr first pixel: " + str(colorBgr[0][0]))

# change the color value of the first pixels
grayscale[0][0] = 0
print ("")
print ("first pixel new value: " + str(grayscale[0][0]))

colorBgr[0][0] = (0, 255, 0)
print ("")
print ("first pixel new value: " + str(colorBgr[0][0]))

#change the color value of first 100 pixels in row 10
for i in range (0,100):
    grayscale[10][i] = 0
```

```
for i in range (0,100):
    # (B, G, R)
    colorBgr [10,i] = (0, 255, 0)

# display the results
cv2.imshow('lenna_grayscale',grayscale)
cv2.imshow('lenna_color',colorBgr)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Portland State UNIVERSITY

Accessing and Setting a pixel value






```
grayscale image matrix: [[132 134 134 ...,  96  97  97]
 [134 133 135 ...,  97  97  96]
 [136 131 136 ...,  98  98  98]
 ...,
 [108 111 116 ...,  48  47  48]
 [107 111 113 ...,  47  47  51]
 [100 108 111 ...,  48  49  55]]

grayscale first pixel: 132

colorBgr image matrix: [[[ 75 111 195]
  [ 77 113 197]
  [ 77 113 197]
  ...,
  [ 55  78 146]
  [ 58  78 149]
  [ 58  78 149]]

 [[ 77 114 196]
  [ 76 112 196]
  [ 78 114 198]
  ...,
 colorBgr first pixel: [ 75 111 195]

 first pixel new value: 0

 first pixel new value: [  0 255   0]
```

Portland State UNIVERSITY

numpy

- NumPy is the fundamental package for scientific computing with Python. It contains among other things:
  - a powerful N-dimensional array object
  - sophisticated (broadcasting) functions
  - tools for integrating C/C++ and Fortran code
  - useful linear algebra, Fourier transform, and random number capabilities
- http://www.numpy.org/
- https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

Numpy Example

```
import numpy as np

#create an 3x5 array with values between 0-15
a = np.arange(15).reshape(3, 5)

#Print the array
print ("Example array: ")
print (a)

#The dimensions of the array.
#This is a tuple of integers indicating the size of the array in each dimension.
print("")
print("The dimensions of the array: "+ str(a.shape))

#The number of axes (dimensions) of the array.
print("")
print("Dimentions: " + str(a.ndim))

#An object describing the type of the elements in the array.
print("")
print("Element type: " + str(a.dtype.name))

# The size in bytes of each element of the array.
print("")
print("Element size in byte: "+ str(a.itemsize))

# The actual elements of the array.
print("")
print("Total number of elements of the array: "+ str(a.size))
```

```
# Create an 1x3 array
b = np.array([6, 7, 8])
print("")
print(b)

# Creare an array full zeros
zero = np.zeros((3,4))
print("")
print("An array full zeros: ")
print(zero)

# Create an array full ones
one = np.ones((2,3,4), dtype=np.int16)
print("")
print("An array full ones: ")
print(one)

# Create an uninitialized array - result may vary
empty = np.empty((2,3))
print("")
print("Uninitialized array: ")
print(empty)

# Create an array of squences of numbers
squence = np.arange(10, 30, 5)
print("")
print("Squence array: " + str(squence))
```

Portland State UNIVERSITY

Numpy Example

```
Example array:
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]

The dimensions of the array: 3L 5L

Dimentions: 2

Element type: int32

Element size in byte: 4

Total number of elements of the array: 15

[6 7 8]

An array full zeros:
[[ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]]

An array full ones:
[[[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]

 [[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]]

Uninitialized array:
[[  1.91209808e-316   2.67770695e-316   2.65168866e-316]
 [  1.96940416e-316   0.00000000e+000   0.00000000e+000]]

Squence array: [10 15 20 25]
```

Portland State UNIVERSITY

Operations on Arrays

- Operations on Arraysabs
- **absdiff**
- **add**
- addWeighted
- **bitwise_and**
- **bitwise_not**
- **bitwise_or**
- **bitwise_xor**
- calcCovarMatrix
- cartToPolar
- checkRange
- **compare**
- completeSymm
- convertScaleAbs
- countNonZero
- cvarrToMat
- dct
- dft
- **divide**
- Determinant
- **eigen**
- exp
- extractImageCOI
- insertImageCOI
- flip
- gemm
- getConvertElem
- getOptimalDFTSize
- Idct
- idft
- inRange
- idft
- inRange
- invert
- log
- LUT
- magnitude
- Mahalanobis
- max
- **mean**
- meanStdDev
- merge
- min
- minMaxIdx
- minMaxLoc
- mixChannels
- mulSpectrums
- **multiply**
- mulTransposed
- norm
- normalize
- PCA
- PCA::PCA
- PCA::operator ()
- PCA::project
- PCA::backProject
- perspectiveTransform
- phase
- polarToCart
- pow
- RNG
- RNG::RNG
- RNG::next
- RNG::operator T
- RNG::operator ()
- RNG::uniform
- RNG::gaussian
- RNG::fill
- randu
- randn
- randShuffle
- reduce
- repeat
- scaleAdd
- setIdentity
- solve
- solveCubic
- solvePoly
- **sort**
- sortIdx
- split
- sqrt
- subtract
- SVD
- SVD::SVD
- SVD::operator ()
- SVD::compute
- SVD::solveZ
- SVD::backSubst
- **sum**
- theRNG
- trace
- transform
- **transpose**

Portland State UNIVERSITY

Operations on Arrays

- **absdiff**
  - Calculates the per-element absolute difference between two arrays or between an array and a scalar.
  - cv2.**absdiff**(src1, src2, dst - value)

Parameters:

- **src1** – first input array or a scalar.
- **src2** – second input array or a scalar.
- **src** – single input array.
- **value** – scalar value.
- **dst** – output array that has the same size and type as input arrays.

Operations on Arrays

- **add**
  - Calculates the per-element sum of two arrays or an array and a scalar.
  - cv2.**add**(src1, src2, dst, mask, dtype)

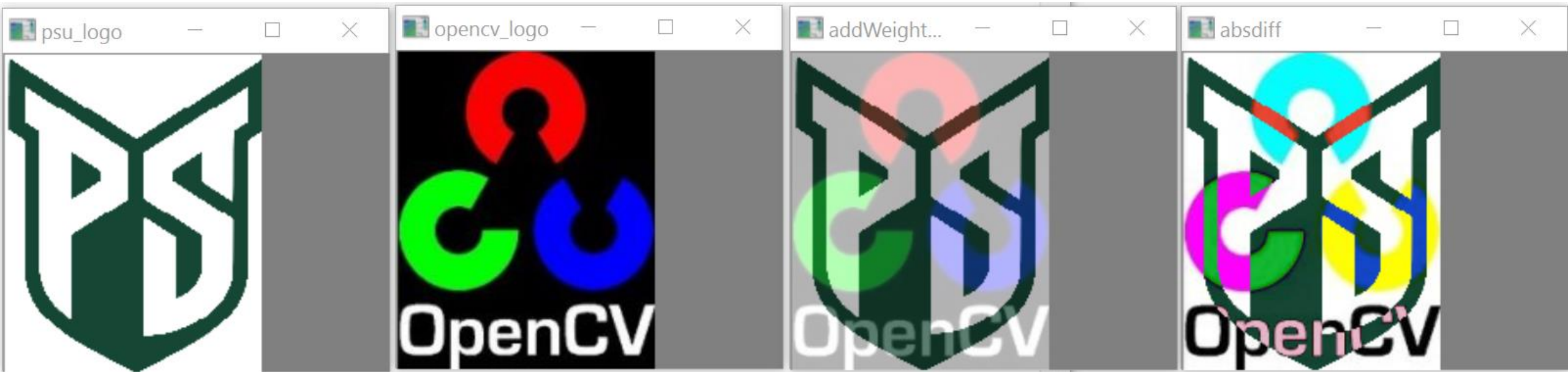
•**src1** – first input array or a scalar.
•**src2** – second input array or a scalar.
•**src** – single input array.
•**value** – scalar value.

Parameters: •**dst** – output array that has the same size and number of channels as the input array(s); the depth is defined by dtype or src1/src2.
•**mask** – optional operation mask - 8-bit single channel array, that specifies elements of the output array to be changed.
•**dtype** – optional depth of the output array (see the discussion below).

Operations on Arrays

```
import cv2
import numpy as np

# Load and display original images
img1 = cv2.imread('psu.png')
img2 = cv2.imread('opencv.jpg')
cv2.imshow('psu_logo',img1)
cv2.imshow('opencv_logo',img2)

# add and absdiff operations
result1 = cv2.addWeighted(img1,0.7,img2,0.3,0)
result2 = cv2.absdiff(img1,img2)

#show results
cv2.imshow('addWeighted',result1)
cv2.imshow('absdiff',result2)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Operations on Arrays



Operations on Arrays – bitwise operations

- **bitwise_and**: Calculates the per-element bit-wise conjunction of two arrays or an array and a scalar.
  - cv2.**bitwise_and**(src1, src2, dst, mask)

- **bitwise:_not**: Inverts every bit of an array.
  - cv2. **bitwise_not**(src1, src2, dst, mask=None)

- **bitwise_or:** Calculates the per-element bit-wise disjunction of two arrays or an array and a scalar.
  - cv2.**bitwise_or**(src1, src2, dst, mask)

- **bitwise_xor:** Calculates the per-element bit-wise “exclusive or” operation on two arrays or an array and a scalar.
  - cv2.**bitwise_xor**(src1, src2[, dst[, mask]])

Operations on Arrays – bitwise operations

```
import cv2
import numpy as np

x = np.array([1, 0, 1])
print ("x: ", x)
y = np.array([1, 0, 0])
print ("y: ", y)
z1 = cv2.bitwise_and(x,y)
print ("x and y: ")
print(z1)
z2 = cv2.bitwise_or(x,y)
print ("x or y: ")
print(z2)
z3 = cv2.bitwise_xor(x, y)
print ("x xor: ")
print(z3)

cv2.imshow('dst',dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
-
('x: ', array([1, 0, 1]))
('y: ', array([1, 0, 0]))
x and y:
[[1]
 [0]
 [0]]
x or y:
[[1]
 [0]
 [1]]
x xor:
[[0]
 [0]
 [1]]
```

Portland State UNIVERSITY

Drawing Functions

Drawing functions work with matrices/images of arbitrary depth. The boundaries of the shapes can be rendered with antialiasing (implemented only for 8-bit images for now). All the functions include the parameter color that uses an RGB value (that may be constructed with CV_RGB or the **Scalar_** constructor) for color images and brightness for grayscale images. For color images, the channel ordering is normally *Blue, Green, Red*.

List of OpenCV drawing functions:

- circle
- clipLine
- ellipse
- ellipse2Poly
- fillConvexPoly
- fillPoly
- LineIterator
- rectangle
- polylines
- putText
- getTextSize
- InitFont
- line
- arrowedLine

Drawing Functions - Circle

cv2.**Circle**(img, center, radius, color, thickness=1, lineType=8, shift=0) → None

Parameters:

- **img** – Image where the circle is drawn.
- **center** – Center of the circle.
- **radius** – Radius of the circle.
- **color** – Circle color.
- **thickness** – Thickness of the circle outline, if positive. Negative thickness means that a filled circle is to be drawn.
- **lineType** – Type of the circle boundary. See the line() description.
- **shift** – Number of fractional bits in the coordinates of the center and in the radius value.

Drawing Functions - Line

cv2.**line**(img, pt1, pt2, color[, thickness[, lineType[, shift]]]) → None

Parameters:

- **img** – Image.
- **pt1** – First point of the line segment.
- **pt2** – Second point of the line segment.
- **color** – Line color.
- **thickness** – Line thickness.
- **lineType** –Type of the line:
  - **8** (or omitted) - 8-connected line.
  - **4** - 4-connected line.
  - **CV_AA** - antialiased line.
- **shift** – Number of fractional bits in the point coordinates.

Drawing Functions - Rectangle

cv2.**rectangle**(img, pt1, pt2, color[, thickness[, lineType[, shift]]]) → None

Parameters:

- **img** – Image.
- **pt1** – Vertex of the rectangle.
- **pt2** – Vertex of the rectangle opposite to pt1 .
- **rec** – Alternative specification of the drawn rectangle.
- **color** – Rectangle color or brightness (grayscale image).
- **thickness** – Thickness of lines that make up the rectangle. Negative values, like CV_FILLED , mean that the function has to draw a filled rectangle.
- **lineType** – Type of the line. See the line() description.
- **shift** – Number of fractional bits in the point coordinates.

Drawing Functions - Polygon

Python: cv2.**polylines**(img, pts, isClosed, color[, thickness[, lineType[, shift]]]) → None

Parameters:

- **img** – Image.
- **pts** – Array of polygonal curves.
- **npts** – Array of polygon vertex counters.
- **ncontours** – Number of curves.
- **isClosed** – Flag indicating whether the drawn polylines are closed or not. If they are closed, the function draws a line from the last vertex of each curve to its first vertex.
- **color** – Polyline color.
- **thickness** – Thickness of the polyline edges.
- **lineType** – Type of the line segments. See the line() description.
- **shift** – Number of fractional bits in the vertex coordinates.

Drawing Functions - Text

cv2.**putText**(img, text, org, fontFace, fontScale, color[, thickness[, lineType[, bottomLeftOrigin]]]) → None

Parameters:

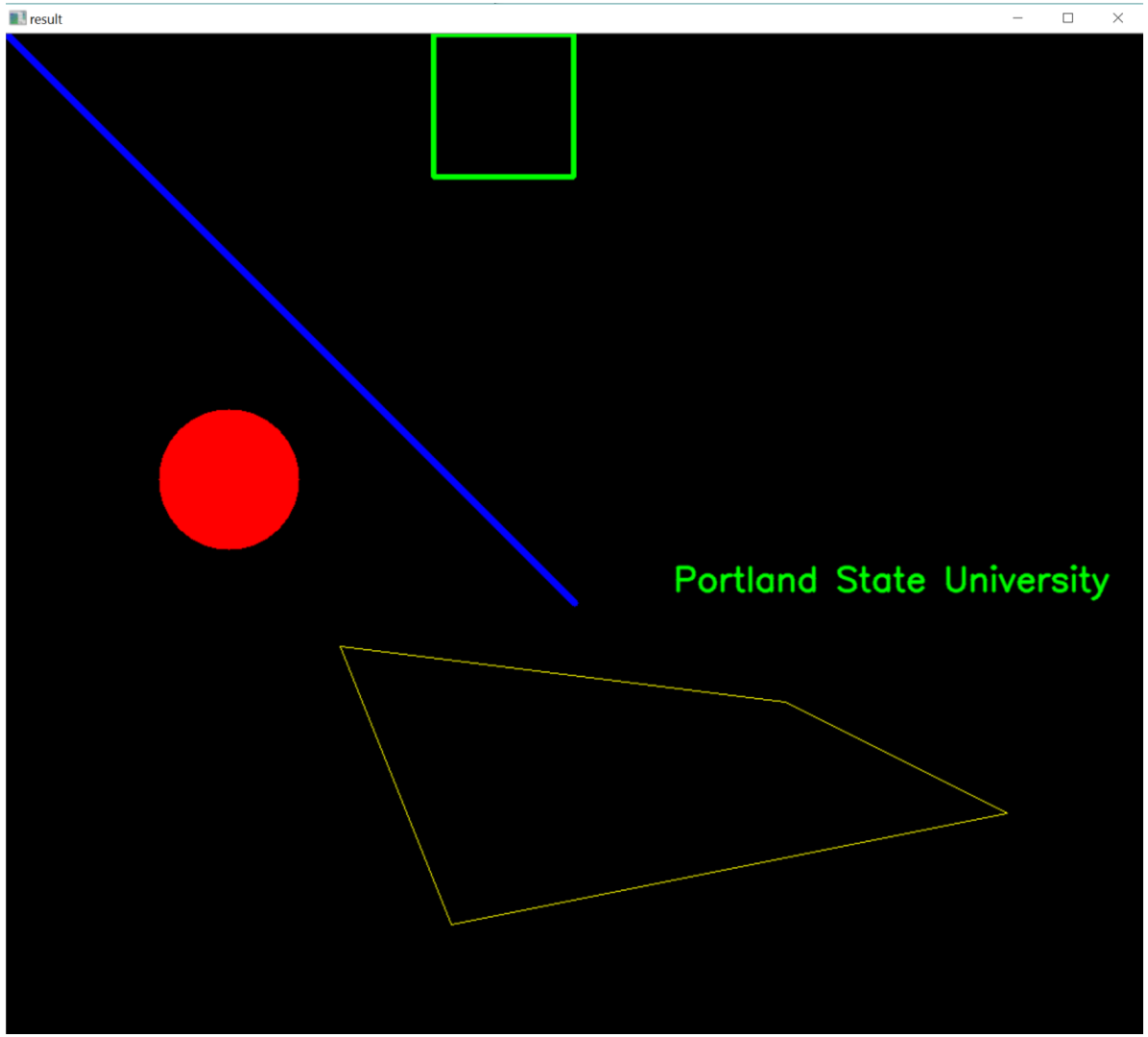
- **img** – Image.
- **text** – Text string to be drawn.
- **org** – Bottom-left corner of the text string in the image.
- **font** – CvFont structure initialized using Intron().
- **fontFace** – Font type. One of FONT_HERSHEY_SIMPLEX, FONT_HERSHEY_PLAIN, FONT_HERSHEY_DUPLEX, FONT_HERSHEY_COMPLEX, FONT_HERSHEY_TRIPLEX, FONT_HERSHEY_COMPLEX_SMALL,FONT_HERSHEY_SCRIPT_SIMPLEX, or FONT_HERSHEY_SCRIPT_COMPLEX, where each of the font ID's can be combined with FONT_ITALIC to get the slanted letters.
- **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.

Drawing Functions

```
import numpy as np
import cv2

# Create a black image
img = np.zeros((900,1024,3), np.uint8)

# Draw a diagonal blue line with thickness of 5 px
cv2.line(img,(0,0),(511,511),(255,0,0),5)

# Draw a green rectangle with thickness of 3 px
cv2.rectangle(img,(384,0),(510,128),(0,255,0),3)

# Draw a circle
cv2.circle(img,(200,400), 63, (0,0,255), -1)

# Draw a polygon
pts = np.array([[300,550],[400,800],[900,700],[700,600]], np.int32)

# Reshape gives a new shape to an array without changing its data.
pts = pts.reshape((-1,1,2))
cv2.polylines(img,[pts],True,(0,255,255))

# Add a text
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(img,'Portland State University',(600,500), font, 1,(0,255,0),2,cv2.LINE_AA)

cv2.imshow("result", img)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Portland State UNIVERSITY

Drawing Functions



Portland State
UNIVERSITY

Image Transformation Review

- Image transforms can be simple arithmetic operations on images or complex mathematical operations which convert images from one representation to another.
- **Mathematical Operations** include simple image arithmetic, Fourier, fast Hartley transform, Hough transform and Radon transform.
- **Histogram Modification** include histogram equalization and adaptive histogram equalization.
- **Image Interpolation** includes various methods for scaling, Kriging, image warping and radial aberration correction.
- **Image Registration** is a tool for registering two 2D or 3D similar images and finding an affine transformation that can be used to convert one into the other. The operation is suitable for registering medical images of the same object.
- **Background Removal** is a process to correct an image for non-uniform background or non-uniform illumination.
- **Image Rotation** is a simple tool to rotate an image about its center by the specified number of degrees.

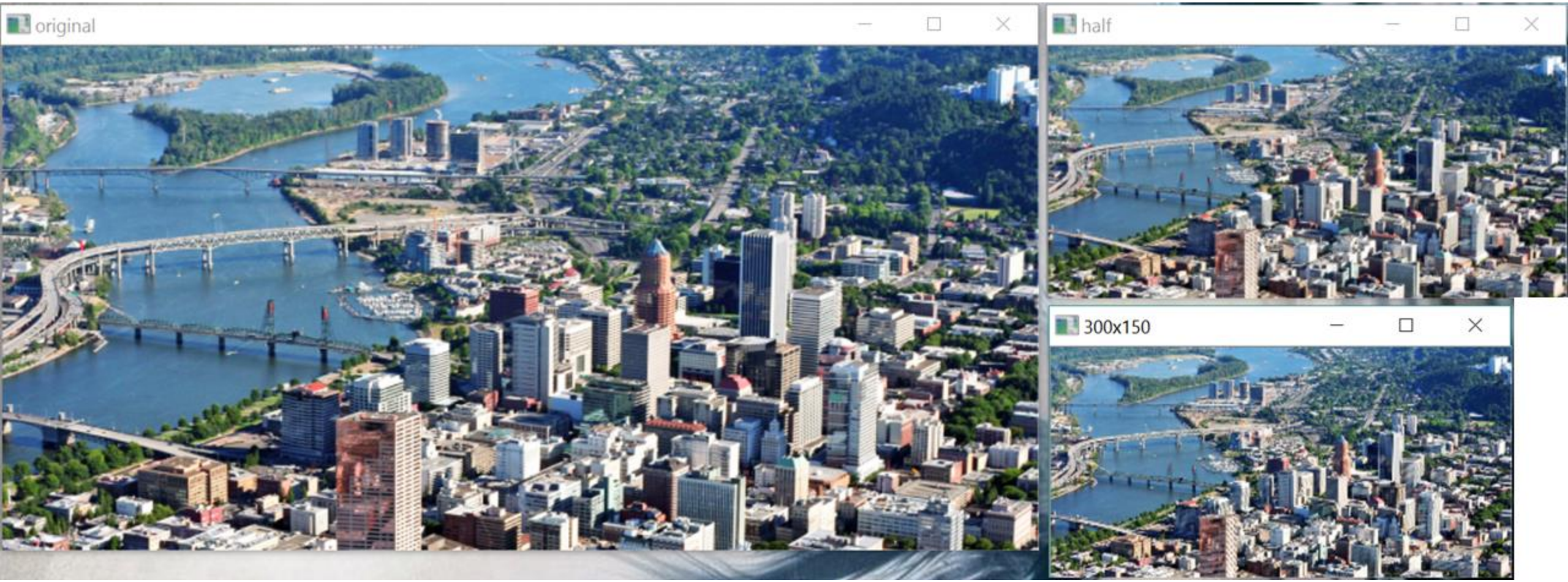
- https://www.tutorialspoint.com/dip/image_transformations.htm, https://www.mathworks.com/discovery/image-transform.html, https://www.wavemetrics.com/products/igorpro/imageprocessing/imagetransforms.htm

Portland State UNIVERSITY

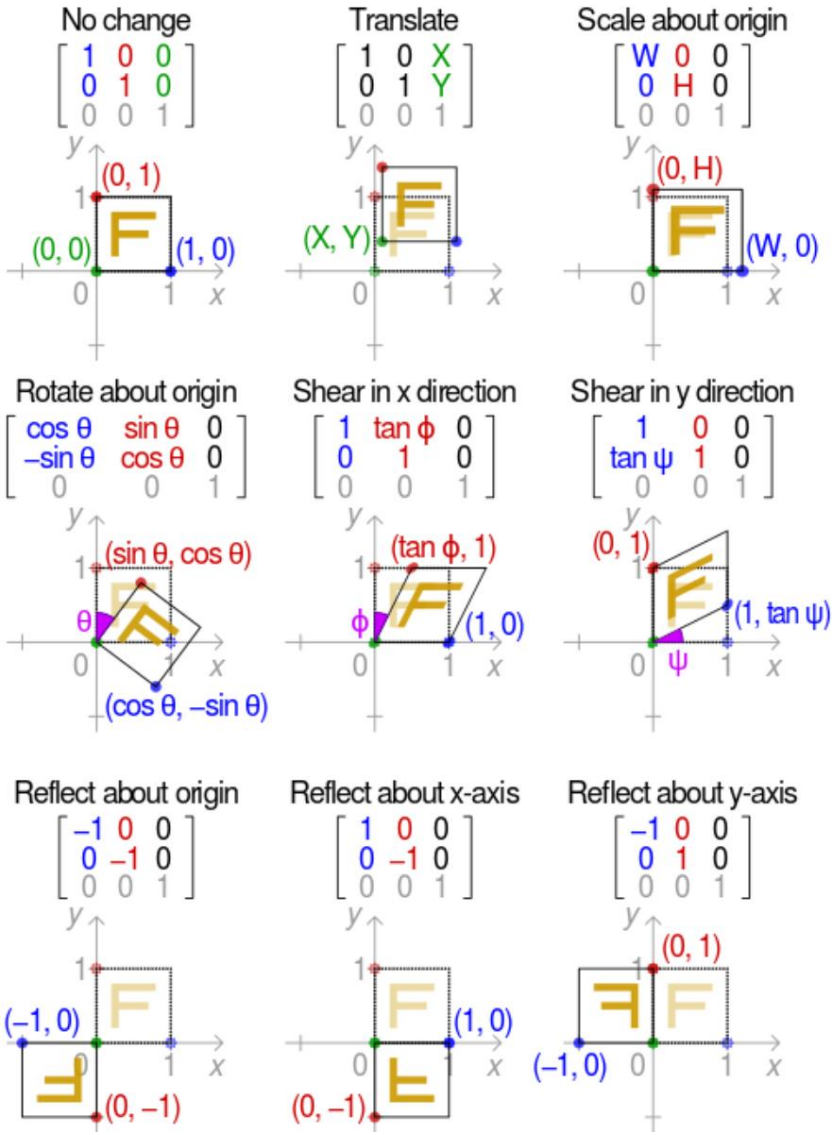
Image Transformation - Scaling

```
import cv2
import numpy as np

img = cv2.imread('portland.jpg')
cv2.imshow("original", img)

#interpolation methods:

#INTER_NEAREST - a nearest-neighbor interpolation
#INTER_LINEAR - a bilinear interpolation (used by default)
#INTER_AREA - resampling using pixel area relation. A preferred method for image decimation, as it gives moire'-free results.
#INTER_CUBIC - a bicubic interpolation over 4x4 pixel neighborhood
#INTER_LANCZOS4 - a Lanczos interpolation over 8x8 pixel neighborhood

#scale down
res = cv2.resize(img,None,fx=0.5, fy=0.5, interpolation = cv2.INTER_AREA)
cv2.imshow("half", res)

#scale up
height, width = img.shape[:2]
res = cv2.resize(img,(300, 150), interpolation = cv2.INTER_CUBIC)
cv2.imshow("300x150", res)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Image Transformation - Scaling



Transformation Matrix



[Image source]

Portland State UNIVERSITY

Image Transformation

The affine equations are expressed as

Transformation Matrix (T)

Mapped Point - q

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Point - p

An equivalent expression using matrix notation is

$$\mathbf{q} = \mathbf{T}\mathbf{p}$$

https://www.cis.rit.edu/class/simg782/lectures/lecture_02/lec782_05_02.pdf

Image Transformation - Translation

Translation is the shifting of object's location. If you know the shift in (x,y) direction, let it be (tx,ty), you can create the transformation matrix M as follows:

$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

```
#translation
rows,cols = img.shape
x =5000
y =200
translation_matrix = np.float32([[1,0,100],[0,1,50]])
dst = cv2.warpAffine(img,translation_matrix,(cols,rows))
cv2.imshow('img',dst)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/warp_affine/warp_affine.html

Image Transformation - Translation

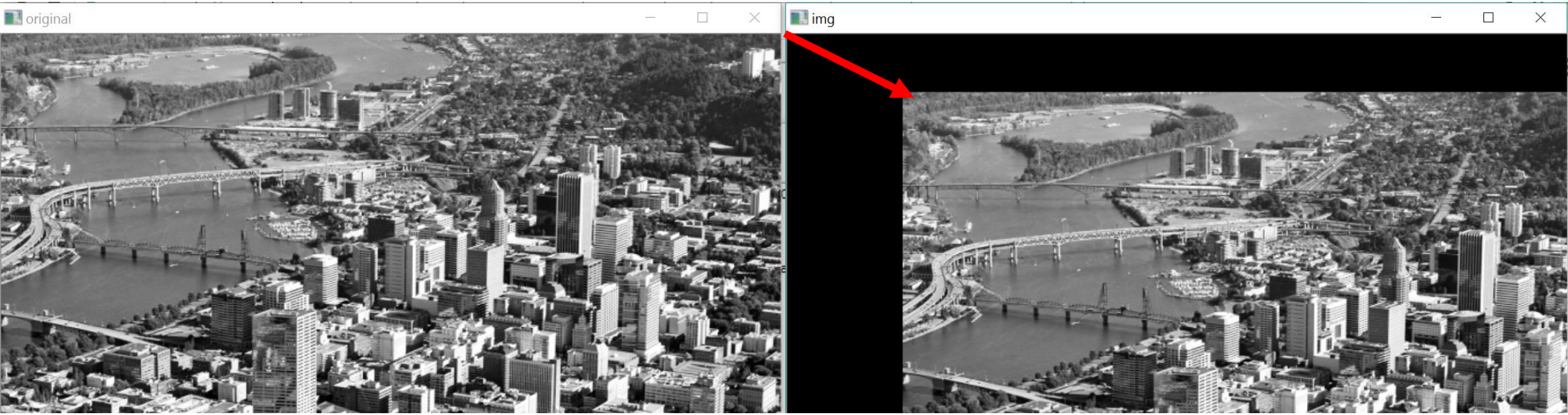


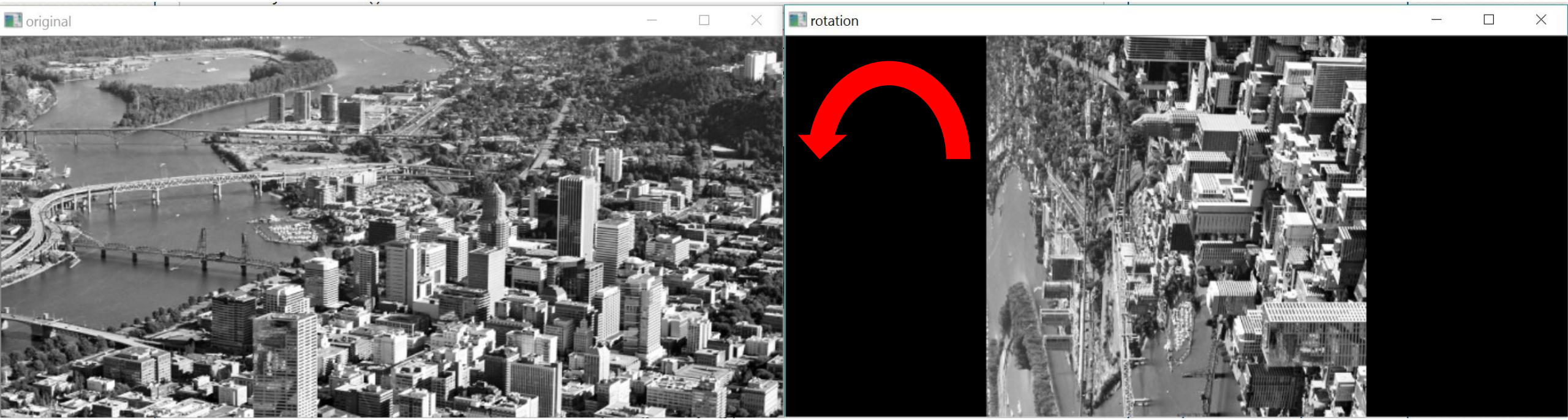
Image Transformation - Rotation

Rotation of an image for an angle Θ is achieved by the transformation matrix of the form

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

```
#rotation
rows,cols = img.shape
M = cv2.getRotationMatrix2D((cols/2,rows/2),90,1)
rotation = cv2.warpAffine(img,M,(cols,rows))
cv2.imshow('rotation',rotation)
```

Image Transformation - Rotation



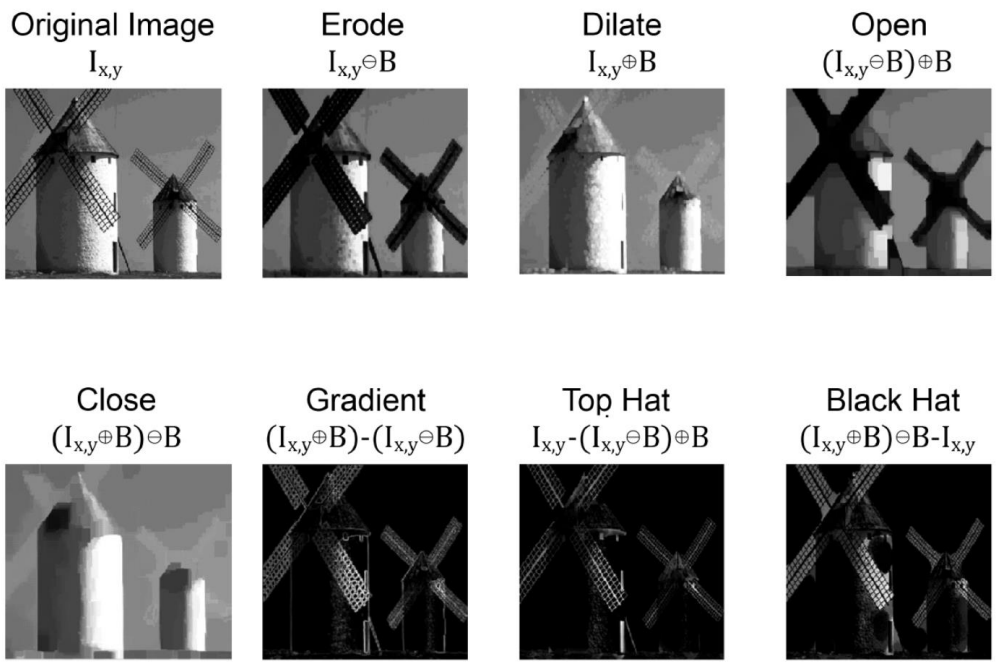
Portland State UNIVERSITY

Morphological Operations Review

- Morphological operators often take a binary image and a structuring element as input and combine them using a set operator (intersection, union, inclusion, complement).

- They process objects in the input image based on characteristics of its shape, which are encoded in the structuring element.

- Dilation
- Erosion
- Opening
- Closing
- Top Hat
- Black Hat

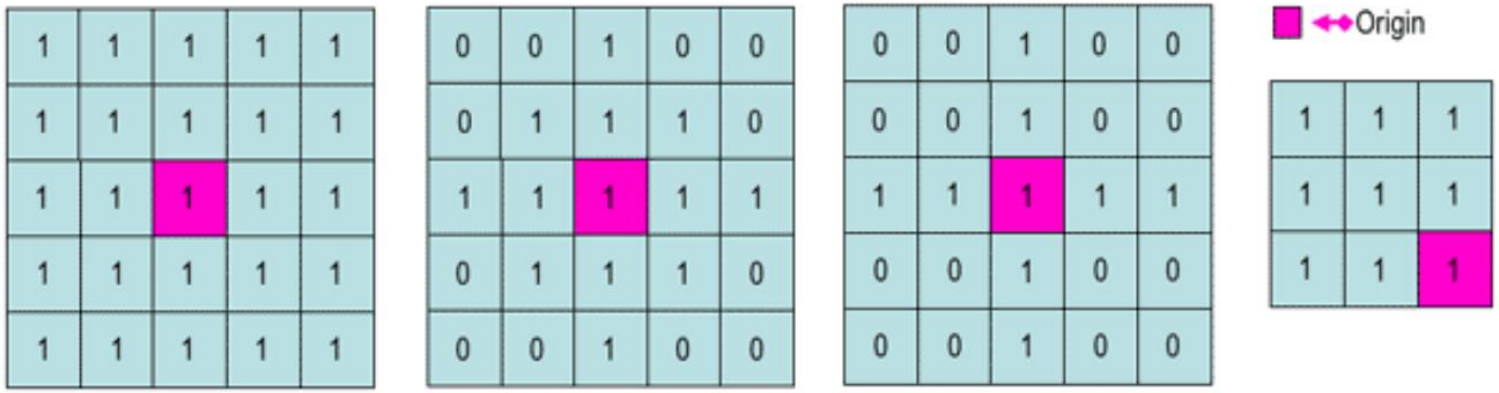


https://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm

Portland State UNIVERSITY

Basic Structuring Element

- Simply a binary image
- The matrix dimensions specify the *size* of the structuring element.
- The pattern of ones and zeros specifies the *shape* of the structuring element.
- An *origin* of the structuring element is usually one of its pixels, although generally the origin can be outside the structuring element.

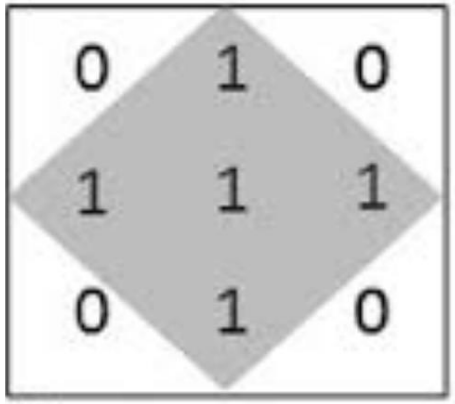


Examples of simple structuring elements.

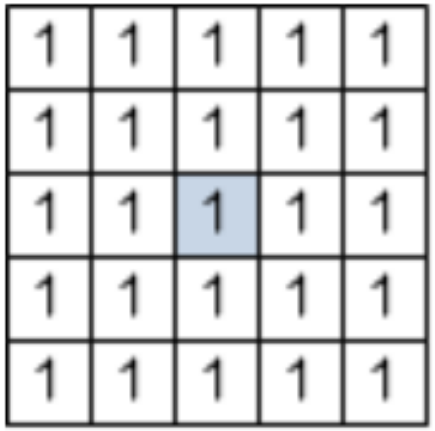
https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm

Portland State UNIVERSITY

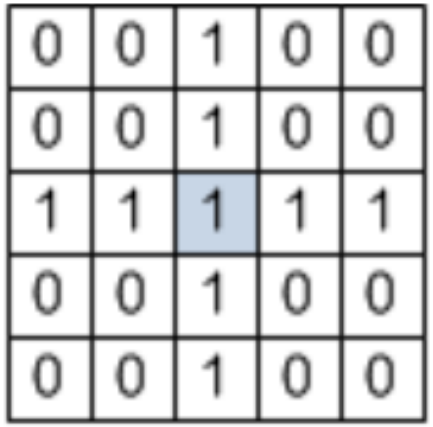
Different Shape Structuring Elements



Diamond



Square



Cross

```
[1 0 0 0 1]
[0 1 0 1 0]
[0 0 1 0 0]
[0 1 0 1 0]
[1 0 0 0 1]
```

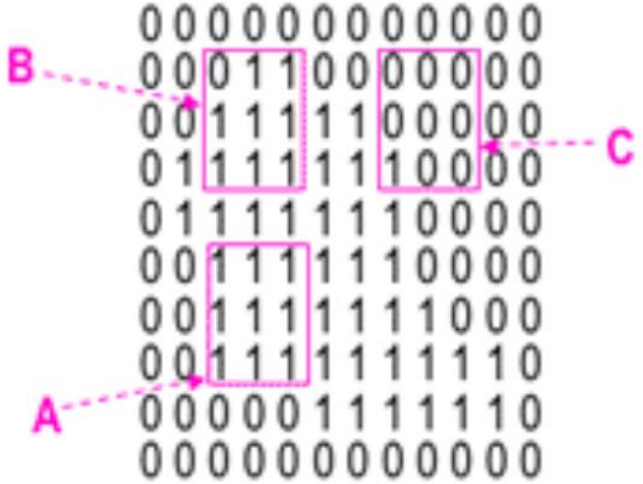
X

Portland State UNIVERSITY

Hit and Fit

When a structuring element is placed in a binary image, each of its pixels is associated with the corresponding pixel of the neighborhood under the structuring element.

- The structuring element is said to **fit** the image if, for each of its pixels set to 1, the corresponding image pixel is also 1.
- Similarly, a structuring element is said to **hit**, or intersect, an image if, at least for one of its pixels set to 1 the corresponding image pixel is also 1.



$$s_1 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

$$s_2 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

		A	B	C
fit	s_1	yes	no	no
	s_2	yes	yes	no
hit	s_1	yes	yes	yes
	s_2	yes	yes	no

Fitting and hitting of a binary image with structuring elements s_1 and s_2.

https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm