數位影像處理 Final Project Report

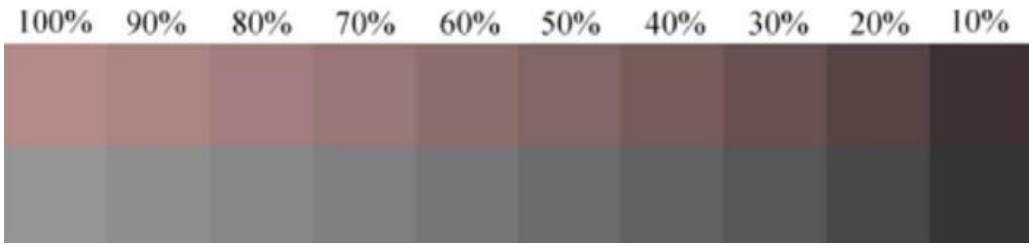
電機所 碩一

陳昭宇 R13921075

陳柏丞 R13921109

1. Abstract：

We referred to Professor Homer Chen's paper, which addresses the issue of reduced brightness and color accuracy under low backlight conditions. Inspired by the methods proposed in the paper, we aim to restore the overall brightness and color accuracy of images under low backlight conditions, making them comparable to the original images.

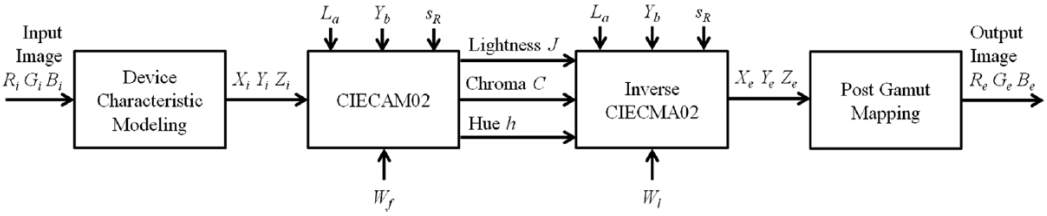


2. Introduction：

Dim backlight conditions significantly degrade image brightness and color accuracy, impacting the visual experience on energy-saving multimedia devices such as smartphones. Inspired by the **human visual system's anchoring property**, this project proposes a method to enhance color appearance under such challenging conditions. The anchoring property allows our perception to adapt to relative visual references, making it possible to compensate for dim lighting. By leveraging this concept, we developed a solution that integrates luminance and chrominance processing for natural color enhancement while maintaining computational efficiency. This approach is capable of processing high-definition

images and videos at real-time speeds, offering an effective and practical enhancement for modern low-backlight displays.

3. Formula



This is the method process we use.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \mathbf{M} \begin{bmatrix} R_l \\ G_l \\ B_l \end{bmatrix} = \begin{bmatrix} m_{rx} & m_{gx} & m_{bx} \\ m_{ry} & m_{gy} & m_{by} \\ m_{rz} & m_{gz} & m_{bz} \end{bmatrix} \begin{bmatrix} R^{\gamma_r} \\ G^{\gamma_g} \\ B^{\gamma_b} \end{bmatrix} \quad (1)$$

It's the device characteristic modeling to transform the RGB values into XYZ values.

$$(J, C, h) = \psi(X_i, Y_i, Z_i, W_f, L_a, Y_b, s_R) \quad (2)$$

Then, we incorporate environmental parameters such , L_a, Y_b *and* S_R along with the comparison value W_f, to compute the perceptual attributes J, C *and* h(Lightness, Chroma, and Hue) using the CIECAM02 model.

$$(X_e, Y_e, Z_e) = \psi^{-1}(J, C, h, W_l, L_a, Y_b, s_R). \quad (3)$$

Next, we apply the same environmental parameters , L_a, Y_b *and* S_R but with a different W_l, to perform an inverse computation. This step converts the perceptual attributes J, C *and* h back into the X_e, Y_e *and* Z_e values.

$$[R_{e,l}\ G_{e,l}\ B_{e,l}]^{\mathrm{T}} = \mathbf{M}_l^{-1}[X_e\ Y_e\ Z_e]^{\mathrm{T}} \quad (4)$$

$$(R', G', B') = \left(R_{e,l}^{\frac{1}{\gamma_{r,l}}}, G_{e,l}^{\frac{1}{\gamma_{g,l}}}, B_{e,l}^{\frac{1}{\gamma_{b,l}}} \right) \quad (5)$$

$$(R_c, G_c, B_c) = (f(R'), f(G'), f(B')) \quad (6)$$

where

$$f(x) = \begin{cases} x, & \text{if } 0 \leq x \leq 1 \\ 1, & \text{if } x > 1 \\ 0, & \text{if } x < 0. \end{cases} \quad (7)$$

After applying a series of inverse transformation equations, we obtained R_c, G_c *and* B_c . The next step is to perform a reallocation of weights.

$$\begin{bmatrix} R_e \\ G_e \\ B_e \end{bmatrix} = (1 - JC) \begin{bmatrix} R_c \\ G_c \\ B_c \end{bmatrix} + JC \begin{bmatrix} R_i \\ G_i \\ B_i \end{bmatrix} \quad (8)$$

At last, these images are generated in such a way that, when illuminated with full backlight, they would appear identical to the enhanced images illuminated with dim backlight .The R_s, G_s *and* B_s of the simulated images is computed follows

$$\begin{bmatrix} R_{s,l} \\ G_{s,l} \\ B_{s,l} \end{bmatrix} = \mathbf{M}_f^{-1} \mathbf{M}_l \begin{bmatrix} R_e^{\gamma_{r,l}} \\ G_e^{\gamma_{g,l}} \\ B_e^{\gamma_{b,l}} \end{bmatrix} \quad (9)$$

$$(R_s, G_s, B_s) = \left(R_{s,l}^{\frac{1}{\gamma_{r,f}}}, G_{s,l}^{\frac{1}{\gamma_{g,f}}}, B_{s,l}^{\frac{1}{\gamma_{b,f}}} \right) \quad (10)$$

4. Program

(1) Define the Class for DisplayModel, and define function **rgb_to_xyz** and **xyz_to_rgb**, the rgb_to_xyz is defined according to the formula(1), and the xyz_to_rgb is defined according to the formula(4)(5).

```python
# 定義轉換矩陣
class DisplayModel:
    def __init__(self, gamma, transform_matrix):
        self.gamma_r = gamma[0]
        self.gamma_g = gamma[1]
        self.gamma_b = gamma[2]
        self.transform_matrix = transform_matrix

    def rgb_to_xyz(self, rgb):
        r_lin = rgb[0] ** self.gamma_r
        g_lin = rgb[1] ** self.gamma_g
        b_lin = rgb[2] ** self.gamma_b
        rgb_lin = np.array([r_lin, g_lin, b_lin])
        xyz = np.dot(self.transform_matrix, rgb_lin)
        return xyz

    def xyz_to_rgb(self,xyz):
        rgb_lin = np.clip(np.dot(np.linalg.inv(self.transform_matrix),xyz),0,1)
        r = rgb_lin[0] ** (1/self.gamma_r)
        g = rgb_lin[1] ** (1/self.gamma_g)
        b = rgb_lin[2] ** (1/self.gamma_b)
        rgb = np.array([r,g,b])
        return rgb
```

(2) Define the Class Enhance_image, and it is designed according to the flow chart,note the anchor W_f and W_l are some R=G=B=1, but there are different transform matrices to translate, and the part of Weight, we use normalization to adjust the value of J and C.

```python
class Enhance_image:
    def __init__(self,display_model_full,display_model_low):
        self.display_model_full = display_model_full
        self.display_model_low = display_model_low
    def Device_Characteristic_Modeling(self,rgb):
        """
        第一層轉換 由 RGB-> XYZ
        """
        xyz = display_model_full.rgb_to_xyz(rgb/255.0)
        return xyz
    def CIECAM02_convert(self,xyz,La,Yb,Sr):
        """
        第二層轉換 由 XYZ-> JCh
        """
        wf = self.display_model_full.rgb_to_xyz([1,1,1])
        jch = colour.XYZ_to_CIECAM02(xyz, wf, La, Yb, Sr)
        return jch
    def CIECAM02_invert(self,jch,La,Yb,Sr):
        """
        第三層轉換 由 JCh-> XYZ
        """
        wl = self.display_model_low.rgb_to_xyz([1,1,1])
        xyz = colour.CIECAM02_to_XYZ(jch,wl,La,Yb,Sr)
        return xyz
    def Post_Gamut_Mapping(self,xyz):
        """
        第四層轉換 由 XYZ-> RGB
        """
        rgb = np.clip(self.display_model_low.xyz_to_rgb(xyz),0,1)*255
        return rgb
    def Weight(self,rgb_i,rgb_c,jch):
        """
        權重分配
        """
        weights = (jch[0]/100)*(jch[1]/55.777)
        rgb = (1-weights)*rgb_c + weights*rgb_i
        return rgb
```

(3) Define Class Simulat it only have two step, translate from Low-BlackLight transform metrice and inverse from Full-BlackLight transform metrice.

```python
class Simulate:
    def __init__(self,display_model_full,display_model_low):
        self.display_model_full = display_model_full
        self.display_model_low = display_model_low
    def Low_BlackLight(self,rgb):
        """
        Low BackLight 轉換 由 RGB-> XYZ
        """
        xyz = display_model_low.rgb_to_xyz(rgb/255.0)
        return xyz

    def Full_BlackLight(self,xyz):
        """
        Full BackLight轉換 由 XYZ-> RGB
        """
        rgb = np.clip(self.display_model_full.xyz_to_rgb(xyz),0,1)*255
        return rgb
```

(4) Parameter of formula (1)(2)(4)(5), different from the paper, we set the paremeter of S_r to “Dim”, because we are presentation of sildes, I think it will be in the dim environment.

```python
gamma_f = [2.4767, 2.4286, 2.3792]
gamma_l = [2.2212, 2.1044, 2.1835]

transform_matrix_full = np.array([[95.57, 64.67, 33.01],
                                  [49.49, 137.29, 14.76],
                                  [0.44, 27.21, 169.83]])

transform_matrix_low = np.array([[4.61, 3.35, 1.78],
                                 [2.48, 7.16, 0.79],
                                 [0.28, 1.93, 8.93]])

display_model_full = DisplayModel(gamma_f, transform_matrix_full)
display_model_low = DisplayModel(gamma_l, transform_matrix_low)

La = 63
Yb = 25
Sr = colour.VIEWING_CONDITIONS_CIECAM02["Dim"]
```

(5) Final, we operate on each pixels, at the same time, the enchanced image of the original image is simulated with Low-BlackLight for the final comparison.

```python
input_path = "./images/ninja.jpg"
input_image = np.array(cv2.imread(input_path))

Enhance = Enhance_image(display_model_full, display_model_low)
Simulate = Simulate(display_model_full, display_model_low)
height, width, _ = input_image.shape
output_image = np.zeros_like(input_image, dtype=np.float32)
simulate_image = np.zeros_like(input_image, dtype=np.float32)
simulate_result_image = np.zeros_like(input_image, dtype=np.float32)
for i in range(height):
    for j in range(width):
        xyz = Enhance.Device_Characteristic_Modeling(input_image[i,j])
        jch = Enhance.CIECAM02_convert(xyz, La, Yb, Sr)
        xyz_e = Enhance.CIECAM02_invert(jch, La, Yb, Sr)
        rgb_c = Enhance.Post_Gamut_Mapping(xyz_e)
        rgb_e = Enhance.Weight(input_image[i,j], rgb_c, jch)
        output_image[i,j] = rgb_e

        xyz_s = Simulate.Low_BlackLight(input_image[i,j])
        rgb_s = Simulate.Full_BlackLight(xyz_s)
        simulate_image[i,j] = rgb_s

        xyz_rs = Simulate.Low_BlackLight(rgb_e)
        rgb_rs = Simulate.Full_BlackLight(xyz_rs)
        simulate_result_image[i,j] = rgb_rs

cv2.imwrite(f'{input_path[:-4]}_result.jpg',output_image)
cv2.imwrite(f'{input_path[:-4]}_simulate.jpg',simulate_image)
cv2.imwrite(f'{input_path[:-4]}_result_simulate.jpg',simulate_result_image)
```

5. Experimental result

On the left is the original image, in the middle is directly simulated image, on the right is the enchanced image of the original image it is simulated with Low-BlackLight.

(1)

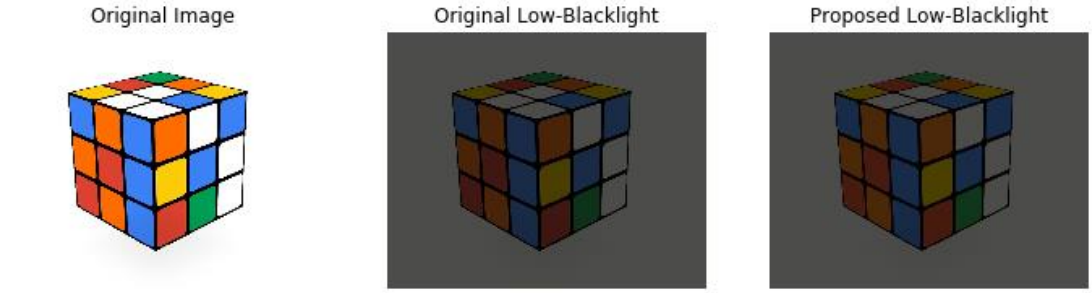
(2)

(3)

(4)

(5)

(6)



(7)

6. Reference

K. T. Shih and H. H. Chen, “Exploiting perceptual anchoring for color image enhancement”, *IEEE Trans. Multimedia*, vol. 18, no. 2, pp. 300-310, Feb. 2016.