```
"""
@author: Zhen Tong
"""
import rawpy
import os
import sys
import time
import argparse
os.environ["OPENCV_IO_ENABLE_OPENEXR"]="1"
import cv2 #only for save images to jpg
import exifread
from matplotlib import pyplot as plt #use it only for debug
import numpy as np
from tqdm import tqdm

from BayerDomainProcessor import *
from RGBDomainProcessor import *
from YUVDomainProcessor import *

def get_arg():
    parser = argparse.ArgumentParser()
    parser.add_argument("--draw-intermediate", type=bool,
default=True)
    parser.add_argument("--set-dir", type=str,
default="../data/set03")
    parser.add_argument("--output-dir", type=str, default="data")
    parser.add_argument("--itmd_dir", type=str, default="itmd", help=\
        "intermediate file root directory.")
    parser.add_argument("--fast", type=bool, action="store_true",
help=\
        "User can choose open cv bilateral filter for larger and
faster convolution computation")
    args = parser.parse_args()
    assert os.path.exists(args.set_dir)
    if os.path.exists(args.output_dir) == False:
        os.makedirs(args.output_dir)

    args.itmd_dir = os.path.join(args.itmd_dir,
args.set_dir.strip("/")[-1])
    if os.path.exists(args.itmd_dir) == False:
        os.makedirs(args.itmd_dir)
    assert os.path.exists(args.output_dir)
    assert os.path.exists(args.itmd_dir)
    return args

# fake args
class ARGs:
    def __init__(self, fast=True):
        self.set_dir = "../data/set03"
        self.draw_intermediate = True
```

```
        self.output_dir = "data"
        self.itmd_dir = "itmd"
        self.itmd_dir = os.path.join(self.itmd_dir,
self.set_dir.strip("/")[-1])
        self.sub_dir = os.path.join(self.output_dir,
self.set_dir.split("/")[-1])
        self.fast = fast
        if os.path.exists(self.itmd_dir) == False:
            os.makedirs(self.itmd_dir)
        if os.path.exists(self.output_dir) == False:
            os.makedirs(self.output_dir)
        if os.path.exists(self.sub_dir) == False:
            os.makedirs(self.sub_dir)
        assert os.path.exists(self.output_dir)
        assert os.path.exists(self.itmd_dir)
args = ARGs()

image_name_list = []
for file in os.listdir(args.set_dir):
    if os.path.splitext(file)[1]=='.ARW':
        file_dir = os.path.join(args.set_dir, file)
        image_name_list.append(file_dir)

image_name_list.sort()

image_list = []
exposure_times = []
raw_list = []
for name in image_name_list:
    raw = rawpy.imread(name)
    raw_list.append(raw)
    image_list.append(raw.raw_image)
    raw_file = open(name, 'rb')

    exif_file = exifread.process_file(raw_file, details=False,
strict=True)

    ISO_str = exif_file['EXIF ISOSpeedRatings'].printable
    ExposureTime = exif_file['EXIF ExposureTime'].printable

    exposure_times.append(float(ExposureTime.split(
        '/')[0]) / float(ExposureTime.split('/')[1]) * float(ISO_str))

clip_list = []
for image in image_list:
    clip_list.append(np.clip(image, raw.black_level_per_channel[0],
2**14))
print(50*'-' + '\nLoading RAW Image Done......')
```

```
--------------------------------------------------
Loading RAW Image Done......

#########################################################################
########################################
####################################  Part 1: Bayer Domain Processing
Steps  ##################################
#########################################################################
########################################
# Step 1. Dead Pixel Correction (10pts)
Bayer_dpcs = []
dpc_thres, dpc_mode, dpc_clip  = 4096, 'gradient',
raw.camera_white_level_per_channel[0]
for i, raw_image in enumerate(clip_list):
    dpc = deadPixelCorrection(raw_image, dpc_thres, dpc_mode,
dpc_clip)
    dpc_output = dpc.execute()
    Bayer_dpcs.append(dpc_output)
    if args.draw_intermediate:
        output_dir = os.path.join(args.itmd_dir, 'Dead-Pixel-
Correction'+str(i)+'.jpg')
        cv2.imwrite(output_dir,dpc_output)
print(50*'-' + '\n 1.1 Dead Pixel Correction Done......')

--------------------------------------------------
 1.1 Dead Pixel Correction Done......

# Step 2.'Black Level Compensation' (5pts)
alpha, beta = 0, 0
parameter = raw.black_level_per_channel + [alpha, beta]
Bayer_blackLevelCompensations = []
for dpc_output in Bayer_dpcs:
    blkC = blackLevelCompensation(dpc_output, parameter,2**14, "rggb")
    blkc_output = blkC.execute()
    Bayer_blackLevelCompensations.append(blkc_output)
print(50*'-' + '\n 1.2 Black Level Compensation Done......')

--------------------------------------------------
 1.2 Black Level Compensation Done......

# Step 3.'lens shading correction
# skip this

# Step 4. Anti Aliasing Filter (10pts)
Bayer_antiAliasingFilters = []

for i, blkc_output in enumerate(Bayer_blackLevelCompensations):
    antiAliasingFilter = AntiAliasingFilter(blkc_output)
    aaf_output = antiAliasingFilter.execute()
    Bayer_antiAliasingFilters.append(aaf_output)
    if args.draw_intermediate:
```

```
        output_dir = os.path.join(args.itmd_dir, 'Bayer-
antiAliasingFilter'+str(i)+'.jpg')
        cv2.imwrite(output_dir, aaf_output)
print(50*'-' + '\n 1.4 Anti-aliasing Filtering Done......')
```

```
--------------------------------------------------
 1.4 Anti-aliasing Filtering Done......
```

```
# Step 5. Auto White Balance and Gain Control (10pts)
r_gain, gr_gain, gb_gain, b_gain = [2,1,2,1]
parameter = [r_gain, gr_gain, gb_gain, b_gain]
Bayer_awbs = []
for i, aff_output in enumerate(Bayer_antiAliasingFilters):
    raw = raw_list[i]
    awb_clip = raw.camera_white_level_per_channel[0]
    awb = AWB(aff_output, parameter, "rggb", awb_clip)
    awb_output = awb.execute()
    Bayer_awbs.append(awb_output)
    if args.draw_intermediate:
        output_dir = os.path.join(args.itmd_dir, "White-Balance-
Gain"+str(i)+".jpg")
        cv2.imwrite(output_dir,awb_output)
print(50*'-' + '\n 1.5 White Balance Gain Done......')
```

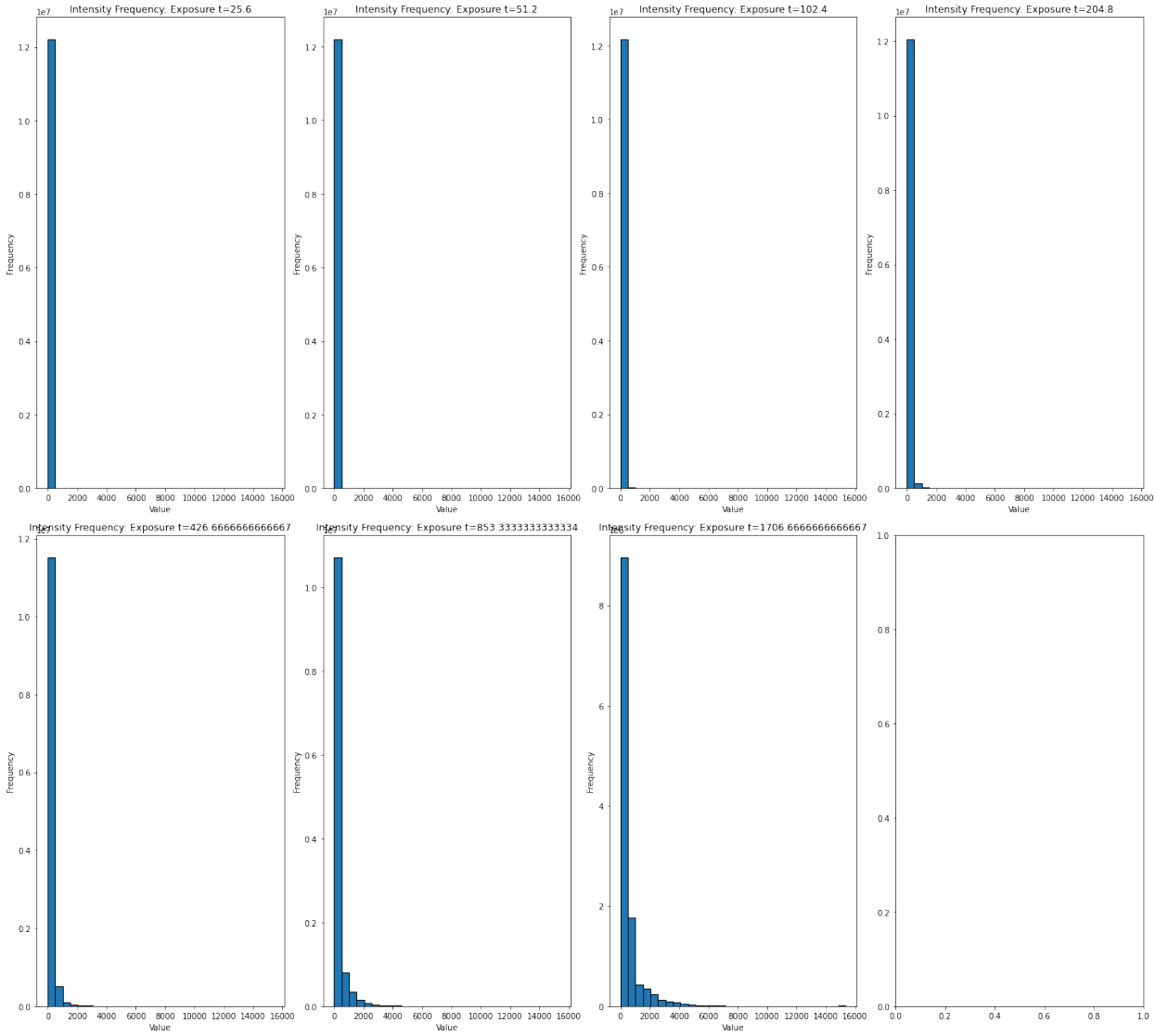
```
--------------------------------------------------
 1.5 White Balance Gain Done......
```

```
# Step 6. Raw Exposure Fusion
# your code here
def stat_draw_times(image_list:list, times:list):
    fig, axs = plt.subplots(2, 4, figsize=(20, 18))
    for i in range(len(times)):
        img = image_list[i]
        axs[i//4, i%4].hist(img[:, :].flatten(), bins=30,
edgecolor='black')
        axs[i//4, i%4].set_title('Intensity Frequency: Exposure
t='+str(times[i]))
        axs[i//4, i%4].set_xlabel('Value')
        axs[i//4, i%4].set_ylabel('Frequency')
    # Adjust layout to prevent overlapping
    plt.tight_layout()

    # Show the plot
    plt.savefig(os.path.join(args.itmd_dir, "Exporture-Intensity-
Frequency.jpg"))
    plt.show()

def stat_draw(img):
    plt.hist(img[:, :].flatten(), bins=30, edgecolor='black')
    plt.title('Intensity Frequency of Merge')
```

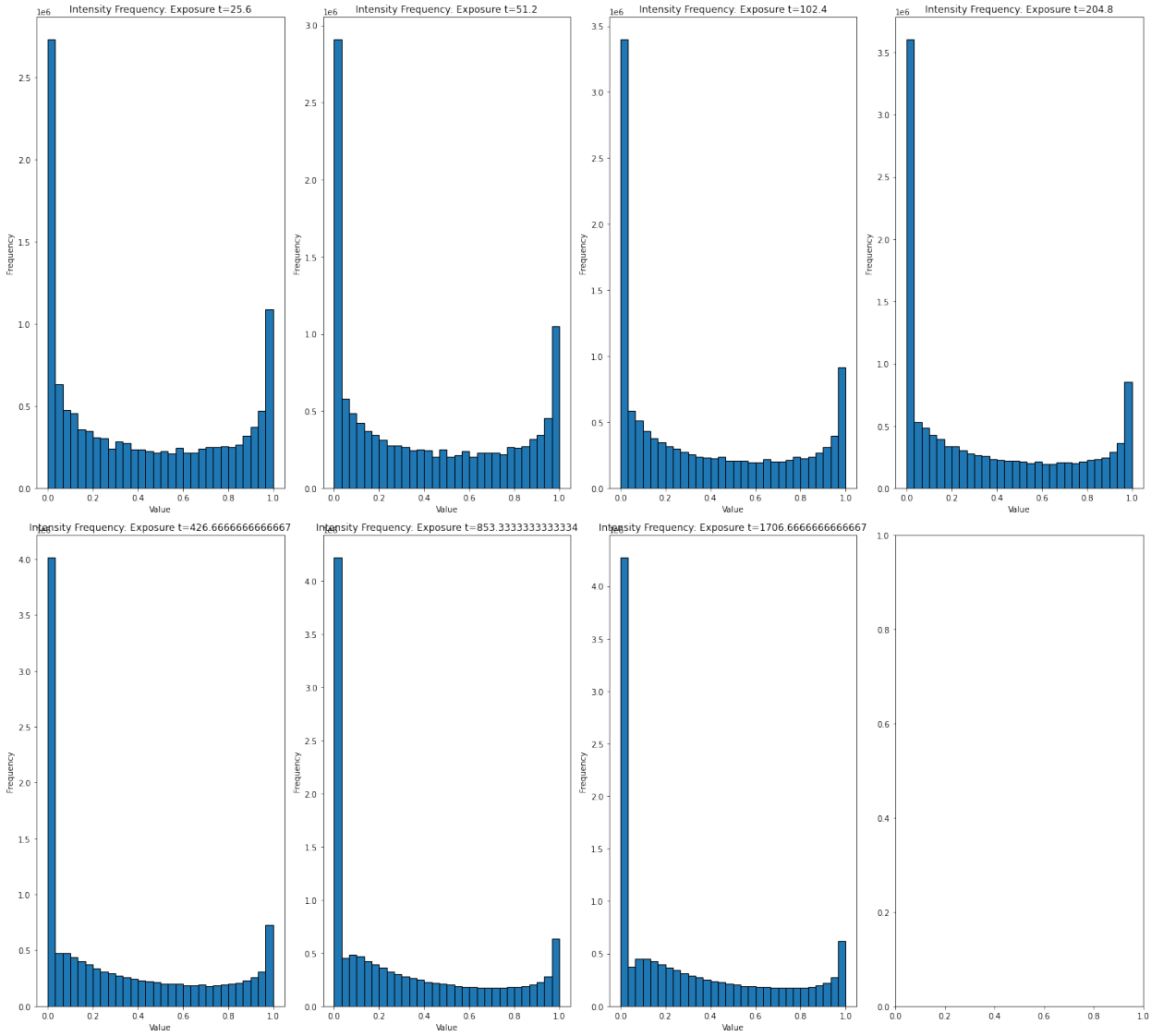
```
    plt.xlabel('Value')
    plt.ylabel('Frequency')
    plt.savefig(os.path.join(args.itmd_dir, "Intensity-Frequency-of-
Merge.jpg"))
    plt.show()

def stat_draw_rbg(img:np.ndarray, use_float = False):
    if (img.dtype != np.int) and (use_float == False):
        img = img.astype(np.int)
    fig, axs = plt.subplots(2, 2, figsize=(10, 8))

    # Plot histograms
    axs[0, 0].hist(img[:, :, 0].flatten(), bins=30, edgecolor='black',
color='r')
    axs[0, 0].set_title('Red Channel Histogram')
    axs[0, 0].set_xlabel('Value')
    axs[0, 0].set_ylabel('Frequency')

    axs[0, 1].hist(img[:, :, 1].flatten(), bins=30, edgecolor='black',
color='g')
    axs[0, 1].set_title('Green Channel Histogram')
    axs[0, 1].set_xlabel('Value')
    axs[0, 1].set_ylabel('Frequency')

    axs[1, 0].hist(img[:, :, 2].flatten(), bins=30, edgecolor='black',
color='b')
    axs[1, 0].set_title('Blue Channel Histogram')
    axs[1, 0].set_xlabel('Value')
    axs[1, 0].set_ylabel('Frequency')

    # Plot the image
    axs[1, 1].imshow(img)
    axs[1, 1].set_title('RGB Image')

    # Hide the x and y ticks for the image plot
    axs[1, 1].tick_params(axis='both', which='both', bottom=False,
left=False, labelbottom=False, labelleft=False)

    # Adjust layout to prevent overlapping
    plt.tight_layout()

    # Show the plot
    plt.show()

def stat_draw_yuv(img:np.ndarray, use_float = False):
    if (img.dtype != np.int) and (use_float == False):
        img = img.astype(int)
    fig, axs = plt.subplots(2, 2, figsize=(10, 8))

    # Plot histograms
```

```
    axs[0, 0].hist(img[:, :, 0].flatten(), bins=30, edgecolor='black',
color='r')
    axs[0, 0].set_title('Y Channel Histogram')
    axs[0, 0].set_xlabel('Value')
    axs[0, 0].set_ylabel('Frequency')

    axs[0, 1].hist(img[:, :, 1].flatten(), bins=30, edgecolor='black',
color='g')
    axs[0, 1].set_title('U Channel Histogram')
    axs[0, 1].set_xlabel('Value')
    axs[0, 1].set_ylabel('Frequency')

    axs[1, 0].hist(img[:, :, 2].flatten(), bins=30, edgecolor='black',
color='b')
    axs[1, 0].set_title('V Channel Histogram')
    axs[1, 0].set_xlabel('Value')
    axs[1, 0].set_ylabel('Frequency')

    # Plot the image
    axs[1, 1].imshow(img)
    axs[1, 1].set_title('YUV Image')

    # Hide the x and y ticks for the image plot
    axs[1, 1].tick_params(axis='both', which='both', bottom=False,
left=False, labelbottom=False, labelleft=False)

    # Adjust layout to prevent overlapping
    plt.tight_layout()

    # Show the plot
    plt.show()

def normalize(img):
    img_min = np.min(img)
    img_max = np.max(img)
    return (img-img_min)/(img_max-img_min)

def normalize_rgb(img, bias = True):
    r_min = np.min(img[:, :, 0])
    g_min = np.min(img[:, :, 1])
    b_min = np.min(img[:, :, 2])
    r_max = np.max(img[:, :, 0])
    g_max = np.max(img[:, :, 1])
    b_max = np.max(img[:, :, 2])

    if bias:
        img[:, :, 0] = (img[:, :, 0]-r_min)/(r_max-r_min)
        img[:, :, 1] = (img[:, :, 1]-g_min)/(g_max-g_min)
        img[:, :, 2] = (img[:, :, 2]-b_min)/(b_max-b_min)
    else:
```

```
        img[:, :, 0] = (img[:, :, 0])/(r_max)
        img[:, :, 1] = (img[:, :, 1])/(g_max)
        img[:, :, 2] = (img[:, :, 2])/(b_max)
    return img

def get_fusion_weights(images:list, median = False):
    w_list = []
    for i, img in tqdm(enumerate(images)):
        w = np.zeros_like(img)
        if median:
            median = np.median(img)
            w = np.exp(-4 * ((img-median)**2/median**2),
dtype=np.float64)
        else:
            w = np.exp(-4 * ((img-0.5)**2/0.5**2), dtype=np.float64)
        w_list.append(w)
    return w_list

# norm_awbs = [normalize(awb) for awb in Bayer_awbs]
stat_draw_times(Bayer_awbs, exposure_times)
weights = get_fusion_weights(Bayer_awbs, median=True)
if args.draw_intermediate:
    for i, w in enumerate(weights):
        output_dir = os.path.join(args.itmd_dir, "fusion-
weight-"+str(i)+".jpg")
        plt.imshow(w)
        plt.title("fusion-weight-"+str(i))
        plt.axis("off")
        plt.savefig(output_dir)
        plt.clf()
```



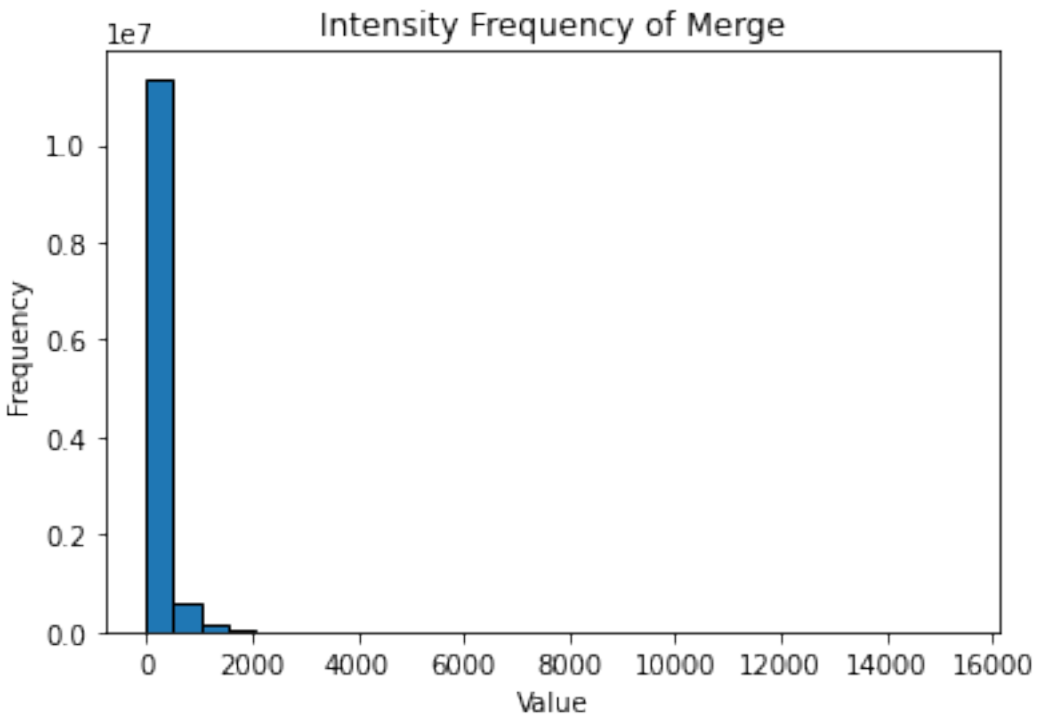

```
7it [00:07,  1.09s/it]
<Figure size 432x288 with 0 Axes>
stat_draw_times(weights, exposure_times)
```




```
def compute_mask(matrix_list, min, max):
    masks = []
    for i, matrix in tqdm(enumerate(matrix_list)):
        mask = (matrix>min) & (matrix<max)
        masks.append(mask)
    return masks

def raw_exposure_fusion(images, exposure_times, weights = None, \
    mask_range = [0.01, 1], mask_percent=0.03):

    if weights == None:
        weights = get_fusion_weights(images)

    img_masks = compute_mask(images, mask_range[0], mask_range[1])
    w_masks = compute_mask(weights, mask_range[0], mask_range[1])
    a = np.zeros_like(images[0], dtype=np.float64)
```

```
    b = np.zeros_like(images[0], dtype=np.float64)
    c = np.mean(np.array(images), axis=0)
    for i, w in tqdm(enumerate(weights)):
        img_i = images[i]
        t_i = exposure_times[i]
        # i_m = img_masks[i]
        w_m = w_masks[i]
        # a[w_m] += w[w_m] * (np.log(img_i[w_m], dtype=np.float64)-
np.log(t_i, dtype=np.float64))
        # b[w_m] += w[w_m]
        i_m = img_masks[i]
        m = i_m & w_m
        a[m] += w[m] * (np.log(img_i[m], dtype=np.float64)-np.log(t_i,
dtype=np.float64))
        b[m] += w[m]

    b_mask = b>0
    c[b_mask] = np.exp(a[b_mask]/b[b_mask])
    return  c.astype(np.float32)


Fusion = raw_exposure_fusion(Bayer_awbs, exposure_times, weights)
# Fusion = normalize(Fusion)
if args.draw_intermediate:
    plt.imshow(Fusion)
    plt.title("Merging LDR Exposures")
    plt.axis("off")
    plt.savefig(os.path.join(args.itmd_dir, "Merging-LDR-
Exposures.jpg"))
    plt.clf()
    stat_draw(Fusion)
```

```
0it [00:00, ?it/s]

7it [00:00, 12.85it/s]
7it [00:00, 20.72it/s]
7it [00:00,  9.65it/s]
```




```
np.max(Fusion)
```

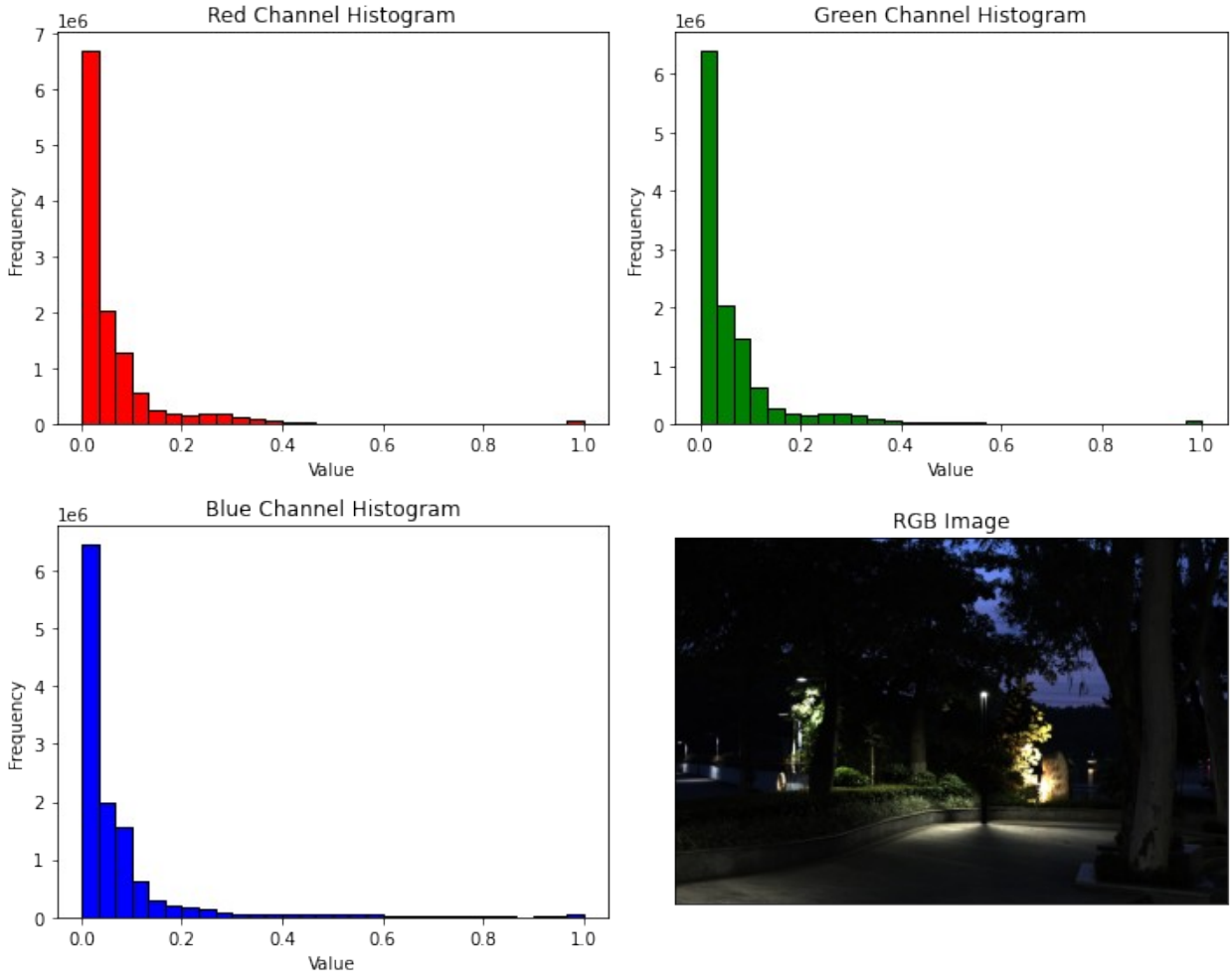
```
15360.0
```

```
class CFA_beta:
    def __init__(self, img, mode, bayer_pattern, clip):
        self.img = img
        self.mode = mode
        self.bayer_pattern = bayer_pattern
        self.clip = clip

    def padding(self):
        # Fill your code here
        img_pad = np.pad(self.img, ((2,2),(2,2)), 'reflect')

        return img_pad

    def clipping(self):
        # Fill your code here

        return


    def execute(self):
        img_pad = self.padding()
        img_pad = img_pad.astype(np.float32)
        raw_h = self.img.shape[0]
        raw_w = self.img.shape[1]
```

```
        cfa_img = np.empty((raw_h, raw_w, 3), np.float32)
        r = np.zeros((self.img.shape))
        g = np.zeros((self.img.shape))
        b = np.zeros((self.img.shape))


        G_at_R = G_at_B = np.array([[0,0,-1,0,0],[0,0,2,0,0],[-
1,2,4,2,-1],[0,0,2,0,0],[0,0,-1,0,0]])
        R_row_at_G = B_row_at_G = np.array([[0,0,1/2,0,0],[0,-1,0,-
1,0],[-1,4,5,4,-1],[0,-1,0,-1,0],[0,0,1/2,0,0]])
        R_col_at_G = B_col_at_G = R_row_at_G.T
        R_at_B = B_at_R = np.array([[0,0,-3/2,0,0],[0,2,0,2,0],[-
3/2,0,6,0,-3/2],[0,2,0,2,0],[0,0,-3/2,0,0]])

        r[::2,::2] = img_pad[2:-2:2,2:-2:2]
        for i in range(-2,3):
            for j in range(-2,3):
                if R_at_B[i+2,j+2] != 0:
                    r[1::2,1::2] +=
(img_pad[3+i:,3+j:]*R_at_B[i+2,j+2])[:raw_h:2,:raw_w:2]
                if R_col_at_G[i+2,j+2] != 0:
                    r[1::2,::2] +=
(img_pad[3+i:,2+j:]*R_col_at_G[i+2,j+2])[:raw_h:2,:raw_w:2]
                if R_row_at_G[i+2,j+2] != 0:
                    r[::2,1::2] +=
(img_pad[2+i:,3+j:]*R_row_at_G[i+2,j+2])[:raw_h:2,:raw_w:2]
        r[1::2,::2] /= 8
        r[::2,1::2] /= 8
        r[1::2,1::2] /= 8

        g[1::2,::2] = img_pad[3:-1:2,2:-2:2]
        g[::2,1::2] = img_pad[2:-2:2,3:-1:2]
        for i in range(-2,3):
            for j in range(-2,3):
                if G_at_B[i+2,j+2] != 0:
                    g[1::2,1::2] +=
(img_pad[3+i:,3+j:]*G_at_B[i+2,j+2])[:raw_h:2,:raw_w:2]
                if G_at_R[i+2,j+2] != 0:
                    g[::2,::2] += (img_pad[2+i:,2+j:]*G_at_R[i+2,j+2])
[:raw_h:2,:raw_w:2]
        g[::2,::2] /= 8
        g[1::2,1::2] /= 8

        b[1::2,1::2] = img_pad[3:-1:2,3:-1:2]
        for i in range(-2,3):
            for j in range(-2,3):
                if B_at_R[i+2,j+2] != 0:
                    b[::2,::2] += (img_pad[2+i:,2+j:]*B_at_R[i+2,j+2])
[:raw_h:2,:raw_w:2]
                if B_row_at_G[i+2,j+2] != 0:
```

```
                    b[1::2,::2] +=
(img_pad[3+i:,2+j:]*B_row_at_G[i+2,j+2])[:raw_h:2,:raw_w:2]
                if B_col_at_G[i+2,j+2] != 0:
                    b[::2,1::2] +=
(img_pad[2+i:,3+j:]*B_col_at_G[i+2,j+2])[:raw_h:2,:raw_w:2]
        b[::2,::2] /= 8
        b[1::2,::2] /= 8
        b[::2,1::2] /= 8

        # Fill your code here
        cfa_img = np.stack([r,g,b],axis=2)
        cfa_img = np.clip(cfa_img, 0, 2**14)
        sorted = np.sort(cfa_img.flatten())
        max = np.percentile(sorted,99.5)
        tone_mapping_factor = 0
        percentage = 0.02
        min = np.percentile(sorted,0.05)
        step_size = 0.01
        while min == 0.0:
            if percentage+step_size>=1:
                break
            percentage+=step_size
            min = np.percentile(sorted,percentage)
            print("percentage = ", percentage)

        tone_mapping_factor = max/min

        cfa_img_norm = (np.clip((cfa_img-min)/ (max-min),1E-8,1))
        return cfa_img, cfa_img_norm, tone_mapping_factor

# Step 7. 'Color Filter Array Interpolation'  Malvar (20pts)
cfa_mode = 'malvar'
cfa_clip = raw_list[0].camera_white_level_per_channel[0]
cfa = CFA_beta(Fusion, cfa_mode, "rggb", cfa_clip)
rgbimg_cfa, rgbimg_cfa_norm, tone_mapping_factor = cfa.execute()
stat_draw_rbg(rgbimg_cfa_norm, use_float = True)
print("tone_mapping_factor = ", tone_mapping_factor)
print(50*'-' + '\n 1.7 Demosaicing Done......')
if args.draw_intermediate:
    output_jpg_dir = os.path.join(args.itmd_dir,
'CFA_Interpolation.jpg')
    output_hdr_dir = os.path.join(args.itmd_dir,
'CFA_Interpolation.hdr')
    plt.imshow(rgbimg_cfa_norm)
    plt.title("CFA Interpolation after Merge")
    plt.axis("off")
    plt.savefig(output_jpg_dir)

cv2.imwrite(output_hdr_dir,cv2.cvtColor(rgbimg_cfa_norm.astype(np.floa
t32),cv2.COLOR_RGB2BGR))
```

```
percentage =  0.03
percentage =  0.04
percentage =  0.05
percentage =  0.060000000000000005
percentage =  0.07
percentage =  0.08
percentage =  0.09
percentage =  0.09999999999999999
percentage =  0.10999999999999999
percentage =  0.11999999999999998
percentage =  0.12999999999999998
percentage =  0.13999999999999999
percentage =  0.15
percentage =  0.16
percentage =  0.17
percentage =  0.18000000000000002
percentage =  0.19000000000000003
percentage =  0.20000000000000004
percentage =  0.21000000000000005
percentage =  0.22000000000000006
percentage =  0.23000000000000007
percentage =  0.24000000000000007
percentage =  0.25000000000000006
percentage =  0.26000000000000006
percentage =  0.2700000000000001
percentage =  0.2800000000000001
percentage =  0.2900000000000001
percentage =  0.3000000000000001
percentage =  0.3100000000000001
percentage =  0.3200000000000001
percentage =  0.3300000000000001
percentage =  0.34000000000000014
percentage =  0.35000000000000014
percentage =  0.36000000000000015
percentage =  0.37000000000000016
percentage =  0.38000000000000017

C:\Users\surface\AppData\Local\Temp\ipykernel_15684\2878482894.py:27:
DeprecationWarning: `np.int` is a deprecated alias for the builtin
`int`. To silence this warning, use `int` by itself. Doing this will
not modify any behavior and is safe. When replacing `np.int`, you may
wish to use e.g. `np.int64` or `np.int32` to specify the precision. If
you wish to review your current use, check the release note link for
additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if (img.dtype != np.int) and (use_float == False):
```




```
tone_mapping_factor =  3258556.027478817
----------------------------------------------------
 1.7 Demosaicing Done......
```

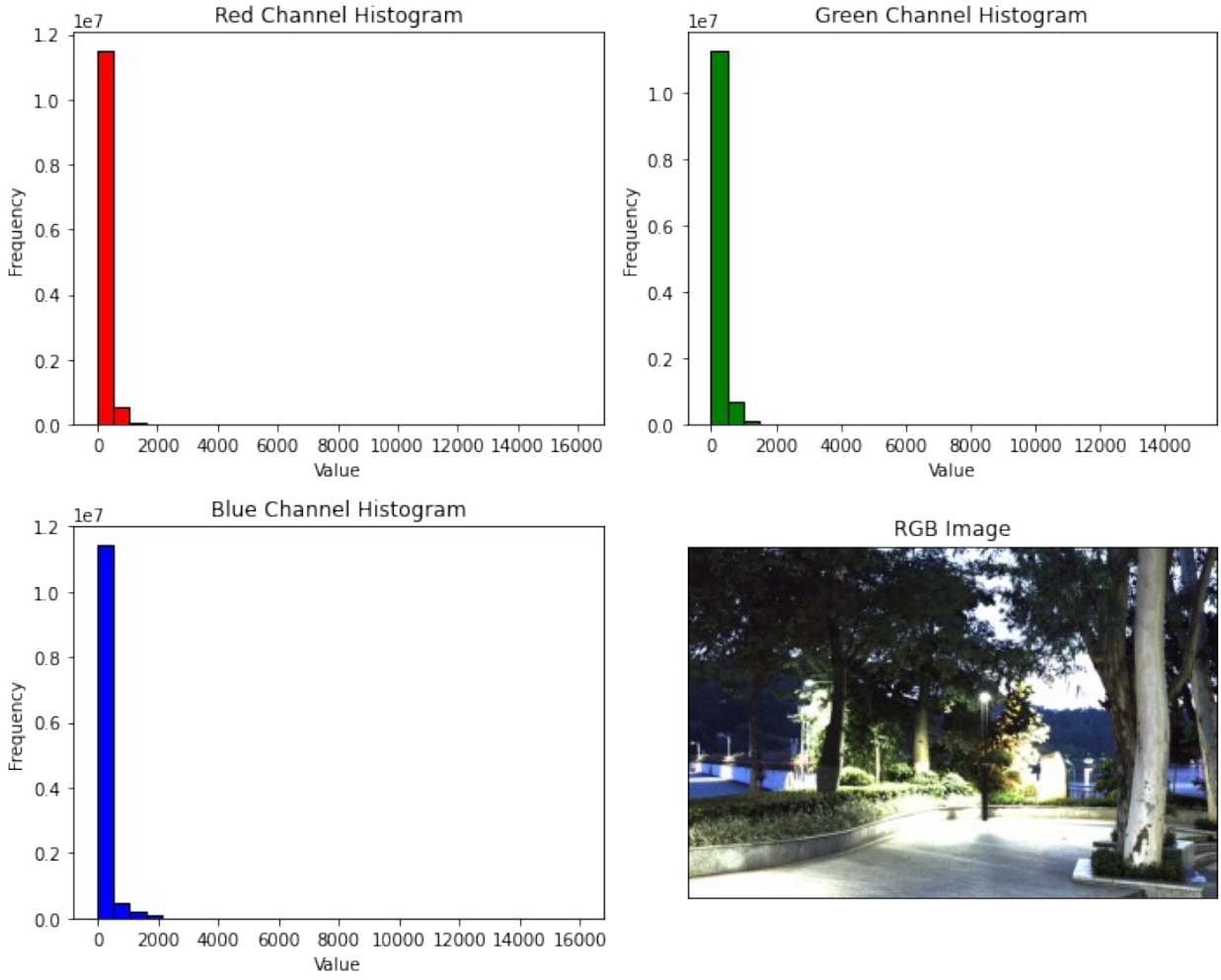

```
if args.set_dir.split("/")[-1] == "set03":
    show_img_list = [
        cv2.imread(os.path.join(args.set_dir, "DSC00163.jpg")),
        cv2.imread(os.path.join(args.set_dir, "DSC00164.jpg")),
        cv2.imread(os.path.join(args.set_dir, "DSC00165.jpg")),
        cv2.imread(os.path.join(args.set_dir, "DSC00166.jpg")),
        cv2.imread(os.path.join(args.set_dir, "DSC00167.jpg")),
        cv2.imread(os.path.join(args.set_dir, "DSC00168.jpg")),
        cv2.imread(os.path.join(args.set_dir, "DSC00169.jpg")),
        rgbimg_cfa_norm
    ]
    show_img_list[:-1] = [cv2.cvtColor(img, cv2.COLOR_BGR2RGB) for img
in show_img_list[:-1]]
    fig, axs = plt.subplots(4, 2, figsize=(20, 18))
    for i, img in enumerate(show_img_list[:-1]):
        time_i = exposure_times[i]
        axs[i//2, i%2].imshow(img)
        axs[i//2, i%2].set_title('Exposure Time:'+str(time_i))
        axs[i//2, i%2].axis("off")
    axs[7//2, 7%2].imshow(rgbimg_cfa_norm)
    axs[7//2, 7%2].set_title('Fusion')
    axs[7//2, 7%2].axis("off")
# Adjust layout to prevent overlapping
plt.tight_layout()
# Show the plot
plt.savefig(os.path.join(args.itmd_dir, ".jpg"))
plt.show()
```


```
import OpenEXR
import Imath

def save_hdr_image_to_exr(rgb_matrix, file_path):
    # Convert the RGB matrix to a 32-bit float array
```

```
    hdr_data = np.array(rgb_matrix, dtype=np.float32)

    # Create an OpenEXR image header with the dimensions of the matrix
    header = OpenEXR.Header(hdr_data.shape[1], hdr_data.shape[0])

    # Set the channels in the header
    header['channels'] = {
        'R': Imath.Channel(Imath.PixelType(Imath.PixelType.FLOAT)),
        'G': Imath.Channel(Imath.PixelType(Imath.PixelType.FLOAT)),
        'B': Imath.Channel(Imath.PixelType(Imath.PixelType.FLOAT))
    }

    # Create an OpenEXR image object
    exr_image = OpenEXR.OutputFile(file_path, header)

    # Convert the RGB matrix to a planar configuration (separate R, G,
B channels)
    r_channel = hdr_data[:, :, 0].tobytes()
    g_channel = hdr_data[:, :, 1].tobytes()
    b_channel = hdr_data[:, :, 2].tobytes()

    # Write the channel data to the EXR image
    exr_image.writePixels({'R': r_channel, 'G': g_channel, 'B':
b_channel})

    # Close the EXR image file
    exr_image.close()

file_path = 'output_image.exr'  # Replace with the desired output file
path

save_hdr_image_to_exr(rgbimg_cfa, file_path)

stat_draw_rbg(rgbimg_cfa)
```

```
C:\Users\surface\AppData\Local\Temp\ipykernel_15684\2878482894.py:27:
DeprecationWarning: `np.int` is a deprecated alias for the builtin
`int`. To silence this warning, use `int` by itself. Doing this will
not modify any behavior and is safe. When replacing `np.int`, you may
wish to use e.g. `np.int64` or `np.int32` to specify the precision. If
you wish to review your current use, check the release note link for
additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if (img.dtype != np.int) and (use_float == False):
C:\Users\surface\AppData\Local\Temp\ipykernel_15684\2878482894.py:28:
DeprecationWarning: `np.int` is a deprecated alias for the builtin
`int`. To silence this warning, use `int` by itself. Doing this will
not modify any behavior and is safe. When replacing `np.int`, you may
wish to use e.g. `np.int64` or `np.int32` to specify the precision. If
```

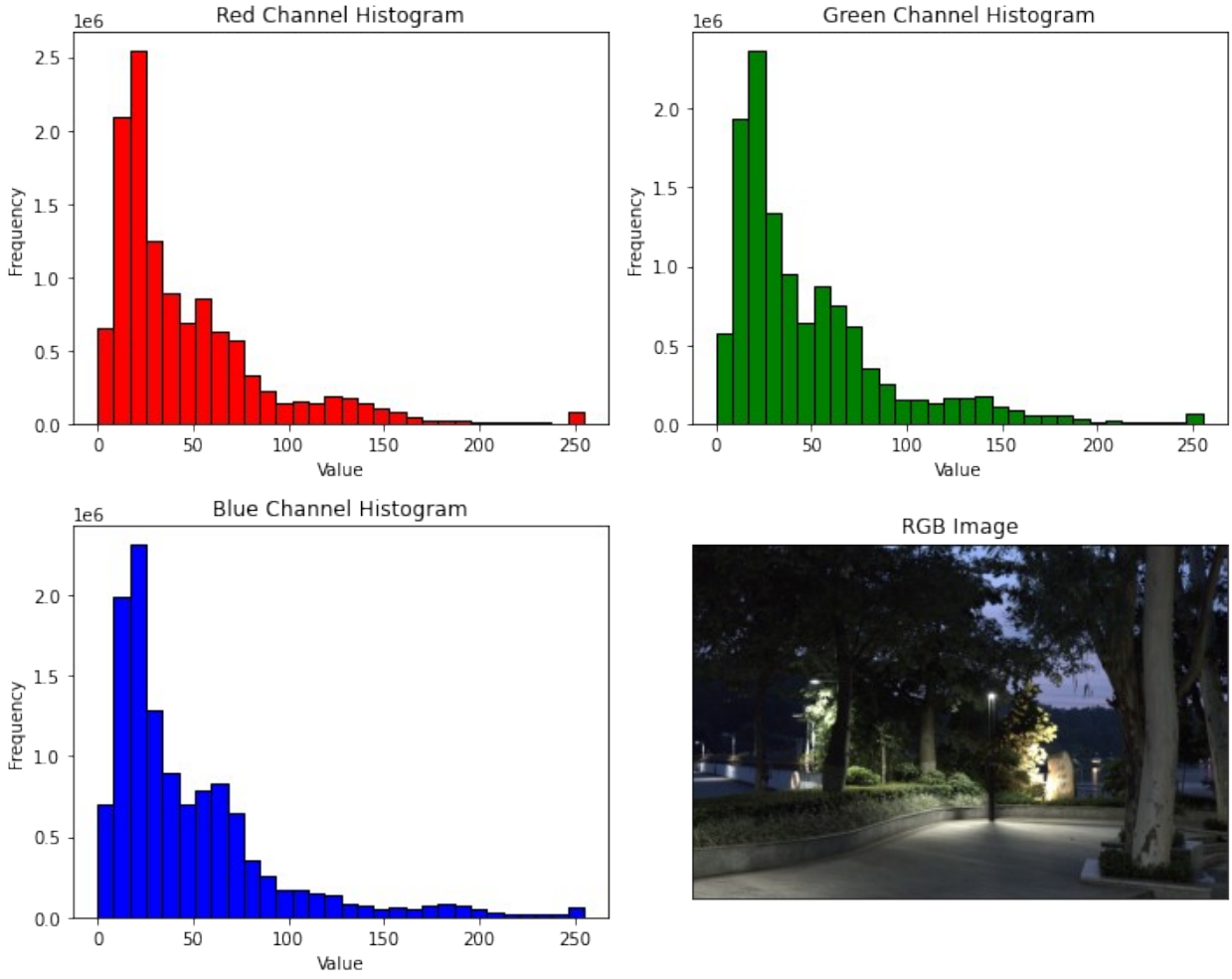
```
you wish to review your current use, check the release note link for
additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  img = img.astype(np.int)
Clipping input data to the valid range for imshow with RGB data
([0..1] for floats or [0..255] for integers).
```




```
#####################################  Bayer Domain Processing end
##########################################
#########################################################################
##########################################
gamma = GammaCorrection(rgbimg_cfa_norm,None,'gamma')
gamma_corrected = gamma.execute()
if args.draw_intermediate:
    output_dir = os.path.join(args.itmd_dir,
"GammaCorrectionWithoutToneMapping.jpg")
    stat_draw_rbg(gamma_corrected)
    plt.imshow(gamma_corrected.astype(np.int))
```

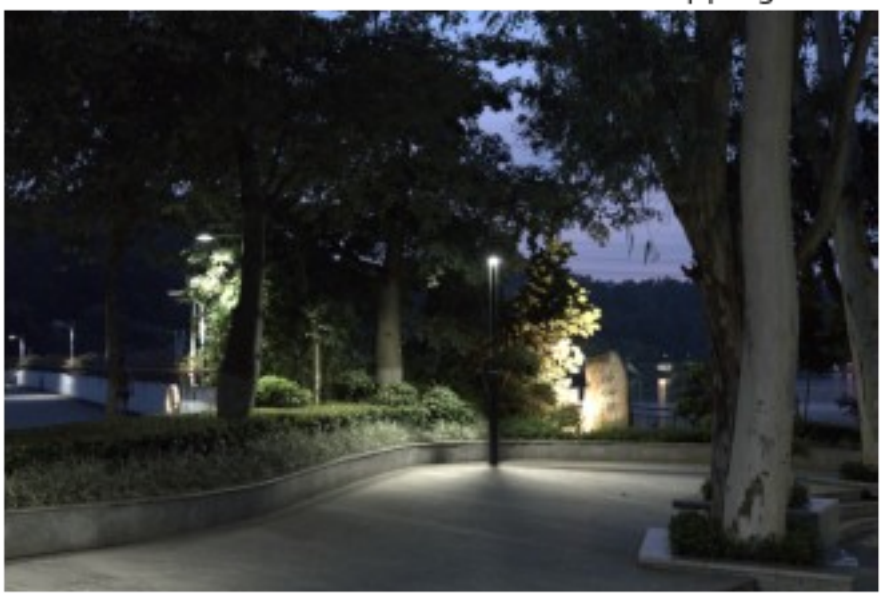
```
    plt.title("Gamma Correction Without Tone Mapping")
    plt.axis("off")
    plt.savefig(output_dir)
```

```
C:\Users\surface\AppData\Local\Temp\ipykernel_15684\2878482894.py:27:
DeprecationWarning: `np.int` is a deprecated alias for the builtin
`int`. To silence this warning, use `int` by itself. Doing this will
not modify any behavior and is safe. When replacing `np.int`, you may
wish to use e.g. `np.int64` or `np.int32` to specify the precision. If
you wish to review your current use, check the release note link for
additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if (img.dtype != np.int) and (use_float == False):
C:\Users\surface\AppData\Local\Temp\ipykernel_15684\2878482894.py:28:
DeprecationWarning: `np.int` is a deprecated alias for the builtin
`int`. To silence this warning, use `int` by itself. Doing this will
not modify any behavior and is safe. When replacing `np.int`, you may
wish to use e.g. `np.int64` or `np.int32` to specify the precision. If
you wish to review your current use, check the release note link for
additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  img = img.astype(np.int)
```



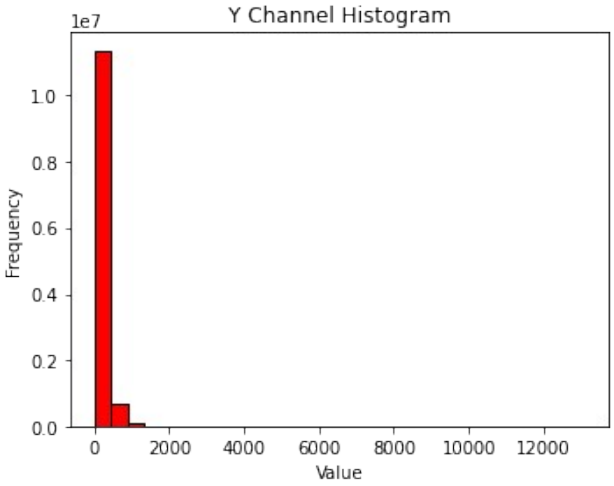

```
C:\Users\surface\AppData\Local\Temp\ipykernel_15684\2046979550.py:8:
DeprecationWarning: `np.int` is a deprecated alias for the builtin
`int`. To silence this warning, use `int` by itself. Doing this will
not modify any behavior and is safe. When replacing `np.int`, you may
wish to use e.g. `np.int64` or `np.int32` to specify the precision. If
you wish to review your current use, check the release note link for
additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  plt.imshow(gamma_corrected.astype(np.int))
```

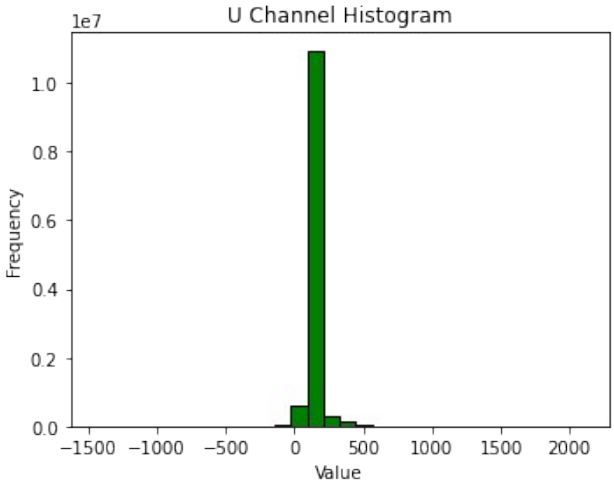
Gamma Correction Without Tone Mapping

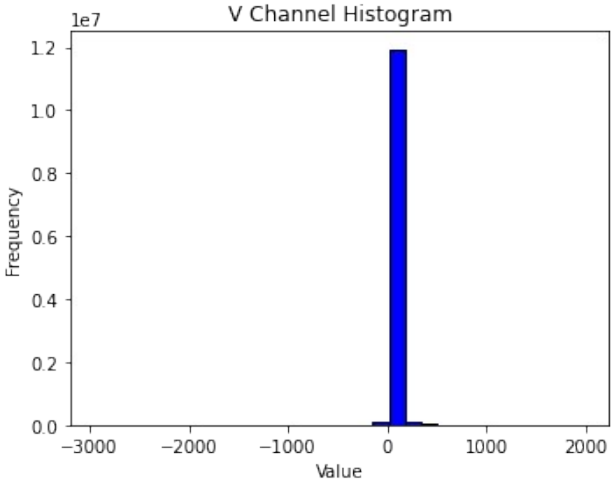


```
YUV = RGB2YUV(rgbimg_cfa)
YUV_norm = RGB2YUV(rgbimg_cfa_norm*255)
stat_draw_yuv(YUV)
stat_draw_yuv(YUV_norm)
```

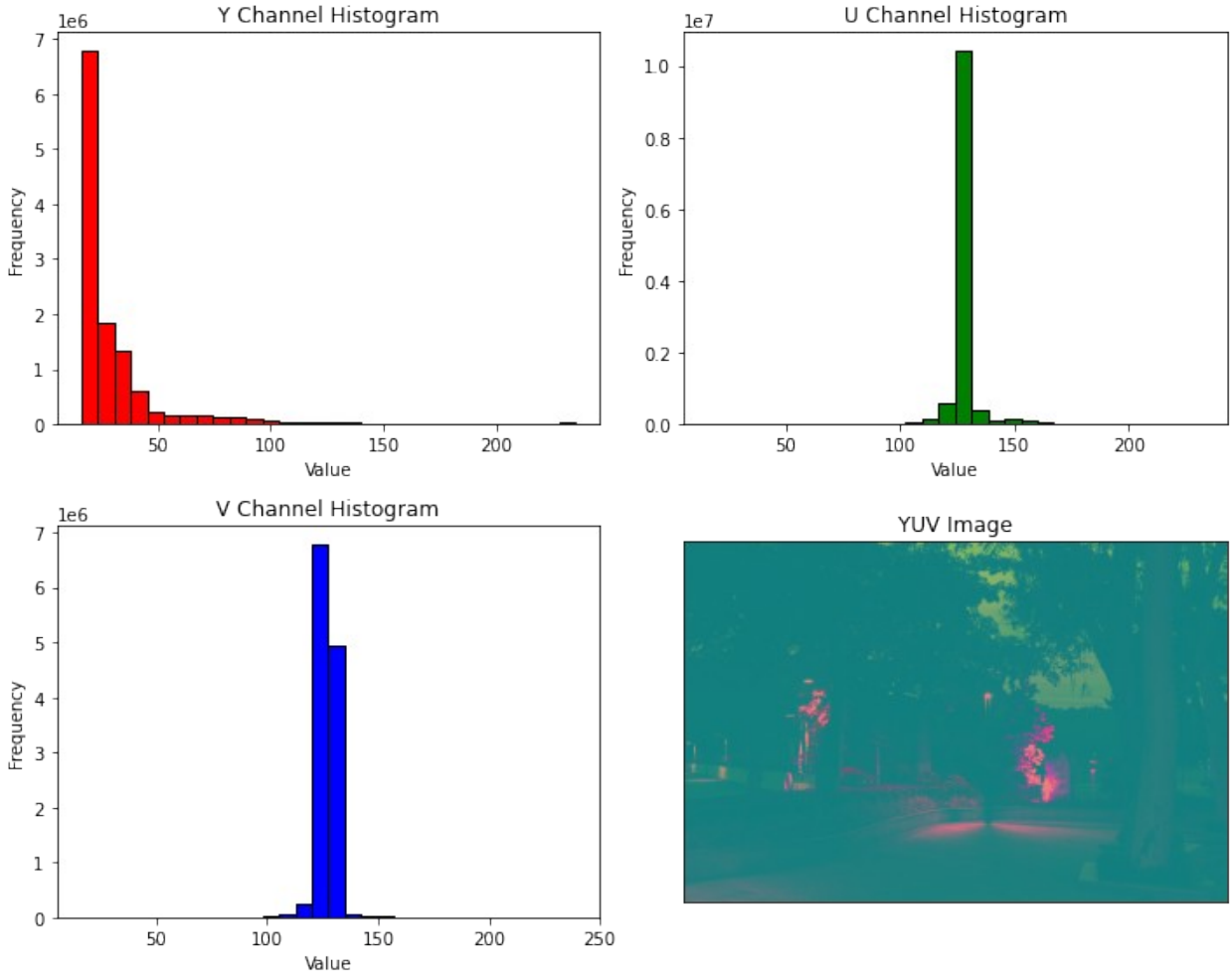
```
d:\2024_Spring\DDA4310\TaCode\RGBDomainProcessor.py:67:
VisibleDeprecationWarning: Creating an ndarray from ragged nested
sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays
with different lengths or shapes) is deprecated. If you meant to do
this, you must specify 'dtype=object' when creating the ndarray.
  yuv = mat.dot(np.array([R,G,B,1.0]))
C:\Users\surface\AppData\Local\Temp\ipykernel_15684\2878482894.py:61:
DeprecationWarning: `np.int` is a deprecated alias for the builtin
`int`. To silence this warning, use `int` by itself. Doing this will
not modify any behavior and is safe. When replacing `np.int`, you may
wish to use e.g. `np.int64` or `np.int32` to specify the precision. If
you wish to review your current use, check the release note link for
additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if (img.dtype != np.int) and (use_float == False):
Clipping input data to the valid range for imshow with RGB data
([0..1] for floats or [0..255] for integers).
```


Y Channel Histogram
1e7
Frequency
Value
0.0
0.2
0.4
0.6
0.8
1.0
0
2000
4000
6000
8000
10000
12000


U Channel Histogram
1e7
Frequency
Value
0.0
0.2
0.4
0.6
0.8
1.0
−1500
−1000
−500
0
500
1000
1500
2000


V Channel Histogram
1e7
Frequency
Value
0.0
0.2
0.4
0.6
0.8
1.0
1.2
−3000
−2000
−1000
0
1000
2000


YUV Image

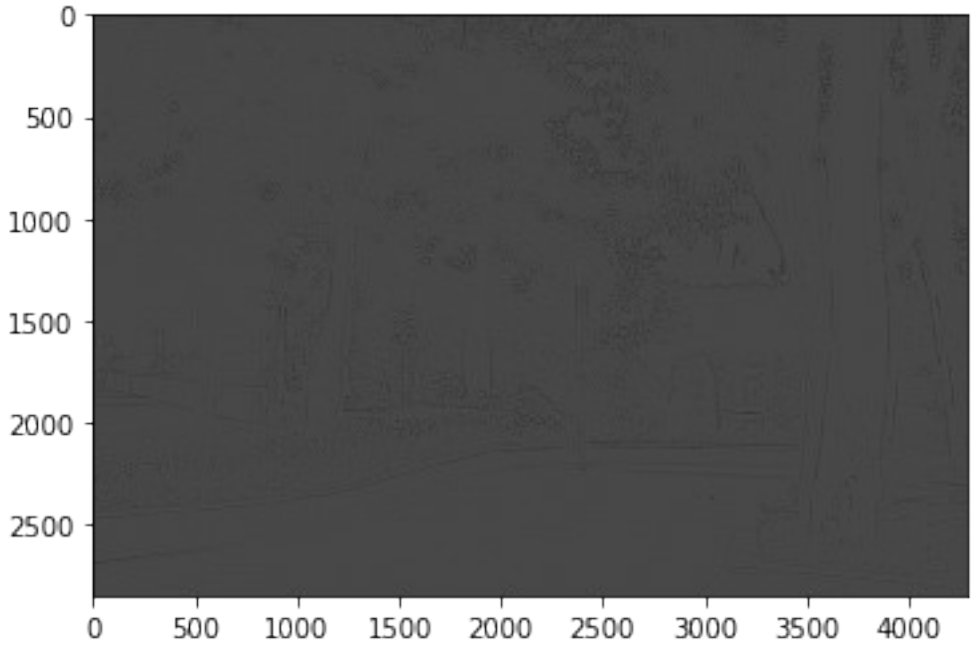



```
def bilateral_filtering(args, img:np.ndarray, sig1, sig2, radius =
15):
    print("range_sigma = ", range_sigma)
    print("spatial_sigma = ", spatial_sigma)
    print(type(img), img.dtype)
    img = img.astype(np.float32)
    if args.fast:
        base = cv2.bilateralFilter(img, radius, range_sigma,
spatial_sigma)

    else:
        radius = 3
        h, w = img.shape[0], img.shape[1]
        img_pad = np.pad(img, radius+1)
        print(img_pad.shape)
        ys, xs = np.meshgrid(np.arange(h), np.arange(w))
        P = img_pad[ys+radius, xs+radius]
        W = np.zeros_like(img.T)
        base = np.zeros_like(img.T)
```

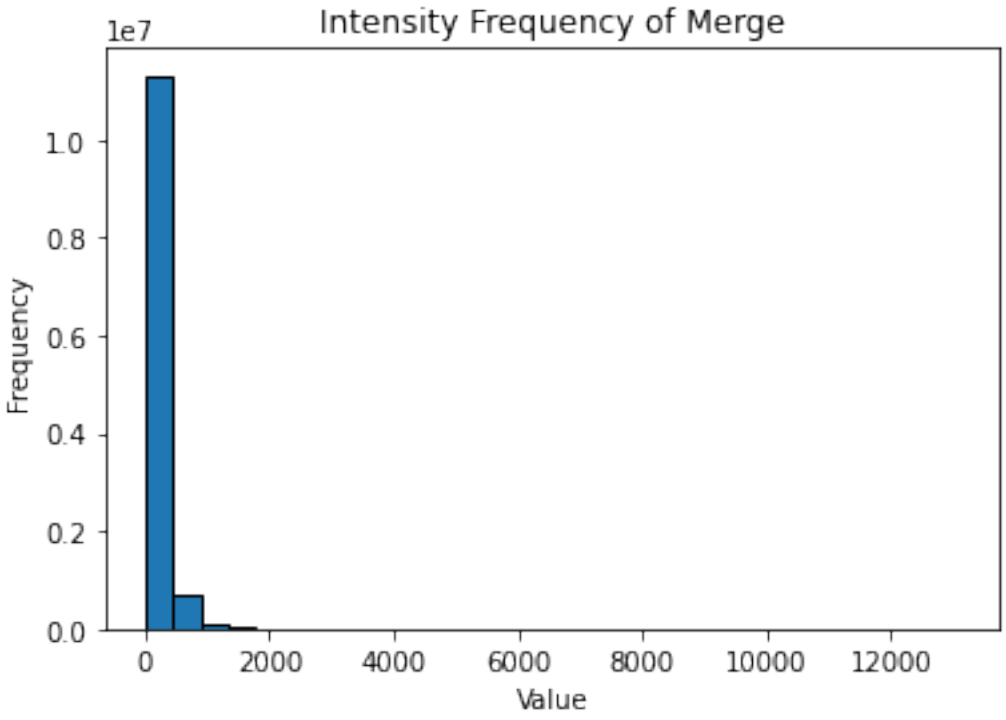
```
        for i in tqdm(range(2*radius-1)):
            for j in tqdm(range(2*radius-1)):

                Q = img_pad[ys+i, xs+j]
                val = P-Q
                gr = 1/(sig1*np.sqrt(2*np.pi)) * np.exp(-
0.5*(val**2)/(sig1**2))
                dis = np.sqrt((i-radius)**2+(j-radius)**2 )
                gs = 1/(sig2*np.sqrt(2*np.pi)) * np.exp(-
0.5*(dis**2)/(sig2**2))

                grgs = gr*gs

                base += grgs*Q
                W += grgs

        base/=W
        base = base.T
    return base


range_sigma = 200
width, height, _ = rgbimg_cfa.shape
spatial_sigma =  0.02 * min(width,height)
Y = YUV[:, :, 0]
Y_base = bilateral_filtering(args, Y, range_sigma, spatial_sigma)
# Y_base = fastbilateral2d(Y, range_sigma, spatial_sigma)
import copy

Y_detail = copy.deepcopy(Y)
m = Y_base!=0
Y_detail[m] = Y[m]/Y_base[m]
cv2.imwrite(os.path.join(args.itmd_dir, "Y.jpg"), Y)
cv2.imwrite(os.path.join(args.itmd_dir, "Y_base.jpg"), Y_base)
cv2.imwrite(os.path.join(args.itmd_dir, "Y_detail.jpg"),
normalize(Y_detail)*255)
plt.imshow(Y_detail, cmap="gray")
```

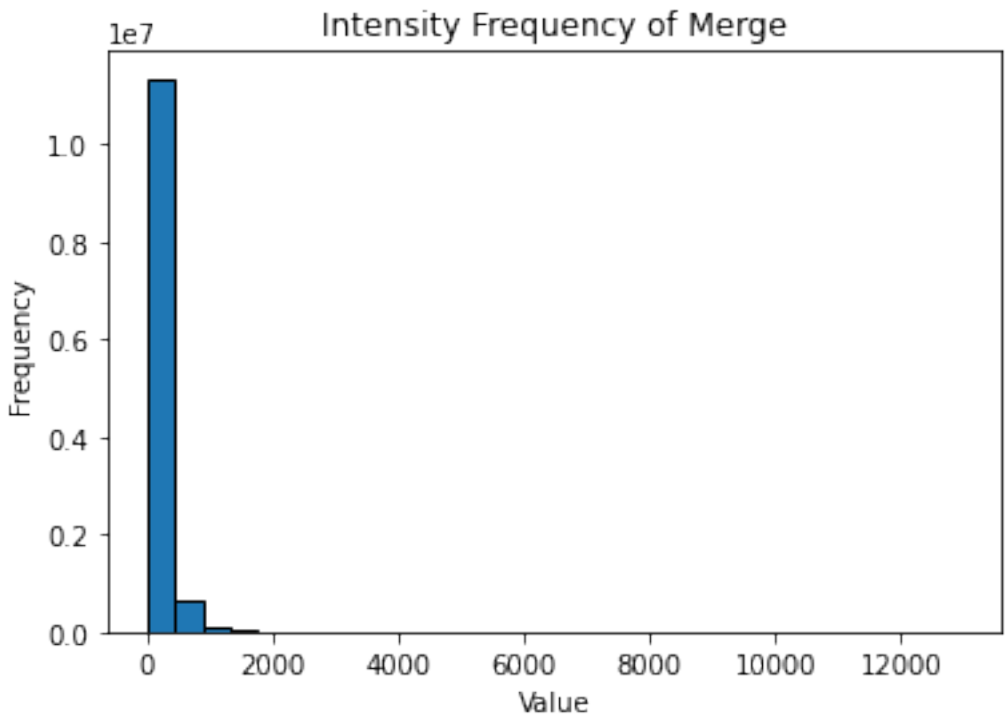
```
range_sigma =  200
spatial_sigma =  56.96
<class 'numpy.ndarray'> float64
```

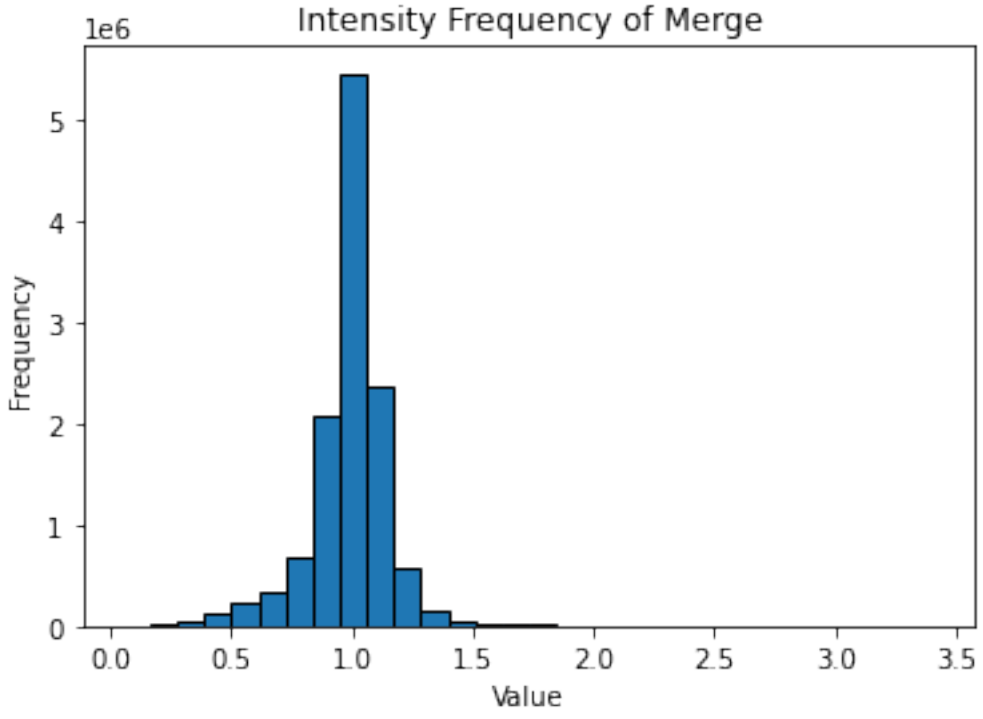
```
<matplotlib.image.AxesImage at 0x245bb897fd0>
```




```
stat_draw(Y)
stat_draw(Y_base)
stat_draw(Y_detail)
```

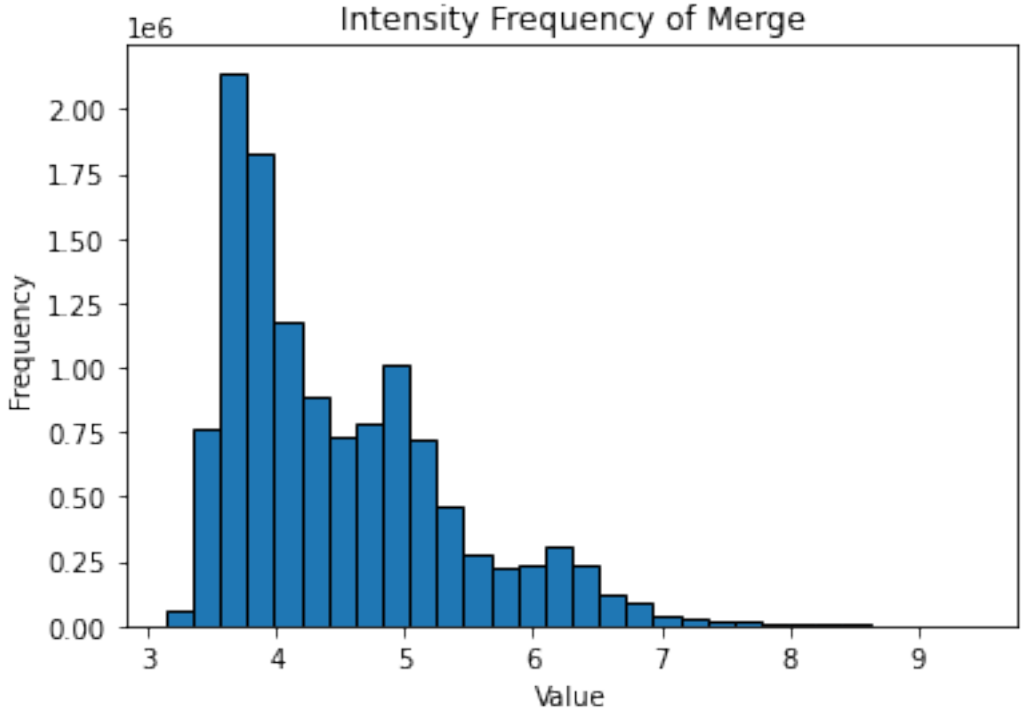


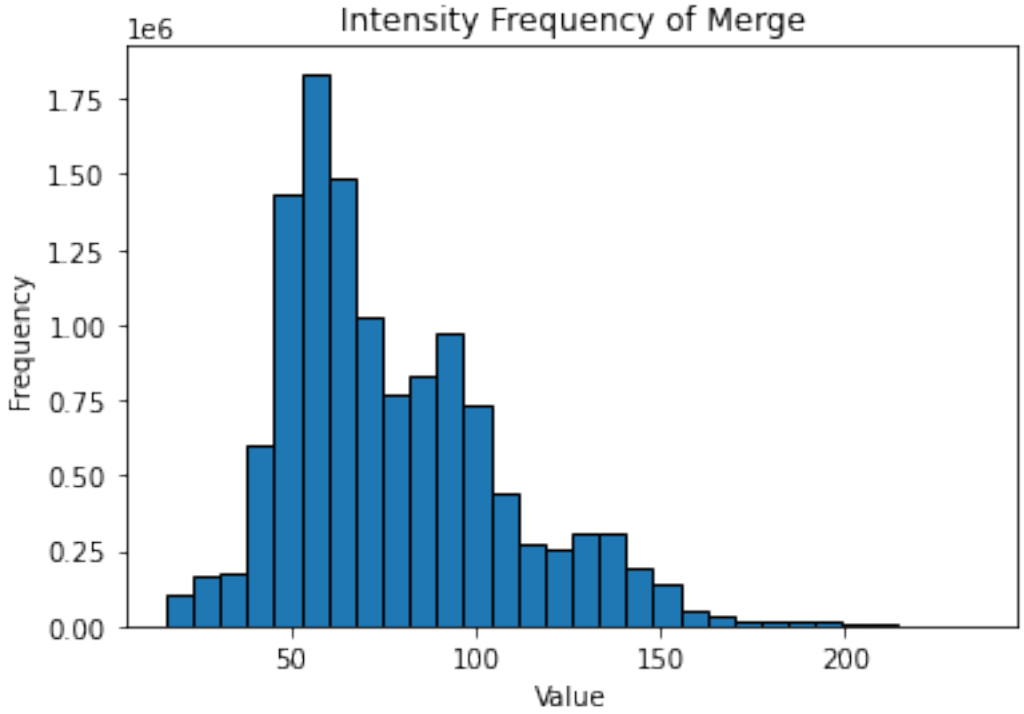


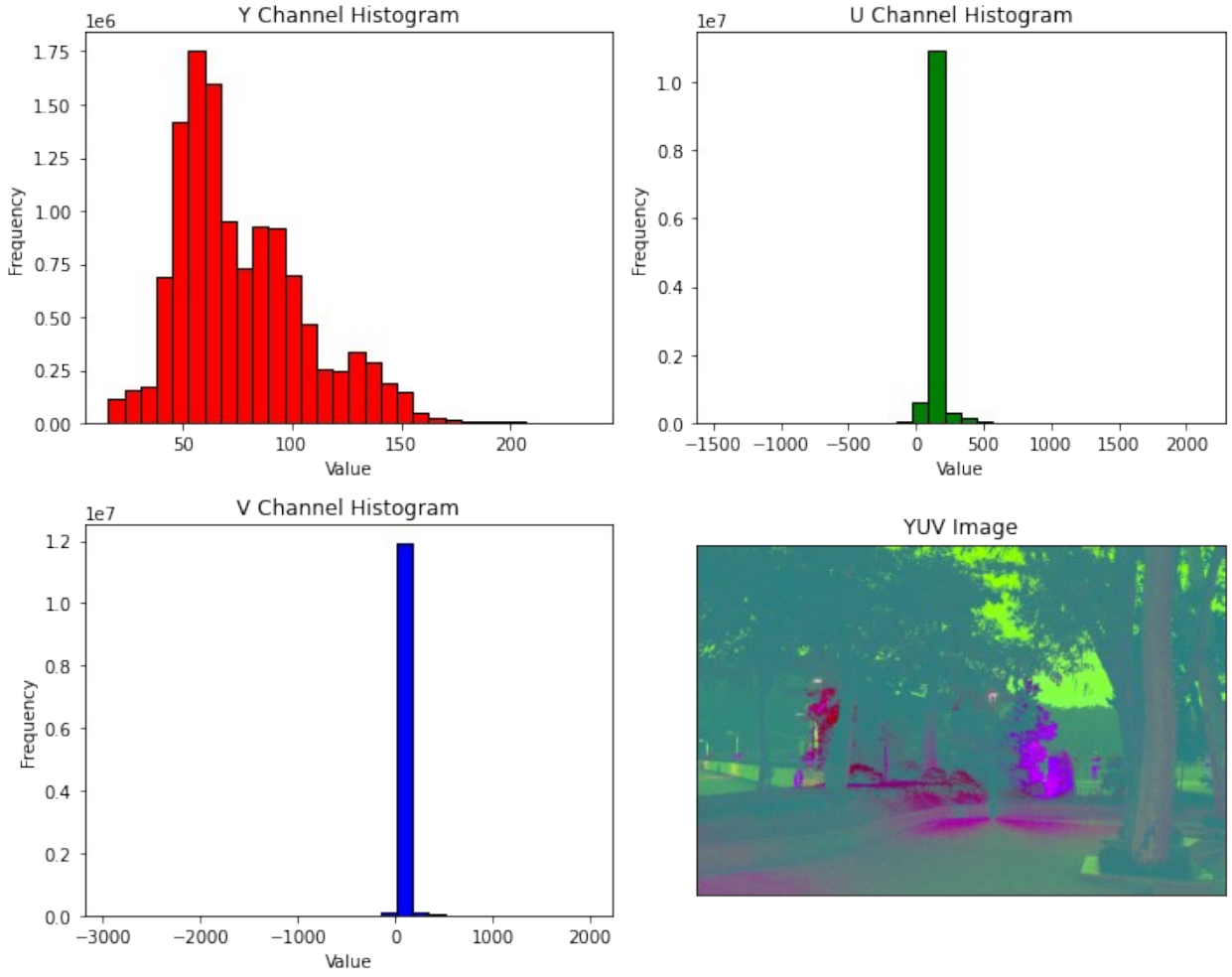

```
Y_base_log = np.clip(np.log(Y_base), 2, 10)
stat_draw(Y_base_log)
Y_new = normalize(normalize(Y_base_log)+normalize(Y_detail))
stat_draw(Y_new*220+16)
```

```
YUV[:, :, 0] = Y_new*220+16
stat_draw_yuv(YUV)
```



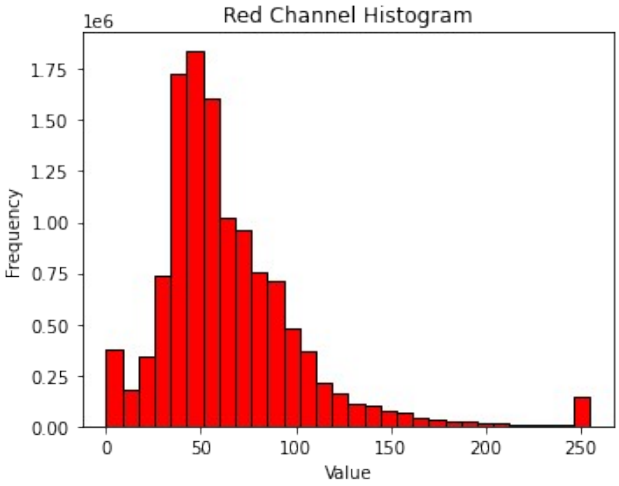
```
C:\Users\surface\AppData\Local\Temp\ipykernel_15684\2878482894.py:61:
DeprecationWarning: `np.int` is a deprecated alias for the builtin
`int`. To silence this warning, use `int` by itself. Doing this will
not modify any behavior and is safe. When replacing `np.int`, you may
wish to use e.g. `np.int64` or `np.int32` to specify the precision. If
you wish to review your current use, check the release note link for
additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if (img.dtype != np.int) and (use_float == False):
Clipping input data to the valid range for imshow with RGB data
([0..1] for floats or [0..255] for integers).
```

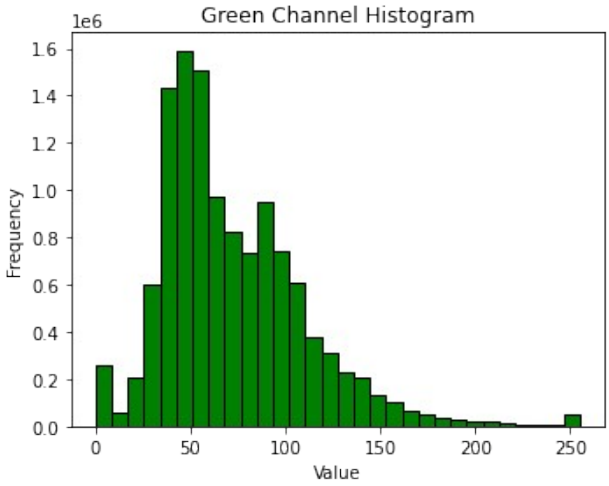


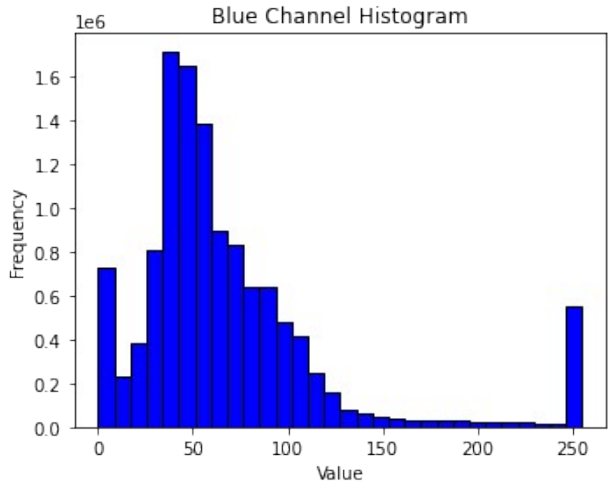

```python
def YUV2RGB(img):
    Y = img[:,:,0] - 16
    U = img[:,:,1] -128
    V = img[:,:,2] -128
    # your code here
    R = 1.164 * Y             + 1.596 * V
```

```
    G = 1.164 * Y - 0.392 * U - 0.813 * V
    B = 1.164 * Y + 2.017 * U
    return np.clip(np.stack([R,G,B], axis=2),0,255)
rgb_new = YUV2RGB(YUV)
stat_draw_rbg(rgb_new)
```

```
C:\Users\surface\AppData\Local\Temp\ipykernel_15684\2878482894.py:27:
DeprecationWarning: `np.int` is a deprecated alias for the builtin
`int`. To silence this warning, use `int` by itself. Doing this will
not modify any behavior and is safe. When replacing `np.int`, you may
wish to use e.g. `np.int64` or `np.int32` to specify the precision. If
you wish to review your current use, check the release note link for
additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if (img.dtype != np.int) and (use_float == False):
C:\Users\surface\AppData\Local\Temp\ipykernel_15684\2878482894.py:28:
DeprecationWarning: `np.int` is a deprecated alias for the builtin
`int`. To silence this warning, use `int` by itself. Doing this will
not modify any behavior and is safe. When replacing `np.int`, you may
wish to use e.g. `np.int64` or `np.int32` to specify the precision. If
you wish to review your current use, check the release note link for
additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  img = img.astype(np.int)
```





```
##########################################################################
########################################
#####################################    Part 2: YUV Domain Processing
Steps  ###################################
##########################################################################
########################################
# Step Luma-2  Edge Enhancement  for Luma (20pts)
edge_filter = np.asarray([[-1.,  0., -1.,  0., -1.], [-1.,  0.,  8.,
0., -1.], [-1.,  0., -1.,  0., -1.]])
ee_gain, ee_thres, ee_emclip = [32, 128],  [32, 64], [-64, 64]

ee = EdgeEnhancement(YUV[:,:,0], edge_filter, ee_gain, ee_thres,
ee_emclip)
yuvimg_ee,yuvimg_edgemap = ee.execute()
print(50*'-' + '\n 3.Luma.2  Edge Enhancement Done......')
```

```
(2850, 4292)
--------------------------------------------------
 3.Luma.2  Edge Enhancement Done......
```

```
# Step Luma-3 Brightness/Contrast Control (5pts)
brightness, contrast, bcc_clip = 0, 0, 255
contrast = contrast / pow(2,5)    #[-32,128]
bcc = BrightnessContrastControl(yuvimg_ee, brightness, contrast,
bcc_clip)
yuvimg_bcc = bcc.execute()
print(50*'-' + '\nBrightness/Contrast Adjustment Done......')
```

```
--------------------------------------------------
Brightness/Contrast Adjustment Done......
```

```
# Step Chroma-1 False Color Suppresion (10pts)
fcs_edge, fcs_gain, fcs_intercept, fcs_slope = [32, 32], 32, 2, 3
fcs = FalseColorSuppression(YUV[:,:,1:3], yuvimg_edgemap, fcs_edge,
fcs_gain, fcs_intercept, fcs_slope)
yuvimg_fcs = fcs.execute()
print(50*'-' + '\n 3.Chroma.1 False Color Suppresion Done......')
```

```
--------------------------------------------------
 3.Chroma.1 False Color Suppresion Done......
```

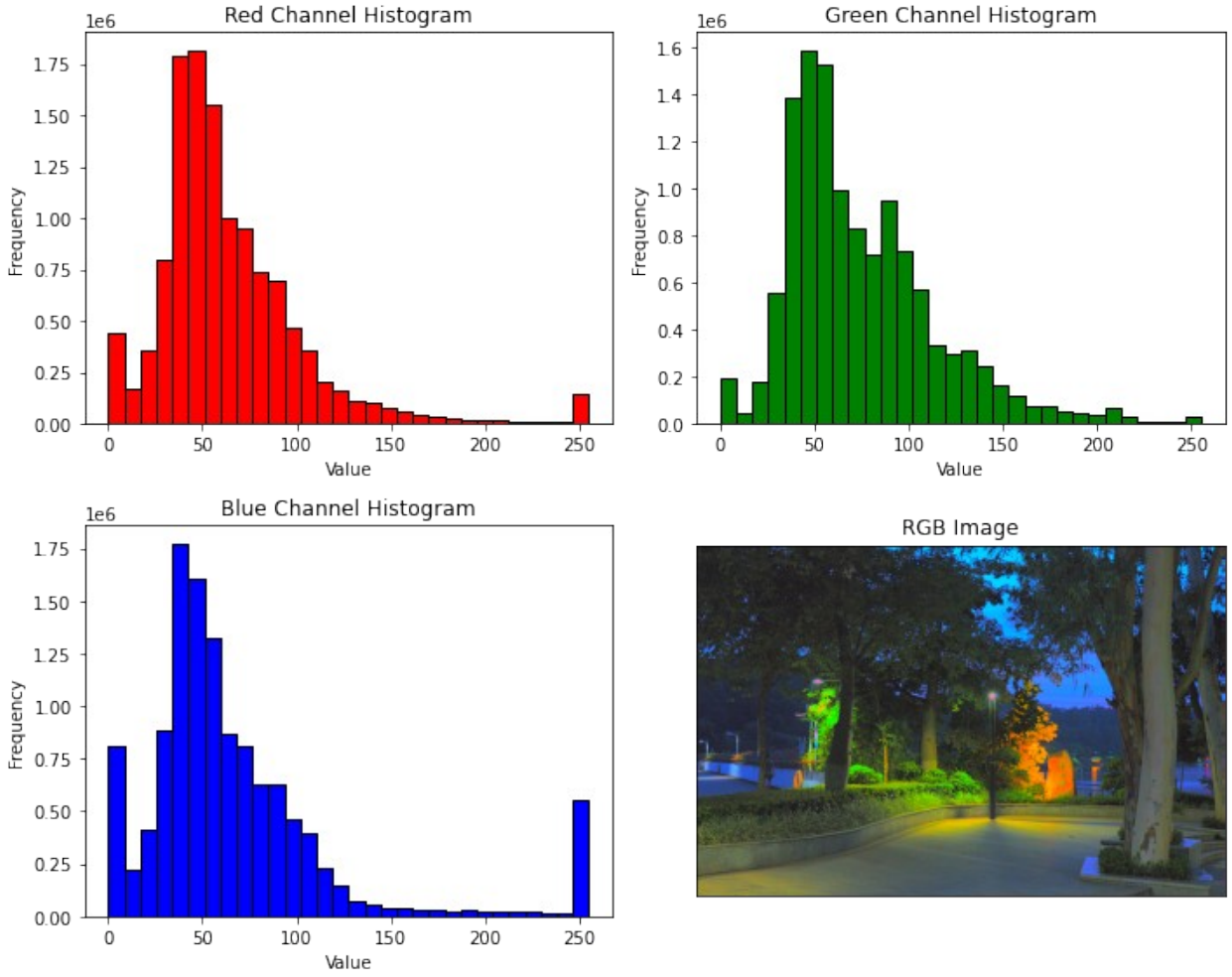
```
# Step Chroma-2 Hue/Saturation control (10pts)
hue, saturation, hsc_clip = 128, 256, 255
hsc = HueSaturationControl(yuvimg_fcs, hue, saturation, hsc_clip)
yuvimg_hsc = hsc.execute()
print(50*'-' + '\n 3.Chroma.2  Hue/Saturation Adjustment Done......')
```

```
--------------------------------------------------
 3.Chroma.2  Hue/Saturation Adjustment Done......
```

```
# Concate Y UV Channels
yuvimg_out = np.zeros_like(YUV)
yuvimg_out[:,:,0] = yuvimg_bcc
yuvimg_out[:,:,1:3] = yuvimg_hsc

RGB = YUV2RGB(yuvimg_out) # Pay attention to the bits
stat_draw_rbg(RGB)
```

```
C:\Users\surface\AppData\Local\Temp\ipykernel_15684\2878482894.py:27:
DeprecationWarning: `np.int` is a deprecated alias for the builtin
`int`. To silence this warning, use `int` by itself. Doing this will
not modify any behavior and is safe. When replacing `np.int`, you may
wish to use e.g. `np.int64` or `np.int32` to specify the precision. If
you wish to review your current use, check the release note link for
additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if (img.dtype != np.int) and (use_float == False):
C:\Users\surface\AppData\Local\Temp\ipykernel_15684\2878482894.py:28:
DeprecationWarning: `np.int` is a deprecated alias for the builtin
`int`. To silence this warning, use `int` by itself. Doing this will
```

```
not modify any behavior and is safe. When replacing `np.int`, you may
wish to use e.g. `np.int64` or `np.int32` to specify the precision. If
you wish to review your current use, check the release note link for
additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  img = img.astype(np.int)
```




```
#####################################   End YUV Domain Processing
Steps  ###################################
#########################################################################
##########################################
RGB = RGB.astype(np.uint8)
result_dir = os.path.join(args.sub_dir, "output.png")

cv2.imwrite(result_dir,cv2.cvtColor(RGB,cv2.COLOR_RGB2BGR))

True
```