Assignment- 4: Color Processing- Documentation

Submitted by
Swagatam Haldar (16EE10063)
28/3/2020

- In this assignment, I have computed the dominant color of an image using *K-Means clustering (in CIE 2D chromaticity plane)* and displayed by marking the pixels belonging to the dominant color as white. Then I took two images as *source* and *target* pair. The RGB color distribution in the **source was modified to match the target illuminant distribution** via an intermediate **l-α-β** color space transformation. Details of those follow.
- The auxiliary helper functions handling all the mathematics are written in the file **utils.py** and the main tasks involving mouse click events and wrapper methods are implemented in the **modules.py** file.
- The user is required to interact with the **main.py** file only.
- **Note**- OpenCV reads, displays and writes images in **BGR** format. However, in this case some operations require *[R, G. B] values in that order*, so in some cases RGB to BGR conversion and vice-versa became necessary and wrapper functions were written to hide the details..

In `utils.py` file-

1. `def img_to_rgb_matrix(img_rgb)`- Takes an **RGB** 3 channel image, *[m, n, 3]* shaped color image and returns a matrix of shape *[3, m*n]*. Each column of the output matrix corresponds to [R, G, B] values belonging to a pixel.
2. `def rgb_matrix_to_img_rgb(rgb_array, m, n)`- Does the inverse of the above function i.e., reconstructs the RGB image from the rgb color values array.
3. `def rgb_to_xyz(rgb_array)`- Takes the rgb_array as constructed from the image and converts it to XYZ color space using a conversion matrix.
4. `def get_2D_chromatics(img)`- Takes a **BGR** 3 channel image and computes its XYZ color space values using the above 3 functions. Finally it computes **x = X / (X + Y + Z)** and **y = Y / (X + Y + Z)** and returns a *[2, m*n]* shaped array having the 2D chromatic characteristics of the original image.
5. `def get_dominant_color(img_rgb, xy_array)`- Using the 2D chromatic characteristics from above, it performs K-Means clustering (with **num_clusters = 3**) to find the color regions. The region/ centroid with the maximum no. of coordinates is chosen as the mode. Then from the *labels* and *indices* of each point, we back-calculate the (i, j) coordinates of the point in the original image and mark it white. It also constructs a *tuple* of x-coordinates and y-coordinates to store the pixels belonging to the dominant color cluster. Returns the whitened **BGR** image **showing dominant color region in white** and pixels.
6. `def rgb_to_lab(img_rgb, arrayflag = False)`-Takes an RGB image and converts its pixel to **l-α-β** color space through intermediate LMS color space. If arrayflag

= True, then it takes a rgb_matrix of shape *[3, p]* and converts it to **l-α-β** space. This case is useful in dominant color transfer where we have to convert the dominant pixels rather than the entire image to **l-α-β** space.

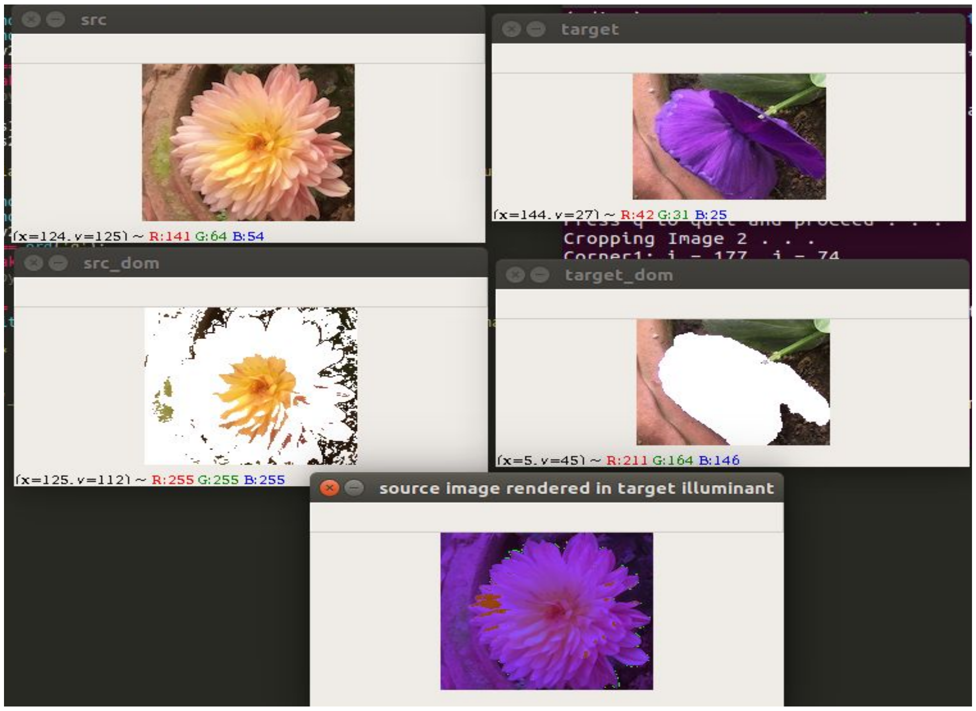
7. `def lab_to_rgb_img(lab, m, n)`- Performs the inverse operation of the above function. **Reconstructs** an RGB image from the **l-α-β** space matrix given row and column dimensions m and n.
8. `def std_devs(lab)`-Takes the *[3, mn]* shaped lab matrix and computes the standard deviation of each row i.e., of each channel (l, α and β). Returns a *[3, 1]* shaped matrix.
9. `def dominant_color_transfer(source, target)`- Both source and target are RGB 3 channel images here. The function **modifies the pixels of the source image** by the series of transformations (mean subtraction, standard deviation scaling, adding mean of target dist.) to make it look like target illuminant. Returns the reconstructed RGB image (modified source image with dominant color transferred from target image).

In `modules.py` file-

1. `def disp_and_quit(img, str = ['image'])`- Displays a single image with window string = 'image' until **q** is pressed.
2. `def click_and_crop(event, x, y, flags, img)`- The mouse callback function which relates mouse left button down and up events to selection of corner points of a rectangle. Note that the *corner points have to be chosen along the principal diagonal from left to right*. **Press** the mouse at one location and **release** at the opposite corner to get the required area cropped.
3. `def crop_region(img)`- Calls cv2.setMouseCallback function using the above callback function. Also displays the cropped image until q is pressed to quit. This cropped images will be used as source and target for later parts.
4. `def display_dominant_color(img)`- Wrapper function for obtaining dominant colors from a BGR image.
5. `def transfer_dominant_colors(img1_cropped, img2_cropped)`- Wrapper function for dominant color transfer for 2 BGR images.

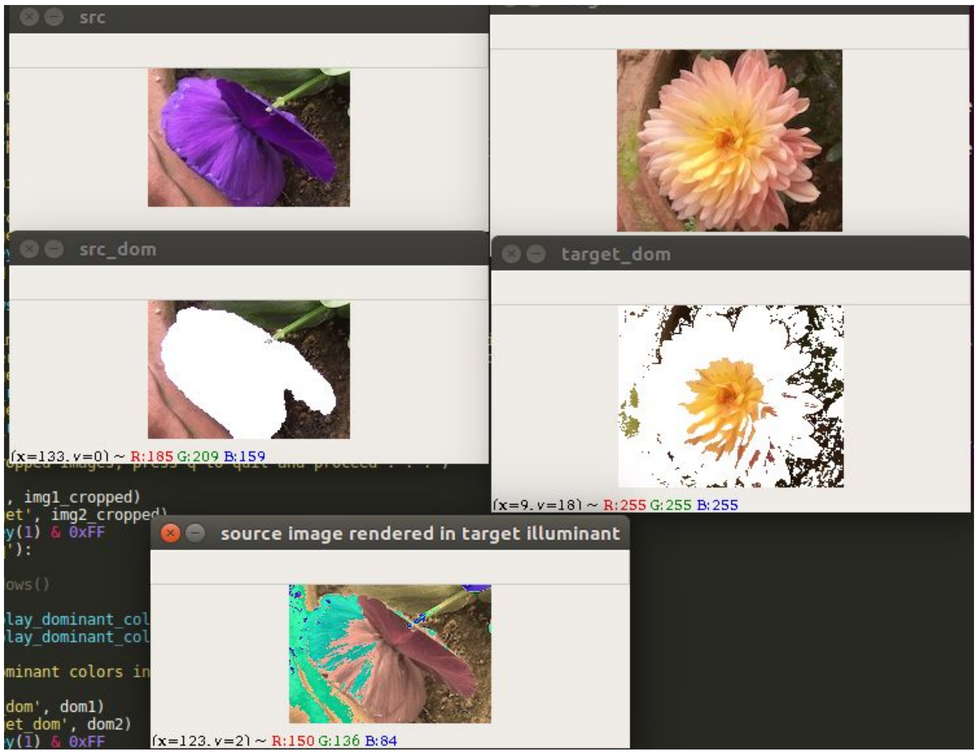
Results

I provide two results here where I transfer colors from a flower to another. Only cropped images, along with their dominant colors and final result are shown. Number of clusters in KMeans algorithm is assumed to be 3 pertaining to foreground, background and noise pixels. Few more results are available in Results folder.



```
(x=124, y=125) ~ R:141 G:64 B:54
(x=144, y=27) ~ R:42 G:31 B:25
Cropping Image 2 . . .
Corner1: i = 177 , j = 74
(x=125, y=112) ~ R:255 G:255 B:255
(x=5, y=45) ~ R:211 G:164 B:146
```

Result-1



```
(x=133, y=0) ~ R:185 G:209 B:159
, img1_cropped)
et', img2_cropped)
y(1) & 0xFF
'):

ows()

lay_dominant_col
lay_dominant_col

minant colors in

dom', dom1)
et_dom', dom2)
y(1) & 0xFF
(x=9, y=18) ~ R:255 G:255 B:255
(x=123, y=2) ~ R:150 G:136 B:84
```

Result-2 (Source and target swapped)

Running Instructions

From the terminal, run the `main.py` file as ”**`python main.py img1.ext img2.ext`**” then follow the instructions displayed in terminal. Img1.ext and img2.txt are supplied as source and target respectively.

Mostly you will be quitting windows by **pressing ‘q’ to proceed** to the next part. While cropping, carefully note that you have to press at one location, then move along principal diagonal of the rectangle (left to right, top to bottom) and release at the other corner point. Once 2 corner points are printed on terminal, you can press ‘q’ to proceed.

Packages Required

Python- numpy, cv2 (I have used version 4.1.0.25, in lower versions mouse callback functions might not work), pandas, sklearn (for K-Means clustering algorithm).