*'''  
Example code for live video processing  
Also multithreaded video processing sample using opencv 3.4  
  
Usage:  
 python testcv\_mt.py {<video device number>|<video file name>}  
  
 Use this code as a template for live video processing  
  
 Also shows how python threading capabilities can be used  
 to organize parallel captured frame processing pipeline  
 for smoother playback.  
  
Keyboard shortcuts: (video display window must be selected  
  
 ESC - exit  
 space - switch between multi and single threaded processing  
 a - adjust contrast, brightness, and gamma  
 d - running difference of current and previous image  
 e - displays canny edges  
 f - displays raw frames  
 b - apply sepFilter2D  
 m - morph erosion  
 c - apply colormap  
 p - apply sobel filter  
 r - colormap in rectangular region  
 h - display image hue band  
 o - apply a 5x5 "fat plus" opening to the thresholded image  
 q - histogram equalize value image  
 t - do thresholding  
 v - write video output frames to file "vid\_out.avi"  
'''*# import the necessary packages  
from \_\_future\_\_ import print\_function  
from \_\_future\_\_ import division  
import cv2 as cv  
import numpy as np  
from multiprocessing.pool import ThreadPool  
from collections import deque  
from common import clock, draw\_str, StatValue  
import video  
import argparse  
# import math  
# from matplotlib import pyplot as plt  
  
  
# used to execute process\_frame when in non threaded mode  
class DummyTask:  
 def \_\_init\_\_(self, data):  
 self.data = data  
  
 def ready(self):  
 return True  
  
 def get(self):  
 return self.data  
  
  
  
# initialize global variables  
frame\_counter = 0  
show\_frames = True  
diff\_frames = False  
show\_edges = False  
show\_hue = False  
do\_threshold = False  
adj\_img = False  
adj\_gam = False  
do\_blur= False  
do\_sobel = False  
do\_detect = False  
color\_map = False  
color\_mapwithrectangle = False  
m\_open = False  
m\_erosion= False  
hist\_eq = False  
vid\_frames = False  
contrast = 128  
contrast\_slider\_max = 255  
brightness = 128  
brightness\_slider\_max = 255  
gamma = 128  
gamma\_slider\_max = 255  
threshold = 128  
threshold\_slider\_max = 255  
sigma= 0  
sigma\_slider\_max=31  
  
  
# this routine is run each time a new video frame is captured  
def process\_frame(frame, prevFrame, t0):  
  
   
  
  
 if adj\_img:  
 global contrast, brightness, gamma  
 # shift value to get actual brightness offset  
 brite = brightness - 128  
 # compute the contrast value from the trackbar setting  
 if contrast > 127:  
 contr = 1+(5\*(contrast - 128)/128)  
 else:  
 contr = 1/(1+(5\*(128 - contrast)/128))  
 # adjust brightness and contrast  
 frame = ((np.float\_(frame)-128) \* contr) + 128 + brite  
 # compute the gamma value from the trackbar setting  
 if gamma > 127:  
 gam = 1+(2\*(gamma - 128)/128)  
 else:  
 gam = 1/(1+(2\*(128 - gamma)/128))  
 # apply the gamma function  
 frame = 255 \* ((frame / 255) \*\* (1 / gam))  
 # then convert the result back to uint8 after clipping at 0 and 255  
 frame = np.uint8(np.clip(frame, 0, 255))  
  
 if hist\_eq:  
 # convert image to HSV  
 hsv = cv.cvtColor(frame, cv.COLOR\_BGR2HSV)  
 hue, sat, val = hsv[:, :, 0], hsv[:, :, 1], hsv[:, :, 2] # separate the channels  
 val = cv.equalizeHist(val)  
 hsv = cv.merge((hue, sat, val))  
 frame = cv.cvtColor(hsv, cv.COLOR\_HSV2BGR)  
  
 if not show\_frames and show\_edges: # edges alone  
 edges = cv.Canny(frame, 100, 200)  
 thisFrame = cv.cvtColor(edges, cv.COLOR\_GRAY2RGB, edges)  
 elif show\_frames and show\_edges: # edges and frames  
 edges = cv.Canny(frame, 100, 200)  
 edges = cv.cvtColor(edges, cv.COLOR\_GRAY2RGB, edges)  
 thisFrame = cv.add(frame, edges)  
 else: # current frame  
 thisFrame = frame.copy()  
  
 if do\_threshold:  
 # create threshold mask  
 threshMask = get\_threshold\_mask(frame)  
 # apply the mask  
 thisFrame = threshMask \* thisFrame  
  
 if do\_blur:  
 kernel = cv.getGaussianKernel(11, 4)  
 blur = cv.sepFilter2D(thisFrame, -1, kernel, kernel)  
 thisFrame = cv.sepFilter2D(blur,-1, kernel,kernel)  
  
 if do\_sobel:  
 # apply sobel filter  
 global sigma  
 Blur = cv.Sobel(thisFrame, -1, 1, 0, ksize=3, scale=1, delta=0)  
 thisFrame = cv.Sobel(Blur, -1, 1, 0, ksize=3, scale=1, delta=0)  
  
 if m\_erosion:  
 kernel = np.ones((7, 7), np.uint8)  
 erosion = cv.erode(thisFrame, kernel, iterations = 1)  
 thisFrame = cv.erode (erosion,kernel, iterations = 1)  
 #dilation = cv.dilate(frame, kernel, iterations=1)  
  
 if color\_map:  
 color = cv.applyColorMap(thisFrame, cv.COLORMAP\_JET)  
 thisFrame = cv.applyColorMap(color,cv.COLORMAP\_JET)  
  
  
 # rectangle = cv.rectangle(frame, (350, 200), (510, 128), (0, 255, 0), 3)  
 #rectangle = frame[350: 250, 510:110]  
 #rec\_color = cv.applyColorMap(rectangle, cv.COLORMAP\_JET)  
 #cv.imshow('rec\_color', rec\_color)  
 #cv.waitKey(0)  
  
  
 if show\_hue:  
 # convert image to HSV  
 hsv = cv.cvtColor(thisFrame, cv.COLOR\_BGR2HSV)  
 hue, sat, val = hsv[:, :, 0], hsv[:, :, 1], hsv[:, :, 2] # separate the channels  
 # the maximum hue is 170, so scale it into [0,255]  
 h32 = np.float32(hue) \* 255 / 170  
 sch = np.uint8(np.clip(h32, 0, 255)) # clip at 255 and convert back to uint8  
 # apply the opencv builtin hue colormap  
 thisFrame = cv.applyColorMap(sch, cv.COLORMAP\_HSV)  
  
 if diff\_frames:  
 # compute absolute difference between the current and previous frame  
 difframe = cv.absdiff(thisFrame, prevFrame)  
 # save current frame as previous  
 prevFrame = thisFrame.copy()  
 # set the current frame to the difference image  
 thisFrame = difframe.copy()  
 else:  
 # save current frame as previous  
 prevFrame = thisFrame.copy()  
  
 return thisFrame, prevFrame, t0  
  
  
def get\_threshold\_mask(frame):  
 global threshold  
 # mB, mG, mR, \_ = np.uint8(cv.mean(frame))  
 B, G, R = frame[:, :, 0], frame[:, :, 1], frame[:, :, 2]  
 # \_, tB = cv.threshold(B, mB, 1, cv.THRESH\_BINARY)  
 # \_, tG = cv.threshold(G, mG, 1, cv.THRESH\_BINARY)  
 # \_, tR = cv.threshold(R, mR, 1, cv.THRESH\_BINARY)  
 \_, tB = cv.threshold(B, threshold, 1, cv.THRESH\_BINARY)  
 \_, tG = cv.threshold(G, threshold, 1, cv.THRESH\_BINARY)  
 \_, tR = cv.threshold(R, threshold, 1, cv.THRESH\_BINARY)  
 if m\_open:  
 # create structuring element for morph ops  
 se = cv.getStructuringElement(cv.MORPH\_ELLIPSE, (5, 5))  
 tB = cv.morphologyEx(tB, cv.MORPH\_OPEN, se, 1)  
 tG = cv.morphologyEx(tG, cv.MORPH\_OPEN, se, 1)  
 tR = cv.morphologyEx(tR, cv.MORPH\_OPEN, se, 1)  
 threshMask = cv.merge((tB, tG, tR))  
  
 return threshMask  
  
  
def on\_brightness\_trackbar(val):  
 global brightness  
 brightness = val  
  
def on\_sigma\_trackbar (val):  
 global sigma  
 sigma = val  
  
  
def on\_contrast\_trackbar(val):  
 global contrast  
 contrast = val  
  
  
def on\_gamma\_trackbar(val):  
 global gamma  
 gamma = val  
  
  
def on\_threshold\_trackbar(val):  
 global threshold  
 threshold = val  
  
  
# create a video capture object  
def create\_capture(source=0):  
  
 # parse source name (defaults to 0 which is the first USB camera attached)  
  
 source = str(source).strip()  
 chunks = source.split(':')  
 # handle drive letter ('c:', ...)  
 if len(chunks) > 1 and len(chunks[0]) == 1 and chunks[0].isthreshold():  
 chunks[1] = chunks[0] + ':' + chunks[1]  
 del chunks[0]  
  
 source = chunks[0]  
 try:  
 source = int(source)  
 except ValueError:  
 pass  
  
 params = dict(s.split('=') for s in chunks[1:])  
  
 # video capture object defined on source  
  
 timeout = 100  
 iter = 0  
 cap = cv.VideoCapture(source)  
 while (cap is None or not cap.isOpened()) & (iter < timeout):  
 time.sleep(0.1)  
 iter = iter + 1  
 cap = cv.VideoCapture(source)  
  
 if iter == timeout:  
 print('camera timed out')  
 return None  
 else:  
 print(iter)  
  
 if 'size' in params:  
 w, h = map(int, params['size'].split('x'))  
 cap.set(cv.CAP\_PROP\_FRAME\_WIDTH, w)  
 cap.set(cv.CAP\_PROP\_FRAME\_HEIGHT, h)  
  
 if cap is None or not cap.isOpened():  
 print('Warning: unable to open video source: ', source)  
 return None  
  
 return cap  
  
# main program  
if \_\_name\_\_ == '\_\_main\_\_':  
 import sys  
  
 # print in the program shell window the text at the beginning of the file  
 print(\_\_doc\_\_)  
  
 # if there is no argument in the program invocation default to camera 0  
 try:  
 fn = sys.argv[1]  
 except:  
 fn = 0  
  
 # grab initial frame, create window  
 cv.waitKey(1) & 0xFF  
 cap = video.create\_capture(fn)  
 ret, frame = cap.read()  
 frame\_counter += 1  
 height, width, channels = frame.shape  
 prevFrame = frame.copy()  
 cv.namedWindow("video")  
  
 # Create video of Frame sequence -- define the codec and create VideoWriter object  
 fourcc = cv.VideoWriter\_fourcc(\*'XVID')  
 cols = np.int(cap.get(cv.CAP\_PROP\_FRAME\_WIDTH))  
 rows = np.int(cap.get(cv.CAP\_PROP\_FRAME\_HEIGHT))  
 vid\_out = cv.VideoWriter('vid\_out.avi', fourcc, 20.0, (cols, rows))  
  
 # Set up multiprocessing  
 threadn = cv.getNumberOfCPUs()  
 pool = ThreadPool(processes=threadn)  
 pending = deque()  
  
 threaded\_mode = True  
 onOff = False  
  
 # initialize time variables  
 latency = StatValue()  
 frame\_interval = StatValue()  
 last\_frame\_time = clock()  
  
 # main program loop  
 while True:  
  
  
 while len(pending) > 0 and pending[0].ready(): # there are frames in the queue  
 res, prevFrame, t0 = pending.popleft().get()  
 latency.update(clock() - t0)  
 # plot info on threading and timing on the current image  
 # comment out the next 3 lines to skip the plotting  
 draw\_str(res, (20, 20), "threaded : " + str(threaded\_mode))  
 draw\_str(res, (20, 40), "latency : %.1f ms" % (latency.value \* 1000))  
 draw\_str(res, (20, 60), "frame interval : %.1f ms" % (frame\_interval.value \* 1000))  
 # write output video frame  
 if vid\_frames:  
 vid\_out.write(res)  
 # show the current image  
 cv.imshow('video', res)  
  
  
 if len(pending) < threadn: # fewer frames than thresds ==> get another frame  
 # get frame  
 ret, frame = cap.read()  
 frame\_counter += 1  
 t = clock()  
 frame\_interval.update(t - last\_frame\_time)  
 last\_frame\_time = t  
 if threaded\_mode:  
 task = pool.apply\_async(process\_frame, (frame.copy(), prevFrame.copy(), t))  
 else:  
 task = DummyTask(process\_frame(frame, prevFrame, t))  
 pending.append(task)  
  
 # check for a keypress  
 key = cv.waitKey(1) & 0xFF  
  
 # threaded or non threaded mode  
 if key == ord(' '):  
 threaded\_mode = not threaded\_mode  
  
 # apply separate 2D filter  
 if key == ord('b'):  
 do\_blur = not do\_blur  
  
 # apply morphological erosion  
 if key == ord('m'):  
 m\_erosion = not m\_erosion  
  
 # apply sobel filter  
 if key == ord('p'):  
 do\_sobel = not do\_sobel  
  
 # apply operation on region of interest (ROI)  
 if key == ord('r'):  
 color\_mapwithrectangle = not color\_mapwithrectangle  
  
 # toggle point processes -- adjust image  
 if key == ord('a'):  
 adj\_img = not adj\_img  
 if adj\_img:  
 cv.createTrackbar("brightness", 'video', brightness, brightness\_slider\_max, on\_brightness\_trackbar)  
 cv.createTrackbar("contrast", 'video', contrast, contrast\_slider\_max, on\_contrast\_trackbar)  
 cv.createTrackbar("gamma", 'video', gamma, gamma\_slider\_max, on\_gamma\_trackbar)  
 else:  
 cv.destroyWindow('video')  
 cv.namedWindow('video')  
 cv.imshow('video', res)  
 # toggle edges  
 if key == ord('e'):  
 show\_edges = not show\_edges  
 if not show\_edges and not show\_frames:  
 show\_frames = True  
  
 # apply color map  
 if key == ord('c'):  
 color\_map = not color\_map  
  
 # toggle frames  
 if key == ord('f'):  
 show\_frames = not show\_frames  
 if not show\_frames and not show\_edges:  
 show\_frames = True  
 # image difference mode  
 if key == ord('d'):  
 diff\_frames = not diff\_frames  
 # display image hue band  
 if key == ord('h'):  
 show\_hue = not show\_hue  
 # equalize image value band  
 if key == ord('q'):  
 hist\_eq = not hist\_eq  
 # threshold the image  
 if key == ord('t'):  
 do\_threshold = not do\_threshold  
 if do\_threshold:  
 cv.createTrackbar("threshold", 'video', threshold, threshold\_slider\_max, on\_threshold\_trackbar)  
 else:  
 cv.destroyWindow('video')  
 cv.namedWindow('video')  
 cv.imshow('video', res)  
 # do morphological opening on thresholded image (only applied to thresholded image)  
 if key == ord('o'):  
 m\_open = not m\_open  
  
 # write video frames  
 if key == ord('v'):  
 vid\_frames = not vid\_frames  
 if vid\_frames:  
 print("Frames are being output to video")  
 else:  
 print("Frames are not being output to video")  
  
 #if key == ord('q')  
  
  
 # ESC terminates the program  
 if key == 27:  
 break  
  
# release video capture object  
cap.release()  
# release video output object  
vid\_out.release()  
cv.destroyAllWindows()