Assignment 1

Anirudha Ramesh (aramesh3)

September 16, 2022

Please look at the jupyter code submitted along-side. There are numerous comments describing what we are doing and answering/elaborating on some answers as well as having the necessary visualizations.

1. Implement a basic image processing pipeline (80 points)

   (a) RAW image conversion (5 points).
   Output :
   Loading Nikon D3400 image from data/campus.nef ...
   Scaling with darkness 150, saturation 4095, and
   multipliers 2.394531 1.000000 1.597656 1.000000
   Building histograms...
   Writing data to data/campus.tiff ...

   (b) Python initials :
   Bits per pixel : 16 (uint type)
   Shape : 4016, 6016

   (c) Think of a way to identify the right bayer pattern.
   Please find attached code for some experiments.
   We initially get a hunch by sampling patches, and then grouping pixels in them. If there is a diagonal with similar entries, this diagonal is likely to correspond with green pixels.
   We also create images with all different possible variations and try to see which looks most reasonable. We know we expect a green tint because we have twice as many green pixels.

   Through this we identify that this is either rggb or bggr. After white-balancing we can see that the rggb images look more like what we would expect in the real world. Notice rggb greyworld and bggr greyworld image colorations.

   (d) White-Balancing
   All images are displayed in the attached code. To me, best looking one is rggb_greyworld and (followed by) rggb_dcraw_scaled, which corresponds to multiplication based on the numbers given by dcraw. We will use rggb_greyworld as our primary image for other downstream tasks.

   (e) Demosaicing In code.

   (f) Color Space Correction In code.

   (g) Brightness Adjustment and Gamma Encoding
   We've displayed some of the tried linear scaling settings. A lot of values, particularly greens and the blues seem to go over the threshold which makes our image look over-exposed.
   There is a general over-exposure towards the sky when we do this, the other developed images

suffer from this as well, and in fact linearly descaling helps see the blues better, but this makes some other areas duller as well. To me however, this looks less over-exposed, and so i prefer this overall.
This causes us to lose a certain 'pop' but we gain a lot more clarity at many places. We perform better than the given image and dcraw developed image at some spots, for example, both of them completely lose information about the bright-white metallic structure at the center of the image, but our development preserves this.

(h) Compression

Visually I cannot tell the difference between png and jpeg-95.

Uncompressed png file size : 32,745,238 bytes
Compressed jpeg 95 file size : 6,028,115 bytes

Compression ratio = $32,745,238/6,028,115 = 5.43$

The artifacting becomes apparent at quality=30, particularly upon zooming, but is still 'passable' but not indistinguishable. From 20, it is not passable.
The compression ratio at quality=30 is,
Compression ratio = $32,745,238/867,827 = 37.732$

2. Manual White Balancing

Manual White balancing suffers from the same problems as automatic white-balancing. That is, in our image, comes out looking very green after demosaicing, and right through the pipeline right after that. Images developed after white balancing from points in different regions, and different kinds of white shown in code.
We can remedy this by aggressively scaling down the values in the green channel.

We perform best when we sample from a point on the electronics box. This is white-brown in color, and in fact is closer to our grey-world assumption rather than white-world assumption.
Other patches still largely give us a green hue overall.
The patch on the map does slightly better, but this is also more on the white-brown color spectrum.
Perhaps this image, and this setting, along with the amount of exposures we have in the white regions make this image ill-suited for white-world assumption in general, and both our other automatic white-balancing schemes perform better.

3. Learn to dcraw

The command we used is below,
./dcraw -v -w -o 1 data/campus.nef

-v : verbose
-w : white-balance with manufacturer given colors
-o 1 : output color-space srgb

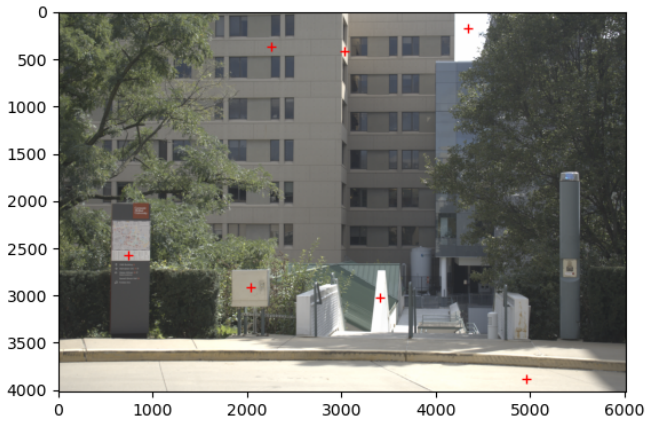
4. Comparing the three developed images

   I believe the image we've developed is the best of all, followed by the dcraw image, followed by the given image.
   As highlighted before, we capture more fine features, and work better against over-exposure. Our image also looks more 'realistic' in terms of the colors, the trees for example have an artificial pop in the other two images, particularly the given image. We have a general cooler shade to the image as well, which I personally like better. The primary reason for my choice is however feature preservation against over-exposure (notice the marks on the central white object on our image, and how much harder they are to spot in the other images, as well as the sky).

5. Camera Obscura

   (a) Build the Pinhole Camera
   All measurements are approximate.
   Screen Size : 10 * 30 cm
   Focal Length : 40 cm
   Field of View : 41 degrees across the horizontal axis, 15-16 on the vertical axis
   Most of these choices were forced by resources (aka box) I had access to, but have looked like sound choices.
   Find images of the camera below.

Figure 1: See all selections, explanation of what they are, showing results with these points selected.



```
# Points selected
# %matplotlib inline # for notebook
selected_pts_gin = plt.imread('ginput_selections.png')
plt.imshow(selected_pts_gin)
plt.show()

# Points previously selected
gin = [(3407.7580645161297, 3021.4870967741936), # whitest point at center towards the bottom
 (739.3709677419356, 2572.7129032258067), # on the map on the left (white-brown)
 (2037.1774193548388, 2912.325806451613), # on the electronics box (white-brown)
 (2255.5, 365.22903225806476), # Building flat plane towards camera (white-brown)
 (3031.758064516129, 413.74516129032236), # Building sideways plane (brightness) (white-ish)
 (4341.693548387097, 171.16451612903256), # sky background (white)
 (4960.274193548387, 3882.648387096774)] # floor at bottom (white-ish)
```

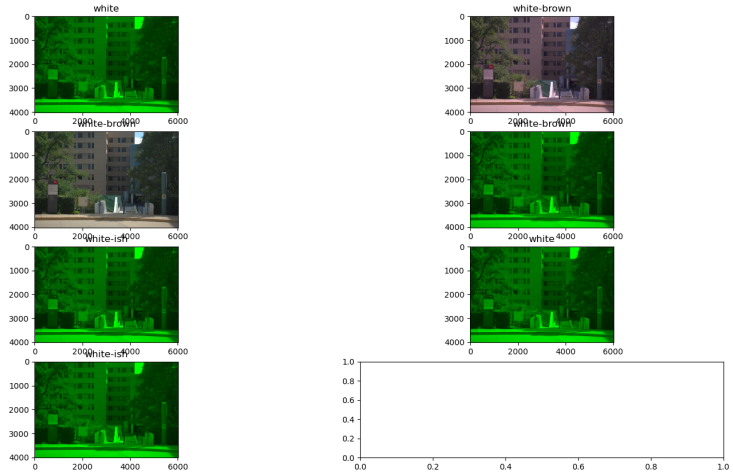
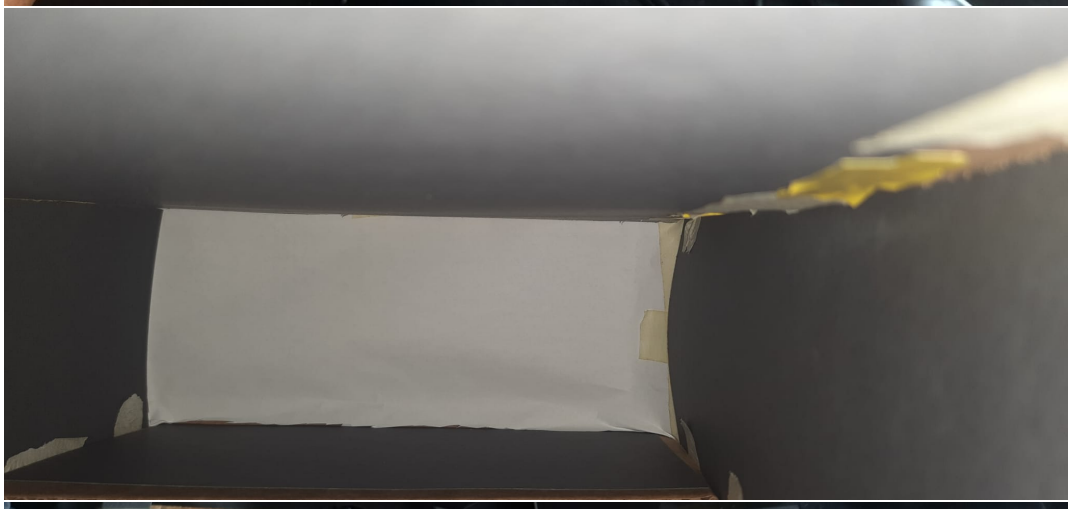


Figure 2: Given image, dcraw developed image, our image.


Flatten Before Recycling
PAPER BOX
INTERNATIONAL PAPER
EDGE CRUSH
TEST (ECT)
SAN ANTONIO TX
free listen
prim


prim




PAPER
BOX
prim

I also built a version with cloth instead of paper hoping the diffused nature of its reflection would darken the box more, but this is counteracted by the fact that holding cloth taut throughout is a lot more challenging to do so with just tape. This done right can however improve our camera.

(b) Use your pinhole Camera

According to the formula, our ideal pinhole size should be around 0.1mm.
We roughly use the three measurements suggested.
Lets call them small, medium, large.

For this comparison we have chosen three very different scenes. The first is a bright outdoor scene, the second a dim outdoor scene, the third a dim indoor scene with a bright outdoor background (window sill).
Following these, we show many more images taken with our 'preferred' pinhole.
Most displayed images aren't cropped to just show the screen, rather we show the pinhole as well to get a sense of the entire shot as it is.

Looking at these and many more trial examples we chose to use the medium sized pinhole for the rest of our photos.
The general trade-off between sizes is as size increases, we get more light, but our image is more blurred. On making pinhole smaller, our image gets sharper, but in turn becomes less bright which also makes it suffer from a reduced signal to noise ratio.

Figure 3: Images corresponding to small, medium, and large pinholes in order.

Figure 4: Images corresponding to small, medium, and large pinholes in order.

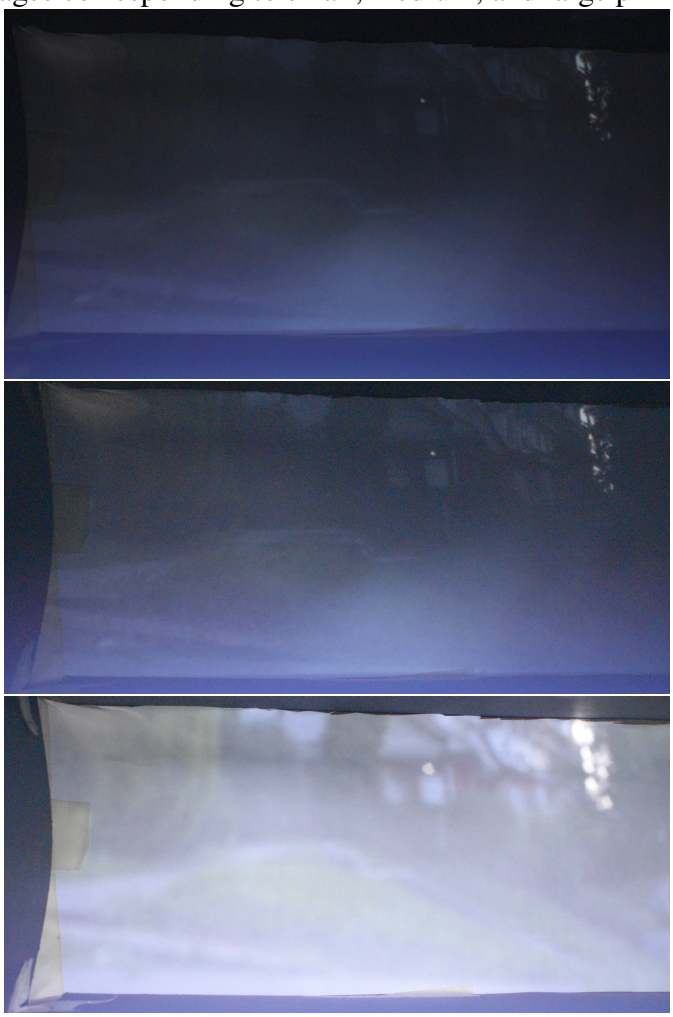


Figure 5: Images corresponding to small, medium, and large pinholes in order.