CV Assignment 0 Report

Ashwin Rao (2019101049)

Task 1: Converting video to images and vice versa

A video is basically a sequence of images (frames). This question has 2 parts:

- Splitting a video into its corresponding frames and saving each frame as a separate image.
- Combining a sequence of images into a video.

Through this question, I learned:

- How to read videos using OpenCV VideoCapture.
- How to write videos using OpenCV VideoWriter given an image sequence.
- How to access properties of a video using OpenCV (width, height, FPS, etc).
- How to access each frame of a video in a loop and perform operations on them.
- How to write an image to a file using OpenCV.
- Working with file structures in python (creating and deleting files and directories).

Experiments

- I experimented with different FPS rates for the video created by combining the image sequence generated in the first part of the question. The required FPS can be passed as a parameter to the function which creates the new video.
    - If the FPS of the new video is the same as the FPS of the original video, then the new video has the same length as the original video.
    - If the FPS of the new video is greater than the FPS of the original video, the resultant video is longer. Similarly, if the FPS of the new video is less than the FPS of the original video, the resultant video is shorter.
- The video can be created by combining the image sequence in two ways:
    - Manually looping over each image in the sequence. In this case, we have no metadata about the original video like its FPS. However, we can get the width and height of the video from its constituent frames.
    - Creating a VideoCapture object from the image sequence and iterating over its frames, similar to the first part of this task. In this case, we have metadata about

the original video like its FPS, width and height readily available from the VideoCapture object.

- The VideoWriter API can create a video in different formats (AVI, MP4, etc). The codecs used for specific formats are operating system dependent. I experimented with different codecs and found that on my system, the codec `mp4v` worked to convert an mp4 into another mp4. Codecs `XVID` and `MJPG` did not work.

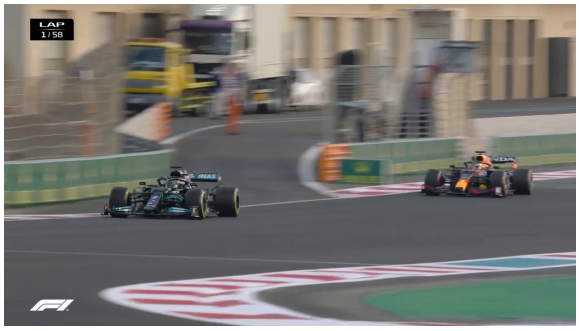
Code

```
import cv2
import os
import shutil


def splitVideoIntoFrames(video_path, save_path):
    video = cv2.VideoCapture(video_path)

    # empty contents of save_path or create it
    if os.path.exists(save_path) and os.path.isdir(save_path):
        shutil.rmtree(save_path)
    os.makedirs(save_path)

    # process video frame by frame
    frame_num = 1
    while True:
        ret, frame = video.read()
        if not ret:
            break

        # write frame to file
        cv2.imwrite(os.path.join(save_path, f"frame{str(frame_num).zfill(6)}.png"), frame)
        frame_num += 1

    video.release()


def mergeFramesIntoVideo(img_path, video_path, fps=None):
    filename = os.listdir(img_path)[0]

    frame_num_len = None
    for i, c in enumerate(filename):
        if c.isdigit():
            frame_num_len = len(filename[i: filename.find(".")])
            break

    video = cv2.VideoCapture(os.path.join(img_path, f"frame%0{frame_num_len}d.png"))
    width = int(video.get(3))
    height = int(video.get(4))
    if fps is None:
        fps = int(video.get(5))

    out = cv2.VideoWriter(video_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (width, height))

    # process each frame
    while True:
```

```
        ret, frame = video.read()
        if not ret:
            break

        # write frame to video
        out.write(frame)

    video.release()
    out.release()


frames_path = "frames"
splitVideoIntoFrames("race.mp4", frames_path)
mergeFramesIntoVideo(frames_path, "merged.mp4", fps=20)
```

Results

Frames from the original video (and new video):



Task 2: Capturing video from a webcam and displaying it

This task builds upon the previous one. We can use the same VideoCapture class of OpenCV to capture videos from a webcam. Instead of specifying a path to the video (as in the previous task), we specify 0 to access the webcam.

Through this question, I learned:

- How to read a live webcam stream using OpenCV.

- How to display images using OpenCV in an interactive window.

Experiments

- Performing specific actions in the webcam stream and observing the generated image sequence to see if they match the original stream.
- Finding FPS of the webcam. This data is available through the VideoCapture object. It is `30fps` for my webcam.
- Comparing the number of frames saved and the FPS of the webcam to the length of the webcam stream. $Length\ of\ video\ (s) = fps \times number\ of\ frames$.

Code

```
import cv2
import os
import shutil


def captureWebcam(save_path):
    video = cv2.VideoCapture(0)

    # fps of webcam
    print(video.get(5))

    # empty contents of save_path or create it
    if os.path.exists(save_path) and os.path.isdir(save_path):
        shutil.rmtree(save_path)
    os.makedirs(save_path)

    # process video frame by frame
    frame_num = 1
    while True:
        ret, frame = video.read()
        if not ret:
            break

        # write frame to file
        cv2.imwrite(os.path.join(save_path, f"frame{str(frame_num).zfill(6)}.png"), frame)
        frame_num += 1

        # display frame
        cv2.imshow("webcam", frame)
        if cv2.waitKey(1) == ord('q'):
            break

    video.release()


frames_path = "frames"
captureWebcam(frames_path)
```

Results

Frames from the captured webcam stream:




Task 3: Chroma Keying

This is popularly known as the **green screen effect**. It involves replacing a green screen (usually in the background) of a photo or video with a new background. In this task, I learned the concepts behind the green screen effect and how to apply it to both images and videos.

Chroma keying is done by simple colour thresholding, that is, identifying all pixels in a certain range of colour and applying a mask to remove them. The specific process is as follows:

- Find all pixels within a certain colour range (in this case, green pixels).
- Create a mask to filter out all these pixels in the specified range.
- Combine the 2 images using this mask as follows:

- Suppose we wish to apply a background `bg` to a frame `img` and the `mask` is 1 at all pixels belonging to the green screen in `img`.
- We combine them as `img[mask != 0] = bg[mask != 0]`.
- This could also be done using bitwise operators like `bitwise_and`.

To apply this effect on 2 videos, we process each video frame by frame and perform chroma keying for each pair of frames.

Experiments

- Finding a good range to properly filter out the green screen involved a significant amount of experimentation. It highly depends on the quality and consistency of the green screen. The video I chose with the green screen also had shadows on the green screen. This means that the colour threshold needed to cover a wider range of green to remove these darker shades as well. In the end, I chose the range [0, 80, 0] to [85, 255, 85].
- After applying chroma keying, a thin green outline was visible on the foreground objects. This is because these pixels on the edges also had green mixed in them. To remove this outline, I tried the following:
  - Detecting edges (using Canny edge detection) and making these pixels transparent (by setting alpha to 0). This can be done by applying the edges as a mask to the original image. Although this works for images, it did not work for videos since the VideoWriter class does not allow RGBA images as input (where A is the alpha channel).
  - Changing the range of colours to be considered as a part of the green screen. After much experimentation, the chosen range seems to minimize the green outline without affecting the rest of the image. For example, an upper threshold of [80, 255, 80] resulted in the green outline being more prominent. An upper threshold of [90, 255, 90] filtered out more points but resulted in other parts of the images being removed as well, which was not desirable.

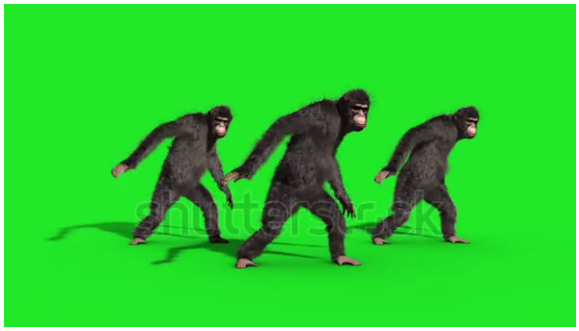
Code

```
import cv2
import numpy as np


def removeGreenScreen(video_path, background_path, output_path):
    # foreground
    video1 = cv2.VideoCapture(video_path)
    width = int(video1.get(3))
    height = int(video1.get(4))
```

```
    # background
    video2 = cv2.VideoCapture(background_path)

    # combined video
    fps = int(video1.get(5))
    out = cv2.VideoWriter(output_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (width, height))

    # process video frame by frame
    while True:
        ret1, frame1 = video1.read()

        ret2, frame2 = video2.read()
        frame2 = cv2.resize(frame2, (width, height), interpolation=cv2.INTER_AREA)

        if not ret1 or not ret2:
            break

        # generate mask
        l_green = np.array([0, 80, 0])
        u_green = np.array([85, 255, 85])
        mask = cv2.inRange(frame1, l_green, u_green)

        # apply mask
        frame1[mask != 0] = frame2[mask != 0]

        # display frame
        cv2.imshow("video", frame1)
        if cv2.waitKey(1) == ord('q'):
            break

        # write frame to video
        out.write(frame1)

    video1.release()
    video2.release()
    out.release()


removeGreenScreen("video.mp4", "background_long.mp4", "combined.mp4")
```

Results

Frames from the original videos. Notice the shadows in the first frame which made choosing the colour threshold for chroma keying more difficult.



Corresponding frame in the combined video after performing chroma keying. Notice that the green outline exists but does not completely surround the monkeys in the foreground.