Python for Engineers

Tassadaq Hussain

Associate Professor Riphah International University

Collaborations:

Microsoft Research and Barcelona Supercomputing Center

Barcelona, Spain

European Network on High Performance and Embedded Architecture and Compilation

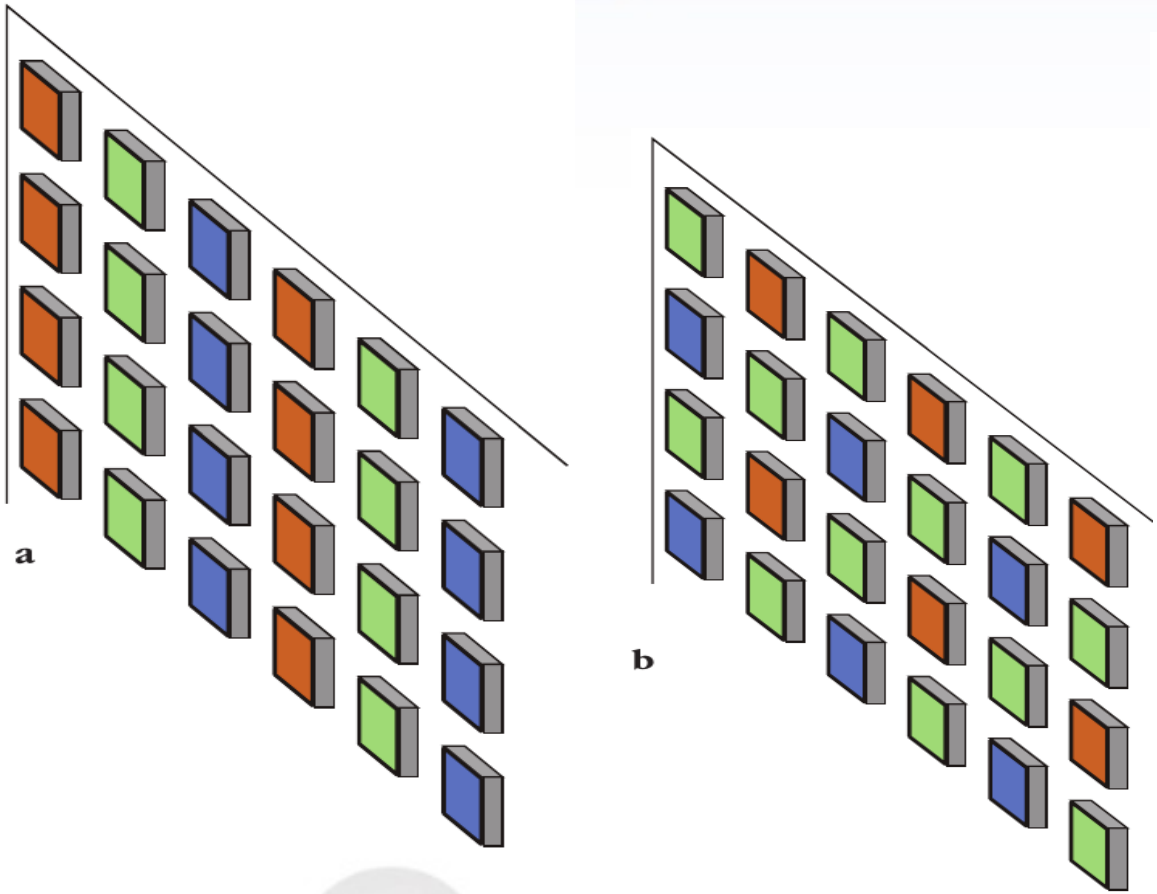
UCERD Pvt Ltd Islamabad

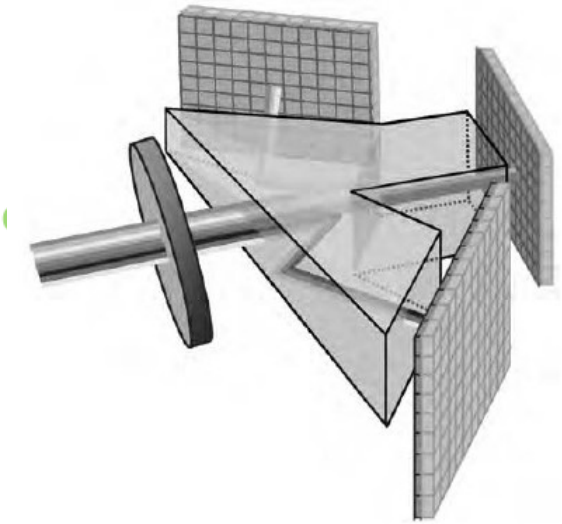
UCERD
Gathering
Intellectuals
www.ucerd.com

Agenda

- Importance of Programming Languages
- Python Language
- Python for Engineers
  - Interfacing with external world (etc)
  - RasppberyPi
  - Computer Vision
- Machine Learning

Three-chip color Camera

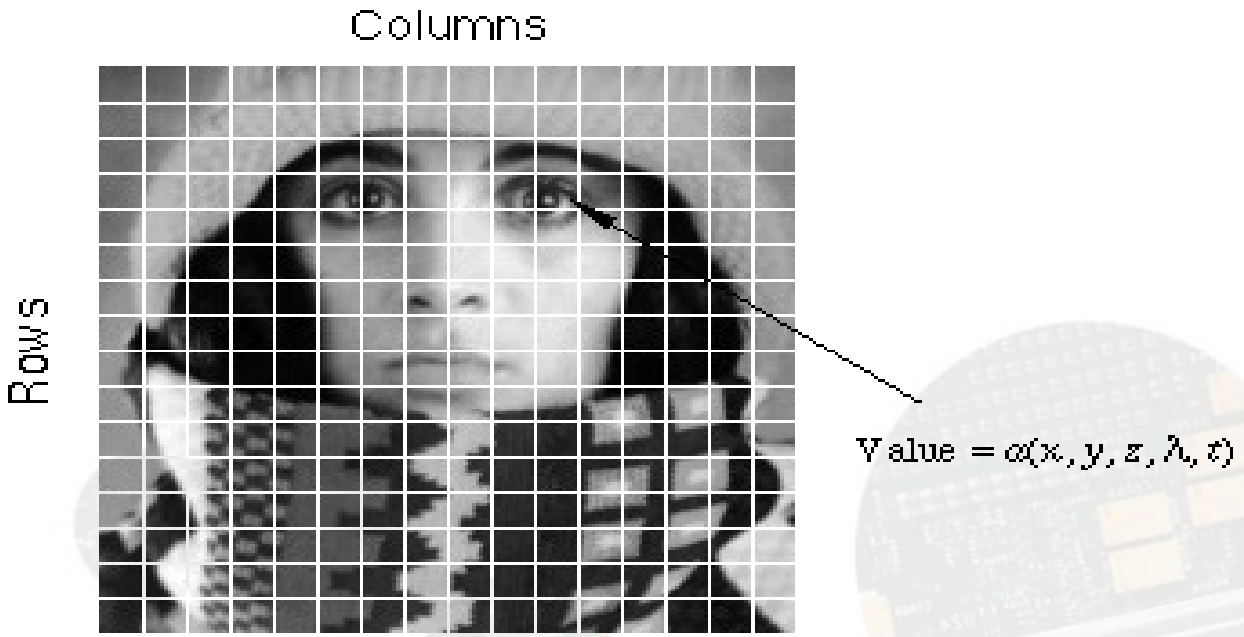




(a) Bayer (b) Filter patterns used in single chip cameras.

Color Pixel = **Red** (8bit) + **Green** (8bit) + **Blue** (8bit)

Gray scale intensity = 0.299 R + 0.587 G + 0.114 B



Pixel >> Image >> Video

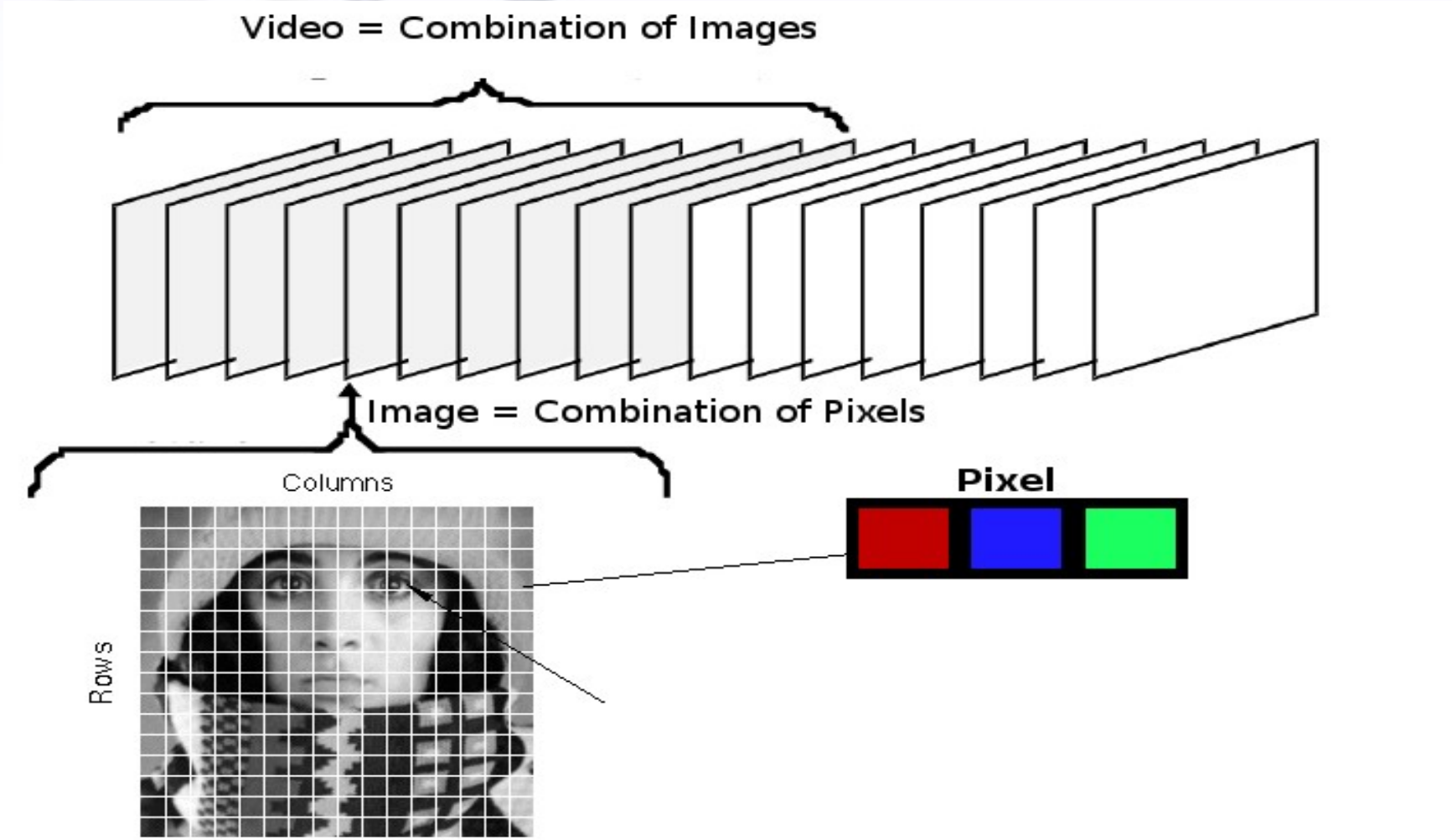


Image Resolution

(a) 256 × 256; (b) 128 × 128; (c) 64 × 64, (d)32 × 32.

Pixel Depth

Image 256x256 array pixels: (a) 32 bit (b) 16 (c) 8 (d) 4

Performance Measures

- 3 Mega Pixel Image = 3145720 pixels
- A 32 bit Processor = 3.14 million operation / sec

  Pixels = 2048 x 1536 x 24 bits/pixel
- Local Memory = 9.4 Mega Byte for single Image
- Video Processing = $3.14 \times 10^6 \times 30$ (fps)

  $= 94.2 \times 10^6$

Processor / System	Dhrystone MIPS / MIPS
Nios II	190 MIPS at 165 MHz
ARM Cortex A7	2,850 MIPS at 1.5 GHz
ARM Cortex-A9 (Dual core)	7,500 MIPS at 1.5 GHz
Raspberry Pi 2	**1186 MIPS per core at 1.0 GHz**
Nvidia Tegra 3 (Quad core Cortex-A9)	13,800 MIPS at 1.5 GHz
Intel Core 2 Extreme QX6700 (Quad core)	49,161 MIPS at 2.66 GHz
Intel Core i7 920 (Quad core)	82,300 MIPS at 2.93 GHz

Uses

3D Vision

Health-care

Security

Communication

Information

Automobile

Computation Required for Simple Thresholding

```
Read Image Pixel // I/O Operation
if(pix_value>value) // Branch Operation
pix_value=value // Assignment Operation
```

A 3 Mega Pix Image requires

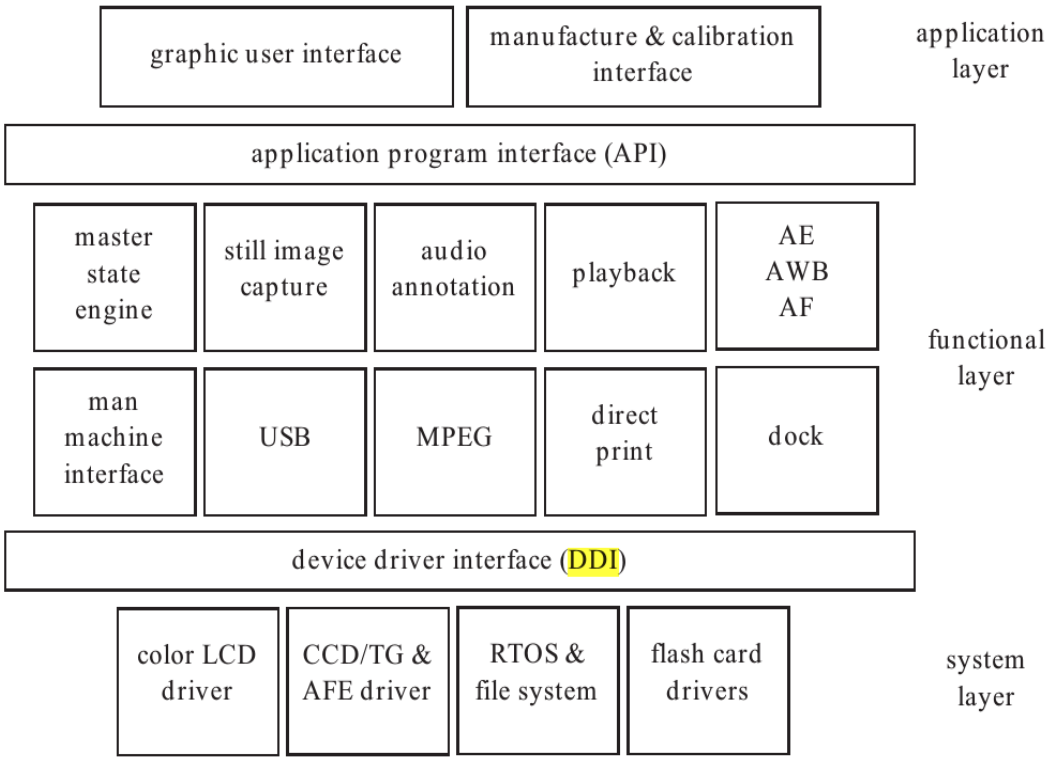
2048 x 1536 Input/Output Operations

2048 x 1536 Branch Operations

2048 x 1536 Assignment Operations

Total = 2048 x 1536 x 3 = 9 Million Operations

Software Platform of Digital Camera



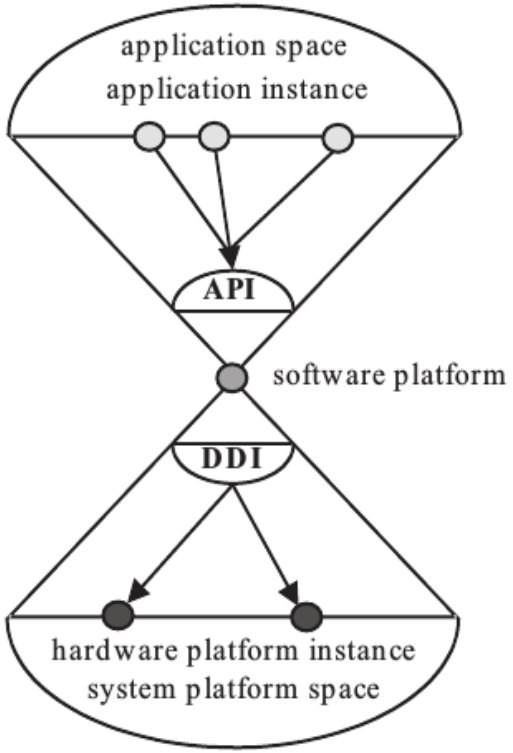
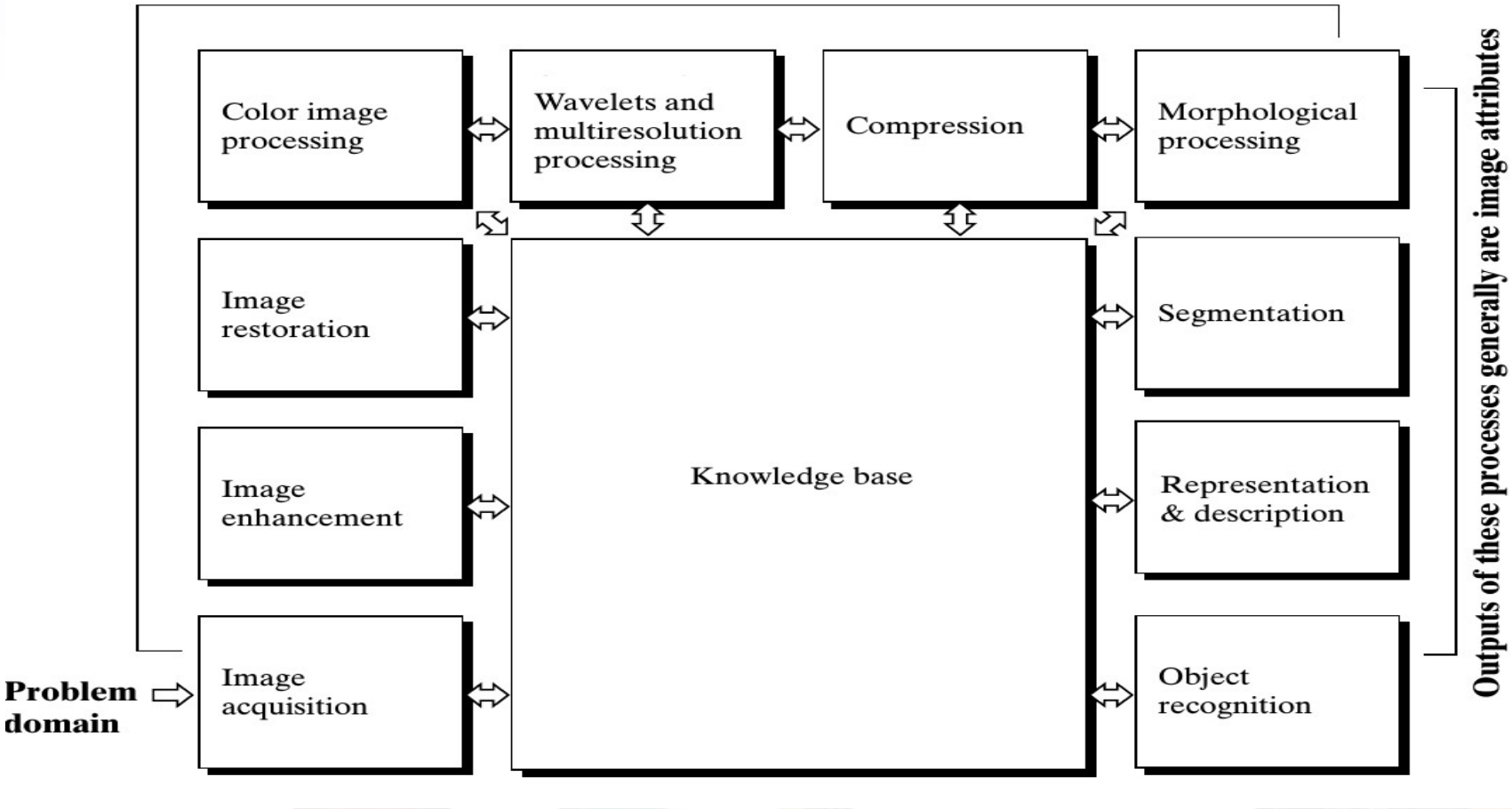


Image Processing Problems



Graphics System

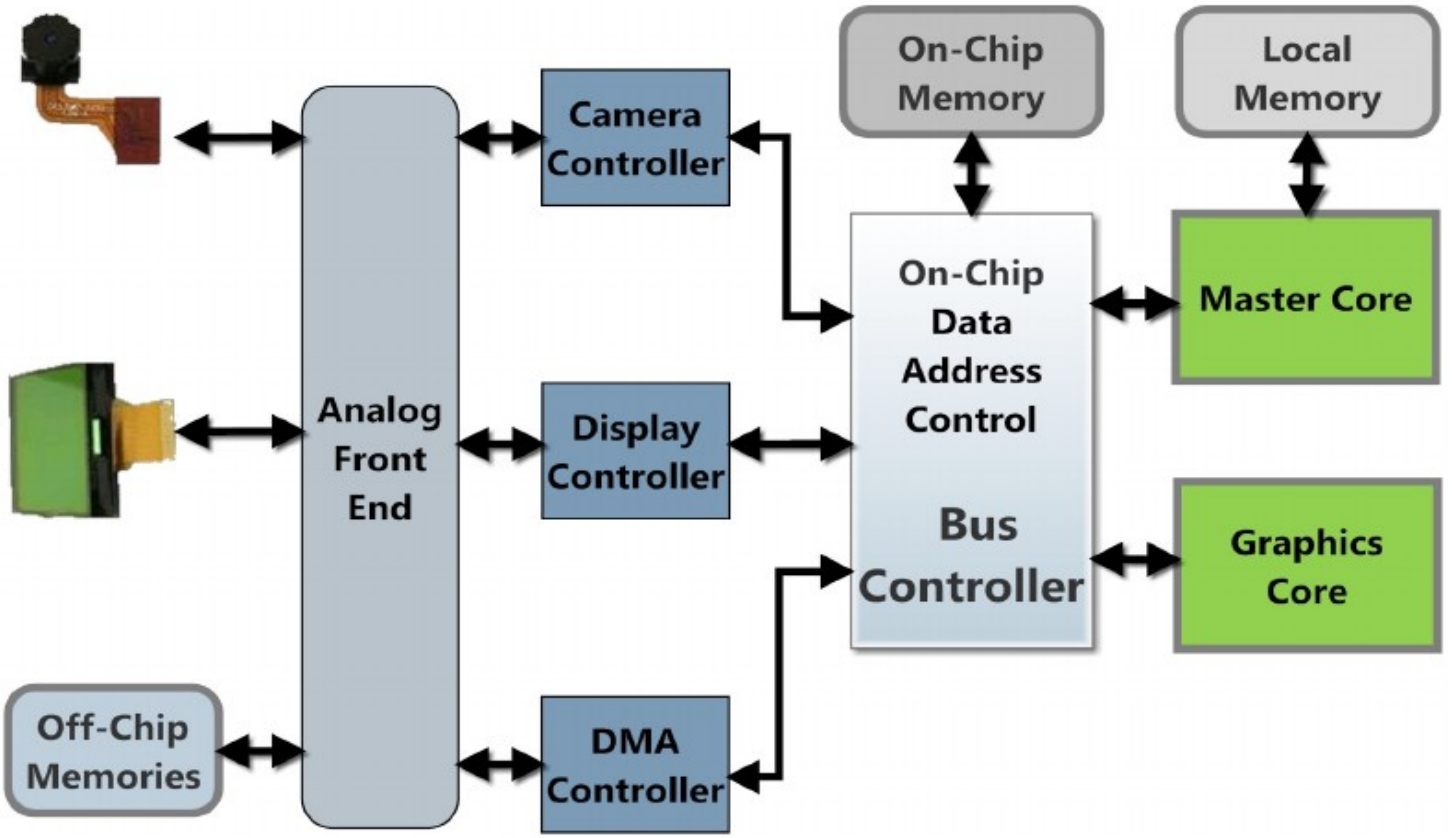


Image Processing: OpenCV

OpenCV (Open Source Computer Vision Library) is released under a BSD license and hence it's free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform.

Source: https://opencv.org/

Read and Display

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

#input handler
img = cv2.imread('./images/L1.jpg')
plt.imshow(img)
plt.show()
```

OpenCV

Intel® OPEN SOURCE COMPUTER VISION LIBRARY

UCERD
Gathering
Intellectuals
www.ucerd.com

Goals

- Develop a universal toolbox for research and development in the field of Computer Vision

OpenCV Functionality (more than 350 algorithms)

- **Basic structures and operations**
- **Image Analysis**
- **Structural Analysis**
- Object Recognition
- Motion Analysis and Object Tracking
- 3D Reconstruction

Basic Structures and Operations

- Image and Video Data Structures

Mat image;

Image = imread (“path”);

- Multidimensional array operations

include operations on images, matrices and histograms.

equalizeHist(src, dst);

- Dynamic structures operations

concern all vector data storages.

- Drawing primitives

allows not only to draw primitives but to use the algorithms for pixel access

- Utility functions

in particular, contain fast implementations of useful math functions.

Image Read/Write

- Import cv2 as cv

```
gray_img = cv2.imread('images/input.jpg', cv2.IMREAD_GRAYSCALE)
cv2.imshow('Grayscale', gray_img)
cv2.waitKey()

cv2.imwrite('images/output.jpg', gray_img)

gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
yuv_img = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
```

Image Analysis

- Thresholds

  threshold(src_gray, dst, threshold_value, max_BINARY_value, threshold_type);

- Statistics
- Pyramids
- Morphology

  Erosion , dilation etc

- Distance transform
- Feature detection

Statistics

- min, max, mean value, standard deviation over the image
- Norms C, L1, L2
- Multidimensional histograms
- Spatial moments up to order 3 (central, normalized, Hu)

Pyramids

An image pyramid is a collection of images - all arising from a single original image - that are successively downsampled until some desired stopping point is reached.

PyrUp()

pyrdown()

Gaussian pyramid:

Laplacian pyramid:

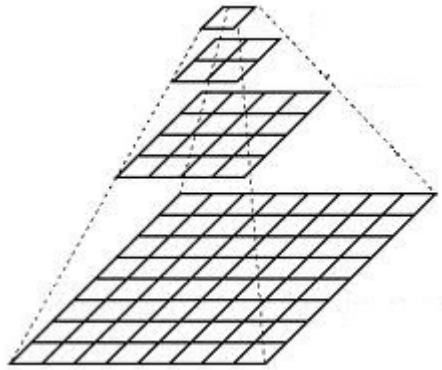
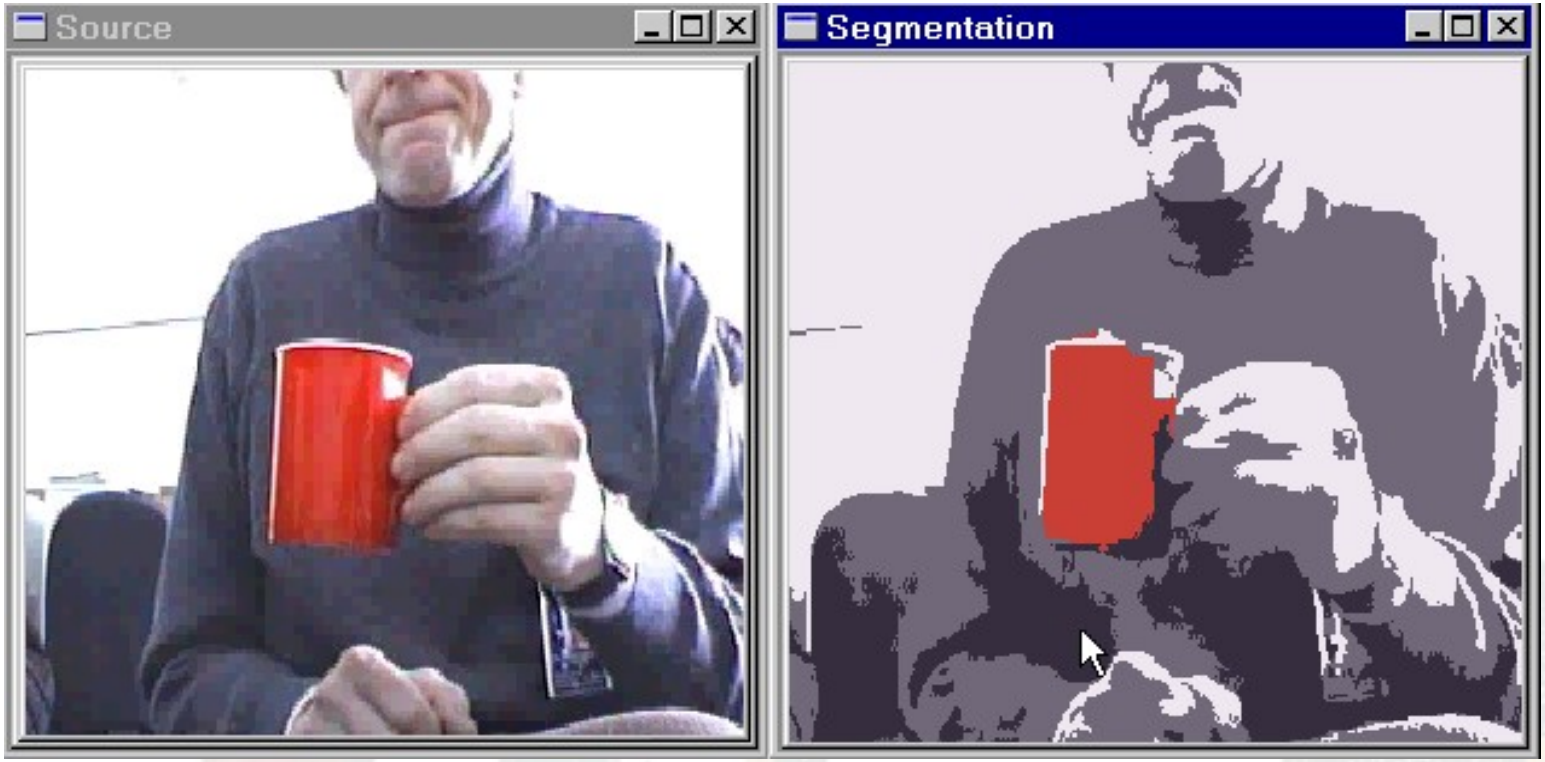


Image Pyramids

- Gaussian and Laplacian

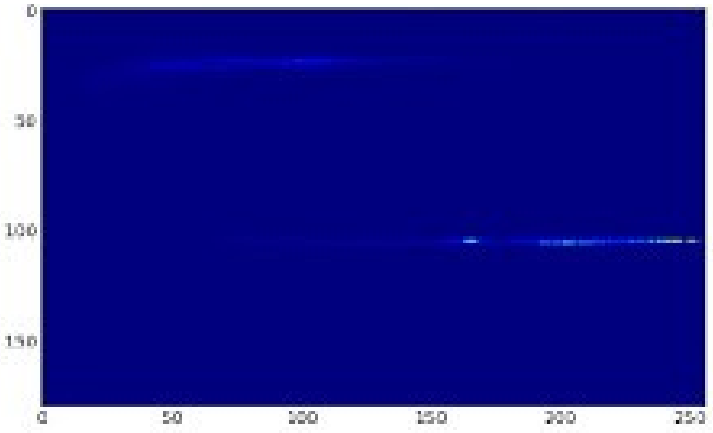
Pyramid-based color segmentation

On still pictures ***And on movies***



Multidimensional Histograms

- Histogram operations : calculation, normalization, comparison, back project
- Histograms types:
  - ✓ Dense histograms
  - ✓ Signatures (balanced tree)



Morphological Operations

- Two basic morphology operations using structuring element:
  - ✓ erosion
  - ✓ dilation
- More complex morphology operations:
  - ✓ opening
  - ✓ closing
  - ✓ morphological gradient
  - ✓ top hat
  - ✓ black hat

Morphological Operations Examples

- Morphology - applying Min-Max. Filters and its combinations

Image I

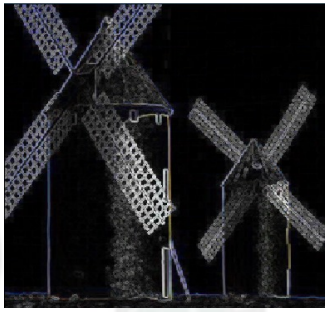
Erosion IΘB

Dilatation I⊕B

Opening IoB= (IΘB)⊕B

Closing I•B= (I⊕B)ΘB

Grad(I)= (I⊕B)-(IΘB)



TopHat(I)= I - (IΘB)

BlackHat(I)= (I⊕B) - I

Distance Transform

The distance transform operator generally takes binary images as inputs. In this operation, the gray level intensities of the points inside the foreground regions are changed to distance their respective distances from the closest 0 value (boundary).

distanceTransform()

- Calculate the distance for all non-feature points to the closest feature point
- Two-pass algorithm, 3x3 and 5x5 masks, various metrics predefined

Flood Filling

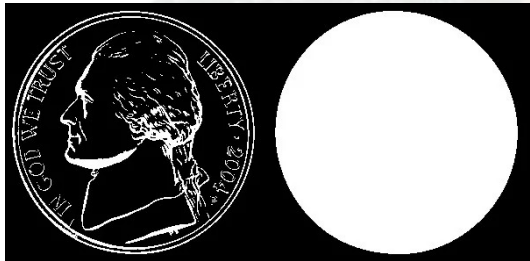
- Simple
- Gradient

```
cv2.floodFill(img, mask, (0,0), 255);
```

Original image

Tolerance interval ± 5

Tolerance interval ± 6



Feature Detection

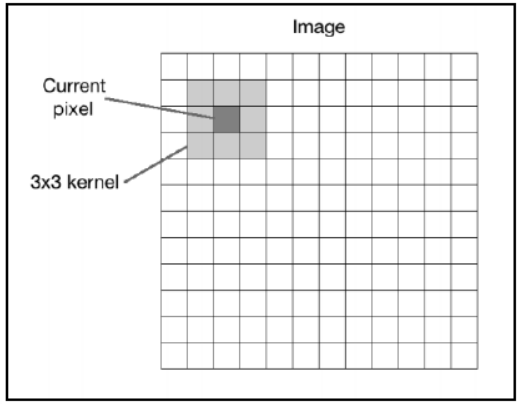
- Fixed filters (Sobel operator, Laplacian);
- Optimal filter kernels with floating point coefficients (first, second derivatives, Laplacian)
- Special feature detection (corners)
- Canny operator
- Hough transform (find lines and line segments)
- Gradient runs

Convolution

Convolution is a fundamental operation in image processing. It basically applies a mathematical operator to each pixel, and change its value in some way.

To apply this mathematical operator, convolution uses another matrix called a kernel. The kernel is usually much smaller in size than the input image. For each pixel in the image, we take the kernel and place it on top so that the center of the kernel coincides with the pixel under consideration.

We then multiply each value in the kernel matrix with the corresponding values in the image, and then sum it up. This is the new value that will be applied to this position in the output image.



```
import cv2
import numpy as np
img = cv2.imread('images/input.jpg')
rows, cols = img.shape[:2]
kernel_identity = np.array([[0,0,0], [0,1,0], [0,0,0]])
kernel_3x3 = np.ones((3,3), np.float32) / 9.0 # Divide by 9 to normalize the kernel
kernel_5x5 = np.ones((5,5), np.float32) / 25.0 # Divide by 25 to normalize the kernel
cv2.imshow('Original', img)
# value -1 is to maintain source image depth
output = cv2.filter2D(img, -1, kernel_identity)
cv2.imshow('Identity filter', output)
output = cv2.filter2D(img, -1, kernel_3x3)
cv2.imshow('3x3 filter', output)
output = cv2.filter2D(img, -1, kernel_5x5)
cv2.imshow('5x5 filter', output)
cv2.waitKey(0)
```

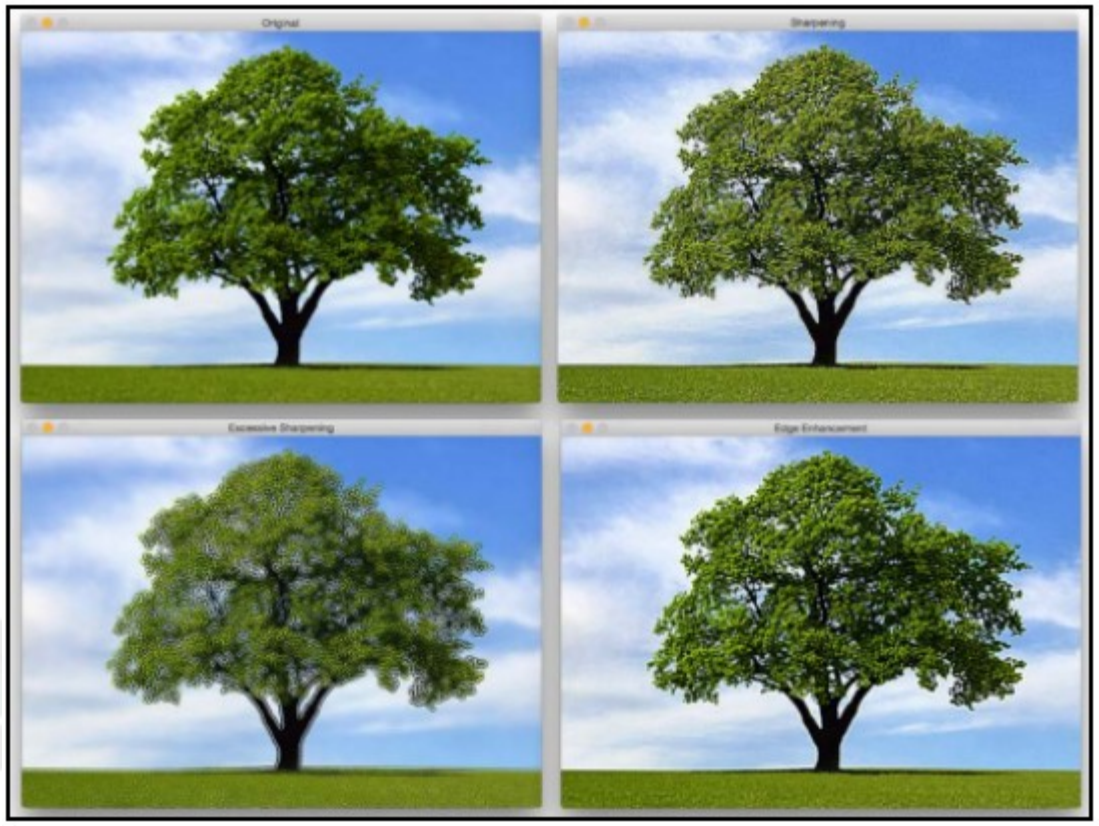
```
import cv2
from matplotlib import pyplot as plt
import numpy as np
img = cv2.imread('images/input.jpg')
cv2.imshow('Original', img)
size = 15
# generating the kernel
kernel_motion_blur = np.zeros((size, size))
kernel_motion_blur[int((size-1)/2), :] = np.ones(size)
kernel_motion_blur = kernel_motion_blur / size
# applying the kernel to the input image
output = cv2.filter2D(img, -1, kernel_motion_blur)
cv2.imshow('Motion Blur', output)
cv2.waitKey(0)
```

Sharpening Images

```
# generating the kernels
kernel_sharpen_1 = np.array([[-1,-1,-1], [-1,9,-1], [-1,-1,-1]])
kernel_sharpen_2  = np.array([[1,1,1], [1,-7,1], [1,1,1]])
kernel_sharpen_3  = np.array([[-1,-1,-1,-1,-1], [-1,2,2,2,-1], [-1,2,8,2,-1], [-1,2,2,2,-1], [-1,-1,-1,-1,-1]]) / 8.0
```

$$M = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$$M = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -7 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

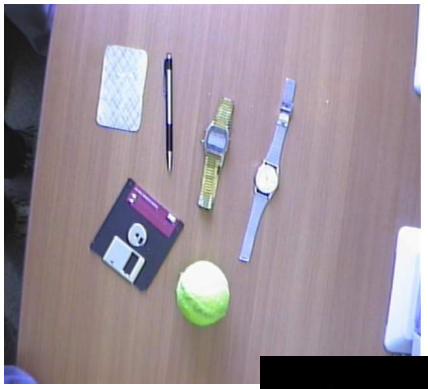


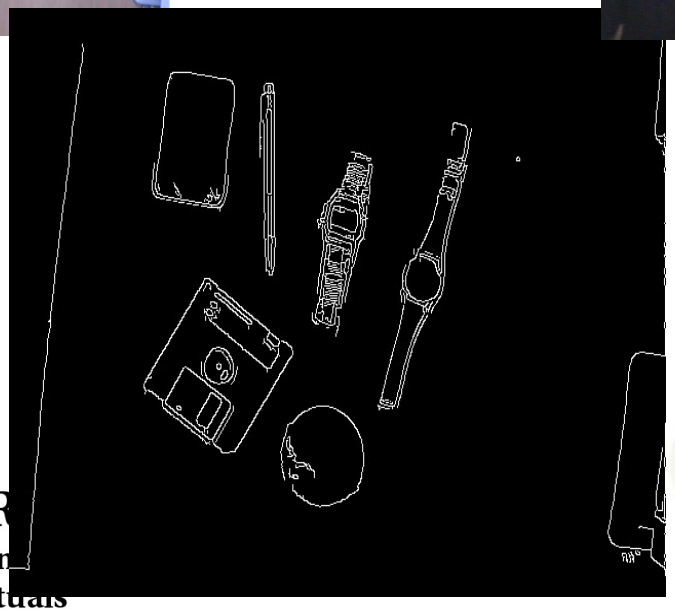
The process of edge detection involves detecting sharp edges in the image, and producing a binary image as the output. Typically, we draw white lines on a black background to indicate those edges.

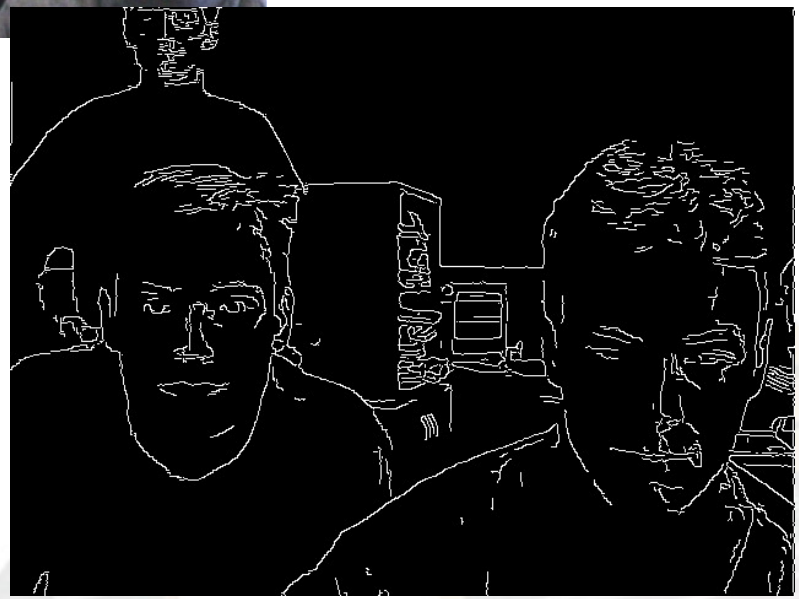
We can think of edge detection as a high pass filtering operation. A high pass filter allows high-frequency content to pass through and blocks the low-frequency content. As we discussed earlier, edges are high-frequency content. In edge detection, we want to retain these edges and discard everything else. Hence, we should build a kernel that is the equivalent of a high pass filter.

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

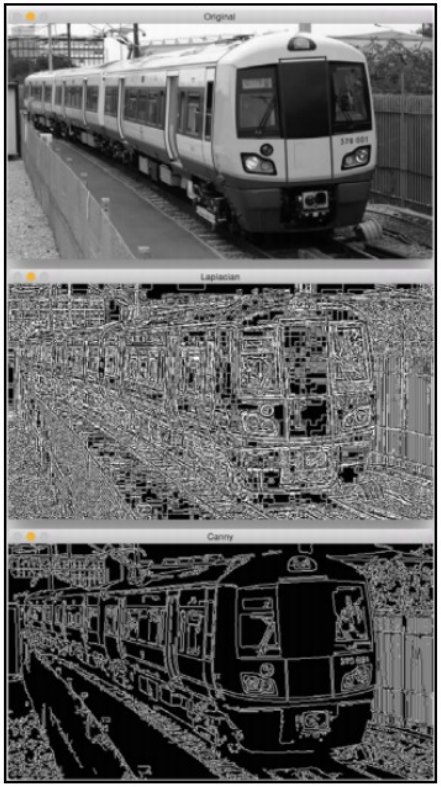
Canny Edge Detector

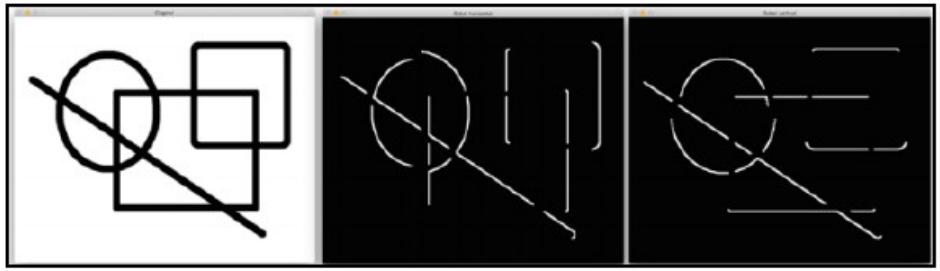






```
import cv2
import numpy as np
img = cv2.imread('images/input_shapes.png',
cv2.IMREAD_GRAYSCALE)
rows, cols = img.shape # It is used depth of
cv2.CV_64F.
sobel_horizontal = cv2.Sobel(img, cv2.CV_64F, 1, 0,
ksize=5)
# Kernel size can be: 1,3,5 or 7.
sobel_vertical = cv2.Sobel(img, cv2.CV_64F, 0, 1,
ksize=5)
cv2.imshow('Original', img)
cv2.imshow('Sobel horizontal', sobel_horizontal)
cv2.imshow('Sobel vertical', sobel_vertical)
cv2.waitKey(0)
```





Machine Learning

Example

Google Colab

Do not know how to install and set up the Python running environment;

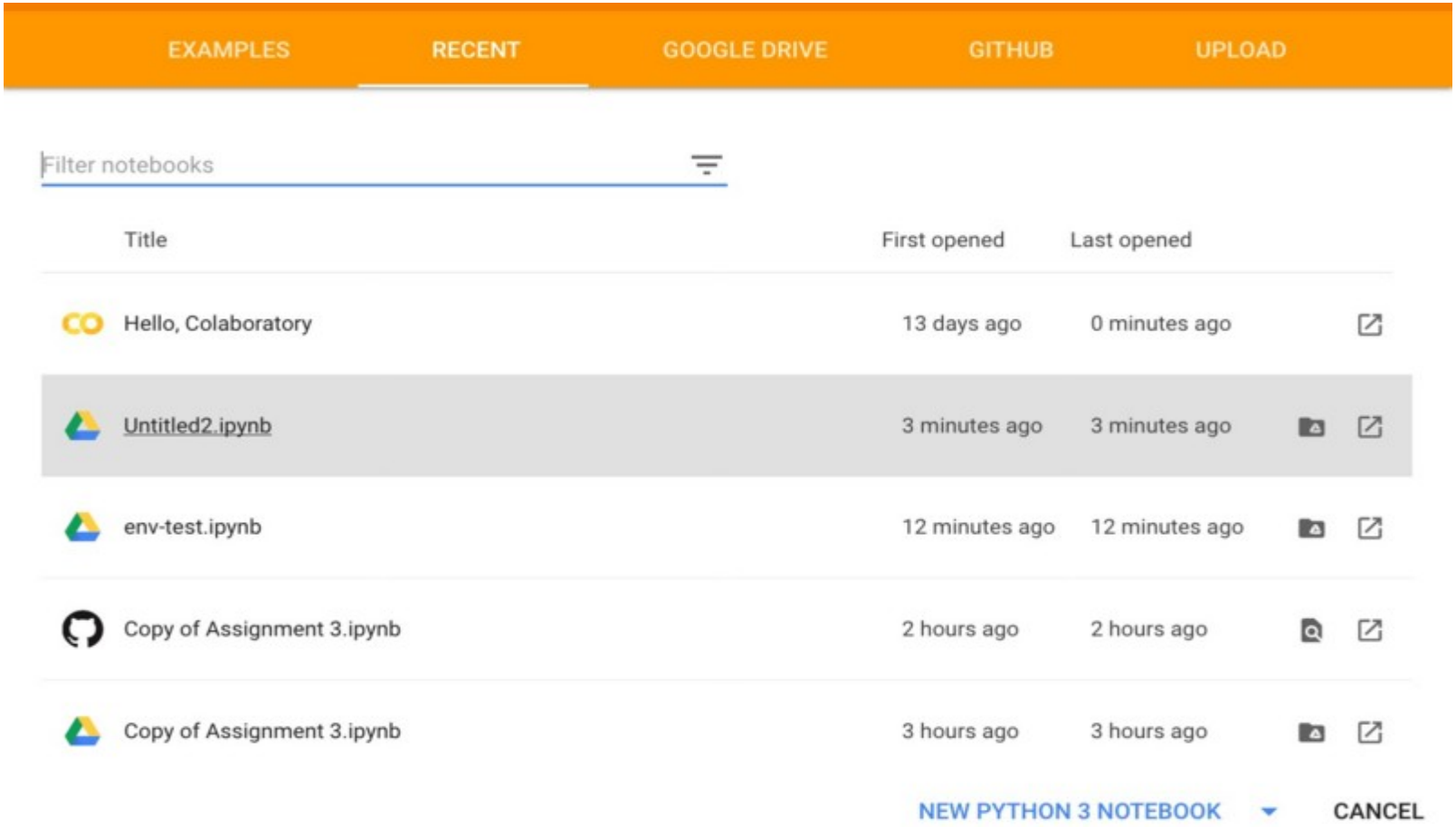
Do not know how to find the solutions effectively when facing the problems;

Do not know how to collaborate with others when trying to finish the group tasks ;

Do not know how to handle version control, which may lead the code to chaotic.

Google Colab can help you with all of those things.

https://colab.research.google.com



Colab hardware support

By default, these cloud computing hardwares are not enabled. You need to choose “runtime” in the menu bar, and then “Change runtime type”.