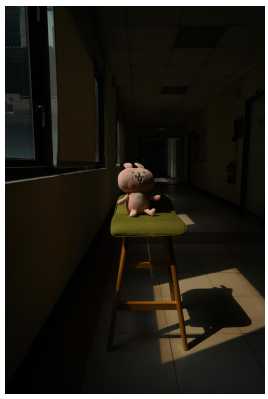
Project #1: High Dynamic Range Imaging

- 資管四 B06705028 朱紹瑜
- 資工所碩一 R09922063 鄭筠庭

Introduction

In this project, we implemented **HDR image** with **image adjustment** and **tone mapping**. For demo, we ran our code on the following image set:

1/10	1/13	1/15	1/20
			
1/25	**1/30**	**1/40**	**1/50**
			
1/60	**1/80**	**1/100**	**1/125**
			

Usage

Run the following command to reproduce the result.

```
python3 alignment.py
python3 hdr.py
python3 toneMapping.py
```

Note that `numpy`, `cv2`, and `PIL` are required.

Image Alignment: Median Threshold Bitmap (MTB)

Image alignment methods: Ward's MTB algorithm
Reference paper：http://www.anyhere.com/gward/papers/jgtpap2.pdf

Step 1: Get grayscale images
Use `cv2.imread` to read in images with different exposure time. The name of each image is its corresponding shutter time. Turn the images into gray scale by $Y = (54R + 183G + 19B)/256$.

Step 2: Build an image pyramid
Run the recursive `GetExpShift()` function. This recursive function will utilze multi-scale technique and generate an image pyramid with log2(max_offset) levels past the base resolution. For each smaller level in the pyramid, we filter the previous grayscale image down by a factor of two in each dimension.

Step 3: Find the minimal error between 9 neighbors
Here we use the first image as baseline.
Call `ComputeBitmaps()` to get *threshold_bitmap* and *exclusion_bitmap* of the baseline image and target image. The threshold for generating black-and-white bitmap is the median value of all pixels. If the threshold value is lower than 20, the result bitmap will have lots of noises, so we set the threshold lower boundary to be 20. The exclusion bitmap sets the pixel to balck if its value is between $(\text{median} - 4)$ to $(\text{median} + 4)$, and sets to white otherwise.

Shift the target image to its 9 neighbors' location with `cv2.warpAffine`, and compare the error value for each location.
The fundamental idea is to calculate `(BaselineImage) XOR (TargetImage) AND (exclusion_bitmap)` as error value. XOR for taking difference, AND with exclusion maps to reduce noise.

Step 4: Shift the image to complete alignment
Find the shift direction that creates the least amount of error, store the shifting value and pass it down the image pyramid. Repeat Step 3 until we get the shifting value for original sized image.
Use `cv2.warpAffine` to align the soure images and save them to the output folder.

HDR

Reference paper: [Recovering high dynamic range radiance maps from photographs]

Step 1: Sample pixels
The original image set contains 12 images with shutter speed ranging from 1/125 to 1/10. To sample pixels with various irradiance, we target at the image with the largest value variance. Within the targeted image, we select pixels with values locating at the 0, 2, ..., 100 percentile. Moreover, to be invulnerable of the appended margin after alignment, pixels on the edges are avoid. A total of 51 pixel positions are sampled.

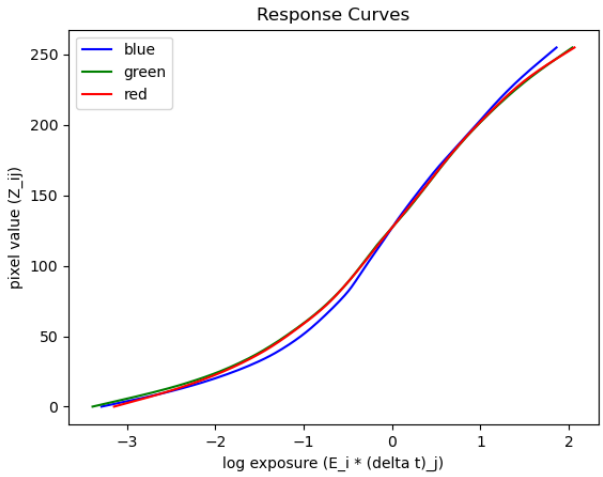
Step 2: Compute Log Inverse of the Response Curve
To recover the response curve, we minimize the following function:

$$\min_{g,E} O = \min_{g,E} \sum_{i=1}^{N} \sum_{j=1}^{P} \{w(Z_{ij})[g(Z_{ij}) - \ln E_i - \ln \Delta t_j]\}^2 + \lambda \sum_{z=0}^{255} [w(z)g''(z)],$$

where Z_{ij} is the intensity of the i-th sampled pixel in the j-th image, E_i is the irradiance at the position of the i-th sampled pixel, Δt_j is the shutter speed of the exposure time of the j-th image, λ is the smoothing coefficient, $g(\cdot)$ is the log inverse of the response curve, and $w(\cdot)$ is the weight function with

$$w(z) = \min\{z, 255 - z\}.$$

We solve the optimization problem above with the least square method by constructing a linear system as mentioned in the lecture. The resulting response curve is shown below.



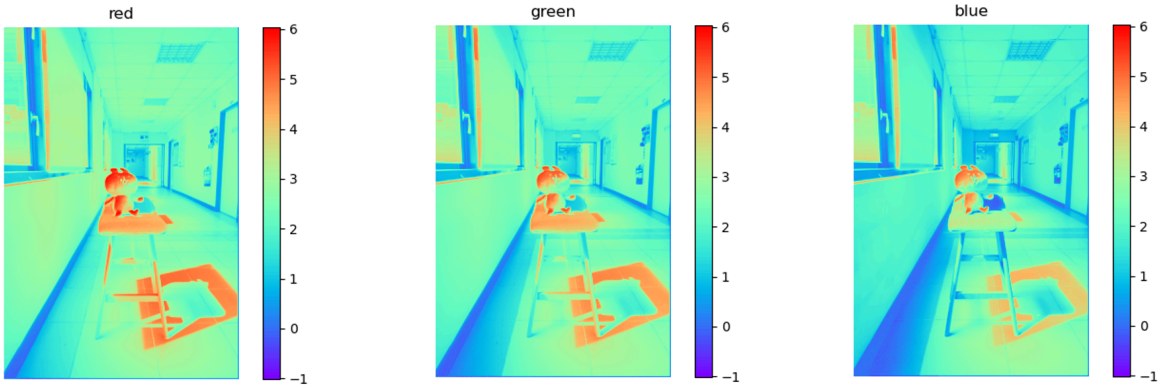
Step 3: Reconstruct the radiance map
With the computed response curve, we can reconstruct the radiance map based on the following equation:

$$\ln E_i = \frac{\sum_{j=1}^{P} w(Z_{ij})[g(Z_{ij}) - \ln \Delta t_j]}{\sum_{j=1}^{P} w(Z_{ij})}$$

$$E_i = \exp\left(\frac{\sum_{j=1}^{P} w(Z_{ij})[g(Z_{ij}) - \ln \Delta t_j]}{\sum_{j=1}^{P} w(Z_{ij})}\right).$$

Note that if the sum of weight equals to 0, we compute the average with equal weight.

The following figures show the computed radiance map of R, G, B channel.



Tone mapping: Fast Bilateral Filtering

Step 1: Get intensity layer
Split the RGB channels. Calculate the intensity using the equation `intensity = 1/61 * (20 * R + 40 * G + B)`, and get the log10 intensity layer value.

Step 2: Bilateral filter
Go through every pixels using two *for loop*, and call the `pixelBilateralFilter()` to calculate the smoothed value for each pixel.
We need to calculate the *denoised intensity* using pixels within *radius*. Gaussian function `ws` considers spatial domain and `wr` considers range domain to be its standard deviation value. Add up `ws * wr` to be the denoised intensity denominator, while denoised intensity fraction is the sum of `w*input_image[k,l]`. Divide these two number to get the bilateral filtered image (log_base).

Step 3: Compress and Normalize
Follow the algorithm to get the detail image, scale, intensity image. Calculate the R, G, B value by multipling 10^log(output intensity).

```
log(detail) = log(input intensity)-log(base)
compressionfactor = log5/(max(log_base) - min(log_base))
log(absolute scale) = max(log_base) * compressionfactor
log(output intensity) = log(base)*compressionfactor + log(detail) -
log(absolute scale)
output = RGBchannel * 10^(log(output intensity))
```

Finally, normalize the 3 channels and merge them together. Adjustment intensity with one original image as template.

The following figure is our HDR image result.

(Figure: Tone mapping result)