- Experiments a.
  - What is the meaning of this figure?

    Figure 2 之左圖為多張不同曝光時間的 bracketing images 的 pixel 值與曝光時間取 log 的關係。同一條線代表各張不同曝光時間的 bracketing images 之同一個 pixel 位置的連線，以左圖為例，因為每條連線有五個點，所以有五張 bracketing images，而三條連線表示有三個不同位置的採樣 pixel。可發現曝光時間越長，pixel value 值越大，呈現正相關。

    欲求得 Figure 2 之右圖，需先將 Implementation 中 1.1 Camera response calibration 中的 equations 用 numpy.linalg.lstsq()求出$g(Z_{ij})$後，再以Z_{ij}(本來就是已知的 pixel value)與$g(Z_{ij})$(Camera response)作圖即可得到 Figure 2 之右圖。從 Figure 2 之右圖可發現，當曝光時間一直變大，並且到達一個閾值之後，其 pixel value 都會落在 256 並且不再增加。

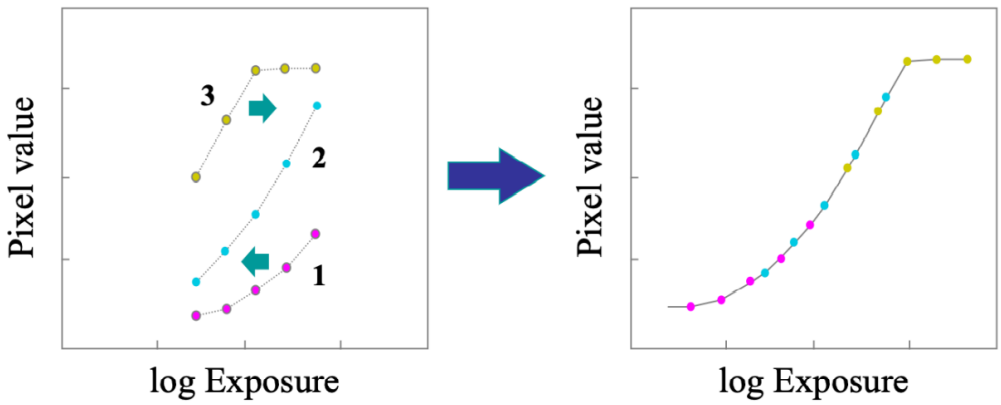
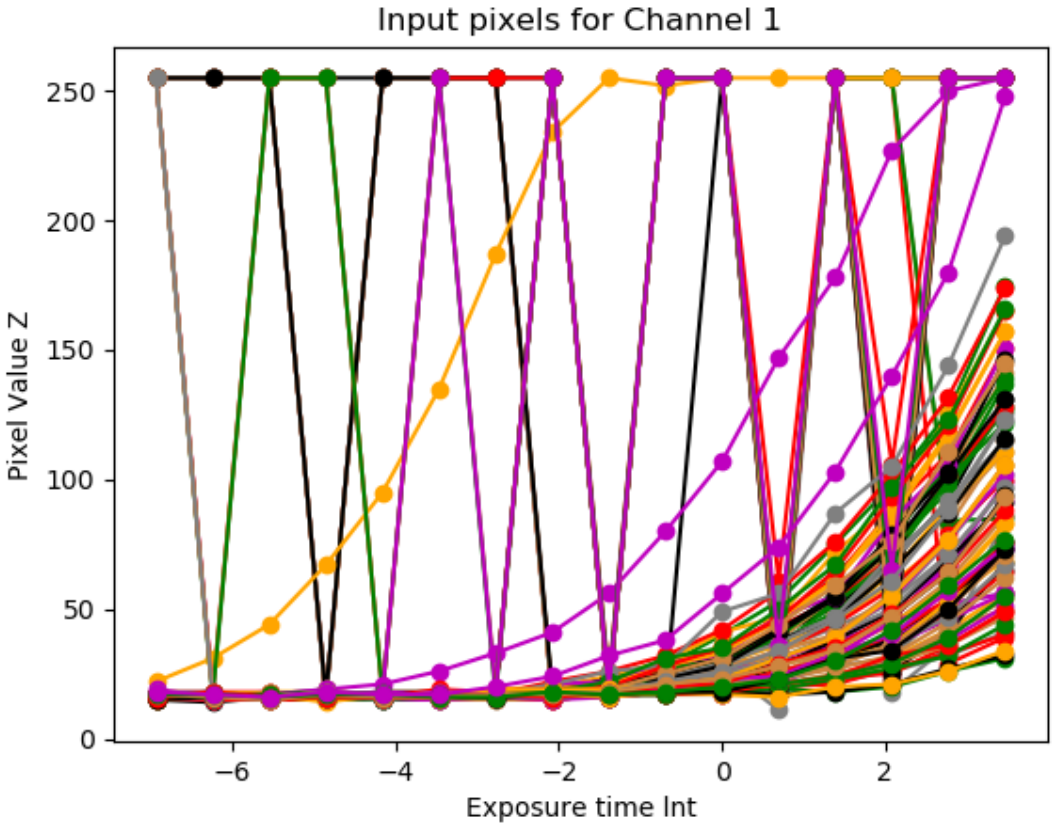
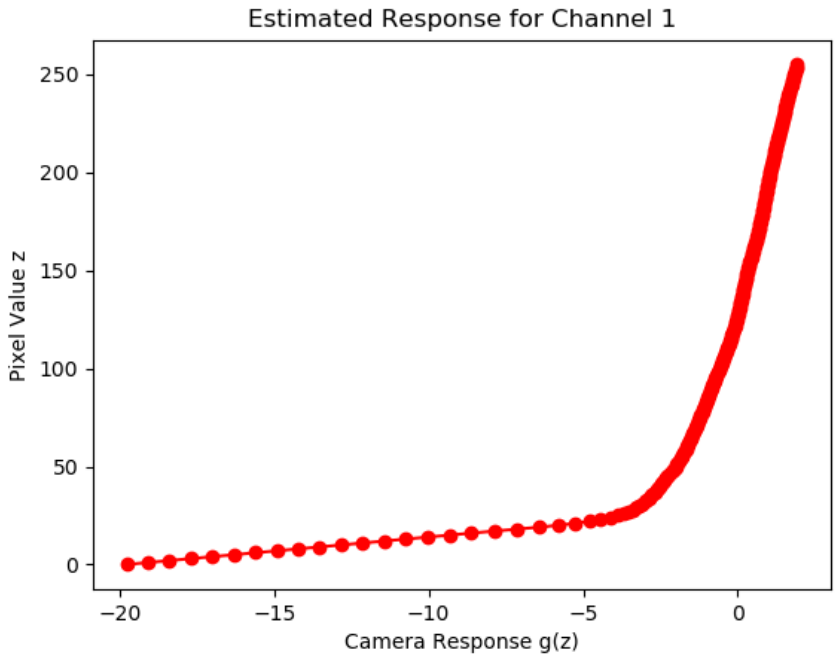


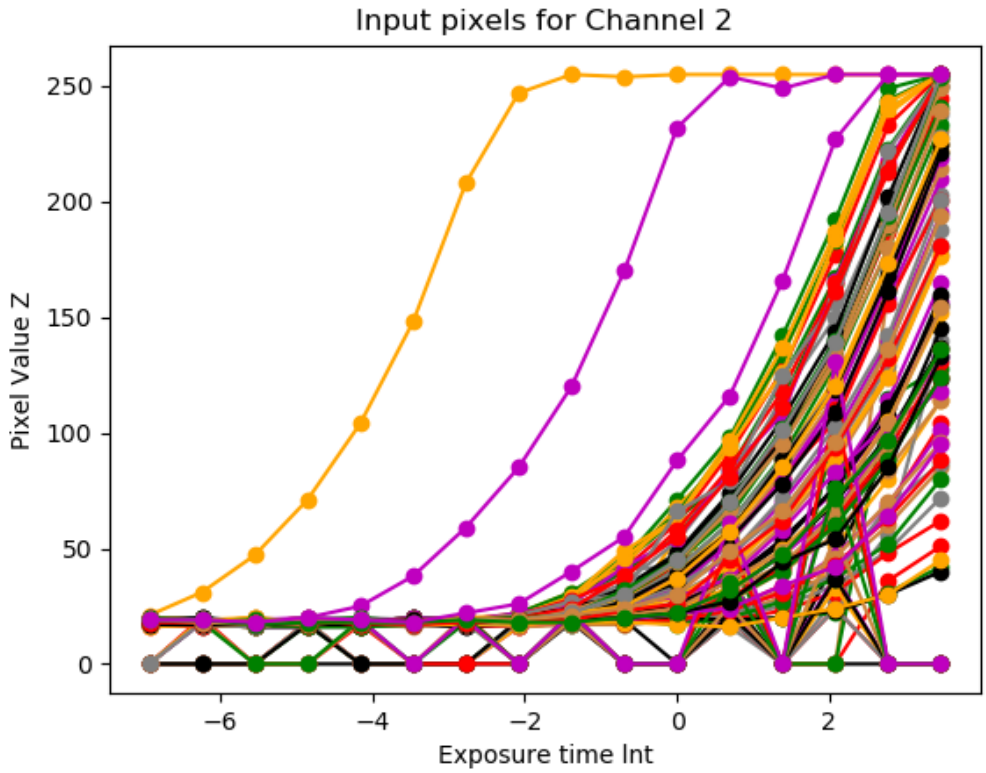
Figure 2: Radiometric Calibration

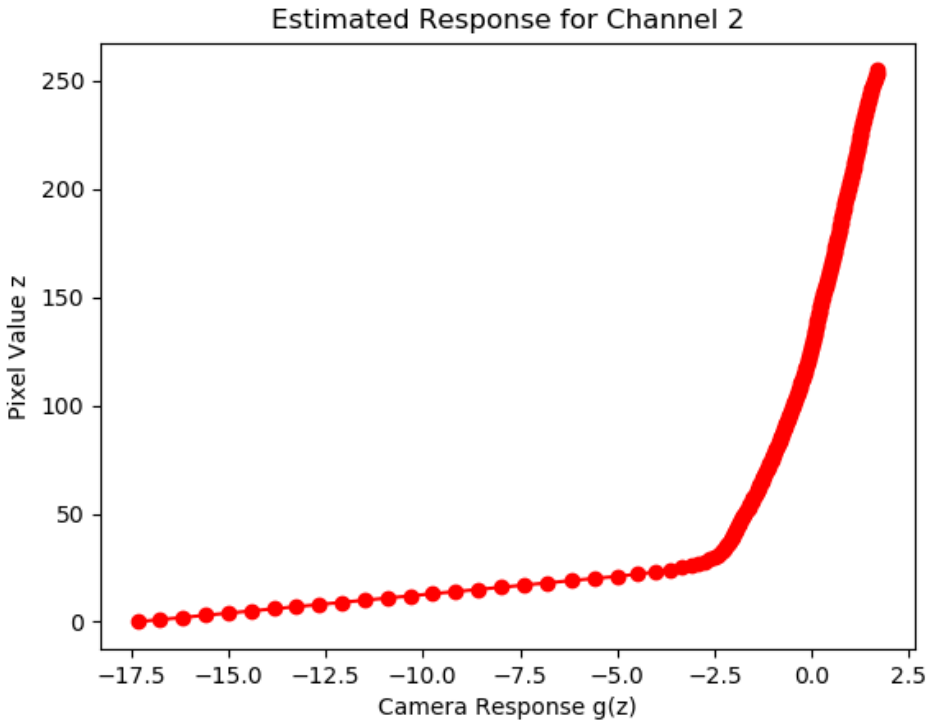
  - Try to plot a same figure for memorial image set

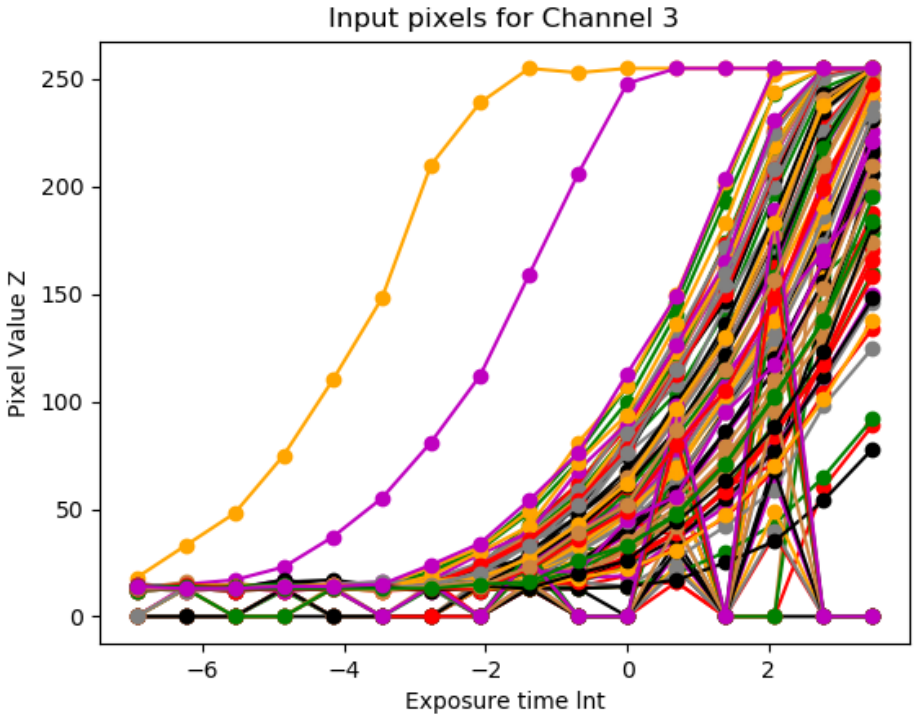
    下面六張圖片儲存在” result/Experiments/a/”資料夾中，並會在執行 cr_calibration.py 時自動秀出並儲存。

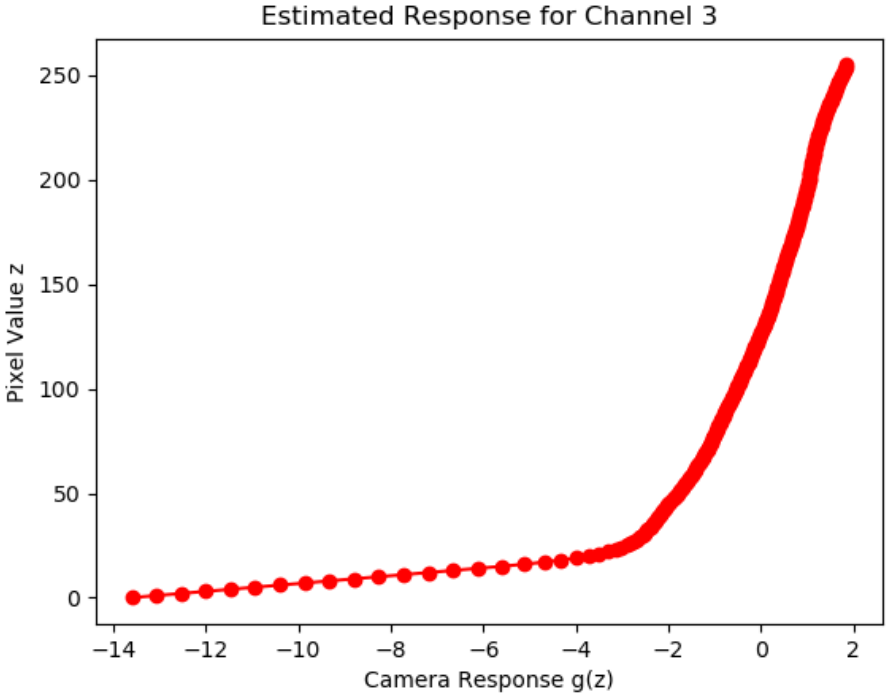

Input pixels for Channel 1
Pixel Value Z
Exposure time lnt
250
200
150
100
50
0
−6
−4
−2
0
2


Estimated Response for Channel 1
Pixel Value z
Camera Response g(z)
250
200
150
100
50
0
−20
−15
−10
−5
0


Input pixels for Channel 2
Pixel Value Z
Exposure time lnt
250
200
150
100
50
0
−6
−4
−2
0
2


Estimated Response for Channel 2
Pixel Value z
Camera Response g(z)
250
200
150
100
50
0
−17.5
−15.0
−12.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5


Input pixels for Channel 3
Pixel Value Z
Exposure time lnt
250
200
150
100
50
0
−6
−4
−2
0
2


Estimated Response for Channel 3
Pixel Value z
Camera Response g(z)
250
200
150
100
50
0
−14
−12
−10
−8
−6
−4
−2
0
2

- How do you plot the figure and why do you choose to do so?
  首先將 16 張不同曝光時間的照片之 pixel value 讀取出來，16 張照片分別各有 768*512 個 pixel，並且都是 3 channels，接著對這些 pixels 位置做採樣，採樣方式為二維每隔 64 個 pixel 位置採樣一次，故這樣總共會採樣出(768*512)/(64*64) = 96 個 pixel 的位置，換句話說這 16 張不同曝光值的照片之 3 個 channels 的採樣點都是這 96 個 pixel 位置。接著用各個 channel 採樣出來的 pixel values(16 張照片*96 個 pixel 位置)，讓 16 張相同 pixel 位置的 pixel value(16,)對其曝光時間(16 種曝光時間)作圖得一連線(故 1 條連線上有 16 個點)，並且用迴圈作 96 次，可得 96 個連線，即可得到"Input pixels for Channel X"的圖。然後將 function estimateResponse()回傳的 Camera Response g(z)對 Pixel Value z(0-255)作圖得"Estimated Response for Channel X"。另一種方式得到"Estimated Response for Channel X"，是透過 function constructRadiance()得到E_i再對E_i取對數得 lnE_i，並採用(lnE_i+lnt_j)對 Pixel Value z(0-255)作圖亦是一種方式，但是我設計的 function constructRadiance()時間複雜度較高，所以我沒有採用此做法。

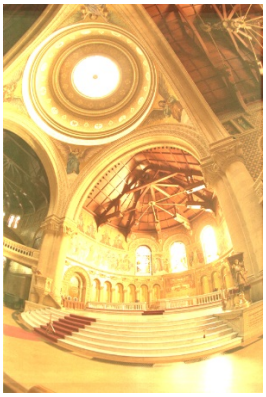
- Experiments b.
  - Scaling factor in global tone mapping

$$\log_2 \hat{X} = s(\log_2 X - \log_2 X_{max}) + \log_2 X_{max},$$

$$X_{max} = \max_{i,j}(X(i,j)),$$

Implementation 中 1.2(Global tone mapping)設定的 s 值為 1，也就是最後得到的radiance $\hat{X}$等於 X，而當 s 值為 0 到 1 之間時，log(X)-log(X_{max})<log($\hat{X}$)- log(X_{max})，所以$\hat{X}$大於 X，也就是得到的圖亮度較原本高，且 s 越接近 0 得到的圖越亮。另外當 s 值大於 1 時，得到的圖就會越暗，且 s 越大會越暗。下圖分別為 s=0.6, 0.8, 1.0, 1.2, 1.4 的實驗結果，實驗程式碼在 tm.py，實驗圖片在" result/Experiments/b/"資料夾中。

s=0.6



s=0.8

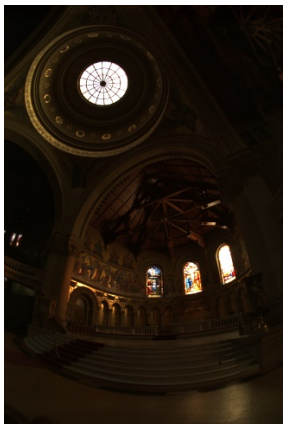
---

s=1.0

s=1.2

---

s=1.4



- Scaling factor in local tone mapping with Gaussian filter

$$L'_B = (L_B - L_{max}) * \frac{scale}{L_{max} - L_{min}}$$

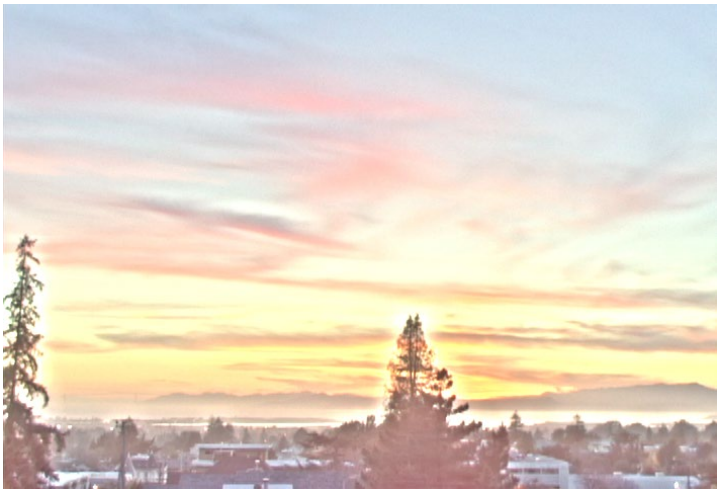
從上式可以發現，$(L_B - L_{max})$為小於等於零的固定值， $(L_{max} - L_{min})$為大於等於零的固定值， 所以在 scale 大於等於 1 的情況下，L'_B是一個小於等於零的值，故 scale 越大，L'_B的絕對值就越大，也就是L'_B就越小，並且在 detail layer 不改動的情況下，Reconstruct color map C 就會越小，再套一層 gamma correction 也會越小，故亮度較暗，；反之若 scale 越小，則亮度會變亮。

而且可以發現是各個 pixel 位置都做一次上式的運算，故L_B比較小的 pixel 位置，$(L_B - L_{max})$較大，此時L'_B的絕對值就越大，也就是L'_B越小，換句話說就是同一張圖越暗的區域，經過 local tone mapping 變暗的幅度會越大。下圖分別為 s=2,6,10 的實驗結果，實驗程式碼在

tm.py，實驗圖片在”result/Experiments/b/”資料夾中。

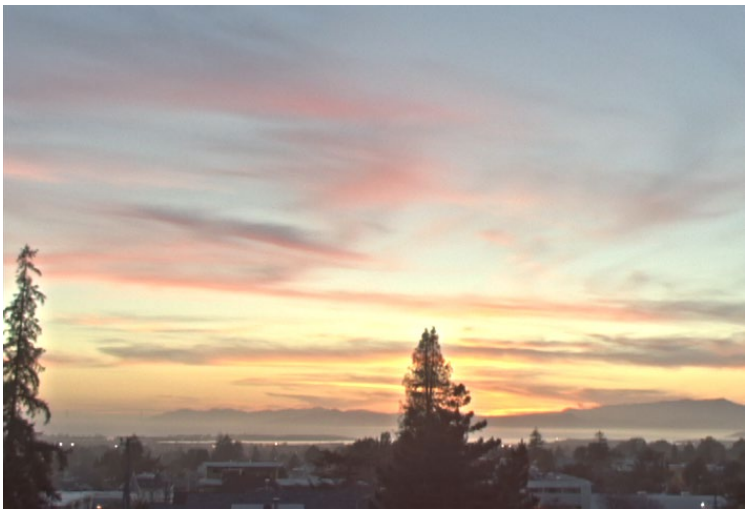
---

s=2



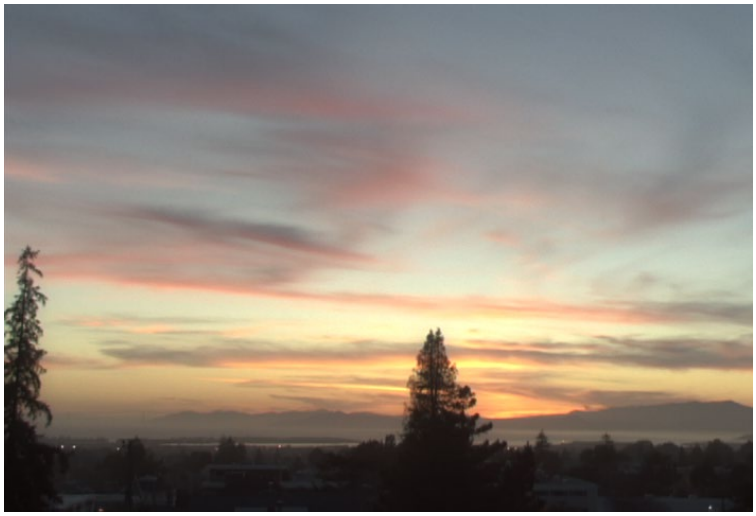
---

s=6



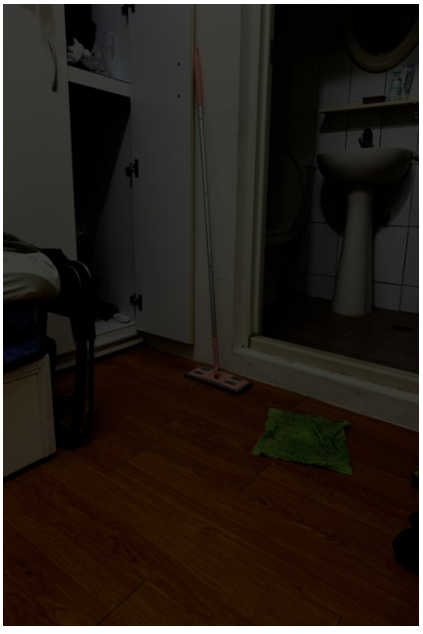
---

s=10

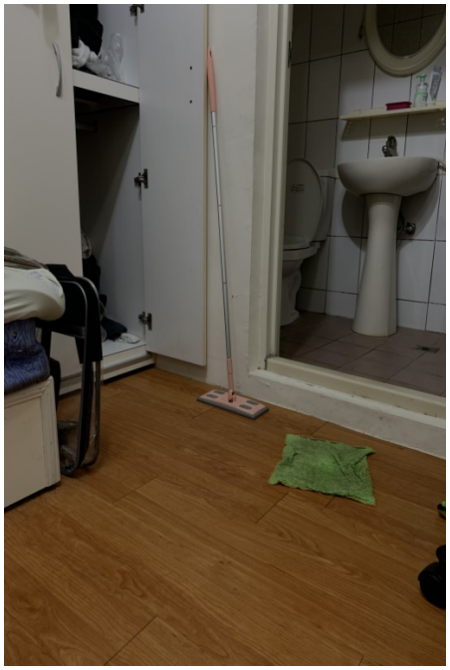


- Experiments c.
  - How do you choose the scenes, photograph setup, and parameters in the flow? and why? Do you apply any additional steps? Why?
    我在宿舍用 iphone 手機拍攝了三張曝光時間不同的照片(用手機腳架以保持三張照片的 pixel 位置對應同樣的拍攝物體)，曝光時間分別是 1/120, 1/60, 1/30，並且為了縮短運算時間，將原照片從原本的 (3024*4032) resize 成(512*768)，並且採用 interpolation=cv.INTER_AREA，它是用 resize 做壓縮較可減少資訊流失的方式，然後用 png format 儲存(原本手機拍照是 jpg format)。如下圖所示(存放於/TestImg/experiment_c/資料夾中):

    曝光時間 1/120 秒:



曝光時間 1/60 秒:



曝光時間 1/30 秒:

這個拍攝場景的選擇，是希望在同一張照片中有亮部也有暗部，從上圖可發現較暗的地方在照片的右邊偏上(廁所的位置)，而較亮的區域是廁所外有室內光線照射的地方。而我的實驗流程是：

1. Whitebalance -> Global Tone Mapping -> 產出 gtm.png (存放於 result/Experiments/c/資料夾中)

2. Whitebalance -> local Tone Mapping with Gaussian Filter -> 產出 ltm_gaussian.png (存放於 result/Experiments/c/資料夾中)
3. Whitebalance -> local Tone Mapping with bilateral Filter -> 產出 ltm_bilateral.png (存放於 result/Experiments/c/資料夾中)

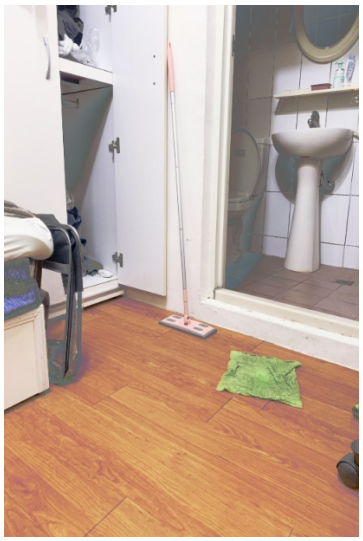
程式碼皆在 code/ experiment_c.py。白平衡定義'known to be white region'是透過最暗的那張拍攝圖，觀察其 RGB value(768*512*3 int numpy array)，找到 RGB 三個 channel 的值都很大的 pixel 位置，大概是在拍攝圖的左邊偏上部分(衣櫥與牆壁處)。比較特別的部分是 Global Tone Mapping，如果照著原本 Implementation 1.2 的作法，會因為 gamma correction 後的值全部都大於一，導致乘上 255 都會大於 255，最後再校正回 255，所以整張圖片呈現全白，為了解決此問題，我將 gamma correction 後的值做 Min-Max Normalization 後，確保值都落在 0-1 之間，再乘上 255。Min-Max Normalization 公式如下：

$$\frac{X - X_{min}}{X_{max} - X_{min}}$$

X_{max}與X_{min}是三個 channel 各別的最大值與最小值，X即為該 channel 上原本的計算值。而使用的參數皆按照 HW1.py 內所設的，只有 local tone mapping 的 scaling factor 是我自己調出來的，我將其調整成 4，讓其不會過量或過暗。最後結果如下：

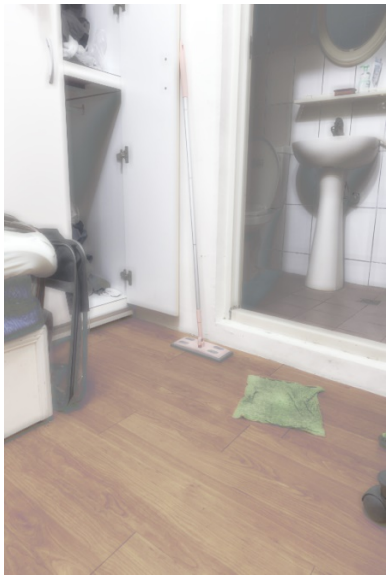
---

1. Whitebalance -> Global Tone Mapping -> 產出 gtm.png (存放於 result/Experiments/c/資料夾中)



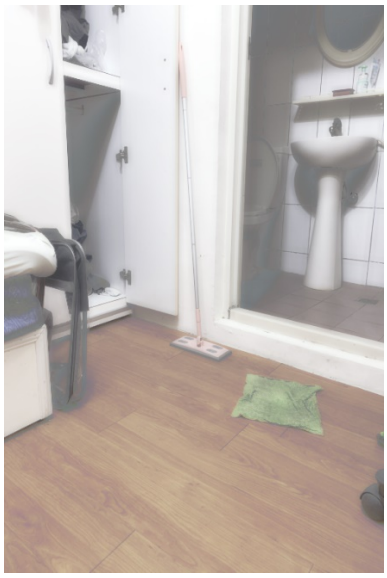
---

2. Whitebalance -> local Tone Mapping with Gaussian Filter -> 產出 ltm_gaussian.png (存放於 result/Experiments/c/資料夾中)



---

3. Whitebalance -> local Tone Mapping with bilateral Filter -> 產出 ltm_bilateral.png (存放於 result/Experiments/c/資料夾中)



可以發現 Global tone mapping 做得還不錯，但是 local Tone Mapping 不論是採用 Gaussian Filter 或 bilateral Filter 都會有一點糊糊、波動的感覺，推測可能的原因是 Filter Windows Size 或是 sigma_s 與 sigma_r 沒有調整得宜。另一個可能做不好的原因是 Camera response calibration 沒有做好，因為只有拍攝三張不同曝光時間的照片，造成樣本數目不足，沒有接近真正的 Camera response，但是我後來推測不太可能是這部分沒做好，因為 Global tone mapping 做得不錯，並且也是用同一組 Camera response。

- Free study – Problem 3
  - Assumption

    理論上我認為可以透過不同組的 bracketing image sets 求出同一個

camera response，就算不是求出同一個 camera response，其求出的 camera response 也會很接近。會有上述的推測，是因為我覺得用 least square solution 求 camera response 與 Machine Learning 中用 least square solution 求 linear regression 的係數有異曲同工之妙，不同的 training data set 可能得出不同的 linear regression function，但是都接近於 ground true solution。套用此觀念在這邊，不同的 training data set 就像是不同組的 bracketing image sets。

而在已知 camera response 的情況下，若只用單張照片我認為無法建立好的 radiance map，或是說可能會跟好的 radiance map 差距甚大。計算 radiance map 的公式如下：

$$\ln E_i = \frac{\sum_j w(Z_{ij})(g(Z_{ij}) - \ln \Delta t_j)}{\sum_j w(Z_{ij})}$$

j 的意思是多張照片中每張照片的編號，如果只有一張照片，也就是 j 等於 1，換句話說就是可以消掉 w(紅色斜線處)，這樣就完全沒有達到加權平均的效果。

- Justification (程式碼在 code/ free_study_p3.py)

  為了驗證上面的第一個推論，我將 TestImg/memorial/資料夾中的 16 張照片評分成兩組，因為此 16 張編號由小到大的順序是由亮而暗，所以我將其中奇數編號的 8 張放在 TestImg/free_study_p3_set1/資料夾中，剩下偶數編號的 8 張放在 TestImg/free_study_p3_set1/資料夾中，用基數偶數的方式來分類，是為了讓資料集內有亮的也有暗的照片，若是單純用編號的前 4 張與編號後 4 張做分類，這樣會變成有一照片集偏亮；另一照片集偏暗，最後算出來的 camera response 可能不夠 general。下面方便說明，這邊先假設 TestImg/free_study_p3_set1/所求得的 camera response 為 camera response 1；TestImg/free_study_p3_set2/所求得的 camera response 為 camera response 2。而我這邊做了兩組 radiance。其中一組 radiance 是由 camera response 1 與 free_study_p3_set1 求得；另一組 radiance 是由 camera response 2 與 free_study_p3_set1 求得。此兩組 radiance 差別在使用不同的 camera response，最後我將此兩組 radiance 取 MSE，得 1.0293611。這邊就可以驗證出由不同 bracketing image sets 產生的 camera response，可以用來產出 radiance，並且跟原本自身 bracketing image sets 產出的 camera response 不至於差上太多，當然如果要讓上面實驗的 MSE 更小的話，每個 bracketing image sets 內的照片量需要更

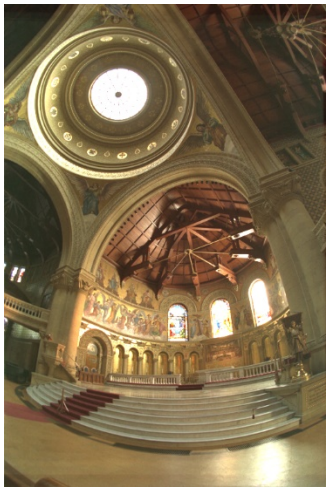
多，這樣求得的 camera response 就可以更 general & robust。
另外為了驗證上面第二個推論，我的作法是用 TestImg/memorial/資料夾中的 16 張照片算出的 camera response 當作已知的 camera response，並且在計算 radiance map 時，都只用 16 張照片中的其中一張做計算，計算了 16 次得到 16 個 radiance maps，再跟原本用 16 張照片計算出的 radiance map 分別取 MSE，得到的 16 個 MSE 值都非常大，值大概落在 106 至 108 之間(實驗程式碼在 code/free_study_p3.py)，可見驗證我的假設。

- Free study – Problem 8
  - Assumption

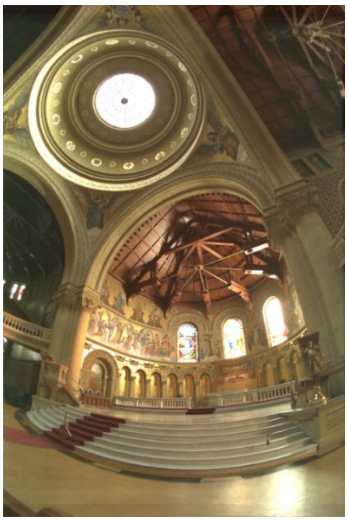
    Global tone mapping 是將三個 channel 的 radiance 分別在 log domain 做線性壓縮，當 scaling factor 的值越大，radiance 被壓縮得越多，也就是會變得越暗。而 White balance 作用是調整每個 pixel value 之 RGB 三色在'known to be white region'的比例，讓這樣的色彩比例在任何光源打進來時，看起來還是有白色的感覺。我覺得這兩個 function 彼此是互不干涉，所以推測 Global tone mapping & White balance 在實作上誰先誰後的結果都一樣。
  - Justification

    為了驗證我的假設，我將原本 test_HW1.py 內的 test_globalTMwb() function 流程做修改，從原本的 White balance -> Global tone mapping -> gamma correction 改成 Global tone mapping -> whiteBalance -> gamma correction，兩個與結果與 golden 相比所得之 PSNR 都是 51.08622831892766，可見 Global tone mapping & White balance 在實作上誰先誰後的結果都一樣。下面兩張圖是我的實作結果。程式碼在 /code/free_study_p8.py，結果圖在 result/Free_Study/problem_8/資料夾中。

    White balance -> Global tone mapping -> gamma correction:

Global tone mapping -> whiteBalance -> gamma correction:



需特別注意的是，gamma correction 必須特別拉出來做，不能放在 Global tone mapping 中，否則順序錯誤無法得到兩張圖一模一樣的結果。