CMPE 264 Project Assignment 1

Mariette Souppe and Andrea David

msouppe@ucsc.edu and andavid@ucsc.edu

November 14, 2018

Part 1: Camera radiometric calibration

Obtaining the average brightness B'

To obtain the average brightness, we started by taking multiple pictures of a flat sheet of white paper with gain, G, set to 100 ISO. We loaded our images from our local directory using the OpenCV library function *imread()* and we cropped out a portion of each image. To compute the average value of brightness of the newly created set of cropped out images, we first needed to split each image by color channel. We computed the average brightness of each channel separately for each of the images and used those values to calculate the average brightness of an entire image. Looking through the images, we needed to make sure that their brightness value is less than 255, since we wanted to avoid saturated pixels. Once we removed the pictures with saturated pixels, we were left with a set of unsaturated images that were taken with exposure times 1/125, 1/180, 1/350, 1/500, 1/750, 1/1000, 1/1500, and 1/2000 seconds. Figure 1 shows the set of unsaturated pictures with their corresponding exposure times. Using these images, we generated an array containing the average brightness values of all these images for each color channel and plotted these values against exposure time for each color channel. Figure 2 shows the plots for the B'(T) as a function of exposure time T for each color channel.

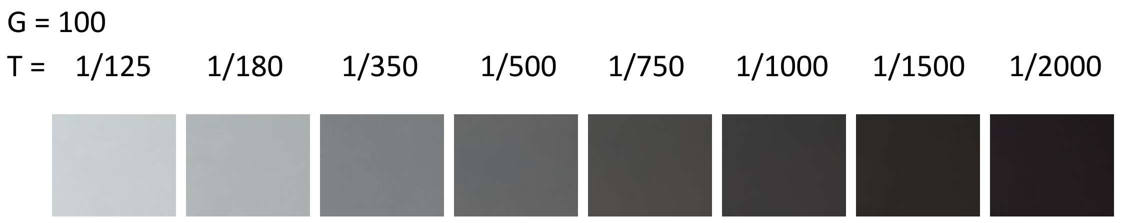


Figure 1. Set of pictures we took with their corresponding exposure times

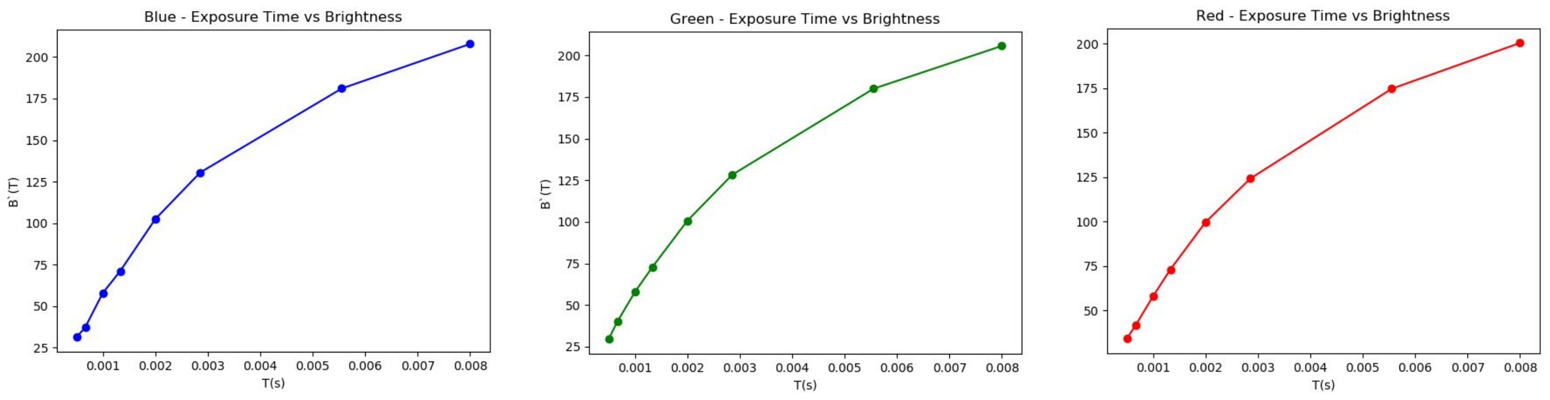


Figure 2. Plots of measured brightness B’ as a function of exposure time T for each color channel

Estimating parameter g using linear regression

After calculating the average brightness, we took that measure to calculated the parameter *g* to get the linear brightness, B, as B= B’^{g}. To calculate the regression line, we used the function *scipy.stats.linregress()* from the Scipy library. This function takes in two arrays as inputs, and outputs the slope and intercept of the regression line — as well as a correlation coefficient, two-sided p-value, and standard error for the estimate. In our case, we only needed the slope of the line, a, to obtain the g value equal to 1/a. Our input arrays to the function were the logarithm array of our measured brightness B’ and logarithm array of exposure times. Below is the equation that was used to find our constants for the linear regression line:

$$B'(T) \ = \ K \cdot T^{(1/g)}$$

$$log(B'(T)) \ = \ log(K \cdot T^{(1/g)}) \ = \ log(K) \ + \ (1/g) \ log(T)$$

$$log(B'(T)) \ = b \ + \ a \ log(T)$$

*Note: $log(B'(T))$ is an affine function, therefore $log(K)$ and $(1/g)$ are constants

Our g parameter values are the following:

- 1.401 for the blue color channel
- 1.411 for the green color channel
- 1.521 for the red color channel

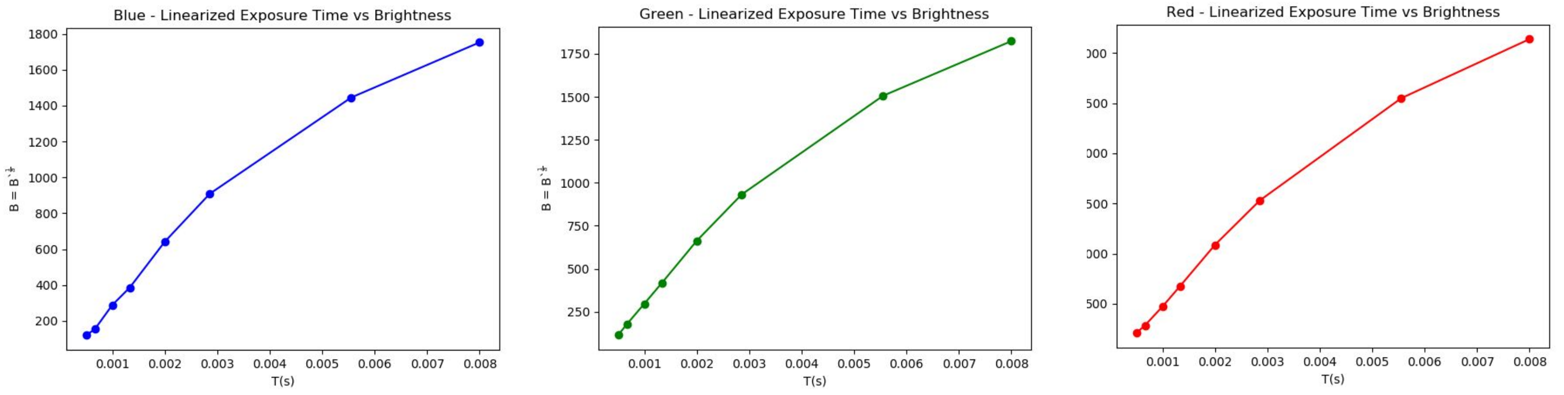


Figure 3. Plot of $B'^g(T)$ as a function of exposure time T for the blue, green, and red color channels

Part 2: Acquire a picture set

In this part, we took multiple images of a high-dynamic scene at fixed gain of 100 ISO. The full stack of pictures we took is shown in Figure 4 below. From these images we needed to choose a stack of 3 with the requirement that the first picture be at optimal exposure. To select this image, we looked at the histogram of each image we took and chose the picture that had the highest exposure time but gave no saturated pixels. The other two pictures were chosen arbitrary from the set of pictures that had higher exposure times than the first one. Values of a_1 and a_2 are calculated by taking the ratio of the exposure times of the second picture the first picture and taking the ratio of the exposure time of the third picture and the first picture, respectively. The histograms of the values of $B'^g(a_i*T)$, where i = 0,1,2, are shown in Figure 5-7 for each of the color channels. Figures 8 and 9 show the histograms of the values $B'^g(a_1*T)/a_1$ for the second picture and the values $B'^g(a_2*T)/a_2$ for the third picture, respectively.

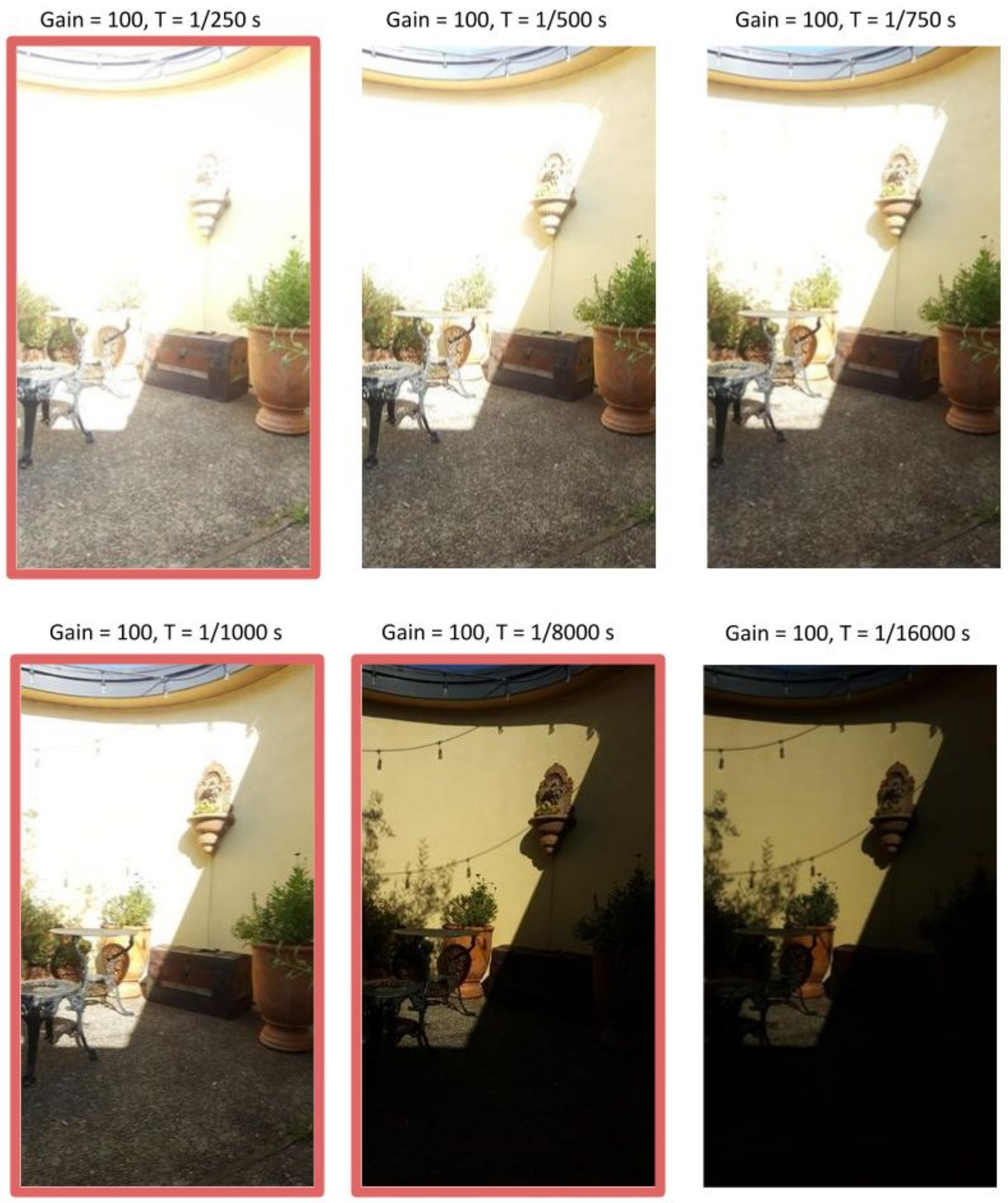


Figure 4. Full stack of pictures taken (chosen images are outlined with a red border)

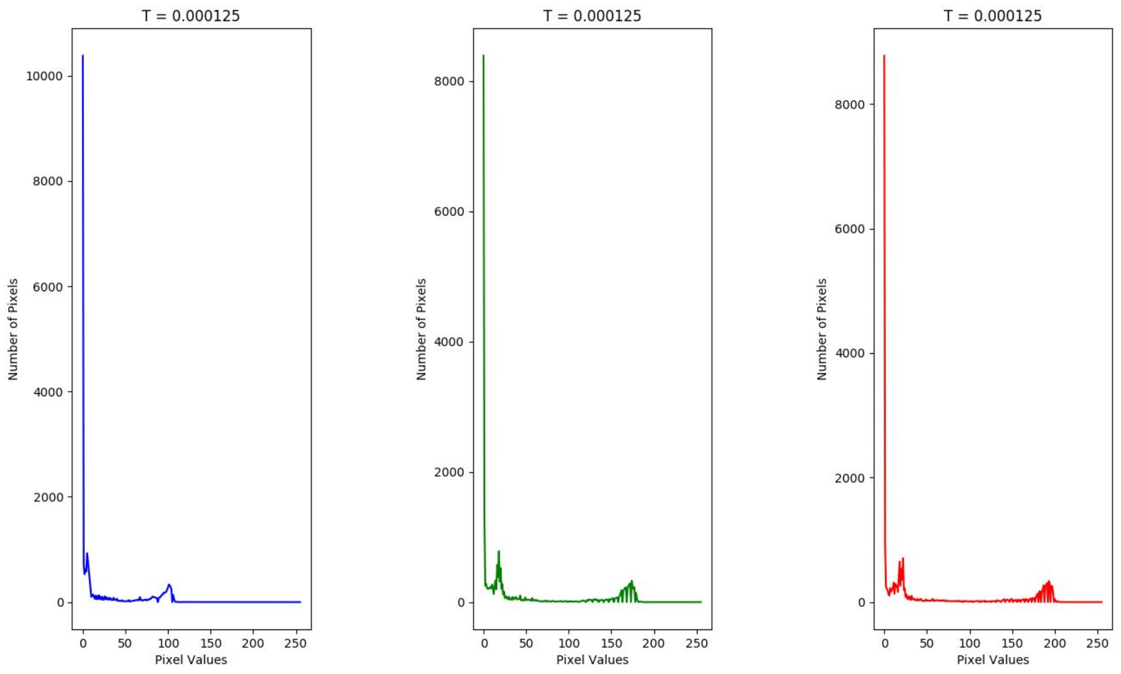


Figure 5. Histograms of the values $B'^{g}(a_0 * T)$ for the blue, green, and red color channels

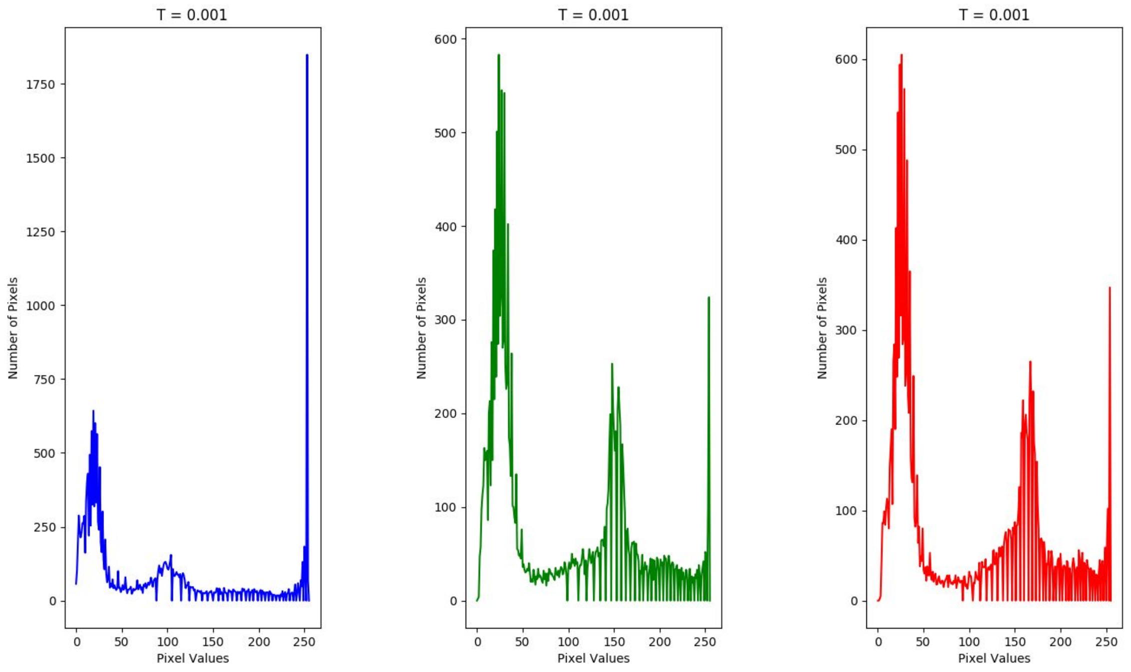


Figure 6. Histograms of the values $B'^{g}(a_1 * T)$ for the blue, green, and red color channels

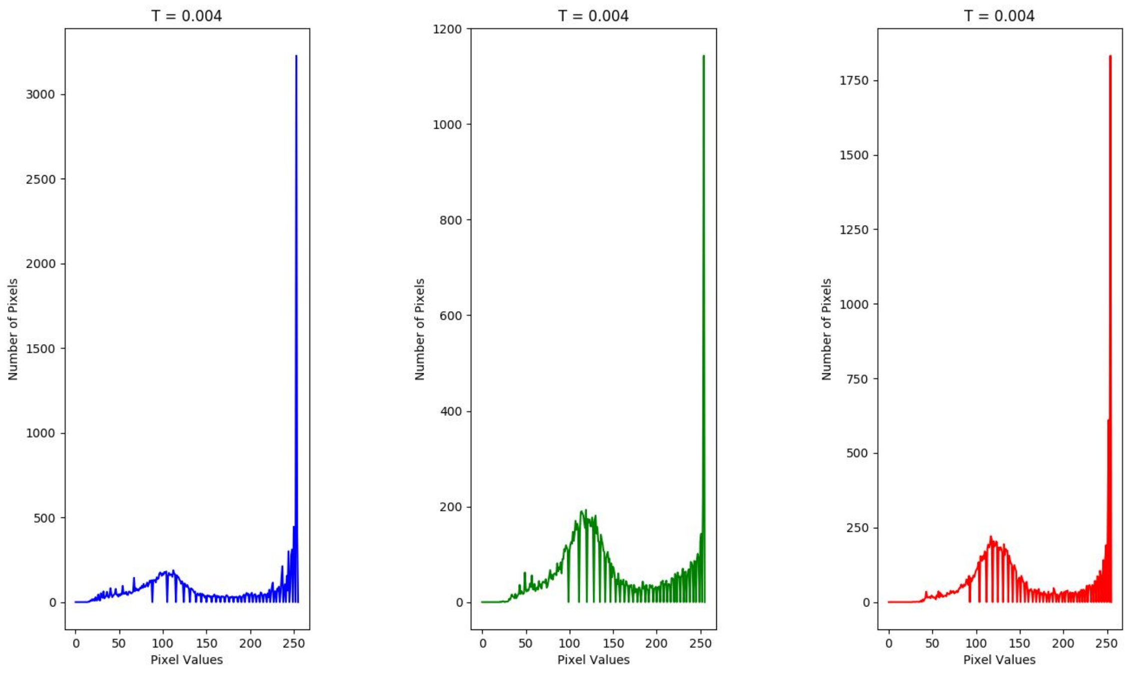


Figure 7. Histograms of the values $B'^{g}(a_2 * T)$ for the red, green, and blue color channels

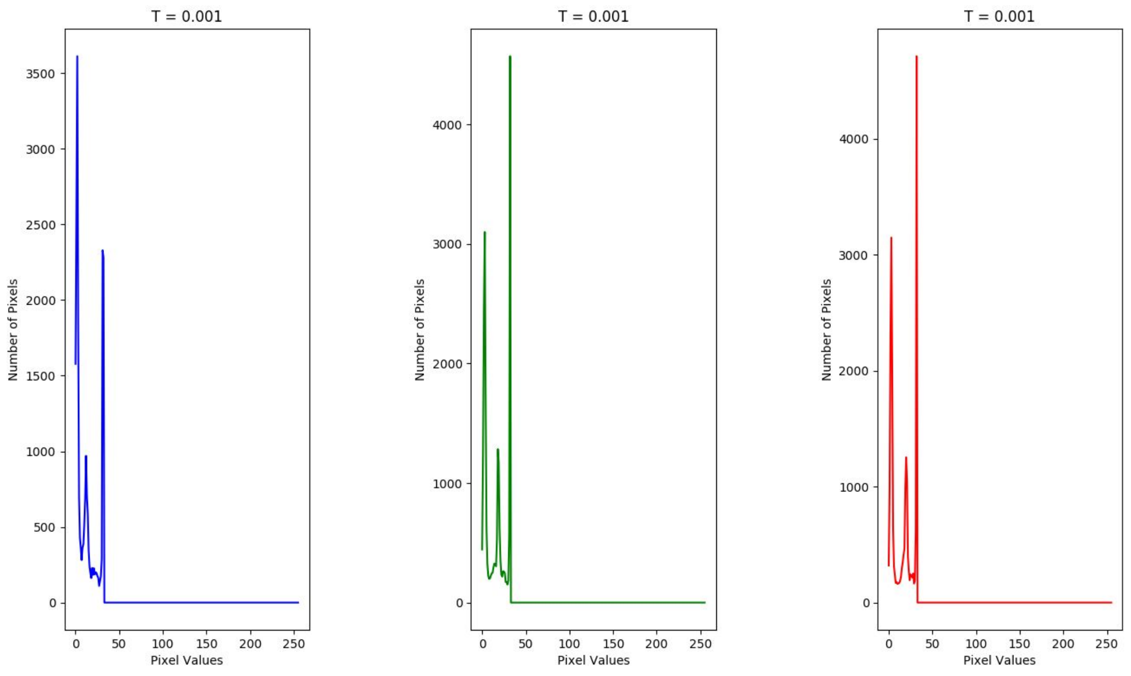


Figure 8. Histogram of the values $B'^{g}(a_1 * T)/a_1$

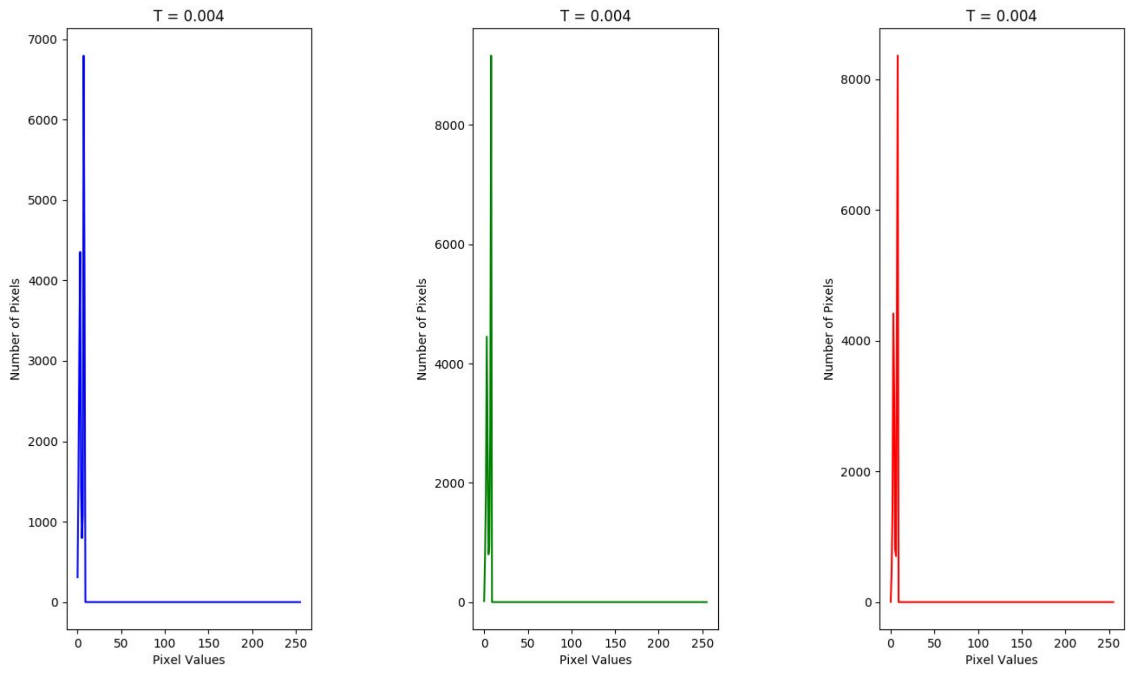


Figure 9. Histogram of the values $B'^{g}(a_2 * T)/a_2$

Part 3: Create a composite image

Using our stack of three linearized pictures from Part 2, we created a composite HDR image using the following two algorithms.

Composite Algorithm 1: We first set the threshold for the pixel values for picture two and three. The first picture we have taken is exposed to the right and has no saturated pixels. We first check the pixel value for the third picture. If this pixel is below the threshold, i.e., is not saturated, we use it in our composite image. However, if the pixel is saturated, we check whether this pixel is saturated in the second picture. If this pixel is not saturated, then we use this pixel from the second picture in our composite image. If the pixel is saturated in both the third and the second image, we take that pixel from the first image to create a composite image. The composite image now has only non-saturated pixels taken from either the first, second, or third image.

Composite Algorithm 2: In this method, we use the same thresholds as in the previous algorithm to determine whether the pixel is saturated. However, in this case if the pixel is unsaturated in the third picture, we take the average of the values of that pixel from the first image, the second image, and the third image. If the pixel is saturated in the third picture, we check whether it is saturated in the second picture. If the pixel is not saturated in the second picture, then we take the average values of that picture from the first and the second image and use it to create our composite picture. Finally, if the pixel is saturated both in the second and third picture, then we only use that pixel in our composite image. Figure 10 and 11 show the histograms for the composite image obtained using Algorithm 1 and Algorithm 2, respectively.

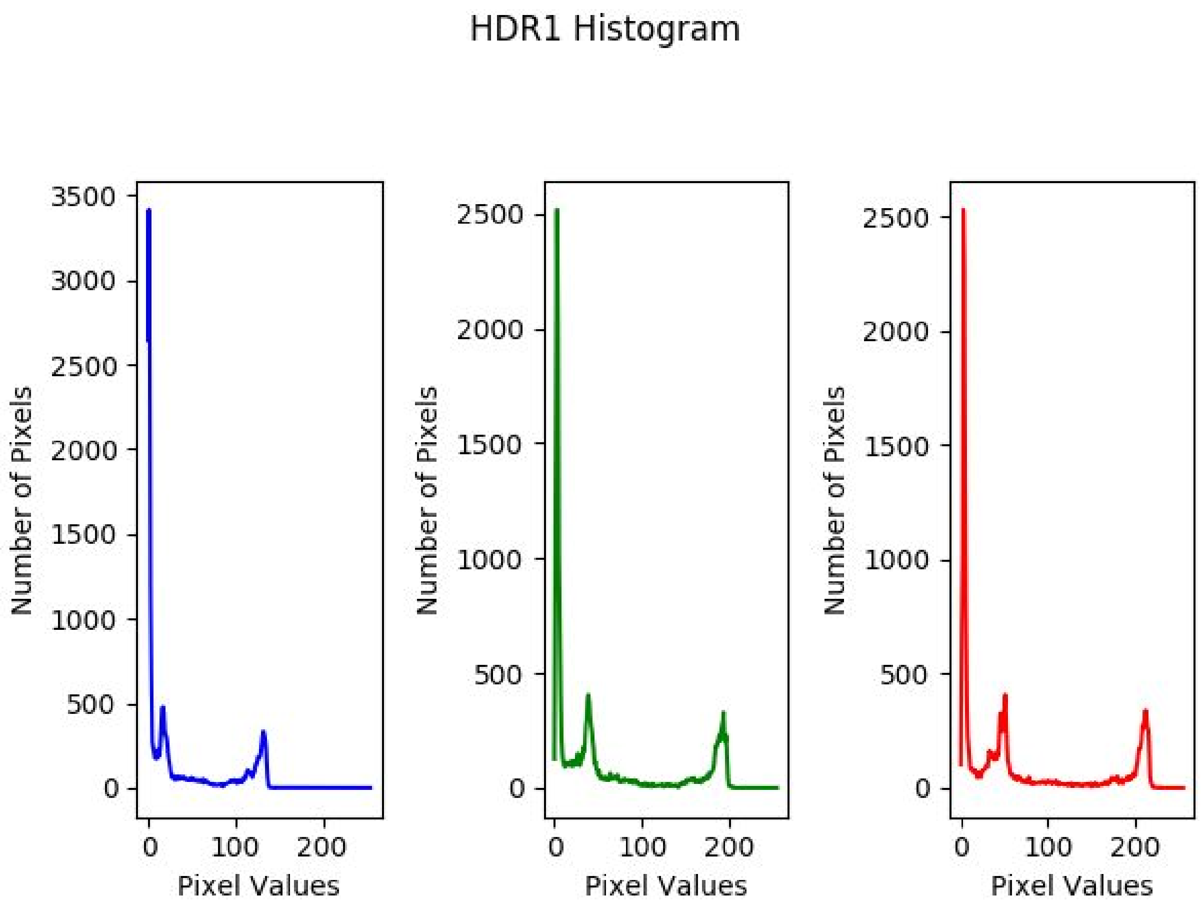


Figure 10. Histogram of the composite image obtained using Algorithm 1

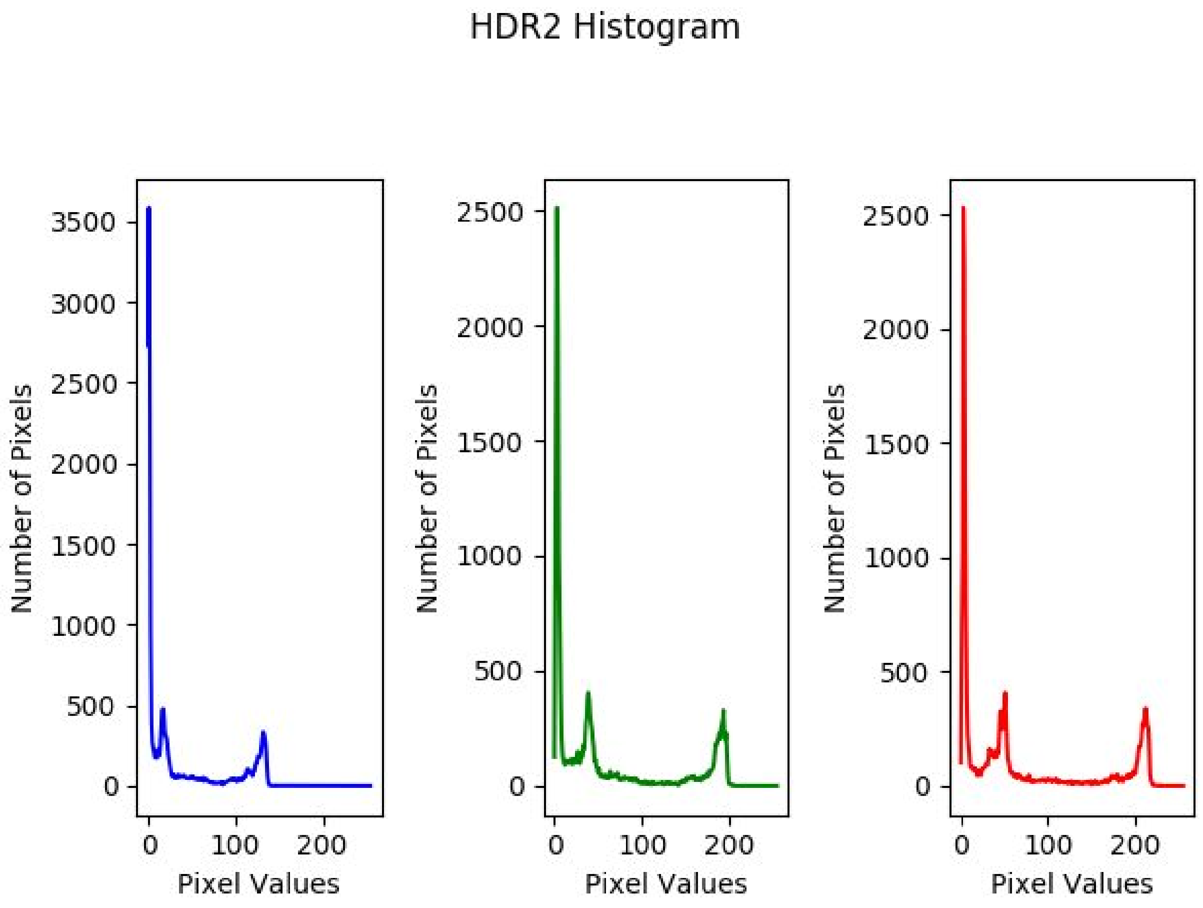


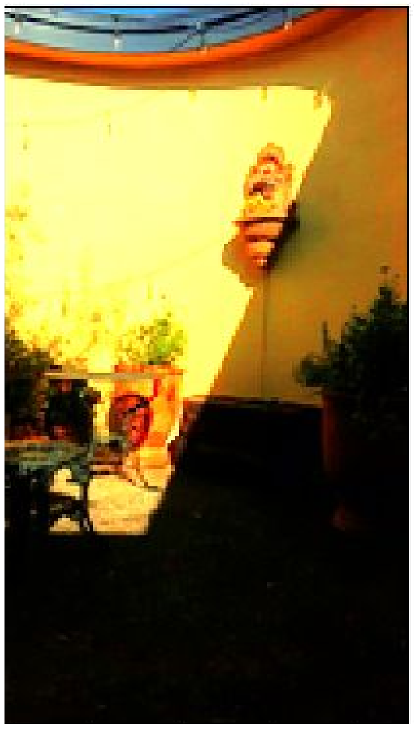
Figure 11. Histogram of the composite image obtained using Algorithm 2

Part 4: Reproduce composite image

After computing the composite HDR image, we used tone mapping to tone the image in which help adjusts our images gamma, saturation, and bias. The tone-mapping function was obtained from the following the tutorial titled “High Dynamic Range (HDR) Imaging using OpenCV (C++/Python)” by Satya Mallick found here. We decided to use the Drago Tonemap algorithm that allowed us to adjust the gamma, saturation, and bias parameters. After experimenting with the values, we found that a gamma of 0.4, saturation of 0.9, and a bias of 0.7 worked best with our image. The new HDR tone-mapped image showed us more detail in the darker areas that we originally could not see in the original picture. Figure 12 shows the toned-mapped composite images from Algorithm 1 and Algorithm 2.

*Note: The HDR composite images are relatively small since my computer could not process an image size larger than the dimensions of 100 x 200. Additionally, looking back at the histograms for the selected images in Part 3, there is a faint difference between the two algorithms. As a result we pretty much obtained the same composite image after tone mapping.

HDR 1



HDR 2

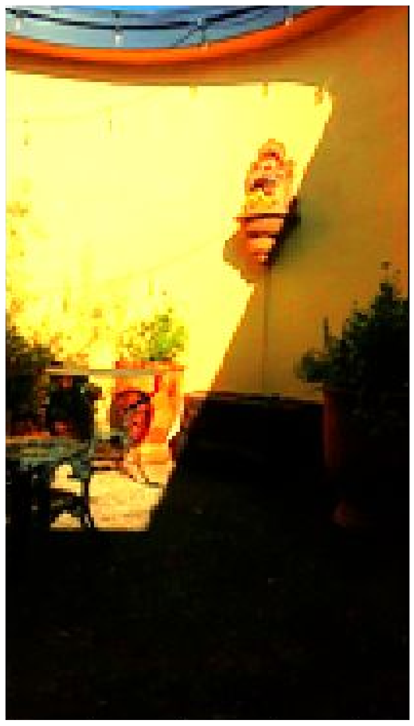


Figure 12. Tone-mapped composite images from Algorithm 1 and Algorithm 2