code from https://www.pyimagesearch.com/2018/07/30/opencv-object-tracking/

import the necessary packages

```
from imutils.video import VideoStream
from imutils.video import FPS
import argparse
import imutils
import time
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt
import glob
```

run at command prompt with:

python opencv_object_tracker.py --video .\vid_1540304255.h264 --tracker kcf

python opencv_cobble_tracker.py --video .\vid_1540311466.h264 --tracker kcf

python opencv_cobble_tracker.py --video .\timeavg_test.webm --tracker kcf

cd Projects\AdvocateBeach2018\src\data\ope

nCV

homechar = os.path.expanduser('~')

tide = 'tide19'
vidchunk = 'position2'
vid = 'vid_1540304255' # pos1

vid = 'vid_1540307860' # pos2, 3

vid = 'vid_1540311466' # pos4

stonefile = 'yellow_01.npy'

stonefile = 'red_01.npy'

stonefile = 'blue_18.npy'

stonefile = 'green_02.npy'

5 is questionable

#pos1 - did 7 move?

load in trajectories already logged

traj_dir =
os.path.join(homechar,'Projects','AdvocateBeach2018','data','interim','cobble_tracking',tide,vidchunk,vid)
yellows = glob.glob(os.path.join(traj_dir, 'yellow_*.npy')) reds = glob.glob(os.path.join(traj_dir, 'red_*.npy'))
blues = glob.glob(os.path.join(traj_dir, 'blue_*.npy'))

sv = os.path.join(traj_dir, stonefile)
if os.path.exists(sv):

```
# print('File already exists! Change filename.')
raise Exception('File already exists! Change filename.')

yellow_trajs = {}
for i in range(len(yellows)):
if os.path.exists(os.path.join(traj_dir, yellows[i])):
yellow_trajs[i] = np.load(os.path.join(traj_dir, yellows[i])).item()

red_trajs = {}
for i in range(len(reds)):
if os.path.exists(os.path.join(traj_dir, reds[i])):
red_trajs[i] = np.load(os.path.join(traj_dir, reds[i])).item()

blue_trajs = {}
for i in range(len(blues)):
if os.path.exists(os.path.join(traj_dir, blues[i])):
blue_trajs[i] = np.load(os.path.join(traj_dir, blues[i])).item()
```

construct the argument parser and parse the arguments

```
ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video", type=str,
help="path to input video file")
ap.add_argument("-t", "--tracker", type=str, default="kcf",
help="OpenCV object tracker type")
args = vars(ap.parse_args())
```

extract the OpenCV version info

```
(major, minor) = cv2.version.split(".")[:2]
```

if we are using OpenCV 3.2 OR BEFORE, we can use a special factory

function to create our object tracker

if int(major) == 3 and int(minor) < 3:
tracker = cv2.Tracker_create(args["tracker"].upper())

otherwise, for OpenCV 3.3 OR NEWER, we need to explicity call the

approrpiate object tracker constructor:

else:
# initialize a dictionary that maps strings to their corresponding
# OpenCV object tracker implementations
OPENCV_OBJECT_TRACKERS = {
"csrt": cv2.TrackerCSRT_create,
"kcf": cv2.TrackerKCF_create,
"boosting": cv2.TrackerBoosting_create,
"mil": cv2.TrackerMIL_create,
"tld": cv2.TrackerTLD_create,
"medianflow": cv2.TrackerMedianFlow_create,
"mosse": cv2.TrackerMOSSE_create
}

```
# grab the appropriate object tracker using our dictionary of
# OpenCV object tracker objects
tracker = OPENCV_OBJECT_TRACKERS[args["tracker"]]()
```

initialize the bounding box coordinates of the object we are going

to track

initBB = None

if a video path was not supplied, raise exception //# grab the reference to the

web cam

if not args.get("video", False):
raise Exception('No video file provided.')

otherwise, grab a reference to the video file

else:
vs = cv2.VideoCapture(args["video"])

initialize the FPS throughput estimator

fps = None

counter = -1 # for keeping track of time

counter_vec = []
position_vec = []

x = []

loop over frames from the video stream

while True:

```
counter += 1

# grab the current frame, then handle if we are using a
# VideoStream or VideoCapture object
frame = vs.read()
frame = frame[1] if args.get("video", False) else frame

# check to see if we have reached the end of the stream
if frame is None:
    break

# resize the frame (so we can process it faster) and grab the
# frame dimensions
frame = imutils.resize(frame, width=1000) # was 500
```

```
(H, W) = frame.shape[:2]


# check to see if we are currently tracking an object
if initBB is not None:
    # grab the new bounding box coordinates of the object
    (success, box) = tracker.update(frame)

    # check to see if the tracking was a success
    if success:
        (x, y, w, h) = [int(v) for v in box]
        cv2.rectangle(frame, (x, y), (x + w, y + h),
            (0, 255, 0), 2)

    # update the FPS counter
    fps.update()
    fps.stop()

    # initialize the set of information we'll be displaying on
    # the frame
    info = [
        ("Tracker", args["tracker"]),
        ("Success", "Yes" if success else "No"),
        ("FPS", "{:.2f}".format(fps.fps())),
    ]

    # loop over the info tuples and draw them on our frame
    for (i, (k, v)) in enumerate(info):
        text = "{}: {}".format(k, v)
        cv2.putText(frame, text, (10, H - ((i * 20) + 20)),
            cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)

    ### draws circle on previously logged positions
    for j in range(len(yellow_trajs)):
        cmin = np.int(np.min(np.abs(counter - np.array(yellow_trajs[j]['count']))))
        if cmin is 0:
            Imin = np.argmin(np.abs(counter - np.array(yellow_trajs[j]['count'])))
            cv2.circle(frame,yellow_trajs[j]['position'][Imin], 5, (0,100,100), -1)

    for j in range(len(red_trajs)):
        cmin = np.int(np.min(np.abs(counter - np.array(red_trajs[j]['count']))))
        if cmin is 0:
            Imin = np.argmin(np.abs(counter - np.array(red_trajs[j]['count'])))
            cv2.circle(frame,red_trajs[j]['position'][Imin], 5, (0,0,100), -1)

    for j in range(len(blue_trajs)):
        cmin = np.int(np.min(np.abs(counter - np.array(blue_trajs[j]['count']))))
        if cmin is 0:
            Imin = np.argmin(np.abs(counter - np.array(blue_trajs[j]['count'])))
            cv2.circle(frame,blue_trajs[j]['position'][Imin], 5, (100,0,0), -1)
    ###
    # added by TBG to deal with loss of tracker
    if not success:
        tracker = OPENCV_OBJECT_TRACKERS[args["tracker"]]()
        cv2.imshow("Frame", frame)
```

```
            # wait for input
            act = cv2.waitKey(0)
            if act == ord("q"):
                break

            elif act == ord("r"):
                # redraw bounding box
                # select the bounding box of the object we want to track (make
                # sure you press ENTER or SPACE after selecting the ROI)
                initBB = cv2.selectROI("Frame", frame, fromCenter=True,
                    showCrosshair=True)

                # start OpenCV object tracker using the supplied bounding box
                # coordinates, then start the FPS throughput estimator as well
                tracker.init(frame, initBB)
                fps = FPS().start()

            #elif act == ord(" "):

        if x:
            position_vec.append((x + np.int(w/2), y + np.int(h/2)))
            counter_vec.append(counter)


            # show the output frame
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF


    # if the 's' key is selected, we are going to "select" a bounding
    # box to track
    if key == ord("s"):
        # select the bounding box of the object we want to track (make
        # sure you press ENTER or SPACE after selecting the ROI)
        initBB = cv2.selectROI("Frame", frame, fromCenter=True,
            showCrosshair=True)

        # start OpenCV object tracker using the supplied bounding box
        # coordinates, then start the FPS throughput estimator as well
        tracker.init(frame, initBB)
        fps = FPS().start()

        # if the `q` key was pressed, break from the loop
    elif key == ord("q"):
        break
```

traj = {'count': counter_vec, 'position': position_vec}

if not os.path.exists(traj_dir):

try:

os.makedirs(traj_dir)

except OSError as exc: # Guard against race condition
if exc.errno != errno.EEXIST:
raise

np.save(sv, traj)

if we are using a webcam, release the pointer

if not args.get("video", False):
vs.stop()

otherwise, release the file pointer

else:
vs.release()

close all windows

cv2.destroyAllWindows()