**IP Task – 2**

**Group – 3**

**Aim:** Video Stabilization

**Procedure:**

1. Using vidgear (Stabilizer) library:

* Vidgear is a high performance frame work that provides all types of video processing techniques for real time stabilization in python.
* Firstly, import ‘vidgear’ library with module named ‘gears’ to implement ‘Stabilizer’ function.
* Then capture video frames from camera and smooth them to stabilize then frames. Also can crop and zoom the frame so as to avoid irregular patterns at the corners.
* Read the frames inside the for loop and then stabilize this frame with the help of ‘Stabilize().stabilize’ function.
* Display the stabilized output frame.

**Code:**

# Importing libraries

import cv2

from vidgear.gears.stabilizer import Stabilizer

# Capturing video

cap = cv2.VideoCapture("v1.mp4")

# Capturing video frames from camera on-device

# Initiating Stabilizer object with parameters of our choice

stab = Stabilizer(smoothing\_radius=30, crop\_n\_zoom=True)

while(1):

# Reading current frames

ret, frame = cap.read()

if not ret:

break

# Applying stabilizer processing to the current frame

stabilized\_frame = stab.stabilize(frame)

if stabilized\_frame is None:

continue

# Display stabilized frame

stabilized\_frame = cv2.resize(stabilized\_frame, (500, 300))

frame = cv2.resize(frame, (500, 300))

cv2.imshow("Stabilized video", stabilized\_frame)

cv2.imshow("Input video", frame)

key = cv2.waitKey(5)

if key == ord('q'):

break

# Exit control statements

cv2.destroyAllWindows()

stab.clean()

cap.release()

1. Using vidgear (Videogear) library:

* Vidgear is a high performance frame work that provides all types of video processing techniques for real time stabilization in python.
* Firstly, import ‘vidgear’ library with module named ‘gears’ to implement ‘VideoGear’ function.
* The function ‘VideoGear has basically two parameters mostly used.
  + source: When source = 0 then we’ll be able to optimize video with real time stabilization features. And if source = “video\_path” then external video source will be choosen for stabilization.
  + stabilize: The default value for this parameter is ‘False’. If the user wants to utilise stabilization features then this parameter should be set to ‘True’.
* Read stabilized frame and display the output inside the while loop.

**Code:**

# Importing libraries

import cv2

from vidgear.gears import VideoGear

# Capturing and stabilizing the frame

# Put stabilize = True to activate stabilization features

cap = VideoGear(source= "v1.mp4", stabilize=True).start()

while (1):

# Reading stabilized frame

frame\_final = cap.read()

# To avoid error returned value

if frame\_final is None:

break

# Display the stabilized frame

cv2.imshow("Stabilized video", frame\_final)

key = cv2.waitKey(1)

if key == 32:

break

# Exit control statement

cap.stop()

cv2.destroyAllWindows()

1. Using vidstab library:

* Vidstab module contains a single class VidStab used for video stabilization. It’s algorithm is basically depends on Euclidean transformation and sliding moving average methods.
* Firstly import the necessary libraries and initialise stabilizer and video reader.
* After reading the frame make use of stabilize\_frame() function.
* stabilize\_frame(): It mainly has three major parameters.
  + input\_frame: pass on the current frame.
  + layer\_func: It has two choice.
    - layer\_blend: It uses blending techniques for mixtures of newly occurring frames ( has blurring effect at borders ).
    - layer\_overlay: It overlays newly occurring frames on over other ( doesn’t has blurring effect at borders ).
  + border\_size: provide size of border of stabilized frame.
* Display the stabilized output generated from stabilized\_frame() function.

**Code:**

# Importing libraries

import cv2

from vidstab import VidStab, layer\_blend

# Initialize stabilizer and video reader

stabilizer = VidStab()

vidcap = cv2.VideoCapture("v1.mp4")

while True:

# Reading current frame

ret, frame = vidcap.read()

# Stabilization process

# Pass frame to stabilizer even if frame is None

stabilized\_frame = stabilizer.stabilize\_frame(input\_frame=frame,

layer\_func=layer\_blend,

border\_size=30)

# If stabilized\_frame is None then there are no frames left to process

if stabilized\_frame is None:

break

# Display stabilized output

cv2.imshow('Stabilized output', stabilized\_frame)

key = cv2.waitKey(5)

if key == ord('q'):

break

vidcap.release()

cv2.destroyAllWindows()

1. Using Eucledian transformation method:

* Read the previous (first) frame and then convert the frame from BGR to Gray scale image.
* To find the transformations we need good features to track and then smooth them. So, ‘cv2.goodFeaturesToTrack()’ function is used to track features points into the next frame.
* Tracking good featured points from previous frame to the next frame is done with Lucas Kanade optical flow method i.e. ‘calcOpticalFlowPyrLK()’.
* Finally eucledian rigid transformation values for x, y and angle are calculated with the help of previous and current points. For this ‘estimateAffinePartial2D()’ function is used.
* After getting the transformation points in the current frame, calculate the trajectory path for this transformation.
* Then smoothing the trajectory with the help of sliding average technique and adding it to original transformation.
* Display the stabilized output frame after affining the smoothed frame.

**Code:**

# Importing libraries

import numpy as np

import cv2

cap = cv2.VideoCapture("v1.mp4")

# cap = cv2.VideoCapture("v2.mp4")

def slidingAverage(curve, radius):

winSize = 2 \* radius + 1

f = np.ones(winSize) / winSize

# padding

padding = np.lib.pad(curve, (radius, radius), 'edge')

# Applying convolution

smoothed = np.convolve(padding, f, mode='same')

# Removing padding

smoothed = smoothed[radius:-radius]

return smoothed

def smoothingTrajectory(trajectory):

smoothed\_trajectory = np.copy(trajectory)

for i in range(3):

smoothed\_trajectory[:, i] = slidingAverage(trajectory[:, i], radius=50)

return smoothed\_trajectory

# Accessing number of frames

noFrames = int(cap.get(cv2.CAP\_PROP\_FRAME\_COUNT))

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

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

# Reading the prevFrame(first frame)

retPrev, prevFrame = cap.read()

prev\_gray = cv2.cvtColor(prevFrame, cv2.COLOR\_BGR2GRAY)

trnsfm = np.zeros((noFrames - 1, 3), np.float32)

for i in range(noFrames - 2):

prevPts = cv2.goodFeaturesToTrack(prev\_gray, maxCorners=200, qualityLevel=0.01, minDistance=30, blockSize=3)

retCurr, currFrame = cap.read()

if not retCurr:

break

currGray = cv2.cvtColor(currFrame, cv2.COLOR\_BGR2GRAY)

# Tracking the featured points

currPts, found, error = cv2.calcOpticalFlowPyrLK(prev\_gray, currGray, prevPts, None)

idx = np.where(found == 1)[0]

prevPts = prevPts[idx]

currPts = currPts[idx]

assert prevPts.shape == currPts.shape

# rigid transformation

rigTrnsfm = cv2.estimateAffinePartial2D(prevPts, currPts)[0]

x = rigTrnsfm[0, 2]

y = rigTrnsfm[1, 2]

angle = np.arctan2(rigTrnsfm[1, 0], rigTrnsfm[0, 0])

trnsfm[i] = [x, y, angle]

prev\_gray = currGray

# Cumulative sum of trnsfm for x, y and angle

trajectory = np.cumsum(trnsfm, axis=0)

smoothed\_trajectory = smoothingTrajectory(trajectory)

diff = smoothed\_trajectory - trajectory

trnsfmSmooth = trnsfm + diff

# Resetting to first frame

cap.set(cv2.CAP\_PROP\_POS\_FRAMES, 0)

for i in range(noFrames - 2):

# Reading nxtFrame

retNxt, nxtFrame = cap.read()

if not retNxt:

break

# Accepting values of x, y and angle

x = trnsfmSmooth[i, 0]

y = trnsfmSmooth[i, 1]

angle = trnsfmSmooth[i, 2]

# Filling transformation values

rigTrnsfm = np.zeros((2, 3), np.float32)

rigTrnsfm[0, 0] = np.cos(angle)

rigTrnsfm[0, 1] = -np.sin(angle)

rigTrnsfm[1, 0] = np.sin(angle)

rigTrnsfm[1, 1] = np.cos(angle)

rigTrnsfm[0, 2] = x

rigTrnsfm[1, 2] = y

# wrapAffine function operation on nxtFrame

fnlFrame = cv2.warpAffine(nxtFrame, rigTrnsfm, (wdth, ht))

# Display stabilized output

cv2.imshow("Output video", fnlFrame)

cv2.imshow("Input video", nxtFrame)

if cv2.waitKey(50) == ord("q"):

break

cap.release()

cv2.destroyAllWindows()

**Conclusion:**

1. VidGear Sabilizer method gave better results and also provided functionality for crop and zoom. Also provided facility for smoothing radius.
2. VidGear method provided the best stabilization output for real time as well as input videos.
3. Vidstab approach gave true results for low filtered videos and real time stabilization as well.
4. Eucledian transformation method provided accurate results for input videos but the process was too slow and reliable on the number of frames per video.