Simplified Guide to Computer Vision

An Introduction to OpenCV with Python

By Z. Paul Weleschuk

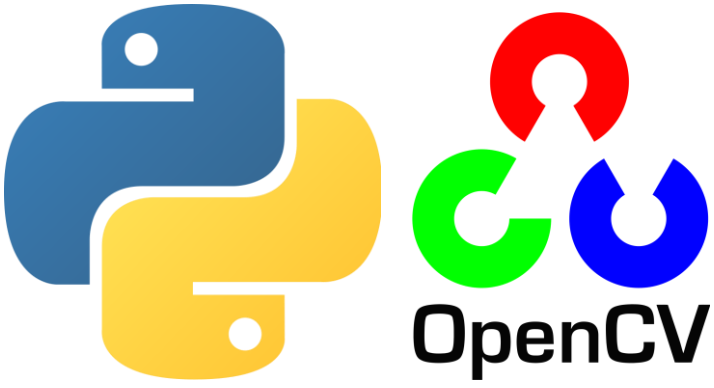


Table of Contents	Slide Number
Title Page	1
Table of Contents	2
What is Computer Vision	3
Reasons for Creating this Guide	4
Pixels	5
Grey Scale	6
Color Images	7
Basic Operations	9
Resizing an Image	10
Concatenating Images	11
Blending Two Images	12
Example: Resize, Concatenate, and Blend	13
Brightness, Contrast, and Gamma	14
Smoothing and Blurring	15
Adding Text and Shapes	16
Bitwise Operations	17
Basic Threshold	18
Adaptive Threshold	19
Erosion and Dilation	20
Matplotlib	21
Track Bars	23
Color Filtering	24
Video Stream from Webcam and File	25
Video Stream from Computer Screen	26
Edge Derivation	27
Canny Edge Detection	29
Template Matching	30
Example: Template Matching Multiple Resistors	31
Corner Detection	34
Brute Force Feature Matching	35
Feature Matching with FLANN	36

What Is Computer Vision And OpenCV

Computer vision is a field of computer science which focuses on how a computer can use digital images to gain a high-level understanding of a scene. Computer vision is often used to automate tasks, for example: object recognition, facial recognition, and motion tracking, just to name a few. Computer vision is used in self-driving cars, camera technology, security cameras, and assembly line automation.

OpenCV is a programming library for computer vision. Intel began development of OpenCV in 1999, version 1.0 was released in 2006, and OpenCV2 was released in 2009. OpenCV is open source and free to use. OpenCV is available in Python, Java, MATLAB, and C++ languages and Windows, Linux, Android and Mac operating systems.

Additional links:

OpenCV home page - https://opencv.org/
OpenCV GitHub page - https://github.com/opencv/opencv
Python home page - https://www.python.org/

All images in this guide were either created by the author or where sourced from their specified sources.

Reasons For Creating This Guide

I created this guide to introduce the reader to computer vision and the functionality of the OpenCV Library. This guide is intended to be more of a 'how to' guide focused on applicability and usability, as opposed to a deep dive into the theory and mathematics taking place behind the scenes.

I created the guide in this way because as one is learning they don't necessarily need to concern themselves with what is happening in the background. For example, when someone is first learning to drive a car, they learn how to move the steering wheel, press the pedals, and put the gear lever into forward or reverse. They are then able drive the car from one place to another. The person learning to drive the car does not need to know how a 4-stroke engine, automatic transmission, and differential work. Only once they can drive competently they should then learn about the inner workings of a vehicle.

So too is my philosophy in learning the OpenCV library. You must learn to use the functions available before you need to learn how the matrices of data are mathematically computed. This guide will teach you how to drive the car and prepare you to look under the hood.

Getting Started

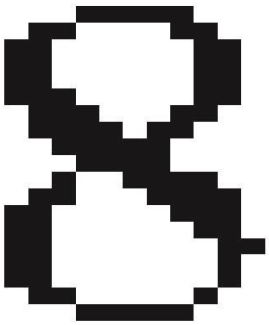
It is recommended to know how to program competently in any language before starting this guide. This guide is written in Python and in order to follow along one must have Python installed on their computer along with OpenCV, NumPy, Matplotlib, and Pillow.

Pixels

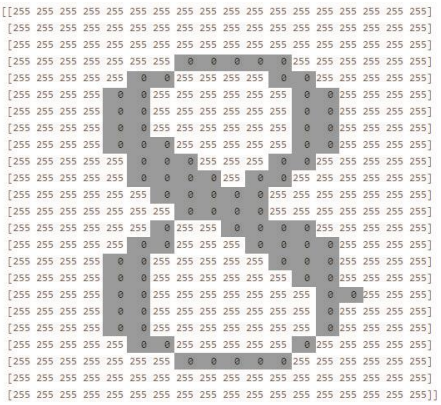
Pixels are the building blocks that together create an image. Pixels are often very small so that the human eye cant easily tell each pixel apart. Computers see images differently than humans. Computers see each pixel as a value or series of values.

Our example is an image of the number eight (a). It is 18 pixels across by 24 pixels long. At its native resolution individual pixels are difficult to see. However when we zoom in, the individual pixels appear (b). Image (c) is a visual representation that overlays how a human sees an image with how a computer sees an image. A computer sees an image by assigning a value to each pixel in a 2 dimensional array/matrix (d).

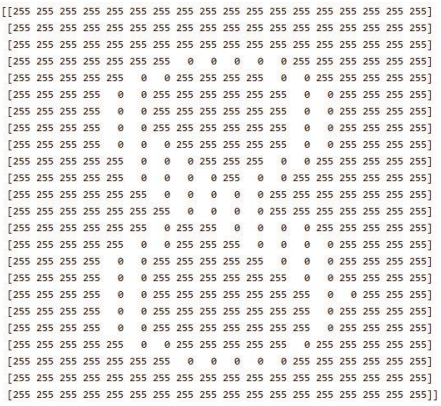
(a)



(b)



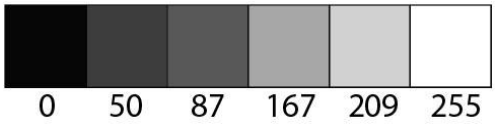
(c)



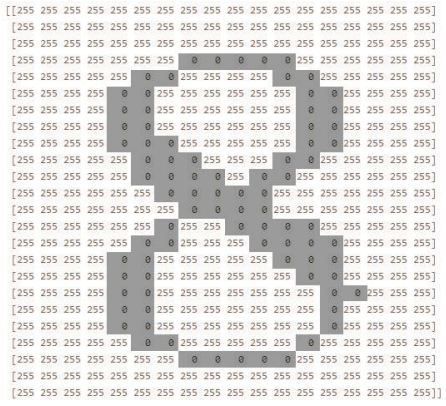
(d)

Grey Scale

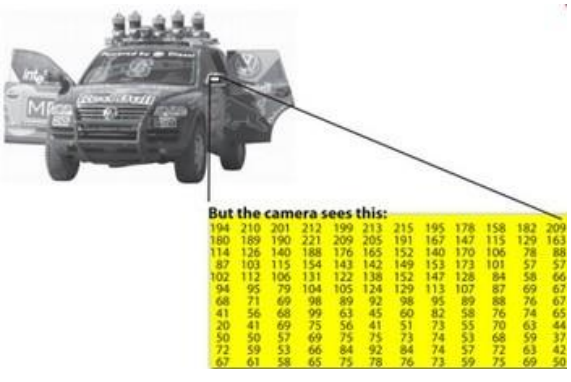
Grey scale is images that only contain a single value per pixel and that value can represent white, black or any shade of grey in between. OpenCV uses 8 bit greyscale as its default grey scale format. 8 bit means that there are 256 different shades and these shades range from 0 to 255. The value of a black pixel is 0 while the value of a white pixel is 255. Image (a) shows a representation of pixel value to the shade of the pixel. Recall from the previous page that the eight was only black and white and thus the computer represented it with the values of 0 and 255 (b). Image (c) is a grey scale example.



(a)



(b)



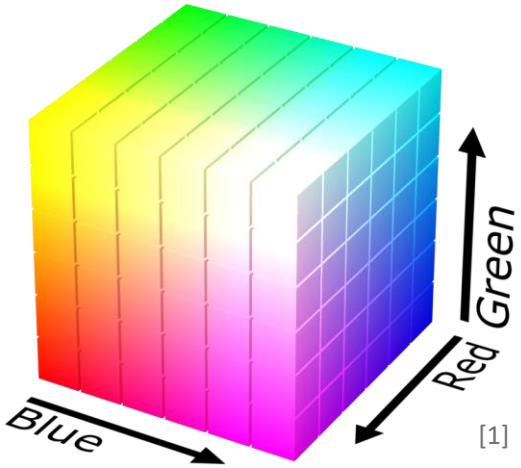
(c)

[1]

[1] https://docs.opencv.org/master/d6/d6d/tutorial_mat_the_basic_image_container.html

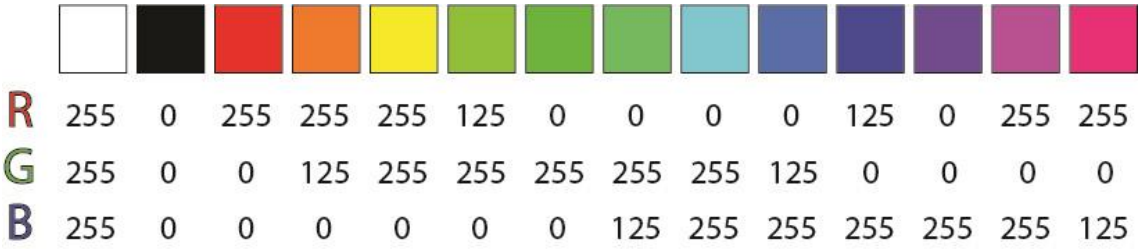
Color Images

In contrast to grey scale images, color images contain multiple values per pixel. OpenCV generally uses 3 channels of 8 bit each for blue, green, and red (a). Therefor 3 channels of 8 bit color is 24 bit. By default OpenCV use the BGR format. Each pixel is represented by three values, one value for blue, one for green, and one for red (b). With the various combinations of these three values many different colors can be made (c).



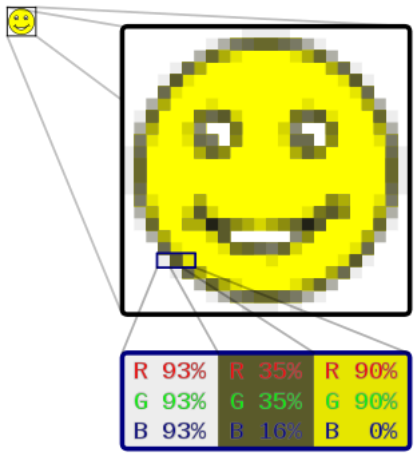
(a)

[1]



(b)

Note: the BGR format was once popular in the late 1990's and early 2000's but has since has falling out of mainstream use and has been over taken by the 4 channel, RGB format (with the fourth channels being alpha which represents opaqueness).



(c)

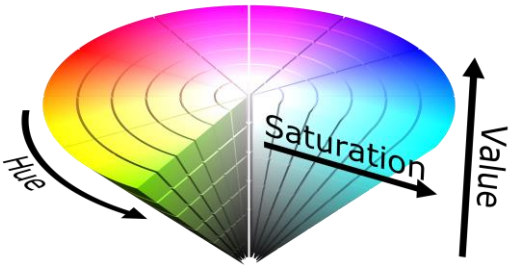
[2]

[1] https://en.wikipedia.org/wiki/RGB_color_model#/media/File:RGB_color_solid_cube.png
[2] https://en.wikipedia.org/wiki/Raster_graphics#/media/File:Rgb-raster-image.svg

Color Images

OpenCV is very flexible and easy to change the color format of the image. When possible it is preferred to use grey scale images as grey scale images contain one third of the information of a BGR. In other situation it is preferred to use other color formats. For example as we will see later the HSV (hue, saturation, value) format is usefully when trying to isolate particular colors (a). OpenCV can convert images to RGB, HVS, HLS, YCrRb, CIELAB, CIELuv, BAYER, and other formats.

Full list of color conversion codes: https://docs.opencv.org/3.4/d8/d01/group__imgproc__color__conversions.html#ga4e0972be5de079fed4e3a10e24ef5ef0



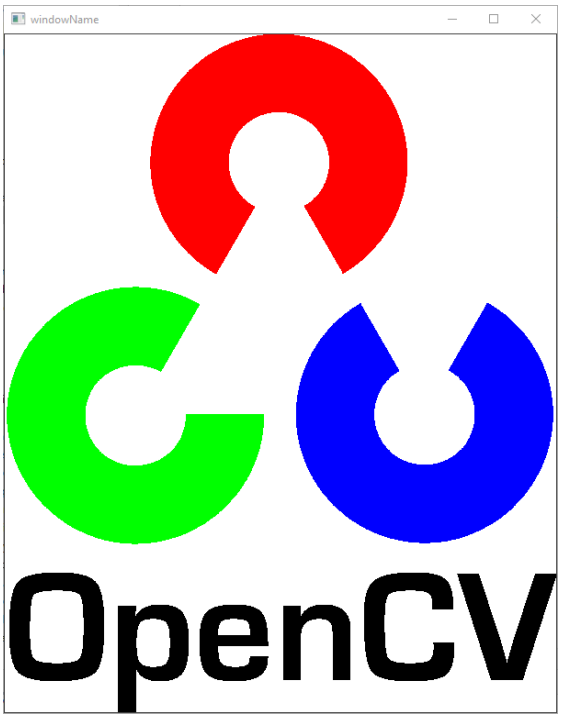
(a)

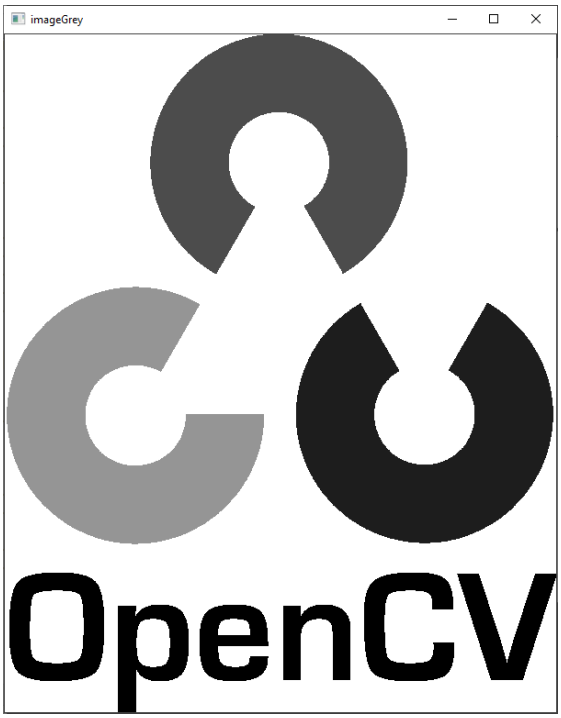
[1]

[1] https://commons.wikimedia.org/wiki/File:HSV_color_solid_cone.png

Basic Operations

```python
import cv2 as cv

#Load image with default colors
image = cv.imread('opencv-logo.png')
#Load image in greyscale
imageGrey= cv.imread(filename='opencv-logo.png', flags=0)

#save an image
cv.imwrite(filename="imageZeroSave.png", img=imageGrey)

#create a window
cv.namedWindow(winname='windowName',flags=cv.WINDOW_AUTOSIZE)
#display an image
cv.imshow(winname='imageGrey',mat=imageGrey)
cv.imshow('windowName',image)

# get pixel value at [y,x]
pixelValue = image[350, 150]
print(pixelValue)

#region of interest
smallImage = image[275:550, 0:300]
cv.imshow("smallImage", smallImage)

#wait till the zero key is pressed
#otherwise the program will end
cv.waitKey(0)
#closes all windows
cv.destroyAllWindows()
```





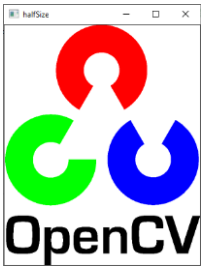
OpenCV supports Windows bitmaps, JPEG, JPEG2000, Portable Network Graphics, WebP, Portable Image Format, PFM, Sun rasters, TIFF, OpenEXR, Radiance HDR, GDAL

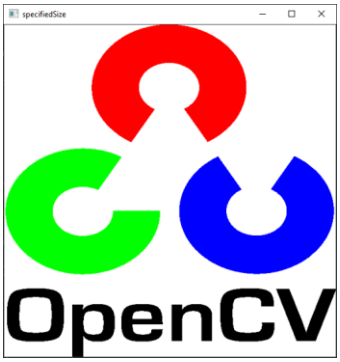

```
[  0 255   0]
```


Image source: https://github.com/opencv/opencv/tree/master/samples/data

Resizing an Image

```
import cv2 as cv

OriginalImage = cv.imread('opencv-logo.png')
cv.imshow('OriginalImage',OriginalImage)

#scale by percent, example 1
fiftyPercent = 0.5
width = int(OriginalImage.shape[1] * fiftyPercent)
height = int(OriginalImage.shape[0] * fiftyPercent)
dimension = (width, height)
#NOTE there are many different types of interpolation
halfSize = cv.resize(OriginalImage, dimension, interpolation= cv.INTER_AREA)
cv.imshow('halfSize', halfSize)

#scale by percent, example 2
twoHundredPercent = 2.0
width = int(OriginalImage.shape[1] * twoHundredPercent)
height = int(OriginalImage.shape[0])
dimension = (width, height)
twiceSize = cv.resize(OriginalImage, dimension, cv.INTER_AREA)
cv.imshow('twiceSize', twiceSize)

#scale by specified dimensions
width = 512
height = 512
dimension = (width, height)
specifiedSize = cv.resize(OriginalImage, dimension, cv.INTER_AREA)
cv.imshow('specifiedSize',specifiedSize)

cv.waitKey(0)
```





List of interpolation flags: https://docs.opencv.org/3.4/da/d54/group__imgproc__transform.html#ga5bb5a1fea74ea38e1a5445ca803ff121

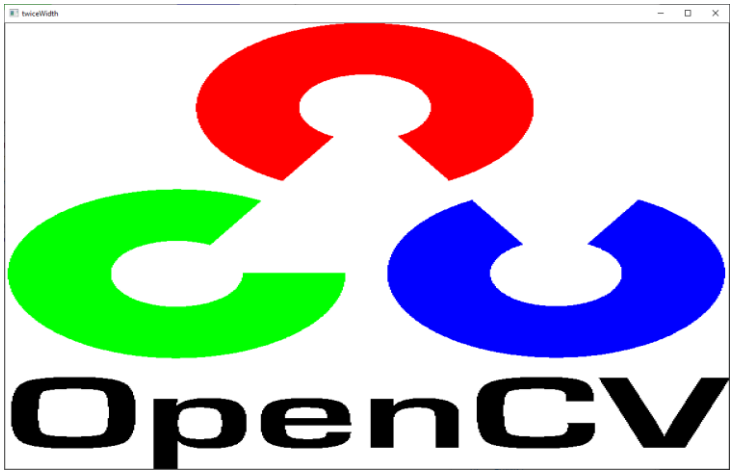


Image source: https://github.com/opencv/opencv/tree/master/samples/data

Concatenating Images

```
import cv2 as cv

image = cv.imread('opencv-logo.png')
imageHorzConcat = cv.hconcat([image,image])

cv.imshow('imageHorzConcat', imageHorzConcat)
cv.waitKey(0)
```

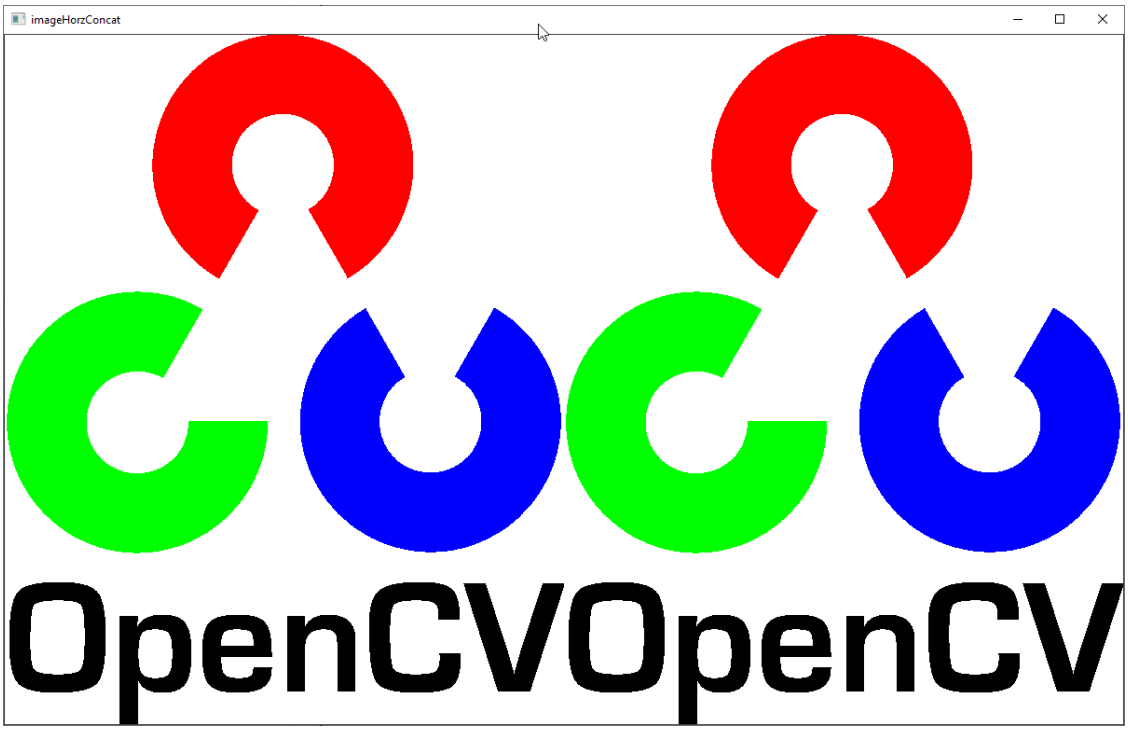
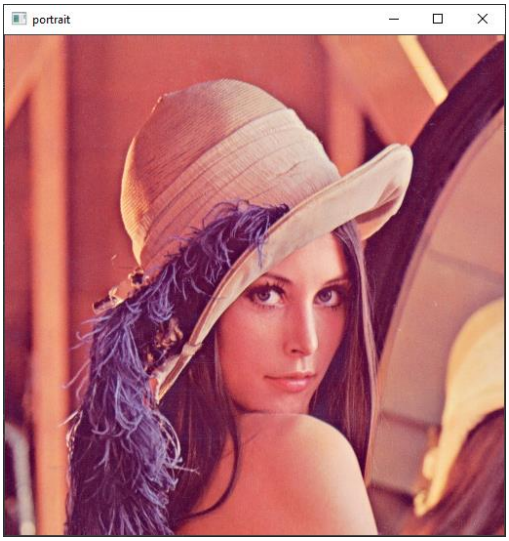


Image source: https://github.com/opencv/opencv/tree/master/samples/data

Blending Two Images

```
import cv2 as cv

#images must be same size
portrait = cv.imread('lena.jpg')
logo = cv.imread('opencv-logo-resize.png')

alpha = 0.15
beta = (1.0 - alpha)
resultImage = cv.addWeighted(src1=logo ,alpha=alpha ,src2=portrait ,beta=beta ,gamma=0.0)

cv.imshow('portrait', portrait)
cv.imshow('logo', logo)
cv.imshow('ResultImage', resultImage)
cv.waitKey(0)
```



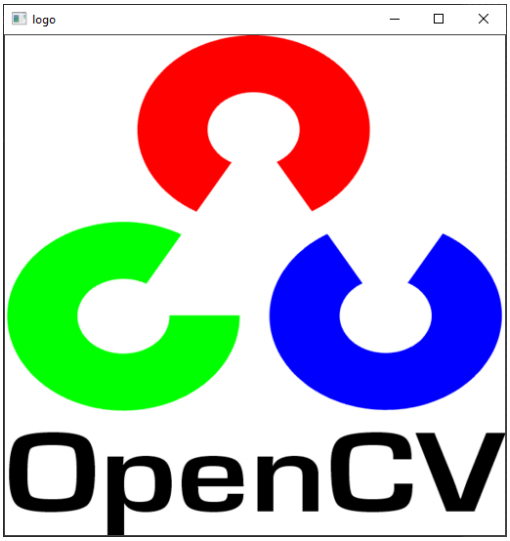
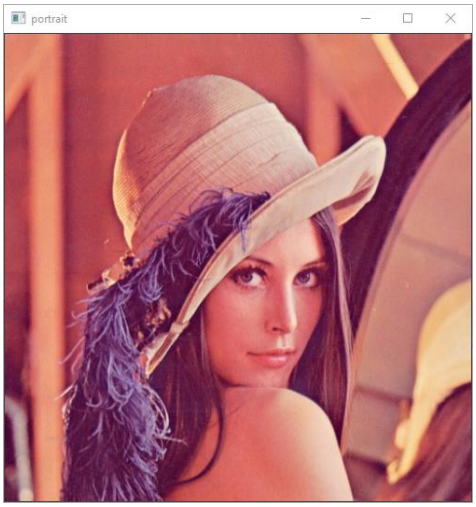
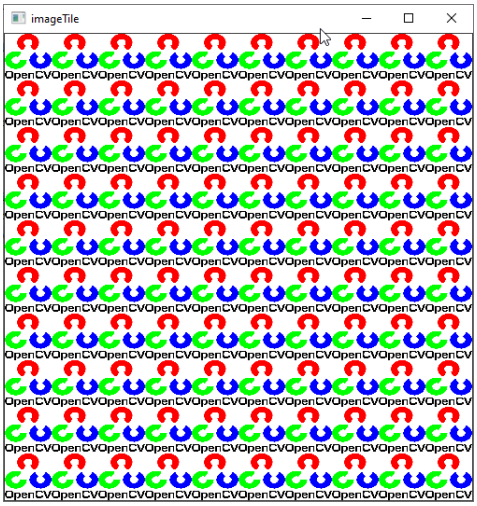


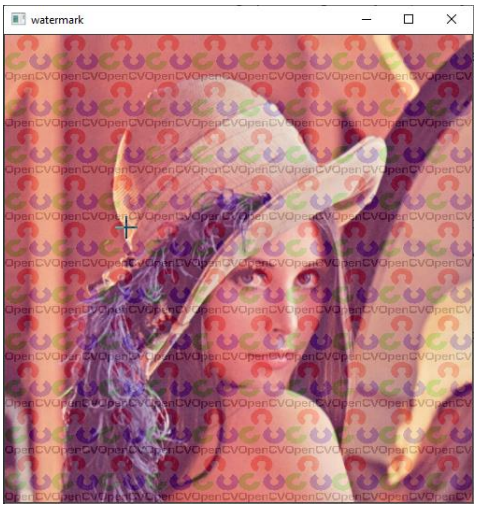
Image source: https://github.com/opencv/opencv/tree/master/samples/data

Example: Resize, Concatenate, and Blend

```
import cv2 as cv
import numpy as np

logo = cv.imread('opencv-logo.png')
smallImage = cv.resize(logo, (50,50), cv.INTER_AREA)

portrait = cv.imread('lena.jpg')
portrait = cv.resize(portrait, (500,500), cv.INTER_AREA)

imageTile = np.tile(smallImage, (10,10,1))

alpha = 0.15
beta = (1.0 - alpha)
watermark = cv.addWeighted(imageTile ,alpha ,portrait
,beta ,0.0)

cv.imshow('imageTile', imageTile)
cv.imshow('portrait', portrait)
cv.imshow('watermark', watermark)

cv.waitKey(0)
```





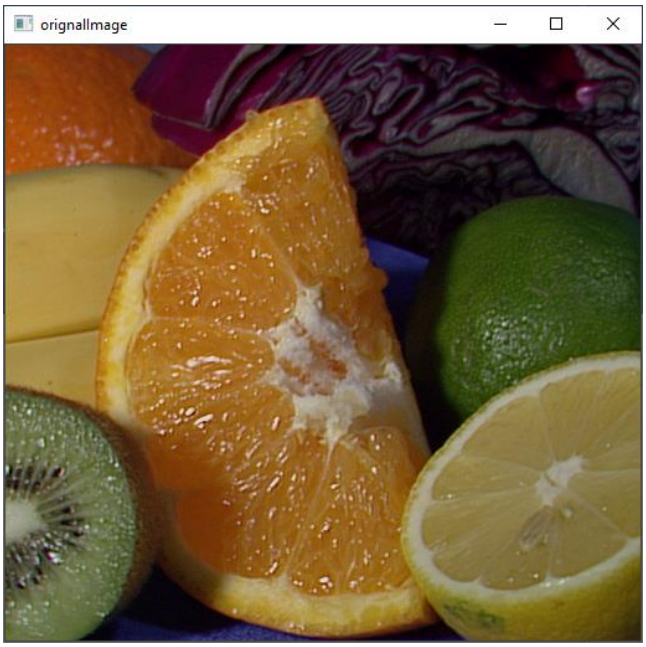


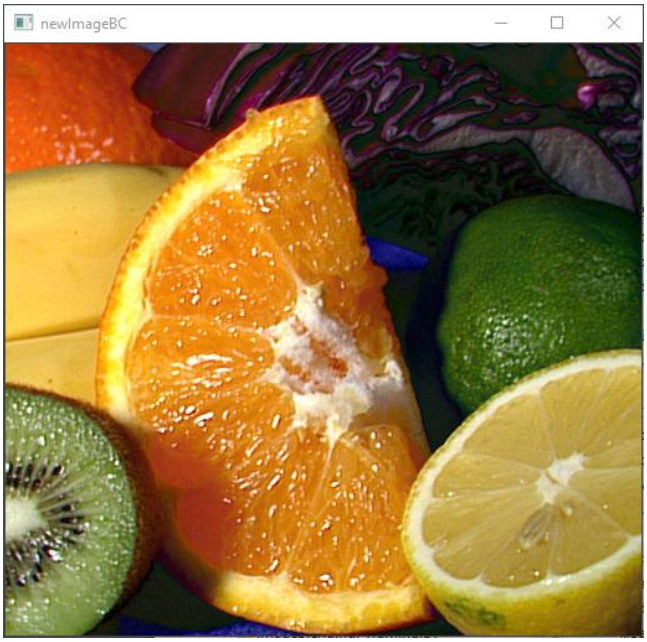
With a camera that names images in a sequential order and at the same resolution, one could write a program that loops through a folder of images and place a water mark on each one.

Image source: https://github.com/opencv/opencv/tree/master/samples/data

Brightness, Contrast, and Gamma

```
import cv2 as cv
import numpy as np
import math

originalImage = cv.imread('fruits.jpg')
cv.imshow('originalImage', originalImage)

#brightness and contrast example
alpha = 1.62
beta = -34
newImageBC = cv.convertScaleAbs(originalImage,
alpha=alpha, beta=beta)
cv.imshow('newImageBC', newImageBC)

#gamma example
#convert image to grey format
gray = cv.cvtColor(originalImage, cv.COLOR_BGR2GRAY)
mid = 0.5
mean = np.mean(gray)
gamma = math.log(mid*255)/math.log(mean)
newImageGamma = np.power(originalImage,
gamma).clip(0,255).astype(np.uint8)
cv.imshow('newImageGamma', newImageGamma)

cv.waitKey(0)
```





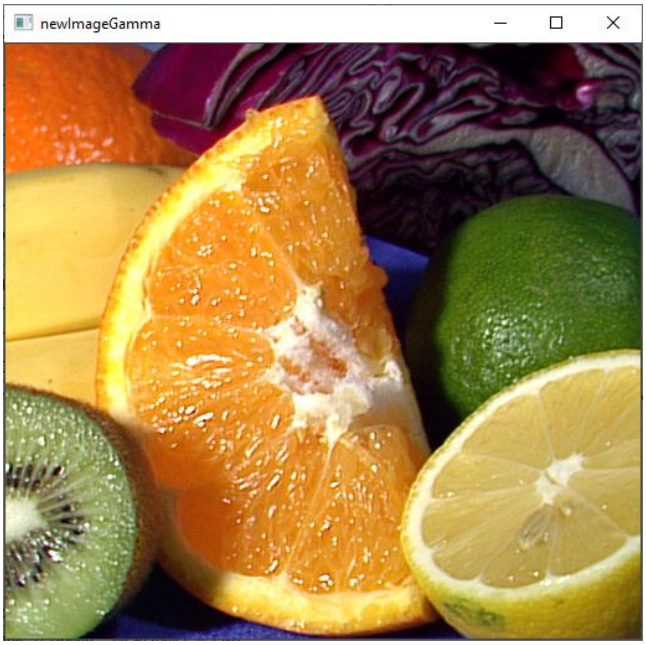
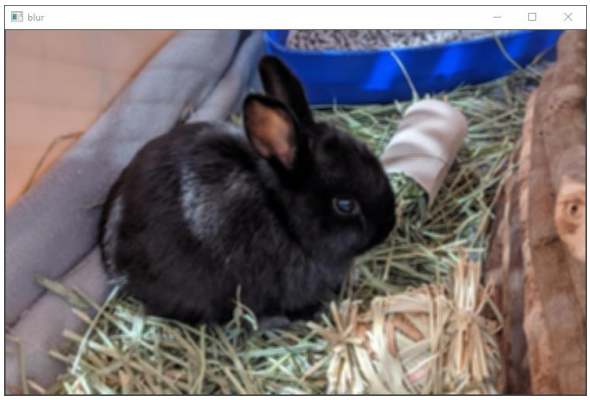


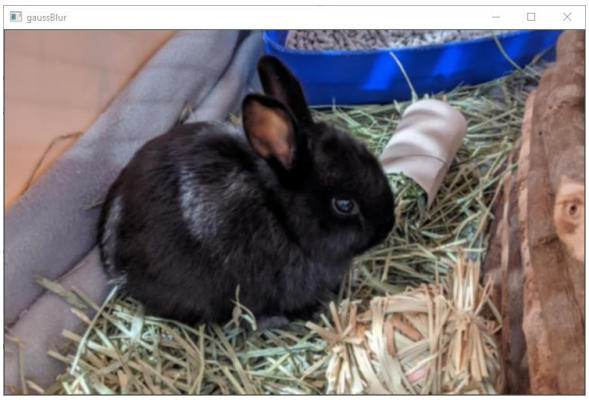
Image source: https://github.com/opencv/opencv/tree/master/samples/data

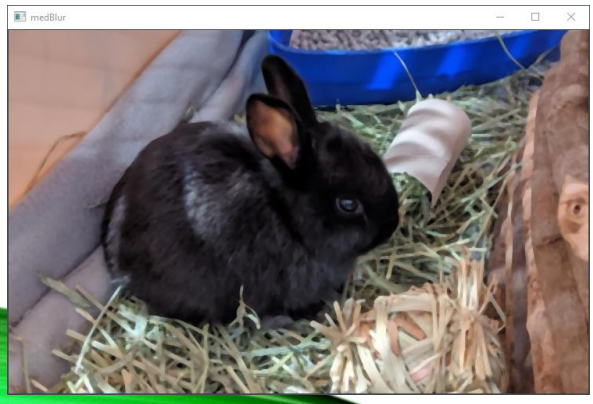
Smoothing and Blurring

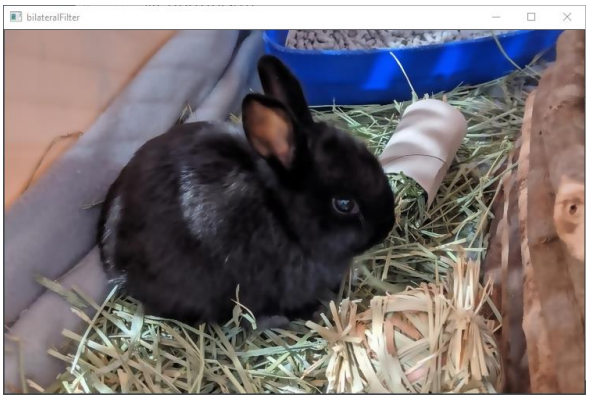
```
import cv2 as cv

image = cv.imread('bunny-resized.jpg')
blur = cv.blur(src=image, ksize=(5,5))
#GaussianBlur for removing high frequency "static" from image
gaussBlur = cv.GaussianBlur(src=image, ksize=(5,5), sigmaX=0)
#medianBlur for dealing with "salt and pepper" noise
medBlur = cv.medianBlur(src=image, ksize=5)
#bilateral filter preserves edges while blurring noise
bilateralFilter =cv.bilateralFilter(src=image, d=9,
sigmaColor=75, sigmaSpace=75)

cv.imshow('image',image)
cv.imshow('blur',blur)
cv.imshow('gaussBlur',gaussBlur)
cv.imshow('medBlur',medBlur)
cv.imshow('bilateralFilter',bilateralFilter)
cv.waitKey(0)
```

Image blurring is useful for reducing image noise, as we will see later this is useful for edge detection.



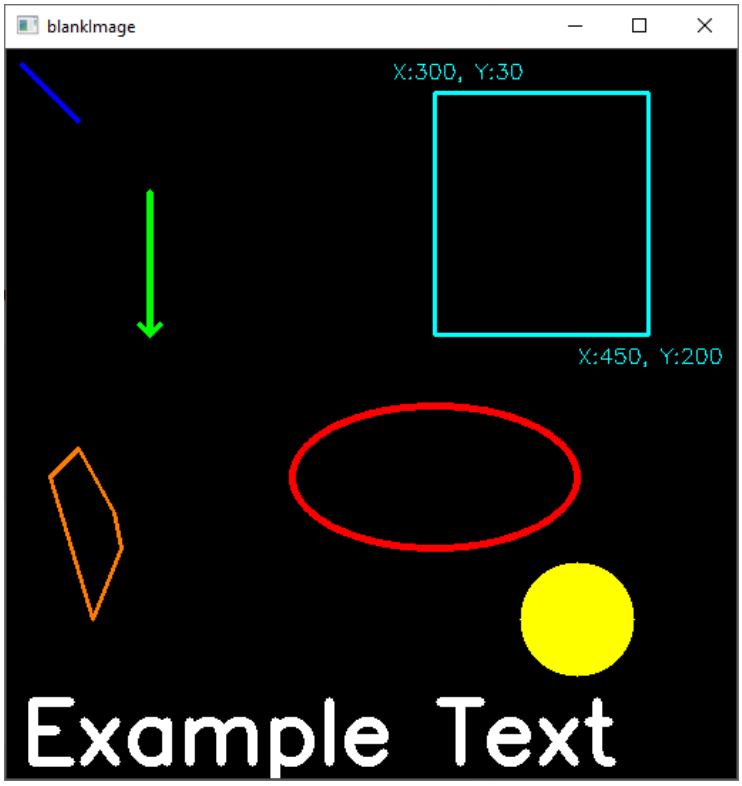






Adding Text and Shapes

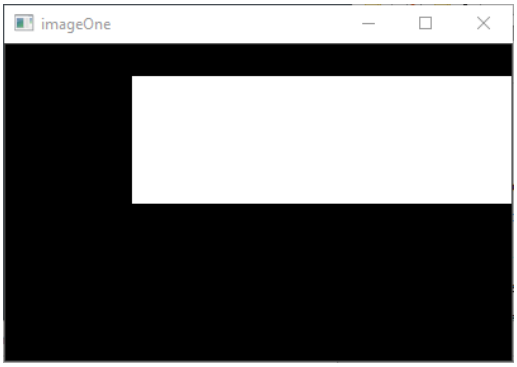
```
import numpy as np
import cv2 as cv

# creates an image filled with zeros (recall zero is black in grayscale)
blankImage = np.zeros([512,512, 3], np.uint8)
#thickness refers to line thickness, with -1 meaning fill
blankImage = cv.line(img=blankImage, pt1=(10,10), pt2=(50,50), color=(255,0,0), thickness=2)
blankImage = cv.arrowedLine(blankImage, (100,100),  (100,200), (0,255,0), 3)
blankImage = cv.ellipse(img=blankImage, center=(300,300), axes=(100,50), angle=0, startAngle=0, endAngle=360,
 color=(0,0,255), thickness=3)
blankImage = cv.circle(img=blankImage, center= (400,400) ,radius=40, color=(0,255,255), thickness=-1)
blankImage = cv.putText(img=blankImage, text='Example Text', org=(10,500), fontFace=cv.FONT_HERSHEY_SIMPLEX,
fontScale=2, color=(255,255,255), thickness=5)

#draw polygon
points = np.array([[30,300],[50, 280],[75,325],[80,350],[60,400]])
points = points.reshape((-1,1,2)) #reorder array for opencv
cv.polylines(img=blankImage, pts=[points], isClosed=True, color=(0,125,255), thickness=2)

#rectangle shape and text example
point1 = (300,30)
point2 = (450,200)
blankImage = cv.rectangle(img=blankImage, pt1=point1 , pt2=point2 , color=(255,255,0), thickness=2)
blankImage = cv.putText(blankImage, 'X:'+str(point1[0])+', Y:'+str(point1[1]), (point1[0]-30,point1[1]-10),
 cv.FONT_HERSHEY_SIMPLEX, 0.5, (255,255,0), 1)
blankImage = cv.putText(blankImage, 'X:'+str(point2[0])+', Y:'+str(point2[1]), (point2[0]-50,point2[1]+20),
 cv.FONT_HERSHEY_SIMPLEX, 0.5, (255,255,0), 1)

cv.imshow('blankImage', blankImage)
cv.waitKey(0)
```

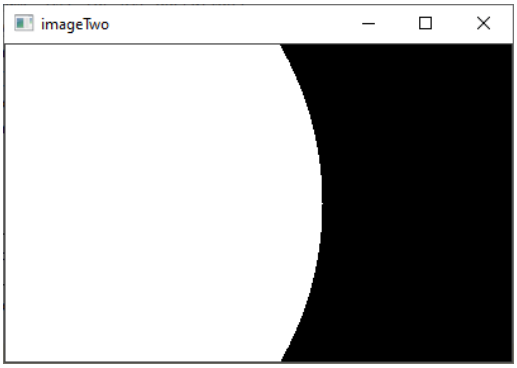


Drawing shapes on an image is useful to highlight and display information about the image.

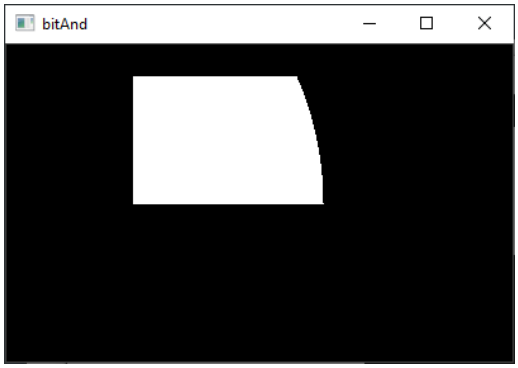
Bitwise Operations

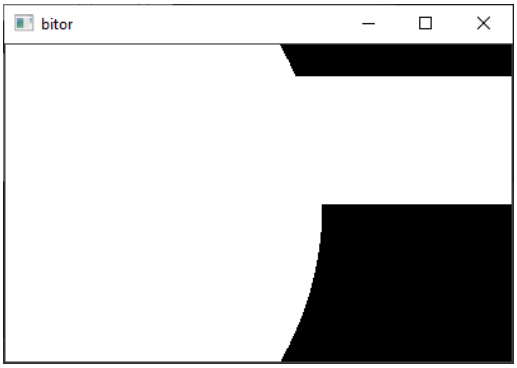
```
import cv2 as cv
import numpy as np

#making imageOne
imageOne = np.zeros((250,400), np.uint8)
imageOne = cv.rectangle(imageOne, (100,25), (400,125),
(255,255,255), -1)

#making imageTwo
imageTwo = np.zeros((250,400), np.uint8)
imageTwo = cv.circle(imageTwo, (0,125), 250,
(255,255,255), -1)

#note images must be same size for bit operations
#images should be in binary colors, white==True,
#black==False
bitAnd = cv.bitwise_and(imageOne, imageTwo)
bitOr = cv.bitwise_or(imageOne, imageTwo)
bitXor = cv.bitwise_xor(imageOne, imageTwo)
bitNot1 = cv.bitwise_not( imageOne)

cv.imshow("imageOne", imageOne)
cv.imshow("imageTwo", imageTwo)
cv.imshow('bitAnd', bitAnd)
cv.imshow("bitor", bitOr)
cv.imshow("bitxor", bitXor)
cv.imshow("bitnot1", bitNot1)

cv.waitKey()
```

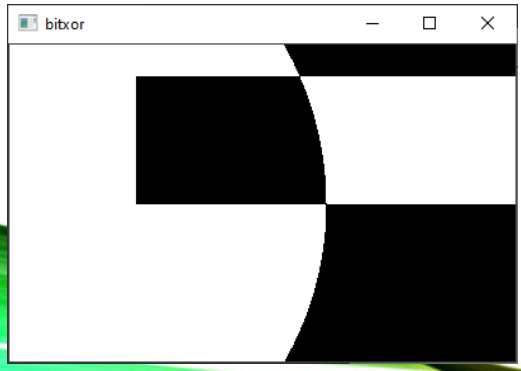


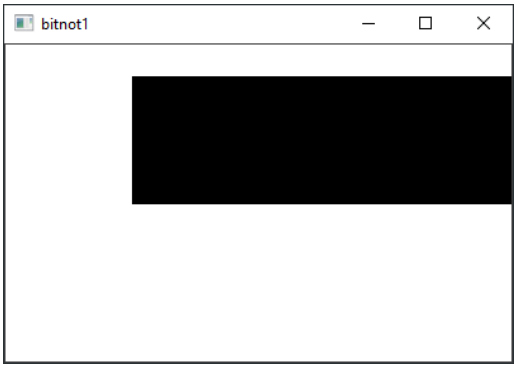


Bitwise operations are logical operations on binary images and are useful when using masks and thresholds.





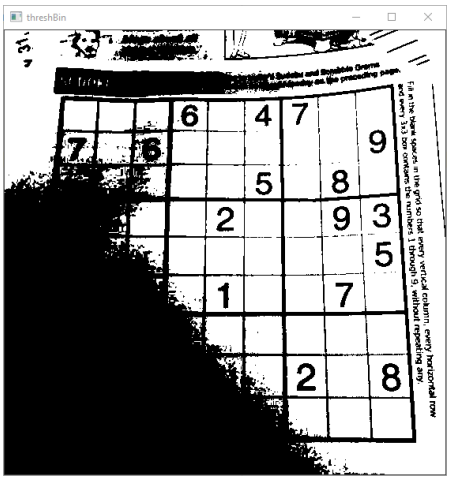


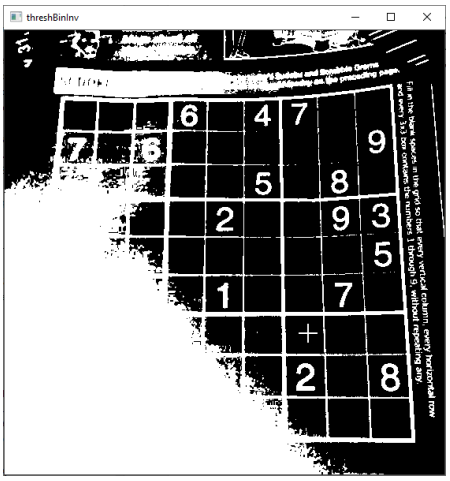


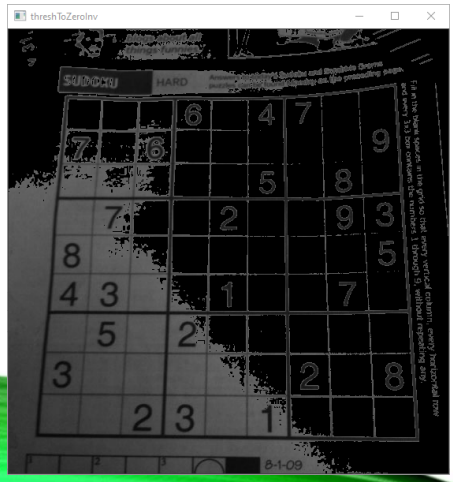
Basic Threshold

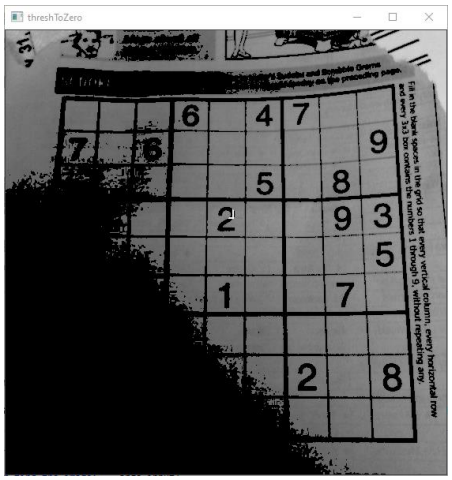
```
import cv2 as cv

image = cv.imread('sudoku.png', 0)

_, threshBin = cv.threshold(src=image, thresh=100, maxval=255, type=cv.THRESH_BINARY)
_, threshBinInv = cv.threshold(image, 100, 255, cv.THRESH_BINARY_INV)
_, threshTrunk = cv.threshold(image, 100, 255, cv.THRESH_TRUNC)
_, threshToZero = cv.threshold(image, 100, 255, cv.THRESH_TOZERO)
_, threshToZeroInv = cv.threshold(image, 100, 255, cv.THRESH_TOZERO_INV)

cv.imshow("threshBin", threshBin)
cv.imshow("threshBinInv", threshBinInv)
cv.imshow("threshTrunk", threshTrunk)
cv.imshow("threshToZero", threshToZero)
cv.imshow("threshToZeroInv", threshToZeroInv)
cv.imshow("image", image)

cv.waitKey()
```

List of basic threshold types:
https://docs.opencv.org/master/d7/d1b/group__imgproc__misc.html#gaa9e58d2860d4afa658ef70a9b1115576









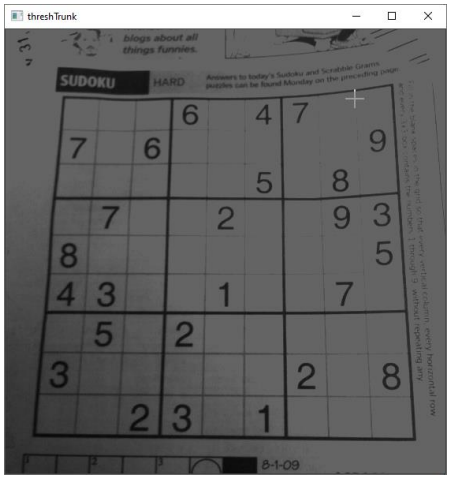
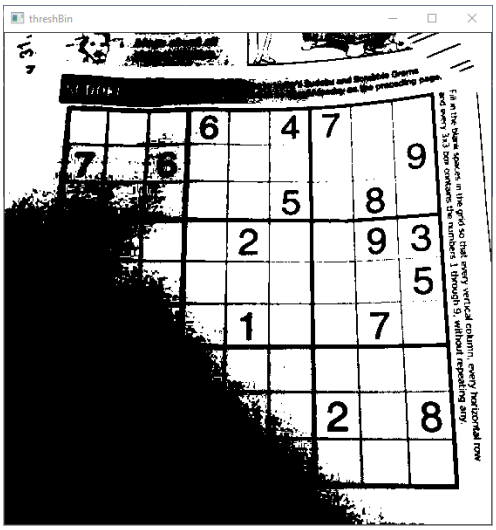


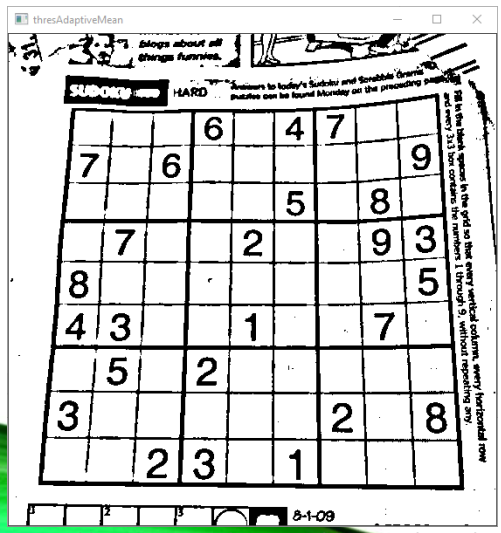
Image source: https://github.com/opencv/opencv/tree/master/samples/data

Adaptive Threshold

```
import cv2 as cv

originalImage = cv.imread('sudoku.png', 0)

_, threshBin = cv.threshold(src=originalImage, thresh=100 ,maxval=255, type=cv.THRESH_BINARY)
thresAdaptiveMean = cv.adaptiveThreshold(src=originalImage, maxValue=255, adaptiveMethod=cv.ADAPTIVE_THRESH_MEAN_C,
thresholdType=cv.THRESH_BINARY, blockSize=25, C=8)
thresAdaptiveGaussian = cv.adaptiveThreshold(src=originalImage, maxValue=255,
adaptiveMethod=cv.ADAPTIVE_THRESH_GAUSSIAN_C, thresholdType=cv.THRESH_BINARY, blockSize=25, C=8)

cv.imshow("originalImage", originalImage)
cv.imshow("threshBin", threshBin)
cv.imshow("thresAdaptiveMean", thresAdaptiveMean)
cv.imshow("thresAdaptiveGaussian", thresAdaptiveGaussian)
cv.waitKey()
```

Adaptive thresholds are preferred when images are not evenly lit.





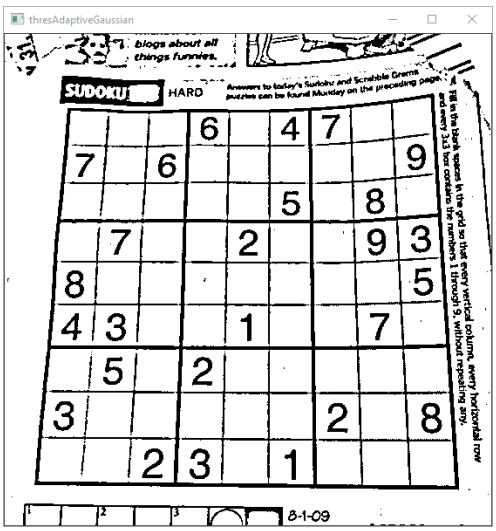


Image source: https://github.com/opencv/opencv/tree/master/samples/data

Erosion and Dilation

```
import cv2 as cv
import numpy as np

image = cv.imread('helloWorld.png', cv.IMREAD_GRAYSCALE)
_, threshBinInv = cv.threshold(image, 100, 255, cv.THRESH_BINARY_INV)

#a kernel is a smaller matrix involved in the computation of effects in opencv
kernel = np.ones((2,2), np.uint8)
dilation = cv.dilate(src=threshBinInv, kernel=kernel, iterations=4)
erosion = cv.erode(src=threshBinInv, kernel=kernel, iterations=4)
#note: dilation and erosion are termed relative to the bright areas of the image


cv.imshow('image', image)
cv.imshow('threshBinInv', threshBinInv)
cv.imshow('dilation', dilation)
cv.imshow('erosion', erosion)

cv.waitKey(0)
```

Matplotlib

```
import cv2
from matplotlib import pyplot as plt

image = cv2.imread('lena.jpg')
# convert color format to RGB for mathplotlib
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.imshow(image)
plt.show()
cv2.waitKey(0)
```

Matplotlib provides additional functionality when displaying images. You can use your mouse to hover over the image and it will display the pixel location and values. This is of particular value when trying to isolate a portion of the image or a particular color value. Matplotlib also allows you to zoom, resize, reposition, and title one or more images.

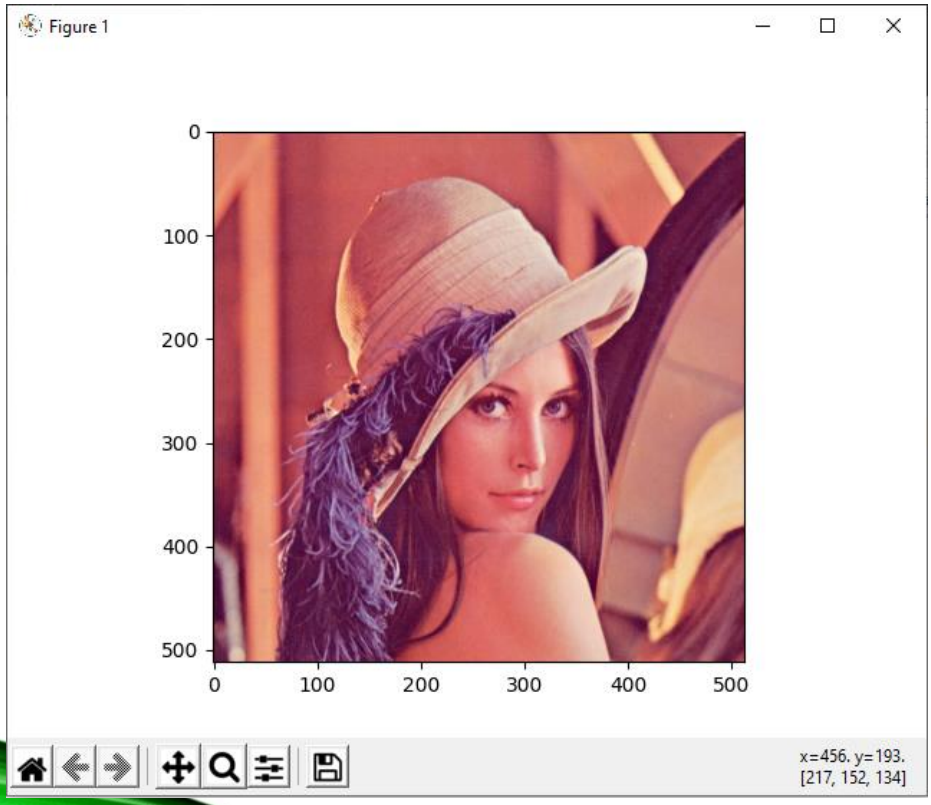


Image source: https://github.com/opencv/opencv/tree/master/samples/data

Matplotlib

```
import cv2 as cv
from matplotlib import pyplot as plt

originalImage = cv.imread('sudoku.png', 0)

_, threshBin = cv.threshold(src=originalImage, thresh=100 ,maxval=255, type=cv.THRESH_BINARY)
thresAdaptiveMean = cv.adaptiveThreshold(src=originalImage, maxValue=255, adaptiveMethod=cv.ADAPTIVE_THRESH_MEAN_C,
thresholdType=cv.THRESH_BINARY, blockSize=25, C=8)
thresAdaptiveGaussian = cv.adaptiveThreshold(src=originalImage, maxValue=255,
adaptiveMethod=cv.ADAPTIVE_THRESH_GAUSSIAN_C, thresholdType=cv.THRESH_BINARY, blockSize=25, C=8)

imageTitles = ['originalImage', 'threshBin', 'thresAdaptiveMean', 'thresAdaptiveGaussian']
imageList = [originalImage, threshBin, thresAdaptiveMean, thresAdaptiveGaussian]

for i in range(4):
    #subplot(nrows= , ncols= , index= )
    plt.subplot(2, 2, i+1), plt.imshow(imageList[i], 'gray')
    plt.title(imageTitles[i])

fig = plt.gcf()
fig.canvas.set_window_title('Threshold Type Example')
plt.show()
```

Here is our adaptive threshold example displayed in Matplotlib.

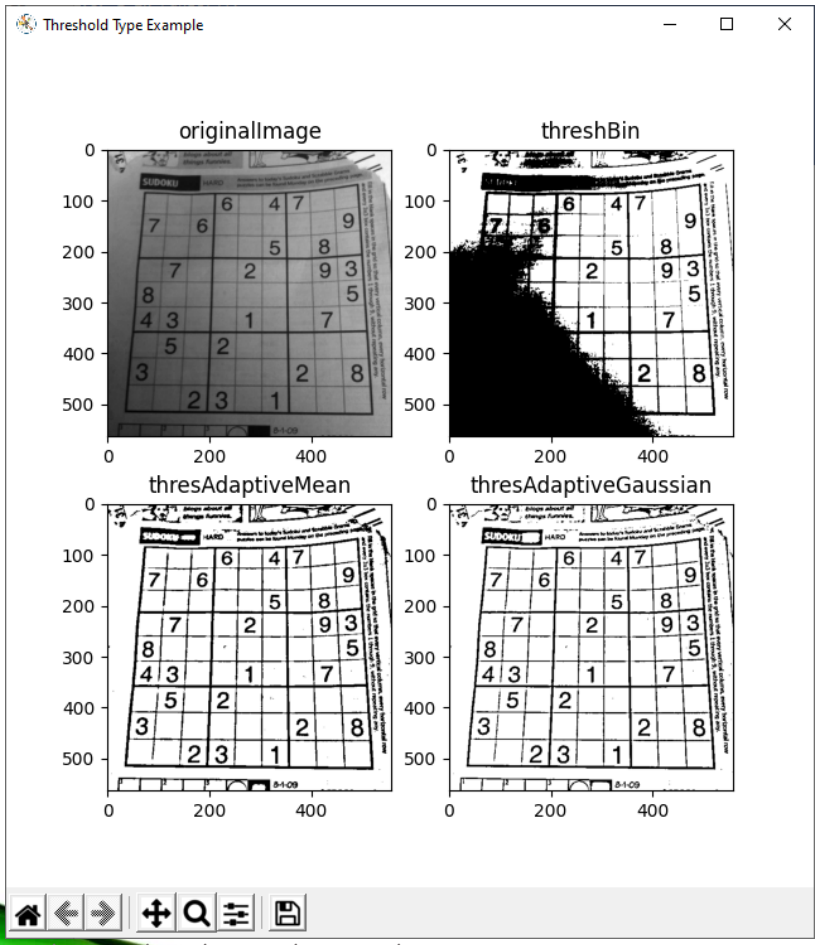
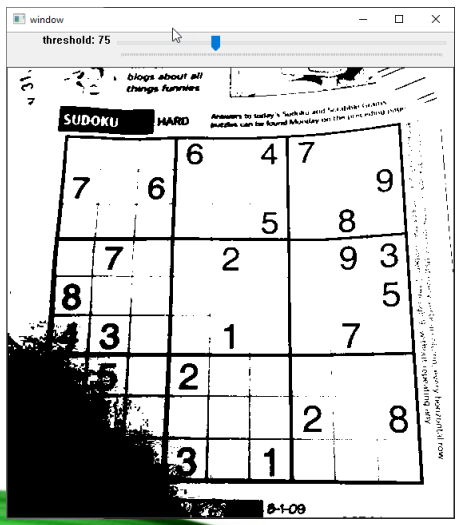


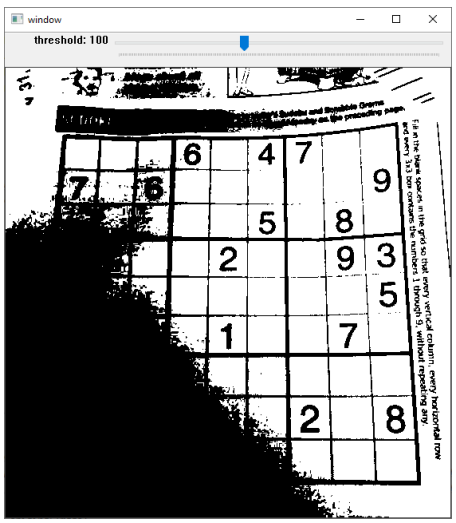
Image source: https://github.com/opencv/opencv/tree/master/samples/data

Track Bars

```
import cv2 as cv

def changeThresh(val):
    global newThresh
    newThresh = val
    callBack()

def callBack():
    newThresh = cv.getTrackbarPos('threshold', window)
    originalImage = cv.imread('sudoku.png', 0)
    _, threshBin = cv.threshold(src=originalImage, thresh=newThresh, maxval=255, type=cv.THRESH_BINARY)
    cv.imshow(window,threshBin)

window = 'window'
cv.namedWindow('window')
threshMax = 255
thresh = 0
#cv.createTrackbar(trackbarName=, windowName=, value=, count=, onChange=)
cv.createTrackbar('threshold', window, thresh, threshMax, changeThresh)
callBack()

cv.waitKey()
```

Track bars are a very useful way of quickly seeing how changes in the inputs affect the resulting image. Instead of trial and error, track bars can be used to quickly determine the optimal input values.





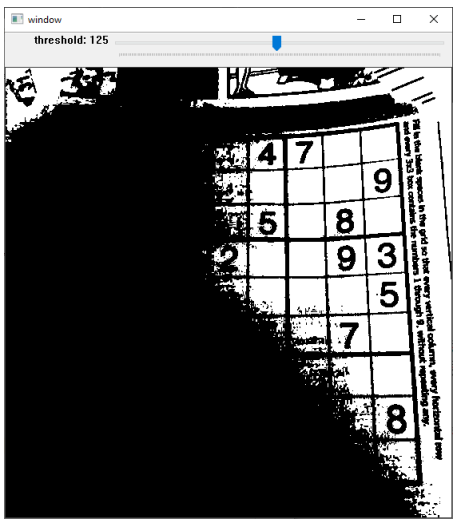
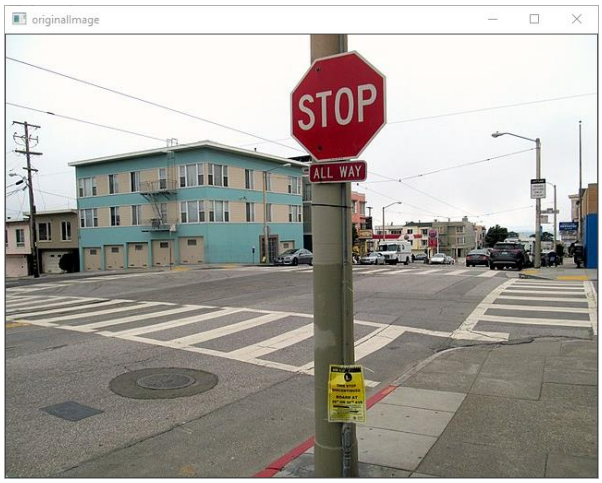


Image source: https://github.com/opencv/opencv/tree/master/samples/data

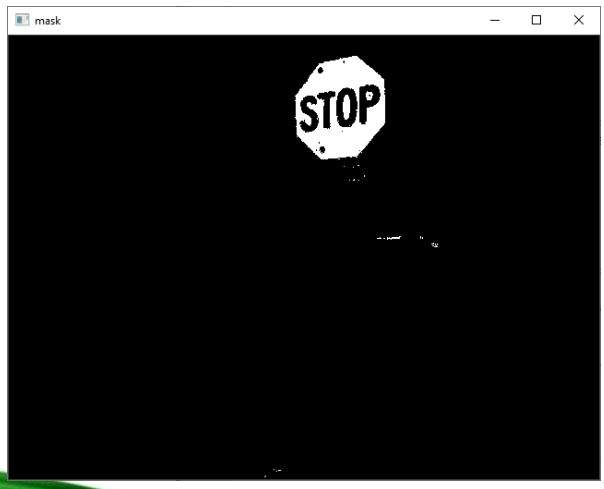
Color Filtering

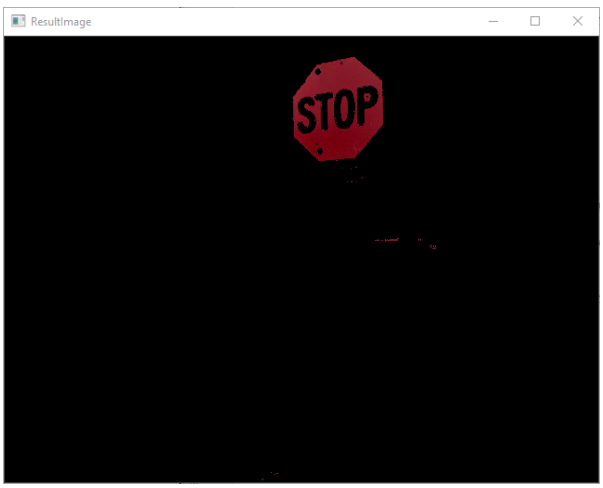


[1]

```
import cv2 as cv
import numpy as np

originalImage = cv.imread('stopSign.jpg')
#convert to HVS format
HVSImage = cv.cvtColor(originalImage, cv.COLOR_BGR2HSV)

lowerHue = 172
upperHue = 181
lowerSat = 162
upperSat = 255
lowerVal = 109
upperVal = 255

lowerBound = np.array([lowerHue,lowerSat,lowerVal])
upperBound = np.array([upperHue,upperSat, upperVal])
mask = cv.inRange(src=HVSImage, lowerb=lowerBound, upperb=upperBound)
ResultImage = cv.bitwise_and(src1=originalImage,src2=originalImage, mask=mask)

cv.imshow('originalImage', originalImage)
cv.imshow('mask', mask)
cv.imshow('ResultImage',ResultImage)
cv.waitKey(0)
```

Color filtering can be a very useful tool to isolate a portion of a image with a particular color range. Note: color filtering works better on image in the HVS format as there is less overlap in the color space and thus are easier to separate.





[1] https://commons.wikimedia.org/wiki/File:Taraval_and_28th_Avenue_with_stop_closure_notice,_June_2017.JPG

Video Stream from Webcam

```
import cv2 as cv

videoCapture = cv.VideoCapture(0)

while (videoCapture.isOpened()):
    _, frame = videoCapture.read()
    cv.imshow("frame", frame)

    # this is the recommended way to exit a video stream in opencv
    # if the q key is pressed, exit loop and close windows
    if cv.waitKey(1) & 0xFF == ord('q'):
        break

videoCapture.release()
cv.destroyAllWindows()
```

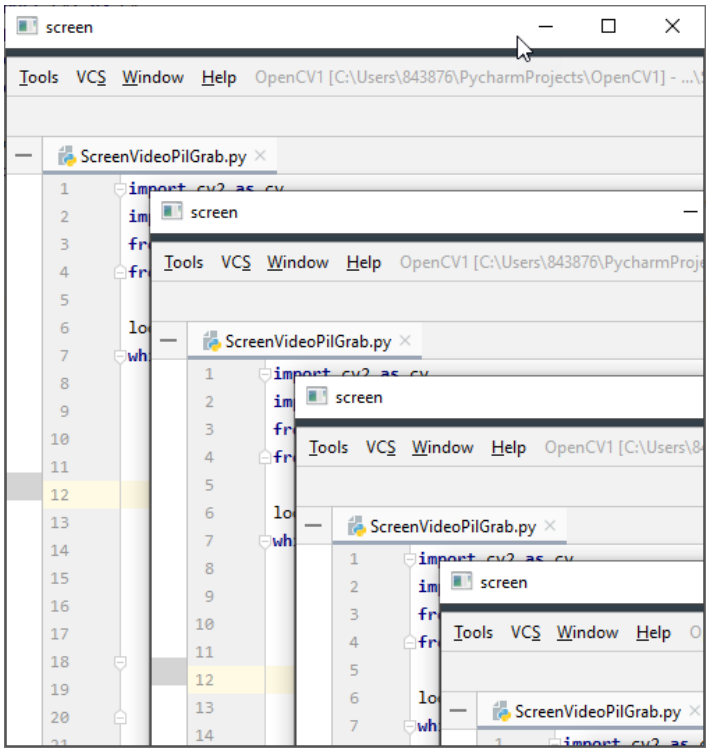
Video Stream from File

```
import cv2 as cv
import time

videoCapture = cv.VideoCapture('vtest.avi')
fps = int(videoCapture.get(cv.CAP_PROP_FPS))

while (videoCapture.isOpened()):
    _, frame = videoCapture.read()
    time.sleep(1/fps)
    cv.imshow("frame", frame)

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

videoCapture.release()
cv.destroyAllWindows()
```

Note: when playing video files, you may need to determine the fps and manually set the frame delay.

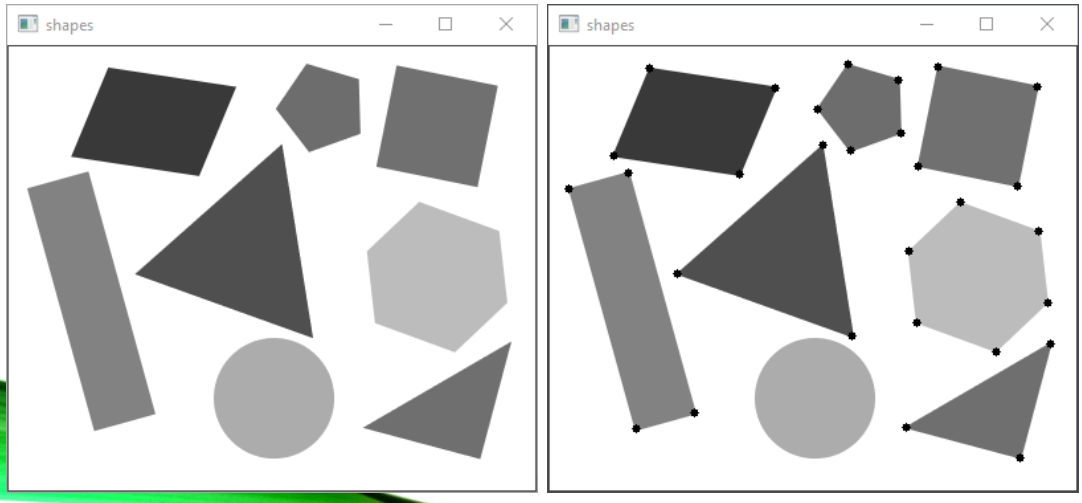
Video Stream from Computer Screen

```
import cv2 as cv
import numpy as np
from time import time
from PIL import ImageGrab

loopTime = time()
while (True):
    screenshot = ImageGrab.grab(bbox=(0,0,500,500)) #bbox=(left, top, width, height)
    #ImageGrab.grab() will grab entire screen
    screenshot = np.array(screenshot)
    screenshot = cv.cvtColor(screenshot, cv.COLOR_RGB2BGR)

    cv.imshow('screen', screenshot)

    print('FPS {}'.format(1/ (time() - loopTime))) #print FPS to console
    loopTime = time()

    if cv.waitKey(1) ==ord('q'):
        break

cv.destroyAllWindows()
```



Corner Detection

```
import cv2 as cv
import numpy as np

#exmaple one
helloWorld = cv.imread('helloWorld.png', 0
hwCorners = cv.goodFeaturesToTrack(image=helloWorld, maxCorners=100, qualityLevel=0.1, minDistance=5)
hwCorners = np.int0(hwCorners)

for hwCorner in hwCorners:
    x,y = hwCorner.ravel()
    cv.circle(helloWorld, (x,y), 3,0,-1)

cv.imshow('helloWorld', helloWorld)

#example two
shapes = cv.imread('shapes.png', 0)
shapesCorners = cv.goodFeaturesToTrack(image=shapes, maxCorners=100, qualityLevel=0.01, minDistance=5)
shapesCorners = np.int0(shapesCorners)

for sCorner in shapesCorners:
    x,y = sCorner.ravel()
    cv.circle(shapes, (x,y), 3,0,-1)

cv.imshow('shapes', shapes)
cv.waitKey(0)
```

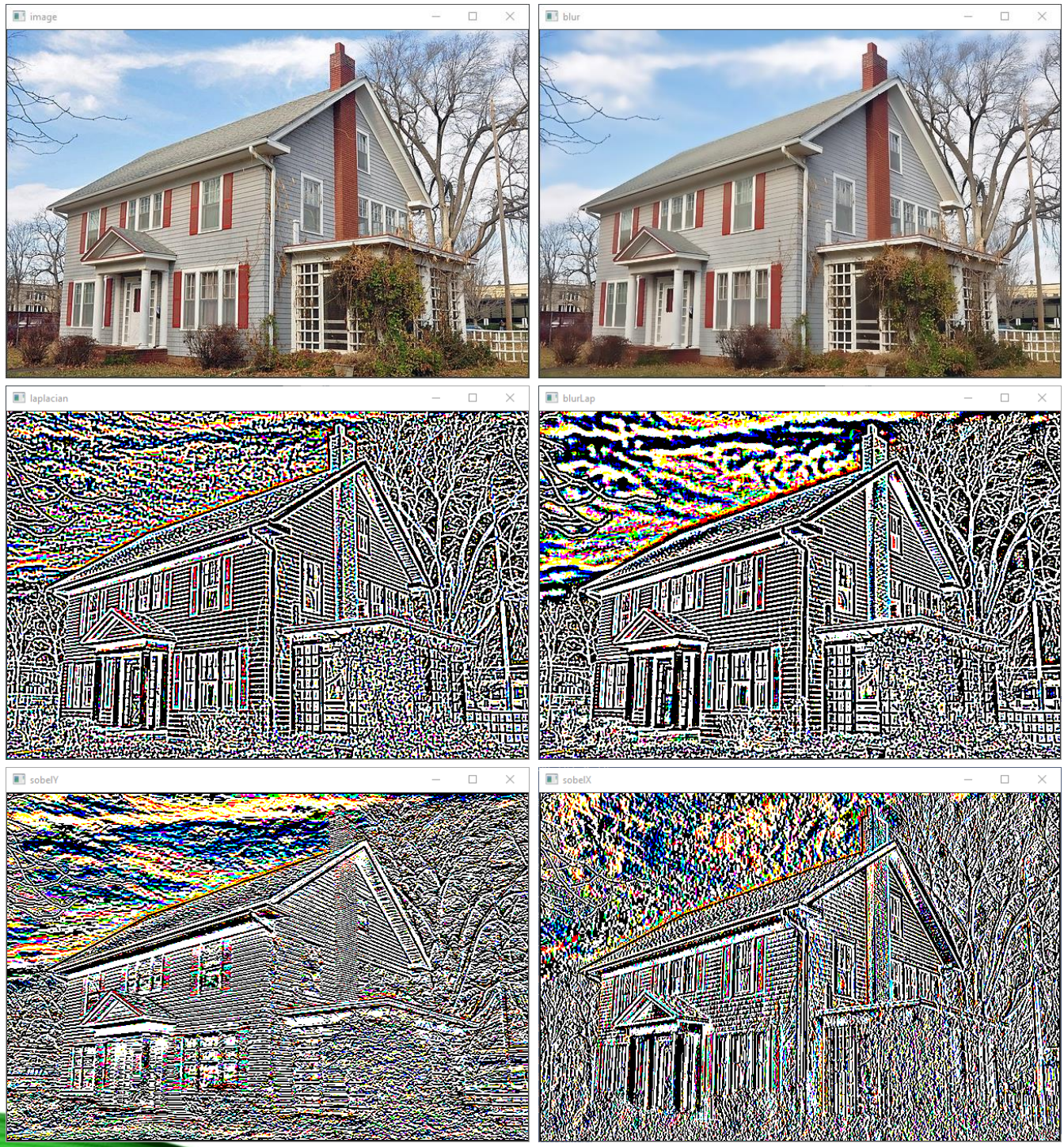


Edge Derivation

```
import cv2 as cv

image = cv.imread('house.jpg', cv.CV_64F)
blurImage = cv.bilateralFilter(src=image, d=10, sigmaColor=75, sigmaSpace=75)

#derive vertical lines
sobelX = cv.Sobel(src=image ,ddepth=cv.CV_64F,dx=1,dy=0, ksize=3)

#derive horizontal lines
sobelY = cv.Sobel(src=image ,ddepth=cv.CV_64F,dx=0,dy=1, ksize=3)
#if kernel size is 3 then use Scharr() as its more accurate for that particular kernel size
#ScharrY = cv.Scharr(src=image, ddepth=cv.CV_64F, dx=0, dy=1)

laplacian = cv.Laplacian(src=image, ddepth=cv.CV_64F, ksize=7)
blurLap = cv.Laplacian(src=blurImage, ddepth=cv.CV_64F, ksize=7)

cv.imshow('image', image)
cv.imshow('blurImage', blurImage)
cv.imshow('sobelX',sobelX)
cv.imshow('sobelY',sobelY)
cv.imshow('laplacian',laplacian)
cv.imshow('blurLap',blurLap)

cv.waitKey(0)
```

Edge derivation is used to highlight edges of an image. You can even specify weather you are interesting in horizontal or vertical edges. Bilateral blurring is recommended when looking for edges as it reduces noise while keeping the edges fairly sharp. See next page for output images.

Edge Derivation



[1]

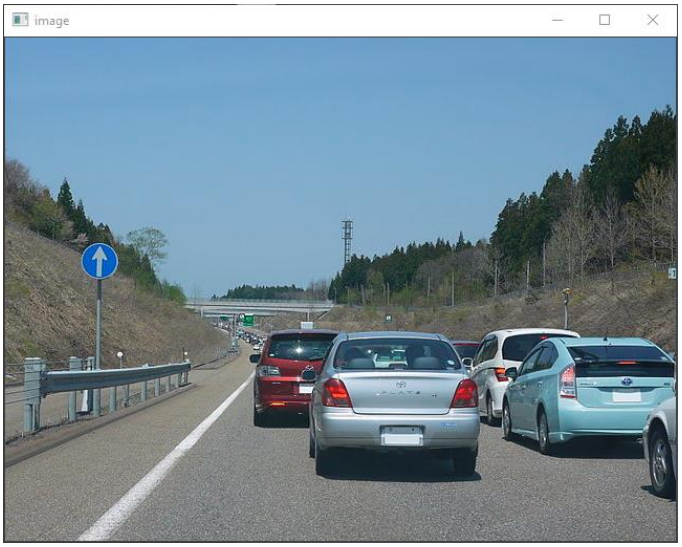
[1] https://commons.wikimedia.org/wiki/File:800_Houston_St_Manhattan_KS_3.jpg

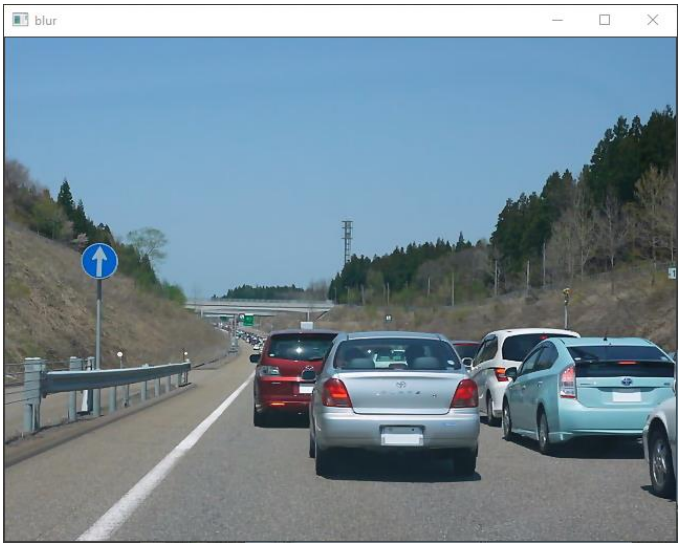
Canny Edge Detection

```
import cv2 as cv

image = cv.imread('traffic.jpg')
blur = cv.bilateralFilter(src=image, d=10, sigmaColor=75, sigmaSpace=75)

edges = cv.Canny(image=image, threshold1=100, threshold2=200)
blurEdges = cv.Canny(image=blur, threshold1=100, threshold2=200)

cv.imshow('image', image)
cv.imshow('blur', blur)
cv.imshow('edges', edges)
cv.imshow('blurEdges', blurEdges)

cv.waitKey(0)
```

Canny edge detection is a very powerful tool to detect edges, and results in a binary image where white pixels represent an edge.

Note how the blurred image results in a image with less false edges.



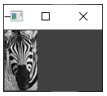


[1]

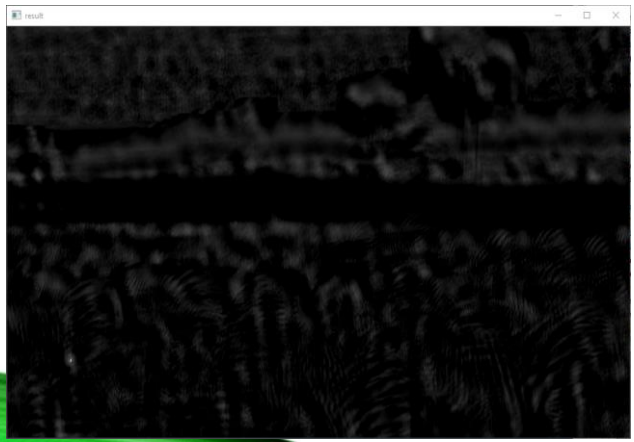
[1] https://commons.wikimedia.org/wiki/File:TrafficJam_near_Nakago_InterChange_20100501.jpg

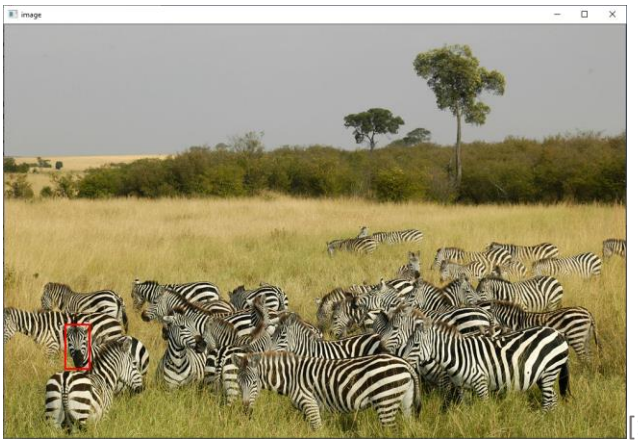
Template Matching

```
import cv2 as cv
import numpy as np

image= cv.imread("zebras.jpg")
imageGrey = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

template = cv.imread("zebraFace.png", 0) #note image must be same resolution
w, h = template.shape[:: -1] #swap width and height

result = cv.matchTemplate(image=imageGrey, templ=template, method=cv.TM_CCOEFF_NORMED)
threshold = 0.9
locations = np.where(result >= threshold)
locations = zip(*locations[::-1]) #reverses list and combines elements at the same index into xy tuples
for loc in locations:
    cv.rectangle(image, loc, (loc[0] + w, loc[1] + h) , (0,0,255), 2)#draw rectangle

cv.imshow("result", result) #light pixels represent better matches
cv.imshow("image", image)
cv.imshow("template", template)
cv.imshow("imageGrey", imageGrey)
cv.waitKey(0)
```



The result image represents how close of a match the template image is to the main image. A lighter pixel represents a higher match value. The np.where() then only takes the location where the pixels are above the specified value.



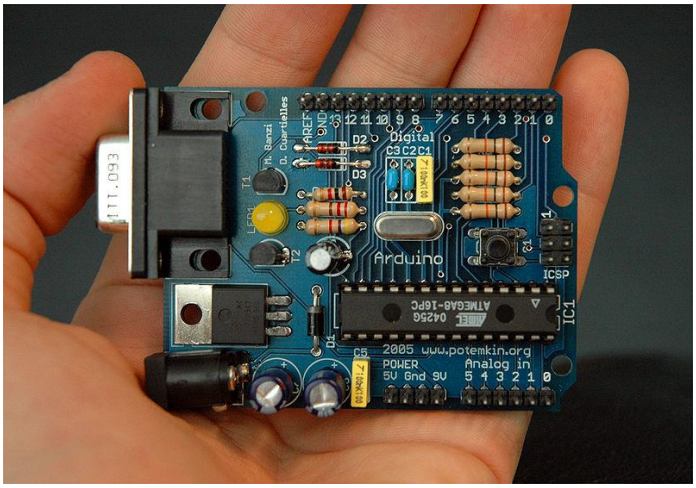


[1]

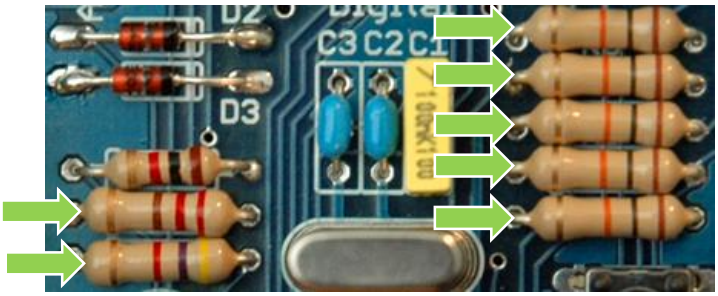
[1] https://commons.wikimedia.org/wiki/File:Common_zebra_at_masai_mara_kenya_01.jpg

Example: Template Matching Multiple Resistors

```
import cv2 as cv
import numpy as np

image= cv.imread("Arduino.jpg")
imageGrey = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

template = cv.imread("ArduinoResister.jpg", 0)
w, h = template.shape[:: -1] #swap width and height

result = cv.matchTemplate(imageGrey, template,
cv.TM_CCOEFF_NORMED)
threshold = 0.67
locations = np.where(result >= threshold)
locations = list(zip(*locations[::-1]))
counter = 0

for loc in locations:
    cv.rectangle(image, loc, (loc[0] + w, loc[1] + h),
(0,0,255), 2)
    counter = counter + 1

print("Resisters found: ", counter)
cv.imshow("result", result)
cv.imshow("image", image)
cv.imshow("template", template)
cv.imshow("imageGrey", imageGrey)
cv.waitKey(0)
```



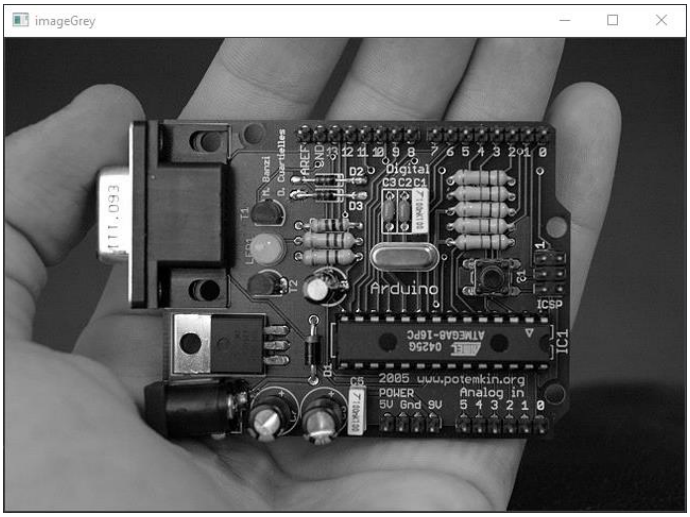
[1]



In this example we are wanting to find the 7 resistors that are all the same size (green arrow). Notice how they have different color strips (different Ohms). In order to template match similar images we need to lower the threshold.

[1] https://commons.wikimedia.org/wiki/File:Arduino316.jpg

Example: Template Matching Multiple Resistors



With a threshold of 0.67, it is saying 37 matches were found.

[1]

[1] https://commons.wikimedia.org/wiki/File:Arduino316.jpg

Example: Template Matching Multiple Resistors

```
import cv2 as cv
import numpy as np

image= cv.imread("Arduino.jpg")
imageGrey = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

template = cv.imread("ArduinoResister.jpg", 0)
w, h = template.shape[:: -1]

result = cv.matchTemplate(imageGrey, template, cv.TM_CCOEFF_NORMED)
threshold = 0.67
locations = np.where(result >= threshold)
locations = list(zip(*locations[::-1]))
counter = 0
rectangles = []

for loc in locations:
    rect = [int(loc[0]), int(loc[1]), w, h]
    rectangles.append(rect)  # Add every box to the list twice in order to retain single (non-overlapping) boxes
    rectangles.append(rect)
rectangles, weights = cv.groupRectangles(rectangles, groupThreshold=1, eps=0.2)  # remove the duplicate results

for i in range(len(rectangles)):
    cv.rectangle(image, (rectangles[i][0], rectangles[i][1]),
                (rectangles[i][0] + rectangles[i][2], rectangles[i][1] + rectangles[i][3]),
                     (0, 255, 0), 2)
    counter = counter + 1

print("Resisters found: ", counter)
cv.imshow("result", result)
cv.imshow("image", image)
cv.imshow("template", template)
cv.imshow("imageGrey", imageGrey)
cv.waitKey(0)
```

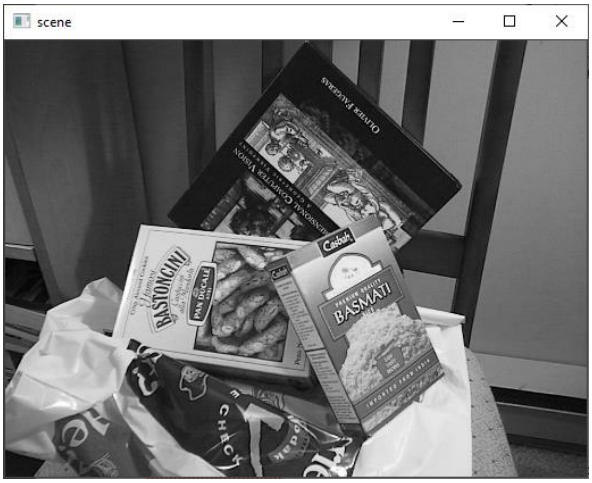
[1]

To solve the problem we can make sure each resistor has two boxes over it, then use the groupRectangles() function to remove the duplicates.

[1] https://commons.wikimedia.org/wiki/File:Arduino316.jpg

Brute Force Feature Matching

```
import cv2 as cv
import matplotlib.pyplot as plt

subImage = cv.imread('box.png', 0)
scene = cv.imread('box_in_scene.png', 0)

orb = cv.ORB_create()
keyPointSubImage, descriptorSubImage = orb.detectAndCompute(subImage, None)
keyPointScene, descriptorScene = orb.detectAndCompute(image=scene, mask=None)
bruteForce = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)
matches = bruteForce.match(descriptorSubImage,descriptorScene)
matches = sorted(matches, key = lambda x:x.distance)

MatchingImage =
cv.drawMatches(subImage,keyPointSubImage,scene,keyPointScene,matches[:10],None,
flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
cv.imshow("subImage", subImage)
cv.imshow("scene", scene)
plt.imshow(MatchingImage)
plt.show()
```



Feature matching matches the features of one images with the features of a second image. This is known as homography and allows rotation and tiling between the two images. It allows more flexibility in image variation than template matching.

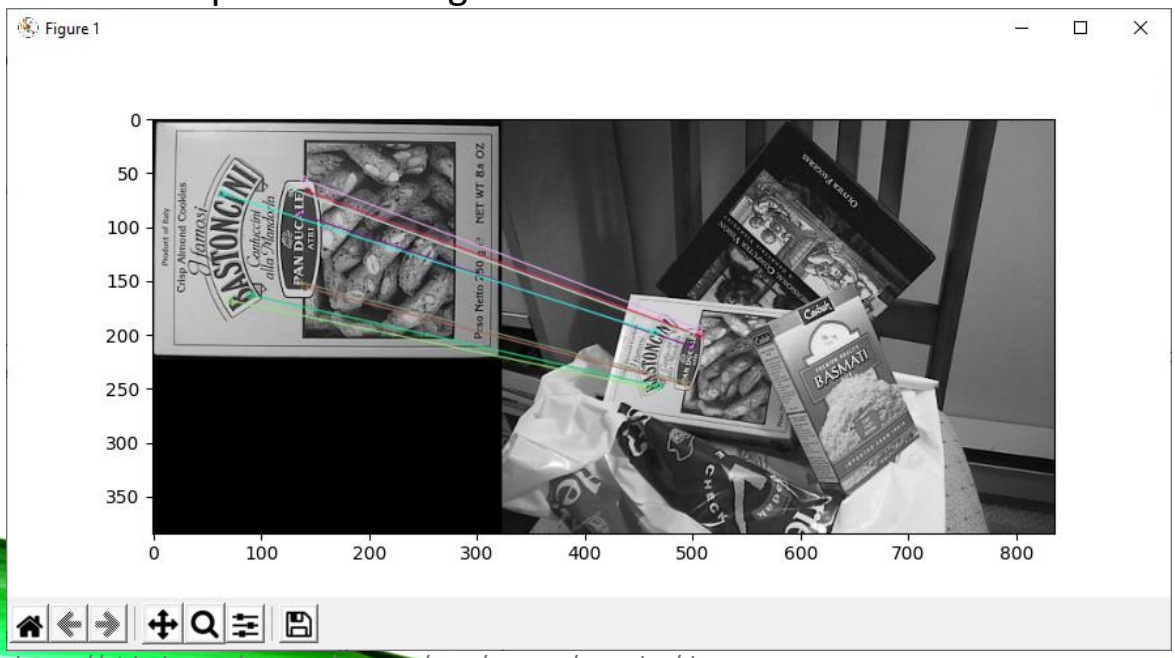


Image source: https://github.com/opencv/opencv/tree/master/samples/data

Feature Matching with FLANN

```
import cv2 as cv
import matplotlib.pyplot as plt

boxImage = cv.imread('box.png',0)        # queryImage
scene = cv.imread('box_in_scene.png',0) # trainImage

# Initiate SIFT detector
sift = cv.SIFT_create()
# find the keypoints and descriptors with SIFT
keyPoint1, descriptor1 = sift.detectAndCompute(boxImage,None)
keyPoint2, descriptor2 = sift.detectAndCompute(scene,None)

# FLANN parameters
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50)   # or pass empty dictionary
flann = cv.FlannBasedMatcher(index_params,search_params)
matches = flann.knnMatch(descriptor1,descriptor2,k=2)

# Need to draw only good matches, so create a mask
matchesMask = [[0,0] for i in range(len(matches))]

# ratio test as per Lowe's paper
for i,(m,n) in enumerate(matches):
    if m.distance < 0.7*n.distance:
        matchesMask[i]=[1,0]

draw_params = dict(matchColor = (0,255,0),
                   singlePointColor = (255,0,0),
                   matchesMask = matchesMask,
                   flags = cv.DrawMatchesFlags_DEFAULT)

img3 = cv.drawMatchesKnn(boxImage,keyPoint1,scene,keyPoint2,matches,None,**draw_params)
cv.imshow("boxImage", boxImage)
cv.imshow("scene", scene)
plt.imshow(img3)
plt.show()
cv.waitKey(0)
```

In this example we are using the FLANN based matcher. FLANN stands for Fast Library for Approximate Nearest Neighbor.

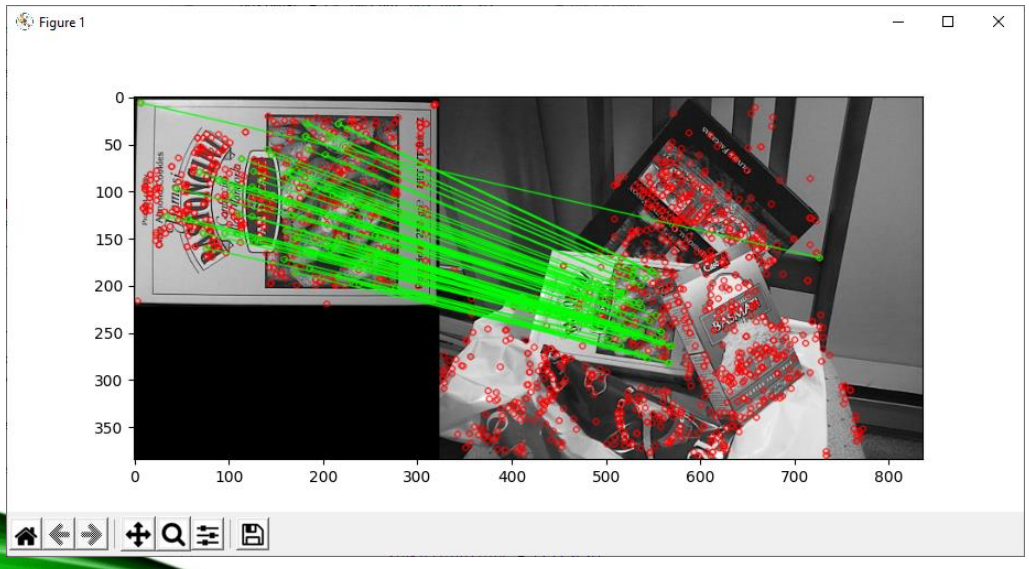


Image source: https://github.com/opencv/opencv/tree/master/samples/data

Video Example

Please see the video example
https://www.youtube.com/watch?v=zlpwFYsqq1c&feature=youtu.be