Computer Vision (Fall 2019)

Assignment 1: Invisible Cloak Using Green Screen Replacement

Guidelines:

- Submit all of your code and results in a single zip file with name FirstName_RollNumber_01.zip
- Submit single zip file containing
  a) code b) results c) readme d) report
  There should be **Readme.txt** explaining how to run your code
  There should be **Report.pdf** detailing your experience and highlighting any interesting result.
- Email instructor or TA if there are any questions. You cannot look at others code or use others code, including from internet, however you can discuss with each other.
- Please use relative addresses for files/folders you create inside code. No absolute addresses.
- Please follow all the naming conventions mentioned in each task
- Each task must have a separate notebook. Name them as task0, task1 and task2
- Make folders Task0, Task1 and Task2. Place code/results of each task in respective folder
- Do not submit the original videos/images provided to you. Just provide the results
- Not following instructions will result in severe deduction of marks
- Deadline for Assignment is Monday 23rd September 2019 before 11:59 pm

Overview:

Green screens are widely used in movies, games and other graphical applications to make selected areas of video/image invisible. The area is then replaced with cool effects by, for example, changing backgrounds, adding magical spells and adding artificially created worlds in a scene. You can Google behind-the-scenes of popular sci-fi type of movies to see the real-world applications of this technique. In this assignment we are going to recreate the invisible cloak effect from the Hollywood blockbuster Harry Potter.

Task 0: Loading and Saving Video in OpenCV

For this task don't submit the resulting video, only submit the code. In this task you will load a video file and then save every 10th frame of the video to a new video file. Below is the step by step description of how to perform this task.

Import libraries:

```
import cv2
import numpy as np
```

To load video using OpenCV, we use cv2.VideoCapture() which returns an object for the video we provide in arguments:

```
cap = cv2.VideoCapture('green_hand.mp4')
```

To loop over video frame by frame run the following:

```
#loop over frames
while(cap.isOpened()):

    #each call to cap.read() returns the next frame
    val, frame = cap.read()
```

In the above script we are essentially just reading each frame and saving it in variable 'frame'. We are not doing any processing yet. The frame is a numpy array, and you can perform any operation on it that numpy supports on its arrays, for example checking its shape.

Now we will save each 10th frame of the video in a list and then convert that list to a numpy array. This numpy array has now frames which can be saved as a new video.

```
i = 0
new_video = []

#loop over frames
while(cap.isOpened()):

    print('frames processed: ', i)

    if frame is None:
        print('Frame returned is None')
        break

    #each call to cap.read() returns the next frame
    val, frame = cap.read()

    if (i%10 == 0):
        new_video.append(frame)

    i=i+1

#convert the list to numpy array
new_video = np.array(new_video)
```

Now we can save this ‘new_video’ numpy array as a video file. Run the following code for it:

```
#saving numpy array to a video file using OpenCV
writer = cv2.VideoWriter('new_video.avi', cv2.VideoWriter_fourcc(*'PIM1'), 30, (1280, 720), True)

for x in new_video:
    writer.write(x)
```

In the above snippet, we are defining a video writer which is a module provided by OpenCV for writing video files. In the above code, video is saved as ‘new_video.avi’. Frame rate is kept to 30 fps. 1280 and 720 are video’s width and height respectively.

The overall code thus is:

```
#load libraries
import cv2
import numpy as np

#load video file
cap = cv2.VideoCapture('hamza.mp4')

i = 0
new_video = []

#loop over frames
while(cap.isOpened()):

    print('frames processed: ', i)

    if frame is None:
        print('Frame returned is None')
        break

    #each call to cap.read() return the next frame
    val, frame = cap.read()

    #append every 10th frame
    if (i%10 == 0):
        new_video.append(frame)

    i=i+1
```

```
#convert the list to numpy array
new_video = np.array(new_video)

#saving numpy array to a video file using OpenCV
writer = cv2.VideoWriter('new_video.avi', cv2.VideoWriter_fourcc(*'PIM1'), 30, (1280, 720), True)
for x in new_video:
   writer.write(x)
```

Task 1:

For this task don’t submit the resulting video, only submit the code and a single frame showing you have added the smiley. In Task 0 you learned to read a video file frame by frame and then save the video. In this task you are required to add a smiley shape to first 100 frames of **‘green_hand.mp4’**. You can add the smiley anywhere on the frames and then save the resulting video as **‘green_hand_smiley.avi’**. An example of a smiley on a frame is shown in Figure 1.

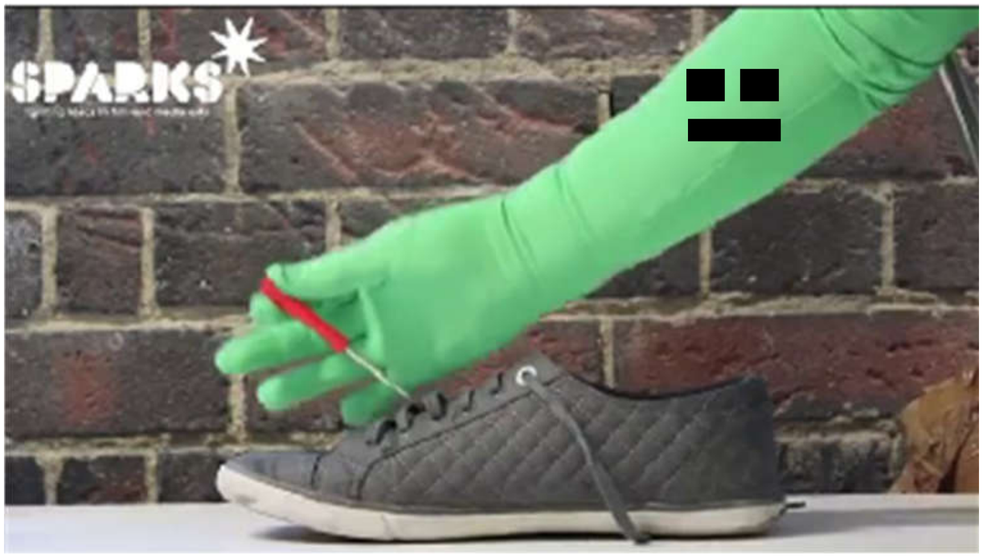


Figure 1: Example of a black smiley added on a frame of ‘green_hand.mp4’

The black boxes can be added by simply replacing the specific pixels of a frame by 0. Please note that you have frames in RGB format, so you need to replace specific pixels at each channel with 0.

Computer Vision (Fall 2019)

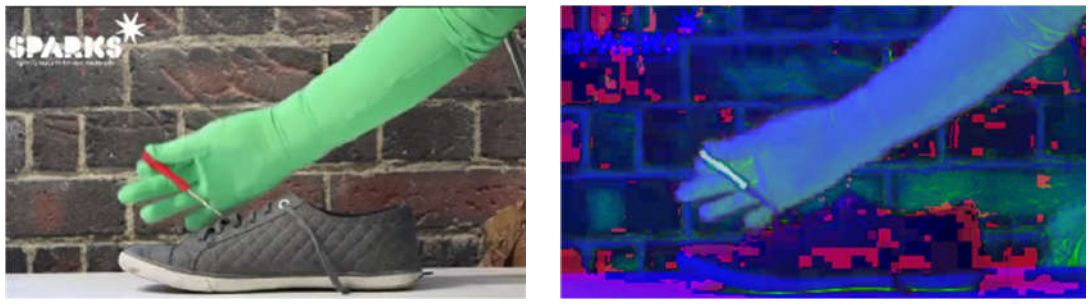
Task 2:

You are given two video clips called **‘green_hand.mp4’** and **‘green_cloak.mp4’** and two still background images called **‘green_hand_background.jpg’** and **‘green_cloak_background.jpg’**. You need to identify the green areas and fill them with pixels from the respective background image to produce the effect of invisibility. There are some basic steps for implementation of green color removal that are discussed below. However you are not restricted to use this approach, we encourage you to put your own thoughts and come up with a better solution for this problem.

1. Load the video clip using OpenCV. Then you need to access each frame of the video to process it.

2. The frames will be in RGB colorspace. You need to convert it to HSV colorspace. You can use OpenCV’s cv2.cvtColor(). See Figure 2 that shows an example frame converted to HSV colorspace.

HSV is basically a different way to describe colors than the classic RGB we are used to, you can learn more about it on Wikipedia. We will explain later why this conversion is necessary.

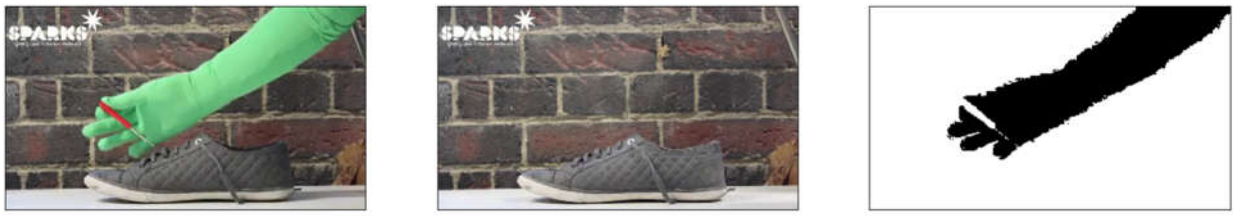


(a) RGB (b) HSV

Figure 2. Images in RGB and HSV color spaces

Note: You are not allowed to use any OpenCV or other library functions for the below tasks. This includes cv2.inRange(), cv2.bitwise_and() and cv2.bitwise_or() etc.

3. Now we need to filter out the green color. We will generate a binary mask whose size will be equal to original frame size. The mask will contain ‘0’ at the locations where color is green and ‘255’ at the places where the color is not green. See Figure 3.



(a) Frame **(b) Background Image** **(c) Binary Mask**

Figure 3: Binary Mask extracted from a frame

This mask will be a numpy array and can be created by looping over each pixel of each channel of HSV image. You will need to check if a pixel lies in the green range. This range will be defined in HSV space. You can take Google’s help to understand ranges of colors in this space. If a pixel lies in green range, save 0 at mask at the corresponding location, otherwise save 255.

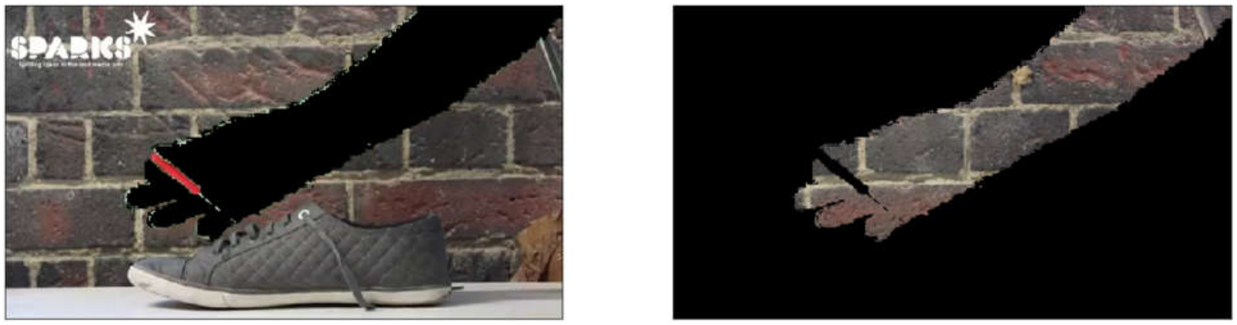
For example to check if the pixel’s H channel from HSV is in green range, we do the following:

```
if (30 < frame[0][row,col] < 60):
        #replace the pixel with 0 else with 255
```

HSV space allows us to filter out colors really well. Here we can define a range of values for Hue, Saturation and Value which helps in handling variations of a specific color. For example the green color will not be all the same in whole frame. There will be slight variations due to changing light exposure of different parts of the frame.

You will notice that HSV has allowed us to remove unwanted pixels with much more ease. We have to look at both Hue and Saturation of the pixel (which color it is and how intense the color is), giving us a room of adjustment for “value” (how dark it is). This setup allows us to handle shadows and lighting variations. To extract the green color you will need to study about HSV color space. Also have a look at this table http://www.rapidtables.com/convert/color/index.htm

4. Once you have extracted the mask, use it to create invisibility effect. You will create a numpy array equal to the size of original frame. This array will contain the processed frame. Then you will loop through the mask created in step 3, and you will fill the array of processed frame with pixels of original frame if mask contains 255 or pixels of background if mask contains 0. See Figure 4



(d) Mask applied to Frame **(e) Inverted mask on Background**

Figure 4. Resulting images from frame and background by applying mask are also shown.

5. The processed frame should look like Figure 5.

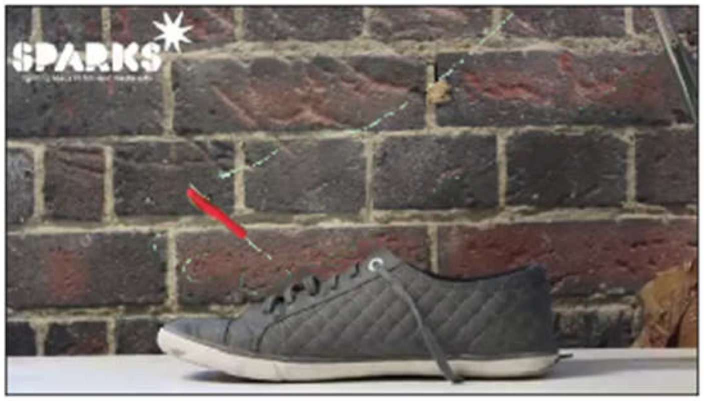


Figure 5: Final output frame with invisibility effect

6. You can store all the processed frames in a NumPy array and then save the final videos as **‘green_hand_processed.avi’** and **‘green_cloak_processed.avi’**.

Computer Vision (Fall 2019)

Tasks:

For this assignment you are given two video clips and their respective still background images. Your job is to:

1. Make the green color invisible and produce final processed video.
2. Please mention in the report what range have you used for Green Color in HSV space
3. Make a report detailing all the steps and any interesting results that you found. Include your analysis/comments on the results that are produced.
4. For each video, take a sample frame and produce all the outputs shown above (original frame, binary mask, mask applied to frame, inverted mask applied to background, final output frame)
5. Save a video of masks for each frame. This will essentially be a black and white video of just masks. Name this video as **'green_hand_masks.avi'** and **'green_cloak_masks.avi'**.
6. Please follow all the naming conventions as mentioned in each step.