# Image processing

# OpenCV 2

# Installation

## pip

* Pip install opencv-python

# Basics

## version check

* Print(cv2.\_\_version\_\_)

## Sample opencv files

* <https://github.com/opencv/opencv>

## default color value

* Cv2 read BGR image

# Read images

img = cv2.imread(path + 'lena.jpg',1)

* Second parameter specifies the way image should read
* 1 – color
* 0 – gray scale
* -1 – load image including alpha channel

# Show image

cv2.imshow("image", img) # show image

cv2.waitKey(5000) # wait for 5000 sec

cv2.destroyAllWindows() # destroy window

# waitKey

cv2.waitKey(0) # show image until user press any keyboard key

## Keyboard key

import cv2

img = cv2.imread(path + 'lena.jpg',1)

cv2.imshow("image", img)

k = cv2.waitKey(0)

if k == 27: # esc key

cv2.destroyAllWindows()

elif k == ord('s'):

cv2.imwrite('lena\_copy.png', img)

cv2.destroyAllWindows()

## 64 bit sys

if cv2.waitKey(1) & 0xFF == ord('q'): # wait for keyboard press 0xFF - support 64 bit system

break

# Save image

cv2.imwrite('lena\_copy.png', img)

# Convert color

## BGR2GRAY

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY) # convert to gray

# Stream webcam video

cap = cv2.VideoCapture(path + 'vtest.avi') # argument - device index, url, video path

# 0 or -1 webcam default cam

while (cap.isOpened()): # check the source is valid

ret, frame = cap.read()

if ret:

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

cv2.imshow('frame', gray)

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

break

else:

break

cap.release()

cv2.destroyAllWindows()

# Split and merge - Channels

img = cv2.imread(path + 'lena.jpg',1)

print(img.shape) # rows, colums, channel

print(img.size) # total number of pixels

print(img.dtype) # Image datatype

b, g, r = cv2.split(img)

img = cv2.merge((b, g, r))

cv2.imshow("image", img)

cv2.waitKey(0)

cv2.destroyAllWindows()

# Cap properties

<https://docs.opencv.org/4.0.0/d4/d15/group__videoio__flags__base.html#gaeb8dd9c89c10a5c63c139bf7c4f5704d>

cap.isOpened() # check the source is valid or not

## cap get

print("Height",cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT)) # frame height

print("Width",cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH)) # frame width

## cap set

cap.set(3, 200) # cap.set(cv2.CAP\_PROP\_FRAME\_WIDTH, 1208)

cap.set(4, 200) # cap.set(cv2.CAP\_PROP\_FRAME\_HEIGHT, 720)

# SAVE video

Fourcc - <http://www.fourcc.org/codecs.php>

cap = cv2.VideoCapture(path + 'vtest.avi') # argument - device index, url, video path

# 0 or -1 webcam default cam

fourcc = cv2.VideoWriter\_fourcc(\*'XVID')

vwidth = int(cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

vheight = int(cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

print(vwidth,vheight)

out = cv2.VideoWriter("output.mp4", fourcc, 20.0 ,(vwidth,vheight))

# video name, fourcc value,

while (cap.isOpened()): # check the source is valid

ret, frame = cap.read()

if ret:

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

out.write(frame)

cv2.imshow('frame', gray)

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

break

else:

break

cap.release()

out.release()

cv2.destroyAllWindows()

# Draw geometric shape

## Line

img = cv2.imread(path + 'lena.jpg',1)

img = cv2.line(img, (0,0), (255,255), (255,0,0), 5)

# img frame, pos start, pos end, color(BGR), thickness number

cv2.imshow("image", img)

k = cv2.waitKey(0)

if k == 27:

cv2.destroyAllWindows()

## arrowed line

img = cv2.arrowedLine(img, (0,0), (255,255), (255,0,0), 5)

## rectangle

img = cv2.rectangle(img, (0,0), (255,255), (255,0,0), 5)

# img frame, top left, bottom right, color(BGR), thickness number

img = cv2.rectangle(img, (0,0), (255,255), (255,0,0), -1) # thickness -1: fill the color

## Circle

img = cv2.circle(img, (255,255), 50, (255,0,0), -1)

# img frame, center, radius, color(BGR), thickness number

## text

img = cv2.putText(img, "Opencv", (50, 500), font, 4, (255,255,255), 10, cv2.LINE\_AA)

# img frame, label, start, font,font size, color(BGR), thickness, line type

## Create numpy image

import numpy as np

img = np.zeros([512,512,3], np.uint8)

font = cv2.FONT\_HERSHEY\_SIMPLEX

img = cv2.putText(img, "Opencv", (50, 500), font, 4, (0,255,255), 10, cv2.LINE\_AA)

# img frame, label, start, font,font size, color(BGR), thickness, line type

cv2.imshow("image", img)

k = cv2.waitKey(0)

if k == 27:

cv2.destroyAllWindows()

# Draw current date and time

from datetime import datetime

cap = cv2.VideoCapture(path + 'vtest.avi')

font = cv2.FONT\_HERSHEY\_SIMPLEX

while (cap.isOpened()):

ret, frame = cap.read()

if ret:

text = str(datetime.now())

frame = cv2.putText(frame, text, (50, 500), font, 1, (0,255,255), 2, cv2.LINE\_AA)

cv2.imshow('frame', frame)

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

break

else:

break

cap.release()

cv2.destroyAllWindows()

# Mouse click

## Print all events

events = [i for i in dir(cv2) if "EVENT" in i]

print(events)

## Position and BGR value

import numpy as np

font = cv2.FONT\_HERSHEY\_SIMPLEX

def click\_event(event, x, y, flags, param):

if event == cv2.EVENT\_LBUTTONDOWN:

# print the position

strxy = str(x)+" "+str(y)

cv2.putText(img, strxy, (x, y), font, .5, (0,255,255), 1)

cv2.imshow("image", img)

if event == cv2.EVENT\_RBUTTONDOWN:

# print bgr value of position

blue = img[y, x, 0]

green = img[y, x, 1]

red = img[y, x, 2]

strbgr = str(blue) + " "+str(green)+" " + str(red)

cv2.putText(img, strbgr, (x, y), font, .5, (0,255,255), 1)

cv2.imshow("image", img)

img = np.zeros((512, 512, 3), np.uint8)

img = cv2.imread(path + 'lena.jpg',1)

cv2.imshow("image", img)

cv2.setMouseCallback("image", click\_event)

cv2.waitKey(0)

cv2.destroyAllWindows()

## Line drawing with mouse click

import numpy as np

font = cv2.FONT\_HERSHEY\_SIMPLEX

def click\_event(event, x, y, flags, param):

if event == cv2.EVENT\_LBUTTONDOWN:

cv2.circle(img, (x,y), 3, (0, 0, 255), -1)

points.append((x,y))

if len(points) >= 2:

cv2.line(img, points[-1], points[-2], (255,0,0), 5)

cv2.imshow("image", img)

img = cv2.imread(path + 'lena.jpg',1)

cv2.imshow("image", img)

points = []

cv2.setMouseCallback("image", click\_event)

cv2.waitKey(0)

cv2.destroyAllWindows()

## show color of point clicked

import numpy as np

font = cv2.FONT\_HERSHEY\_SIMPLEX

def click\_event(event, x, y, flags, param):

if event == cv2.EVENT\_LBUTTONDOWN:

blue = img[y, x, 0]

green = img[y, x, 1]

red = img[y, x, 2]

cv2.circle(img, (x,y), 3, (0, 0, 255), -1)

mycolorImage = np.zeros((512,512,3), np.uint8)

mycolorImage[:] = [blue, green, red]

cv2.imshow("color", mycolorImage)

cv2.imshow("image", img)

img = cv2.imread(path + 'lena.jpg',1)

cv2.imshow("image", img)

points = []

cv2.setMouseCallback("image", click\_event)

cv2.waitKey(0)

cv2.destroyAllWindows()

# ROI

Region of interest

## Copy ROI

img = cv2.imread(path + 'messi5.jpg',1)

ball = img[280:340, 330:390]

img[273:333, 100:160] = ball

cv2.imshow("image", img)

cv2.waitKey(0)

cv2.destroyAllWindows()

## Add two images with deferent shape

img = cv2.imread(path + 'messi5.jpg')

img2 = cv2.imread(path+ 'opencv-logo.png')

ball = img[280:340, 330:390]

img[273:333, 100:160] = ball

img = cv2.resize(img, (512,512))

img2 = cv2.resize(img2, (512,512))

dst = cv2.add(img, img2)

cv2.imshow("image", dst)

cv2.waitKey(0)

cv2.destroyAllWindows()

## Add two images with weight

img = cv2.imread(path + 'messi5.jpg')

img2 = cv2.imread(path+ 'opencv-logo.png')

ball = img[280:340, 330:390]

img[273:333, 100:160] = ball

img = cv2.resize(img, (512,512))

img2 = cv2.resize(img2, (512,512))

# dst = cv2.add(img, img2)

dst = cv2.addWeighted(img, .9, img2, .1, 0)

cv2.imshow("image", dst)

cv2.waitKey(0)

cv2.destroyAllWindows()

# Bitwise operations

## BITAND, OR, NOT, NOR

import numpy as np

img1 = np.zeros((250, 500, 3), np.uint8)

img1 = cv2.rectangle(img1, (200, 0), (300, 100), (255, 255, 255), -1)

img2 = np.zeros((250, 500, 3), np.uint8)

img2 = cv2.rectangle(img2, (250,0),(500,250),(255,255,255), -1)

img2 = cv2.rectangle(img2, (0,0),(250,250),(0,0,0), -1)

# bit\_and = cv2.bitwise\_and(img2, img1)

img1 = cv2.bitwise\_not(img1)

img2 = cv2.bitwise\_not(img2)

cv2.imshow("img1", img1)

cv2.imshow("img2", img2)

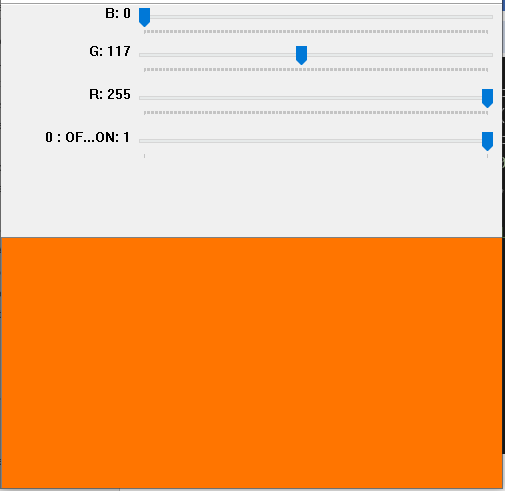
# cv2.imshow("bit\_and", bit\_not)

cv2.waitKey(0)

cv2.destroyAllWindows()

# Trackbar

## Example 1



import numpy as np

img = np.zeros((250, 500, 3), np.uint8)

cv2.namedWindow("image")

def nothing(x):

# print(x)

pass

cv2.createTrackbar("B", 'image', 0, 255, nothing)

cv2.createTrackbar("G", 'image', 0, 255, nothing)

cv2.createTrackbar("R", 'image', 0, 255, nothing)

# (trackbar name, named window, intial value trackbar set, last value, onChange function)

switch = "0 : OFF 1 : ON"

cv2.createTrackbar(switch, "image", 0, 1, nothing)

while (1):

cv2.imshow("image", img)

k = cv2.waitKey(1) & 0xFF

if k == 27:

break

b = cv2.getTrackbarPos("B", "image")

g = cv2.getTrackbarPos("G", "image")

r = cv2.getTrackbarPos("R", "image")

s = cv2.getTrackbarPos(switch, "image")

if s == 0:

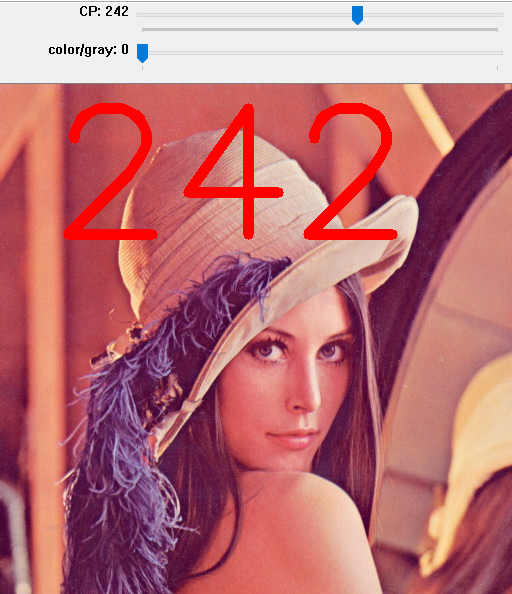
img[:] = 0

else:

img[:] = [b, g, r]

cv2.destroyAllWindows()

## Example 2



import numpy as np

cv2.namedWindow("image")

def nothing(x):

# print(x)

pass

font = cv2.FONT\_HERSHEY\_SIMPLEX

cv2.createTrackbar("CP", 'image', 10, 400, nothing)

# (trackbar name, named window, intial value trackbar set, last value, onChange function)

switch = "color/gray"

cv2.createTrackbar(switch, "image", 0, 1, nothing)

while (1):

img = cv2.imread(path + 'lena.jpg',1)

k = cv2.waitKey(1) & 0xFF

if k == 27:

break

pos = cv2.getTrackbarPos("CP", "image")

cv2.putText(img, str(pos), (50,150), font, 6, (0,0,255), 10)

s = cv2.getTrackbarPos(switch, "image")

if s == 0:

pass

else:

img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

cv2.imshow("image", img)

cv2.destroyAllWindows()

# HSV (Hue, Saturation and value)

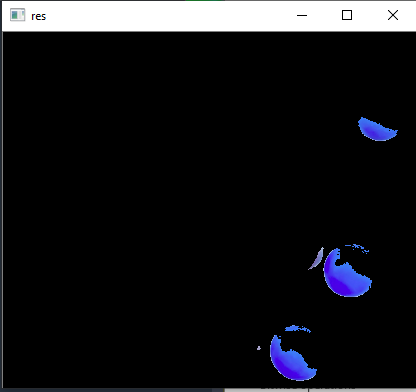
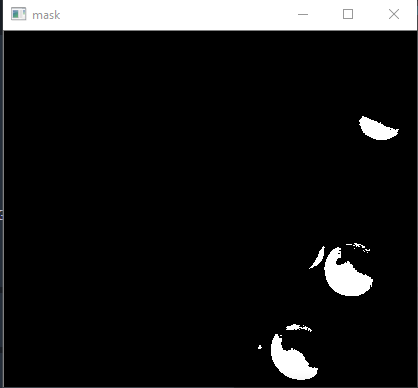
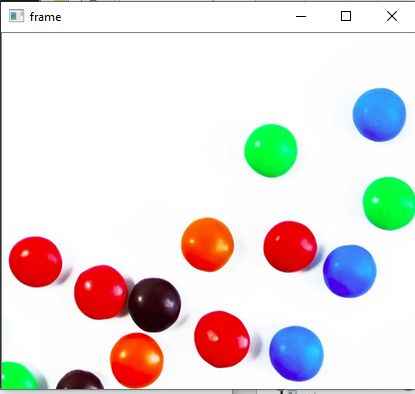
Hue – color components(base pigment) (0-360)

Saturation – amount of color(depth) (0-100%)

Value – brightness of color (0-100%)

# HSV object detection

## Blue color detection



img = cv2.imread(path + "smarties.png")

# cv2.namedWindow("Tracking")

while (1):

k = cv2.waitKey(1) & 0xFF

if k == 27:

break

frame = cv2.imread(path + "smarties.png")

hsv = cv2.cvtColor(frame,cv2.COLOR\_BGR2HSV)

# lower bound and upper bound of blue value

l\_b = np.array([110,50,50])

u\_b = np.array([130,255,255])

# create a mask of blue color

mask = cv2.inRange(hsv, l\_b, u\_b)

res = cv2.bitwise\_and(frame,frame, mask = mask)

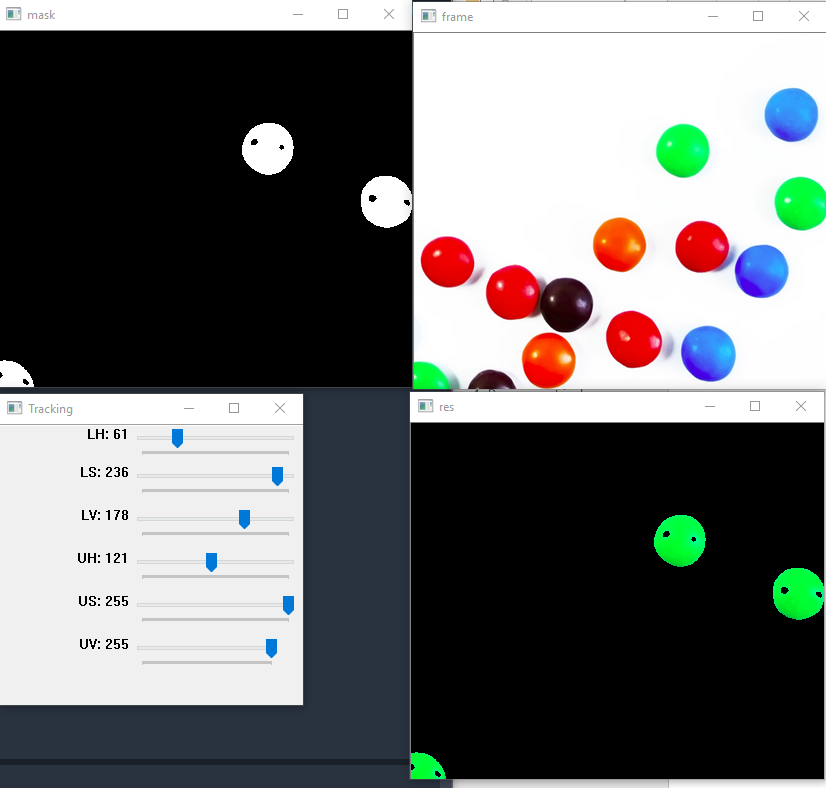
cv2.imshow("mask", mask)

cv2.imshow("frame", frame)

cv2.imshow("res", res)

cv2.destroyAllWindows()

## Color range detection – trackball



img = cv2.imread(path + "smarties.png")

cv2.namedWindow("Tracking")

def nothing(x):

pass

cv2.createTrackbar("LH", "Tracking", 0, 255, nothing)

cv2.createTrackbar("LS", "Tracking", 0, 255, nothing)

cv2.createTrackbar("LV", "Tracking", 0, 255, nothing)

cv2.createTrackbar("UH", "Tracking", 255, 255, nothing)

cv2.createTrackbar("US", "Tracking", 255, 255, nothing)

cv2.createTrackbar("UV", "Tracking", 255, 255, nothing)

while (1):

k = cv2.waitKey(1) & 0xFF

if k == 27:

break

frame = cv2.imread(path + "smarties.png")

l\_h = cv2.getTrackbarPos("LH", "Tracking")

l\_s = cv2.getTrackbarPos("LS", "Tracking")

l\_v = cv2.getTrackbarPos("LV", "Tracking")

u\_h = cv2.getTrackbarPos("UH", "Tracking")

u\_s = cv2.getTrackbarPos("US", "Tracking")

u\_v = cv2.getTrackbarPos("UV", "Tracking")

hsv = cv2.cvtColor(frame,cv2.COLOR\_BGR2HSV)

# lower bound and upper bound of blue value

l\_b = np.array([l\_h,l\_s,l\_v])

u\_b = np.array([u\_h,u\_s,u\_v])

# create a mask of blue color

mask = cv2.inRange(hsv, l\_b, u\_b)

res = cv2.bitwise\_and(frame,frame, mask = mask)

cv2.imshow("mask", mask)

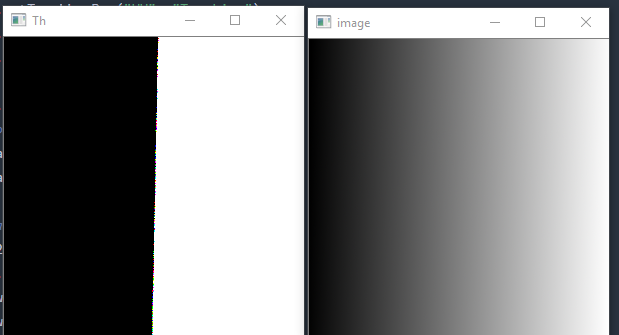
cv2.imshow("frame", frame)

cv2.imshow("res", res)

cv2.destroyAllWindows()

# Thresholding

## Binary threshold



img = cv2.imread(path + "gradient.png")

\_, th1 = cv2.threshold(img, 127,255,cv2.THRESH\_BINARY)

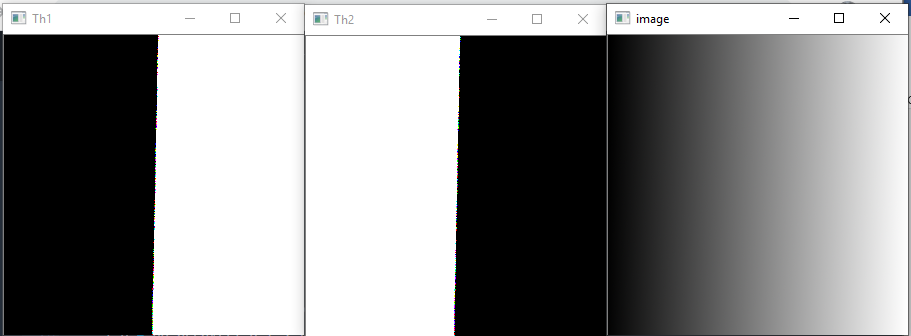
cv2.imshow("image", img)

cv2.imshow("Th", th1)

k = cv2.waitKey(0)

cv2.destroyAllWindows()

## binary inv



img = cv2.imread(path + "gradient.png")

\_, th1 = cv2.threshold(img, 127,255,cv2.THRESH\_BINARY)

\_, th2 = cv2.threshold(img, 127,255,cv2.THRESH\_BINARY\_INV)

cv2.imshow("image", img)

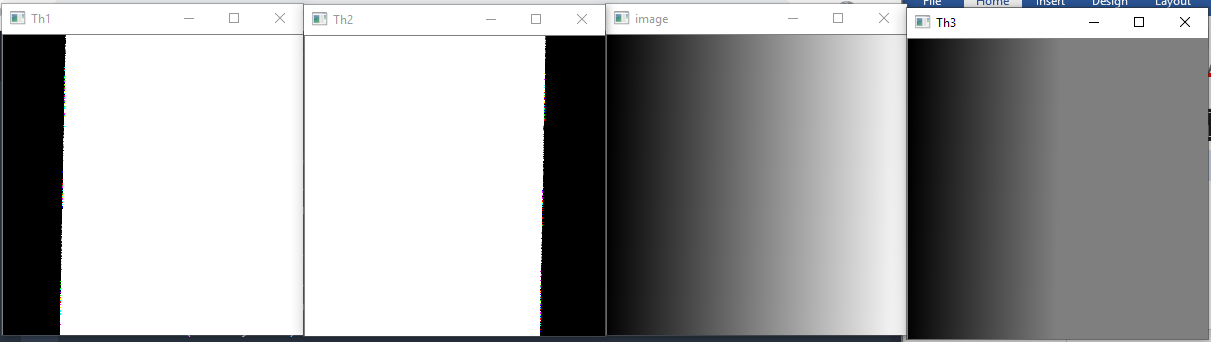
cv2.imshow("Th1", th1)

cv2.imshow("Th2", th2)

k = cv2.waitKey(0)

cv2.destroyAllWindows()

## trunc



img = cv2.imread(path + "gradient.png")

\_, th1 = cv2.threshold(img, 50,255,cv2.THRESH\_BINARY)

\_, th2 = cv2.threshold(img, 200,255,cv2.THRESH\_BINARY\_INV)

\_, th3 = cv2.threshold(img, 127,255,cv2.THRESH\_TRUNC)

# after 127 the pixel value will be same

cv2.imshow("image", img)

cv2.imshow("Th1", th1)

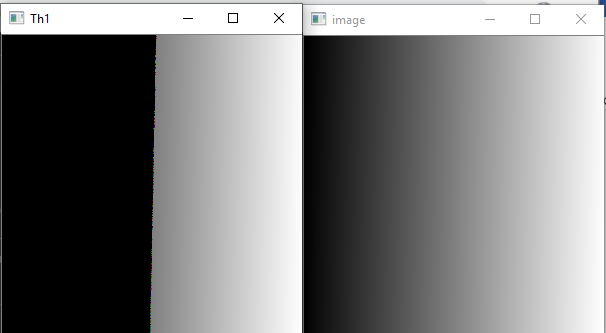
cv2.imshow("Th2", th2)

cv2.imshow("Th3", th3)

k = cv2.waitKey(0)

cv2.destroyAllWindows()

## Thresh\_tozero



img = cv2.imread(path + "gradient.png")

\_, th1 = cv2.threshold(img, 127,255,cv2.THRESH\_TOZERO)

# less than threshol value is zero else reamain same

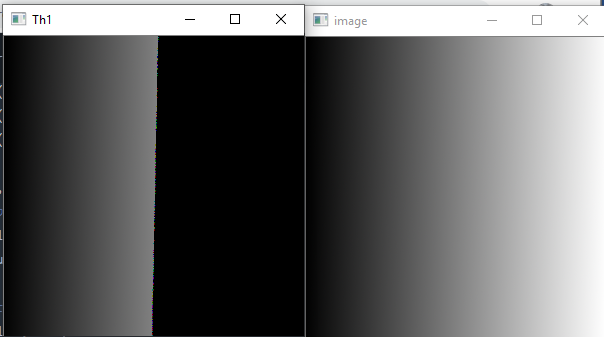
cv2.imshow("image", img)

cv2.imshow("Th1", th1)

k = cv2.waitKey(0)

cv2.destroyAllWindows()

## Thres\_tozero\_inv



\_, th1 = cv2.threshold(img, 127,255,cv2.THRESH\_TOZERO\_INV)

## Adaptive thresholding

img = cv2.imread(path + "sudoku.png",0)

# img = cv2.medianBlur(img,5)

# img = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE\_THRESH\_MEAN\_C,cv2.THRESH\_BINARY,11,2)

th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C,

cv2.THRESH\_BINARY,11,2)

# img, max, adoptive method - ADAPTIVE\_THRESH\_MEAN\_C - mean of neighbourhood area,

# threshold type, block size, C - substract from mean

cv2.imshow("image", img)

cv2.imshow("Th2", th2)

cv2.imshow("Th3", th3)

k = cv2.waitKey(0)

cv2.destroyAllWindows()

# Matplotlib

## imshow

from matplotlib import pyplot as plt

%matplotlib inline

img = cv2.imread(path + "lena.jpg",-1)

cv2.imshow('image', img)

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

plt.imshow(img)

cv2.waitKey(0)

cv2.destroyAllWindows()

## Subplot

from matplotlib import pyplot as plt

%matplotlib inline

img = cv2.imread(path + "gradient.png",1)

\_, th1 = cv2.threshold(img, 50,255,cv2.THRESH\_BINARY)

\_, th2 = cv2.threshold(img, 200,255,cv2.THRESH\_BINARY\_INV)

\_, th3 = cv2.threshold(img, 127,255,cv2.THRESH\_TRUNC)

\_, th4 = cv2.threshold(img, 127,255,cv2.THRESH\_TOZERO)

\_, th5 = cv2.threshold(img, 127,255,cv2.THRESH\_TOZERO\_INV)

titles = ["orginal img","BINARY", "BINARY\_INV", "TRUNC", "TOZERO", "TOZERO\_INV"]

images = [img, th1,th2,th3,th4,th5]

for i in range(6):

plt.subplot(2,3, i+1), plt.imshow(images[i],"gray")

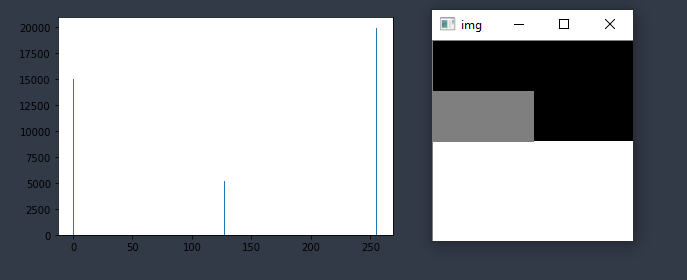
plt.title(titles[i])

plt.xticks([]),plt.yticks([])

# rows, colums, index of img, img

plt.show()

## color distribution



from matplotlib import pyplot as plt

%matplotlib inline

img = np.zeros((200,200), np.uint8)

cv2.rectangle(img, (0,100), (200,200),(255),-1)

cv2.rectangle(img, (0,50), (100,100),(127),-1)

plt.hist(img.ravel(), 256,[0,256])

plt.show()

cv2.imshow("img", img)

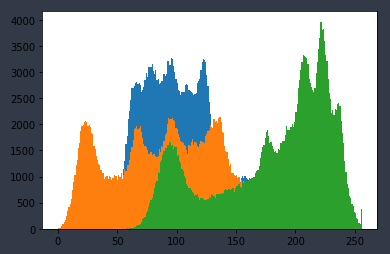
cv2.waitKey(0)

cv2.destroyAllWindows()

## b,g,r color distribution

Applications

1. Calculate the lighting condition of picture



from matplotlib import pyplot as plt

%matplotlib inline

img = cv2.imread(path + 'lena.jpg', 1)

b, g, r = cv2.split(img)

cv2.imshow("img", img)

cv2.imshow("img", b)

cv2.imshow("img", g)

cv2.imshow("img", r)

plt.hist(b.ravel(), 256,[0,256])

plt.hist(g.ravel(), 256,[0,256])

plt.hist(r.ravel(), 256,[0,256])

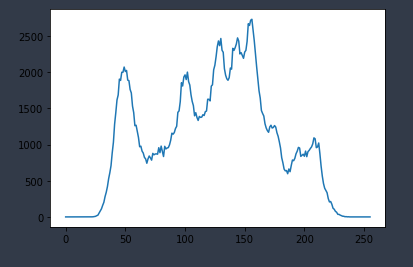
# plt.hist(img.ravel(), 256,[0,256])

plt.show()

cv2.waitKey(0)

cv2.destroyAllWindows()

## opencv-hist



from matplotlib import pyplot as plt

%matplotlib inline

img = cv2.imread(path + 'lena.jpg', 0)

hist = cv2.calcHist([img], [0],None, [256], [0, 256])

plt.plot(hist)

plt.show()

cv2.waitKey(0)

cv2.destroyAllWindows()

# morphological transformation

## Dilation, erosion, opening, closing, GRADIENT, TOPHAT

from matplotlib import pyplot as plt

%matplotlib inline

img = cv2.imread(path + "smarties.png",0)

\_, mask = cv2.threshold(img, 220, 255, cv2.THRESH\_BINARY\_INV)

mask = cv2.imread(path + "mask.png",0)

kernel = np.ones((5,5), np.uint8)

dilation = cv2.dilate(mask, kernel, iterations=2)

# smoothing inner area of shape

erosion = cv2.erode(mask, kernel, iterations=1)

# sharpening edges

opening = cv2.morphologyEx(mask, cv2.MORPH\_OPEN, kernel)

# opening - errorsion followed by dilation

closing = cv2.morphologyEx(mask, cv2.MORPH\_CLOSE, kernel)

mg = cv2.morphologyEx(mask, cv2.MORPH\_GRADIENT, kernel)

th = cv2.morphologyEx(mask, cv2.MORPH\_TOPHAT, kernel)

titles = ['image', 'mask', 'dilation', 'erosion', 'opening', 'closing', 'Gradient', "TH"]

images = [img, mask, dilation, erosion, opening, closing, mg,th]

for i in range(len(images)):

plt.subplot(3,3,i+1), plt.imshow(images[i], 'gray')

plt.title(titles[i])

plt.xticks([]),plt.yticks([])

plt.show()

# Smoothing, bluring

from matplotlib import pyplot as plt

%matplotlib inline

img = cv2.imread(path + "baboon.jpg",1)

# img = cv2.imread( "table1.jpg")

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

# homogeneouse filter

kernel = np.ones((5,5), np.float32) / 25

dst = cv2.filter2D(img, -1, kernel)

blur = cv2.blur(img, (5,5))

# gausian filter - high freequency noice

gblur = cv2.GaussianBlur(img, (5,5), 0)

# median filter - reaplace each pixel with median of neighbouring value

# It is also known as salt and pepper noice

median = cv2.medianBlur(img, 5)

# preserve the edges of image

bilater = cv2.bilateralFilter(img, 9, 75,75)

# cv2.imshow("table", median)

# cv2.waitKey(0)

# cv2.destroyAllWindows()

titles = ['image', '2d convolution', 'Blur', 'gblur', 'median', 'bilater']

images = [img, dst, blur, gblur, median, bilater]

for i in range(len(images)):

plt.subplot(2,3,i+1), plt.imshow(images[i])

plt.title(titles[i])

plt.xticks([]),plt.yticks([])

plt.show()

# Gradient and edge detection

## Laplacian, sobelX, sobelY

from matplotlib import pyplot as plt

%matplotlib inline

img = cv2.imread(path + " sudoku.png", cv2.IMREAD\_GRAYSCALE)

img = cv2.imread("table1.jpg", cv2.IMREAD\_GRAYSCALE)

lap = cv2.Laplacian(img, cv2.CV\_64F, ksize = 1)

lap = np.uint8(np.absolute(lap))

sobelX = cv2.Sobel(img, cv2.CV\_64F, 0, 1)

sobelY = cv2.Sobel(img, cv2.CV\_64F, 1, 0)

# img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

sobelX = np.uint8(np.absolute(sobelX))

sobelY = np.uint8(np.absolute(sobelY))

titles = ['image', 'Laplacian', 'sobelX', 'sobelY']

images = [img, lap, sobelX, sobelY]

cv2.imshow("table", lap)

cv2.waitKey(0)

cv2.destroyAllWindows()

for i in range(len(images)):

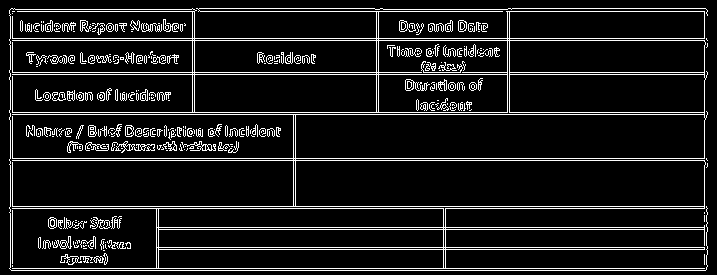
plt.subplot(2,3,i+1), plt.imshow(images[i], 'gray')

plt.title(titles[i])

plt.xticks([]),plt.yticks([])

plt.show()

Canny



from matplotlib import pyplot as plt

%matplotlib inline

img = cv2.imread(path + "sudoku.png", cv2.IMREAD\_GRAYSCALE)

img = cv2.imread("table1.jpg", cv2.IMREAD\_GRAYSCALE)

# Canny detect wide range of edges in images

# 1.Noise reduction

# 2. Gradient calculation

# 3. Non-maximum suppression

# 4. double threshold

# 5. Edge Tracking by Hysteresis

canny = cv2.Canny(img, 100,200)

titles = ['image', 'Canny']

images = [img, canny]

cv2.imshow("table", canny)

cv2.waitKey(0)

cv2.destroyAllWindows()

for i in range(len(images)):

plt.subplot(1,2,i+1), plt.imshow(images[i], 'gray')

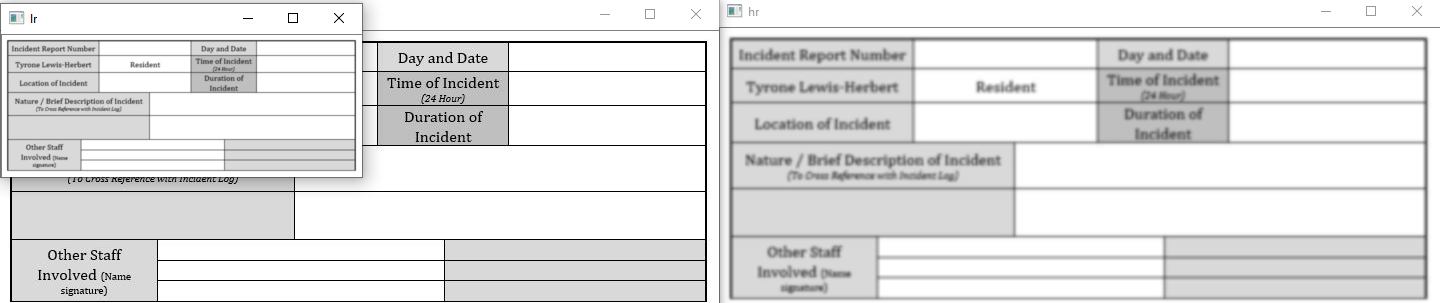
plt.title(titles[i])

plt.xticks([]),plt.yticks([])

plt.show()

# Pyramid reprasentation

## basics



img = cv2.imread("table1.jpg")

lr = cv2.pyrDown(img)

hr = cv2.pyrUp(lr)

cv2.imshow("table", img)

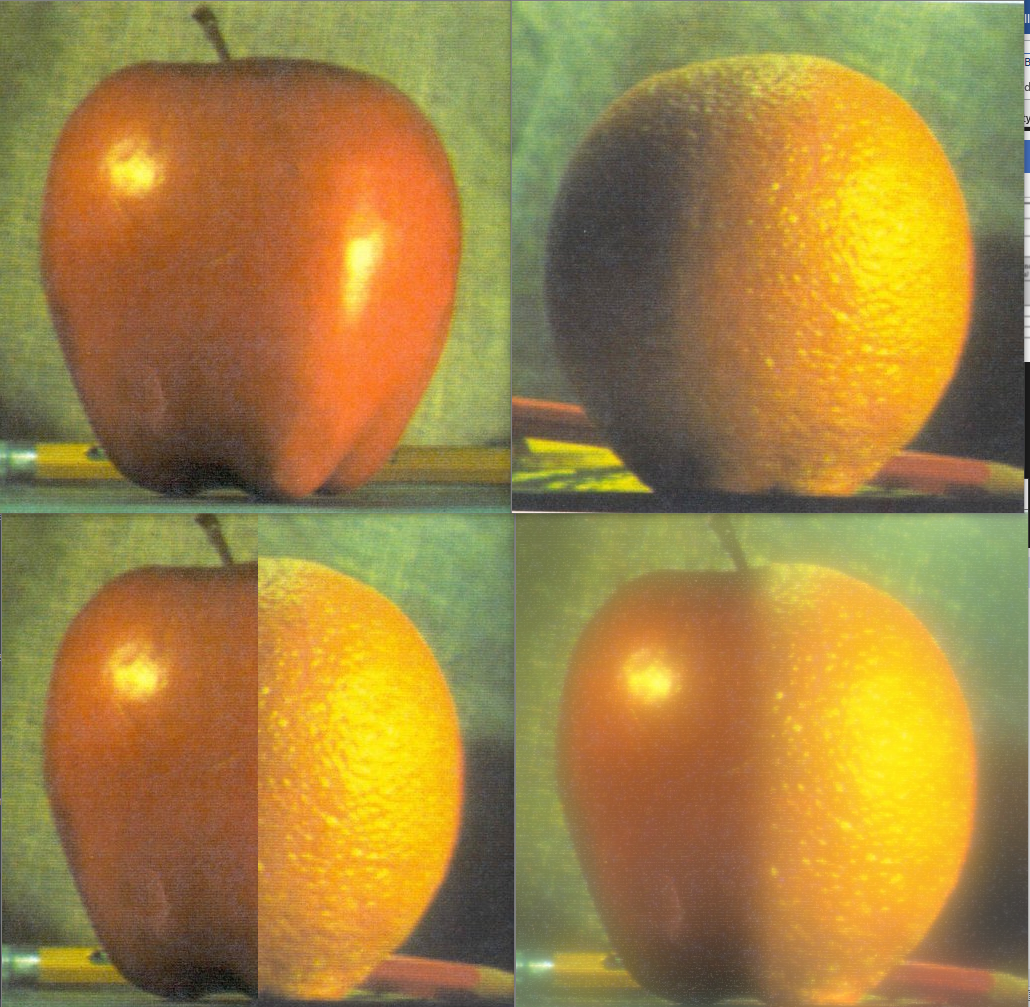
cv2.imshow("lr", lr)

cv2.imshow("hr", hr)

cv2.waitKey(0)

cv2.destroyAllWindows()

## Stack images - blend



apple = cv2.imread(path + "apple.jpg")

orange = cv2.imread(path + "orange.jpg")

apple\_orange = np.hstack((apple[:,:256], orange[:,256:]))

# blend image

# 1. Find gaussian pyramids for apple and orange (num level around 6)

# 2. From gaussian pyramid find Laplacian pyramid

# 3. join left and right half in each level of laplacian

# 4. Joint image pyramid reconstruct orginal image

apple\_copy = apple.copy()

gp\_apple = [apple\_copy]

for i in range(6):

apple\_copy = cv2.pyrDown(apple\_copy)

gp\_apple.append(apple\_copy)

orange\_copy = orange.copy()

gp\_orange = [orange\_copy]

for i in range(6):

orange\_copy = cv2.pyrDown(orange\_copy)

gp\_orange.append(orange\_copy)

# laplacian pyramid

apple\_copy = gp\_apple[5]

lp\_apple = [apple\_copy]

for i in range(5,0,-1):

gaussian\_extended = cv2.pyrUp(gp\_apple[i])

laplacian = cv2.subtract(gp\_apple[i-1], gaussian\_extended)

lp\_apple.append(laplacian)

orange\_copy = gp\_orange[5]

lp\_orange = [orange\_copy]

for i in range(5,0,-1):

gaussian\_extended = cv2.pyrUp(gp\_orange[i])

laplacian = cv2.subtract(gp\_orange[i-1], gaussian\_extended)

lp\_orange.append(laplacian)

# join the half of two images

apple\_orange\_pyr = []

n = 0

for apple\_lap, orange\_lap in zip(lp\_apple, lp\_orange):

n += 1

cols, rows, ch = apple\_lap.shape

laplacian = np.hstack((apple\_lap[:,0:int(cols/2)], orange\_lap[:,int(cols/2):]))

apple\_orange\_pyr.append(laplacian)

# now reconstruct

apple\_orange\_reconstruct = apple\_orange\_pyr[0]

for i in range(1, 6):

apple\_orange\_reconstruct = cv2.pyrUp(apple\_orange\_reconstruct)

apple\_orange\_reconstruct = cv2.add(apple\_orange\_pyr[i], apple\_orange\_reconstruct)

cv2.imshow("apple", apple)

cv2.imshow("orange", orange)

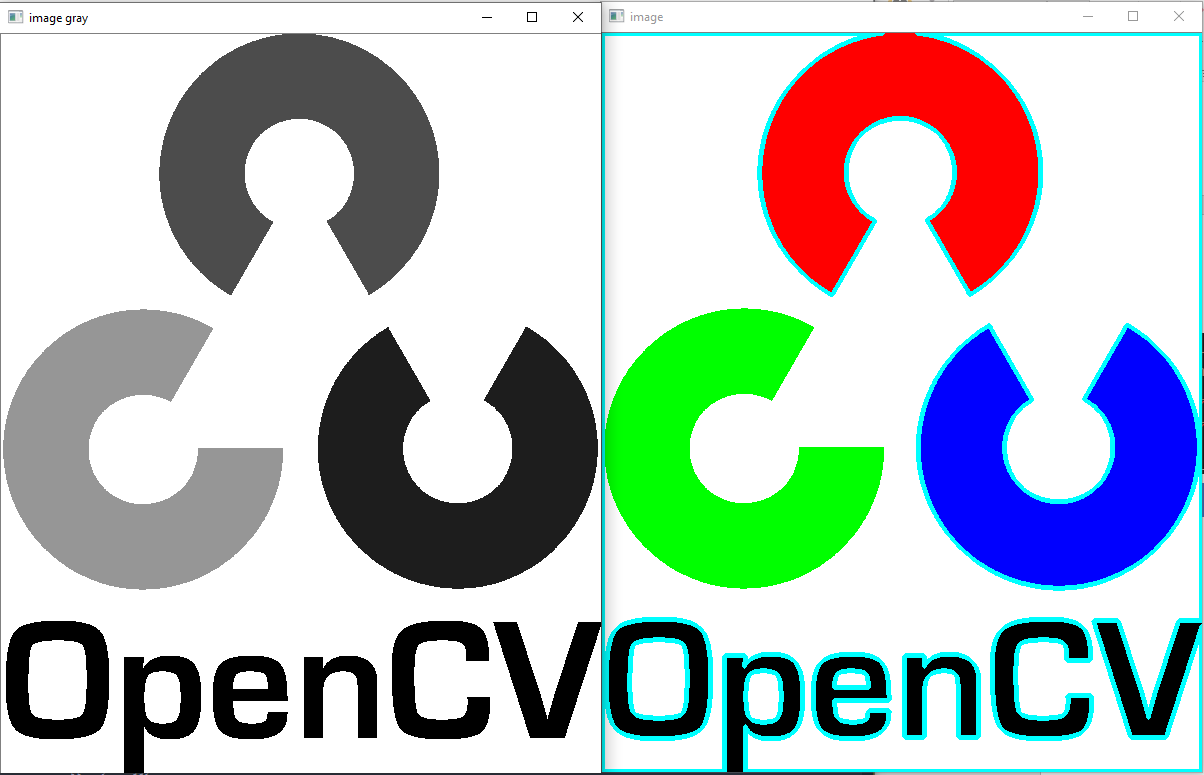
cv2.imshow("apple\_orange", apple\_orange)

cv2.imshow("apple\_orange\_reconstruct", apple\_orange\_reconstruct)

cv2.waitKey(0)

cv2.destroyAllWindows()

# Contours



img = cv2.imread(path + 'opencv-logo.png')

imgray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

ret, thresh = cv2.threshold(imgray, 127, 255, 0)

hierarchy, contours, \_ = cv2.findContours(thresh,cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_NONE)

cv2.drawContours(img, contours, -1, (255,255,0), 3)

# drawContours(image frame, contour matrix, -1 = all contour, color, thickness)

# contours = np array

print("number of contours = ", len(contours))

cv2.imshow("image",img)

cv2.imshow("image gray",imgray)

cv2.waitKey(0)

cv2.destroyAllWindows()

## Motion detection

import cv2

import numpy as np

path = "opencv-master\\samples\\data\\"

cap = cv2.VideoCapture(path + 'vtest.avi')

ret, frame1 = cap.read()

ret, frame2 = cap.read()

while cap.isOpened():

if ret:

diff = cv2.absdiff(frame1, frame2)

gray = cv2.cvtColor(diff, cv2.COLOR\_BGR2GRAY)

blur = cv2.GaussianBlur(gray, (5,5), 0)

\_, thresh = cv2.threshold(blur, 20, 255, cv2.THRESH\_BINARY)

dilated = cv2.dilate(thresh, None, iterations=3)

hierarchy, contours, \_ = cv2.findContours(dilated,cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

for contour in contours:

(x, y, w, h) = cv2.boundingRect(contour)

if cv2.contourArea(contour) < 900:

continue

cv2.rectangle(frame1, (x,y),(x+w,y+h), (0,255,0), 2)

cv2.putText(frame1,"status: {}".format("Movement"), (10,20), cv2.FONT\_HERSHEY\_SIMPLEX,

1, (0,0,255), 3)

# cv2.drawContours(frame1, contours, -1, (0, 255,0), 2)

cv2.imshow("inter", frame1)

frame1 = frame2

ret, frame2 = cap.read()

else:

break

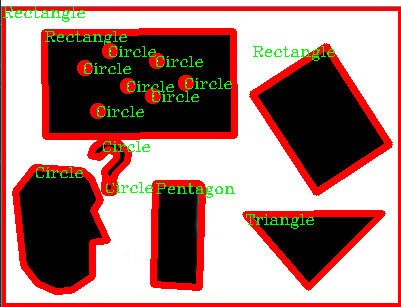
if cv2.waitKey(40) == 27:

break

cv2.destroyAllWindows()

cap.release()

## Shape detection



img = cv2.imread(path + 'right01.jpg')

imGray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

\_, thresh = cv2.threshold(imGray, 240, 255, cv2.THRESH\_BINARY)

hierarchy, contours, \_ = cv2.findContours(thresh,cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_NONE)

for contour in contours:

approx = cv2.approxPolyDP(contour, 0.01 \* cv2.arcLength(contour, True), True)

cv2.drawContours(img, [approx], 0, (0,0,255), 5)

x = approx.ravel()[0]

y = approx.ravel()[1]

if len(approx) == 3:

cv2.putText(img, "Triangle", (x, y+10), cv2.FONT\_HERSHEY\_COMPLEX, 0.5,(0, 255, 0))

elif len(approx) == 4:

x, y, w, h = cv2.boundingRect(approx)

aspect\_ratio = float(w) / h

print(aspect\_ratio)

if aspect\_ratio >= 0.95 and aspect\_ratio <= 1.05:

cv2.putText(img, "Square", (x, y+10), cv2.FONT\_HERSHEY\_COMPLEX, 0.5,(0, 255, 0))

else:

cv2.putText(img, "Rectangle", (x, y+10), cv2.FONT\_HERSHEY\_COMPLEX, 0.5,(0, 255, 0))

elif len(approx) == 5:

cv2.putText(img, "Pentagon", (x, y+10), cv2.FONT\_HERSHEY\_COMPLEX, 0.5,(0, 255, 0))

elif len(approx) == 10:

cv2.putText(img, "Star", (x, y+10), cv2.FONT\_HERSHEY\_COMPLEX, 0.5,(0, 255, 0))

else:

cv2.putText(img, "Circle", (x, y+10), cv2.FONT\_HERSHEY\_COMPLEX, 0.5,(0, 255, 0))

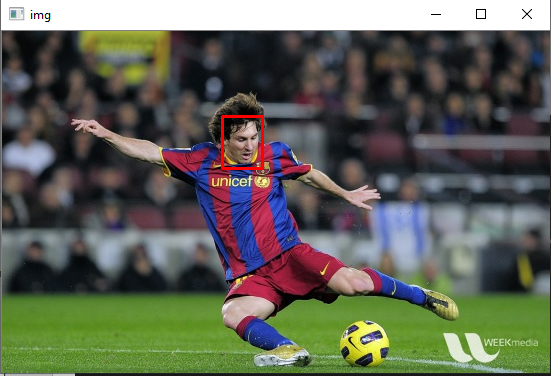
cv2.imshow("shapes", img)

cv2.waitKey(0)

cv2.destroyAllWindows()

# Template matching

## cv2.matchTemplate



import numpy as np

img = cv2.imread(path+"messi5.jpg")

template = cv2.imread(path+"messi\_face.jpg", 0)

gray\_img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

res = cv2.matchTemplate(gray\_img, template, cv2.TM\_CCOEFF\_NORMED)

w, h = template.shape[::-1]

print(res)

threshold = 0.9

loc = np.where(res >= threshold)

print(loc)

for pt in zip(\*loc[::-1]):

cv2.rectangle(img, pt, (pt[0] + w, pt[1] + h), (0, 0, 255), 2)

cv2.imshow("img", img)

cv2.waitKey(0)

cv2.destroyAllWindows()

## Hough transform

The hough Transform is a popular technique to detect any shape, if you can represent that shape in a mathematical form. It can detect the shape even if it is broken or distorted a little bit.

* A line in the image space can be expressed with two variables. For example:
* In the catesian coordinate system yi = mxi + c

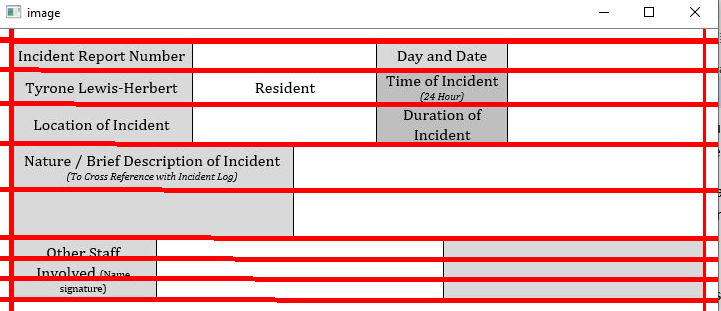
Algorithm

1. Edge detection, eg. Using the canny edge detector.
2. Mapping of edge points to the Hough space and storage in an accumulator.
3. Interpretation of the accumulator to yield lines of infinite length. The interpretation is done by thresholding and possibly other constraints.
4. Conversion of infinite lines to finite lines.

Opencv implements two kind of hough line transforms

* The standard hough transform
* The probabilistic hough line transform

### cv2.HoughLines



img = cv2.imread(path + "sudoku.png")

img = cv2.imread("table1.jpg")

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

edges = cv2.Canny(gray, 50, 150, apertureSize = 3)

lines = cv2.HoughLines(edges, 1, np.pi / 180, 200)

# lines = cv2.HoughLines(image, rho, theta, threshold)

# image: source image

# lines: output vector of lines. Each line is represented by a 2 or 3 element vector (rho, theta)

# or (rho, theta, votes).

# rho is the distance from the coordinate orgin (0, 0) (top-left corner of the image). theta is the

# line rotation angle in radians. votes is the value of accumulator

# rho: Distance reolution of the accumulator in pixels

# theta: Angle resolution of the accumulator in pixels

# threshold: Accumulator threshold parameter. Only those lines are returned that get enough votes

for line in lines:

rho, theta = line[0]

a = np.cos(theta)

b = np.sin(theta)

x0 = a + rho

y0 = b + rho

# x1 stores the rounded off value of (r \* cos(theta) - 1000 \* sin(theta))

x1 = int(x0 + 1000 \* (-b))

#y1 stores the rounded off value of (r\* sin(theta) + 1000 \* cos(theta))

y1 = int(y0 + 1000 \* (a))

# x2 stores the rounded off value of (r \* cos(theta) + 1000 \* sin(theta))

x2 = int(x0 -1000 \* (-b))

# y2 stores the rounded off value of (r \* sin(theta) - 1000 \* cos(theta))

y2 = int(y0 - 1000 \* (a))

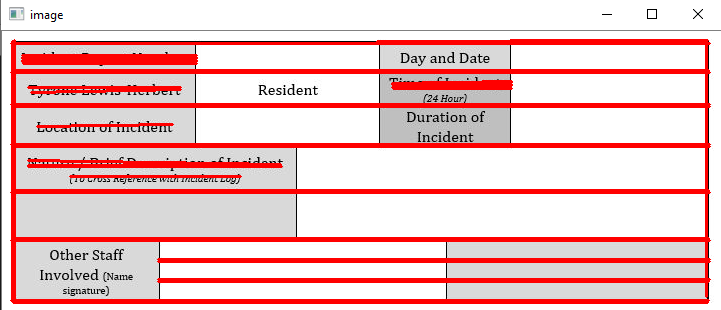
cv2.line(img,(x1, y1), (x2, y2), (0, 0, 255), 2)

cv2.imshow("image", img)

k = cv2.waitKey(0)

cv2.destroyAllWindows()

### cv2.HoughLinesP



img = cv2.imread(path + "sudoku.png")

img = cv2.imread("table1.jpg")

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

edges = cv2.Canny(gray, 50, 150, apertureSize = 3)

lines = cv2.HoughLinesP(edges, 1, np.pi / 180, 100,minLineLength = 100, maxLineGap = 10)

for line in lines:

x1, y1, x2, y2 = line[0]

cv2.line(img,(x1, y1), (x2, y2), (0, 0, 255), 2)

cv2.imshow("edges", edges)

cv2.imshow("image", img)

k = cv2.waitKey(0)

cv2.destroyAllWindows()