CV Assignment 0

Name: Ayush Kumar Lall
Roll No.: 2020122001
Batch: UG2k19

Question 1:

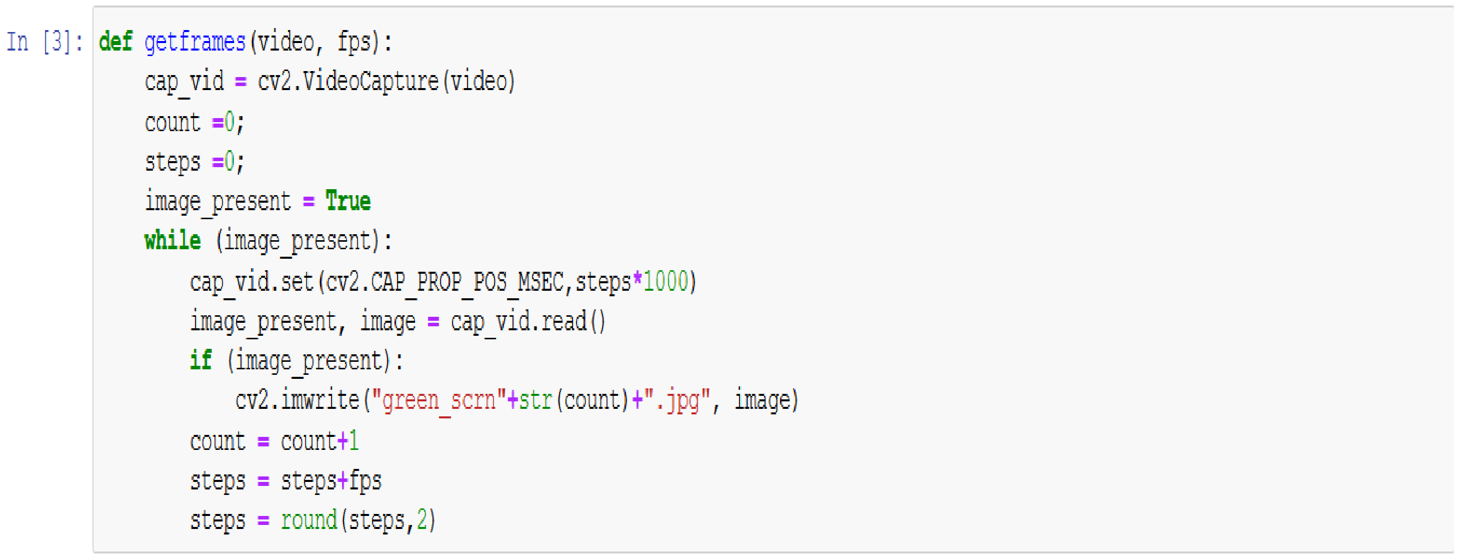
In this question it was asked to convert a video into its frame. So basically every video is just made with the help of the combination of several pictures which is made to play at certain rate, which is basically called FPS (frames per second) where each fame is a static picture and combination of these several picture result into required video.

- In order to do that I have used openCV function cv2.VideoCapture which will be used to capture the video that is given as the input.
- After the video is imported, I have used another openCV function given as cv2.CAP_PROP_POS_MSEC, which is used to get the frames of the video, in this we can also set the steps, such as 1/4th of the second an image/frame has to be captured.
- So in one second we will get 4 frames, thus if we have a video with the length of 30 secs then total there will be 4*30 =120 frames in total.

In this we can experiment with the no. of frames that we want to capture, so in my code, I used the 1/4th, 1/10th and 1/1 of the second as frame rate. As an observation it was observed that, as we take smaller steps, more no. Of frames were capture. So, when we reconstruct video with the help of more frames we will get a smoother output video.

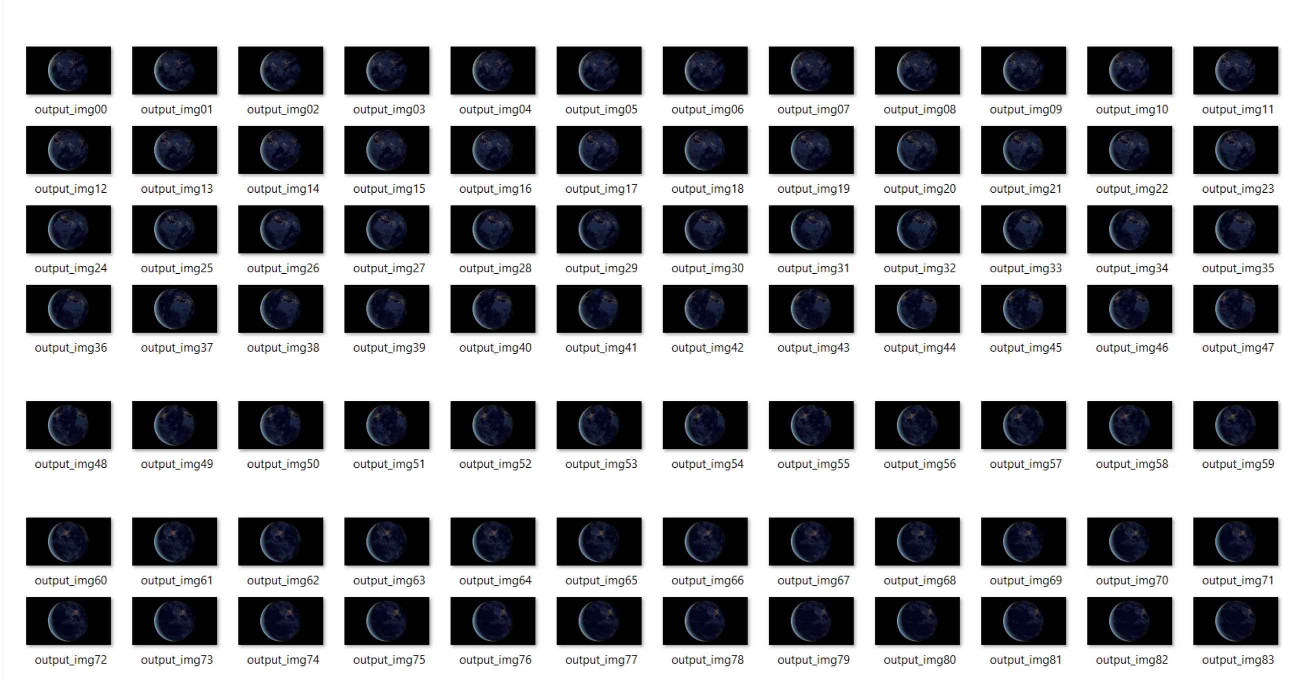
Challenges Faced:- In this question it was important to get the proper frames, or else the reconstructed video will be having lot of lags.

Below is the code snippet :-



```
In [3]: def getframes(video, fps):
            cap_vid = cv2.VideoCapture(video)
            count =0;
            steps =0;
            image_present = True
            while (image_present):
                cap_vid.set(cv2.CAP_PROP_POS_MSEC,steps*1000)
                image_present, image = cap_vid.read()
                if (image_present):
                    cv2.imwrite("green_scrn"+str(count)+".jpg", image)
                count = count+1
                steps = steps+fps
                steps = round(steps,2)
```

Results were like:-



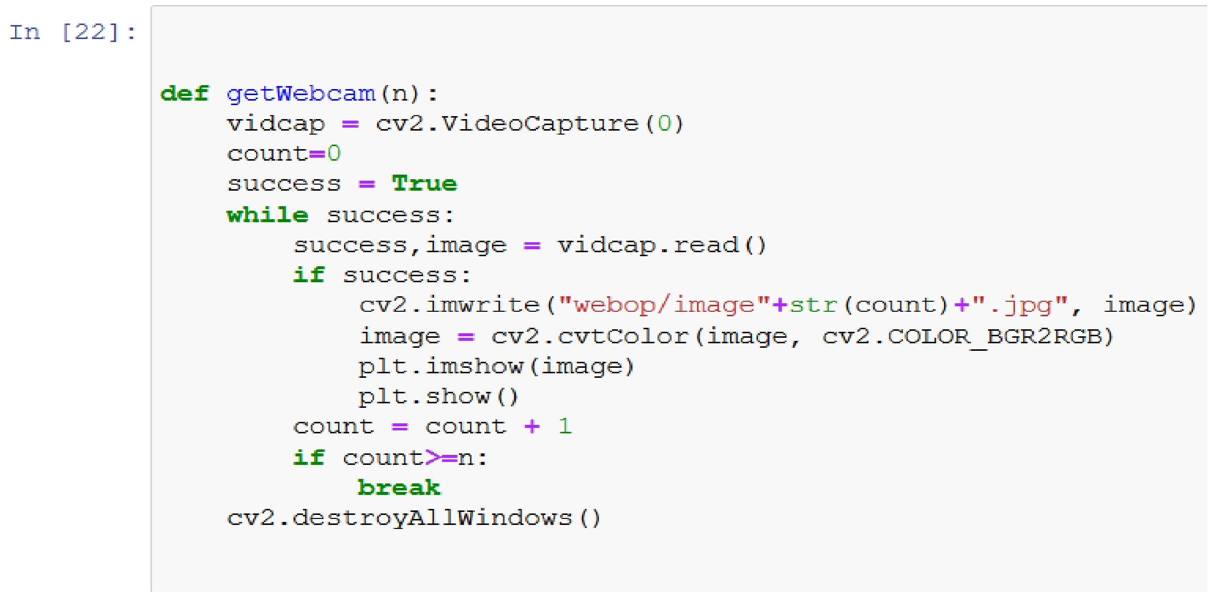
Question 2:

In this question it was asked to similar thing as done in the question 1, but this time we have to use our WebCam in order take the video, and then convert that video to its respective frames. The concept used here is also same as the first question, but we have to do additional task to access our webcam.

- In this question also I have used the same CV function, i.e. cv.VideoCapture in order to get the video from the WebCam.
- The code is written such that, when there are enough frame (i.e. given during the input) the code will stop executing and all the frame will be stored in a folder.

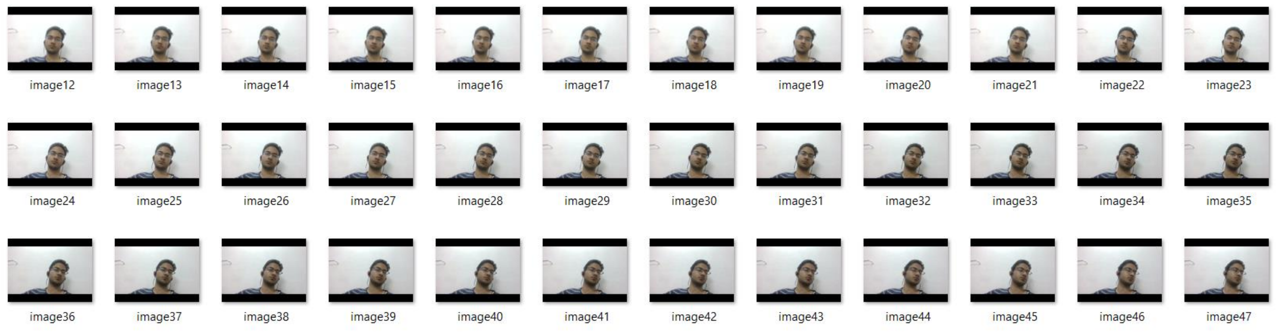
For experiment purpose in this question, we can change the threshold of the frames that we require.

Below is the code snippet:-



```
In [22]:

def getWebcam(n):
    vidcap = cv2.VideoCapture(0)
    count=0
    success = True
    while success:
        success,image = vidcap.read()
        if success:
            cv2.imwrite("webop/image"+str(count)+".jpg", image)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            plt.imshow(image)
            plt.show()
        count = count + 1
        if count>=n:
            break
    cv2.destroyAllWindows()
```

Results were like this:



The images were also displayed in the jupyter notebook, which can be done with the help of the plt.imshow command.

Challenges Faced: - One thing to keep in mind when doing this question is that, we should try to avoid using colab for this question as this, colab was not able to give permission for accessing the WebCam of the computer, so its better to run the code using jupyter rather than using online based colab.

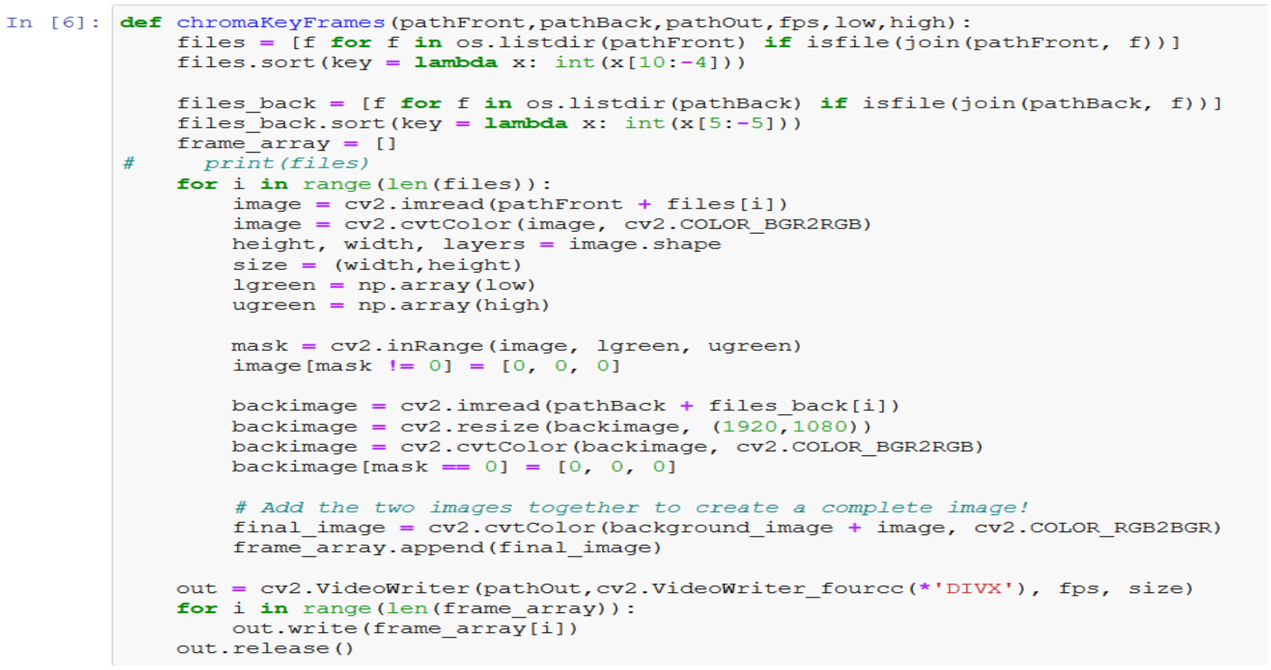
Question 3:

- In this question it was asked to use the chroma keying method which is basically used showing special background for a specific foreground. That is, consider a foreground image that is having a green background.
- If we take a real-life example of weather reporters then we will see that they also use the green screen in order to show the interactive background.
- The green screen is chosen such that it has a very distinct colour when compared to human skin so we can easily separate the foreground and the background image.
- Similar task we have to do in this question also, where we are taking a video with the background as green colour (it can be other colour like blue) and replacing that green background with another video.
- In order to do that we will work on the problem by taking each frame of the video and then we will try to replace the green background of the frame with the frame of any other video.

-This question also included the concepts from the question 1, where we converted the videos into their respective frames, and then performed the chroma keying on them.
-So, in this question first I'm segmenting the foreground part of the video based on the threshold of the green colour of the background.
-Thus, we get a mask and then we will put this mask on the background video and the finally combine the videos.

Below is the code snippet of the question:

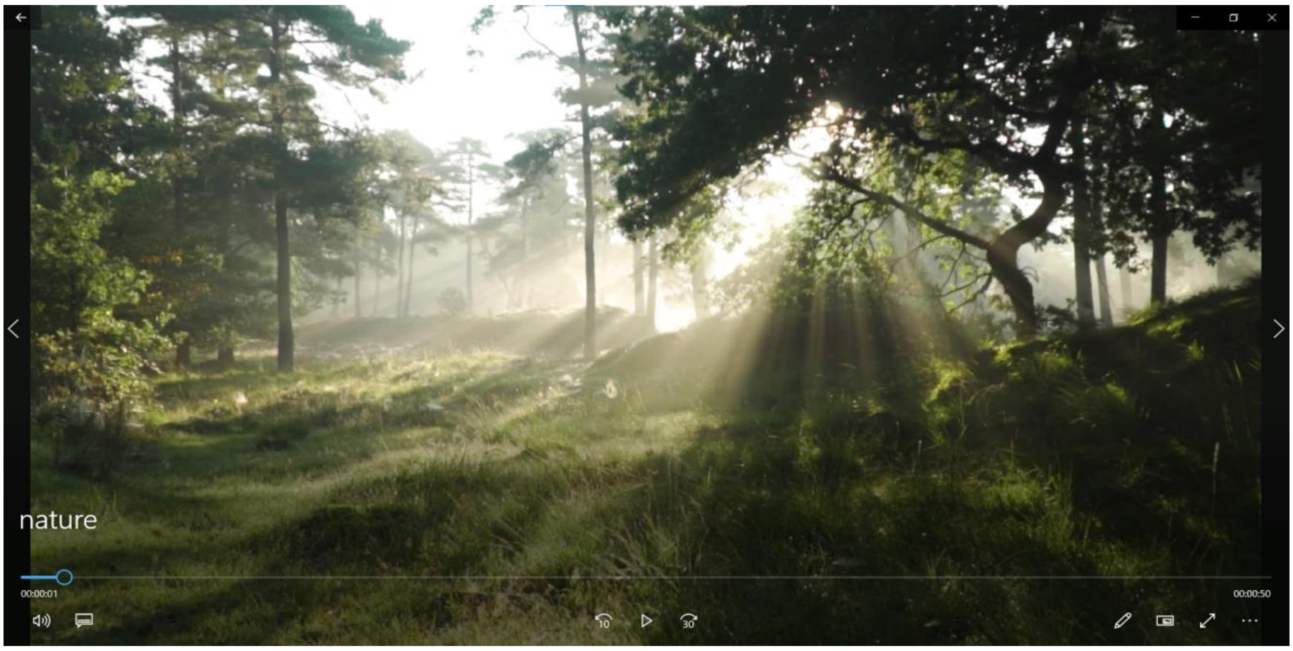
The results are given as below:



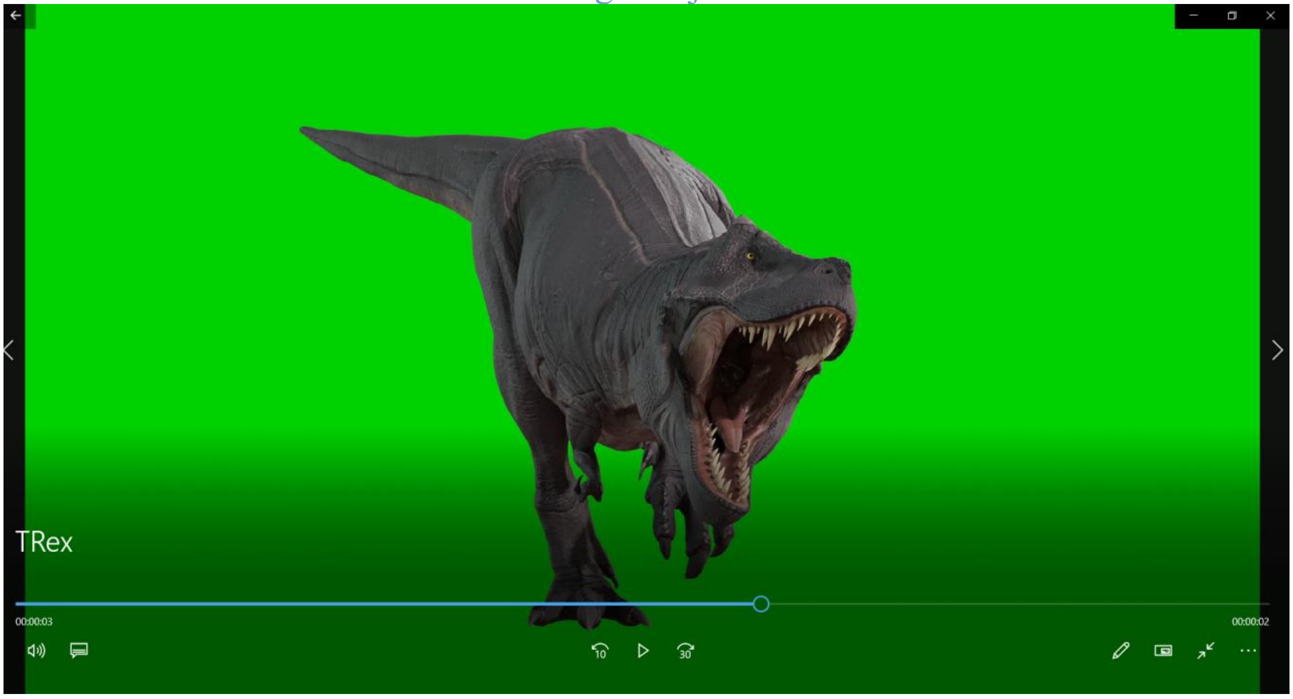
```
In [6]: def chromaKeyFrames(pathFront,pathBack,pathOut,fps,low,high):
            files = [f for f in os.listdir(pathFront) if isfile(join(pathFront, f))]
            files.sort(key = lambda x: int(x[10:-4]))

            files_back = [f for f in os.listdir(pathBack) if isfile(join(pathBack, f))]
            files_back.sort(key = lambda x: int(x[5:-5]))
            frame_array = []
        #     print(files)
            for i in range(len(files)):
                image = cv2.imread(pathFront + files[i])
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                height, width, layers = image.shape
                size = (width,height)
                lgreen = np.array(low)
                ugreen = np.array(high)

                mask = cv2.inRange(image, lgreen, ugreen)
                image[mask != 0] = [0, 0, 0]

                backimage = cv2.imread(pathBack + files_back[i])
                backimage = cv2.resize(backimage, (1920,1080))
                backimage = cv2.cvtColor(backimage, cv2.COLOR_BGR2RGB)
                backimage[mask == 0] = [0, 0, 0]

                # Add the two images together to create a complete image!
                final_image = cv2.cvtColor(background_image + image, cv2.COLOR_RGB2BGR)
                frame_array.append(final_image)

            out = cv2.VideoWriter(pathOut,cv2.VideoWriter_fourcc(*'DIVX'), fps, size)
            for i in range(len(frame_array)):
                out.write(frame_array[i])
            out.release()
```

Background video:-

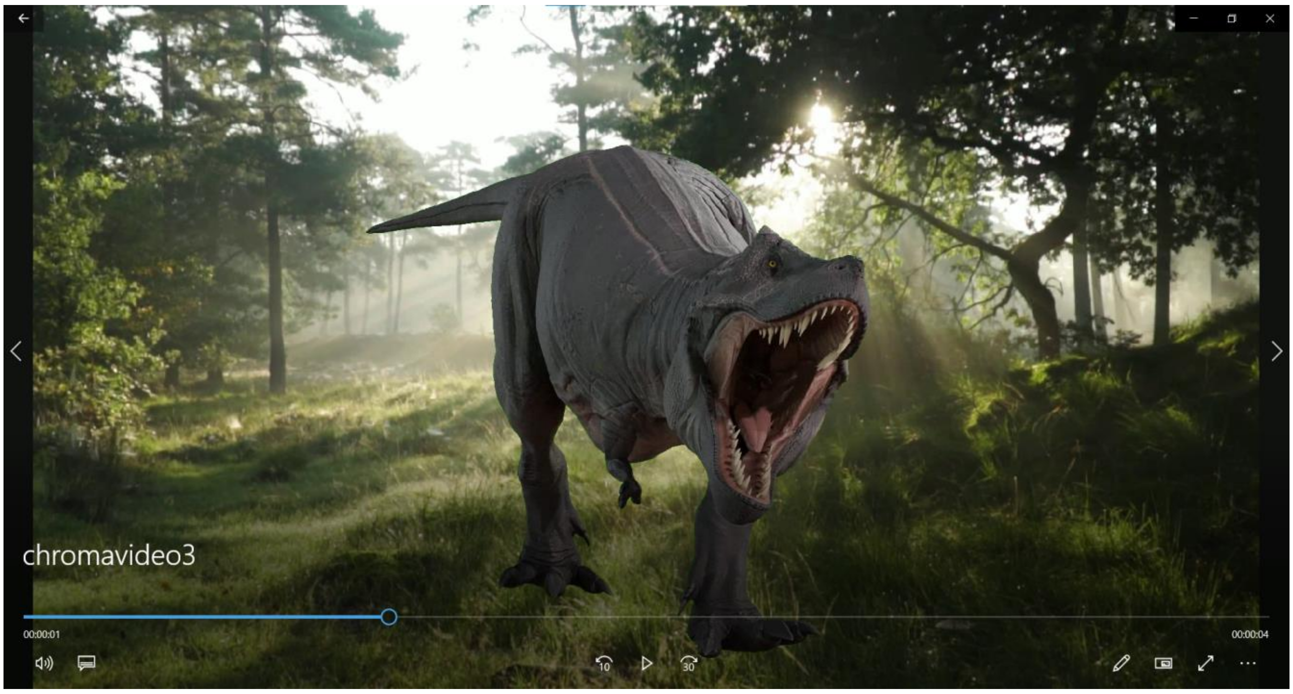
https://iiitaphyd-my.sharepoint.com/:v:/g/personal/ayush_lall_research_iiit_ac_in/EY5srfG8bOJFn12qtVMVQWgBxTTtOwDO4E6i_64c8d9VaA?e=hhFx6b



Foreground video:- https://iiitaphyd-my.sharepoint.com/:v:/g/personal/ayush_lall_research_iiit_ac_in/EbvuA2BCjRxLuY2v7Etz-tsB1c7RUISrUDLDhU8u7I5-Eg?e=EjGbrt

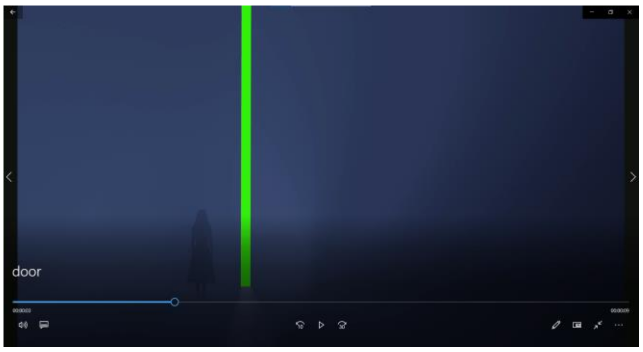


Combined video: https://iiitaphyd-my.sharepoint.com/:v:/g/personal/ayush_lall_research_iiit_ac_in/Ec7JwER_DPtOuQQVnkTMvG8Byb78550XZa54ZxoSzYje_A?e=iKXlJt

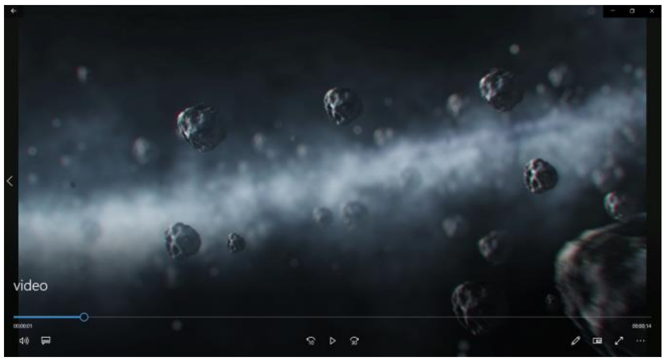


As an experimentation purpose I tried with different videos for foreground and background.
There were several challenges also that I faced and one of them was to properly decide the threshold of the green screen in order to separate the foreground and the background images.

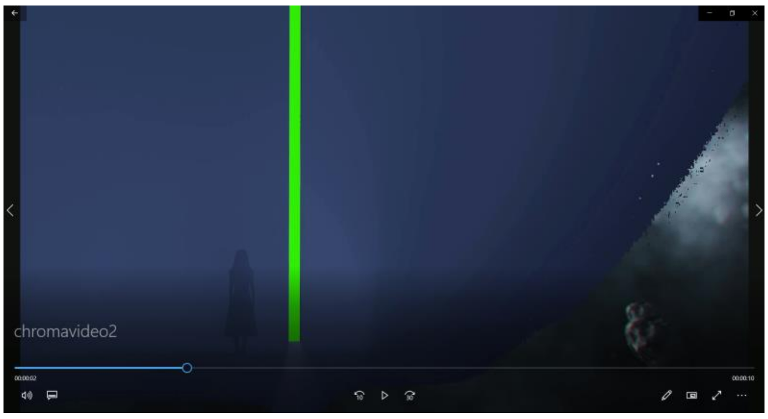
If the proper threshold is not chosen, then we will get absurd results for example:



Foreground Image



Background Image



Combined Image

So we can see that there was improper combination of the two videos, which is not good. So we have to take care of the thresholding parameters we are providing.