Computer Vision Assignment#0

# Video 🡨🡪 Images:

## Write a program to convert a given video to its constituent images. Your output should be in a specified folder.

‘video2images.py’ is created for this task with command line parameters.

Usage: video2images.py [-h] -i INPUT [-o OUTPUT]

-h is help

-i for input

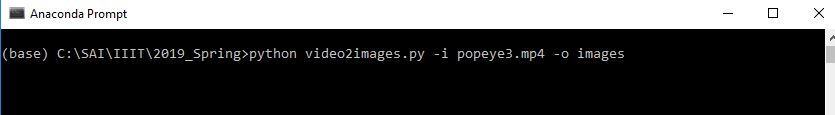
-o for output

INPUT Path to video file

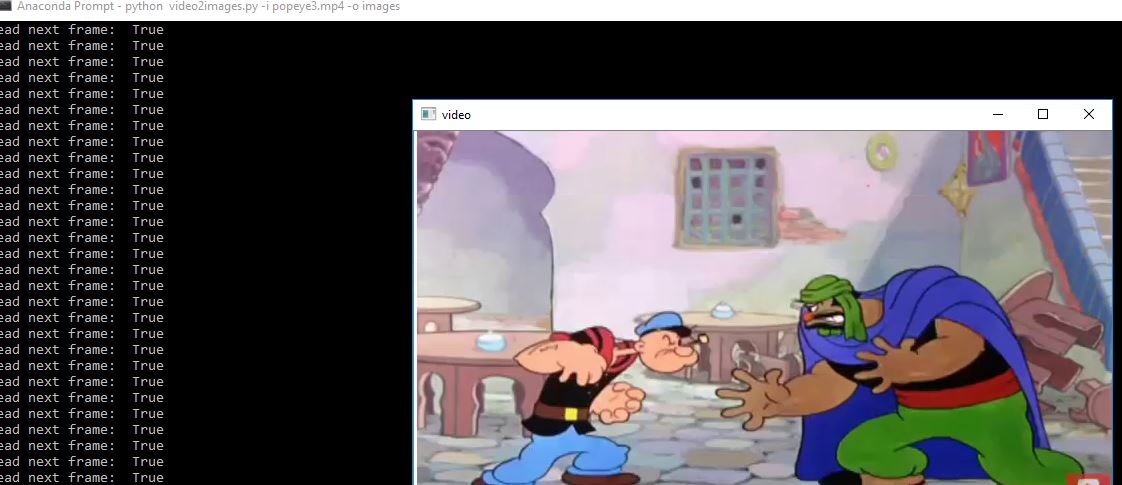
OUTPUT Specify output folder

Example1: python video2images.py -i popeye3.mp4 -o images

Run the python script from the command prompt



Observe intermediate steps as each frame is saved to disk while the video plays



video2images.py

import argparse

import sys

import cv2

import os

def video2images(inputvideofile,outputimagesfolder):

vidcap = cv2.VideoCapture(inputvideofile)

returnvalue,image = vidcap.read() #returns true/false and an image

count = 0

while returnvalue:

filename=os.path.join(outputimagesfolder,"image%d.jpg" % count)

cv2.imwrite(filename, image) # save frame as JPEG/png file

returnvalue,image = vidcap.read()

print('Read next frame: ', returnvalue)

cv2.imshow('video',image)

if (cv2.waitKey(1) & 0xFF) == ord('q'): # Hit `q` to exit

break

count += 1

cv2.destroyAllWindows()

print('\nCompleted Saving video frames to',outputimagesfolder)

def main(argv):

ap = argparse.ArgumentParser()

ap.add\_argument("-i", "--input", required=True, help="Input video file ")

ap.add\_argument("-o", "--output", required=False, default='.', help="Output images directory")

args = vars(ap.parse\_args())

inputvideofile = args['input']

outputimagesfolder = args['output']

# inputvideofile = 'popeye3.mp4'

# inputvideofile =r'C:\SAI\IIIT\2019\_Spring\cv0\durga\_new.mp4'

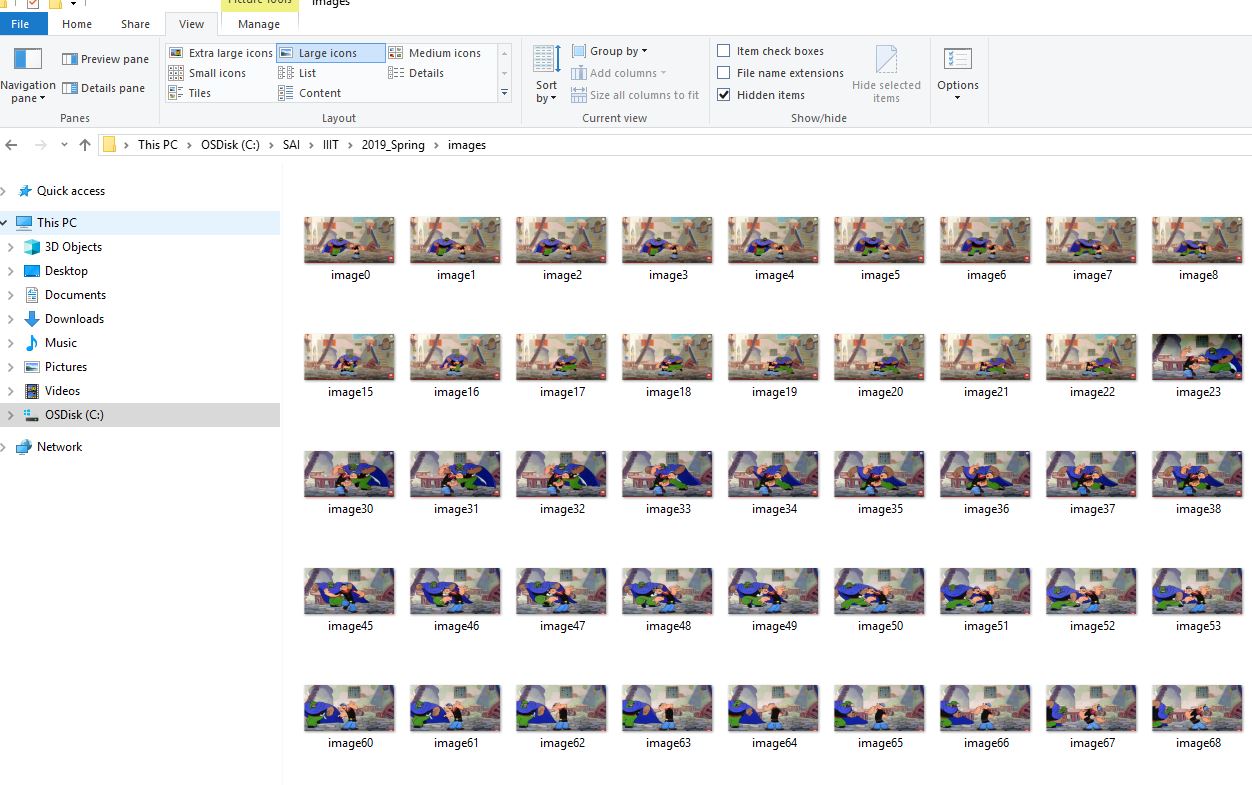
# outputimagesfolder=r' C:\SAI\IIIT\2019\_Spring\cv0\Images

video2images(inputvideofile,outputimagesfolder)

if \_\_name\_\_ == "\_\_main\_\_":

main(sys.argv[1:])

Output images generated in the ‘images’ folder



## Write another program that will merge a set of images in a folder into a single video. You should be able to control the frame rate in the video that is created.

‘images2video.py’ is created for this task with command line parameters.

Usage: images2video.py [-h] -i INPUT [-o OUTPUT] [-f FRAME\_RATE]

-h is help

-i for input

-o for output

-f for frame rate

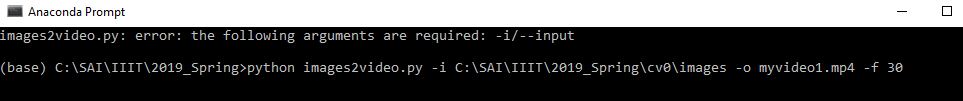
INPUT Path to images folder

OUTPUT Path to output file

FRAME\_RATE Frame rate of the output video

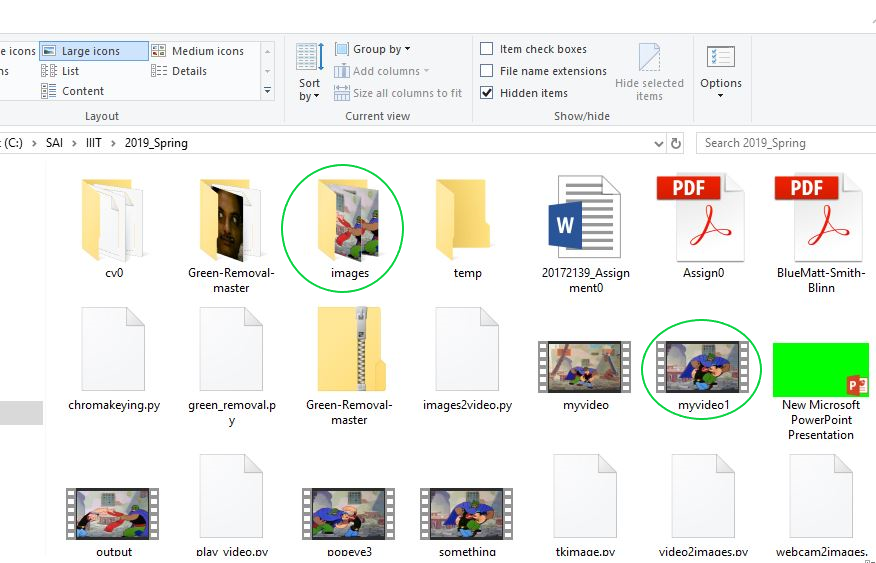
Example1: python images2video.py -i C:\cv0\images -o myvideo1.mp4 -f 30

Run the python script from the command prompt



Observe intermediate steps as each frame is read from the disk and displayed





images2video.py

import argparse

import sys, os

import cv2

import re

'''

To maintain images sequence, sort the file names numerically

#https://stackoverflow.com/questions/12093940/reading-files-in-a-particular-order-in-python

'''

def numericalSort(value):

numbers = re.compile(r'(\d+)')

parts = numbers.split(value)

parts[1::2] = map(int, parts[1::2])

return parts

def images2video(inputimagesfolder,outputvideofile,frame\_rate):

included\_extensions = ['jpg','jpeg', 'bmp', 'png', 'gif'] #Filter folder for images

file\_names = [fn for fn in os.listdir(inputimagesfolder)

if any(fn.endswith(ext) for ext in included\_extensions)]

images\_temp=[]

images=[]

#Get image paths

for file in file\_names:

image\_path=os.path.join(inputimagesfolder,file)

images\_temp.append(image\_path)

#Sort the paths according to numerical sequence in which they are generated

for infile in sorted(images\_temp, key=numericalSort):

images.append(infile)

image\_path = images[0]

frame = cv2.imread(image\_path)

cv2.imshow('video',frame)

height, width, bands = frame.shape # Get dimension of the images

#Add these images to a video file

fourcc = cv2.VideoWriter\_fourcc(\*'mp4v')

video = cv2.VideoWriter(outputvideofile, fourcc, frame\_rate, (width, height))

for image in images:

frame = cv2.imread(image)

video.write(frame) # Append frame to video

cv2.imshow('images2video',frame)

if (cv2.waitKey(1) & 0xFF) == ord('q'): # Hit `q` to exit

break

video.release()

cv2.destroyAllWindows()

print('Completed converting images to video')

def main(argv):

ap = argparse.ArgumentParser()

ap.add\_argument("-i", "--input", required=True, help="Input Images directory ")

ap.add\_argument("-o", "--output", required=False, default='somevideo.mp4', help="output video file")

ap.add\_argument("-f", "--frame\_rate", required=False, default=24.0, help="Frame rate of the video")

args = vars(ap.parse\_args())

inputimagesfolder = args['input']

outputvideofile = args['output']

frame\_rate=int(args['frame\_rate'])

print(inputimagesfolder)

print(outputvideofile)

images2video(inputimagesfolder,outputvideofile,frame\_rate)

# images2video(r'C:\SAI\IIIT\2019\_Spring\images','something.mp4',24)

if \_\_name\_\_ == "\_\_main\_\_":

main(sys.argv[1:])

### Observations:

Images read from the disk shall be in a **sequence**.

The cyan section of the script provides below output, which is not in the time/numerical order in which the files were supposed to be in the video.

image6.jpg

image60.jpg

image600.jpg

image601.jpg

image602.jpg

image603.jpg

image604.jpg

image605.jpg

image606.jpg

image607.jpg

image608.jpg

image609.jpg

image61.jpg

image610.jpg

image611.jpg

image612.jpg

image613.jpg

image614.jpg

image615.jpg

Images are to be sorted in the correct numerical/time order. This is achieved with the section of the script in green.

image600.jpg

image601.jpg

image602.jpg

image603.jpg

image604.jpg

image605.jpg

image606.jpg

image607.jpg

image608.jpg

image609.jpg

image610.jpg

image611.jpg

image612.jpg

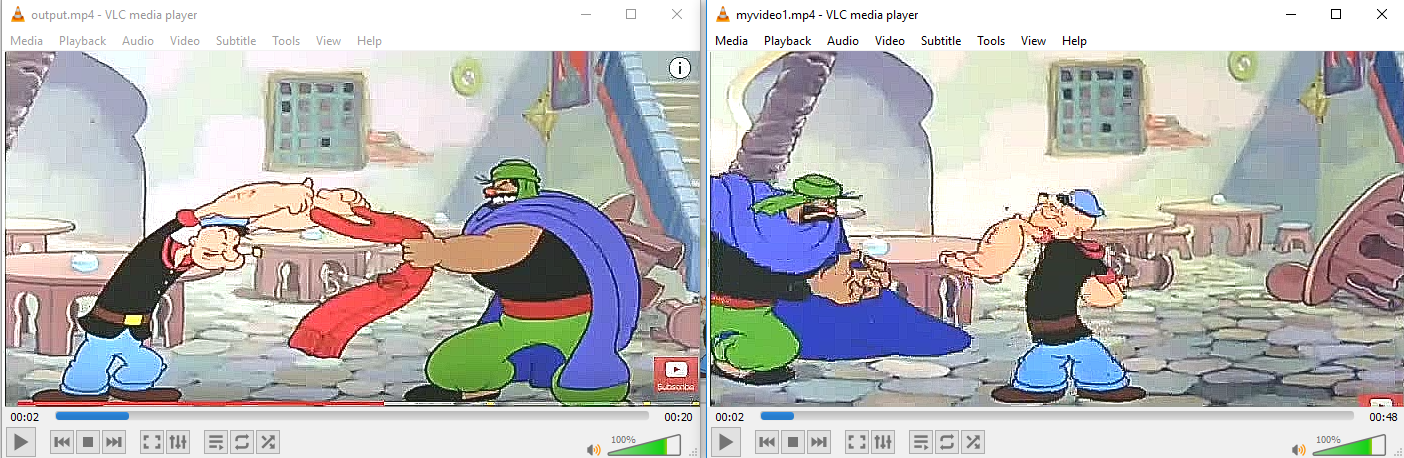
image613.jpg

image614.jpg

image615.jpg

In the image below at 0.02 seconds, notice the snapshot differences for incorrect order (cyan)

on the left. After sorting the order to represent proper sequence (green), correct image obtained in the video at right.



### **Frame rates** tried at 24, 40 and 60 fps. Resulted in videos with below properties.

* Smooth transition for lower number and vice-versa
* Higher fps number has lower file size
* Track Duration : 19.28s for 60 fps
* Track Duration : 28.93s for 40 fps
* Track Duration : 48s for 24 fps

# Capturing Images:

## Learn how to capture frames from a webcam connected to your computer and save them as images in a folder. You may use either the built-in camera of your laptop or an external one connected through USB. You should also be able to display the frames (the video) on the screen while capturing.

‘webcam2images.py’ is created for this task with command line parameters.

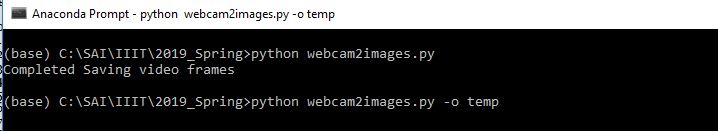
Usage: webcam2images.py -o [OUTPUT]

-o for output

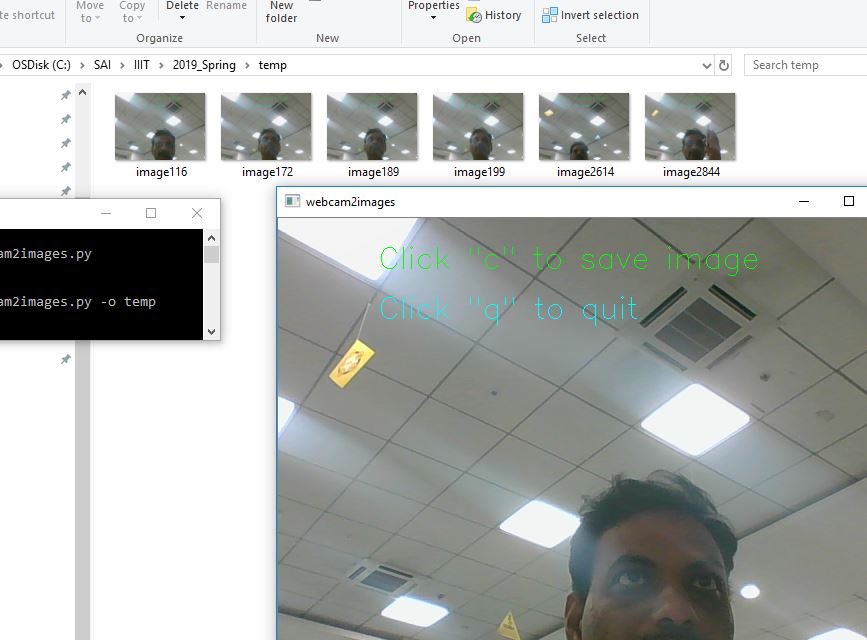
OUTPUT Specify output folder to save images

Example1: python webcam2images.py -o temp

Run the python script from the command prompt



Click ‘C’ to save a snapshot to disk.



webcam2images.py

import argparse

import sys

import os

import cv2

import time

def getframerate(video):

begin = time.time()

num\_frames=100

for i in range(0, num\_frames) :

ret, frame = video.read()

end = time.time()

seconds = end – begin

frame\_rate = num\_frames / seconds

return frame\_rate

def webcam2images(outputimagesfolder):

vidcap = cv2.VideoCapture(0)

frame\_rate=getframerate(vidcap)# Get frame rate

returnvalue,image = vidcap.read() #returns true/false and an image

count=0

font = cv2.FONT\_HERSHEY\_SIMPLEX

text\_BL = (100,50)

fontScale = 1

fontColor = (0, 255, 0)

lineType = 1

text\_BL2 = (100,100)

fontColor2 = (255, 255, 0)

text\_BL3 = (100,0)

fontColor3 = (255, 0, 255)

while returnvalue:

filename=os.path.join(outputimagesfolder,"image%d.jpg" % count)

returnvalue,image = vidcap.read()

cv2.putText(image,'Click "c" to save image ', text\_BL,font,fontScale,fontColor,lineType)

cv2.putText(image,'Click "q" to quit ', text\_BL2,font,fontScale,fontColor2,lineType)

cv2.putText(image,str(frame\_rate), text\_BL3,font ,fontScale,fontColor3,lineType)

cv2.imshow('webcam2images',image)

if (cv2.waitKey(1) & 0xFF) == ord('q'): # Hit `q` to exit

break

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

cv2.imwrite(filename, image)

count += 1

cv2.destroyAllWindows()

print('Completed Saving video frames')

def main(argv):

ap = argparse.ArgumentParser()

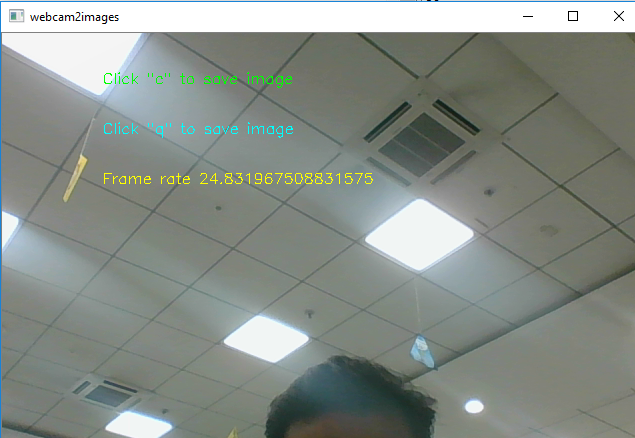
ap.add\_argument("-o", "--output", required=False, default='.', help="output images directory")

args = vars(ap.parse\_args())

outputimagesfolder = args['output']

webcam2images(outputimagesfolder)

Frame rates are computed in green section of the script



# Chroma Keying:

## Create an interesting composite of two videos using this technique, possibly with one video including yourselves.

Read about the technique of chroma keying. Following are a few good starting points:

* Introduction: http://en.wikipedia.org/wiki/Chroma key
* Alvy Ray Smith and James F Blinn, "Blue Screen Matting", SIGGRAPH'96.

‘chromakeying.py’ is created for this task with command line parameters.

Usage: chromakeying.py [-h] -s SOURCE -t TARGET [-o OUTPUT]

-h is help

-s for source

-t for target

-o for output

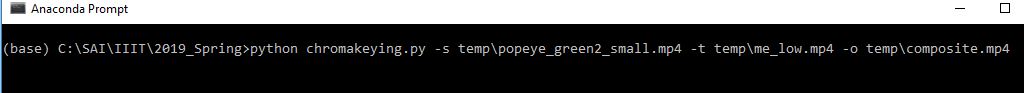
SOURCE Path to green screen video file

TARGET Path to background video file

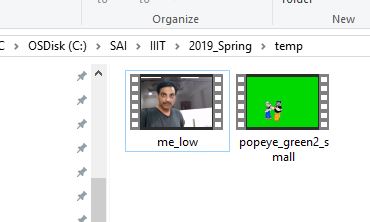
OUTPUT Specify output composite video name

Example1: python chromakeying.py -s temp\popeye\_green2\_small.mp4 -t temp\me\_low.mp4 -o temp\composite.mp4

Run the python script from the command prompt



Source and Target files before running the script



chromakeying.py

import argparse

import os,sys

import cv2

import re

import numpy as np

"""

https://stackoverflow.com/questions/36817133/identifying-the-range-of-a-color-in-hsv-using-opencv

Choose any range around your target value. for example, yellow has hue val 60 degrees.

(Use https://www.tydac.ch/color/)

So your hue range might be from 60/2 - 10 to 60/2 + 10 OR

from 60/2-5 to 60/2+5 depends on how far you want to go away from perfect yellow.

For HSV, Hue range is [0,179], Saturation range is [0,255] and Value range is [0,255]. This is true for OpenCV,could be different for GIMP

https://www.learnopencv.com/color-spaces-in-opencv-cpp-python/

#To maintain images sequence, sort the file names numerically

#https://stackoverflow.com/questions/12093940/reading-files-in-a-particular-order-in-python

'''

def numericalSort(value):

numbers = re.compile(r'(\d+)')

parts = numbers.split(value)

parts[1::2] = map(int, parts[1::2])

return parts

def video2images(inputvideofile,outputimagesfolder):

vidcap = cv2.VideoCapture(inputvideofile)

returnvalue,image = vidcap.read() #returns true/fals and an image

count = 0

while returnvalue:

filename=os.path.join(outputimagesfolder,"image%d.jpg" % count)

cv2.imwrite(filename, image) # save frame as JPEG/png file

returnvalue,image = vidcap.read()

# print('Read next frame: ', returnvalue)

# cv2.imshow('video',image)

# if (cv2.waitKey(1) & 0xFF) == ord('q'): # Hit `q` to exit

# break

count += 1

cv2.destroyAllWindows()

print('\nCompleted Saving video frames to ',outputimagesfolder)

def createdirectory(dirName):

# Create target Directory if don't exist

if not os.path.exists(dirName):

os.mkdir(dirName)

print("\nDirectory " , dirName , " Created ")

else:

print("\nDirectory " , dirName , " Using existing directory")

def generateimages(videofile):

head, tail = os.path.split(videofile)

temp=tail.split('.')[0]

outputimagesfolder=os.path.join(head,temp)

createdirectory(outputimagesfolder)

print('Created ', outputimagesfolder)

video2images(videofile,outputimagesfolder)

#Filter folder for images

included\_extensions = ['jpg','jpeg', 'bmp', 'png', 'gif']

file\_names = [fn for fn in os.listdir(outputimagesfolder)

if any(fn.endswith(ext) for ext in included\_extensions)]

# print(file\_names)

images\_temp=[]

images=[]

#Get image paths

for file in file\_names:

image\_path=os.path.join(outputimagesfolder,file)

images\_temp.append(image\_path)

#Sorth the paths according to numnerical sequence in which they are generated

for infile in sorted(images\_temp, key=numericalSort):

images.append(infile)

return images

def chromakey(green\_video,target\_video,output\_video):

green\_images=generateimages(green\_video)

target\_images=generateimages(target\_video)

#Make sure that the number of frames in source and target are same

target\_images=target\_images[:len(green\_images)]

lower\_green = np.array([50])

upper\_green = np.array([70])

#Chroma Keyed video - widht and height as that of source/target video

frame = cv2.imread(green\_images[0])

height, width, bands = frame.shape

fourcc = cv2.VideoWriter\_fourcc(\*'mp4v')

video = cv2.VideoWriter(output\_video, fourcc, 24, (width, height))

for green\_image\_path,target\_frame\_path in zip(green\_images,target\_images):

green\_frame = cv2.imread(green\_image\_path)

target\_frame = cv2.imread(target\_frame\_path)

# BGR to HSV

hsv = cv2.cvtColor(green\_frame, cv2.COLOR\_BGR2HSV)

# HSV bands

h = hsv[:,:,0]

#Threshold the HSV image to get only green color,#hue\_mask\_source is 255 where color between #lower\_green & upper\_green [background],otherwise it is zero [foreground]

hue\_mask\_source = cv2.inRange(h, lower\_green, upper\_green)

#Generate Panchromatic image by inverting the values of FG and BG of hue\_mask\_source

#Background is 0 ,Foreground is 255

hue\_masked\_source\_image=np.where(hue\_mask\_source != 0,0,255)

# From original green screen image, identify the fg identified by using hue\_mask\_source as cookie cutter

masked\_image = np.copy(green\_frame)

masked\_image[hue\_masked\_source\_image == 0] = [0, 0, 0]

#Make a hole in the target image to the extent of source FG boundary

output\_video\_frame = target\_frame.copy()

output\_video\_frame[hue\_masked\_source\_image != 0] = [0, 0, 0]

#Add Source FG and background image

output\_video\_frame = output\_video\_frame + masked\_image

video.write(output\_video\_frame) # Append frame to video

cv2.imshow('Chroma Key',output\_video\_frame)

if (cv2.waitKey(1) & 0xFF) == ord('q'): # Hit `q` to exit

break

video.release()

cv2.destroyAllWindows()

print('\nCompleted Chorma keying')

def main(argv):

ap = argparse.ArgumentParser()

ap.add\_argument("-s", "--source", required=True, help="Source green screen background videofile ")

ap.add\_argument("-t", "--target", required=True, help="Target video file ")

ap.add\_argument("-o", "--output", required=False, default='merged.mp4', help="matted video")

args = vars(ap.parse\_args())

green\_video = args['source']

target\_video = args['target']

output\_video = args['output']

# green\_video=os.path.join(r'C:\SAI\IIIT\cv0','durga\_new\_small2.mp4')

# target\_video=os.path.join(r'C:\SAI\IIIT\cv0','popeye3\_small.mp4')

# output\_video=os.path.join(r'C:\SAI\IIIT\cv0','durga\_with\_popeye2.mp4')

# green\_video=os.path.join(r'C:\SAI\IIIT\cv0','popeye\_green2\_small.mp4')

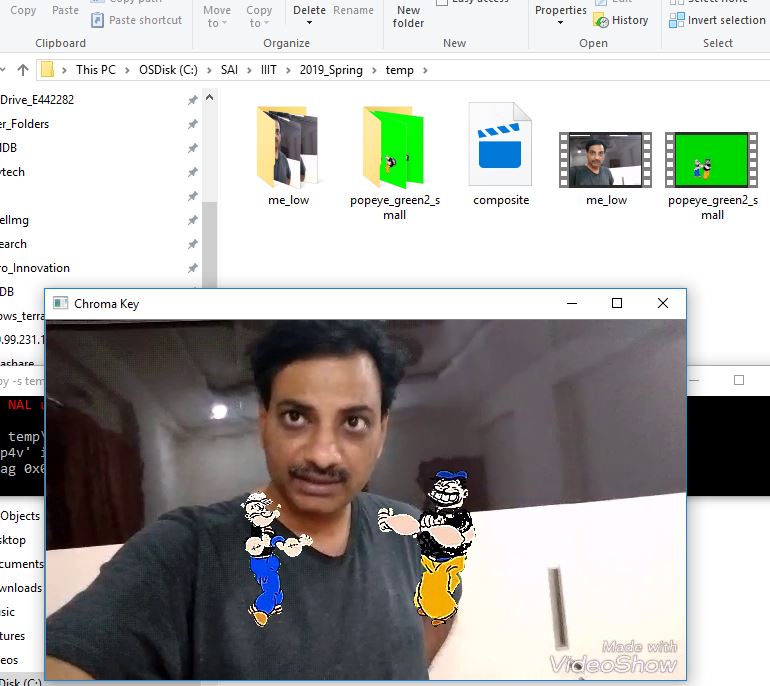
# target\_video=os.path.join(r'C:\SAI\IIIT\cv0','me\_low.mp4')

# output\_video=os.path.join(r'C:\SAI\IIIT\cv0','durga\_with\_popeye22.mp4')

print('Started Chroma Keying..')

chromakey(green\_video,target\_video,output\_video)

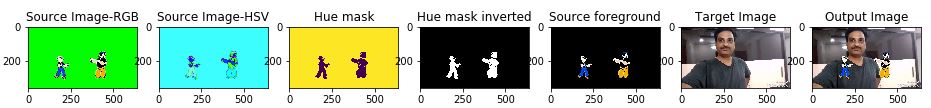
Observe intermediate files, Source, Target and composite video files after running the chromakeying script.

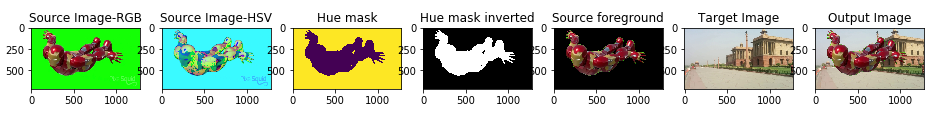


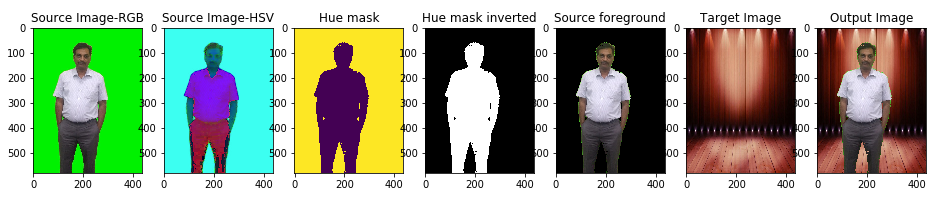
### Method

Brief outline of the method used and corresponding snapshots:

1. Filter the green screen image – create mask by eliminating green background. Convert RGB to HSV and used hue to do this.
2. Inver the hue mask to separate foreground and background
3. Use the foreground mask to extract roi from the original green screen
4. Use the same mask to remove values in the target video
5. Merge target video and foreground extracted in step#3





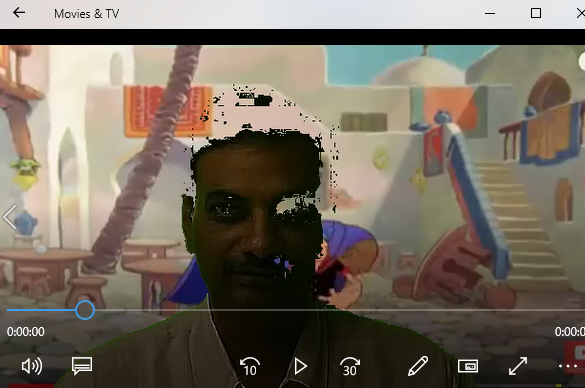


### Observations:

Artifacts observed

* Potential causes - Lighting, shadows, variation of color shades
* Majority of issues are occurring close to boundary of the object of interest
* Dark shades are causing issues













Improved result in some cases using HSV or RGB

Noticed that green with HSV computations is replaced by RGB script as below,provided better result in some cases.

rgb = cv2.cvtColor(green\_frame, cv2.COLOR\_BGR2RGB)

blues = rgb[:,:,0]

greens = rgb[:,:,1]

reds = rgb[:,:,2]

hue\_masked\_source\_image=((greens<220)|(reds>=greens)|(blues>=greens))\*255





But introduced green halo for other scenarios.

