BBM 446 - Computational Photography Laboratory
Assignment 1 Report

Spring 2022
Assoc. Prof. Dr. Erkut ERDEM

Arda Hüseyinoğlu
Student ID: 21627323

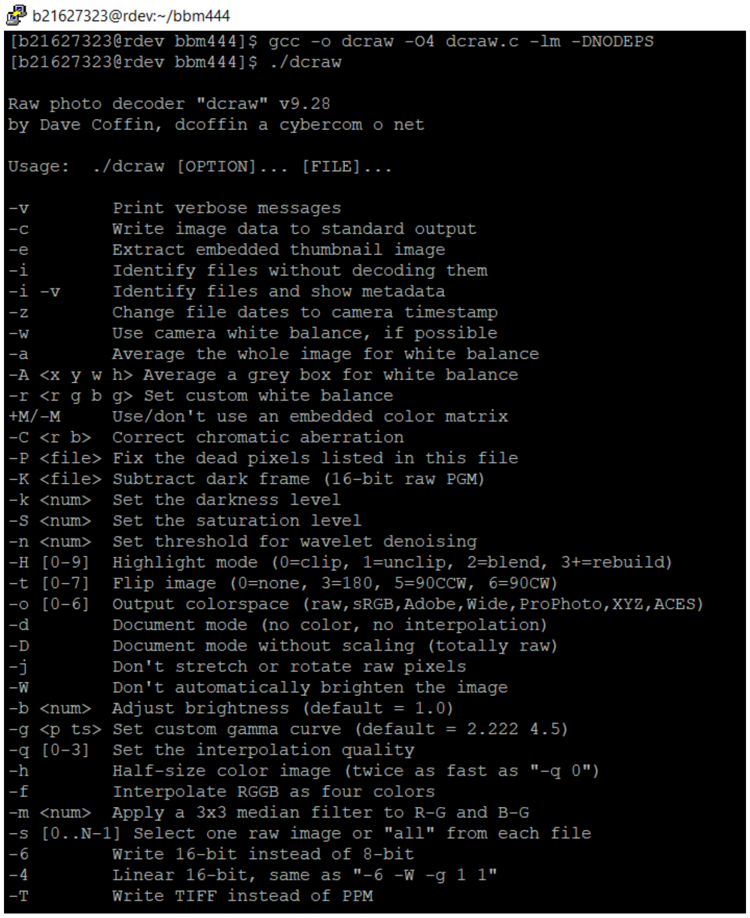
Department of Computer Engineering, Hacettepe University
March 8, 2022

1. Implement a basic image processing pipeline

a. RAW image conversion

Firstly, we use a command-line tool called *dcraw* to convert the RAW image file into a .tiff file in order to read it via *skimage*. To use *dcraw*, we first get the ANSI C program (which can be found here *https://www.dechifro.org/dcraw/dcraw.c*). After compiling the program with :

"gcc -o dcraw -O4 dcraw.c -lm -DNODEPS", we can work with it:



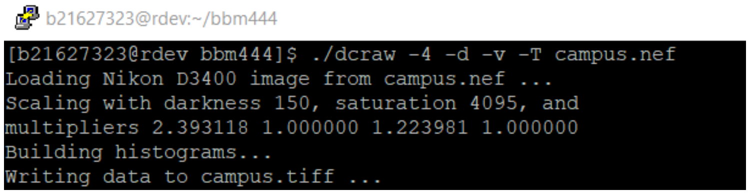
```
b21627323@rdev:~/bbm444
[b21627323@rdev bbm444]$ gcc -o dcraw -O4 dcraw.c -lm -DNODEPS
[b21627323@rdev bbm444]$ ./dcraw

Raw photo decoder "dcraw" v9.28
by Dave Coffin, dcoffin a cybercom o net

Usage:  ./dcraw [OPTION]... [FILE]...

-v        Print verbose messages
-c        Write image data to standard output
-e        Extract embedded thumbnail image
-i        Identify files without decoding them
-i -v     Identify files and show metadata
-z        Change file dates to camera timestamp
-w        Use camera white balance, if possible
-a        Average the whole image for white balance
-A <x y w h> Average a grey box for white balance
-r <r g b g> Set custom white balance
+M/-M     Use/don't use an embedded color matrix
-C <r b>  Correct chromatic aberration
-P <file> Fix the dead pixels listed in this file
-K <file> Subtract dark frame (16-bit raw PGM)
-k <num>  Set the darkness level
-S <num>  Set the saturation level
-n <num>  Set threshold for wavelet denoising
-H [0-9]  Highlight mode (0=clip, 1=unclip, 2=blend, 3+=rebuild)
-t [0-7]  Flip image (0=none, 3=180, 5=90CCW, 6=90CW)
-o [0-6]  Output colorspace (raw,sRGB,Adobe,Wide,ProPhoto,XYZ,ACES)
-d        Document mode (no color, no interpolation)
-D        Document mode without scaling (totally raw)
-j        Don't stretch or rotate raw pixels
-W        Don't automatically brighten the image
-b <num>  Adjust brightness (default = 1.0)
-g <p ts> Set custom gamma curve (default = 2.222 4.5)
-q [0-3]  Set the interpolation quality
-h        Half-size color image (twice as fast as "-q 0")
-f        Interpolate RGGB as four colors
-m <num>  Apply a 3x3 median filter to R-G and B-G
-s [0..N-1] Select one raw image or "all" from each file
-6        Write 16-bit instead of 8-bit
-4        Linear 16-bit, same as "-6 -W -g 1 1"
-T        Write TIFF instead of PPM
```

Then, we do a "reconnaissance run" to extract some information about the RAW image:



```
b21627323@rdev:~/bbm444
[b21627323@rdev bbm444]$ ./dcraw -4 -d -v -T campus.nef
Loading Nikon D3400 image from campus.nef ...
Scaling with darkness 150, saturation 4095, and
multipliers 2.393118 1.000000 1.223981 1.000000
Building histograms...
Writing data to campus.tiff ...
```

We record the values of darkness, saturation and multipliers to utilize them for the upcoming parts of the assignment. From the output, we can also see that image was captured with a Nikon D3400 DSLR camera (This information will be useful for the color space correction step of the image processing pipeline).

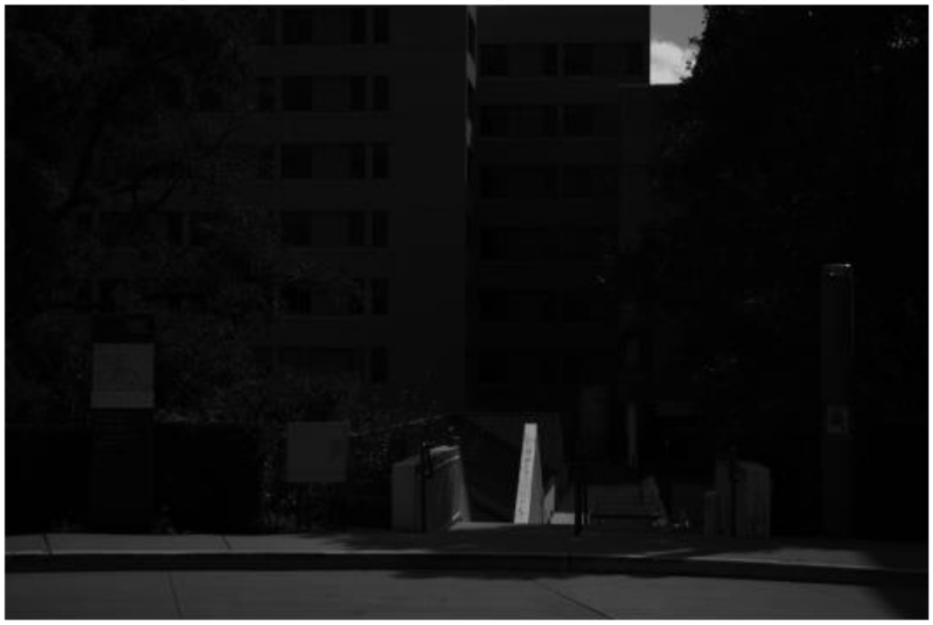
Then we call dcraw once more to produce a new campus.tiff file that we can use for the rest of this problem.

```
dcraw -4 -D -T campus.nef
```

b. Python initials

- Image width is **6016**.
- Image height is **4016**.
- 'dtype' of the numpy array 'im_tiff' is **uint16**, which means the image is **16 bits per pixel (unsigned integer)**

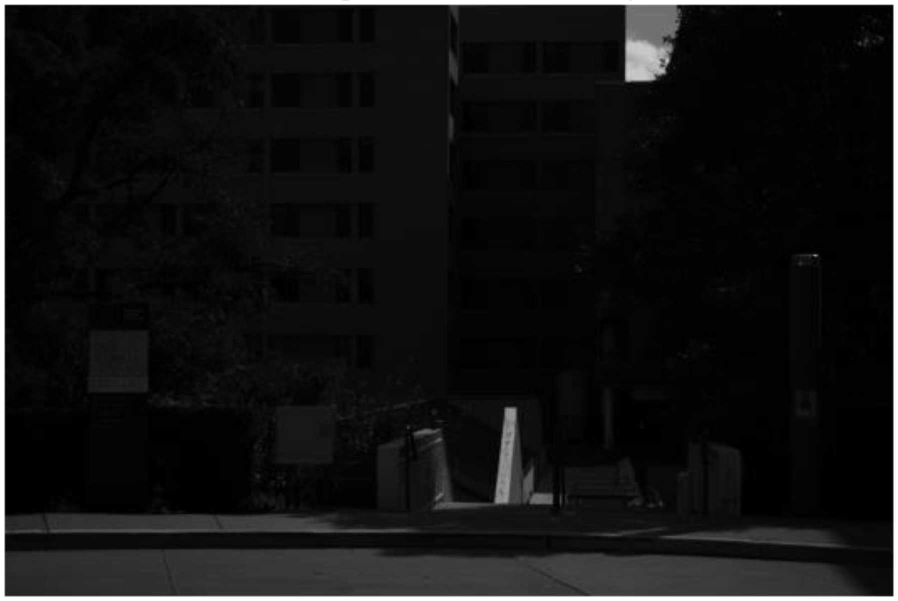
The image we produce after converting .nef file to .tiff file (campus.tiff)



c. Linearization

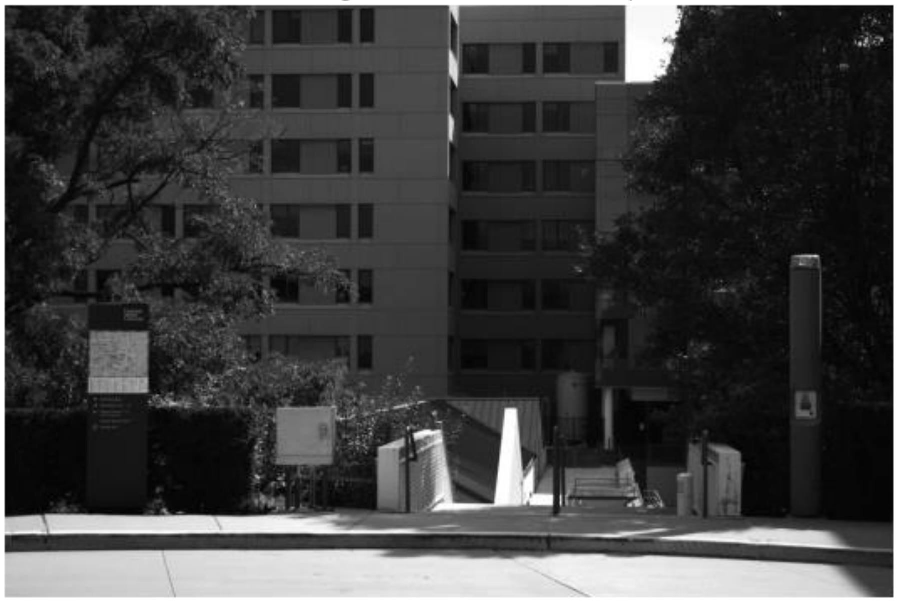
Then, we convert the image into a linear array within the range [0, 1] so that the value "black" is mapped to 0, and the value "white" is mapped to 1. To do that, we apply normalization to our image. In this case, minimum value will be taken as value of "black", and maximum value will be taken as value of "white", since we want that the value "black" is mapped to 0, and the value "white" is mapped to. After that, we clip negative values to 0, and values greater than 1 to 1.

The image after the linearization step



In assignment paper, when we display intermediate results of the image, we are recommended to apply scaling to the image to be able to see the image more meaningful. So, here's the result:

The image after the linearization step



d. White balancing

In this step, we will perform a set of experiments to figure out the correct settings for our image. To perform white balancing, We will use 2 different automatic white balancing algorithms (white world and gray world) and the ready-to-use scale factors that we have obtained from the reconnaissance run. Also, we consider all possible Bayer patterns to identify the correct Bayer pattern of our image. Thus we realize 12 experiments in total: 3 (AWB methods) x 4 (all possible Bayer patterns).

We first get the image pixels for each channel (r,g,b). (Since we need only pixel values to calculate the mean (gray world) and max (white world) values for each channel, it is not important that they are in order. So, we can concatenate the im_g1_vx and im_g2_vx one under the other.)

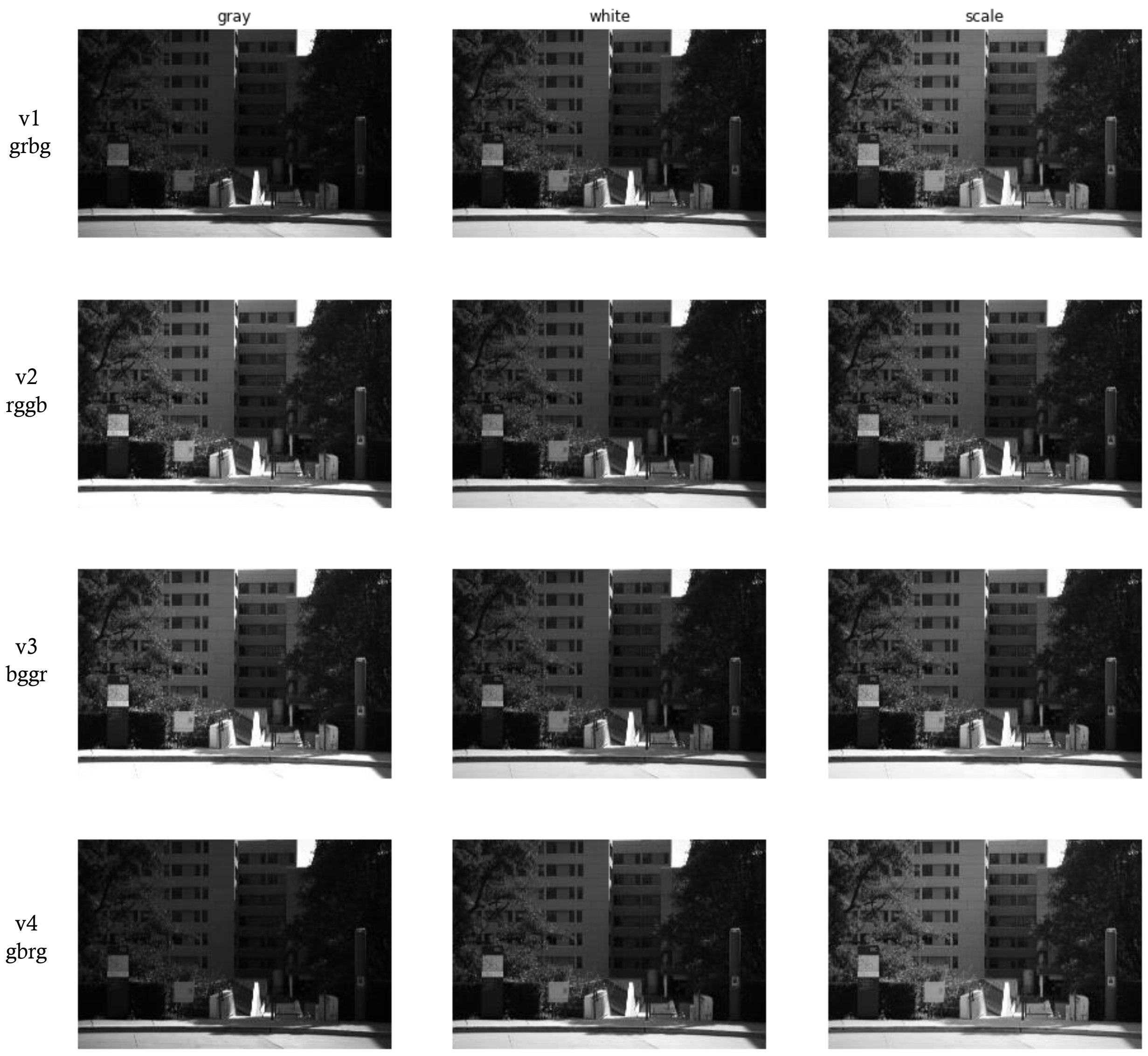


Figure. The images after different types of white balancing

The images without scaling and without the parameter of cmap='gray' are as follows:

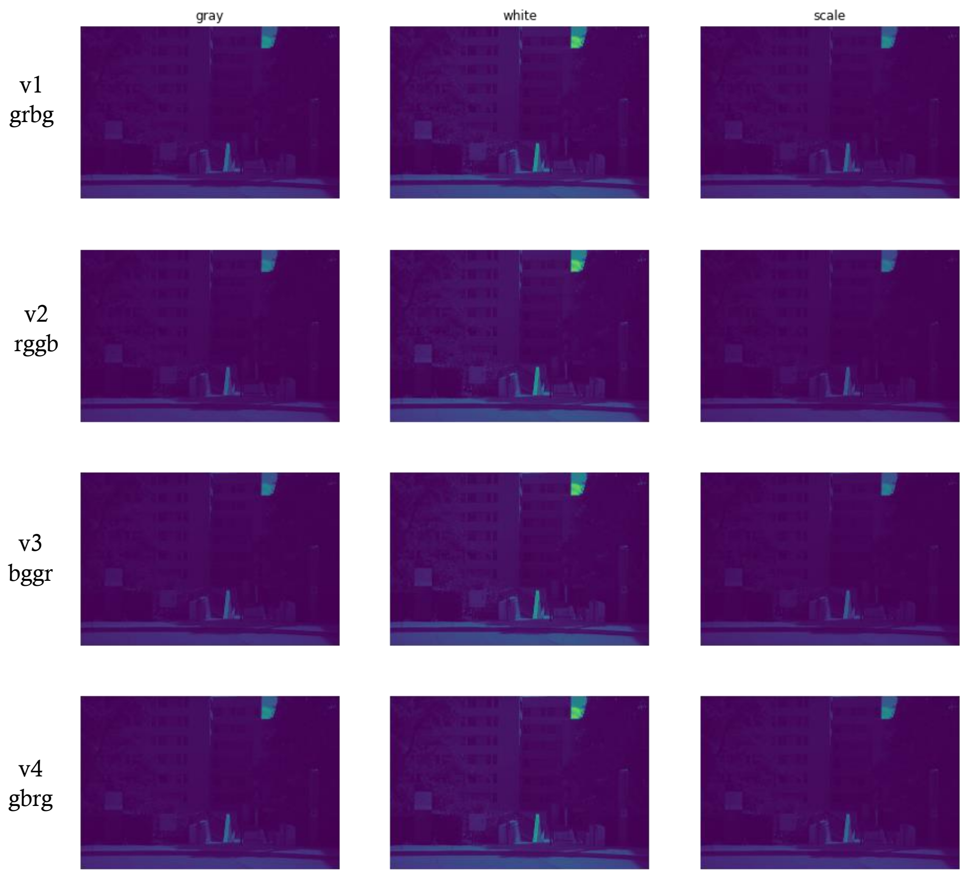


Figure. The images after different types of white balancing

In this step, we could not decide which one is the best. So, we will proceed to the demosaicing step with all these combinations to find out which Bayer pattern version our image has.

e. Demosaicing

In this step, we use bilinear interpolation for demosaicing. We use two different kernels for different channel patterns (one for red and blue channel, one for green channel). After getting each channel separately, we convolve the corresponding filters over each channel to fill the missing pixel intensities and then we stack them to get the corresponding RGB image.

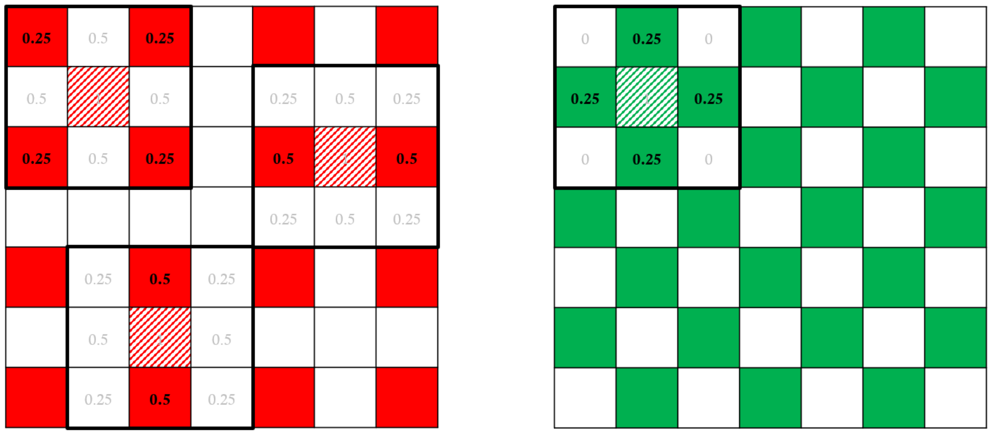


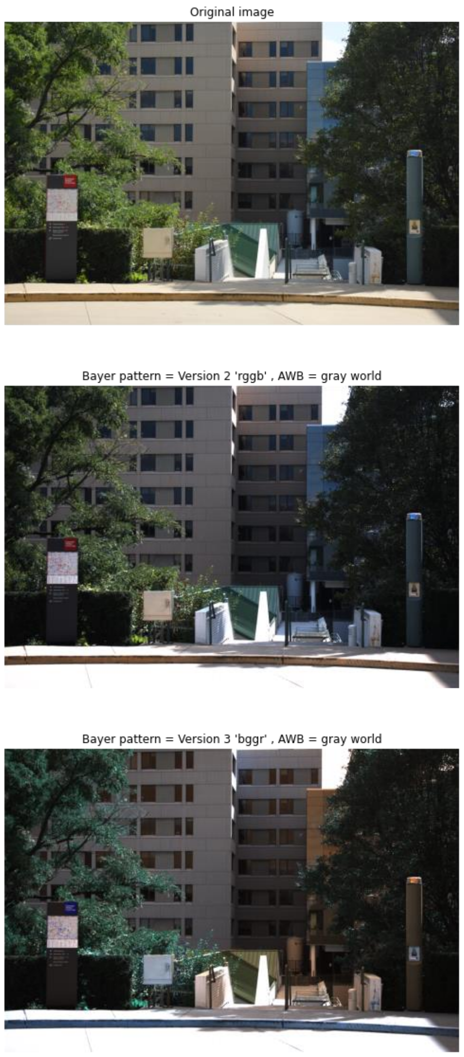
Figure. Two different kernels due to different patterns. One the red and blue channel and one for green channel. Figure shows how the missing pixel intensity values are filled for different cases.

f. Identifying the correct Bayer pattern

From the plots in the next page, we can clearly see that the images in the second row (Bayer pattern version2) looks more natural. Thus, we can decide that exact Bayer pattern of our image is 'rggb'.

Maybe we would suspect that the third row also looks okay (We do not consider version1 (row1) and version4 (row4) anyway, because we can clearly see that the colors in them does not look normal). However if we look closer, the background color of the "Carnegie Mellon University" text on the map panel at the bottom left of the photo is red in the original image, while it is blue and its tones in the images in version3.

Also, the glass color and other blue colors look more reddish in version3. So, in version3, blues look like red, and reds look like blue. Here, we come to the conclusion that the 'bggr' sequence in version3 does not fit our picture, 'b' and 'r' must be replaced, and thus our picture's Bayer pattern must be 'rggb' (version2).


Original image
Bayer pattern = Version 2 'rggb' , AWB = gray world
Bayer pattern = Version 3 'bggr' , AWB = gray world

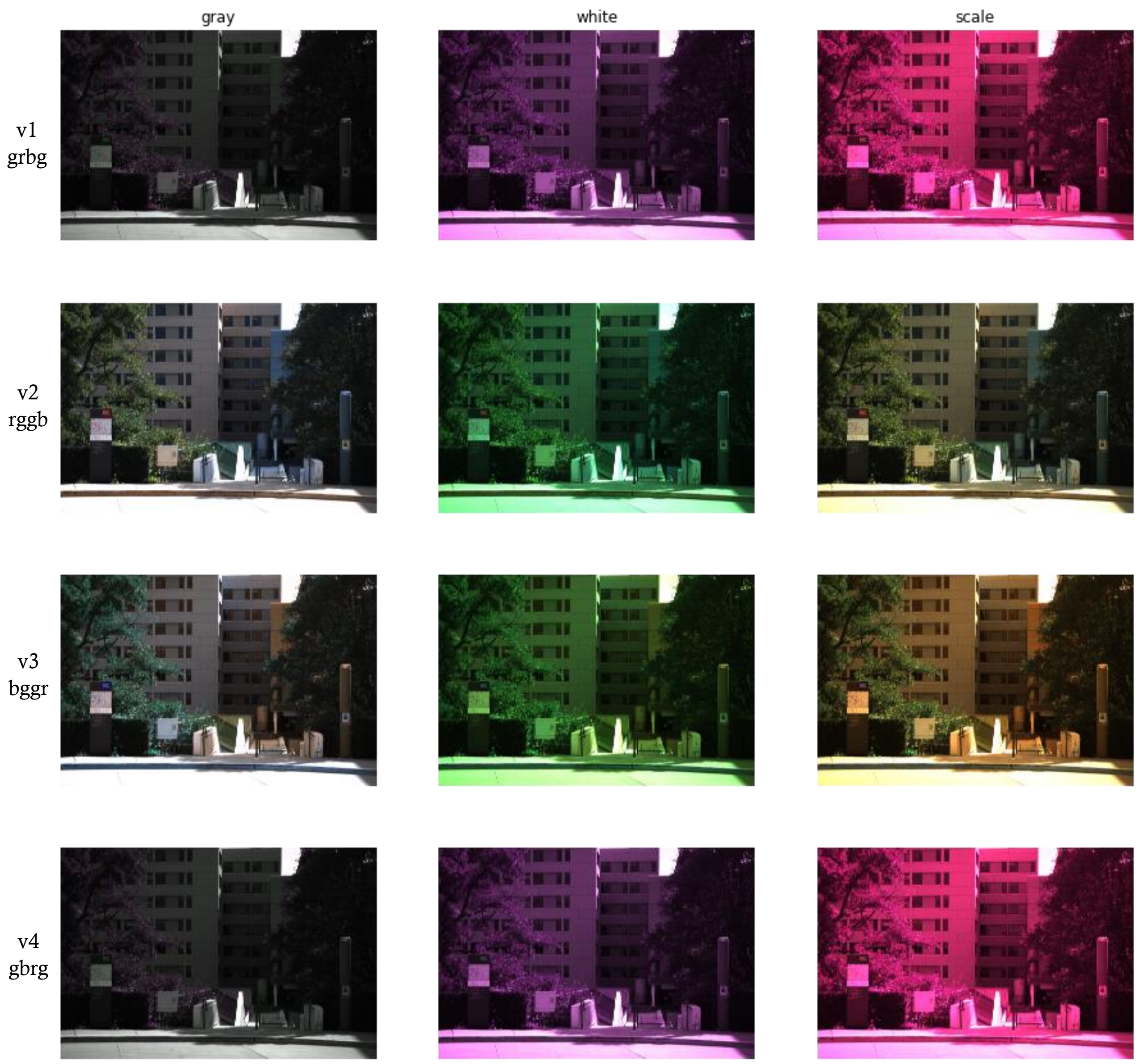
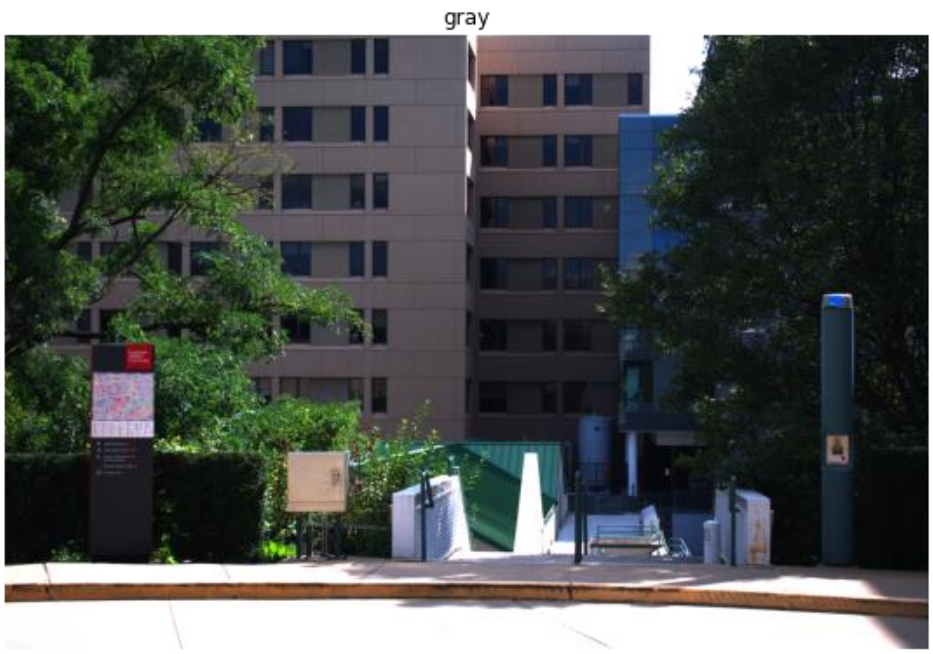


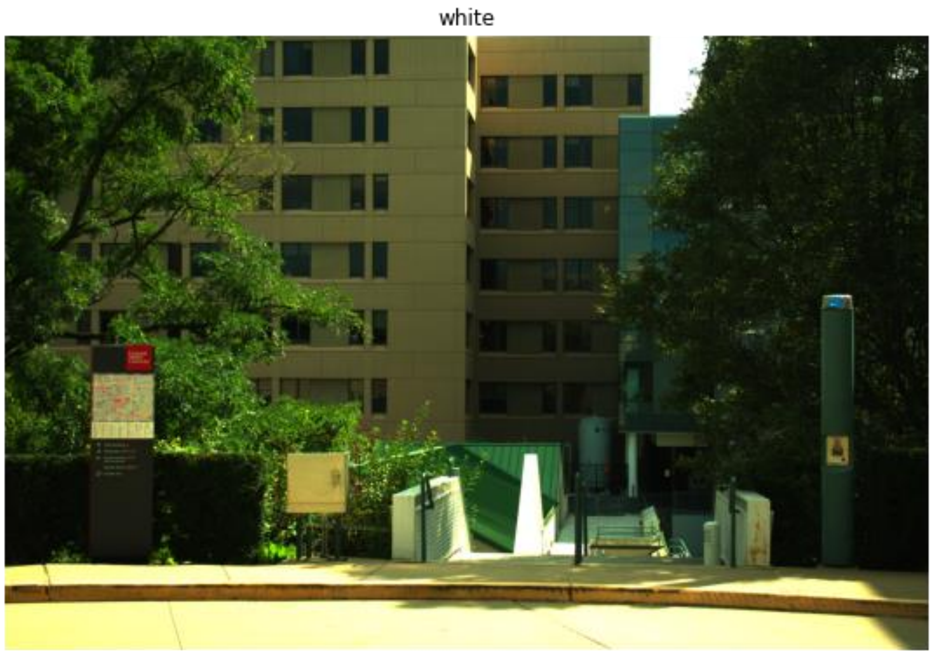
Figure. The images after demosaicing

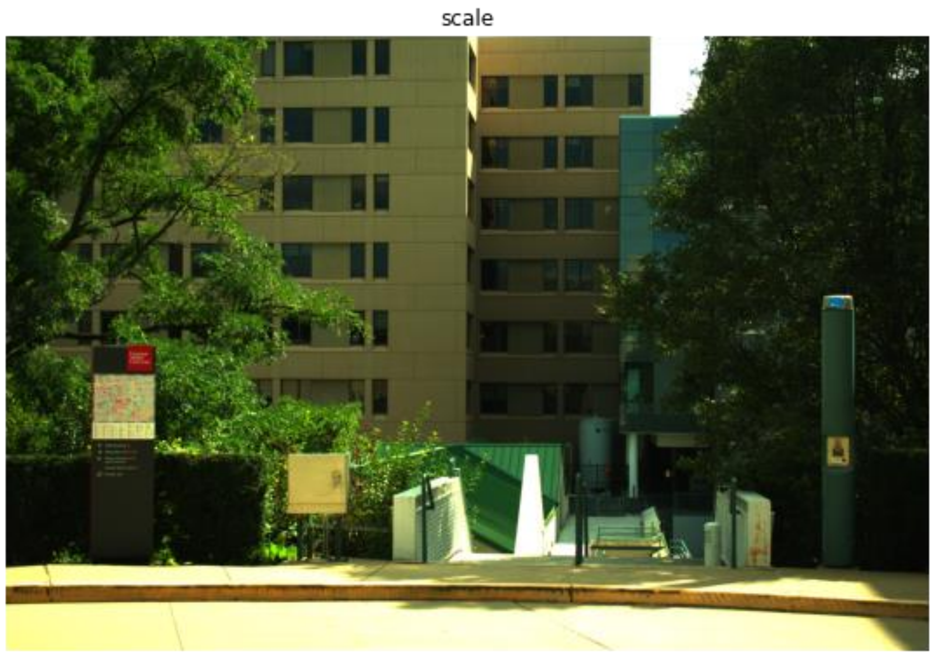
g. Color space correction

In this step, we transform our image to the linear sRGB color space. To do that, we need a transformation matrix that transform RGB coordinates in the color space which is determined by the camera's spectral sensitivity functions to the RGB coordinates that image viewing software expects (in this case, we assume that it is in the sRGB color space).

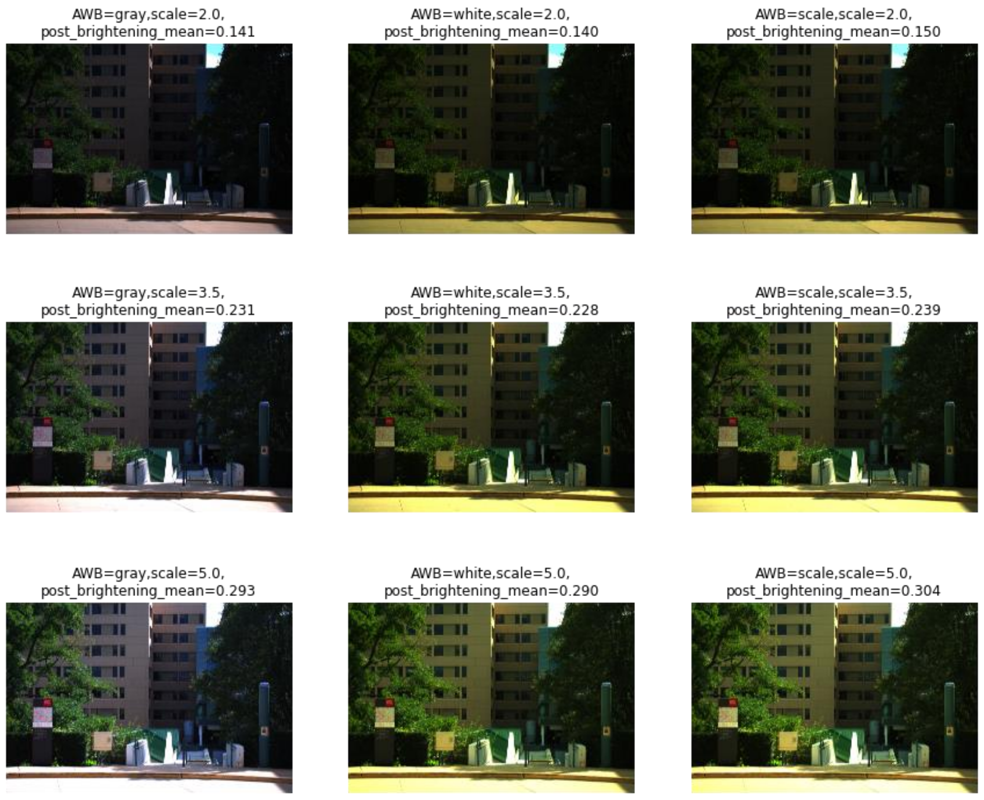


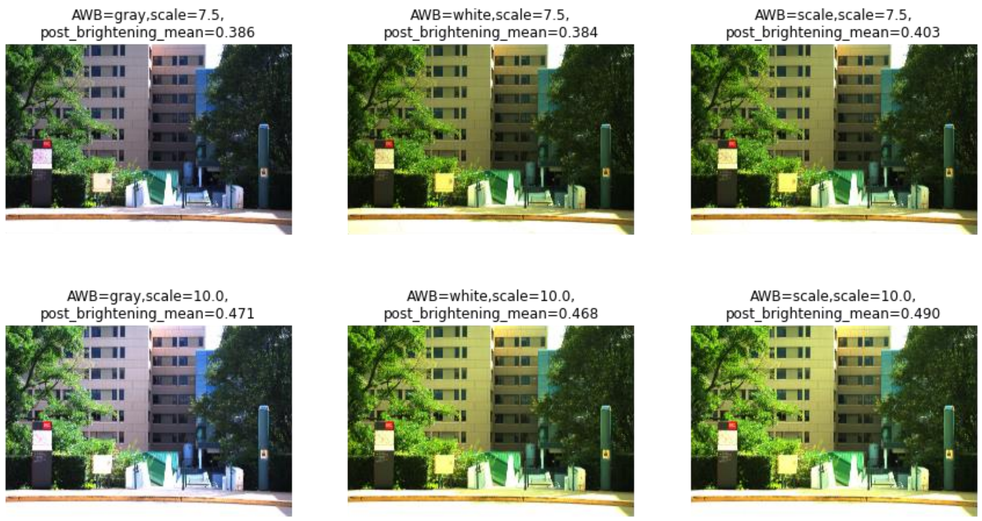
During the experiments, we realized that if the white world algorithm is used for AWB, normalizing the matrix during color space correction results in a very green density image. Therefore, if the white world algorithm is used, we do not normalize the matrix in color space correction phase.



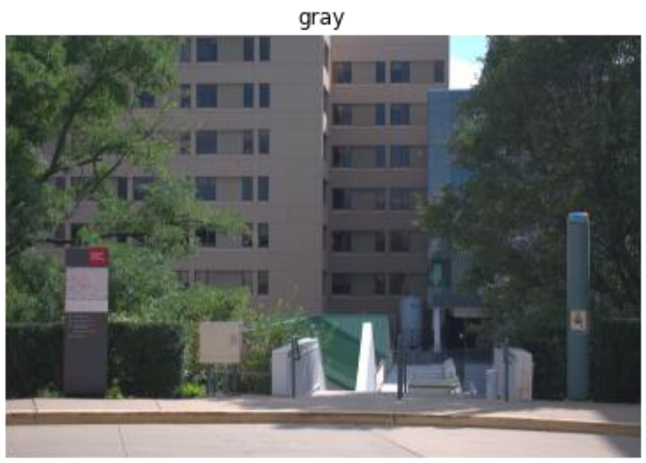


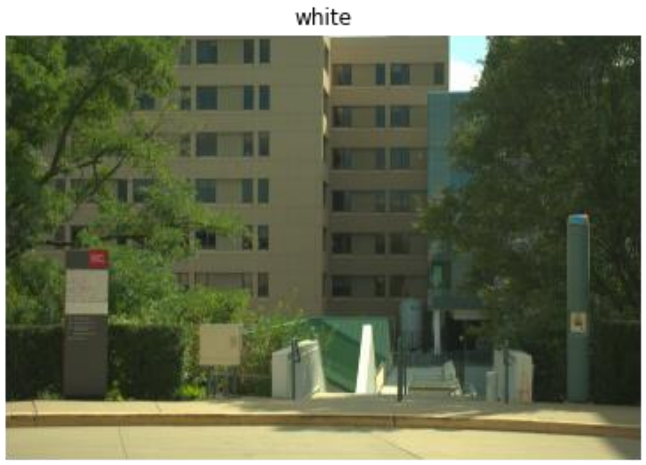
h. Brightness adjustment and gamma encoding


AWB=gray,scale=2.0, post_brightening_mean=0.141
AWB=white,scale=2.0, post_brightening_mean=0.140
AWB=scale,scale=2.0, post_brightening_mean=0.150
AWB=gray,scale=3.5, post_brightening_mean=0.231
AWB=white,scale=3.5, post_brightening_mean=0.228
AWB=scale,scale=3.5, post_brightening_mean=0.239
AWB=gray,scale=5.0, post_brightening_mean=0.293
AWB=white,scale=5.0, post_brightening_mean=0.290
AWB=scale,scale=5.0, post_brightening_mean=0.304


AWB=gray,scale=7.5,
post_brightening_mean=0.386
AWB=white,scale=7.5,
post_brightening_mean=0.384
AWB=scale,scale=7.5,
post_brightening_mean=0.403
AWB=gray,scale=10.0,
post_brightening_mean=0.471
AWB=white,scale=10.0,
post_brightening_mean=0.468
AWB=scale,scale=10.0,
post_brightening_mean=0.490

From the experiments that we realized above, we proceed with brightness scale = 2, which corresponds to the post-brightening mean value = 0.141 (AWB=gray), 0.140 (AWB=white), 0.150 (AWB=scale). We continue with this setup since highlights and shadows are preserved more.





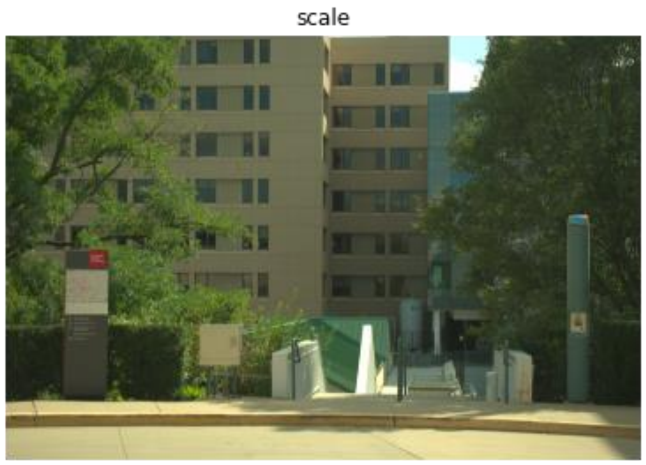


Figure. The images after gamma encoding

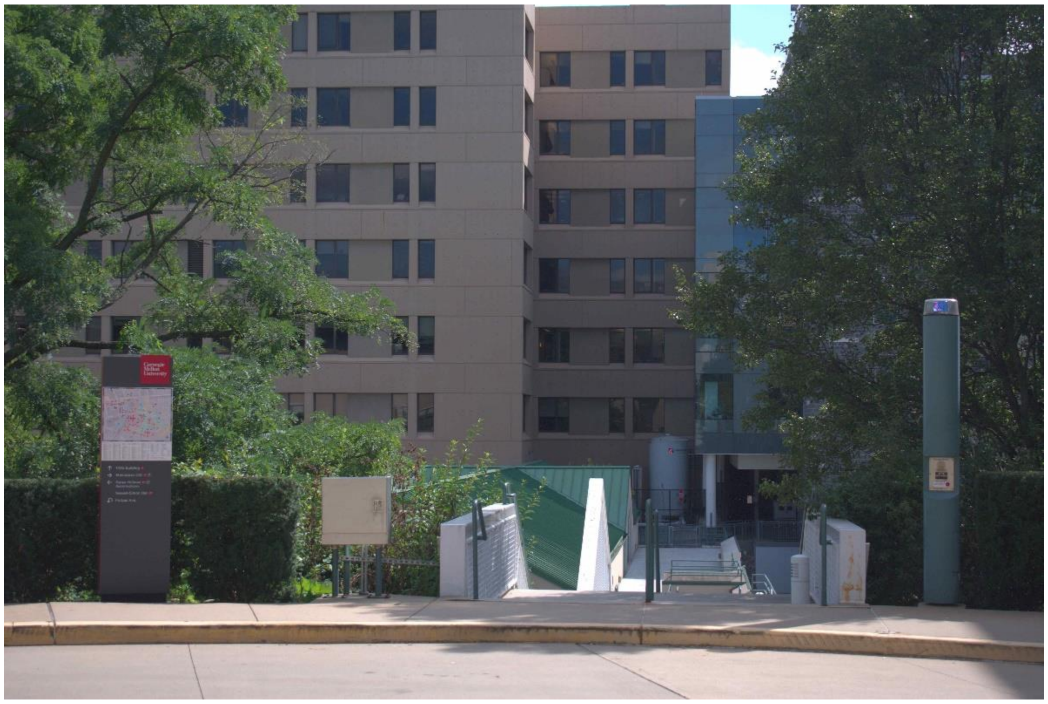
i. Compression

As can be seen from the images below, the differences between the PNG file and the JPG file with .95 compression ratio is **indistinguishable**. And also, lowest setting for which the compressed image is indistinguishable from the original is the JPG image with **.35** quality. Compression ratio between compressed file (.jpeg with .95 compression) and uncompressed file (.png) is **0.19.** Compression ratio between compressed file (.jpeg with . 35 compression which is the lowest setting for which the compressed image is indistinguishable from the original) and uncompressed file (.png) is **0.029**.

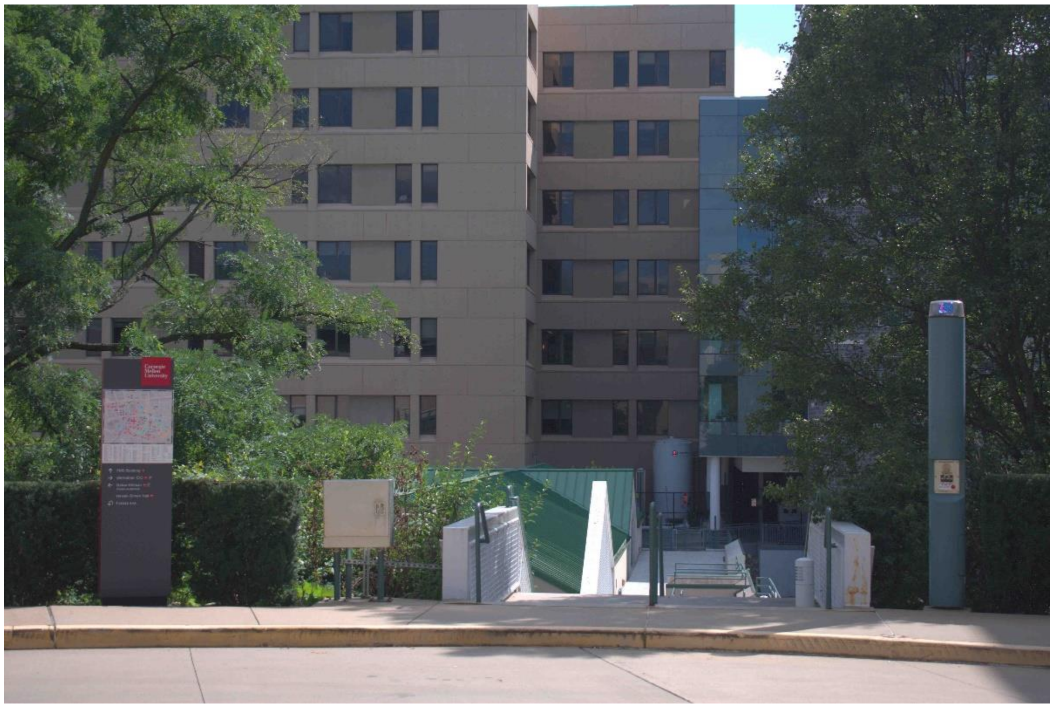
.PNG file:

.JPG file with .95 quality:

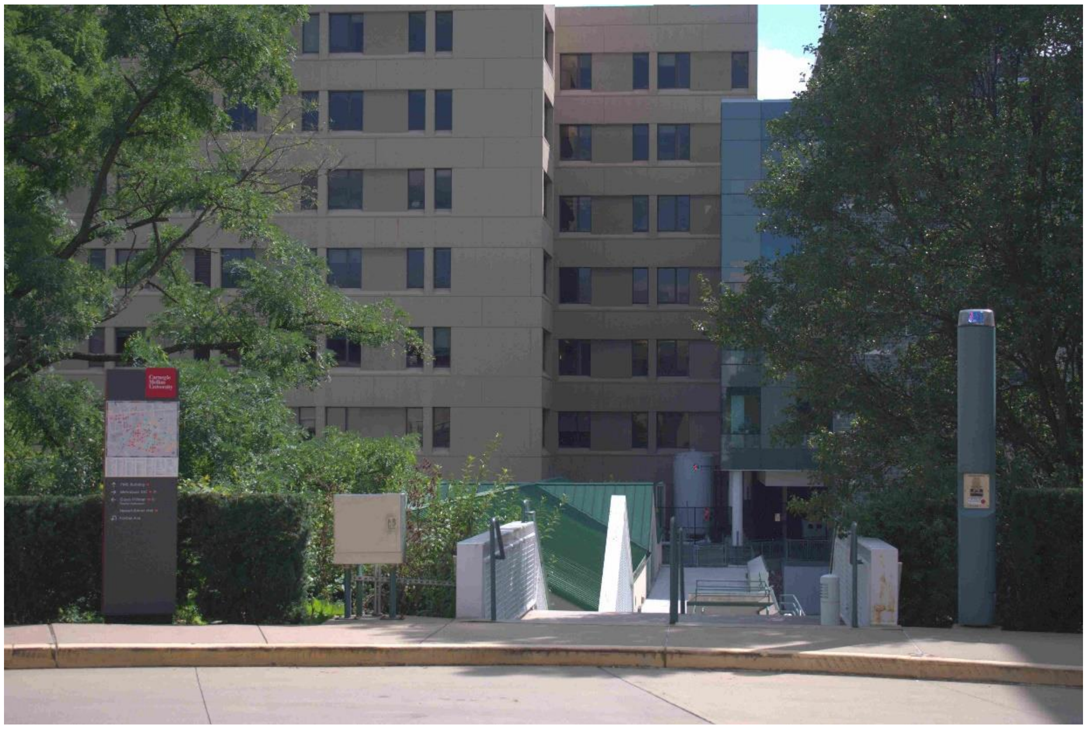
.JPG file with .35 quality:



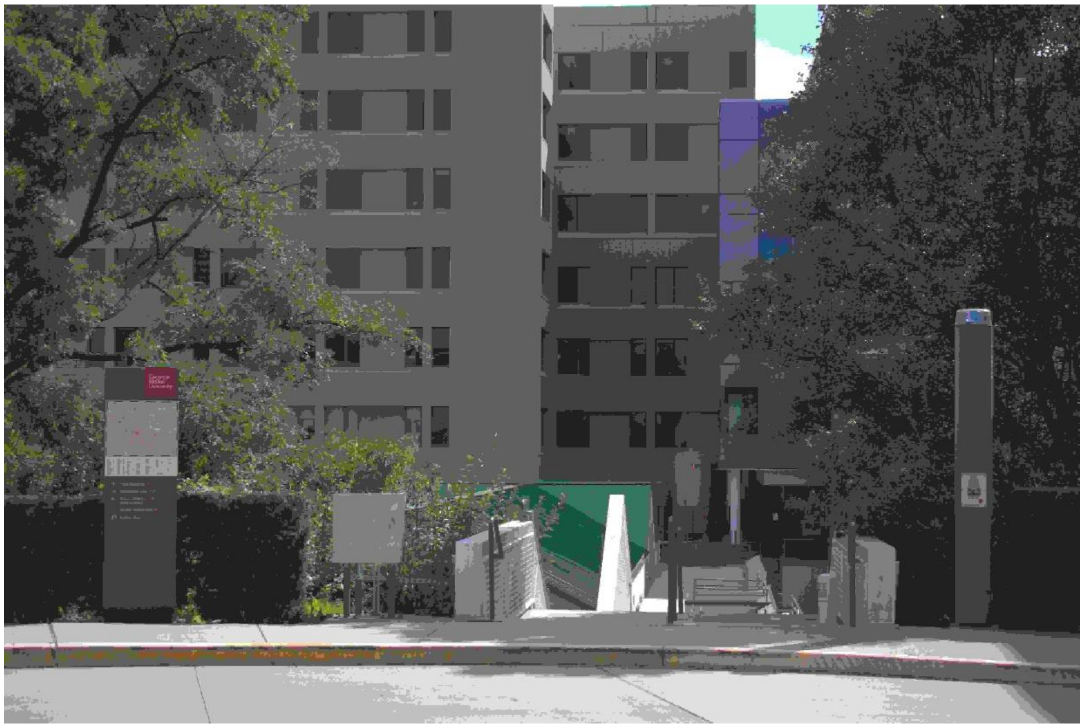
.JPG file with .25 quality:



.JPG file with .15 quality:



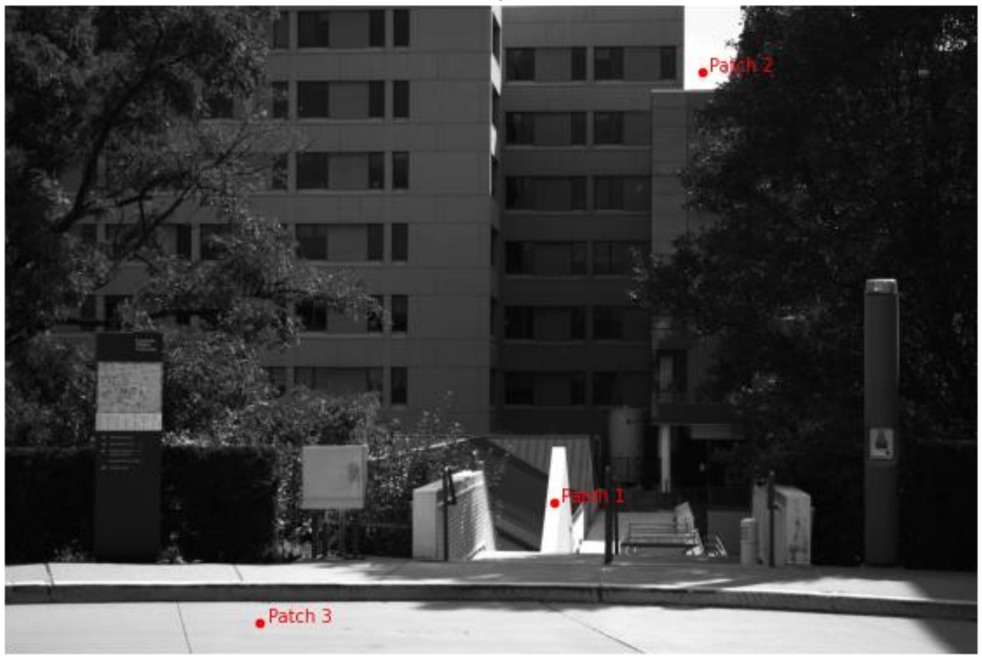
.JPG file with.1 quality:



2. Perform manual white balancing

We first determine the patch coordinates, which we expect to be white. We use ginput to select the coordinates interactively from the image:

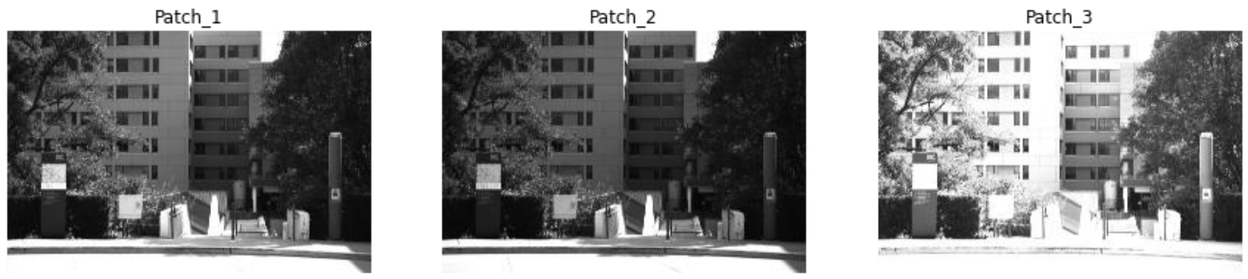
Selected patches



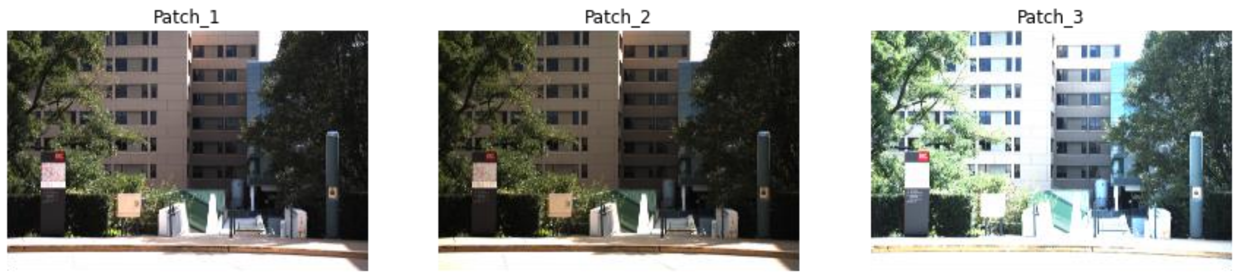
Since we have 'rggb' Bayer pattern, we can create an array that composes of rggb pattern, which helps us to determine channel of the selected coordinates from the image. Then, we can create the patch(rggb block). After assigning intensity values for each channel and for each patch, we find the weights (a,b,c) that make the red, green, and blue channel values of this patch be equal. We choose the x value such that it will be the white color is represented in the color space. So we choose x as the maximum intensity value in the image array

$$\begin{pmatrix} x \\ x \\ x \end{pmatrix} = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix} . \begin{pmatrix} R_{patch} \\ Gpatch \\ Bpatch \end{pmatrix}$$

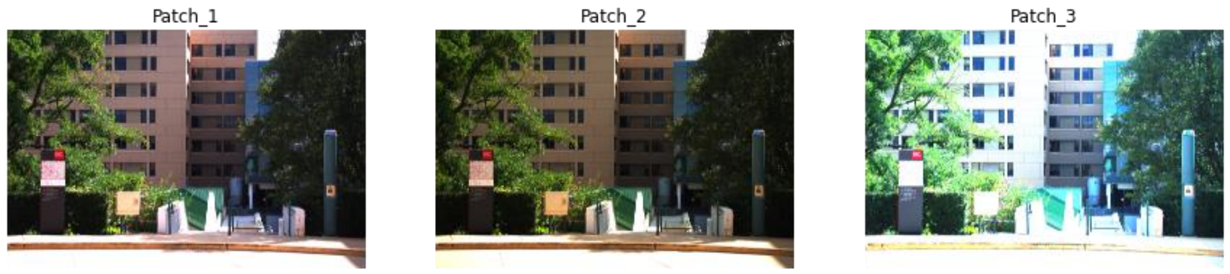
After getting weights for three different patches, we normalize all three channels by using these weights (**Manual white balancing**):



After demosaicing:

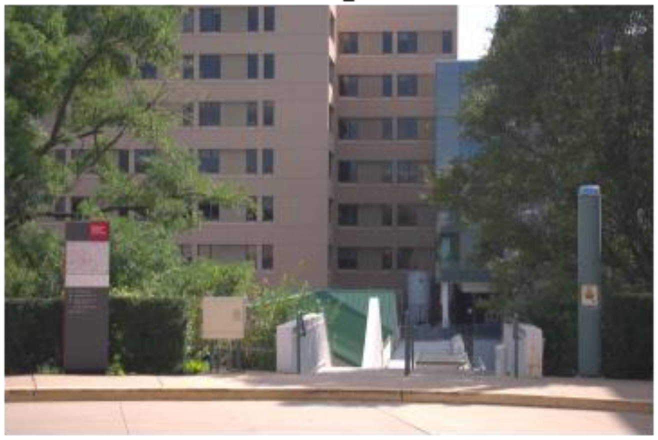


After color space correction:

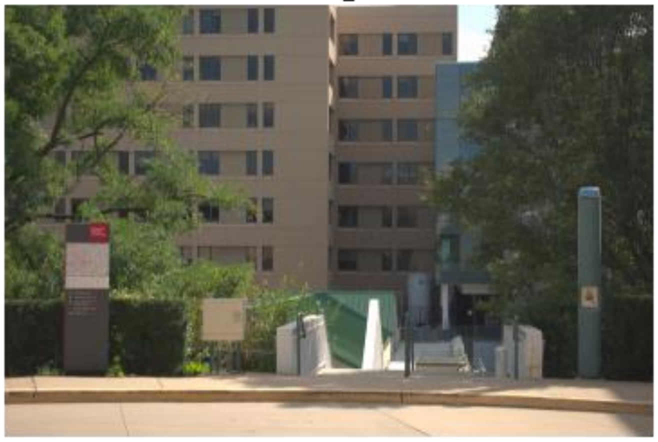


After brightness and gamma encoding:

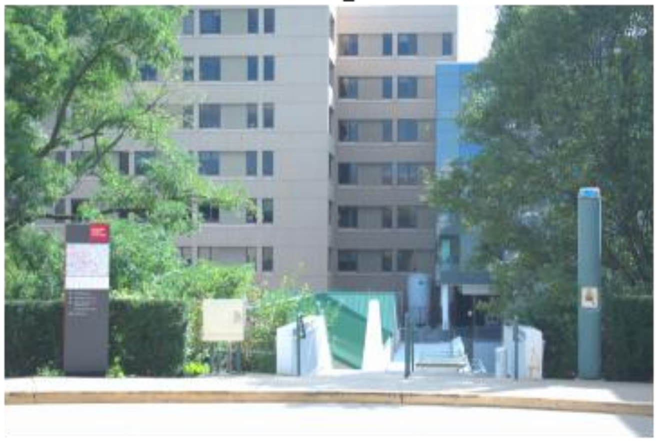
Patch_1



Patch_2



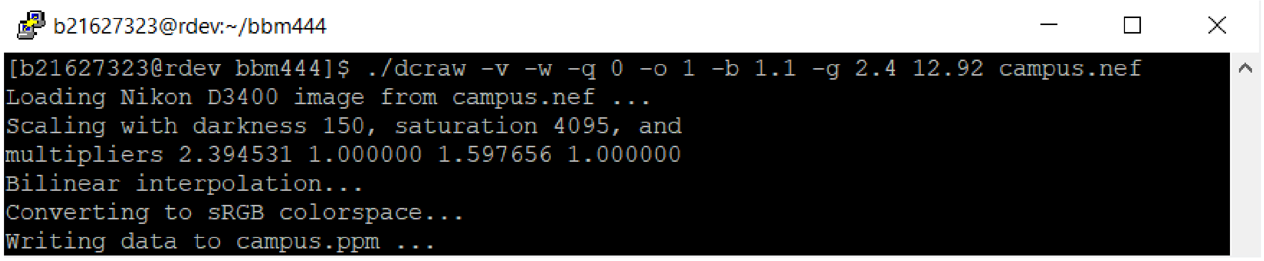
Patch_3



3. Learn to use dcraw

We run the dcraw with the following flags to perform all the image processing pipeline steps we implemented in Python.

```
dcraw -v -w -q 0 -o 1 -b 1.1 -g 2.4 12.92 campus.nef
```



```
[b21627323@rdev bbm444]$ ./dcraw -v -w -q 0 -o 1 -b 1.1 -g 2.4 12.92 campus.nef
Loading Nikon D3400 image from campus.nef ...
Scaling with darkness 150, saturation 4095, and
multipliers 2.394531 1.000000 1.597656 1.000000
Bilinear interpolation...
Converting to sRGB colorspace...
Writing data to campus.ppm ...
```

Used flags:
- `-w` White Balancing: Image is white balanced using the camera data.
- `-q 0` Demosaicing: Set interpolation algorithm to high-speed, low-quality bilinear interpolation.
- `-o 1` Color Space Correction: Set the output color space to sRGB D65.
- `-b 1.1` Brightness: Adjust brightness.
- `-g 2.4 12.92` Gamma Encoding (Tone Reproduction): Set sRGB tonal curve , the numbers correspond to the tone reproduction curve specified in the sRGB standard.
- `-v` Provides textual information about the development process.

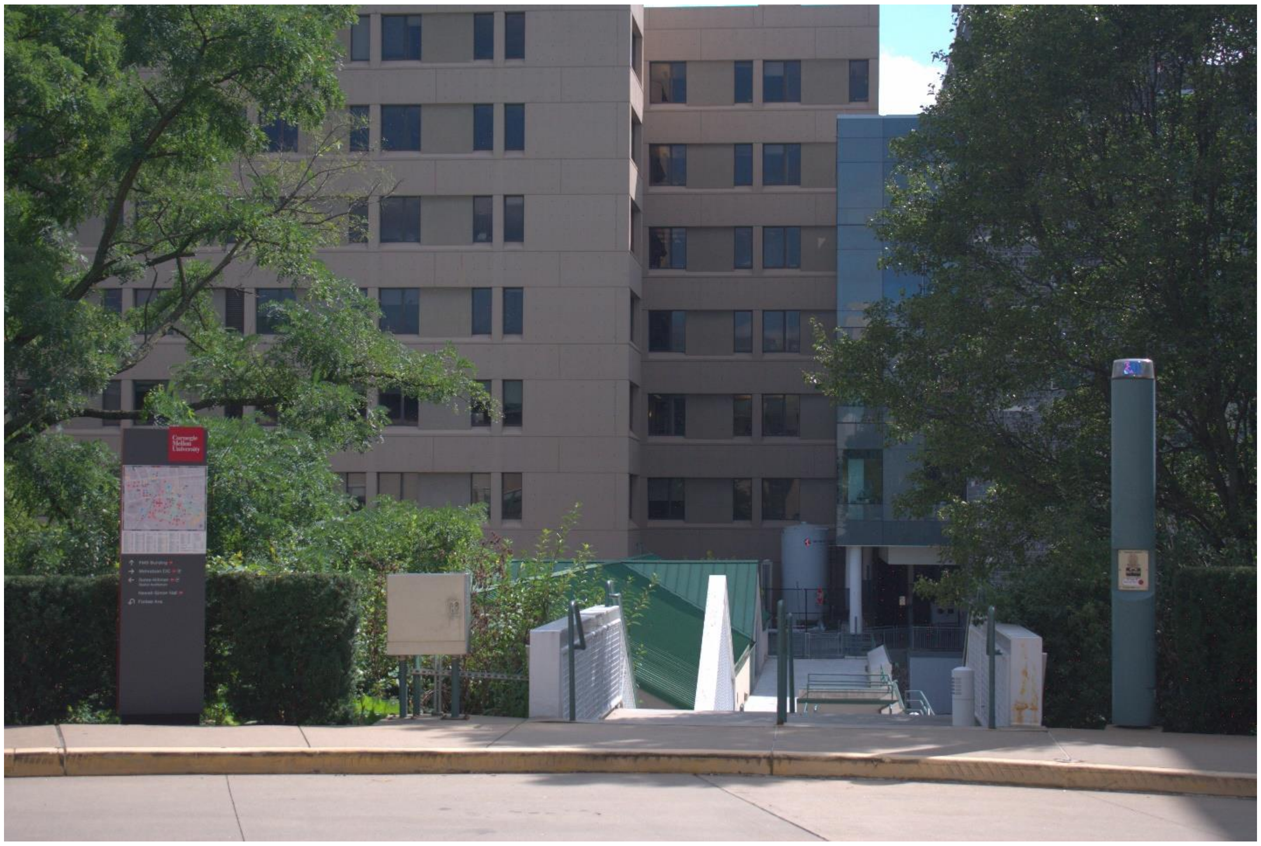
After we get image file with PPM extension, we convert it to a file with PNG extension by using *ImageMagick*:

```
magick campus.ppm campus.png
```

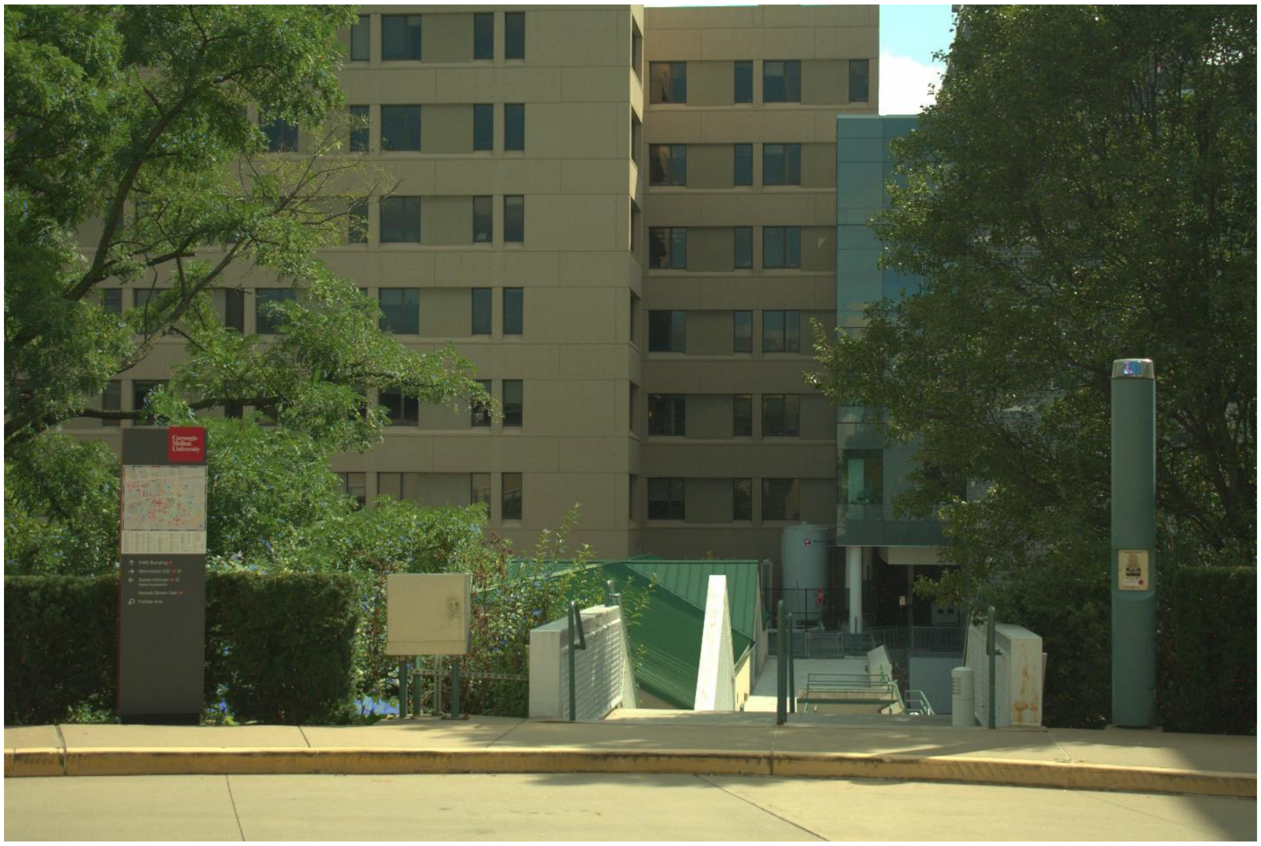
Lastly, we compare the 3 developed images which are the image produced by the camera's own image processing pipeline, the image produced using dcraw and the image produced using our implementation of the image processing pipeline. I think, the image produced using our implementation of the image processing pipeline with manual white balance (patch 2) and with AWB (gray-world) look pleasant and resembles most closely to the developed version of the RAW image produced by the camera's own image processing pipeline. I think, the main differences between the images that we produce, and the image produced by the camera's own image processing pipeline are the contrast and white balance. The original image has more contrast and a different white balance setting.

In next pages you can find the developed version of the RAW image produced using *dcraw,* as well as other output images.

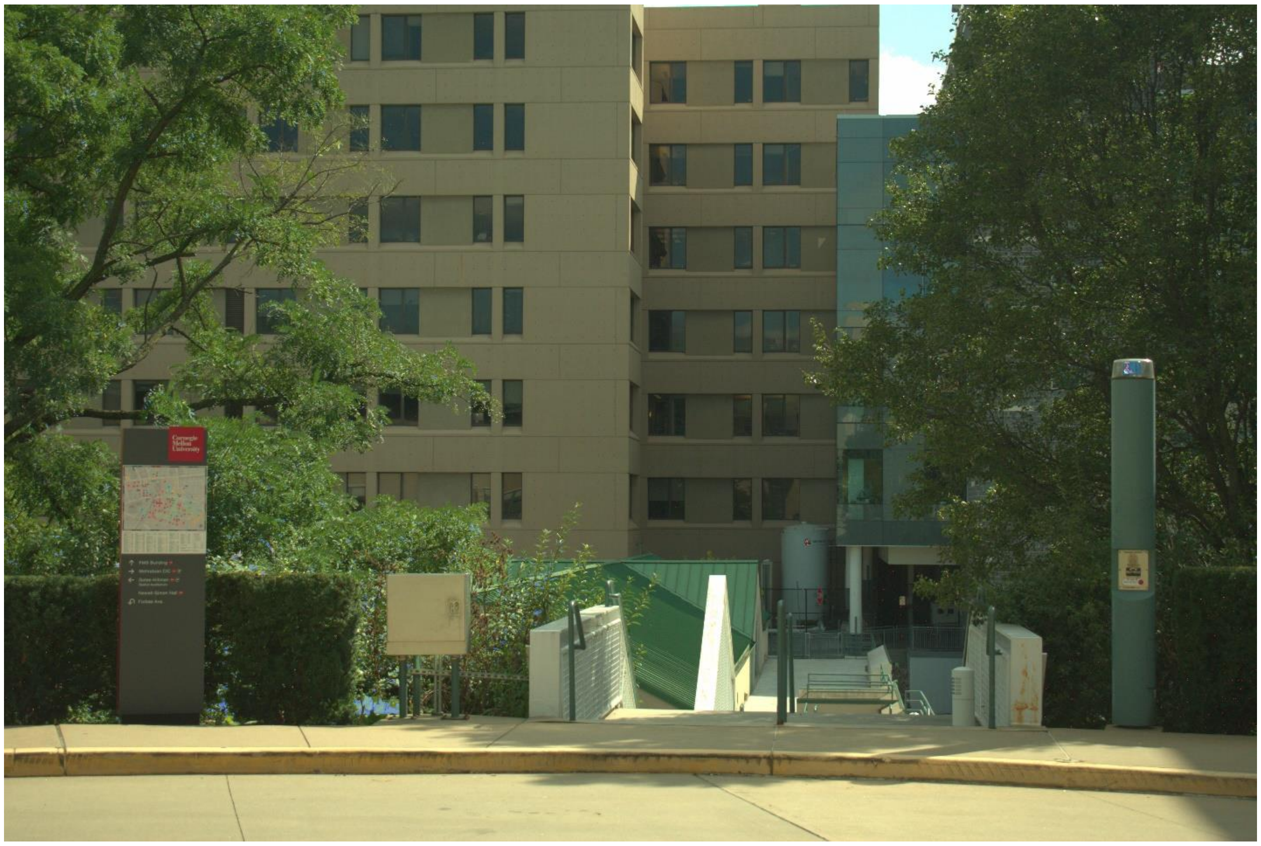
Developed version of the RAW image produced using our implementation of the image processing pipeline (AWB with gray world algorithm):



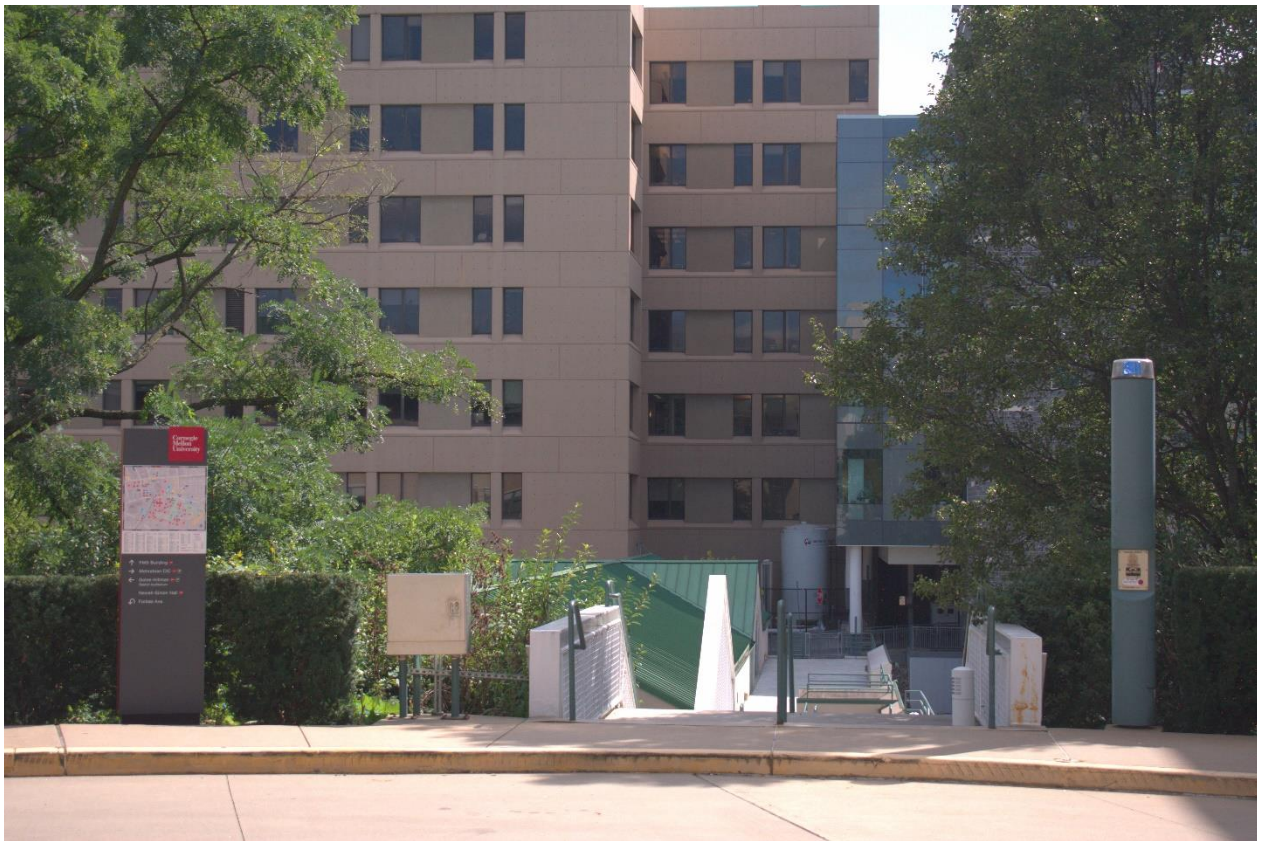
Developed version of the RAW image produced using our implementation of the image processing pipeline (AWB with white world algorithm):



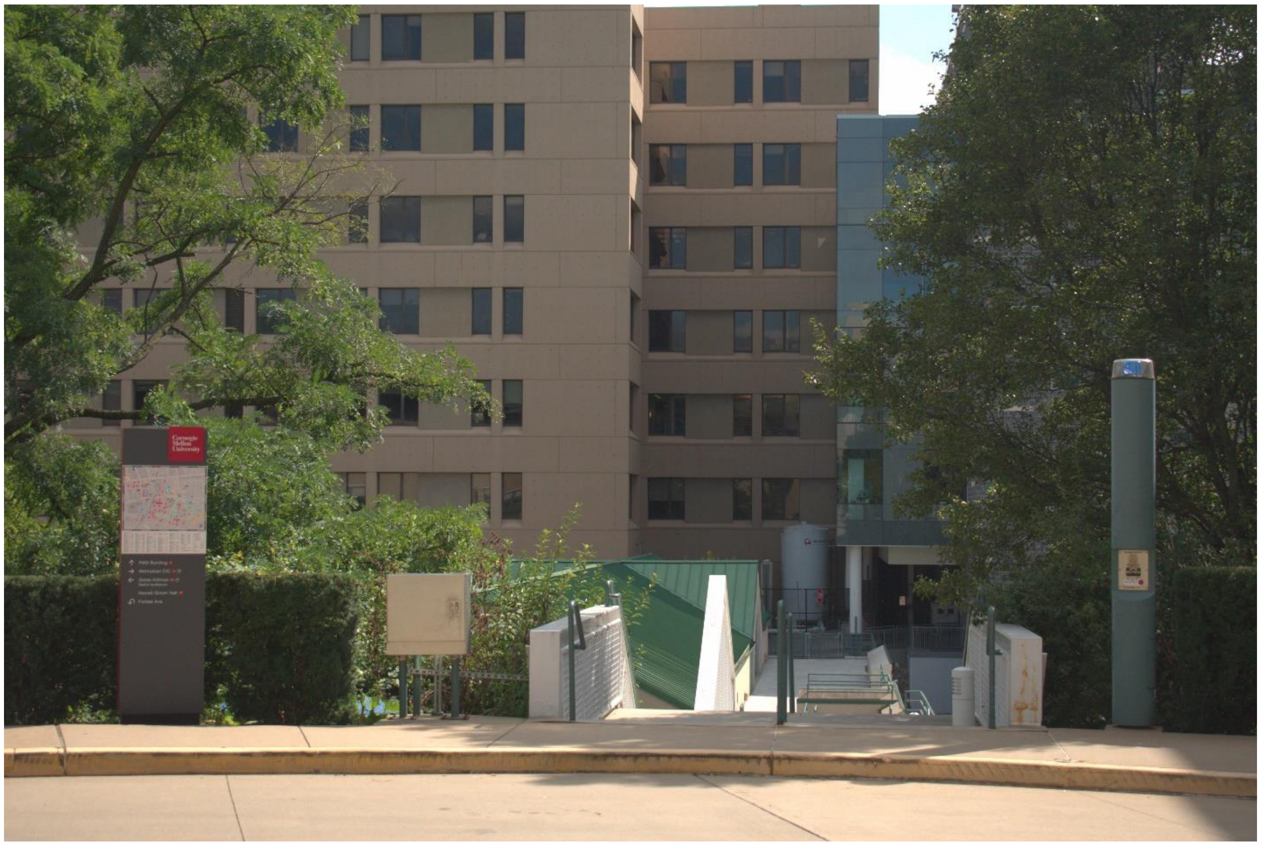
Developed version of the RAW image produced using our implementation of the image processing pipeline (AWB with scale factors that we have obtained from the reconnaissance run):



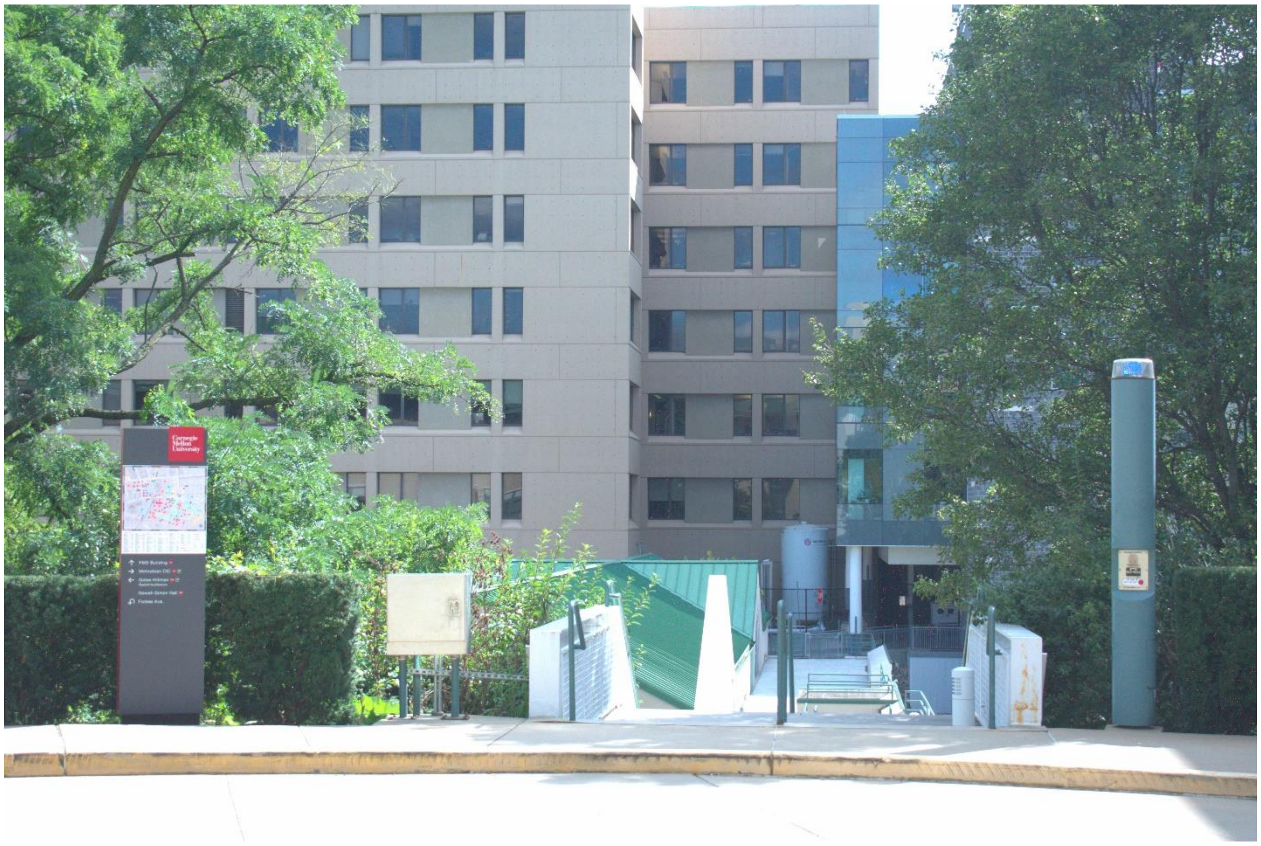
Developed version of the RAW image produced using our implementation of the image processing pipeline (Manual White Balance - Patch 1):



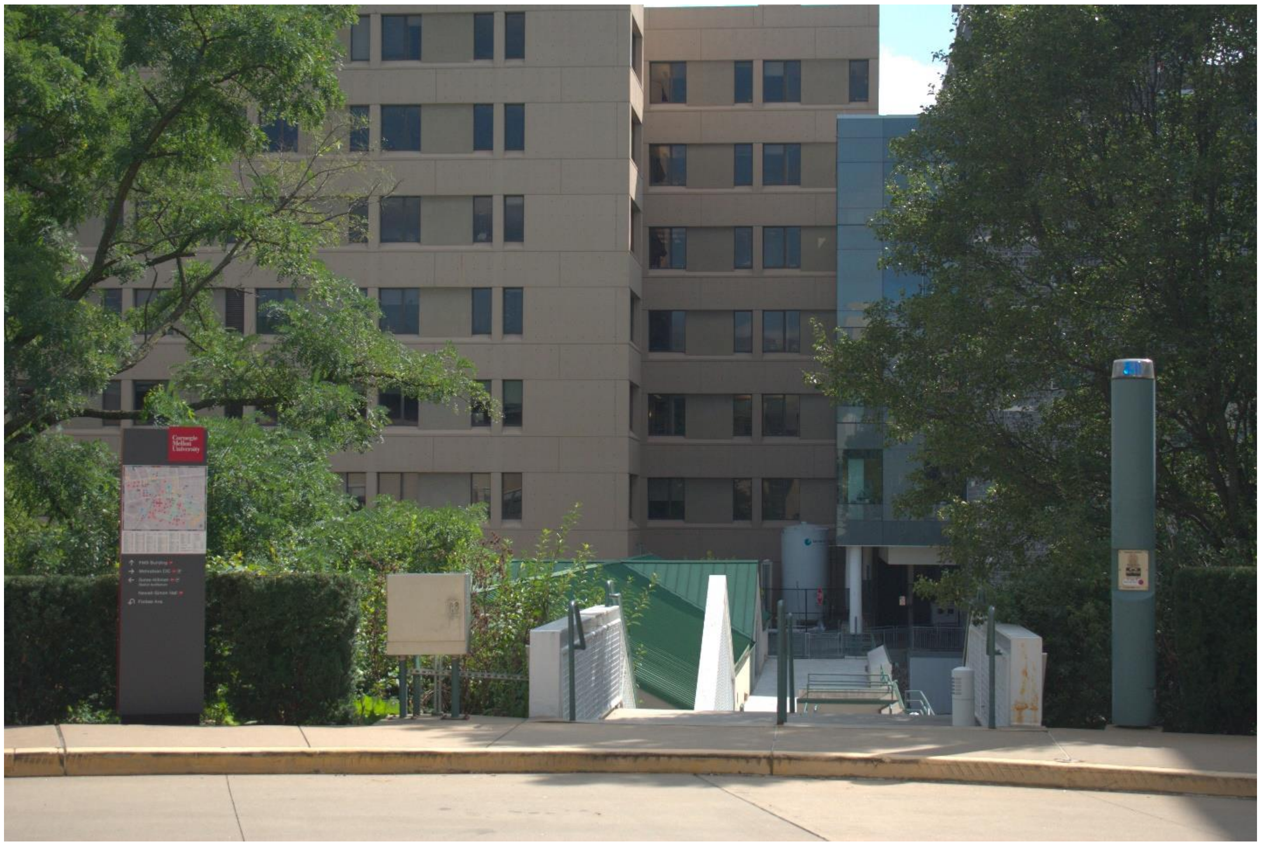
Developed version of the RAW image produced using our implementation of the image processing pipeline (Manual White Balance - Patch 2):



Developed version of the RAW image produced using our implementation of the image processing pipeline (Manual White Balance - Patch 3):



Developed version of the RAW image produced using *dcraw*:



Developed version of the RAW image produced by the camera's own image processing pipeline: