Webcam Motion Detector

This python program will allow you to detect motion and also store the time interval of the motion.

**Requirement:**

1. Python3
2. OpenCV(libraries)
3. Pandas(libraries)

**Install Requirments :**Install [Python3](https://www.python.org/downloads/), install [Pandas](https://stackoverflow.com/questions/42907331/how-to-install-pandas-from-pip-on-windows-cmd) and [OpenCV](https://pypi.python.org/pypi/opencv-python" \t "_blank) libraries.

**Main Logic :** Videos can be treated as stack of pictures called frames. Here i am comparing different frames(pictures) to the first frame which should be static(No movements initially). We compare two images by comparing the intensity value of each pixels. In python we can do it easily as you can see in following code:

# Pyhton program to implement

# WebCam Motion Detector

# importing OpenCV, time and Pandas library

import cv2, time, pandas

# importing datetime class from datetime library

from datetime import datetime

# Assigning our static\_back to None

static\_back = None

# List when any moving object appear

motion\_list = [ None, None ]

# Time of movement

time = []

# Initializing DataFrame, one column is start

# time and other column is end time

df = pandas.DataFrame(columns = ["Start", "End"])

# Capturing video

video = cv2.VideoCapture(0)

# Infinite while loop to treat stack of image as video

while True:

# Reading frame(image) from video

check, frame = video.read()

# Initializing motion = 0(no motion)

motion = 0

# Converting color image to gray\_scale image

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

# Converting gray scale image to GaussianBlur

# so that change can be find easily

gray = cv2.GaussianBlur(gray, (21, 21), 0)

# In first iteration we assign the value

# of static\_back to our first frame

if static\_back is None:

static\_back = gray

continue

# Difference between static background

# and current frame(which is GaussianBlur)

diff\_frame = cv2.absdiff(static\_back, gray)

# If change in between static background and

# current frame is greater than 30 it will show white color(255)

thresh\_frame = cv2.threshold(diff\_frame, 30, 255, cv2.THRESH\_BINARY)[1]

thresh\_frame = cv2.dilate(thresh\_frame, None, iterations = 2)

# Finding contour of moving object

(\_, cnts, \_) = cv2.findContours(thresh\_frame.copy(),

cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

for contour in cnts:

if cv2.contourArea(contour) < 10000:

continue

motion = 1

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

# making green rectangle arround the moving object

cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 3)

# Appending status of motion

motion\_list.append(motion)

motion\_list = motion\_list[-2:]

# Appending Start time of motion

if motion\_list[-1] == 1 and motion\_list[-2] == 0:

time.append(datetime.now())

# Appending End time of motion

if motion\_list[-1] == 0 and motion\_list[-2] == 1:

time.append(datetime.now())

# Displaying image in gray\_scale

cv2.imshow("Gray Frame", gray)

# Displaying the difference in currentframe to

# the staticframe(very first\_frame)

cv2.imshow("Difference Frame", diff\_frame)

# Displaying the black and white image in which if

# intencity difference greater than 30 it will appear white

cv2.imshow("Threshold Frame", thresh\_frame)

# Displaying color frame with contour of motion of object

cv2.imshow("Color Frame", frame)

key = cv2.waitKey(1)

# if q entered whole process will stop

if key == ord('q'):

# if something is movingthen it append the end time of movement

if motion == 1:

time.append(datetime.now())

break

# Appending time of motion in DataFrame

for i in range(0, len(time), 2):

df = df.append({"Start":time[i], "End":time[i + 1]}, ignore\_index = True)

# Creating a csv file in which time of movements will be saved

df.to\_csv("Time\_of\_movements.csv")

video.release()

# Destroying all the windows

cv2.destroyAllWindows()

TERMINOLOGY:

**What are contours?**

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

* For better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection.
* Since OpenCV 3.2, **[findContours()](https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html" \l "ga17ed9f5d79ae97bd4c7cf18403e1689a" \o "Finds contours in a binary image. )** no longer modifies the source image but returns a modified image as the first of three return parameters.
* In OpenCV, finding contours is like finding white object from black background. So remember, object to be found should be white and background should be black.

# Contour Approximation Method

This is the third argument in **[cv.findContours](https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html" \l "ga17ed9f5d79ae97bd4c7cf18403e1689a" \o "Finds contours in a binary image. )** function. What does it denote actually?

Above, we told that contours are the boundaries of a shape with same intensity. It stores the (x,y) coordinates of the boundary of a shape. But does it store all the coordinates ? That is specified by this contour approximation method.

If you pass **[cv.CHAIN\_APPROX\_NONE](https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html" \l "gga4303f45752694956374734a03c54d5ffaf7d9a3582d021d5dadcb0e37201a62f8)**, all the boundary points are stored. But actually do we need all the points? For eg, you found the contour of a straight line. Do you need all the points on the line to represent that line? No, we need just two end points of that line. This is what **[cv.CHAIN\_APPROX\_SIMPLE](https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html" \l "gga4303f45752694956374734a03c54d5ffa5f2883048e654999209f88ba04c302f5)**does. It removes all redundant points and compresses the contour, thereby saving memory.

Below image of a rectangle demonstrate this technique. Just draw a circle on all the coordinates in the contour array (drawn in blue color). First image shows points I got with **[cv.CHAIN\_APPROX\_NONE](https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html" \l "gga4303f45752694956374734a03c54d5ffaf7d9a3582d021d5dadcb0e37201a62f8)** (734 points) and second image shows the one with **[cv.CHAIN\_APPROX\_SIMPLE](https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html" \l "gga4303f45752694956374734a03c54d5ffa5f2883048e654999209f88ba04c302f5)** (only 4 points). See, how much memory it saves!!!

**Simple Thresholding**

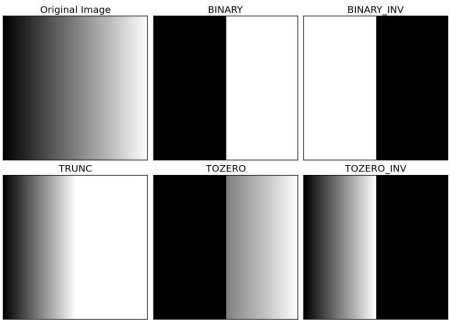
Here, the matter is straight forward. If pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black). The function used is **[cv.threshold](https://docs.opencv.org/3.4.0/d7/d1b/group__imgproc__misc.html" \l "gae8a4a146d1ca78c626a53577199e9c57" \o "Applies a fixed-level threshold to each array element. )**. First argument is the source image, which **should be a grayscale image**. Second argument is the threshold value which is used to classify the pixel values. Third argument is the maxVal which represents the value to be given if pixel value is more than (sometimes less than) the threshold value. OpenCV provides different styles of thresholding and it is decided by the fourth parameter of the function. Different types are:

* [**cv.THRESH\_BINARY**](https://docs.opencv.org/3.4.0/d7/d1b/group__imgproc__misc.html#ggaa9e58d2860d4afa658ef70a9b1115576a147222a96556ebc1d948b372bcd7ac59)
* [**cv.THRESH\_BINARY\_INV**](https://docs.opencv.org/3.4.0/d7/d1b/group__imgproc__misc.html#ggaa9e58d2860d4afa658ef70a9b1115576a19120b1a11d8067576cc24f4d2f03754)
* [**cv.THRESH\_TRUNC**](https://docs.opencv.org/3.4.0/d7/d1b/group__imgproc__misc.html#ggaa9e58d2860d4afa658ef70a9b1115576ac7e89a5e95490116e7d2082b3096b2b8)
* [**cv.THRESH\_TOZERO**](https://docs.opencv.org/3.4.0/d7/d1b/group__imgproc__misc.html#ggaa9e58d2860d4afa658ef70a9b1115576a0e50a338a4b711a8c48f06a6b105dd98)
* [**cv.THRESH\_TOZERO\_INV**](https://docs.opencv.org/3.4.0/d7/d1b/group__imgproc__misc.html#ggaa9e58d2860d4afa658ef70a9b1115576a47518a30aae90d799035bdcf0bb39a50)

Documentation clearly explain what each type is meant for. Please check out the documentation.

Two outputs are obtained. First one is a **retval** which will be explained later. Second output is our **thresholded image**.

:



**Adaptive Thresholding**

In the previous section, we used a global value as threshold value. But it may not be good in all the conditions where image has different lighting conditions in different areas. In that case, we go for adaptive thresholding. In this, the algorithm calculate the threshold for a small regions of the image. So we get different thresholds for different regions of the same image and it gives us better results for images with varying illumination.

It has three ‘special’ input params and only one output argument.

**Adaptive Method** - It decides how thresholding value is calculated.

* [**cv.ADAPTIVE\_THRESH\_MEAN\_C**](https://docs.opencv.org/3.4.0/d7/d1b/group__imgproc__misc.html#ggaa42a3e6ef26247da787bf34030ed772cad0c5199ae8637a6b195062fea4789fa9) : threshold value is the mean of neighbourhood area.
* [**cv.ADAPTIVE\_THRESH\_GAUSSIAN\_C**](https://docs.opencv.org/3.4.0/d7/d1b/group__imgproc__misc.html#ggaa42a3e6ef26247da787bf34030ed772caf262a01e7a3f112bbab4e8d8e28182dd) : threshold value is the weighted sum of neighbourhood values where weights are a gaussian window.

**Block Size** - It decides the size of neighbourhood area.

**C** - It is just a constant which is subtracted from the mean or weighted mean calculated.

Below piece of code compares global thresholding and adaptive thresholding for an image with varying illumination:

