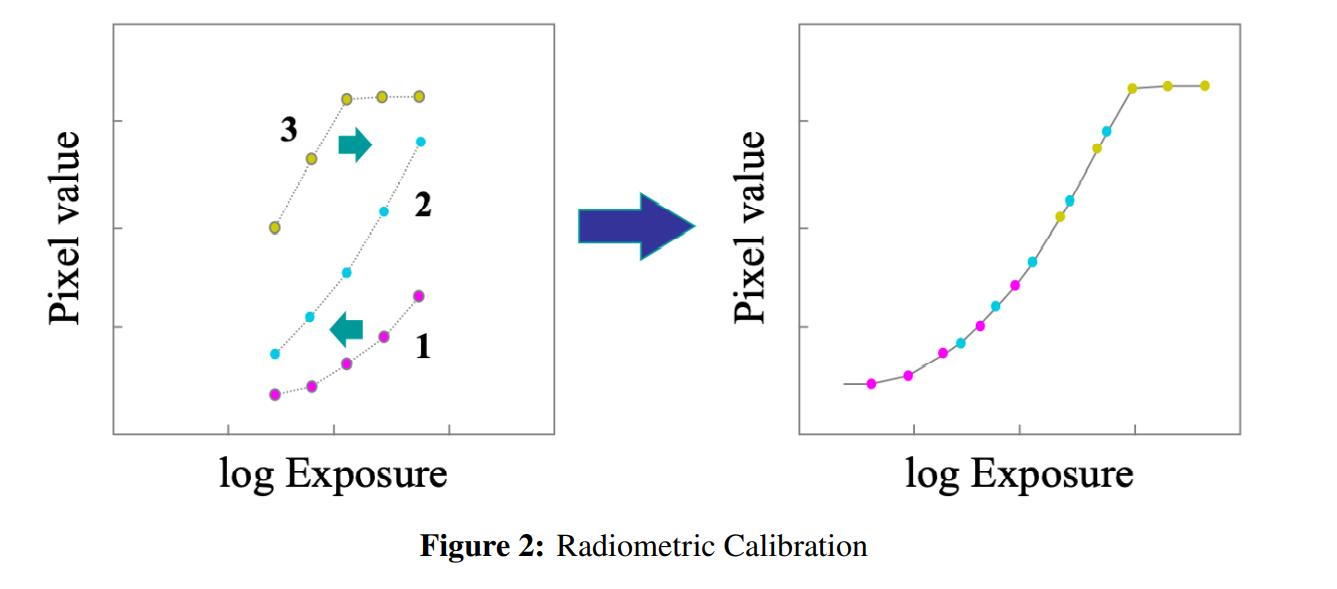
* Experiments a.
  + What is the meaning of this figure?

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。

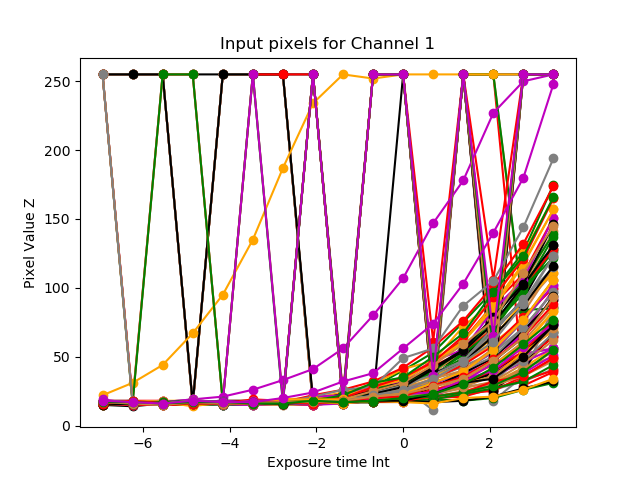
欲求得Figure 2之右圖，需先將Implementation中1.1 Camera response calibration中的equations用numpy.linalg.lstsq()求出後，再以(本來就是已知的pixel value)與(Camera response)作圖即可得到Figure 2之右圖。從Figure 2之右圖可發現，當曝光時間一直變大，並且到達一個閾值之後，其pixel value都會落在256並且不再增加。

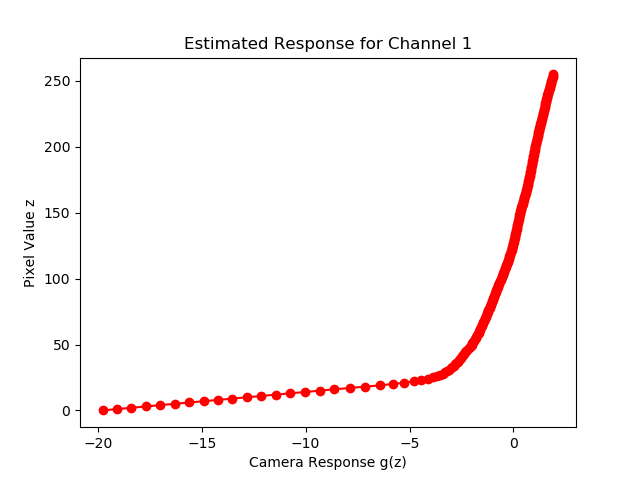


* + Try to plot a same figure for memorial image set

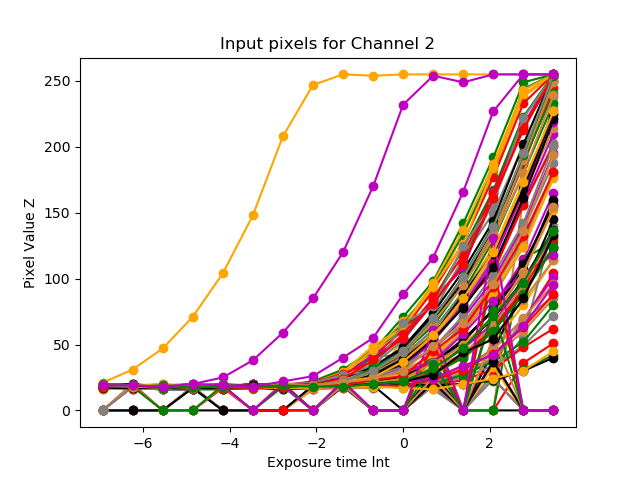
下面六張圖片儲存在” result/Experiments/a/”資料夾中，並會在執行cr\_calibration.py時自動秀出並儲存。

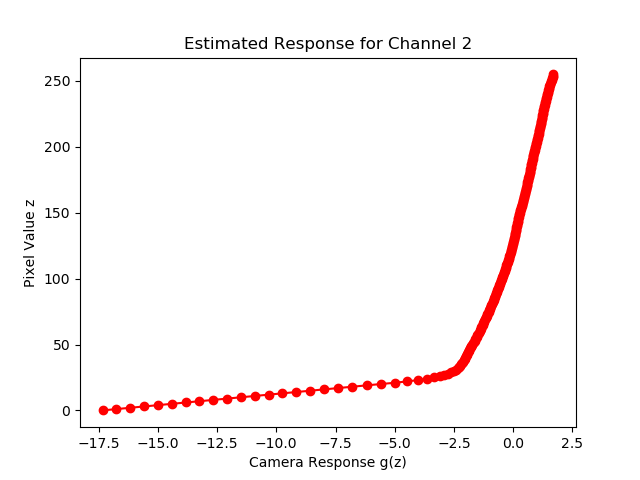
---------------------------------------------------------------------------------------------------



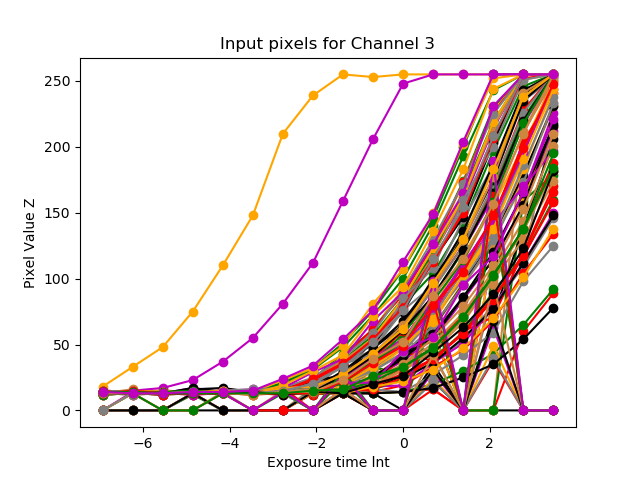


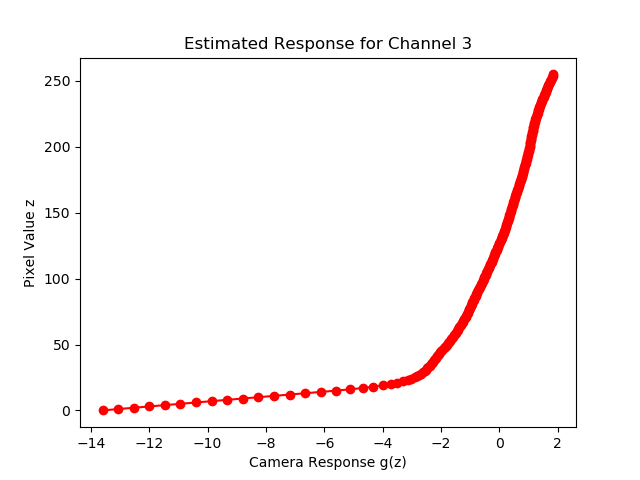
----------------------------------------------------------------------------------------------------------





----------------------------------------------------------------------------------------------------------



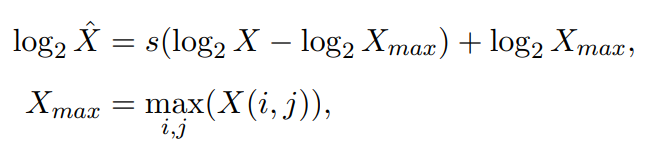


----------------------------------------------------------------------------------------------------------

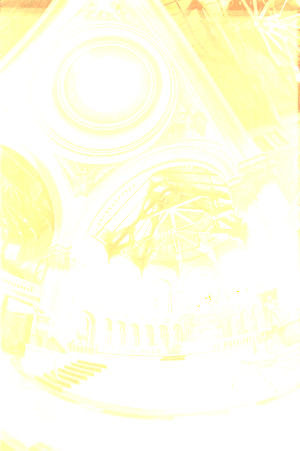
* + How do you plot the figure and why do you choose to do so?

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()得到再對取對數得ln，並採用(ln+ln)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



Implementation中1.2(Global tone mapping)設定的s值為1，也就是最後得到的等於*X*，而當s值為 0到1之間時，log(X)-log()<log()- log()，所以大於*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

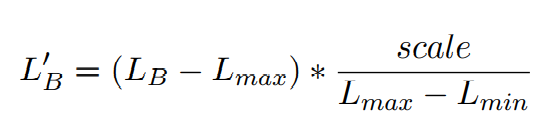
s=1.4

-----------------------------------------------------------------------------------------------------------------



-----------------------------------------------------------------------------------------------------------------

* + Scaling factor in local tone mapping with Gaussian filter



從上式可以發現，為大於等於零的固定值， 所以在scale大於等於1的情況下，是一個小於等於零的值，故scale越大，的絕對值就越大，也就是就越小，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，故比較小的pixel位置，較大，此時的絕對值就越大，也就是越小，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：

與是三個channel各別的最大值與最小值，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)



-------------------------------------------------------------------------------------------

1. 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)



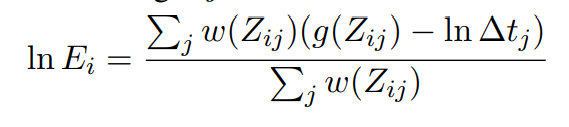
-------------------------------------------------------------------------------------------

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：



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。