```
# *** Video Processing
# when running video files mk sure only 1 kernel is running
# connecting to the camera
import cv2
# Connects to your computer's default camera
cap = cv2.VideoCapture(0)
# a VC is a series of imgs
# a grp of imgs getting updtd continually


# Get the width and height from video feed
# (returns float which we need to convert to integer for later)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
            # tuple unpack
    # Our operations on the frame come here
            # convert it to gray frame
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Display the resulting frame
    cv2.imshow('frame',gray)
    # This command let's us quit with the "q" button on a keyboard.
    # Simply pressing X on the window won't work!
    if cv2.waitKey(1) & 0xFF == ord('q'):
```

```
        break
# When everything done, release the capture and destroy the windows
cap.release() # stop capturing the video
cv2.destroyAllWindows()
```

```
# what is the diff here
cap = cv2.VideoCapture(0)
while True:
    ret, frame = cap.read()
    cv2.imshow('frame',frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release() # stop capturing the video
cv2.destroyAllWindows()
```

```
# ## Writing a Video Stream to File

# ### Notebook :  Make sure its all in the same cell!

# FourCC is a 4-byte code used to specify the video codec.

# The list of available codes can be found in fourcc.org. It is platform dependent.

# INFO ON CODECS: https://docs.opencv.org/3.0-
beta/doc/py_tutorials/py_gui/py_video_display/py_video_display.html#saving-a-video

cap = cv2.VideoCapture(0)

width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))

height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

# MACOS AND LINUX: *'XVID' (MacOS users try VIDX )

# WINDOWS *'VIDX'

writer = cv2.VideoWriter('my_capture.mp4', cv2.VideoWriter_fourcc(*'DIVX'),25, (width, height))

#writer = cv2.VideoWriter('my_capture.mp4', cv2.VideoWriter_fourcc(*'XVID'),25, (width, height))

# cv2.VideoWriter_fourcc(*'XVID') is the video codec
```

```
# ths is dfrnt from ur OS

# fourcc is a 4 byte code used to specify video codec

# 25 is the num of frames/sec v want to record

# the more the frames/sec the largr the file


## This loop keeps recording until you hit Q or escape the window

## instead use some sort of timer, like from time import sleep and then just record for 5 seconds.

while True:

    # Capture frame-by-frame

    ret, frame = cap.read()

    # Write the video

    writer.write(frame)

    # Display the resulting frame

    cv2.imshow('frame',frame)

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

        break

cap.release()

writer.release()

cv2.destroyAllWindows()
```

```
# # OpenCV with Video Files

# open the recorded video from the last ,

# you can use this code to open any major video format.

# Run everything in one cell!

import cv2

import time

# Same command function as streaming, its just now we pass in the file path, nice!

cap = cv2.VideoCapture('my_capture.mp4')

# FRAMES PER SECOND FOR VIDEO

fps = 25


# check if the video was acutally there

# If you get an error at thsi step, triple check your file path!!

if cap.isOpened()== False:

   print("Error opening the video file. Please double check your file path for typos. Or move the movie file to the same location as this script/notebook")
```

```
# While the video is opened
while cap.isOpened():
    # Read the video file.
    ret, frame = cap.read()
    # If we got frames, show them.
    if ret == True:
        # Display the frame at same frame rate of recording
        #time.sleep(1/fps)
        cv2.imshow('frame',frame)
        # Press q to quit
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break
    # Or automatically break this whole loop if the video is over.
    else:
        break
cap.release()
cv2.destroyAllWindows()
```

```
# wht is the dfrnc

cap = cv2.VideoCapture('my_capture.mp4')

fps = 25

if cap.isOpened()== False:

   print("Error opening the video file. Please double check your file path for typos. Or move the movie file to the same location as this script/notebook")

while cap.isOpened():

   ret, frame = cap.read()

   if ret == True:

      time.sleep(1/fps)

      cv2.imshow('frame',frame)

      if cv2.waitKey(25) & 0xFF == ord('q'):

         break

   else:

      break

cap.release()

cv2.destroyAllWindows()
```

```
# # Drawing on Video
# to analyze video using techniques like object detection or facial recognition,
# we want to draw an image on the video, like a box around a face.
cap = cv2.VideoCapture(0)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
```

```
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

# using // here because Python // allows for int classical division,
# because we can't pass a float to the cv2.rectangle function
# Coordinates for Rectangle, top left corner
x = width//2
y = height//2
# Width and height

w = width//4
h = height//4
# botton right x+w, y+h

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

            # Draw a rectangle on stream
    cv2.rectangle(frame, (x, y), (x+w, y+h), color=(0,0,255),thickness= 4)

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

```
cv2.destroyAllWindows()
```

```
# ## Interactive Drawing on Video
# Create a function based on a CV2 Event (Left button click)
# mouse callback function
def draw_rectangle(event,x,y,flags,param):
    global pt1,pt2,topLeft_clicked,botRight_clicked
            # get mouse click
    if event == cv2.EVENT_LBUTTONDOWN:
        if topLeft_clicked == True and botRight_clicked == True:
            topLeft_clicked = False
            botRight_clicked = False
```

```
        pt1 = (0,0)
        pt2 = (0,0)

    if topLeft_clicked == False:
        pt1 = (x,y)
        topLeft_clicked = True

    elif botRight_clicked == False:
        pt2 = (x,y)
        botRight_clicked = True

# Global vars
pt1 = (0,0)
pt2 = (0,0)
topLeft_clicked = False
botRight_clicked = False

# Connect to the callback
cap = cv2.VideoCapture(0)
# Create a named window for connections
cv2.namedWindow('Test')

# Bind draw_rectangle function to mouse cliks
cv2.setMouseCallback('Test', draw_rectangle)
```

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

    if topLeft_clicked:
        cv2.circle(frame, center=pt1, radius=5, color=(0,0,255), thickness=-1)

    #drawing rectangle
    if topLeft_clicked and botRight_clicked:
        cv2.rectangle(frame, pt1, pt2, (0, 0, 255), 2)

    # Display the resulting frame
    cv2.imshow('Test', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

```
# *** Object Detection *** #

# # Template Matching, v need to hv exact match

# ### Full Image

full = cv2.imread('sammy.jpg')

full = cv2.cvtColor(full, cv2.COLOR_BGR2RGB)

plt.imshow(full)


# ### Template Image

# A subset of the image. its actually the exact image.

face= cv2.imread('sammy_face.jpg')
```

```
face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)

plt.imshow(face)

# this is an exact match v need to hv exact img with same shape as in orgl img

# it is a simple matching technq


print(full.shape)

print(face.shape)









# # Template Matching Methods

# **eval()** function

sum([1,2,3])


mystring = 'sum'


eval(mystring)


myfunc = eval(mystring)


myfunc([1,2,3])


height, width,channels = face.shape
```

```
print(width)

print(height)




# The Full Image to Search

# large img

full = cv2.imread('sammy.jpg')

full = cv2.cvtColor(full, cv2.COLOR_BGR2RGB)


# The Template to Match

# small img

face= cv2.imread('sammy_face.jpg')

face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)



my_method = eval('cv2.TM_CCOEFF')

res = cv2.matchTemplate(full,face,my_method)

plt.imshow(res)

# gvs u a heatmap of where it thinks the highest correlation of the match occured



# All the 6 methods for comparison in a list
```

```
# we are using strings, we'll use the eval() function to convert to function
methods = ['cv2.TM_CCOEFF', 'cv2.TM_CCOEFF_NORMED',
'cv2.TM_CCORR','cv2.TM_CCORR_NORMED', 'cv2.TM_SQDIFF', 'cv2.TM_SQDIFF_NORMED']
for m in methods:
    # Create a copy of the image
    full_copy = full.copy()
    # Get the actual function instead of the string
    method = eval(m)
    # Apply template Matching with the method
    res = cv2.matchTemplate(full_copy,face,method)

    # Grab the Max and Min values, plus their locations of the match
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)

    # Then draw Rectangle
    # If the method is TM_SQDIFF or TM_SQDIFF_NORMED, take minimum vals of corr is used
    # Notice the coloring on the last 2 left hand side images.
    if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
        top_left = min_loc
    else:
        top_left = max_loc

    height, width,channels = face.shape

    # Assign the Bottom Right of the rectangle
    bottom_right = (top_left[0] + width, top_left[1] + height)
```

```
# Draw the Red Rectangle

cv2.rectangle(full_copy,top_left, bottom_right, 255, 10)


# Plot the Images

plt.subplot(121)

plt.imshow(res)

plt.title('Result of Template Matching')


plt.subplot(122)

plt.imshow(full_copy)

plt.title('Detected Point')

plt.suptitle(m)



plt.show()

print('\n')

print('\n')
```

```
# # Corner Detection

# ### The Image Data


flat_chess = cv2.imread('flat_chessboard.png')

flat_chess = cv2.cvtColor(flat_chess,cv2.COLOR_BGR2RGB)

plt.imshow(flat_chess)
```

```
gray_flat_chess = cv2.cvtColor(flat_chess,cv2.COLOR_BGR2GRAY)

plt.imshow(gray_flat_chess,cmap='gray')



real_chess = cv2.imread('real_chessboard.jpg')

real_chess = cv2.cvtColor(real_chess,cv2.COLOR_BGR2RGB)

plt.imshow(real_chess)



gray_real_chess = cv2.cvtColor(real_chess,cv2.COLOR_BGR2GRAY)

plt.imshow(gray_real_chess,cmap='gray')



# # Harris Corner Detection

# **cornerHarris Function**

# *  src Input single-channel 8-bit or floating-point image.

# *  dst Image to store the Harris detector responses. It has the type CV_32FC1 and the same size as src .

# *  blockSize Neighborhood size (see the details on #cornerEigenValsAndVecs ).

# *  ksize Aperture parameter for the Sobel operator.

# *  k Harris detector free parameter. See the formula in DocString

# *  borderType Pixel extrapolation method. See #BorderTypes.



# all integers

gray_flat_chess
```

```
# Convert Gray Scale Image to Float Values
gray = np.float32(gray_flat_chess)
gray


# Corner Harris Detection
dst = cv2.cornerHarris(src=gray,blockSize=2,ksize=3,k=0.04)
# blksize is the neighborhood size, it detects edges
# ksize is the par for sobel oper, kernel size
# cornerHarris uses these internally
# k is the harris detector free par
dst




# result is dilated for marking the corners, not important to actual corner detection
# this is so we can plot out the points on the image shown
dst = cv2.dilate(dst,None)
# dilate is a morphological oper
dst



# Threshold for an optimal value, it may vary depending on the image.
```

```
flat_chess
```

```
flat_chess[dst>0.01*dst.max()]=[255,0,0]
# whenever the corner harris is gt 1% of max val
# reassign 1% of max val to color red
# this is for visualziation
flat_chess
```

```
plt.imshow(flat_chess)
# ntc all corners r not detected
```

```
# try on real chess board
# Convert Gray Scale Image to Float Values
gray = np.float32(gray_real_chess)
# Corner Harris Detection
dst = cv2.cornerHarris(src=gray,blockSize=2,ksize=3,k=0.04)
# result is dilated for marking the corners, not important to actual corner detection
# this is toplot out the points on the image shown
dst = cv2.dilate(dst,None)
# Threshold for an optimal value, it may vary depending on the image.
real_chess[dst>0.01*dst.max()]=[255,0,0]
```

plt.imshow(real_chess)

 # more corners on blk pieces r detected

 # all the major corners r detected

 # the outside corners r not detected still

 # ## Shi-Tomasi Corner Detector & Good Features to Track Paper

 # goodFeatureToTrack Function Parameters

 # * image Input 8-bit or floating-point 32-bit, single-channel image.

 # * corners Output vector of detected corners.

 # * maxCorners Maximum number of corners to return. If there are more corners than are found,the strongest of them is returned. `maxCorners <= 0` implies that no limit on the maximum is set and all detected corners are returned.

 # * qualityLevel Parameter characterizing the minimal accepted quality of image corners. The parameter value is multiplied by the best corner quality measure, which is the minimal eigenvalue (see #cornerMinEigenVal) or the Harris function response (see #cornerHarris). The corners with the quality measure less than the product are rejected. For example, if the best corner has the quality measure = 1500, and the qualityLevel=0.01 , then all the corners with the quality measure less than 15 are rejected.

 # Need to reset the images since we drew on them

```
flat_chess = cv2.imread('flat_chessboard.png')
plt.imshow(flat_chess)
plt.show()

flat_chess = cv2.cvtColor(flat_chess,cv2.COLOR_BGR2RGB)
plt.imshow(flat_chess)
plt.show()

gray_flat_chess = cv2.cvtColor(flat_chess,cv2.COLOR_BGR2GRAY)
plt.imshow(gray_flat_chess)
plt.show()



corners = cv2.goodFeaturesToTrack(gray_flat_chess,5,0.01,10)
# 5 is max num of corners to return
# 0.01 is the quality lvel par
# the par val is mult by the best quality measure
# whi is the min eigen val
# 10
corners = np.int0(corners)
for i in corners:
    x,y = i.ravel() # ravel is flattening
    cv2.circle(flat_chess,(x,y),3,255,-1)
plt.imshow(flat_chess)
```

```
corners = cv2.goodFeaturesToTrack(gray_flat_chess,64,0.01,10)
corners = np.int0(corners)
for i in corners:
    x,y = i.ravel()
    cv2.circle(flat_chess,(x,y),3,255,-1)
plt.imshow(flat_chess)
# these r the best corners



# try on real
real_chess = cv2.imread('real_chessboard.jpg')
real_chess = cv2.cvtColor(real_chess,cv2.COLOR_BGR2RGB)
gray_real_chess = cv2.cvtColor(real_chess,cv2.COLOR_BGR2GRAY)


corners = cv2.goodFeaturesToTrack(gray_real_chess,60,0.01,10)
corners = np.int0(corners)
for i in corners:
    x,y = i.ravel()
    cv2.circle(real_chess,(x,y),3,255,-1)
plt.imshow(real_chess)
```

```
# sm of the main chess board corners r not detected



corners = cv2.goodFeaturesToTrack(gray_real_chess,100,0.01,10)

corners = np.int0(corners)

for i in corners:

    x,y = i.ravel()

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

plt.imshow(real_chess)
```

```
# # Canny Edge Detection

img = cv2.imread('sammy_face.jpg')

plt.imshow(img)

plt.show()


edges = cv2.Canny(image=img, threshold1=127, threshold2=127)

plt.imshow(edges)

plt.show()



edges = cv2.Canny(image=img, threshold1=0, threshold2=255)

plt.imshow(edges)

plt.show()
```

```
# ## Choosing Thresholds, formulat for choosing thresholds

# Calculate the median pixel value

med_val = np.median(img)

# Lower bound is either 0 or 70% of the median value, whicever is higher

lower = int(max(0, 0.7* med_val))

# Upper bound is either 255 or 30% above the median value, whichever is lower

upper = int(min(255,1.3 * med_val))

edges = cv2.Canny(image=img, threshold1=lower , threshold2=upper)

plt.imshow(edges)
```

```
# Sometimes it helps to blur the images first, so we don't pick up minor edges.

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

edges = cv2.Canny(image=blurred_img, threshold1=lower , threshold2=upper)

plt.imshow(edges)
```

```
print(lower)

print(upper)
```

```
# play with threshold values

edges = cv2.Canny(image=blurred_img, threshold1=lower , threshold2=upper+50)

plt.imshow(edges)

# this is the best
```

```
# # Grid Detection


flat_chess = cv2.imread('flat_chessboard.png')

plt.imshow(flat_chess,cmap='gray')




# this works only for chessboard like grids

found, corners = cv2.findChessboardCorners(flat_chess,(7,7))

# 7,7 as it is 8 x 8 it ds not find last one.

if found:

   print('OpenCV was able to find the corners')

else:

   print("OpenCV did not find corners. Double check your patternSize.")

corners

corners.shape



flat_chess_copy = flat_chess.copy()

cv2.drawChessboardCorners(flat_chess_copy, (7, 7), corners, found)

plt.imshow(flat_chess_copy)
```

```
# it marks the corners, and then marks the rows in raibow colors
```

```
# # Circle Based Grids
# this is another grid like pattern
dots = cv2.imread('dot_grid.png')
plt.imshow(dots)
```

```
found, corners = cv2.findCirclesGrid(dots, (10,10), cv2.CALIB_CB_SYMMETRIC_GRID)
found
```

```
dbg_image_circles = dots.copy()
cv2.drawChessboardCorners(dbg_image_circles, (10, 10), corners, found)
plt.imshow(dbg_image_circles)
```

```
# # Contour Detection

# ## External vs Internal Contours

img = cv2.imread('internal_external.png',0)

img.shape

plt.imshow(img,cmap='gray')



# **findContours**
```

 # function will return back contours in an image, and based on the RETR method called, you can get back external, internal, or both:

 # * cv2.RETR_EXTERNAL:Only extracts external contours

 # * cv2.RETR_CCOMP: Extracts both internal and external contours organized in a two-level hierarchy

 # * cv2.RETR_TREE: Extracts both internal and external contours organized in a tree graph

 # * cv2.RETR_LIST: Extracts all contours without any internal/external relationship

#image, contours, hierarchy = cv2.findContours(img, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)

contours, hierarchy = cv2.findContours(img, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)

type(contours)

print(len(contours))

 # 22, if v count the shapes thy will b 22

print(type(hierarchy))

print(hierarchy.shape)

hierarchy

 # -1s r ext contrs

 # 0s r eyes smiley face

 # 4s r pepprni slices

 # Draw External Contours

```
# Set up empty array
#external_contours = np.zeros(image.shape)
external_contours = np.zeros(img.shape)
# external contours r triangle, circle face or white circle of shape cutout
external_contours.shape # 652,1080 is same as orignal img
# this is pure black image

img.shape



plt.imshow(img)


list(range(len(contours)))


# For every entry in contours
for i in range(len(contours)):
   # last column in the array is -1 if an external contour (no contours inside of it)
   if hierarchy[0][i][3] == -1:
                    # v r checking if the last cell val == -1
                    # then it is ext otherwise it is int contour
      # We can now draw the external contours from the list of contours
      cv2.drawContours(external_contours, contours, i, 255, -1)
                    # 255 is for white


plt.imshow(external_contours,cmap='gray')
```

```
# ext contours whi r touching the background so they r ext contours
# int contours r touching the fg which r the eyes and other shapes inside



# Create empty array to hold internal contours
# internal contours r the eyes & smiley & slices of pepperoni
image_internal = np.zeros(img.shape)

# Iterate through list of contour arrays
for i in range(len(contours)):
    # If third column value is NOT equal to -1 than its internal
    if hierarchy[0][i][3] != -1:

        # Draw the Contour
        cv2.drawContours(image_internal, contours, i, 255, -1)

plt.imshow(image_internal,cmap='gray')




# slices
image_internal = np.zeros(img.shape)
for i in range(len(contours)):
```

```
    if hierarchy[0][i][3] == 4:
        cv2.drawContours(image_internal, contours, i, 255, -1)
plt.imshow(image_internal,cmap='gray')



image_internal = np.zeros(img.shape)
for i in range(len(contours)):
    if hierarchy[0][i][3] == 0:
        cv2.drawContours(image_internal, contours, i, 255, -1)
plt.imshow(image_internal,cmap='gray')
```

```
# # Feature Matching, Real world CV starts here
def display(img,cmap='gray'):
    fig = plt.figure(figsize=(12,10))
    ax = fig.add_subplot(111)
    ax.imshow(img,cmap='gray')

reeses = cv2.imread('reeses_puffs.png')
display(reeses)

reeses = cv2.imread('reeses_puffs.png',0)
display(reeses)
# this is not an exact photo but a print out of it
# v dont need an exact match photo or template here which is an adv in real world


# this is a real pic of a cereals isle with diff cereals
# none of them r exactly front facing like the abv img
cereals = cv2.imread('many_cereals.jpg')
display(cereals)

# this is the target img
```

```
cereals = cv2.imread('many_cereals.jpg',0)

display(cereals)

# at the btm left corner is the reesespuffs but it is not a family size

# like the orgnl img

# there r sm of the other boxes whi hv the family size on em

# this may affect our feature matching as v hv family size on the top of our orgnl img









# # this is a Brute Force Detection with ORB Descriptors

# Initiate ORB detector

orb = cv2.ORB_create()

# this creates an obj of orb detector

# and detects features


# find the keypoints and descriptors with ORB

# this is for computing features

kp1, des1 = orb.detectAndCompute(reeses,None)

# reeses is our orgnl img that v r searching for and None for masking

# v r not masking anything here

# kp is the keyopint & des1 is the descriptor


# this is for cereals

kp2, des2 = orb.detectAndCompute(cereals,None)
```

```
# create BFMatcher object
# stands for brute force matcher
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
# the abv 2 r default pars

# Match descriptors.
matches = bf.match(des1,des2)
# these r the matches

sngl_mtch = matches[0]
sngl_mtch.distance
# gvs the dstnc, the less dstnc the btr the match and vice versa

# Sort them in the order of their distance.
matches = sorted(matches, key = lambda x:x.distance)
matches
# a grp of match objs
# the less the distance the more closer and viceversa

len(matches)
# there r 265 matches

reeses_matches = cv2.drawMatches(reeses,kp1,cereals,kp2,matches[:25],None,flags=2)
```

```
display(reeses_matches)

# Draw first 25 matches.

# none of them r v good, this techniq ds not work unlike the target img

# looks like the orgnl img
```

```
# ******** From below in testing issues with sift creator *** in dev works in prodn


# # Brute-Force Matching with SIFT Descriptors and Ratio Test

# Create SIFT Object

# scale in variant feature transform, it helps when img sizes r in dfrnt scale

# our orgnl img is much larger than the cereals in the target img

sift = cv2.xfeatures2d.SIFT_create()


# find the keypoints and descriptors with SIFT

kp1, des1 = sift.detectAndCompute(reeses,None)

kp2, des2 = sift.detectAndCompute(cereals,None)
```

 # run upto here separately

 # if v get err for the abv

 # error: OpenCV(3.4.3) C:\projects\opencv-python\opencv_contrib\modules\xfeatures2d\src

 # \sift.cpp:1207: error: (-213:The function/feature is not implemented)

 # This algorithm is patented and is excluded in this configuration;

 # Set OPENCV_ENABLE_NONFREE CMake option and rebuild the library in function

 # 'cv::xfeatures2d::SIFT::create

 # run the below one in cmd prompt as admin

 # C:\PrgramData\Anaconda3

first try this

#!conda install -c menpo opencv

if it ds not work try this

open Ana prompt as Admin

 # pip3 uninstall opencv-contrib-python

 # pip install opencv-contrib-python

```
# BFMatcher with default params
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1,des2, k=2)
matches
# first match is btr than a 2nd match and so on
# first col is the first best mtch
# sec col is the 2nd best mtch
# if first mtch is close in dstnc to the 2nd mtch then
# overal it is a good feature to mtch on
# also if v hv a strong mtch in the first col
# nd 2nd bst mtch is far away in dstnc
# then this descriptor whi is in the first row

# Apply ratio test
# is used for checking if the 2 mtches r close in dist or not
good = []
for match1,match2 in matches:
   if match1.distance < 0.75*match2.distance:
                    # if m1 dist is lt 75% of mtch2 dist
                    # then the descriptor is a gd mtch
      good.append([match1])
                    # mtch1 is a ratio of 75% of mtch2 v call it a ratio test
```

```
                    # less dist == btr mtch

print(good)


print(len(good))

# 78 r best mtches


print(matches)









# cv2.drawMatchesKnn expects list of lists as matches.

sift_matches = cv2.drawMatchesKnn(reeses,kp1,cereals,kp2,good,None,flags=2)

display(sift_matches)
```

```
# # FLANN based Matcher

# Initiate SIFT detector

sift = cv2.xfeatures2d.SIFT_create()


# find the keypoints and descriptors with SIFT

kp1, des1 = sift.detectAndCompute(reeses,None)

kp2, des2 = sift.detectAndCompute(cereals,None)


# FLANN parameters

FLANN_INDEX_KDTREE = 0

index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)

search_params = dict(checks=50)


flann = cv2.FlannBasedMatcher(index_params,search_params)


matches = flann.knnMatch(des1,des2,k=2)


good = []


# ratio test

for i,(match1,match2) in enumerate(matches):

   if match1.distance < 0.7*match2.distance:
```

```
        good.append([match1])



flann_matches = cv2.drawMatchesKnn(reeses,kp1,cereals,kp2,good,None,flags=0)

display(flann_matches)









# Initiate SIFT detector

sift = cv2.xfeatures2d.SIFT_create()


# find the keypoints and descriptors with SIFT
```

```
kp1, des1 = sift.detectAndCompute(reeses,None)

kp2, des2 = sift.detectAndCompute(cereals,None)


# FLANN parameters

FLANN_INDEX_KDTREE = 0

index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)

search_params = dict(checks=50)


flann = cv2.FlannBasedMatcher(index_params,search_params)


matches = flann.knnMatch(des1,des2,k=2)


# Need to draw only good matches, so create a mask

matchesMask = [[0,0] for i in range(len(matches))]


# ratio test

for i,(match1,match2) in enumerate(matches):

   if match1.distance < 0.7*match2.distance:

      matchesMask[i]=[1,0]


draw_params = dict(matchColor = (0,255,0),

                   singlePointColor = (255,0,0),

                   matchesMask = matchesMask,

                   flags = 0)
```

```
flann_matches = cv2.drawMatchesKnn(reeses,kp1,cereals,kp2,matches,None,**draw_params)


display(flann_matches)


# ******** Upto abv in testing issues with sift creator *** in dev works in prodn
```

```
# *** Watershed Algorithm *** #

import numpy as np
import cv2
import matplotlib.pyplot as plt
get_ipython().magic('matplotlib inline')



def display(img,cmap=None):
    fig = plt.figure(figsize=(10,8))
    ax = fig.add_subplot(111)
    ax.imshow(img,cmap=cmap)
```

 # ## Task: Draw Contours Around the Coins

 # ## Common Coin Example

 # ## Naive Approach

 # simply use a threshold and then use findContours.

sep_coins = cv2.imread('pennies.jpg')

display(sep_coins)

 # for humans it is ez to tell thy r 6 sep coins but for a comp it may think

 # f it as 1 giant img

 # our task is to segment these into 7 dfrnt segments

 # 6 diff coins and 1 bg

 # ### Apply Median Blurring

 # too much detail in this image, including light, the face edges on the coins,

 # and too much detail in the background. use Median Blur Filtering to blur the image a bit,

 # which will be useful later on when we threshold.

sep_blur = cv2.medianBlur(sep_coins,25)

display(sep_blur)

 # 25 is the kernel size

 # the img is 4000 x 3000 pxls

 # convert this to grayscale

```
gray_sep_coins = cv2.cvtColor(sep_blur,cv2.COLOR_BGR2GRAY)

display(gray_sep_coins,cmap='gray')



# ## Binary Threshold for seprting fg & bg

ret, sep_thresh = cv2.threshold(gray_sep_coins,160,255,cv2.THRESH_BINARY_INV)

display(sep_thresh,cmap='gray')

# the inversion is for inverting black & white as w & b


ret, sep_thresh = cv2.threshold(gray_sep_coins,127,255,cv2.THRESH_BINARY_INV)

display(sep_thresh,cmap='gray')

# ntc as v lower it there is distortion due to the faces on the coins


ret, sep_thresh = cv2.threshold(gray_sep_coins,160,255,cv2.THRESH_BINARY_INV)

display(sep_thresh,cmap='gray')



# ## FindContours

#image, contours, hierarchy = cv2.findContours(sep_thresh.copy(), cv2.RETR_CCOMP,
cv2.CHAIN_APPROX_SIMPLE)

contours, hierarchy = cv2.findContours(sep_thresh.copy(), cv2.RETR_CCOMP,
cv2.CHAIN_APPROX_SIMPLE)

# For every entry in contours

for i in range(len(contours)):

   # last column in the array is -1 if an external contour (no contours inside of it)
```

```
    if hierarchy[0][i][3] == -1:  # this is an ext contour
        # We can draw the external contours from the list of contours
        cv2.drawContours(sep_coins, contours, i, (255, 0, 0), 10)

display(sep_coins)
# nte this is 1 giant contour and the gaps r not compltely clsd
# this is a problem
# so bring in the WS alg
```

```
# # Watershed Algorithm
# the watershed algorithm apporach to draw contours around the pennies.
# ## Using the WaterShed Algorithm
# #### Step 1: Read Image
img = cv2.imread('pennies.jpg')
```

```
# #### Step 2: Apply Blur
img = cv2.medianBlur(img,35) # v r using huge kernel size
display(img)



# #### Step 3: Convert to Grayscale
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray,127,255,cv2.THRESH_BINARY_INV)
display(thresh,cmap='gray')
# nt the noise/distortions whi v dont need at this stg



# #### Step 4: use OTSUs methd of thresholding, Apply Threshold (Inverse Binary with OTSU )
ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
display(thresh,cmap='gray')
# ntc they r still connected and v hv not yet achieved the separation



# ### Optional Step 5: Noise Removal
```

```
# noise removal

kernel = np.ones((3,3),np.uint8)

kernel # create a kernel


opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 2)

display(opening,cmap='gray')

# no effect here but for discipline

# this works if v use thresh_bin_inv instead of otsu v wld see the dfrncs



# the fund prblm v hv is tht the coins r still cnnctd to ea othr

# and it is treated as 1 big img and not sep coins inspite of all abv steps

# wt v need to do for WS alg is set seeds that v r sure in the fg

# v wnt 6 seeds one for ea of the cntr of the coins

# so how to grab the things in the bg & fg

# using dist transform

# in a binry img v hv 0s and 1s or 0s and 255s

# wht DT ds is as the pixls is away frm 0s the val gets higher

# means they look higher

# #### Step 6: Grab Background that you are sure of

# sure background area

sure_bg = cv2.dilate(opening,kernel,iterations=3)

display(sure_bg,cmap='gray')
```

```
# #### Step 7: Find Sure Foreground

# Finding sure foreground area

dist_transform = cv2.distanceTransform(opening,cv2.DIST_L2,5)

display(dist_transform,cmap='gray')

# v hv 6 clear pts in the fg




# nxt apply thresholding to this to describe the dots or centers

# then apply to WS alg whi will then und the 6 segments it will look into

ret, sure_fg = cv2.threshold(dist_transform,0.7*dist_transform.max(),255,0)

# 70% of max val in dist transform

display(sure_fg,cmap='gray')

# these 6 pts r absolutely in the fg coz v did a thresholding and then a DT


display(dist_transform,cmap='gray')




# #### Step 8: Find Unknown Region

# Finding unknown region is the reg whi is anything whi is in DT but not is sure_fg

# thts the U R nd thts wht v want WS alg to find
```

```
sure_fg = np.uint8(sure_fg)

sure_fg


unknown = cv2.subtract(sure_bg,sure_fg)

# unk reg is remvng bg frm fg

display(unknown,cmap='gray')

# the whte reg is the Unknown Reg means v dont know if it blngs to fg or bg




# nxt label mark the 6 pts as seeds and hv the WS alg use them to find the segments

# #### Step 9: Label Markers of Sure Foreground

# Marker labelling

# feed in 6 pts as the markers

ret, markers = cv2.connectedComponents(sure_fg)

markers

# all r 0s



# Add one to all labels so that sure background is not 0, but 1

markers = markers+1

# this will mark the region of unknown with zero

markers[unknown==255] = 0
```

```
display(markers,cmap='gray')

# this is manually marking the labels

# next automatic way of marking the labels

# the black region in the mid is the unknown region

# means it is not sure it is fg or bg

# these markers will act as seeds to the WS alg




# #### next Step 10: Apply Watershed Algorithm to find Markers

markers = cv2.watershed(img,markers)

display(markers)


# #### Step 11: Find Contours on Markers

#image, contours, hierarchy = cv2.findContours(markers.copy(), cv2.RETR_CCOMP,
cv2.CHAIN_APPROX_SIMPLE)

contours, hierarchy = cv2.findContours(markers.copy(), cv2.RETR_CCOMP,
cv2.CHAIN_APPROX_SIMPLE)

# For every entry in contours

for i in range(len(contours)):

   # last column in the array is -1 if an external contour (no contours inside of it)

   if hierarchy[0][i][3] == -1:

      # We can now draw the external contours from the list of contours

      cv2.drawContours(sep_coins, contours, i, (255, 0, 0), 10)
```

```
display(sep_coins)

# this will segment the coins seprtly
```

```
# # Custom Seeds with the WaterShed Algorithm

#

# Previously we did a lot of work for OpenCV to set Markers to provide seeds to the

# Watershed Algorithm. but v can auto do this

# ### Read in the Image and Make a Copy
```

```
road = cv2.imread('road_image.jpg')

road_copy = np.copy(road)

plt.imshow(road)



# #### Create an empty space for the results to be drawn

print(road.shape)


print(road.shape[:2])


marker_image = np.zeros(road.shape[:2],dtype=np.int32)

# this is for markers, this will tk only x & y

plt.imshow(marker_image)


segments = np.zeros(road.shape,dtype=np.uint8)

# this is for segments

segments.shape



# ### Create colors for Markers

from matplotlib import cm
```

```
# Returns (R,G,B,Alpha) we only need RGB values

cm.tab10(0) # these r like templates of colors

# they r scaled btw 0 & 1


cm.tab10(1)


np.array(cm.tab10(0))

# cnvrt to an np arr


np.array(cm.tab10(0))[:3]

# gvs us r g b


np.array(cm.tab10(0))[:3]*255

# mult ply by 255


x = np.array(cm.tab10(0))[:3]*255


tuple(x.astype(int))

# cnvrt to a tuple


# a function for all those steps

def create_rgb(i):

    x = np.array(cm.tab10(i))[:3]*255

    return tuple(x)
```

```
colors = []
# One color for each single digit
for i in range(10):
    colors.append(create_rgb(i))
colors
# ea tuple is a uniq mapping of r g b
```

```
# ### Setting Up Callback Function
# Numbers 0-9
n_markers = 10
# Default settings
current_marker = 1
marks_updated = False
def mouse_callback(event, x, y, flags, param):
    global marks_updated
    if event == cv2.EVENT_LBUTTONDOWN:
        # TRACKING FOR MARKERS
                    # Markers pasd to the WS algo
        cv2.circle(marker_image, (x, y), 10, (current_marker), -1)
```

```
                # DISPLAY ON USER IMAGE
                # User sees on the img
        cv2.circle(road_copy, (x, y), 10, colors[current_marker], -1)
        marks_updated = True

cv2.namedWindow('Road Image')
cv2.setMouseCallback('Road Image', mouse_callback)

while True:
    # SHow the 2 windows
    cv2.imshow('WaterShed Segments', segments)
            # this is the black one with segments hidden
    cv2.imshow('Road Image', road_copy)
            # this gets updated when clkd
    # Close everything if Esc is pressed
    k = cv2.waitKey(1)
    if k == 27:
        break

            # Clear all colors and start over if 'c' is pressed
    elif k == ord('c'):
        road_copy = road.copy()
        marker_image = np.zeros(road.shape[0:2], dtype=np.int32)
        segments = np.zeros(road.shape,dtype=np.uint8)
```

```
            # If a number 0-9 is chosen index the color
    elif k > 0 and chr(k).isdigit():
        # chr converts to printable digit
        current_marker  = int(chr(k))
        # CODE TO CHECK INCASE USER IS CARELESS
        '''n = int(chr(k))
        if 1 <= n <= n_markers:
            current_marker = n'''


            # If we clicked somewhere, call the watershed algorithm on our chosen markers
    if marks_updated:
        marker_image_copy = marker_image.copy()
        cv2.watershed(road, marker_image_copy)


        segments = np.zeros(road.shape,dtype=np.uint8)


        for color_ind in range(n_markers):
            segments[marker_image_copy == (color_ind)] = colors[color_ind]


        #marks_updated = False
cv2.destroyAllWindows()
```

```
# *** Part 3 ***#

# # Face Detection with Haar Cascades

#

# **This is face *detection* NOT face *recognition*.

# We are only detecting if a face is in an image,

# not who the face actually is.

# That requires deep learning **


import numpy as np

import cv2

import matplotlib.pyplot as plt
```

```
get_ipython().magic('matplotlib inline')



# ## Images

nadia = cv2.imread('Nadia_Murad.jpg',0)

denis = cv2.imread('Denis_Mukwege.jpg',0)

solvay = cv2.imread('solvay_conference.jpg',0)


plt.imshow(nadia,cmap='gray')

plt.show()


plt.imshow(denis,cmap='gray')

plt.show()


plt.imshow(solvay,cmap='gray')

plt.show()



# ## Cascade Files

# OpenCV comes with these pre-trained cascade files,

# ve located the .xml files for you in our own DATA folder.

# ## Face Detection


face_cascade = cv2.CascadeClassifier('haarcascades\\haarcascade_frontalface_default.xml')

# this is a list of 6000 classifiers or features tht r going to b
```

```
# pssd thru the img to see if it fits and indicate if a face is there


def detect_face(img):
    face_img = img.copy()
    face_rects = face_cascade.detectMultiScale(face_img)
            # face_rects r a grp of x & y positions and wid & hts of rects
    for (x,y,w,h) in face_rects:
        cv2.rectangle(face_img, (x,y), (x+w,y+h), (255,255,255), 10)
                    #255 s r for white img
                    #10 is thickness
    return face_img


result = detect_face(denis)


plt.imshow(result,cmap='gray')
plt.show()


result = detect_face(nadia)
plt.imshow(result,cmap='gray')
plt.show()


# Gets errors!
result = detect_face(solvay)
plt.imshow(result,cmap='gray')
plt.show()
```

```
# mult faces, sm of em r not looking at camera
# ntc the dbl face




def adj_detect_face(img):
    face_img = img.copy()
    face_rects = face_cascade.detectMultiScale(face_img,scaleFactor=1.2, minNeighbors=5)
            # scaleFac is specifying how much the img size is reduced
            # how many neighbors ea rect must hv
    for (x,y,w,h) in face_rects:
        cv2.rectangle(face_img, (x,y), (x+w,y+h), (255,255,255), 10)
    return face_img

# Doesn't detect the side face.
result = adj_detect_face(solvay)
plt.imshow(result,cmap='gray')
plt.show()
# ntc the side face is not detected
# this is tradeoff btwn abv one & this one
# play with scalefac & min neighbors
```

```
# ## Eye Cascade File
eye_cascade = cv2.CascadeClassifier('haarcascades\\haarcascade_eye.xml')
def detect_eyes(img):
    face_img = img.copy()
            #eyes = eye_cascade.detectMultiScale(face_img)
    eyes = eye_cascade.detectMultiScale(face_img,scaleFactor=1.2, minNeighbors=5)
    for (x,y,w,h) in eyes:
        cv2.rectangle(face_img, (x,y), (x+w,y+h), (255,255,255), 10)
    return face_img

result = detect_eyes(nadia)
plt.imshow(result,cmap='gray')
# play with scalefac & min neighbors



eyes = eye_cascade.detectMultiScale(denis)
# White around the pupils is not distinct enough to detect Denis'
# eyes means white in the eyes is not white but dark
# they r same color as skin, this may bcoz of photo editing done
# by the photographer, but for nadia white is white
# and that is one of the main features or cascade is looking for
```

```
# so the solution is find an un edited photo of denis

result = detect_eyes(denis)

plt.imshow(result,cmap='gray')
```

```
# ## Conjunction with Video

cap = cv2.VideoCapture(0)

while True:
```

```
    ret, frame = cap.read(0)
    frame = detect_face(frame)
    cv2.imshow('Video Face Detection', frame)
    c = cv2.waitKey(1)
    if c == 27:
        break

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

```
# Detect Face & Eyes & Smile
import cv2

# Loading the cascades
face_cascade = cv2.CascadeClassifier('haarcascades\\haarcascade_frontalface_default.xml')
```

```
eye_cascade = cv2.CascadeClassifier('haarcascades\\haarcascade_eye.xml')
smile_cascade = cv2.CascadeClassifier('haarcascades\\haarcascade_smile2.xml')

# Defining a function that will do the detections
def detect(gray, frame):
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = frame[y:y+h, x:x+w]
        eyes = eye_cascade.detectMultiScale(roi_gray, 1.1, 22)
        for (ex, ey, ew, eh) in eyes:
            cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh), (0, 255, 0), 2)
        smiles = smile_cascade.detectMultiScale(roi_gray, 1.7, 22)
        for (sx, sy, sw, sh) in smiles:
            cv2.rectangle(roi_color, (sx, sy), (sx+sw, sy+sh), (0, 0, 255), 2)
    return frame

# Doing some Face Recognition with the webcam
video_capture = cv2.VideoCapture(0)
while True:
    _, frame = video_capture.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    canvas = detect(gray, frame)
    cv2.imshow('Video', canvas)
```

```
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
video_capture.release()
cv2.destroyAllWindows()
```

```
# Cartoon & Edge effect
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np
import cv2

def Cartoon(image_color):
    output_image = cv2.stylization(image_color, sigma_s=100, sigma_r=0.3)
    return output_image



def LiveCamEdgeDetection_canny(image_color):
    threshold_1 = 30
    threshold_2 = 80
    image_gray = cv2.cvtColor(image_color, cv2.COLOR_BGR2GRAY)
    canny = cv2.Canny(image_gray, threshold_1, threshold_2)
    return canny



cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read() # Cap.read() returns a ret bool to indicate success.
    cv2.imshow('Live Edge Detection', Cartoon(frame))
```

```
    #cv2.imshow('Live Edge Detection', LiveCamEdgeDetection_canny(frame))
    cv2.imshow('Webcam Video', frame)
    if cv2.waitKey(1) == 13: #13 Enter Key
        break

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

```
# ## Russian License Plate Blurring
#
# object detection our goal will be to use Haar Cascades to blur license
# plates detected in an image!
# Russians are famous for having some of the most entertaining
# DashCam footage on the internet Google Search "Russian DashCam".
# a lot of the footage contains license plates,
# help and create a license plat blurring tool?
# OpenCV comes with a Russian license plate detector .xml file
# we can use like we used the face detection files
# ( it does not come with license detectors for other countries!)

# Import the usual libraries you think you'll need.**
import cv2
import numpy as np
import matplotlib.pyplot as plt
get_ipython().magic('matplotlib inline')

# Read in the car_plate.jpg file from the DATA folder.**
img = cv2.imread('car_plate.jpg')
```

```
# Create a function that displays the image in a larger scale and correct coloring for matplotlib.**
def display(img):
    fig = plt.figure(figsize=(10,8))
    ax = fig.add_subplot(111)
    new_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    ax.imshow(new_img)
display(img)



# Load the haarcascade_russian_plate_number.xml file.**
plate_cascade = cv2.CascadeClassifier('haarcascades\\haarcascade_russian_plate_number.xml')


# Create a function that takes in an image and draws a
# rectangle around what it detects to be a license plate.
# just draw a rectangle around it for now
# later adjust this function to blur.
# play with the scaleFactor and minNeighbor numbers to get good results.**
def detect_plate(img):
    plate_img = img.copy()
    plate_rects = plate_cascade.detectMultiScale(plate_img,scaleFactor=1.3, minNeighbors=3)
    for (x,y,w,h) in plate_rects:
        cv2.rectangle(plate_img, (x,y), (x+w,y+h), (0,0,255), 4)
    return plate_img


result = detect_plate(img)
```

```
display(result)



# **FINAL Edit the function so that is effectively blurs the detected plate, instead of just drawing a rectangle around it. the steps:**

# 1. The hardest part is converting the (x,y,w,h) information into the

# dimension values v need to grab an ROI

# (just need to convert the information about the top left corner of the

# rectangle and width and height, into indexing position values.

# 2. Once you've grabbed the ROI using the (x,y,w,h) values returned,

# v blur that ROI. use cv2.medianBlur for this.

# 3. Now that v have a blurred version of the ROI (the license plate)

# v will want to paste this blurred image back on to the original image

# at the same original location. Simply using Numpy indexing and slicing to

# reassign that area of the original image to the blurred roi.


def detect_and_blur_plate(img):
    plate_img = img.copy()
    roi = img.copy()
    plate_rects = plate_cascade.detectMultiScale(plate_img,scaleFactor=1.3, minNeighbors=3)
    for (x,y,w,h) in plate_rects:
        roi = roi[y:y+h,x:x+w]
                    #specify roi
        blurred_roi = cv2.medianBlur(roi,7)
                    # blur it
        plate_img[y:y+h,x:x+w] = blurred_roi
```

```
                # replace the roi with blurd img
    return plate_img


result = detect_and_blur_plate(img)
display(result)
```

```
# *** Object Tracking *** #


# # Optical Flow

# restart the kernel if v ever run these cells,

# as the tracking algo can get caught in a loop with camera.

# ## Lucas-Kanade Optical Flow

import numpy as np

import cv2

# Parameters for ShiTomasi corner detection (good features to track paper)

# these r corner tracking pars

corner_track_params = dict(maxCorners = 10,

                    qualityLevel = 0.3,

                    minDistance = 7,

                    blockSize = 7 )

# detect 10 corners on first frame and track m
```

```
# ### Parameters for Lucas Kanade Optical Flow
# Detect the motion of specific points or the aggregated motion of regions by modifying
# the winSize argument. This determines the integration window size. Small windows are
# more sensitive to noise and miss larger motions. Large windows will “survive” an
# occlusion.
# The integration appears smoother with the larger window size.
# criteria has two here - the max number (10 above) of iterations and epsilon
# (0.03 above). More iterations means a more exhaustive search, and a smaller epsilon
# finishes earlier. These are useful in exchanging speed vs accuracy, but mainly
# stay the same.
# When maxLevel is 0, it is the same algorithm without using pyramids
# (ie, calcOpticalFlowLK). Pyramids allow finding optical flow at various resolutions
# of the image.
# Parameters for lucas kanade optical flow
lk_params = dict( winSize  = (200,200),
                  maxLevel = 2,
                  criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10,0.03))
# eps is epsilon 0.03
# max num of iterations is 10
# more iters means more search in the crnt frame vs prev frame
# a samll eps means finish earlier
# v need to play with these vals
# v r exchanging speed of tracking vs accuracy of tracking
# these r lucas kanade pars
```

```
# Capture the video
cap = cv2.VideoCapture(0)
# Grab the very first frame of the stream
ret, prev_frame = cap.read()
# read first frame and rename it as prev frame
# Grab a grayscale image (We will refer to this as the previous frame)
prev_gray = cv2.cvtColor(prev_frame, cv2.COLOR_BGR2GRAY)

# Wht r the pts to track
# Get the corners
prevPts = cv2.goodFeaturesToTrack(prev_gray, mask = None, **corner_track_params)

# Create a matching mask of the previous frame for drawing later
mask = np.zeros_like(prev_frame)
# For displaying the pts and tracking em
# creates a np arr of same shape as prev_frame

while True:
    # Grab current frame
    ret,frame = cap.read()

        # Grab gray scale
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Calculate the Optical Flow on the Gray Scale Frame
```

```
nextPts, status, err = cv2.calcOpticalFlowPyrLK(prev_gray, frame_gray, prevPts, None, **lk_params)
        # pyr stands for pyramid lucas kanade
        # 3 objs r returnd
        # v r passing in the prev frame/img, then the crnt frame/img,
        # then the prev pts
        # nd get the nextPts


        # Using the returned status array (the status output)
# status output status vector (of unsigned chars);
        # each element of the vector is set to 1 if
# the flow for the corresponding features has been found,
        # otherwise, it is set to 0.
good_new = nextPts[status==1]
good_prev = prevPts[status==1]
        # this is connecting the prev pts to the next pts



# Use ravel to get points to draw lines and circles
for i,(new,prev) in enumerate(zip(good_new,good_prev)):
   x_new,y_new = new.ravel()
   x_prev,y_prev = prev.ravel()
                # flatten
                # a cmplx np arr tracking 10 dfrnt pts or corners


                # draw lines using the mask created
```

```
                # from the prev/first frame to crnt frame
                # green color & 3 is thickness
    mask = cv2.line(mask, (x_new,y_new),(x_prev,y_prev), (0,255,0), 3)
                # draw line connecting prev pts to crnt pts

                # Draw red circles at corner points
    frame = cv2.circle(frame,(x_new,y_new),8,(0,0,255),-1)
                # Draw circles where the crnt frma is

        # Display the image along with the mask we drew the line on.
  img = cv2.add(frame,mask)
        # add frame of circles and masks of lines
  cv2.imshow('tracking',img)
  k = cv2.waitKey(30) & 0xff
  if k == 27:
    break

        # update the previous frame and previous points for the next iter
  prev_gray = frame_gray.copy()
  prevPts = good_new.reshape(-1,1,2)

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

 # # Dense Optical Flow in OpenCV

 # calcOpticalFlowFarneback(prev, next, flow, pyr_scale, levels, winsize, iterations,

 # poly_n, poly_sigma, flags) -> flow

 # This function computes a dense optical flow using the Gunnar Farneback's algorithm.

 # parameters for the function :

 # * prev first 8-bit single-channel input image.

 # * next second input image of the same size and the same type as prev.

 # * flow computed flow image that has the same size as prev and type CV_32FC2.

 # * pyr_scale parameter, specifying the image scale (\<1) to build pyramids for each image

```
#     * pyr_scale=0.5 means a classical pyramid, where each next layer is twice smaller than the previous one.

#

# * levels number of pyramid layers including the initial image;

# levels=1 means that no extra layers are created and only the original images are used.

# * winsize averaging window size

#     * larger values increase the algorithm robustness to image

# * noise and give more chances for fast motion detection, but yield more blurred motion field.

# * iterations number of iterations the algorithm does at each pyramid level.

# * poly_n size of the pixel neighborhood used to find polynomial expansion in each pixel

#     * larger values mean that the image will be approximated with smoother surfaces, yielding more robust algorithm and more blurred motion field, typically poly_n =5 or 7.

# * poly_sigma standard deviation of the Gaussian that is used to smooth derivatives used as a basis for the polynomial expansion; for poly_n=5, you can set poly_sigma=1.1, for poly_n=7, a good value would be poly_sigma=1.5.


import cv2

import numpy as np


# Capture the frame

cap = cv2.VideoCapture(0)

ret, frame1 = cap.read()


# Get gray scale image of first frame and make a mask in HSV color

prvsImg = cv2.cvtColor(frame1,cv2.COLOR_BGR2GRAY)


# create a hsv mask
```

```
hsv_mask = np.zeros_like(frame1)
hsv_mask[:,:,1] = 255
# all x & y pts in 1 is saturation channel
# get the sat channel and set it to 255 means fully saturated



while True:
    ret, frame2 = cap.read()
            # get the frame
    nextImg = cv2.cvtColor(frame2,cv2.COLOR_BGR2GRAY)
            # convert to gray

    # Check out the markdown text above for a break down of these paramters,
            # these are suggested defaults
    flow = cv2.calcOpticalFlowFarneback(prvsImg,nextImg, None, 0.5, 3, 15, 3, 5, 1.2, 0)
    # use defaults

    # Color the channels based on the angle of travel
    # Pay close attention to your video, the path of the direction of flow will determine color!
    mag, ang = cv2.cartToPolar(flow[:,:,0], flow[:,:,1],angleInDegrees=True)
            # cnvrts cart cords to polar cords, cnvrt x, y coords to mag, ang

    hsv_mask[:,:,0] = ang/2
            # get the hue chnl and div by 2
            # this will reduce the hues nd chng the colros when v mv lft to rt
```

```
    hsv_mask[:,:,2] = cv2.normalize(mag,None,0,255,cv2.NORM_MINMAX)
            # normalize the val chnl
            # stretching the min & max to 0 to 255

    # Convert back to BGR to show with imshow from cv
    bgr = cv2.cvtColor(hsv_mask,cv2.COLOR_HSV2BGR)
    cv2.imshow('frame2',bgr)

    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

    # Set the Previous image as the next iamge for the loop
    prvsImg = nextImg
cap.release()
cv2.destroyAllWindows()
# Mv frm left to rt vs rt to left vs up to below vs below to up

import cv2
cv2.cartToPolar
```

```
# # MeanShift Tracking

import numpy as np

import cv2


# Capture a video stream

cap = cv2.VideoCapture(0)


# take first frame of the video

ret,frame = cap.read()


print(frame)
```

```
# Set Up the Initial Tracking Window

# first detect the face and set that as our starting box.

face_cascade = cv2.CascadeClassifier('haarcascades\\haarcascade_frontalface_default.xml')

face_rects = face_cascade.detectMultiScale(frame)

# this gvs a list of np arrs

# v only tk one 1 arr to detect face

# prev v used corner dection now v r using obj detection

# to get face loc, whi is a grp of pxls and apply MS tracking

# in other words v r telling MS to detect face 1 time in the begining

# nd then tell MS alg to track tht set of pxls


print(face_rects)


# Convert this list of a single array to a tuple of (x,y,w,h)

(face_x,face_y,w,h) = tuple(face_rects[0])

# get the first face only


track_window = (face_x,face_y,w,h)

# this is a rect, then track it


# set up the ROI for tracking

roi = frame[face_y:face_y+h, face_x:face_x+w]

#get roi
```

```
# Use the HSV Color Mapping
hsv_roi =  cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)

# Find histogram to backproject the target on each frame for calculation
# of meanshit
roi_hist = cv2.calcHist([hsv_roi],[0],None,[180],[0,180])
# this is hist

# Normalize the histogram array values given a min of 0 and max of 255
cv2.normalize(roi_hist,roi_hist,0,255,cv2.NORM_MINMAX)

# Setup the termination criteria, either 10 iterations or
# move by at least 1 pt
term_crit = ( cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1 )
# these r default vals, v can chng them

while True:
   ret ,frame = cap.read()
         # if the frame is returnd
   if ret == True:
      # Grab the Frame in HSV
      hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

            # Calculate the Back Projection based off the roi_hist created
      dst = cv2.calcBackProject([hsv],[0],roi_hist,[0,180],1)
```

```
                # Apply meanshift to get the new coordinates of the rectangle
        ret, track_window = cv2.meanShift(dst, track_window, term_crit)


                # Draw the new rectangle on the image
        x,y,w,h = track_window
        img2 = cv2.rectangle(frame, (x,y), (x+w,y+h), (0,0,255),5)
        cv2.imshow('img2',img2)
        k = cv2.waitKey(1) & 0xff
        if k == 27:
            break
    else:
        break


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

 # Cam shift tracking:

```
import numpy as np
import cv2

# Capture a video stream
cap = cv2.VideoCapture(0)

# take first frame of the video
ret,frame = cap.read()



# Set Up the Initial Tracking Window

print(frame)
# first detect the face and set that as our starting box.
face_cascade = cv2.CascadeClassifier('haarcascades\\haarcascade_frontalface_default.xml')
face_rects = face_cascade.detectMultiScale(frame)
```

```
# Convert this list of a single array to a tuple of (x,y,w,h)
(face_x,face_y,w,h) = tuple(face_rects[0])
track_window = (face_x,face_y,w,h)
# set up the ROI for tracking
roi = frame[face_y:face_y+h, face_x:face_x+w]


# Use the HSV Color Mapping
hsv_roi =  cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)


# Find histogram to backproject the target on each frame for calculation of meanshit
roi_hist = cv2.calcHist([hsv_roi],[0],None,[180],[0,180])


# Normalize the histogram array values given a min of 0 and max of 255
cv2.normalize(roi_hist,roi_hist,0,255,cv2.NORM_MINMAX)


# Setup the termination criteria, either 10 iteration or move by at least 1 pt
term_crit = ( cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1 )


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


      # Grab the Frame in HSV
```

```
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Calculate the Back Projection based off the roi_hist created earlier
    dst = cv2.calcBackProject([hsv],[0],roi_hist,[0,180],1)

    ########## CAM SHIFT ####################################
    # Apply Camshift to get the new coordinates of the rectangle
    ret, track_window = cv2.CamShift(dst, track_window, term_crit)
    # resize the rec based on the position of the face
                # if it is close draw a bigger one vs vice versa

    # Draw it on image
    pts = cv2.boxPoints(ret)
    pts = np.int0(pts)
                # convert em all to ints
    img2 = cv2.polylines(frame,[pts],True, (0,0,255),5)
                # red color
    cv2.imshow('img2',img2)
    ########################################################

    k = cv2.waitKey(1) & 0xff
    if k == 27:
       break

else:
```

```
        break


cv2.destroyAllWindows()

cap.release()
```

```
# Other Tracking apis
def ask_for_tracker():
    print("What Tracker API would you like to use?")
    print("Enter 0 for BOOSTING: ")
    print("Enter 1 for MIL: ")
    print("Enter 2 for KCF: ")
    print("Enter 3 for TLD: ")
    print("Enter 4 for MEDIANFLOW: ")
    choice = input("Please select your tracker: ")

    if choice == '0':
        tracker = cv2.TrackerBoosting_create()
    if choice == '1':
        tracker = cv2.TrackerMIL_create()
    if choice == '2':
        tracker = cv2.TrackerKCF_create()
    if choice == '3':
        tracker = cv2.TrackerTLD_create()
    if choice == '4':
        tracker = cv2.TrackerMedianFlow_create()
    return tracker


tracker = ask_for_tracker()
```

```
tracker_name = str(tracker).split()[0][1:]
print(tracker_name)


# Read video
cap = cv2.VideoCapture(0)
# Read first frame.
ret, frame = cap.read()
# Special function allows us to draw on the very first frame our desired ROI
roi = cv2.selectROI(frame, False)
# Initialize tracker with first frame and bounding box
ret = tracker.init(frame, roi)
while True:
    # Read a new frame
    ret, frame = cap.read()
    # Update tracker
    success, roi = tracker.update(frame)
    # roi variable is a tuple of 4 floats
    # We need each value and we need them as integers
    (x,y,w,h) = tuple(map(int,roi))
    # Draw Rectangle as Tracker moves
    if success:
        # Tracking success
        p1 = (x, y)
        p2 = (x+w, y+h)
        cv2.rectangle(frame, p1, p2, (0,255,0), 3)
```

```
    else :

        # Tracking failure

        cv2.putText(frame, "Failure to Detect Tracking!!", (100,200), cv2.FONT_HERSHEY_SIMPLEX,
1,(0,0,255),3)

    # Display tracker type on frame

    cv2.putText(frame, tracker_name, (20,400), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255,0),3);

    # Display result

    cv2.imshow(tracker_name, frame)

    # Exit if ESC pressed

    k = cv2.waitKey(1) & 0xff

    if k == 27 :

        break


cap.release()

cv2.destroyAllWindows()
```

```
'''python C:\Sai\SaiPython\road-traffic-sign.py
Stop
Forward
Right'''
# Traffic Sign
import cv2
import numpy as np
from scipy.stats import itemfreq


def get_dominant_color(image, n_colors):
    pixels = np.float32(image).reshape((-1, 3))
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 200, .1)
    flags = cv2.KMEANS_RANDOM_CENTERS
    flags, labels, centroids = cv2.kmeans(
        pixels, n_colors, None, criteria, 10, flags)
    palette = np.uint8(centroids)
    '''print('palette')
    print(palette)
```

```
    print('palette[np.argmax(itemfreq(labels)[:, -1])]')'''
    print(palette[np.argmax(itemfreq(labels)[:, -1])])
    return palette[np.argmax(itemfreq(labels)[:, -1])]



clicked = False
def onMouse(event, x, y, flags, param):
    global clicked
    if event == cv2.EVENT_LBUTTONUP:
        clicked = True



cameraCapture = cv2.VideoCapture(0)
cv2.namedWindow('camera')
cv2.setMouseCallback('camera', onMouse)

# Read and process frames in loop
success, frame = cameraCapture.read()

while success and not clicked:
    cv2.waitKey(1)
    success, frame = cameraCapture.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    img = cv2.medianBlur(gray, 37)
```

```
circles = cv2.HoughCircles(img, cv2.HOUGH_GRADIENT,
                    1, 50, param1=120, param2=40)


if not circles is None:
   circles = np.uint16(np.around(circles))
   max_r, max_i = 0, 0
   for i in range(len(circles[:, :, 2][0])):
      if circles[:, :, 2][0][i] > 50 and circles[:, :, 2][0][i] > max_r:
         max_i = i
         max_r = circles[:, :, 2][0][i]
   x, y, r = circles[:, :, :][0][max_i]
   if y > r and x > r:
      square = frame[y-r:y+r, x-r:x+r]


      dominant_color = get_dominant_color(square, 2)
      if dominant_color[2] > 100:
         print("STOP")
      elif dominant_color[0] > 80:
         zone_0 = square[square.shape[0]*3//8:square.shape[0]
                  * 5//8, square.shape[1]*1//8:square.shape[1]*3//8]
         cv2.imshow('Zone0', zone_0)
         zone_0_color = get_dominant_color(zone_0, 1)


         zone_1 = square[square.shape[0]*1//8:square.shape[0]
                  * 3//8, square.shape[1]*3//8:square.shape[1]*5//8]
```

```
        cv2.imshow('Zone1', zone_1)
        zone_1_color = get_dominant_color(zone_1, 1)

        zone_2 = square[square.shape[0]*3//8:square.shape[0]
                        * 5//8, square.shape[1]*5//8:square.shape[1]*7//8]
        cv2.imshow('Zone2', zone_2)
        zone_2_color = get_dominant_color(zone_2, 1)

        if zone_1_color[2] < 60:
            if sum(zone_0_color) > sum(zone_2_color):
                print("LEFT")
            else:
                print("RIGHT")
        else:
            if sum(zone_1_color) > sum(zone_0_color) and sum(zone_1_color) > sum(zone_2_color):
                print("FORWARD")
            elif sum(zone_0_color) > sum(zone_2_color):
                print("FORWARD AND LEFT")
            else:
                print("FORWARD AND RIGHT")
    else:
        print("N/A")

for i in circles[0, :]:
    cv2.circle(frame, (i[0], i[1]), i[2], (0, 255, 0), 2)
```

```
            cv2.circle(frame, (i[0], i[1]), 2, (0, 0, 255), 3)
    cv2.imshow('camera', frame)
            # Exit if ESC pressed
    k = cv2.waitKey(1) & 0xff
    if k == 27 :
        break

cv2.destroyAllWindows()
cameraCapture.release()
```

```
# Car detection

cd E:\...\R_Datasets

python vehicle_detection.py


E:\...\R_Datasets>

# -*- coding: utf-8 -*-


import cv2

print(cv2.__version__)


cascade_src = 'cars.xml'

video_src = 'video2.avi'

#video_src = 'dataset/video2.avi'


cap = cv2.VideoCapture(video_src)

car_cascade = cv2.CascadeClassifier(cascade_src)


while True:
```

```
    ret, img = cap.read()
    if (type(img) == type(None)):
        break

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    cars = car_cascade.detectMultiScale(gray, 1.1, 1)

    for (x,y,w,h) in cars:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,255),2)

    cv2.imshow('video', img)

    if cv2.waitKey(33) == 27:
        break

cv2.destroyAllWindows()
```

 # make sure the following files r avlbl in Datasets dir

'''

yolo.cfg

yad2k.py

darknet53.py

yolo_model2.py

coco_classes.txt

yolo.h5

'''

```
import numpy as np
import matplotlib.pyplot as plt
get_ipython().magic('matplotlib inline')
from PIL import Image
import os
print(os.getcwd())
os.chdir('E:\\Locker\\Sai\\SaiHCourseNait\\DecBtch\\R_Datasets\\')
print(os.getcwd())


import os
import time
import cv2
import numpy as np
from yolo_model2 import YOLO

def process_image(img):
    """Resize, reduce and expand image.

    # Argument:
        img: original image.

    # Returns
        image: ndarray(64, 64, 3), processed image.
    """
    image = cv2.resize(img, (416, 416),
```

```
                interpolation=cv2.INTER_CUBIC)
    image = np.array(image, dtype='float32')
    image /= 255.
    image = np.expand_dims(image, axis=0)

    return image

def get_classes(file):
    """Get classes name.

    # Argument:
        file: classes name for database.

    # Returns
        class_names: List, classes name.

    """
    with open(file) as f:
        class_names = f.readlines()
    class_names = [c.strip() for c in class_names]

    return class_names

def draw(image, boxes, scores, classes, all_classes):
    """Draw the boxes on the image.
```

```
# Argument:
    image: original image.
    boxes: ndarray, boxes of objects.
    classes: ndarray, classes of objects.
    scores: ndarray, scores of objects.
    all_classes: all classes name.
"""
for box, score, cl in zip(boxes, scores, classes):
    x, y, w, h = box

    top = max(0, np.floor(x + 0.5).astype(int))
    left = max(0, np.floor(y + 0.5).astype(int))
    right = min(image.shape[1], np.floor(x + w + 0.5).astype(int))
    bottom = min(image.shape[0], np.floor(y + h + 0.5).astype(int))

    cv2.rectangle(image, (top, left), (right, bottom), (255, 0, 0), 2)
    cv2.putText(image, '{0} {1:.2f}'.format(all_classes[cl], score),
                (top, left - 6),
                cv2.FONT_HERSHEY_SIMPLEX,
                0.6, (0, 0, 255), 1,
                cv2.LINE_AA)

    print('class: {0}, score: {1:.2f}'.format(all_classes[cl], score))
    print('box coordinate x,y,w,h: {0}'.format(box))
```

```
    print()


def detect_image(image, yolo, all_classes):
    """Use yolo v3 to detect images.

    # Argument:
        image: original image.
        yolo: YOLO, yolo model.
        all_classes: all classes name.

    # Returns:
        image: processed image.
    """
    pimage = process_image(image)

    start = time.time()
    boxes, classes, scores = yolo.predict(pimage, image.shape)
    end = time.time()

    print('time: {0:.2f}s'.format(end - start))

    if boxes is not None:
        draw(image, boxes, scores, classes, all_classes)
```

```
    return image


def detect_video(video, yolo, all_classes):
    """Use yolo v3 to detect video.

    # Argument:
        video: video file.
        yolo: YOLO, yolo model.
        all_classes: all classes name.
    """
    video_path = os.path.join("videos", "test", video)
    camera = cv2.VideoCapture(video_path)
    cv2.namedWindow("detection", cv2.WINDOW_AUTOSIZE)

    # Prepare for saving the detected video
    sz = (int(camera.get(cv2.CAP_PROP_FRAME_WIDTH)),
        int(camera.get(cv2.CAP_PROP_FRAME_HEIGHT)))
    fourcc = cv2.VideoWriter_fourcc(*'mpeg')


    vout = cv2.VideoWriter()
    vout.open(os.path.join("videos", "res", video), fourcc, 20, sz, True)

    while True:
        res, frame = camera.read()
```

```
        if not res:
            break

        image = detect_image(frame, yolo, all_classes)
        cv2.imshow("detection", image)

        # Save the video frame by frame
        vout.write(image)

        if cv2.waitKey(110) & 0xff == 27:
                break

    vout.release()
    camera.release()


yolo = YOLO(0.6, 0.5)
# this is keras model, look at yolo.py
file = 'coco_classes.txt' #'data/coco_classes.txt'
all_classes = get_classes(file)

f = 'person.jpg' #'jingxiang-gao-489454-unsplash.jpg'
path = f #'images/'+f
image = cv2.imread(path)
```

```
plt.imshow(image)


image = detect_image(image, yolo, all_classes)


cv2.imwrite('images/res/' + f, image)

plt.imshow(image)




# lower the thr, probs

yolo = YOLO(0.3, 0.5)

file = 'coco_classes.txt'

all_classes = get_classes(file)

f = 'four_people.jpg'

path = f

image = cv2.imread(path)

plt.imshow(image)

image = detect_image(image, yolo, all_classes)

cv2.imwrite('images/res/' + f, image)

plt.imshow(image)
```