SCHOOL OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTING AND INFORMATION SYSTEMS

CSC2014: DIGITAL IMAGE PROCESSING

ACADEMIC SESSION: APRIL 2024

COURSEWORK PROJECT **DUE DATE: 26 JULY 2024**

Academic Honesty Acknowledgement

"I **Chew Win Nee, Chia Pei Qin** (Student Name), **23027964, 23015860** (Student ID). verify that this paper contains entirely my own work. I have not consulted with any outside person or materials other than what was specified (an interviewee, for example) in the assignment or the syllabus requirements. Further, I have not copied or inadvertently copied ideas, sentences, or paragraphs from another student. I realize the penalties *(refer to page 16, 5.5, Appendix 2, page 44 of the student handbook diploma and undergraduate programme)* for any kind of copying or collaboration on any assignment."


….., ….., (Student's signature / Date: 26th July 2024)

Table of Contents

1.0 Introduction

This assignment is required to help our friend, who works as a YouTuber, automate his video editing process by using digital image processing techniques. The functions include detecting the time of day a video was recorded and adjusting the brightness if it is nighttime, blurring faces for privacy, overlaying additional videos, adding logos and watermarks, merging end screen videos, and creating fade-in and fade-out effects. This assignment will be carried out using Python and OpenCV, using libraries like NumPy and Matplotlib to achieve the desired outcomes. The expected output will involve processing a total of four videos. The brightness will be increased if they are recorded at nighttime, and faces appearing in the videos will be blurred. Our friend who is talking will be overlaid on the bottom left after resizing and removing the background. Additionally, a watermark and logos will be added to protect intellectual property rights, an end screen video will be added at the end, and a fade-in fade-out effect will be applied at the opening and ending of the video and end screen video. In this report, we will discuss the justification for the proposed solution and the proper measurements used to evaluate its performance.

2.0 Methodology

2.1 Daytime & Nighttime Detection

To determine the time taken of the frame, it is converted to a grey image to calculate the mean intensity. To adjust the brightness of a nighttime image, the intensity of each pixel is increased with a value (*diff)* by using the *cv2.addWeighted* function.

2.2 Face Detection & Face Blurring

The Haar cascade model is implemented to detect the camera-facing faces in a frame and return a faces list with their coordinate and size. The *faceBlurring* will loop through each face in the faces list and perform a blurring function *cv2.blur* to blur the specified region.

However, the Haar cascade model might fail to detect the same face in subsequent frames. For instance, a face detected in frame X-1 and X+1 but not detected in frame X. This probably proves face detection failure occurs as a normal human might not be able to disappear from the camera in 1/30 seconds. Therefore, a *secondaryFaceDetection* function is developed to manually check if the detection failure occurs and fix the issue. A *templst* is used to store the information of faces occurring in three subsequent frames. The function will first check each face occurring in frame X-1, then check each face occurring in frame X+1 to calculate the differences of faces located in these two frames. A standard tolerance, *standard_diff* is used to store a constant pixel value which defines the distance allowed between two faces to be assumed as the same face. If the same face is detected in frame X-1 and X+1, it will only scan through the face in frame X to find whether face is found at the location around frame X-1 or frame X+1 by using the *standard_diff*. If the same face is found in frame X, a Boolean *face_fail_to_detect* is set to false which indicates the face is successfully detected. While if a face fails to detect, a face region is calculated by using the mean area of faces in frame X-1 and X+1. This face will be added to frame X as a result of secondary face detection.

2.3 Narrator

A variable *narrator_frame_count* stores the frame count of the *talking.mp4* video to match the respective frame count of the original video. The particular narrator frame will be obtained by using the *narrator_frame_count* to be further processed *(refer to figure 1)*

2.3.1 Extracting

To extract the foreground (the person) of the narrator frame a mask is created by calculating the peak colour through HSV colour scale. Tolerance margins for each H, S and V channel are set to highlight the pixels between the peak value and their margins. Each of the channels have the lower value by computing peak value subtracting by the margin while upper value by computing peak value adding by the margin. Note that the HSV range in OpenCV is as follows: H value is between 0 to 179 while S and V values are between 0 to 255. To ensure the lower and upper value computed is within the range, the max and min function is used. Afterwards, the function *cv2.inRange* is used to highlight (will be set to intensity of 255) the pixels in the frame which falls within the range which actually forms the inverse mask *(refer figure 2)*. NOT operation will be performed to get the mask with the correct region colours (intensity of background is 0 while foreground is 255) *(refer figure 3).*

2.3.2 Resizing

This logic is coded under the self-defined function *resize().*The mask and the narrator frame will be resized, meaning it will be smaller and at the position of bottom left corner of a background with resolution as the video size. To achieve this approach, the start and end coordinate for each x and y are calculated to specify the position to place the resized narrator frame. The *cv2.resize* function is used to resize the image to the desired resolution. The Boolean *grayStatus* is used to indicate if the image is a grayscale image. If the image is a grayscale image, conversion will be performed to convert the resized image to bgr to avoid shape mismatch error when placing it to the desired position of a black background (black background is just simply used as it is not important and will be discarded later. *(refer figure 4 & 5)*

2.3.3 Adding Narrator

This logic is coded under the self-defined function *mergeBackgroundnForeground().* After getting the resized mask and narrator frame, the narrator can now be added to the video frame. Operation NOT will be performed for the mask to get the inverse mask. Then operation AND will be performed for the inverse mask and background to obtain the masked background *(refer to figure 6)*. While another operation AND will be performed for the original mask and the narrator frame to obtain the masked foreground *(refer figure 7).* Afterwards, the masked background and background are added to achieve the approach of adding the narrator to the background *(refer figure 8)*.

2.4 Watermark and Logo

2.4.1 Adding Watermark

To add the watermark to the frame, *cv2.add* function is used to add the frame and the watermark image.

2.4.2 Alternating between Two Different Watermark

A watermarks list is created to store the two watermarks. A calculation to compute an index to decide which watermark in the watermarks list will be used in the current frame. This approach achieves the purpose of alternating the two watermarks between two different watermarks every 5 seconds,

2.4.3 Creating and Adding Logo

To create a logo array with resolution 64 x 64, a black array with the same size is created. Another array *rectangles* is created to store the information of every rectangle including start and end coordinate for each x and y. Afterwards, each area of the rectangle in the rectangles list will be coloured to white in the *logo array.* This logo array can also act like a mask to be extracted with a background colour later *(refer figure 9)*. To obtain a gradient logo, a gradient array with size 64 x 64 is created by using the *np.linspace* and *np.tile* functions *(refer figure 10)*.

The same self-defined function resize() and mergeBackgroundnForeground are used. This is because the operations are very similar to resizing and adding a narrator. The distinct feature is that the logo will be added at the top right corner of the frame instead of bottom left corner. A parenthesis position is received in the resize function to modify the start and end coordinate for each x and y to place the logo in the frame. Since the logo mask is grey, the grayStatus is set to TRUE to convert it to bgr to avoid shape mismatch in further operations. The following figures clearly show the operations. *(refer figure 11, 12, 13, 14 & 15)*

2.5 Fade effect

To achieve a fade effect at the beginning and end of the video, the self-defined function *applyFadeEffect()* is created. The parameter *fade_time* indicates the fade time in seconds and the parameter *effect* indicates either fade in or fade out effect will need to apply.
The *total_fade_frames* is calculated by multiplying fade time and the fps. If the fade frame count is smaller than the total fade frame, it will calculate the *effect_factor* by using formula: $\frac{fade\_frame\_count}{total\_fade\_frames}$. The frame will then multiply the *effect_factor* to achieve fade effect. The following table shows the relationship between total_fade

2.5.1 Fade-In Effect

The *applyFadeEffect()* will be called by passing in the effect as “Fade In”. The *fade_frame_count* will be the frame_count of the frame in the entire video. As the frame goes on, the intensity of the frame increases from 0% to 100% compared to the original. Assume the total fade frame = 150, the table shows the calculations of fade in effect. *(refer table 1)*

2.5.2 Fade-Out Effect

The *applyFadeEffect()* will be called by passing. In the effect as “Fade Out”. The *fade_frame_count* will be the total frame minus frame count of the frame in the entire video. As the frame goes on, the intensity of the frame decreases from 100% to 0% compared to the original. Assume the total fade frame = 150 and video total frame = 300, the table shows the calculations of fade out effect. *(refer table 2)*

2.6 Adding End Screen Video

To add the end screen video, the frame is written one by one to the output video. The fade in and fade out effect at the beginning and the end of the end screen are using the same function in section 2.5.

3.0 Results and Discussions

3.1 Daytime and Nighttime detection

The mean intensity is calculated by adding all the pixel values in the image and then dividing them by the total number of pixels. A low mean intensity indicates that the image is generally dark, while a high mean intensity represents that the image is generally lighter. Since the lighting conditions in the video are relatively uniform, the mean intensity provides a reliable measure of overall brightness. After several experiments, the video is taken in nighttime when the *mean_intensity* of the frame is less than 100. After a few examinations, a value higher than 100, which is 110 is suitable to compute the difference with mean intensity. This is because we assume 100 is the threshold value between nighttime and daytime. if it is nighttime, the value is probably smaller than 100. Therefore, we just make it slightly more than the threshold value by adding the difference value to each pixel.
As you can see, the brightness of the image is increased. There are more details in the image that can be identified (*refer table 3*).

3.2 Face detection & Face blurring

Before blurring the face, face detection is a need. The Haar feature-based cascade classifier, which is pre-trained for face detection is commonly used for this purpose. Due to some constraints, the detection of side faces is not effective. As a result, the face detection feature may incorrectly classify a person's face as a non-face object, resulting to a false negative detection by the Haar Cascade Model. Although a secondary face detector is implemented, the issue remains unresolved. However, most faces are still detected. After detecting the faces, they are blurred to ensure privacy. This is done by using the blurring function in OpenCV.

There is a total of 5 faces appear in this image. However, the side faces cannot be detected, so only 3 faces are blurred (*refer figure 16*).

3.3 Narrator

To overlay the talking video onto the main video, we first need to remove the background of the narrator and resize it to an appropriate location. To achieve this, the HSV color space was used to create masks for foreground extraction. Next, peak color values were determined, and tolerance margins were applied to extract the foreground. Consequently, the masks accurately highlighted the narrator with minimal background interference. Following this, the resizing function ‘resize()’ adjusted the dimensions of the mask and the narrator frame to fit within the desired region. Finally, the function ‘mergeBackgroundnForeground()’ was used to overlay the narrator frame onto the background.
The methods used for extracting, resizing, and integrating the narrator were chosen based on their ability to provide high accuracy and efficiency. The HSV color model accurately represents how humans perceive colors, unlike the RGB or CMYK models, which simply combine primary colors to create the spectrum. Therefore, the HSV color model is the most accurate depiction of how we perceive colors on a computer screen. In HSV, H stands for Hue, S stands for Saturation, and V stands for Value (GeeksforGeeks, 2022).

Figure 17 shows the output image after applying all the functions. The resized frames were placed correctly in the bottom left corner of the background (*refer figure 17*).

3.4 Adding watermark and logo

Please refer to table 4 in the appendix to see the results of alternating the watermarks.
As a result, the watermark changes every 5 seconds.
Refer to figure 18 & figure 19 to see the final output image of the logo and the location of the logo.

3.5 Fade-in Fade-out effect

Please refer to table 5 in appendix to see the results of fade-in fade-out effect.
As you can see, the fade-in fade-out effect is performed within a time frame of 5 seconds.

3.6 Adding End Screen Video

Refer to table 6 to see the result of adding the end screen video.

x

Appendix

Figure 1: Narrator frame

Figure 2: Inverse mask (foreground is black & background is white)- after applying cv2.inRange function by using the lower and upper boundary of peak colours

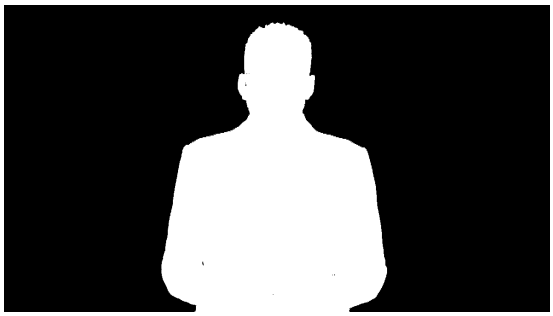


Figure 3: Mask - after applying NOT operation to the inverse mask

Figure 4: The mask after resized and placed at the bottom left of image with resolution that same as the video to add

Figure 5: The narrator frame after resized and placed at the bottom left of image with resolution that same as the video to add

Figure 6: The masked background (operation AND performed on inverse mask and background frame)

Figure 7: The masked foreground (operation AND performed on mask and narrator frame)

Figure 8: The combined image (Added masked foreground and masked background)

Figure 9: Mask of logo (with resolution 64 x 64)

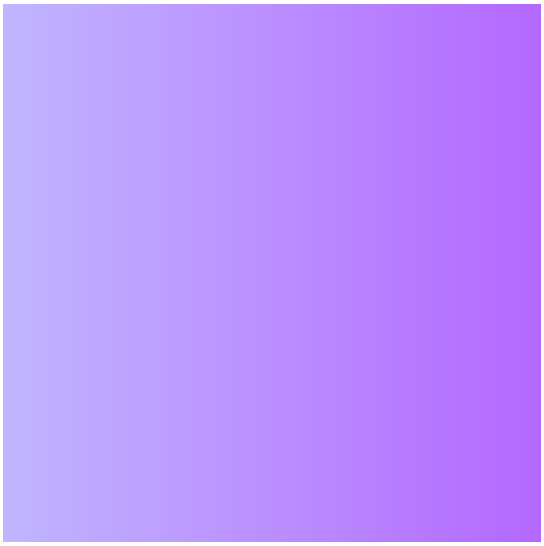


Figure 10:: Gradient background (with resolution 64 x64)

Figure 11: The logo mask after resized and placed at the top right corner

Figure 12: The gradient image after resized and placed at the top right corner

Figure 13: The masked background (Operation AND with background frame and inverse logo mask)

Figure 14: The masked foreground (Operation AND with logo mask and gradient image)

Figure 15: Combined image (Added mask foreground and background)

Figure 16: Face blurring output

Figure 17: Output image

Figure 11: Final output image of the logo

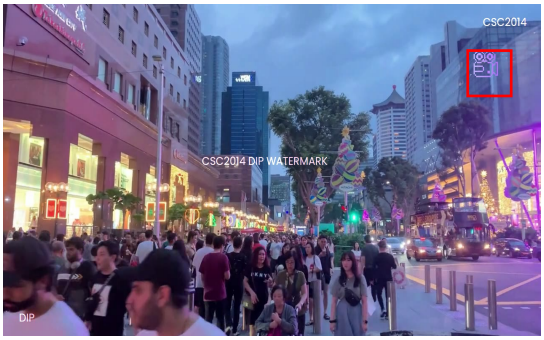


Figure 29: Location of the logo

Table 1: Fade In Calculations

Frame_count (in entire video)	Fade_frame_count (= frame_count)	Effect_factor	Expected Effect
0	0	$\frac{0}{150} = 0$	Each pixel in the frame multiply by 0, resulting in a black colour frame
50	50	$\frac{50}{150} = 0.3333$	The overall intensity of the pixels decreased to 0.3333% from the original.
100	100	$\frac{100}{150} = 0.6667$	The overall intensity of the pixels decreased to 0.6667% from the original.
150	150	$\frac{150}{150} = 1$	No changes

Table 2:Fade Out Calculations

Frame_count (in entire video)	Fade_frame_count (vid_total_frames - frame_count)	Effect_factor	Expected Effect
150	300 - 150 = 150	$\frac{150}{150} = 1$	No changes
200	300 – 200 = 100	$\frac{100}{150} = 0.6667$	The overall intensity of the pixels decreased to 0.6667% from the original.
250	300 – 250 = 50	$\frac{50}{150} = 0.3333$	The overall intensity of the pixels decreased to 0.3333% from the original.
300	300 – 300 = 0	$\frac{0}{150} = 0$	Each pixel in the frame multiply by 0, resulting in a black colour frame

Table 3: Results of adjusting brightness

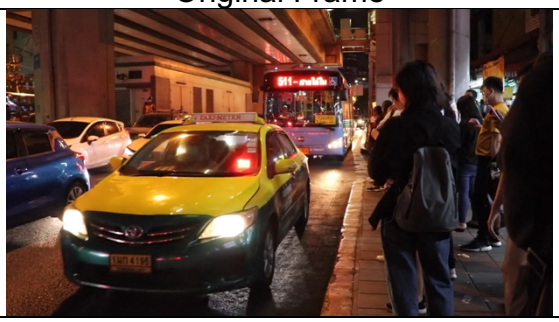
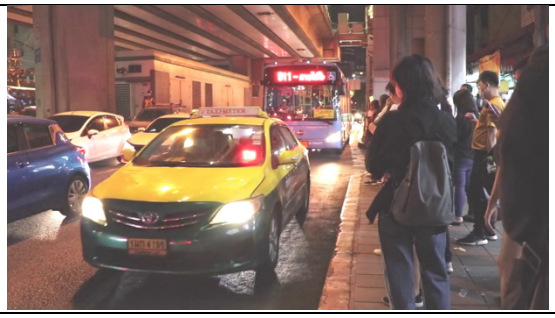
Original Frame	Processed Frame
	

Table 4: Results of alternating watermarks

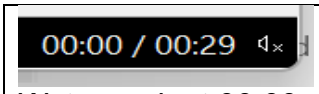
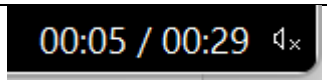
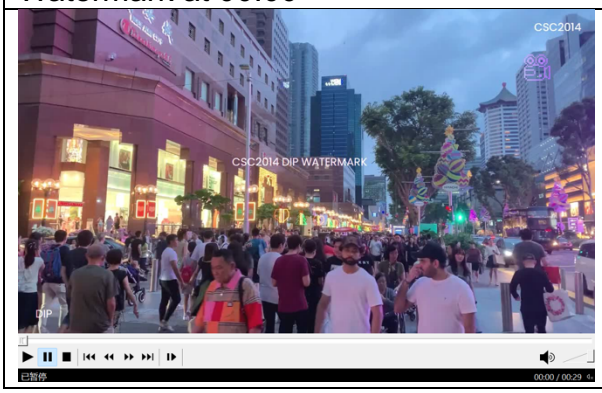
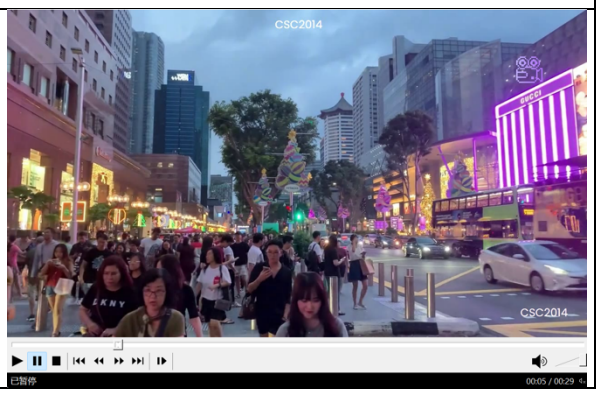
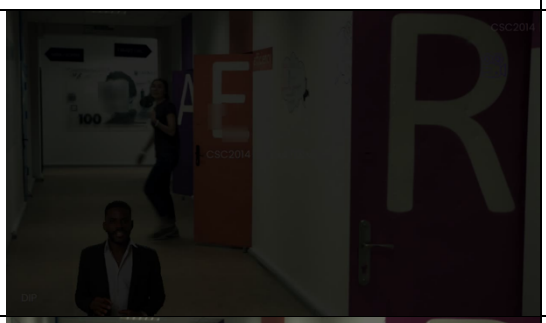
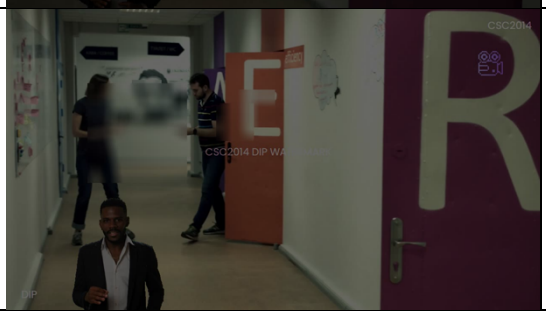
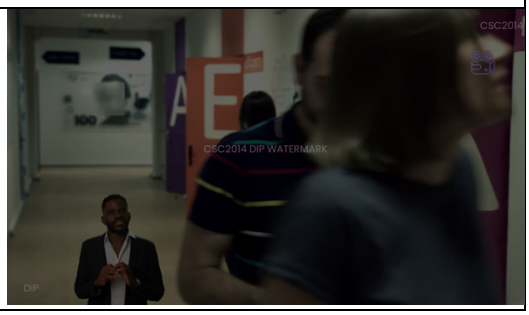
 Watermark at 00:00	 Watermark at 00:05
	

Table 5: Results of Fade in fade out effect

seconds	Fade-in effect	Fade-out effect
0		
2		

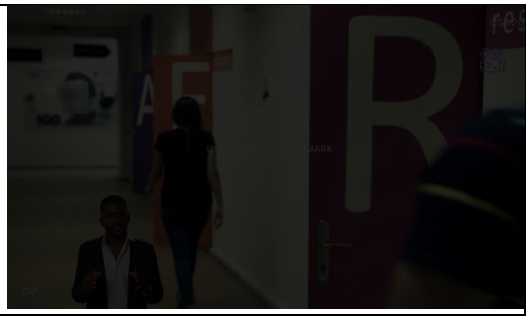
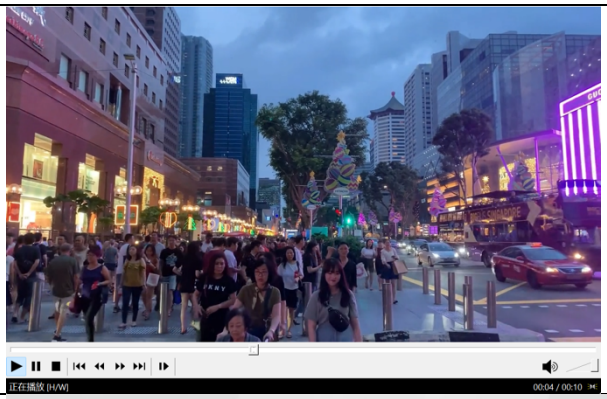
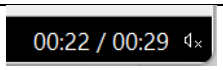
5		

Table 6: Results of adding the endscreen video

Video before adding end screen video, the total video length is 10 seconds.		
After adding the end screen video, the video length becomes 29 seconds.		

References

GeeksforGeeks. (2022). *HSV color model in Computer Graphics*. GeeksforGeeks. https://www.geeksforgeeks.org/hsv-color-model-in-computer-graphics/