Image Processing: Homework 1

Due on November 2, 2019 at 2:00pm

Yamin Sarcheshmehpour
94109827

Problem 1

Solution

In this problem, I have used power-low transformation to make the photo brighter. To find the best γ I tried multiple γ from 0.1 to 1.0 with the step of 0.1, I have plotted the results in $final01.jpg$.

As you see the best γ would be between 0.5 and 0.6, so I chose 0.55 as the best γ and I plotted the result for $\gamma = 0.55$ in $im01.jpg$.

Describe functions

1. *power_law_tranformation*:

   In this function I have implemented *low_power* trasformation function, which is $255(\frac{x}{255})^{\gamma}$ and I apply it to all pixels.

The code is in $f01.py$.

Problem 2

Solution In this problem, The target histogram is uniform and sI have tried 3 methods for matching histograms.

1. **Devide histogram by r,g,b channels**:

   In this method, for each channel, I will sort pixels by their *pixel_value* and *x_axis* and *y_axis* and start iterate from the beginning of the list, so it would be as follow:

   $$source\_list = [(0, x_{0,1}, y_{0,1}), (0, x_{0,2}, y_{0,2}), ..., (255, x_{255,n}, y_{255,n})]$$

   I have also made a sorted list for target histogram in which the number of 0's is $target\_hist(0)$ and the number of 1's is $target\_hist(1)$, and so on. So it would be as follow:

   $$target\_list = [0, 0, 0, ..., 0, 1, .., 255]$$

   I start iterating *source_list* and for each index i, suppose that it is $(value_i, x_i, y_i)$ I will change the value of the x_i, y_i pixel in the source image to $target\_list[i]$.

   I have implemented this method(each channel iterations) in *devide* function, and I have implemented the method for all channels is *divide_rgb* function in the code, I have also plotted the resulting image in *matched_divide.jpg*

2. **Comulative match histogram by r,g,b channels**:

   In this method, for each channel, I will calculate cumulative CDF for the source image and map source histogram to target histogram as follows:

   For each value i suppose that $cdf\_target(j) <= cdf\_source(i) < cdf\_target(j+1)$ and

   (a) $cdf\_target(j+1) - cdf\_source(i) < cdf\_source(i) - cdf\_target(j)$: I will change all pixels with value i in source to $j+1$.

   (b) $cdf\_source(i) - cdf\_target(j) < cdf\_target(j+1) - cdf\_source(i)$: I will change all pixels with value i in source to j.

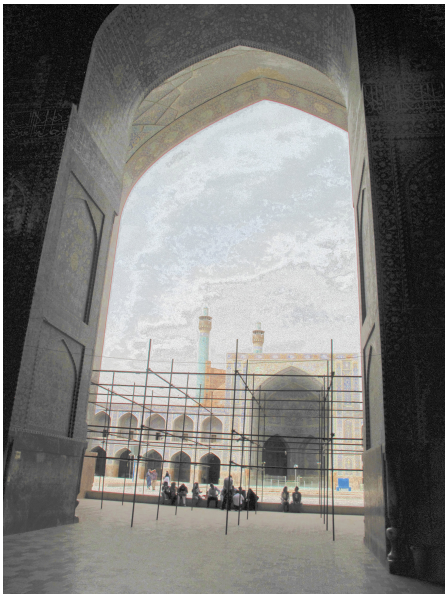
   I have implemented this algorithm(for each channel) in *_one_channel_normal* and the method for all channels is *normal* function in the code, I have also plotted the resulting image in *matched_normal.jpg*

3. **Devide histogram by h,s,v channels**:

In this method instead of using r,g,b channels, I have used h,s,v channels, and I just matched the v channel histogram to target histogram, the same as the first method, made the sorted list for source histogram and the list for target histogram and iterate lists and change the value of source pixels.

I have implemented this method in *divide_hsv* function in the code, I have also plotted the resulting image in *matched_hsv.jpg*

As you see the best result is for the third method, and I have saved its result as $im02.jpg$.



The code is in $f02.py$.

Problem 3

Solution

In this problem for matching channels, I choose a sub-image from original images(img[1000:1500, 1000:1500]), and choose a channel as fixed channel and move others in all directions(with the limit of 100 pixels in each direction) to find the best match.
I have implemented this function in *_match_imgs* function at the code. In this function the inputs are images to be matched, we fix the second image and select its sub-image then for each direction we select the first image sub-image then we calculate *norm_1* of the sub-images and choose the min value in all directions as the best match.
I do this first for the first and the second images, you can see the best match for sub-image for these 2 images in *best_match*10.*jpg* image.

Then I do the same for the second and the third images(I have chosen the second image as the fixed channel) you can see the best match in *best_match*12.*jpg* image.

Then I have found the common space by considering moving direction for each image, and crop the common space at each one and create the aggregated r,g,b image, I have plotted the resulting image in *im*03.*jpg*.

The code is in $f03.py$.
Because of the space limit of cw, I had to delete $BW.tif$, please copy it to the folder before running the code