Computer Vision Assignment 0

Nilabja Bhattacharya (Roll 2018201036)

11th January 2019

1 Introduction

Chroma key compositing, or chroma keying, is a visual effects/post-production technique for compositing (layering) two images or video streams together based on color hues (chroma range). The technique has been used heavily in many fields to remove a background from the subject of a photo or video – particularly the newscasting, motion picture, and video game industries. A color range in the foreground footage is made transparent, allowing separately filmed background footage or a static image to be inserted into the scene. The chroma keying technique is commonly used in video production and post-production. This technique is also referred to as color keying, colour-separation overlay (CSO; primarily by the BBC), or by various terms for specific color-related variants such as green screen, and blue screen – chroma keying can be done with backgrounds of any color that are uniform and distinct, but green and blue backgrounds are more commonly used because they differ most distinctly in hue from most human skin colors. No part of the subject being filmed or photographed may duplicate the color used as the backing.

Figure 1: Green Screen Chroma Keying

2 Problem 1

2.1 Problem Description

The objective of this problem was splitting a video into its constituent frames and also to merge frames in a given folder to form a video

2.2 Solution

I've read the video using openCV function VideoCapture, then extracted each frame of the video using read function of openCV and saved these video into the given folder.
To merge videos, I've read images in the given folder and then written it to a VideoWriter variable to form video of given fps.

2.3 Code

```
from __future__ import print_function
import cv2 as cv
import os
import numpy as np
import argparse
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

parser = argparse.ArgumentParser()
parser.add_argument('--input',
                    type=str,
                    help='input video name.',
                    default=0
                    )
parser.add_argument('--algo',
                    type=str,
                    help='specify operation to be performed ie merge or split',
                    default='split')
parser.add_argument('--dir',
                    type=str,
                    help='Path to directory',
                    default='result')
parser.add_argument('--fps',
                    type=float,
                    help='fps for video',
                    default=10)
args = parser.parse_args()
def split(capture, fps, dest):
    i=0
```

```
    os.mkdir(dest)
    while capture.isOpened():
        ret, frame = capture.read()
        if frame is None:
            break
        i=i+1
        cv.imshow('Frame', frame)
        cv.imwrite(dest+ "/" + str(i)+".jpg", frame)

        keyboard = cv.waitKey(1)
        if keyboard == 'q' or keyboard == 27:
            break

def merge(source, v_n, fps):
    image_folder = source
    video_name = v_n

    images = [img for img in os.listdir(image_folder) if img.endswith(".jpg")]
    frame = cv.imread(os.path.join(image_folder, images[0]))
    height, width, layers = frame.shape
    images = sorted(images)
    video = cv.VideoWriter(video_name, 0,fps, (width,height))
    for i in range(1,len(images)):
        #print(os.path.join(image_folder, image))
        frame = cv.imread(os.path.join(image_folder, str(i)+".jpg"))
        video.write(frame)
        cv.imshow('f',frame)
        if cv.waitKey(1) & 0xFF == ord('q'):
            break
    cv.destroyAllWindows()
    video.release()

if args.algo=='split':
    if args.input==0:
        capture = cv.VideoCapture(0)
    else:
        capture = cv.VideoCapture(cv.samples.findFileOrKeep(args.input))
    if not capture.isOpened:
        print('Unable to open: ' + args.input)
        exit(0)
    split(capture, args.fps, args.dir)
else:
    merge(args.dir, args.input, args.fps)
```

2.4 Usage

Split

```
python3 2_1.py --input v34.mp4 --algo split --dir outputdir
```

Merge

```
python3 2_1.py --input v34.avi --algo merge --dir inputdir --fps 27.97
```

2.5 Implementation Result

- Implemented **split** function on a video, to split it into frames - Link
- Implemented **merge** function on images in a folder to form a video - Link

2.6 Learning outcome

I experimented how changing the **fps** value while merging frames to form video can alter the way video gets rendered. Apart from that I've learned how to use openCV for doing experiment on images and videos

3 Problem 2

3.1 Problem description

In this problem I've to used openCV function to capture frames using webcam and save them into the given folder

3.2 Solution

If we change the input for openCV function VideoCapture to 0, then openCV renders frame from webcam and I've saved the frame into the output folder using imwrite function

3.3 Code

```
from __future__ import print_function
import cv2 as cv
import os
import numpy as np
import argparse
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

parser = argparse.ArgumentParser()
parser.add_argument('--dir',
```

```
                        type=str,
                        help='Path to directory',
                        default='result')
args = parser.parse_args()

capture = cv.VideoCapture(0)
i=0
os.mkdir(args.dir)
while True:
    ret, frame = capture.read()
    if frame is None:
        break
    i=i+1
    cv.imshow('Frame', frame)
    cv.imwrite(args.dir +"/"+ str(i)+".jpg", frame)

    keyboard = cv.waitKey(30)
    if keyboard == 'q' or keyboard == 27:
        break
```

3.4 Usage

```
python3 2_2.py --dir outputdir
```

3.5 Implementation Result

Implemented this code to take live video, thus render the frames to users and save the frames into the given folder - Link

3.6 Learning outcome

By doing this problem I've learnt how to use opencv to capture and save frames using a webcam

4 Problem 3

4.1 Problem Description

In this problem I've used openCV to chroma key a video, i.e., given that a video is shot in a constant colored background possibly green or blue, we can replace the background with another video, such that the foreground remains same and background gets replaced.

4.2 Solution

I've computed the mean and deviation of the background to compute the range within which the background pixel will exist, by taking a section of background.

After calculating the range of pixel intensity, I've replaced the background of first video with the content of second video wherever pixel is in this range.I apply this to every frame of the video and merge these frame to form the resultant video. For edges I've used an α value using which I decide what should be the ratio of pixel intensity for first video and second video.

4.3 Code

```
from __future__ import print_function
import cv2
import os
import numpy as np
import pandas as pd
import argparse
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

low = []
high = []
parser = argparse.ArgumentParser()
parser.add_argument('--input_1',
                    type=str,
                    help='Path to video 1')
parser.add_argument('--input_2',
                    type=str,
                    help='Path to video 2')
parser.add_argument('--output',
                    type=str,
                    help='Path to output video',
                    default='result.avi')
#parser.add_argument('--background', type=str, help='xyz')
args = parser.parse_args()
def restrict(color_component):
    return np.clip(color_component, 0, 255)

def calculate(frame_1):
    top_crop_y = 20
    bottom_crop_y = 359
    f_flattened = []
    f_hsv = cv2.cvtColor(frame_1, cv2.COLOR_RGB2HSV)

    for index in range(3):
        top = np.array(f_hsv[:top_crop_y, :, index]).flatten()
        bottom = np.array(f_hsv[bottom_crop_y:, :, index]).flatten()
```

```
            top_and_bottom = np.append(top, bottom)
            top_and_bottom_series = pd.Series(top_and_bottom)
            #print(top_and_bottom_series.describe())
            f_flattened.append(top_and_bottom_series)


        z_value = 10.0
        global low
        global high
        for i in range(3):
            mu = f_flattened[i].values.mean()
            sigma = f_flattened[i].values.std()
            deviation = z_value*sigma
            #print(mu)
            #print(deviation)
            low.append(restrict(mu-deviation-20))
            high.append(restrict(mu+deviation+20))
        print(low)
        print(high)


def chroma_key(frame_1, frame_2):
    bg_cropped = frame_2[:len(frame_1), :len(frame_1[0]), :]
    plt.imshow(frame_1)
    #print(frame_1.shape)
    #plt.show()

    global low
    global high
    #print(low)
    #print(high)
    mask_lower = np.array([low[0], low[1], low[2]])
    mask_higher = np.array([ high[0], high[1], high[2]])
    f_hsv = cv2.cvtColor(frame_1, cv2.COLOR_RGB2HSV)
    f_mask = cv2.inRange(f_hsv, mask_lower, mask_higher)

    masked_f = np.copy(frame_1)
    masked_f[f_mask != 0] = [0, 0, 0]

    f_hand = np.copy(frame_1)
    masked_f[f_mask != 0] = [0, 0, 0]

    bg_masked = np.copy(bg_cropped)
    bg_masked[f_mask == 0] = [0,0,0]

    full_picture = bg_masked + masked_f
```

```
    cv2.imshow('f_1',full_picture)
    return full_picture

if args.input_1 == None or args.input_2 == None:
    print("Please give input file")
    exit(0)
else:
    v1 = cv2.VideoCapture(args.input_1)
    v2 = cv2.VideoCapture(args.input_2)
    width = v1.get(cv2.CAP_PROP_FRAME_WIDTH)
    height = v1.get(cv2.CAP_PROP_FRAME_HEIGHT)
    fps = v1.get(cv2.CAP_PROP_FPS)
    ov = cv2.VideoWriter(args.output, 0,fps, (int(width),int(height)))
    i=0
    #calculate(cv2.imread(args.background))
    while v1.isOpened() and v2.isOpened():
        ret_1, frame_1 = v1.read()
        ret_2, frame_2 = v2.read()
        if i==0:
            calculate(frame_1)
            i+=1
        full_picture = chroma_key(frame_1, frame_2)
        ov.write(full_picture)
        #cv2.imshow('frame_1', frame_1)
        #cv2.imshow('frame_2', frame_2)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    v1.release()
    cv2.destroyAllWindows()
```

4.4 Usage

```
python3 2_3.py --input_1 1.mp4 --input_2 v33.mp4 --output edited.avi
```

4.5 Implemention Result

Implemented Chroma-Key algorithm on two different videos and the results can be found on following links

1. Link
2. Link

4.6 Learning Outcome

I've read the research paper for Blue Screen Matting[1] and I've realised that Traingulation matting is most efficient method to replace background but the

only flaw that it has is, it needs four images to extract the foreground from background efficiently and for videos its two times the effort because we need to shoot the video two times in different background.

I've shot video under varying background and lighting conditions and I've realised that chroma-key methods works efficiently only when the first video is shot with a fixed green or blue background and has proper lighting conditions

I've tried to chroma-key on live video shot using webcam but it didn't work efficiently because background wasn't of fixed color.

Apart from that I've also tried applying background substraction to get the alpha matte of the first video and then apply aplha matting to merge first and second video. But the problem that I faced was it was blending of two videos in some proportion and it was not a chroma key video.

Finally I came to the conclusion that only way to chroma-key a video is to have a fixed background of green or blue color and with proper lighting conditions, so that we can track the range of background pixel intensity and thus replace them with that of second video to form the resultant video

5 Links

- Github Link
- Google drive Link

6 Conclusion

This assignment helped me understand how to use openCV and apart from that it was helpful in understanding how chroma-key algorithm is used in movie industry effectively to merge two videos and render the merged video to viewers.

References

[1] Alvy Ray Smith and James F. Blinn. *Blue Screen Mating.*