hw2-my-hdr-stack

October 4, 2022

[17]:

```
import numpy as np
import cv2
# import skimage
import matplotlib.pyplot as plt
import src.cp_hw2
# import src.cp_exr
```

[21]:

```
# Q1.2) Linearize Rendered Images

# Read images
image_files = ['data/gphoto-exposure-stack-dump/exposure'+str(i)+'.jpg' for i␣
 ↪in range(1,19)]
images_stack_for_lin = []
for imf in image_files :
    img = plt.imread(imf)
    img = img[::200, ::200]
    images_stack_for_lin.append(img)
images_stack_for_lin = np.array(images_stack_for_lin)
images_stack_for_lin_reshaped = images_stack_for_lin.
 ↪reshape(images_stack_for_lin.shape[0],-1).swapaxes(0,1)
```

[22]:

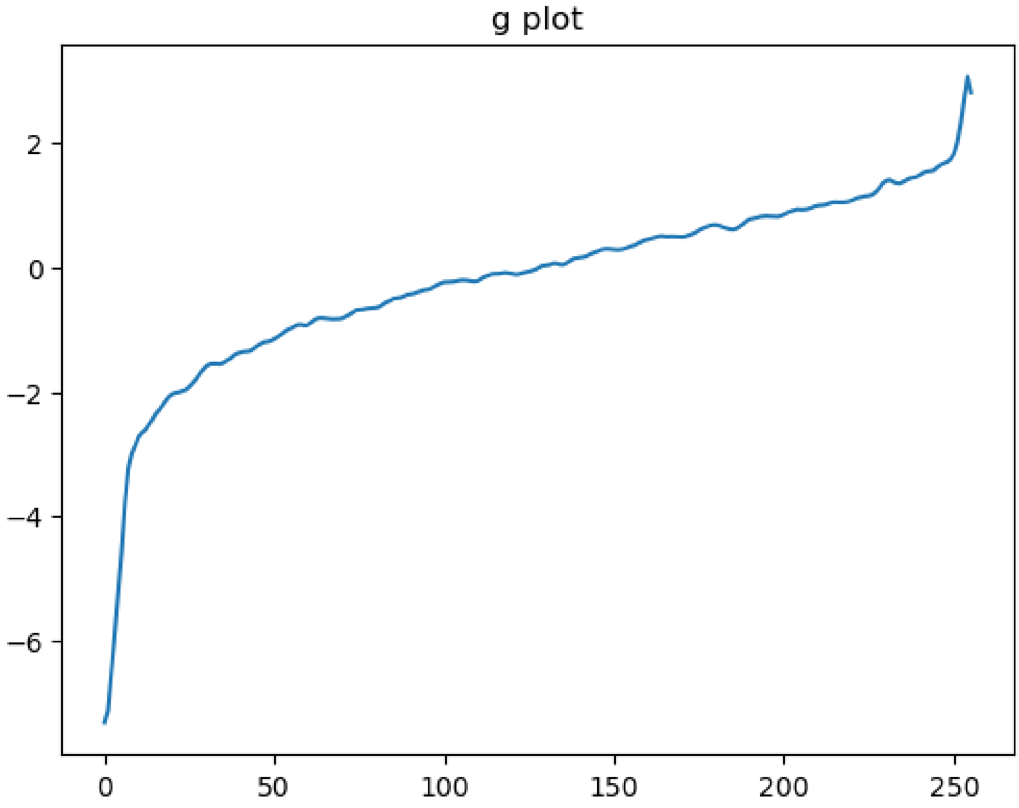
```
# t_filler = lambda x : 2.0 ** (x-1-12)
# t = [t_filler(i) for i in range(1,19)]
t = [1/4000,1/2000,1/1000,1/500,1/250,1/125,1/60,1/30,1/15,1/8,1/4,1/
 ↪2,1,2,4,8,15,30]
# variable t used in general scope later on in the program
def get_g(images_stack_reshaped, images_stack, w, t, r_l=1, is_w_photon=False) :

    n = 256;
    A = np.zeros((images_stack_reshaped.shape[0]*images_stack_reshaped.
 ↪shape[1]+n+1,
                  n+images_stack_reshaped.shape[0]))
    b = np.zeros((A.shape[0],1))
    cur_row_A = 0
    for i in range(images_stack_reshaped.shape[0]) : # Corresponds to pixel in␣
 ↪image
```

```
        for j in range(images_stack_reshaped.shape[1]) : # Corresponds to image␣
↪in sequence
    #         wt_ij = 1 # placeholder
            if is_w_photon :
                wt_ij = w(images_stack_reshaped[i,j], t[j])
            else :
                wt_ij = w(images_stack_reshaped[i,j])
#             print(wt_ij)
            A[cur_row_A, images_stack_reshaped[i,j]] = wt_ij
            A[cur_row_A, n+i-1] = -wt_ij
            b[cur_row_A, 0] = wt_ij * np.log(t[j])
            cur_row_A = cur_row_A + 1

    # Adding the terms corresponding to the smoothing regularization
    for i in range(n) :
        if is_w_photon :
            wt_i = 1
        else :
            wt_i = w(i)
        A[cur_row_A,i-1] = r_l * wt_i
        A[cur_row_A,i] = -2*r_l*wt_i
        A[cur_row_A,i+1] = r_l*wt_i
        cur_row_A += 1

    # Normalize the curve by setting its middle value to 0
    A[cur_row_A,128] = 1 # Corresponds to g128 in g0-g255

    v = np.linalg.lstsq(A, b, rcond=None)
#     print(v)
    v_sol = v[0]
    g = v_sol[:n]
#     L = v_sol[n:]
#     L = L.reshape(images_stack.shape[1], images_stack.shape[2], images_stack.
↪shape[3])

    return g

w = lambda x, zmin=0.05, zmax=0.95 : 0.01
g = get_g(images_stack_for_lin_reshaped, images_stack_for_lin, w, t, 10)
```

[23]:

```
# Plotting g
plt.plot(np.arange(0,256),g)
plt.title('g plot')
plt.show()
# Linearizing the image
# The other image is too small to make sense of, so looking at a larger image
images_stack = []
```

```
for imf in image_files :
    img = plt.imread(imf)
    img = img[::20, ::20]
    images_stack.append(img)
images_stack = np.array(images_stack)

def linearize_image(images_stack, g) :
    lin_images_stack = np.exp(g[images_stack])
    lin_images_stack = lin_images_stack.squeeze(-1)
    return lin_images_stack

lin_images_stack = linearize_image(images_stack, g)
plt.imshow(lin_images_stack[14])
# display_hdr_image(lin_images_stack[8],0.05)
```



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

[23]:
```
<matplotlib.image.AxesImage at 0x7f9fd7c61690>
```

[24]:
```
# Q1.3) Merge exposure stack into HDR image (30 points)
# We've already loaded .jpg images. Lets load .tiff images as well.
image_files_tiff = ['data/gphoto-exposure-stack-dump/exposure'+str(i)+'.tiff'␣
 ↪for i in range(1,19)]
images_stack_tiff = []
for imf in image_files_tiff :
#     img = plt.imread(imf)
    # TO CHANGE TODOOOOO
    img = cv2.imread(imf, -1)
    img = img[...,::-1]
    img = img[::5, ::5]
    images_stack_tiff.append(img)
images_stack_tiff = np.array(images_stack_tiff)
```

[25]:
```
images_stack_normalized = images_stack / 255.0
images_stack_tiff_normalized = images_stack_tiff / (2**16 - 1)
#
# Over-exposure vs under-exposed pixel selection is still off. Fix later!
def linear_merging(im_s, lin_im_s, t, w_v, is_w_photon=False) :

    I_hdr_num = np.zeros(im_s[0].shape)
    I_hdr_den = np.zeros(im_s[0].shape)
    for k in range(len(im_s)) :
        if is_w_photon :
            if np.max(lin_im_s) > 256 :
                w_v_imk = w_v(im_s[k],t[k],0.0001, 0.995)
            else :
                w_v_imk = w_v(im_s[k],t[k],0.00001, 0.92)
        else :
            if np.max(lin_im_s) > 256 :
                w_v_imk = w_v(im_s[k], 0.02, 0.98)
            else :
                w_v_imk = w_v(im_s[k])
        I_hdr_num += (w_v_imk * lin_im_s[k] / t[k] )
        I_hdr_den += (w_v_imk)

    I_hdr = I_hdr_num / I_hdr_den

    # where num is zero, hdr final should also be zero
    I_hdr_num_zero = np.where(I_hdr_num == 0)
    if len(I_hdr_num_zero) == 3 :
        I_hdr_num_zero = np.vstack((np.
↪vstack((I_hdr_num_zero[0],I_hdr_num_zero[1])), I_hdr_num_zero[2])).T
    else :
        I_hdr_num_zero = np.vstack((I_hdr_num_zero[0],I_hdr_num_zero[1])).T


    # where num is not zero, but den is zero, all hdr final values should be␣
↪max val
    I_hdr_den_zero = np.where(I_hdr_den == 0)
    if len(I_hdr_den_zero) == 3 :
        I_hdr_den_zero = np.vstack((np.
↪vstack((I_hdr_den_zero[0],I_hdr_den_zero[1])), I_hdr_den_zero[2])).T
    else :
        I_hdr_den_zero = np.vstack((I_hdr_den_zero[0],I_hdr_den_zero[1])).T

    I_hdr_max_val = []

    if I_hdr_num_zero.size > 0 :
        if len(I_hdr.shape) == 3 :
            I_hdr[I_hdr_num_zero[:,0],I_hdr_num_zero[:,1],I_hdr_num_zero[:,2]]␣
↪= 0
        else :
            I_hdr[I_hdr_num_zero[:,0],I_hdr_num_zero[:,1]] = 0

    if I_hdr_den_zero.size > 0 :
        for index in I_hdr_den_zero :
            if len(I_hdr.shape) == 3 :
                if I_hdr_num[index[0],index[1],index[2]] != 0 :
                    I_hdr_max_val.append(index)
            else :
                if I_hdr_num[index[0],index[1]] != 0 :
                    I_hdr_max_val.append(index)
        I_hdr_max_val = np.array(I_hdr_max_val)
        if I_hdr_max_val.size > 0 :
            I_hdr_max_val = np.stack(I_hdr_max_val)
            if len(I_hdr.shape) == 3 :
                I_hdr[I_hdr_max_val[:,0],I_hdr_max_val[:,1],I_hdr_max_val[:,2]]␣
 ↪= np.nanmax(I_hdr)
            else :
                I_hdr[I_hdr_max_val[:,0],I_hdr_max_val[:,1]] = np.nanmax(I_hdr)

    return I_hdr

def linear_merging_channelwise(im_s, lin_im_s, t, w_v, is_w_photon=False) :

    I_hdr = np.zeros(im_s[0].shape)
    I_hdr[...,0] = linear_merging(im_s[...,0], lin_im_s[...,0], t, w_v,␣
 ↪is_w_photon)
    I_hdr[...,1] = linear_merging(im_s[...,1], lin_im_s[...,1], t, w_v,␣
 ↪is_w_photon)
    I_hdr[...,2] = linear_merging(im_s[...,2], lin_im_s[...,2], t, w_v,␣
 ↪is_w_photon)

    return I_hdr

def log_merging(im_s, lin_im_s, t, w_v, is_w_photon=False) :

    eps = 1e-8
    I_hdr_num = np.zeros(im_s[0].shape)
    I_hdr_den = np.zeros(im_s[0].shape)
    for k in range(len(im_s)) :

        if is_w_photon :
            if np.max(lin_im_s) > 256 :
                w_v_imk = w_v(im_s[k],t[k],0.0001, 0.995)
            else :
                w_v_imk = w_v(im_s[k],t[k],0.00001, 0.92)
        else :
            if np.max(lin_im_s) > 256 :
                w_v_imk = w_v(im_s[k], 0.02, 0.98)
            else :
                w_v_imk = w_v(im_s[k])
        I_hdr_num += ( w_v_imk * ( np.log(lin_im_s[k] + eps) - np.log(t[k])))
        I_hdr_den += w_v_imk

    I_hdr = I_hdr_num / I_hdr_den
    # where num is zero, hdr final should also be zero (regardless of den)
    I_hdr_num_zero = np.where(I_hdr_num == 0)
    I_hdr_num_zero = np.vstack((np.vstack((I_hdr_num_zero[0],I_hdr_num_zero[1])), I_hdr_num_zero[2])).T

    # where num is not zero, but den is zero, all hdr final values should be max val
    I_hdr_den_zero = np.where(I_hdr_den == 0)
    I_hdr_den_zero = np.vstack((np.vstack((I_hdr_den_zero[0],I_hdr_den_zero[1])), I_hdr_den_zero[2])).T
    I_hdr_max_val = []

    if I_hdr_num_zero.size > 0 :
        I_hdr[I_hdr_num_zero[:,0],I_hdr_num_zero[:,1],I_hdr_num_zero[:,2]] = 0

    if I_hdr_den_zero.size > 0 :
        for index in I_hdr_den_zero :
            if I_hdr_num[index[0],index[1],index[2]] != 0 :
                I_hdr_max_val.append(index)
        I_hdr_max_val = np.array(I_hdr_max_val).T
        if I_hdr_max_val.size > 0 :
            I_hdr_max_val = np.stack(I_hdr_max_val)
            I_hdr[I_hdr_max_val[:,0],I_hdr_max_val[:,1],I_hdr_max_val[:,2]] = np.nanmax(I_hdr)

    return np.exp(I_hdr)

# setting w for merging testing
# w_v = np.vectorize(w)

# I_hdr_lin_merged = linear_merging(images_stack_normalized, lin_images_stack, t, w_v)
# I_hdr_tiff_lin_merged = linear_merging(images_stack_tiff_normalized, images_stack_tiff, t, w_v)

# I_hdr_log_merged = log_merging(images_stack_normalized, lin_images_stack, t, w_v)
# I_hdr_tiff_log_merged = log_merging(images_stack_tiff_normalized, images_stack_tiff, t, w_v)

# src.cp_hw2.writeHDR('I_jpg_lin_merged.HDR', I_hdr_lin_merged)
# src.cp_hw2.writeHDR('I_tiff_lin_merged.HDR', I_hdr_tiff_lin_merged)

# src.cp_hw2.writeHDR('I_jpg_log_merged.HDR', I_hdr_log_merged)
# src.cp_hw2.writeHDR('I_tiff_log_merged.HDR', I_hdr_tiff_log_merged)
```

[26]:

```
# Weighting Schemes
def w_uniform(z, Z_min = 0.05, Z_max=0.95) :
    if isinstance(z,np.uint8) or isinstance(z,int):
        z = z / 255.0
    if Z_min <= z <= Z_max :
        return 1
    return 0

def w_tent(z, Z_min = 0.001, Z_max=0.999) :
    if isinstance(z,np.uint8) or isinstance(z,int) :
        z = z / 255.0
    Z_min = 0.001
    Z_max = 0.999
    if Z_min <= z <= Z_max :
        return np.min([z, 1.0-z])
    return 0

def w_gaussian(z, Z_min = 0.001, Z_max=0.999) :
    if isinstance(z,np.uint8) or isinstance(z,int):
        z = z / 255.0
    Z_min = 0.001
    Z_max = 0.999
    if Z_min <= z <= Z_max :
        return np.exp(-4 * (((z-0.5)**2) / ((0.5)**2)))
    return 0

def w_photon(z, tk, Z_min = 0.05, Z_max=0.95) :
# for tiff
#         Z_min = 0.0001
#         Z_max = 0.995
# # for jpeg
#         Z_min = 0.00001
#         Z_max = 0.92

    if isinstance(z,np.uint8) or isinstance(z,int):
        z = z / 255.0
    if Z_min <= z <= Z_max :
        return tk
    return 0
```

[27]:

```
# HDR display util functions
def linear_scaling(img_cc, scale, should_plot=False):
#     img_scaled = np.clip(skimage.color.rgb2gray(img_cc) * scale, 0, 1)
    img_scaled = np.clip(img_cc * scale, 0, 1)
    if should_plot :
        plt.imshow(img_scaled)
        plt.title('linear scaling')
        plt.show()
    return img_scaled

# Gamma Encoding
def gamma_encoding(val) :
    if val <= 0.0031308 :
        return 12.92 * val
    else :
        return (1+0.055) * np.power(val, 1/2.4) - 0.055

def display_hdr_image(img, scale, title='', should_plot=True) :
    img_scaled = linear_scaling(img, scale)
    ge_f = np.vectorize(gamma_encoding)
    image_ge = ge_f(img_scaled)
    if should_plot :
        plt.imshow(image_ge)
        plt.title(title)
        plt.show()
    return image_ge
```

[28]:

```
w_uniform_vectorized = np.vectorize(w_uniform)
w_tent_vectorized = np.vectorize(w_tent)
w_gaussian_vectorized = np.vectorize(w_gaussian)
w_photon_vectorized = np.vectorize(w_photon)

# Creating all merged raw images
I_hdr_tiff_lin_merged_w_uniform = linear_merging(images_stack_tiff_normalized,␣
 ↪images_stack_tiff, t, w_uniform_vectorized)
I_hdr_tiff_lin_merged_w_uniform_channelwise =␣
 ↪linear_merging_channelwise(images_stack_tiff_normalized, images_stack_tiff,␣
 ↪t, w_uniform_vectorized)
I_hdr_tiff_log_merged_w_uniform = log_merging(images_stack_tiff_normalized,␣
 ↪images_stack_tiff, t, w_uniform_vectorized)

I_hdr_tiff_lin_merged_w_tent = linear_merging(images_stack_tiff_normalized,␣
 ↪images_stack_tiff, t, w_tent_vectorized)
I_hdr_tiff_log_merged_w_tent = log_merging(images_stack_tiff_normalized,␣
 ↪images_stack_tiff, t, w_tent_vectorized)

I_hdr_tiff_lin_merged_w_gaussian = linear_merging(images_stack_tiff_normalized,␣
 ↪images_stack_tiff, t, w_gaussian_vectorized)
I_hdr_tiff_log_merged_w_gaussian = log_merging(images_stack_tiff_normalized,␣
 ↪images_stack_tiff, t, w_gaussian_vectorized)
```

```
I_hdr_tiff_lin_merged_w_photon = linear_merging(images_stack_tiff_normalized,␣
 ↪images_stack_tiff, t, w_photon_vectorized, is_w_photon=True)
I_hdr_tiff_log_merged_w_photon = log_merging(images_stack_tiff_normalized,␣
 ↪images_stack_tiff, t, w_photon_vectorized, is_w_photon=True)
```

```
/tmp/ipykernel_5126/2109691339.py:23: RuntimeWarning: invalid value encountered
in divide
  I_hdr = I_hdr_num / I_hdr_den
/tmp/ipykernel_5126/2109691339.py:95: RuntimeWarning: invalid value encountered
in divide
  I_hdr = I_hdr_num / I_hdr_den
```

[29]:

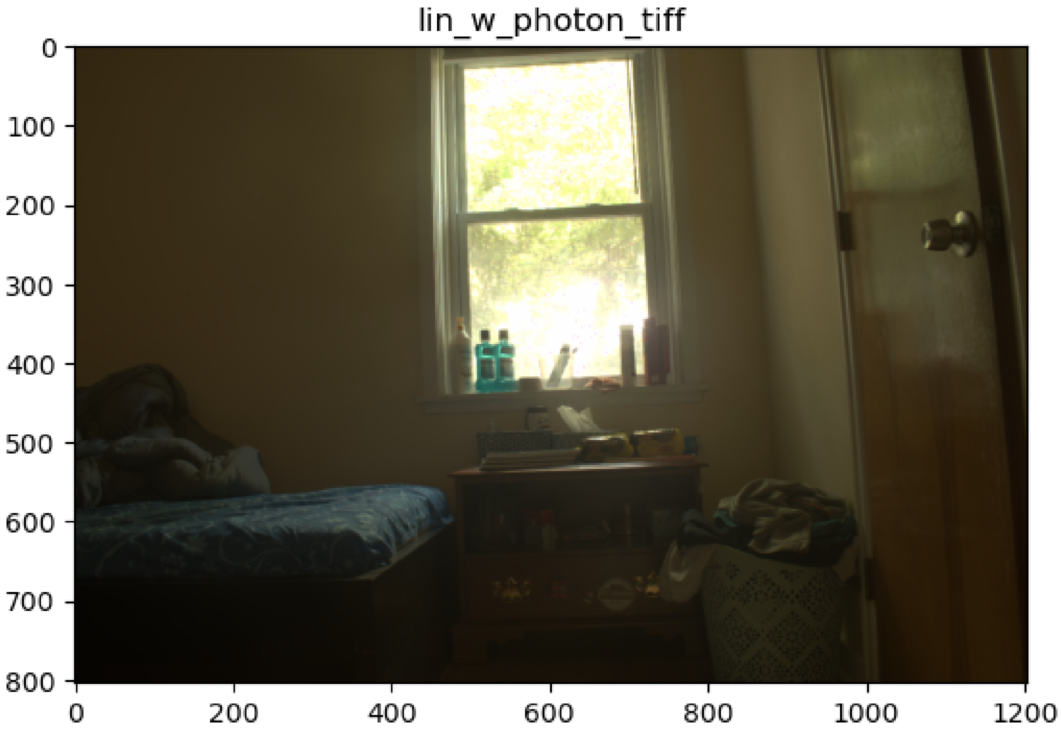
```
# # Creating all merged jpeg images

g_uniform = get_g(images_stack_for_lin_reshaped, images_stack_for_lin,␣
 ↪w_uniform_vectorized,t, 10)
g_tent = get_g(images_stack_for_lin_reshaped, images_stack_for_lin,␣
 ↪w_tent_vectorized,t, 10)
g_gaussian = get_g(images_stack_for_lin_reshaped, images_stack_for_lin,␣
 ↪w_gaussian_vectorized,t, 10)
g_photon = get_g(images_stack_for_lin_reshaped, images_stack_for_lin,␣
 ↪w_photon_vectorized,t, 10, is_w_photon=True)

lin_images_stack_uniform = linearize_image(images_stack, g_uniform)
lin_images_stack_tent = linearize_image(images_stack, g_tent)
lin_images_stack_gaussian = linearize_image(images_stack, g_gaussian)
lin_images_stack_photon = linearize_image(images_stack, g_photon)

I_hdr_jpeg_lin_merged_w_uniform = linear_merging(images_stack_normalized,␣
 ↪lin_images_stack_uniform, t, w_uniform_vectorized)
I_hdr_jpeg_log_merged_w_uniform = log_merging(images_stack_normalized,␣
 ↪lin_images_stack_uniform, t, w_uniform_vectorized)

I_hdr_jpeg_lin_merged_w_tent = linear_merging(images_stack_normalized,␣
 ↪lin_images_stack_tent, t, w_tent_vectorized)
I_hdr_jpeg_log_merged_w_tent = log_merging(images_stack_normalized,␣
 ↪lin_images_stack_tent, t, w_tent_vectorized)

I_hdr_jpeg_lin_merged_w_gaussian = linear_merging(images_stack_normalized,␣
 ↪lin_images_stack_gaussian, t, w_gaussian_vectorized)
I_hdr_jpeg_log_merged_w_gaussian = log_merging(images_stack_normalized,␣
 ↪lin_images_stack_gaussian, t, w_gaussian_vectorized)

I_hdr_jpeg_lin_merged_w_photon = linear_merging(images_stack_normalized,␣
 ↪lin_images_stack_photon, t, w_photon_vectorized, is_w_photon=True)
```

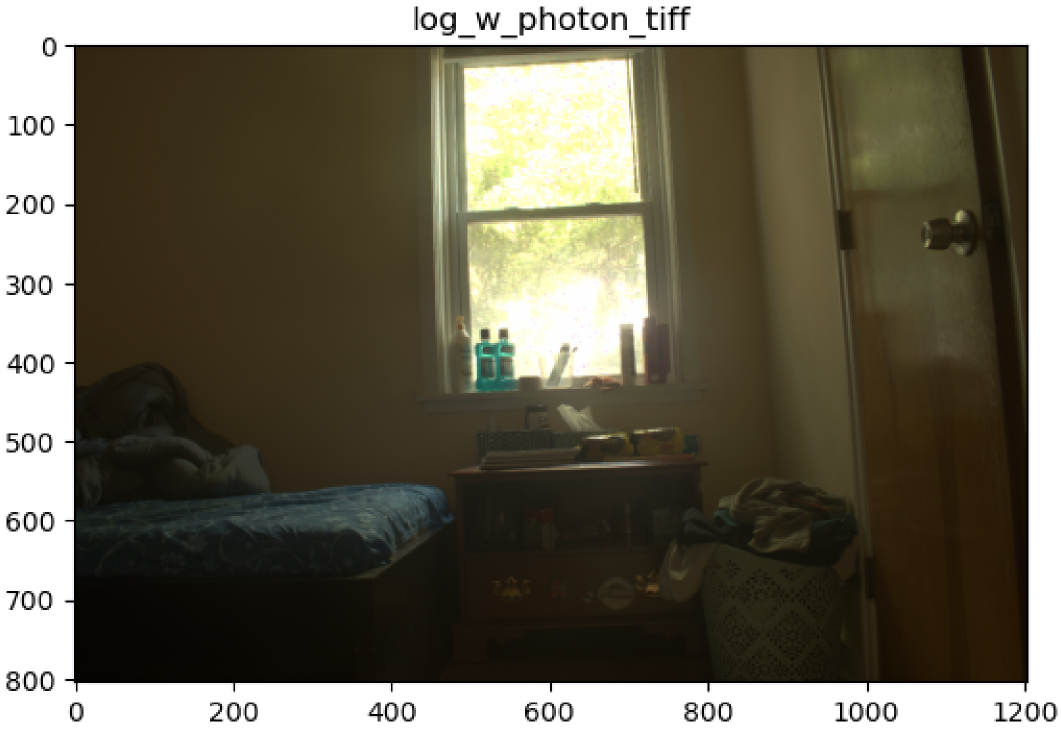
```
I_hdr_jpeg_log_merged_w_photon = log_merging(images_stack_normalized,␣
 ↪lin_images_stack_photon, t, w_photon_vectorized, is_w_photon=True)
```

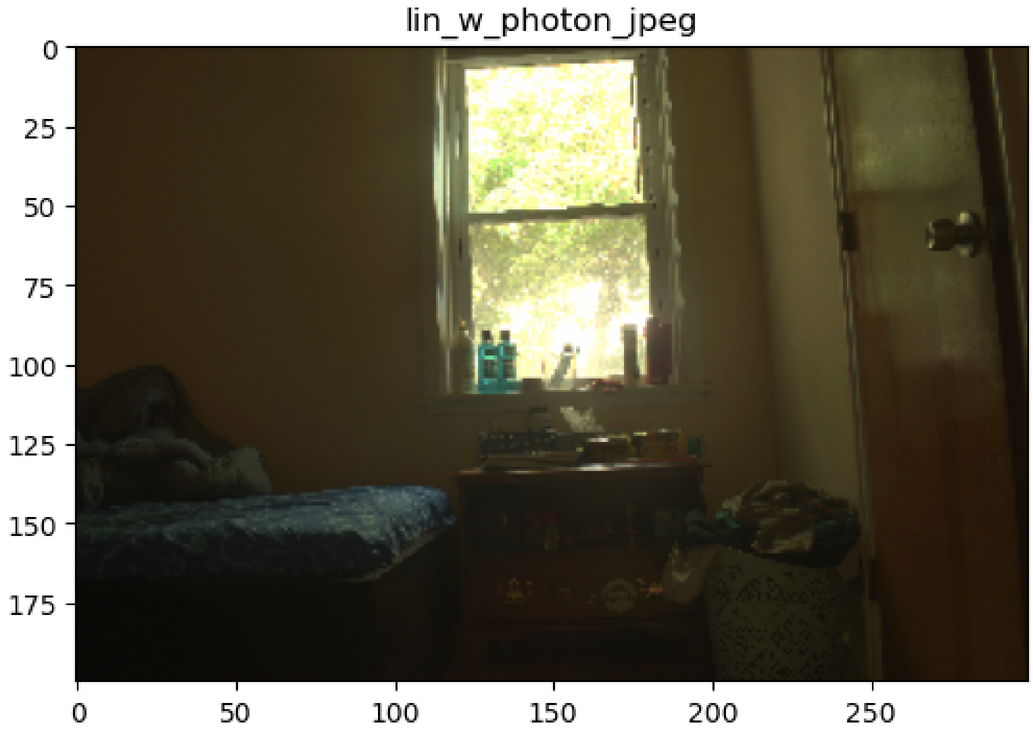
```
/tmp/ipykernel_5126/2109691339.py:23: RuntimeWarning: invalid value encountered
in divide
  I_hdr = I_hdr_num / I_hdr_den
/tmp/ipykernel_5126/2109691339.py:95: RuntimeWarning: invalid value encountered
in divide
  I_hdr = I_hdr_num / I_hdr_den
```

[31]:

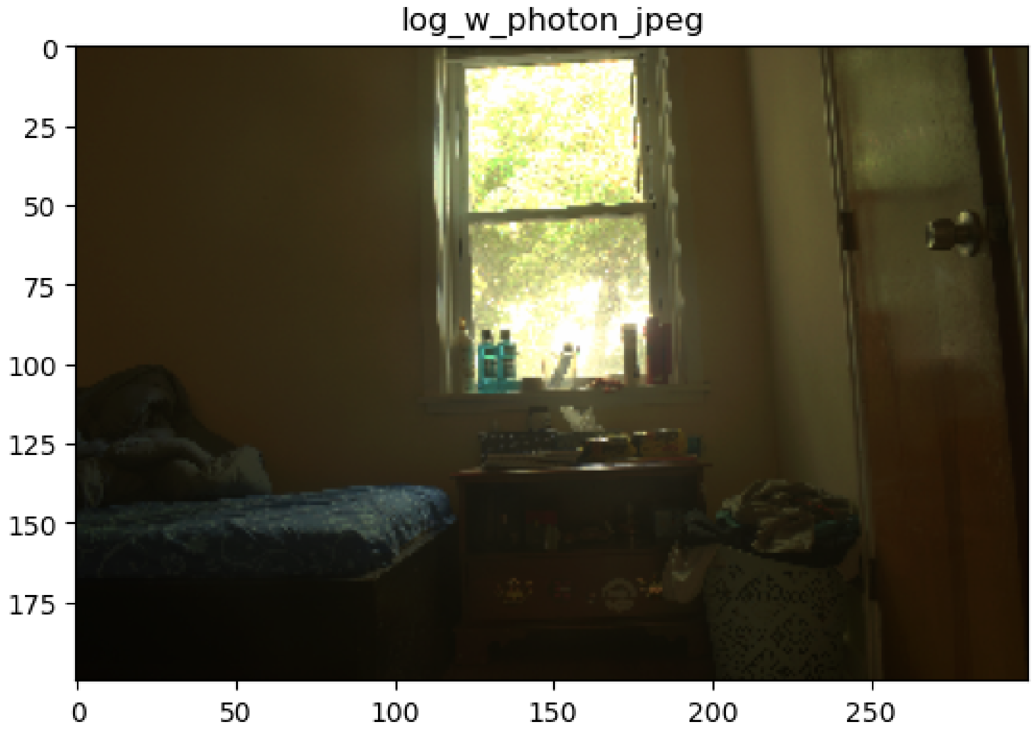
```
# Displaying images
# Displaying RAW source generated images
# _ = display_hdr_image(I_hdr_tiff_lin_merged_w_uniform,0.
 ↪00002,'lin_w_uni_tiff')
# _ = display_hdr_image(I_hdr_tiff_lin_merged_w_uniform_channelwise, 0.00002,␣
 ↪'channelwise up')
# _ = display_hdr_image(I_hdr_tiff_lin_merged_w_tent,0.00002,'lin_w_tent_tiff')
# _ = display_hdr_image(I_hdr_tiff_lin_merged_w_gaussian,0.
 ↪00002,'lin_w_gaussian_tiff')
_ = display_hdr_image(I_hdr_tiff_lin_merged_w_photon,0.
 ↪00002,'lin_w_photon_tiff')

# _ = display_hdr_image(I_hdr_tiff_log_merged_w_uniform,0.
 ↪00002,'log_w_uni_tiff')
# _ = display_hdr_image(I_hdr_tiff_log_merged_w_tent,0.00002,'log_w_tent_tiff')
# _ = display_hdr_image(I_hdr_tiff_log_merged_w_gaussian,0.
 ↪00002,'log_w_gaussian_tiff')
_ = display_hdr_image(I_hdr_tiff_log_merged_w_photon,0.
 ↪00002,'log_w_photon_tiff')

# Displaying JPEG source generated images

# _ = display_hdr_image(I_hdr_jpeg_lin_merged_w_uniform,0.1,'lin_w_uni_jpeg')
# _ = display_hdr_image(I_hdr_jpeg_lin_merged_w_tent,0.1,'lin_w_tent_jpeg')
# _ = display_hdr_image(I_hdr_jpeg_lin_merged_w_gaussian,0.
 ↪1,'lin_w_gaussian_jpeg')
_ = display_hdr_image(I_hdr_jpeg_lin_merged_w_photon,0.1,'lin_w_photon_jpeg')

# _ = display_hdr_image(I_hdr_jpeg_log_merged_w_uniform,0.1,'log_w_uni_jpeg')
# _ = display_hdr_image(I_hdr_jpeg_log_merged_w_tent,0.1,'log_w_tent_jpeg')
# _ = display_hdr_image(I_hdr_jpeg_log_merged_w_gaussian,0.
 ↪1,'log_w_gaussian_jpeg')
_ = display_hdr_image(I_hdr_jpeg_log_merged_w_photon,0.1,'log_w_photon_jpeg')
```


lin_w_photon_tiff


log_w_photon_tiff



lin_w_photon_jpeg



log_w_photon_jpeg

[33]:

```
# Selecting an image
# lin_w_photon_tiff and lin_w_photon_jpeg
src.cp_hw2.writeHDR('I_hdr_tiff_lin_merged_w_photon-part4b.HDR',␣
 ↪I_hdr_tiff_lin_merged_w_photon)
src.cp_hw2.writeHDR('I_hdr_jpeg_lin_merged_w_photon-part4b.HDR',␣
 ↪I_hdr_jpeg_lin_merged_w_photon)
im_hdr_selected = I_hdr_tiff_lin_merged_w_photon.copy()
```

[13]:

```

```

[35]:

```
# Photographic ToneMapping

# kinda sus, make sure its right
def calc_geometric_mean_log(im) :

    if im.min() < 0 :
        # clipping out negative values and raising them to be over 0
        g_mean = np.exp(np.log(im - (1.00000001) * im.min()).mean())
    else :
        # else
        g_mean = np.exp(np.log(im + 1e5).mean())

    return g_mean

def calc_geometric_mean(im) :
    pass

def place_key(g_mean, im, K) :
    im_keyed= K * (im) / g_mean
    return im_keyed

def set_white(im_keyed, B) :
    im_white = B * np.max(im_keyed)
    return im_white

# Note : don't really understand what the difference
# would be upon applying tonemapping for the 3 channels separately
# in the case mentioned in the hints. That would be same as this right?
def complete_photographic_tonemap(im, K, B) :

    g_mean = calc_geometric_mean_log(im)
    im_keyed = place_key(g_mean, im, K)
    im_white = set_white(im_keyed, B)

    im_tonemapped_num = im_keyed * (1 + im_keyed / (im_white**2))
    im_tonemapped_den = 1 + im_keyed

    im_tonemapped = im_tonemapped_num / im_tonemapped_den

    return im_tonemapped

def complete_photographic_tonemap_throughxyY(im, K, B) :

    im_XYZ = src.cp_hw2.lRGB2XYZ(im)
    x = im_XYZ[...,0].copy() / im_XYZ.sum(2)
    y = im_XYZ[...,1].copy() / im_XYZ.sum(2)
    Y = im_XYZ[...,1].copy()

    Y_tonemapped = complete_photographic_tonemap(Y, K, B)

    X = (x * Y_tonemapped) / y
    Y = Y_tonemapped
    Z = (1 - x - y) * Y_tonemapped / y

    im_XYZ_tonemapped = np.concatenate((np.stack((X,Y),-1),np.
 ↪expand_dims(Z,-1)),-1)
    im_RGB_tonemapped = src.cp_hw2.XYZ2lRGB(im_XYZ_tonemapped)

    return im_RGB_tonemapped

im_hdr_selected_cc_wb_ptm = complete_photographic_tonemap(im_hdr_selected, K=4,␣
 ↪B=0.95)
im_hdr_selected_cc_wb_ptm_tonemapped =␣
 ↪display_hdr_image(im_hdr_selected_cc_wb_ptm, 1, 'image after ptm tonemapped')
# src.cp_hw2.writeHDR('I_hdr_tiff_lin_merged_w_photon-part4b.HDR',␣
 ↪I_hdr_tiff_lin_merged_w_photon)


im_hdr_selected_cc_wb_ptm_xyY =␣
 ↪complete_photographic_tonemap_throughxyY(im_hdr_selected, K=1.5, B=0.95)
im_hdr_selected_cc_wb_ptm_tonemapped =␣
 ↪display_hdr_image(im_hdr_selected_cc_wb_ptm_xyY, 1.25, 'image after ptm␣
 ↪tonemapped-xyY')
src.cp_hw2.writeHDR('im_hdr_selected_cc_wb_ptm_xyY-part4b.HDR',␣
 ↪im_hdr_selected_cc_wb_ptm_xyY)
```

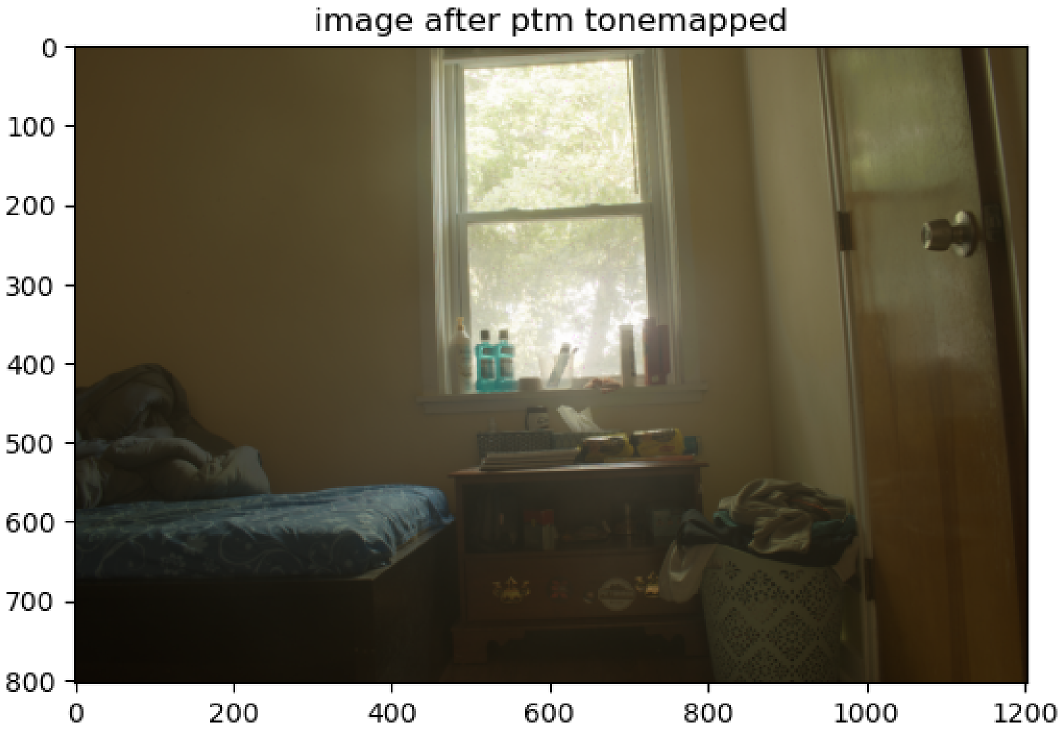


image after ptm tonemapped
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000
1200

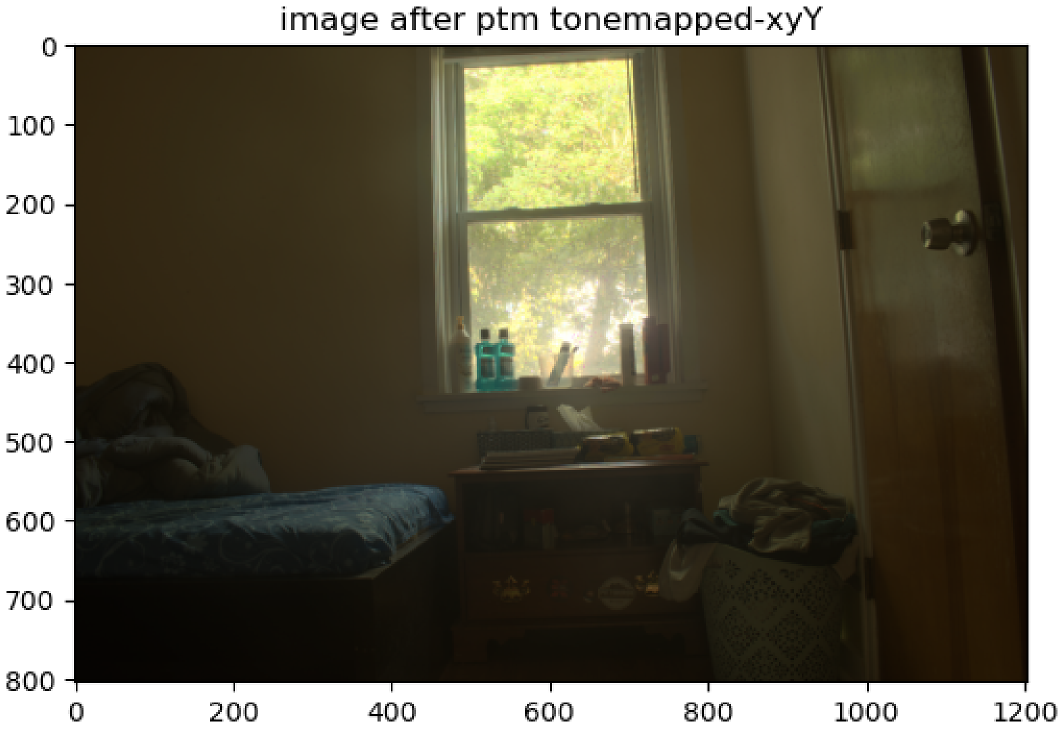


image after ptm tonemapped-xyY
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000
1200

0.1

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: