IIAI30013

Computer Vision

HW1: Image Sensing Pipeline

Instructor: YuanFu Yang

TA: 陳政錡

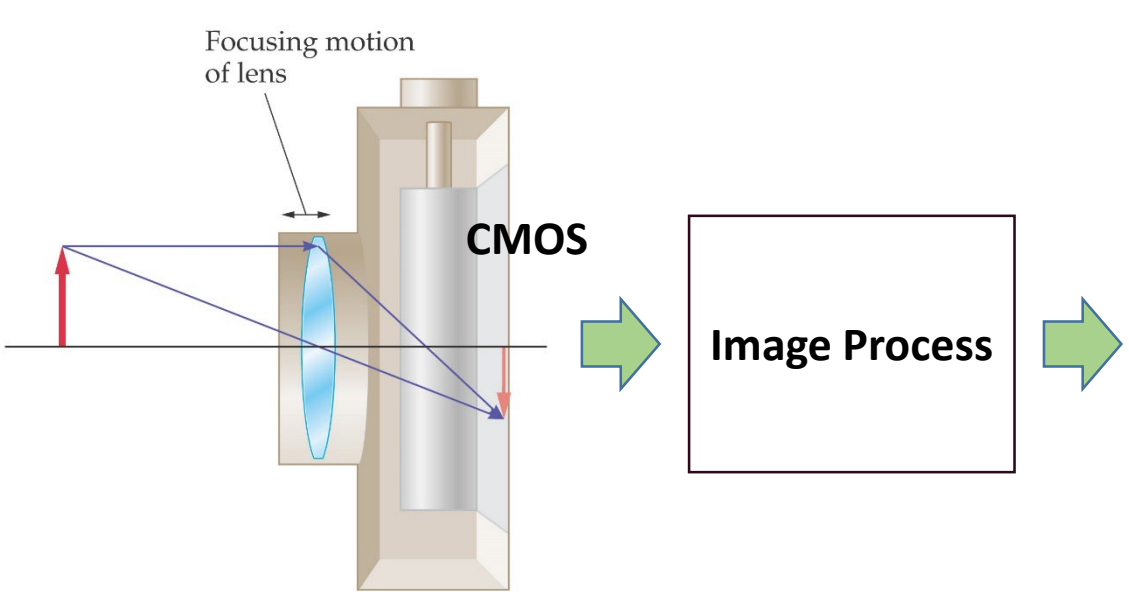
https://github.com/Rossi-Laboratory/Course-Lectures/blob/main/Computer%20Vision/Assignment1/Image%20Sensing%20Pipeline.md

Image Sensing Pipeline

- Homework due: 3/18
- Late submissions will incur a penalty of one point for each day overdue.
- The assignment allows a maximum extension of 3 days (it will not be accepted if submitted later than 3 days).
- Submit files: code and report (4 questions), and submit them in both **.ipynb** and **PDF** file formats respectively.
- This assignment can be carried out using Colab or completed on your PC.
- Please refer to the questions on page 19 of HW1.pdf for the four questions in the report.

Image Sensing Pipeline

- Digital Camera



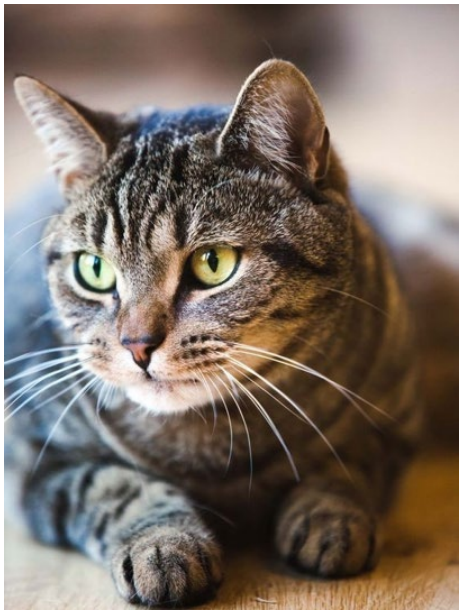


Image Sensing Pipeline

- Image Sensing Pipeline

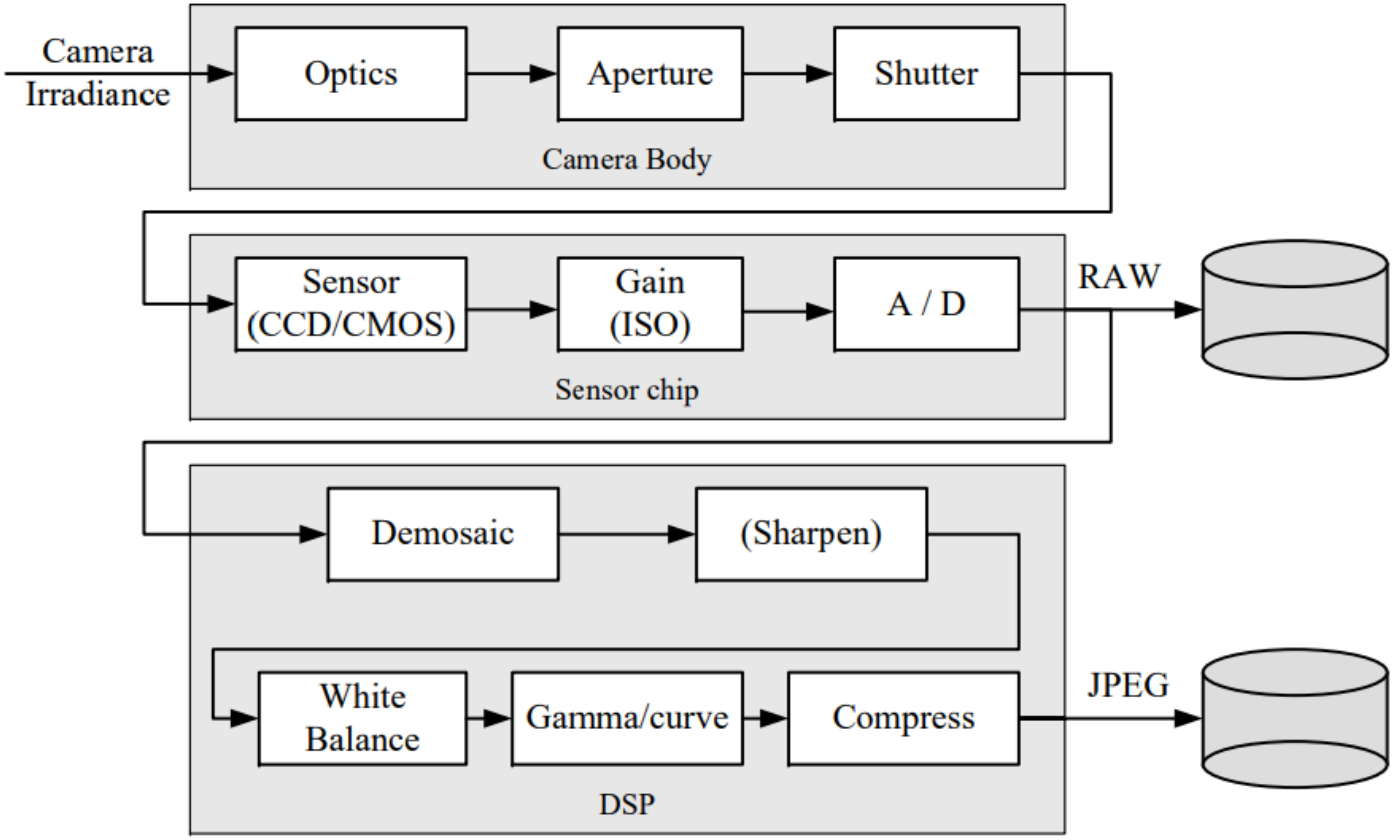


Image Sensing Pipeline

- This homework:

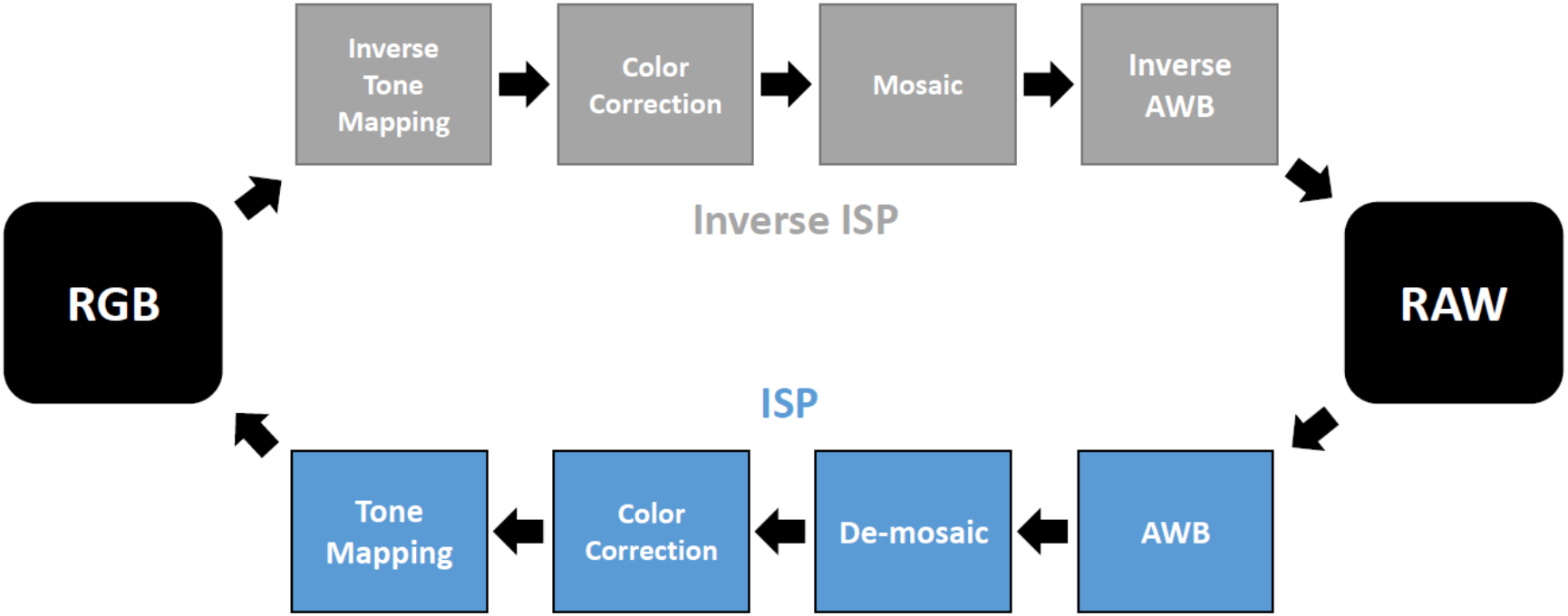


Image Sensing Pipeline

- Example: Inverse ISP

Oringal Image

Image after inverse tone

Image after CIE

Image after CCM

Image after mosaic

Image Sensing Pipeline

- Example: ISP

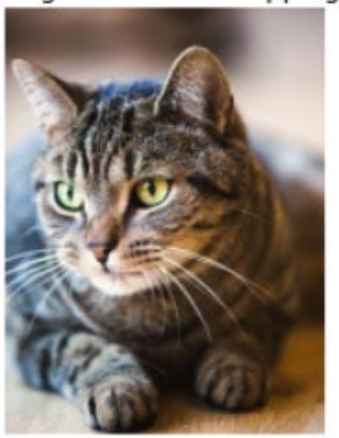
Image after WG

Image after demosaic

Image after Color Correction

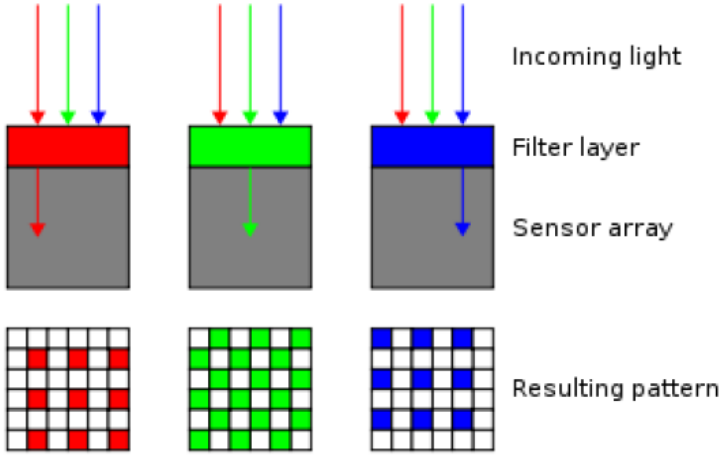
Image after XYZ2RGB

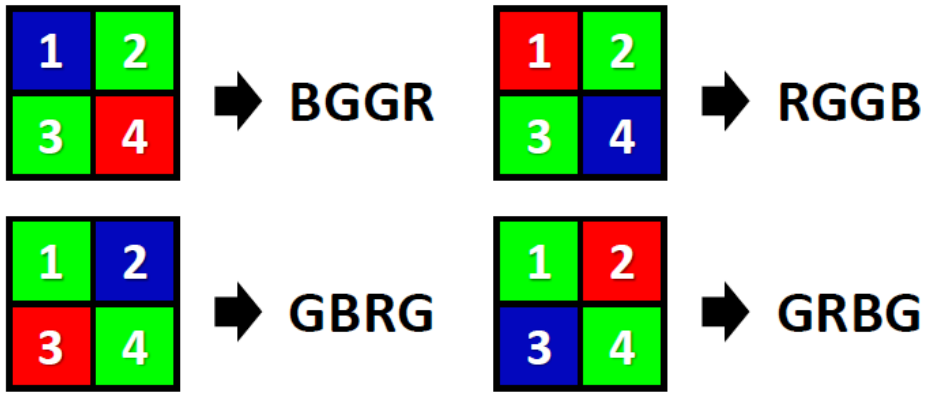
Image after tone mapping



Raw Data (rawRGB)

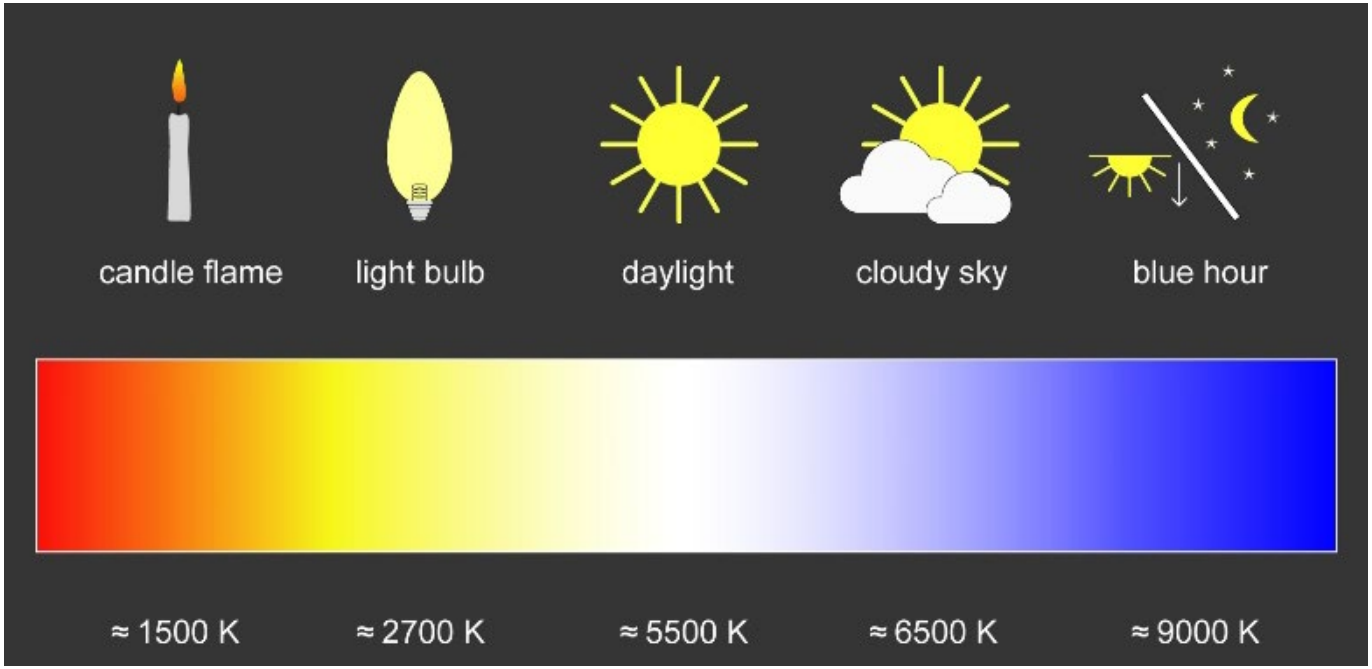
- Bayer pattern (CFA, Color Filter Arrays)
  - Green photo sensors: luminance sensitive elements
  - Red and blue photo sensors: chrominance sensitive elements
  - Twice as many green elements as red or blue to mimic the physiology of the human eye
  - Four patterns: BGGR, RGGB, GBRG, GRBG





AWB (Auto White Balance)

- WB: mimics chromatic adaptation of the eye
  - Can be manual settings
  - Stored in metadata



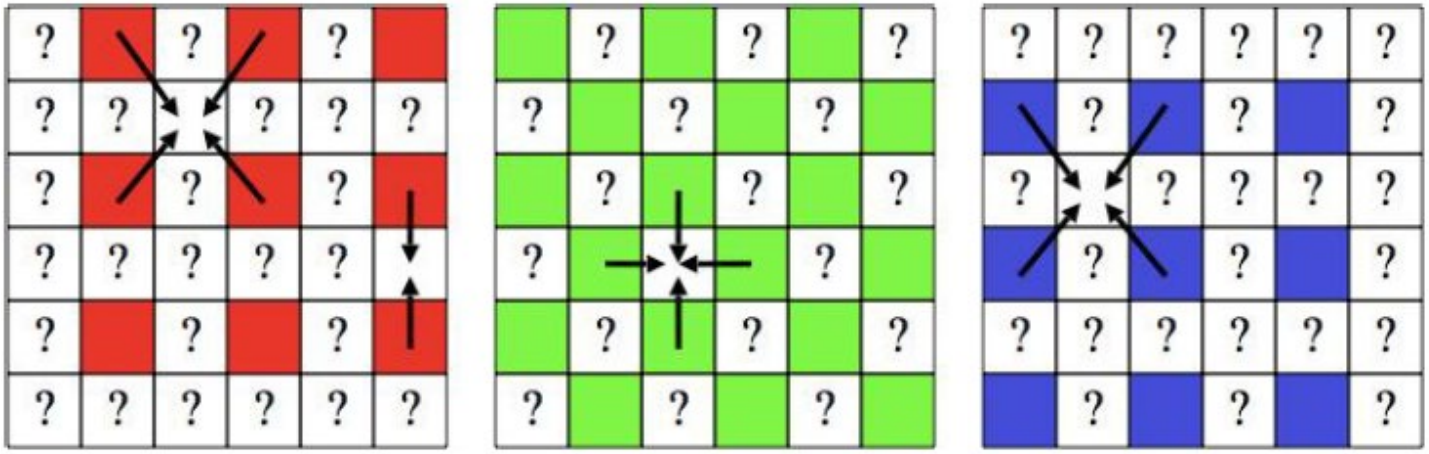
AWB (Auto White Balance)

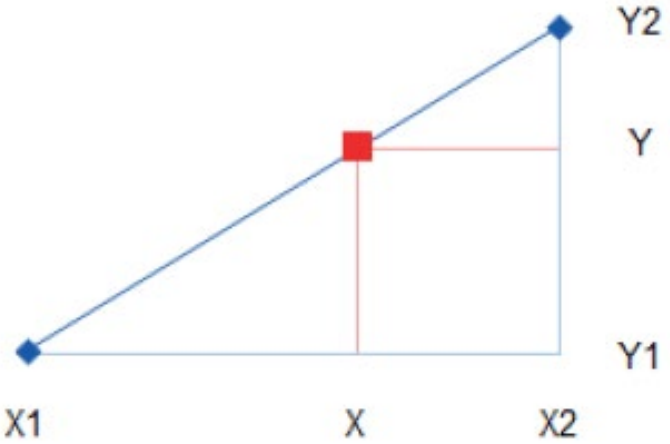
- AWB: attempts to make what is assumed to be white map to “pure white
  - Two classical methods: gray world algorithm and white patch algorithm
  - In this homework, you only need to implement easiest AWB method (plz refer to TODO)

```
def generate_wb_mask(img, pattern, fr, fb):
    '''
    Input:
        img: H*W numpy array, RAW image
        pattern: string, 4 different Bayer patterns (GRBG, RGGB, GBRG, BGGR)
        fr: float, white balance factor of red channel
        fb: float, white balance factor of blue channel
    Output:
        mask: H*W numpy array, white balance mask
    '''
    ############################################################################
    # TODO:                                                                    #
    #   1. Create a numpy array with shape of input RAW image.                 #
    #   2. According to the given Bayer pattern, fill the fr into              #
    #      correspinding red channel position and fb into correspinding        #
    #      blue channel position. Fill 1 into green channel position           #
    #      otherwise.                                                          #
    ############################################################################
```

Demosaic

- Easiest method Linear Interpolation





$$\frac{(X - X1)}{(X2 - X1)} = \frac{(Y - Y1)}{(Y2 - Y1)}$$

$$Y = Y1 + (X - X1)\frac{(Y2 - Y1)}{(X2 - X1)}$$

Demosaic

- Python Library colour_demosaicing
  - demosaicing_CFA_Bayer_bilinear
  - demosaicing_CFA_Bayer_Malvar2004

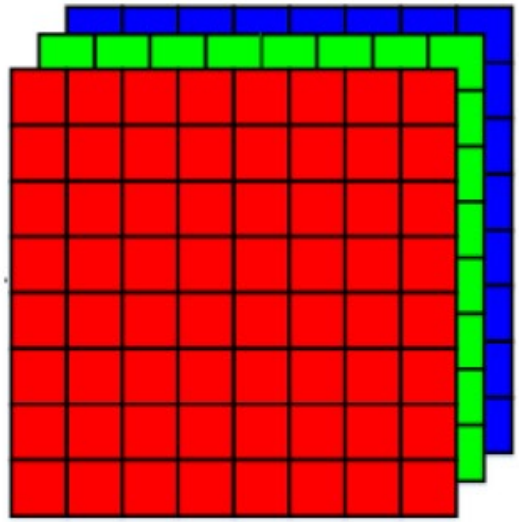
    [MHCW04] Henrique S Malvar , Li Wei He, Ross Cutler, and One Microsoft Way. High Quality Linear Interpolation for Demosaicing of Bayer Patterned Color Images. In International Conference of Acoustic, Speech and Signal Processing, 5 8. Institute of Electrical and Electronics Engineers, Inc., May 2004.
  - demosaicing_CFA_Bayer_Menon2007

    [MAC07] Daniele Menon, Stefano Andriani , and Giancarlo Calvagno . Demosaicing With Directional Filtering and a posteriori Decision. IEEE Transactions on Image Processing, 16(1):132 141, January 2007.

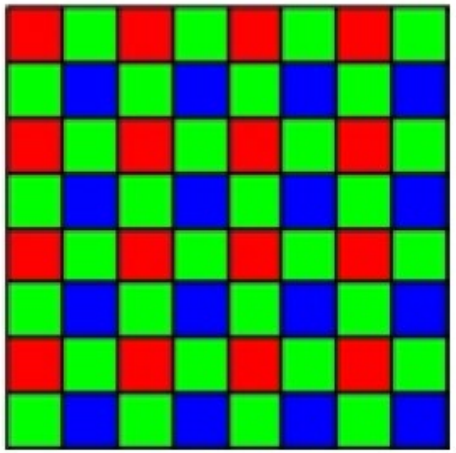
```
from colour_demosaicing import (
    EXAMPLES_RESOURCES_DIRECTORY,
    demosaicing_CFA_Bayer_bilinear,
    demosaicing_CFA_Bayer_Malvar2004,
    demosaicing_CFA_Bayer_Menon2007,
    mosaicing_CFA_Bayer)
```

Mosaic

- Discard the value of other 2 channels
  - H*W*3 → H*W
- Python Library colour_demosaicing
  - mosaicing_CFA_Bayer



Bayer Pattern : RGGB



Mosaic

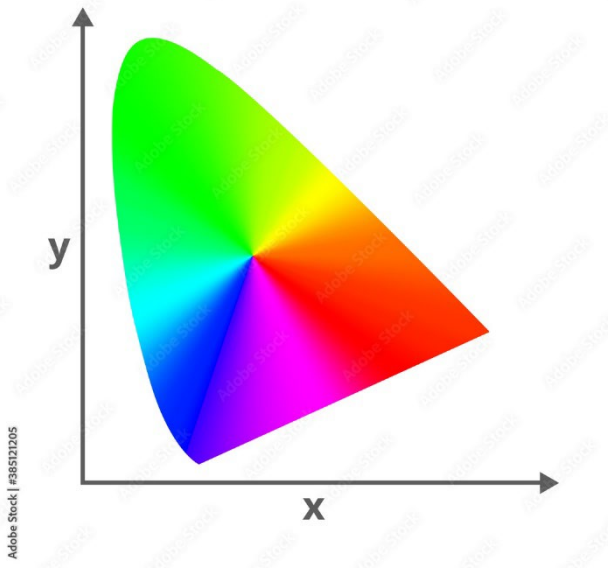
```python
def mosaic(img, pattern):
    '''
    Input:
        img: H*W*3 numpy array, input image.
        pattern: string, 4 different Bayer patterns (GRBG, RGGB, GBRG, BGGR)
    Output:
        output: H*W numpy array, output image after mosaic.
    '''
    ###############################################################################
    # TODO:                                                                       #
    #   1. Create the H*W output numpy array.                                     #
    #   2. Discard other two channels from input 3-channel image according        #
    #      to given Bayer pattern.                                                #
    #                                                                             #
    #   e.g. If Bayer pattern now is BGGR, for the upper left pixel from          #
    #        each four-pixel square, we should discard R and G channel            #
    #        and keep B channel of input image.                                   #
    #        (since upper left pixel is B in BGGR bayer pattern)                  #
    ###############################################################################



    ###############################################################################
    #                                                                             #
    #                           End of your code                                  #
    #                                                                             #
    ###############################################################################

    return output
```

Color Collection

- CCM (Color Correction Matrix)
  - Transforms sensor native **RGB** values into **CIE XYZ** color space.
  - It is important that the white balance has been performed correctly.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4887180 & 0.3106803 & 0.2006017 \\ 0.1762044 & 0.8129847 & 0.0108109 \\ 0.0000000 & 0.0102048 & 0.9897952 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

The Color Correction Matrix



Color Collection

- CCM (Color Correction Matrix)

1.2 Color Correction Matrix

```
ccm = np.array([1.0234, -0.2969, -0.2266,
                -0.5625, 1.6328, -0.0469,
                -0.0703, 0.2188, 0.6406])
ccm = np.reshape(ccm, (3, 3))
ccm = (ccm / np.tile(np.sum(ccm, axis=1), [3, 1]).T).T
ccm_inv = np.linalg.inv(np.copy(ccm))
```

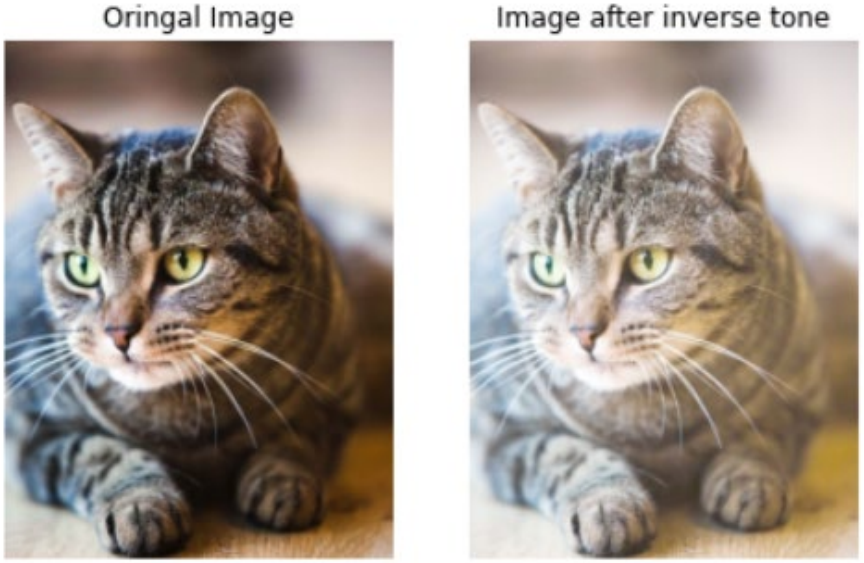
```
def color_correction(img, ccm):
    '''
    Input:
        img: H*W*3 numpy array, input image
        ccm: 3*3 numpy array, color correction matrix
    Output:
        output: H*W*3 numpy array, output image after color correction
    '''
    ###########################################################################
    # TODO:                                                                   #
    #   Following above instruction to get color correction result            #
    #                                                                         #
    ###########################################################################
```

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = ccm \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

```
    ###########################################################################
    #                                                                         #
    #                           End of your code                              #
    #                                                                         #
    ###########################################################################

    #### Prevent the value larger than 1 or less than 0
    output = np.clip(output, 0, 1)
    return output
```

Tone Mapping

- Display cannot afford the brightness in real world
  - use more sensor bits to store
- High Dynamic Range (HDR) => Low Dynamic Range (LDR)
  - compress sensor bits into 8 bit (RGB24)
  - makes images suitable to be viewed on a digital screen
- Tone curve: Non linear mapping of RGB tones



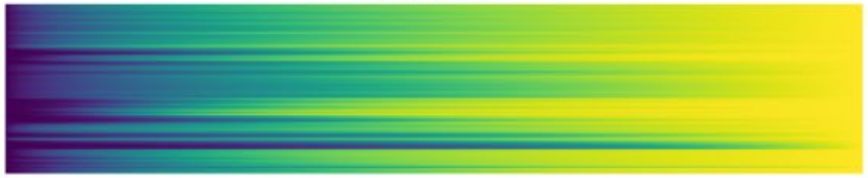
Tone Mapping

- Download the ipynb file, tone_curves.mat, and tone_curves_inv.mat.
- Ensure that all files are placed in the same folder directory.

```
curve_name = os.path.join(curve_path, 'tone_curves.mat')
curve_inv_name = os.path.join(curve_path, 'tone_curves_inv.mat')
```

Irradiance

Brightness



```
import numpy as np
import math

def tone_mapping(img, I, B, index=0, inv=False):
    '''
    Input:
        img: H*W*3 numpy array, input image.
        I: 201*1024 array, represents 201 tone curves for Irradiance.
        B: 201*1024 array, represents 201 tone curves for Brightness.
        index: int, choose which curve to use, default is 0
        inv: bool, judge whether tone mapping (False) or inverse tone mapping (True), default is False
    Output:
        output: H*W*3 numpy array, output image afte (inverse) tone mapping.
    '''
    ###########################################################################
    # TODO:                                                                   #
    #   Following above instruction to get tone mapping as output.            #
    #   and inverse tone mapping result as output                             #
    ###########################################################################
```

End Result

- Calculate the PSNR of the original image and the restored image generated after the Inverse ISP and ISP process.

Oringal Image

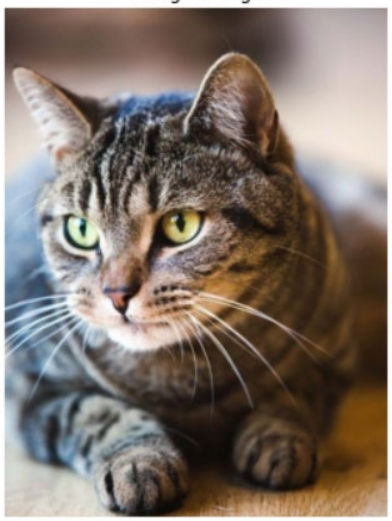


Image after inverse process

```python
def calculate_psnr(img1, img2):
    '''
    Input:
        img1, img2: H*W*3 numpy array
    Output:
        psnr: the peak signal-to-noise ratio value
    '''
    ###########################################################################
    # TODO:                                                                   #
    #   Following above instruction to get PSNR as output.                    #
    #                                                                         #
    ###########################################################################


    ###########################################################################
    #                                                                         #
    #                           End of your code                              #
    #                                                                         #
    ###########################################################################
    return psnr
```

```
calculate_psnr(img, img_tm)
```

```
81.43537077717903
```

Image Sensing Pipeline

- Homework Question:

  a. Discuss different treatments of different Bayer patterns when: i. applying white balance mask into original image. ii. doing mosaic algorithm.

  b. Show the image results of each step as p.6/7 in HW1.pdf.

  c. Show the image results of inverse ISP and ISP as p.19 in HW1.pdf. Additionally, compare the performance results of this task using PSNR.

  d. In recent AI de-noising methods, in order to generate paired data for training, we will add synthetic noise to clean image on RAW domain instead of RGB domain. Explain the reason.

IIAI30013

Computer Vision

HW1: Image Sensing Pipeline

Q & A

Instructor: YuanFu Yang

TA: 陳政錡

https://github.com/Rossi-Laboratory/Course-Lectures/blob/main/Computer%20Vision/Assignment1/Image%20Sensing%20Pipeline.md