ISTANBUL TECHNICAL UNIVERSITY

Faculty of Computer Science and Informatics

BLG453E

Computer Vision Homework-1 Report

Author

Cansu Yanık -150170704

NOTE: Output videos can be access via the Drive and Dropbox links below.

https://drive.google.com/open?id=1inkuN2UuSKMQqF17csBjMs0KljTNBx78

https://www.dropbox.com/sh/flp5e3rjlnjxtyi/AAAQ8_8Y8GUUoG4POZI0sdgva?dl=0

1. Part 1: "I am feeling blue"

For the first part, I chose the image sequence named "walking" from the DAVIS Challenge database. In order to read the images in order, I kept the image names into an array called "all_images". I wrote the names which are 0 to 9 to the array manually. For pictures after 10, I used for loop. Also the libraries I use are given in Figure1.

```
import cv2
import os
import moviepy.editor as mpy
import numpy as np

main_img_dir = './DAVIS-JPEGImages/JPEGImages/walking/000'
main_seg_dir = './DAVIS-JPEGImages/Annotations/walking/000'

all_images = ['00.jpg', '01.jpg', '02.jpg', '03.jpg', '04.jpg', '05.jpg', '06.jpg', '07.jpg', '08.jpg', '09.jpg']
#Keeps the images' names in order, I wrote the names which are 0 to 9 to the array manually.

#For pictures after 10, I used for loop.
list2 = list(range(10, 72))
for i in range(len(list2)):
    all_images.append(str(list2[i])+'.jpg')
```

Figure-1: Saving Images Names into Array to Read Them

In the Figure-2, for loop reads each image and the segmentation maps that belong to that image, and using the segmentation maps, masks some color channels of the images. Since the segmentation maps are saved as indexed images, *"cv2.IMREAD GRAY SCALE"* flag has to be written into the function. I selected the index of the guy in the image to mask. To be able to detect red color, 38 is given as index value. To be able to mask the out of the guy, I used *"cv2.bitwise_not(mask)"* function. I used the *"cv2.bitwise_and ()"* method to take the object from the original image. I gave the input image and mask segment as parameters. Thus, by applying the "and operation" to the input image and the segment I want to extract, I have received only the object I want. The "image_without_the_guy" and the "image_of_the_guy" outputs are shown in Figure-3. As stated in the description of homework, I decreased the values of red and green channels by 75%. Then I merged the extracted object and the area outside the object as an output image.

```
#This for loop reads each image and the segmentation maps that belong to that image,
#and using the segmentation maps, masks some color channels of the images.
for i in range(len (all_images)) :
    image = cv2.imread(main_img_dir +all_images[i])
    # Image is a numpy array with shape(800 , 1920 , 3)
    seg = cv2.imread(main_seg_dir +all_images[i].split('.')[0]+'.png ',cv2.IMREAD_GRAYSCALE)
    #Seg is the segmentation map of the image with shape(800 , 1920)

    #I selected the index of the guy in the image to mask. To be able to detect red color, 38 is given as index value.
    mask = cv2.inRange(seg, 38, 38)
    #To be able to mask the out of the guy
    mask2 = cv2.bitwise_not(mask)

    #To be able to take object from the input image, cv2.bitwise_and() is used
    image_without_the_guy = cv2.bitwise_and(image,image, mask= mask2)
    image_of_the_guy = cv2.bitwise_and(image,image, mask= mask)
    image_of_the_guy[:,:,1:3]=image_of_the_guy[:,:,1:3]*25/100
    #I decreased the values of red and green channels by 75%.

    #Merges the extracted object and the area outside the object as an output image.
    image = (image_without_the_guy + image_of_the_guy)
    #Saves the masked image into array
    images_list.append(image)
```

Figure-2: Reading Images from the Folder and Performing Mask Operation

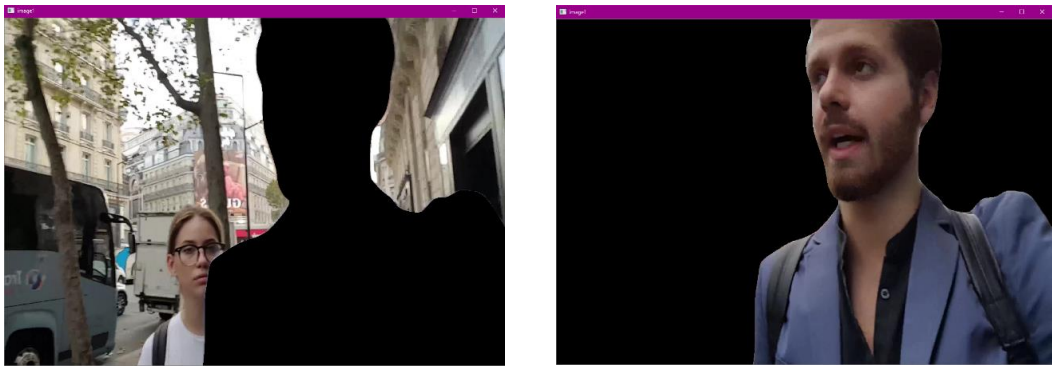

Figure-3: Outputs of "image_without_the_guy" and the "image_of_the_guy"

After applying mask operation to all images, frames are converted to a video in the code in Figure-4. For video writing operation, I used OpenCV. To be able to create a video with the size of the images, the size info is saved. I created a video structure using the *"cv2.VideoWriter()"* method. Some parameters like the video name, fps and video size are given into the method. Then, all the images are given into the video structure one by one. Figure-5 shows a part from the video.

```
44  #To be able to create a video with the size of the images, the size info is saved.
45  height,width,layers=images_list[1].shape
46
47  #Creates a video structure
48  video=cv2.VideoWriter('part1.mp4',-1,15,(width,height))
49
50  for j in range(len(images_list)):
51      video.write(images_list[j])
52
53  cv2.destroyAllWindows()
54  video.release()
```

Figure-4: Creating a Video using Image Frames

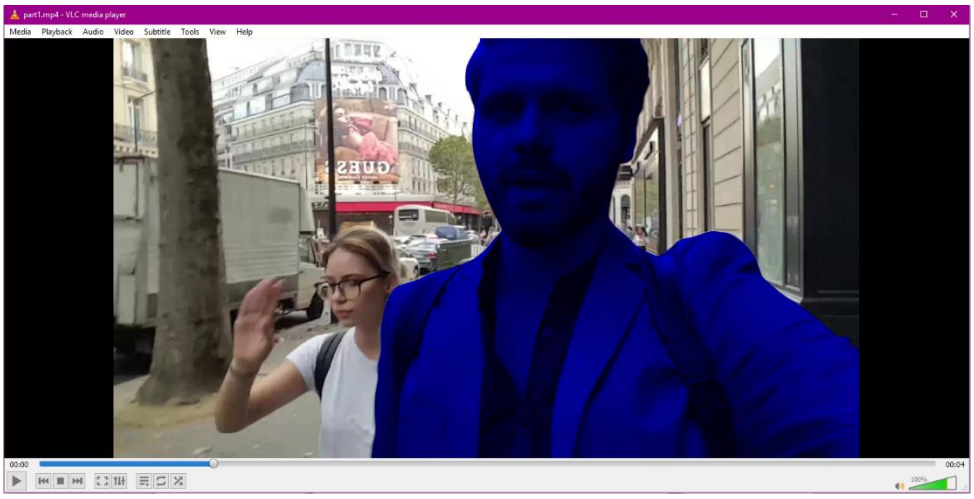

Figure-5: A Part from the Video

2. Part 2: Histogram matching

In the second part, I chose the "car-turn" image sequence. As a target picture, I searched and found the example image given in the assignment description. In Part 2, I took the histogram of the video frames one by one and found the average of all of them. So I worked with the average histogram when applying the histogram matching to all video frames. Since the input and target images are colored, they have 3 channels: Blue, Green and Red. Therefore, I applied histogram matching separately for all three channels. After finding the average histograms for the all channels, by using *"cumsum()"* method, I found the CDF function of the histograms of the channels and to be able to get values between 0 to 1, I found the normalized of the CDFs.

```
#Keeps Row, Coloumn and Bin information of the image and the target
R, C, B =  img.shape
R_t, C_t, B_t =  img2.shape


#Variables for finding average histogram matching, keeps the sum of the histogram of the channels
hist_b_sum = 0
hist_g_sum = 0
hist_r_sum = 0

for i in range(len(all_images)) :
    #Reads images from the folder
    img = cv2.imread(main_img_dir +all_images[i])

    #Finds summation of the histogram of the blue channels
    hist_b,bins_b = np.histogram(img[:,:,0].flatten(),256,[0,256])
    hist_b_sum = hist_b_sum + hist_b

    #Finds summation of the histogram of the green channels
    hist_g,bins_g = np.histogram(img[:,:,1].flatten(),256,[0,256])
    hist_g_sum = hist_g_sum + hist_g

    #Finds summation of the histogram of the red channels
    hist_r,bins_r = np.histogram(img[:,:,2].flatten(),256,[0,256])
    hist_r_sum = hist_r_sum + hist_r


#Takes the average of the all channels
hist_b = hist_b_sum/len(all_images)*1.0
hist_g = hist_g_sum/len(all_images)*1.0
hist_r = hist_r_sum/len(all_images)*1.0

#Finds the cdf function of the histograms of the channels
#And to be able to get values between 0 to 1, it finds the normalized of the cdf
cdf_b = hist_b.cumsum()
cdf_normalized_b = cdf_b / cdf_b.max()*1.0

cdf_g = hist_g.cumsum()
cdf_normalized_g = cdf_g / cdf_g.max()*1.0

cdf_r = hist_r.cumsum()
cdf_normalized_r = cdf_r / cdf_r.max()*1.0
```

Figure-6: Process of Taking Average of Histograms and Finding CDFs

Also I did the above operations for the target image. I took the histograms of each channel, found the CDF function and normalized it.

```
#Same operations are applied: histograms of each channel, found the cdf and normalized it
hist_t_b,bins_t_b = np.histogram(img2[:,:,0].flatten(),256,[0,256])
cdf_t_b = hist_t_b.cumsum()
cdf_normalized_t_b = cdf_t_b / cdf_t_b.max()*1.0


hist_t_g,bins_t_g = np.histogram(img2[:,:,1].flatten(),256,[0,256])
cdf_t_g = hist_t_g.cumsum()
cdf_normalized_t_g = cdf_t_g / cdf_t_g.max()*1.0


hist_t_r,bins_t_r = np.histogram(img2[:,:,2].flatten(),256,[0,256])
cdf_t_r = hist_t_r.cumsum()
cdf_normalized_t_r = cdf_t_r / cdf_t_r.max()*1.0
```

Figure-7: Process of Calculating Histogram, CDF and Normalization

I created a LUT table for each channel to make a histogram matching between the input and the target image. I used the algorithm in the course slides to create the LUT tables. In the while loop, each intensity value is checked for the number of pixels, and the first intensity value greater than the input value is recorded. Thus LUT table is used for matching the values. Then I created a matrix with the size of the input image and I found the proper intensity values corresponding to each pixel in the input picture from LUT and wrote them into the new matrix. I did this separately for each of the three channels. I applied this process to all video frames. and I saved the processed images in an array to prepare the video. The video creation step is done in the same way as in part1.

```
98   #LUT tables for each channel
99   LUT_b = np.zeros(256)
100  LUT_g = np.zeros(256)
101  LUT_r = np.zeros(256)
102
103  #index
104  gj = 0
105
106  #The process for the creating LUT tables for each channel (Taken from course slides)
107  for gi in range(256):
108      while cdf_normalized_t_b[gj] < cdf_normalized_b[gi] and gj < 255:
109          gj = gj + 1
110      LUT_b[gi] = gj
111
112  gj = 0 #indexes
113  gi = 0
114  for gi in range(256):
115      while cdf_normalized_t_g[gj] < cdf_normalized_g[gi] and gj < 255:
116          gj = gj + 1
117      LUT_g[gi] = gj
118
119  gj = 0 #indexes
120  gi = 0
121  for gi in range(256):
122      while cdf_normalized_t_r[gj] < cdf_normalized_r[gi] and gj < 255:
123          gj = gj + 1
124      LUT_r[gi] = gj
125
126
127  #Keeps all image frames which are outputs of the histogram matching
128  video_frames = []
129
130
131  for i in range(len(all_images)):
132      K = np.zeros([R, C, B], dtype=np.uint8)
133      img = cv2.imread(main_img_dir +all_images[i])
134
135      K[:,:,0] = np.uint8(LUT_b[img[:,:,0]])
136      K[:,:,1] = np.uint8(LUT_g[img[:,:,1]])
137      K[:,:,2] = np.uint8(LUT_r[img[:,:,2]])
138
139      video_frames.append(K)
```

Figure-8: Creation of LUTs and the Process of Histogram Matching

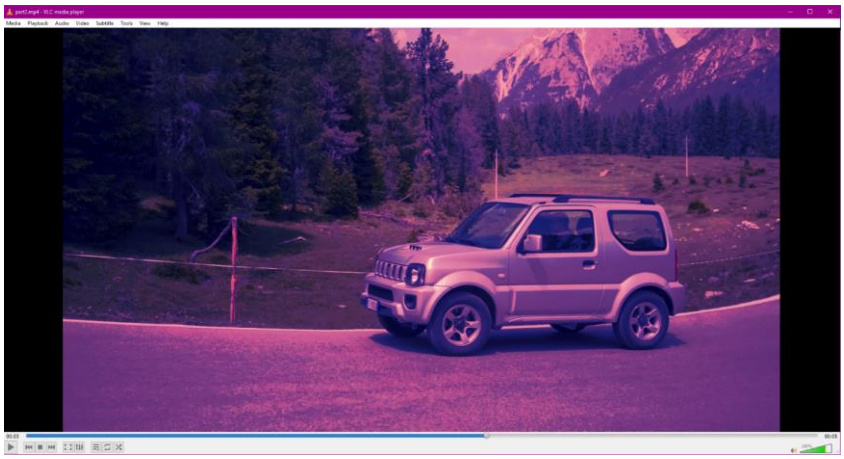

Figure-9: A Part from the Video

3. Part 3: Histogram matching from segmentation maps

In the third part, I chose the " night-race" image sequence. I used the images given in Figure-10 as target images.

Figure-10: Target Images I used

As in part2, I found the average histogram of each input image. I did this for all three channels. The code is as given in part2 (Figure-6). Again, I found the CDF function of the histograms of the channels and to be able to get values between 0 to 1, I found the normalized of the CDFs. Then I wrote a function called *"histogram_matching"*. The function takes the input and target images as parameters and performs a histogram matching between these two images and gives the output image. Inside the function is the same as the codes in Figure-7 and Figure-8 in part 2.

Then I extracted three objects from the images: these are the first car, the second car and the background. First I obtained a new image by applying histogram matching between the target image and the input image (I called my histogram matching function). Then I extracted the desired object from the image using segmentation map. The indexes which are 0 (black) for the background, 38 (red) for the first car, and 75 (green) for the second car were used. To get a single image, I combined the three images that are applied the histogram matching and that have individual objects. I saved output images in the array to prepare the video.

```
126 for i in range(len (all_images)) :
127     image = cv2.imread(main_img_dir +all_images[i])
128     seg = cv2.imread(main_seg_dir +all_images[i].split('.')[0]+'.png ',cv2.IMREAD_GRAYSCALE)
129
130     #histogram matching between the target image and the input image
131     object1 = histogram_matching(image, target1)
132     #I selected the index of the background in the image to mask.
133     #To be able to detect black color, 0 is given as index value.
134     mask = cv2.inRange(seg, 0, 0)
135     #To be able to take object from the input image, cv2.bitwise_and() is used
136     background = cv2.bitwise_and(object1,object1, mask= mask)  #background object
137
138
139     object2 = histogram_matching(image, target2)
140     #To be able to detect red color, 38 is given as index value.
141     mask2 = cv2.inRange(seg, 38, 38)
142     image_of_the_guy = cv2.bitwise_and(object2,object2, mask= mask2)   #first guy object
143
144
145     object3 = histogram_matching(image, target3)
146     #To be able to detect green color, 75 is given as index value.
147     mask3 = cv2.inRange(seg, 75, 75)
148     image_of_the_guy2 = cv2.bitwise_and(object3,object3, mask= mask3)   #second guy object
149
150     #Combines the three images that are applied the histogram matching and that have individual objects.
151     video_frames.append(image_of_the_guy2+image_of_the_guy+background)
152
```

Figure-11: Extracting three objects, Calling "histogram_matching" Function and Applying Mask Operations

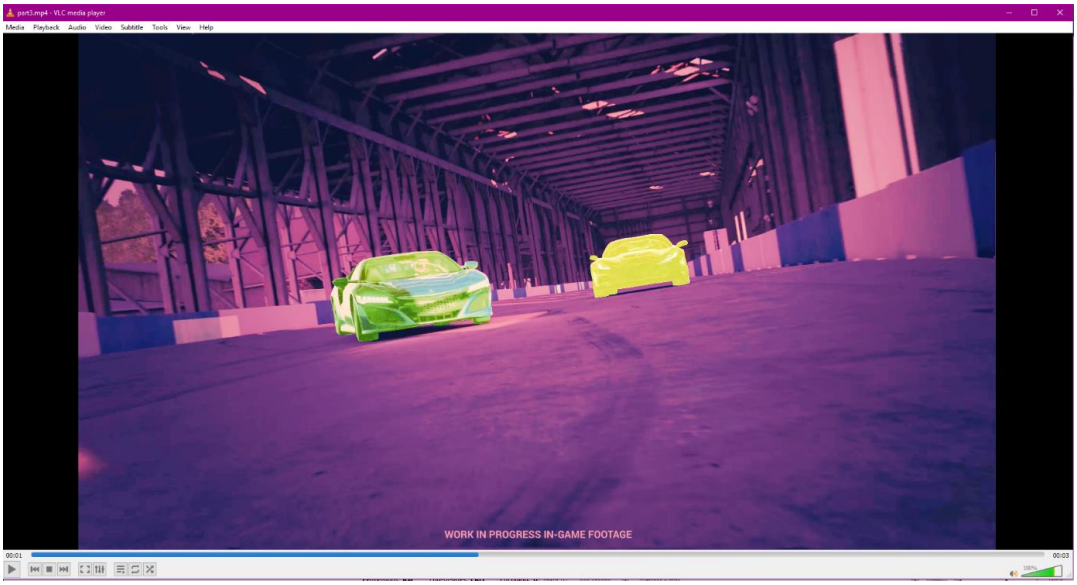

Figure-12: A Part from the Video