High Dynamic Range Imaging(Project 1)

Digital Visual Effects(Spring 2023)

Group 5

B09201049 - Lu Guan Ting

B09705024 - Liu Chih Hsin

---

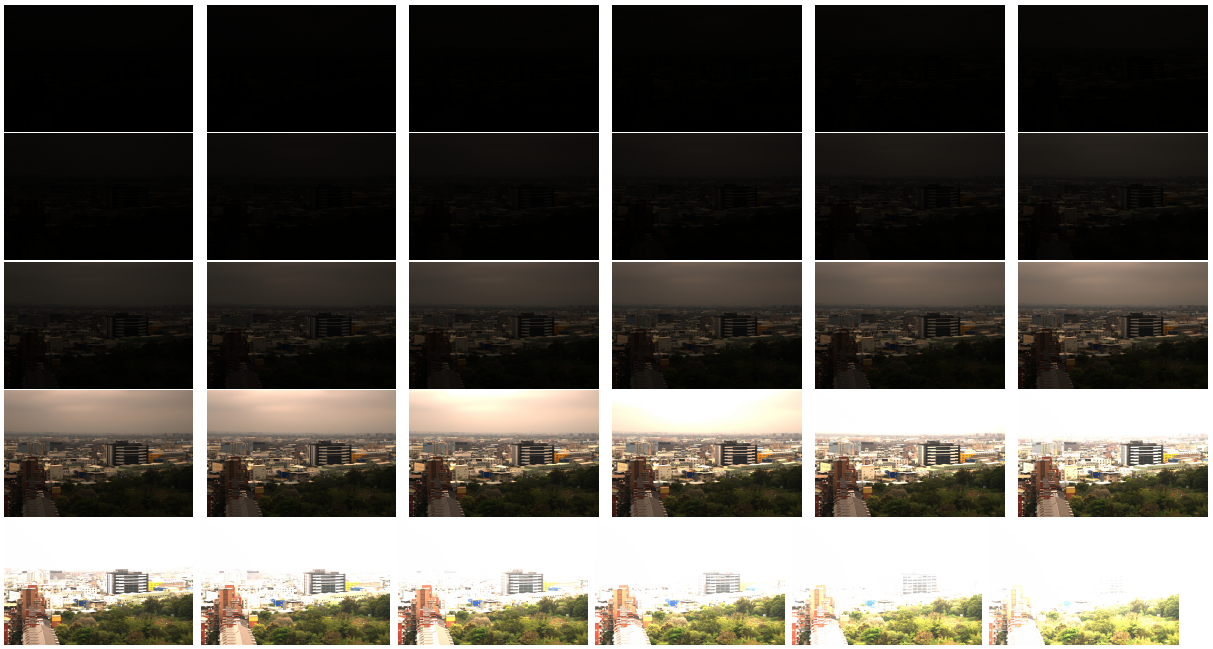
1 Image Collection

Camera configuration

Device: NIKON D90
Focal length: 17mm
F number: f/8
ISO: L1.0
Exposure time: 1/4000, 1/3200, 1/2500, 1/2000, 1/1600, 1/1250, 1/1000, 1/800, 1/640, 1/500, 1/400, 1/320, 1/250, 1/200, 1/160, 1/125, 1/100, 1/80, 1/60, 1/50, 1/40, 1/30, 1/25, 1/20, 1/15, 1/13, 1/10, 1/8, 1/6, 1/5, 1/4

Original images

These 30 images are what we originally took.



Notice that we only chose the 1st, 6th, 9th, 11th, 13th, 15th, 17th, 19th, 22nd and 27th images a our input images below.

2 Image Alignment

To avoid misaligned composition of images, we adopted **MTB Algorithm** here. Therefore, we need to generate image pyramid, construct bit map, and also calculate XOR difference at each candidate offset. Moreover, we used some functions from **OpenCV** and **Numpy** to speed up the algorithm.

1. Convert the input image into greyscale with equation

$$Y = \frac{54R + 183G + 19B}{256}.$$

   This can be found in function **convert_gray**

2. Generate image pyramid and also create bitmap when we get each image in a pyramid
   For generating image pyramid, we need to resize the image to 1/4 smaller. I use **cv2.resize** here and resize its width and height to 1/2 smaller hence generating a image with 1/4 smaller size of the origin image. This can be found in function **generate_pyramids**

3. Construct bitmap
   It's easy to understand the process of constructing a bitmap. We only need to set a threshold which is the median value of the image. If the pixel value is larger or smaller than the threshold, we set the value into 1, or 0 otherwise. This can be found in function **get_bitmap**

4. calculate XOR difference at each candidate offset.
   There is a relationship between each image in different pyramid level:

$$\Delta x_2 = 2\Delta x_1 \pm (1, 0)$$

$$\Delta y_2 = 2\Delta y_1 \pm (1, 0)$$

$$\Delta x_3 = 2\Delta x_2 \pm (1, 0)$$

$$\Delta y_3 = 2\Delta y_2 \pm (1, 0)$$

   Moreover, for each image, we need to consider 9 direction in total. Hence, For each image, we find the offset of that image in different pyramid level. Next on, go to another image and do it repeatedly. During the process of finding the best offset, we need to compute the XOR difference at each offset and taking the coordinate pair corresponding to the minimum difference, We use **cv2.warpAffine** to help generating shifted image for each candidate offset faster. Besides, We also use **np.logical_xor** to efficiently compute the XOR difference. . This can be found in function **calculate_offset**, **shift_img**, and **get_best_shift**

In the processes mentioned above, ignoring some noises(we choose "median ± 2" here), we derived following grey images.

As a result, we picked up one of them(We choose the 6th image here) as a reference image and calculated the offsets with respect to the other images, and the aligned image was yielded. The offsets for each image are [0, 0], [0, 0], [0, -1], [0, -1], [0, 0], [0, 0], [0, -1], [0, -1], [0, 0], [0, -1] respectively. Note that relating codes are in **MTB_alignment.py**.

3 Generate HDR Image

Due to the linear property of digital value function saved in RAW image, we can solely divide sensor exposure with exposure time to reconstruct the target radiance function. That is,

$$E_i = \frac{X_{ij}}{\Delta t_j} \qquad \text{(for i-th point and j-th image)}$$

However, we found that the calculated radiance E_i was not always the same among these 10 images we chose, so we sampled their average value to reconstruct our radiance map at first. Nevertheless, we also encountered the problem that we initially selected too much low shutter-speed images, so we wondered if we could only used the average as our basis of radiance map due to the fact that the high value might overrate the map. Hence, we also multiply this radiance map with **sum(exposure_time)** to reduce the effect of overrating to some degree.

We have described a lot about constructing a radiance map above, though merely assembling HDR image without tone mapping generated the image with too high radiance to be observed. Here it is.

Note that relating codes are in the function **map_to_hdr** in **hdr.py**.

4 Tone Mapping

Due to some high radiance pixels which are not observable, we map the radiance of these pixels to the range which is visible for us with tone mapping. However, there are various algorithms can help us implement tone mapping, and we adopt **Reinhard tone mapping method** in the beginning. Specifically, we compute luminances as following

$$\overline{L}_w = \exp\left(\frac{1}{N}\sum_{x,y}\log(\delta + L_w)\right)$$

where δ is a small value. Then, we turn it into

$$L_m = \alpha \times \frac{L_w}{\overline{L}_w}$$

to derive displayed luminances

$$L_d = \frac{L_m\left(1 + \frac{L_m}{L_{white}^2}\right)}{1 + L_m}$$

Then, we derived the image below.

We also try another tone mapping methods packed in **OpenCV**. This is the image using Drago's tone mapping algorithm which can be used through function **cv2.createTonemapDrago()** and its processor. Here we set its gamma to be 1.2, saturationton be 1.0, bias to be 0.7. And scale 3 times in the end. The derived image is shown as below.

Compared to these images above, we can see that there is a color distortion in the image we derived. As a result, we revised our function **Reinhard_tonemap** in **tone_mapping.py**. Our goal is to do some color correction, so we provide a raw image as reference. We multiply a correction coefficient to each channel to make the color intensity of the resulting image similar to our the color of the reference image. Then, we derived the image below.

Note that relating codes are in the **tone_mapping.py**.

5 Results

We initially selected all of images as inputs, and the following image was generated, which had been processed with alignment, HDR assembling and tone mapping. The image on the left-hand side is the resulting image with color correction and there is no color correction performing on the other side.

However, we found that ten images were sufficient to attain the ideal HDR image. Hence, we chose the 1st, 6th, 9th, 11th, 13th, 15th, 17th, 19th, 22nd and 27th images to generate our HDR image. Then, we derived the image below. The image on the left-hand side is the resulting image with color correction and there is no color correction performing on the other side.

parameters:
alignment = 1 (whether aligned = True)
δ = 0.000001 (tone mapping)
α = 0.5 (tone mapping)
L_{white} = 1.3 (tone mapping)
noise = 2 (MTB alignment)
level = 4 (number of octave in image pyramid in MTB alignment)

6 Review

Lu Guan Ting

In this project, I have learned not only the reconstruction of radiance function but also the implementation of MTB alignment algorithm as well as tone mapping, and such techniques absolutely help me reassemble images to derive a new one containing more details and fitting into our visual system. In conclusion, I dig deeper into photography in this topic, and relating knowledge is really useful.

Liu Chih Hsin

In this project, several algorithms are put into practice to generate aligned and tone mapped HDR images, and I realize which algorithms are suitable to create a HDR image for different photos. Furthermore, I remember that I thought that there were some mistakes during the generation of HDR image at first, but the truth is that the image would become much better after

tone mapping, which plays an crucial role in this project. That is, tone mapping makes details stand out evidently. Hence, I really get a sense of accomplishment from the implementation of tone mapping. Nevertheless, compared with the work generated via the inbuilt functions in **OpenCV**, our images still lack a few details, and the issue about color distortion can not be well fixed. However, I believed that we can derive images of higher quality, if we dedicated ourselves to improving algorithms and adjusting better parameters.

References

[1] MTB alignment
http://www.anyhere.com/gward/papers/jgtpap2.pdf

[2] Tone mapping Methods
Reinhard's Tone Mapping
https://www-old.cs.utah.edu/docs/techreports/2002/pdf/UUCS-02-001.pdf
OpenCV class: TonemapDrago
https://docs.opencv.org/4.x/da/d53/classcv_1_1TonemapDrago.html