Outils Numériques pour l'Ingénieur·e en Physique

2023-2024

6N-076-PHY / ONIP-2

Bloc 4 - Objets / Projet C (100%)

Concepts étudiés

[Phys] Colorimétrie
[Phys] Traitement d'image
[Phys] Utilisation de différents espaces de couleur
[Num] Affichage 2D

Mots clefs

Colorimétrie; ColorChecker; Espaces colorimétriques; Segmentation d'image

Sessions

0 Cours(s) - 1h30
0 TD(s) - 1h30
6 TD(s) Machine - 2h00
0 TP(s) - 4h30

Travail

Par binôme

Institut d'Optique
Graduate School, *France*
https://www.institutoptique.fr

GitHub - Digital Methods

https://github.com/IOGS-Digital-Methods

Calibration d'image numérique

Dans ce projet, on cherche à **calibrer une image à l'aide d'une mire ColorChecker SG**.

Le LEnsE dispose d'une mire ColorChecker SG (Spectral Gloss) fabriquée par Calibrite, c'est un outil de calibration des couleurs utilisé en imagerie numérique. Il se compose d'une série de petits carrés colorés (des *"patches"*) qui représentent des couleurs dont les coordonnées colorimétriques sont connues. Ce type de mire est utilisé par exemple pour les photographies d'œuvre d'art.

D'un point de vue programmation, vous devrez développer ce projet selon les règles de la **programmation orientée objet**.

Aucune fonction ne devra être utilisée en dehors d'un objet.

Acquis d'Apprentissage Visés

En résolvant ce problème, les étudiant·e·s seront capables de :

Côté Numérique

1. **Créer des classes** (Rectangles, ColorChecker, Correction...)
2. **Définir et documenter les méthodes et attributs** de chaque classe
3. **Produire des figures** claires et légendées à partir de signaux numériques - incluant un titre, des axes, des légendes

Côté Physique

1. **Lire des données d'une ColorChecker** à partir d'une image numérique
2. **Lire des données de références d'une ColorChecker** à partir de données constructeur
3. **Corrigé le rendu d'image** à l'aide d'une matrice de correction de couleurs

Livrables attendus

Voir la fiche introductive du module ONIP-2 pour connaître les livrables attendus.

Ressources

Cette séquence est basée sur le langage Python. Vous pouvez utiliser l'environnement **Spyder 5** inclus dans *Anaconda 3*. Des tutoriels Python (et sur les bibliothèques classiques : Numpy, Matplotlib or Scipy) sont disponibles à l'adresse : http://lense.institutoptique.fr/python/.

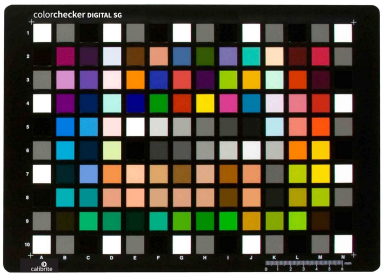
Sujet C - Calibration d'image numérique

Ce projet repose sur l'aide de Mathieu Hebert, en particulier de son ouvrage "Optical Models for Material Appearance", celui-ci étant disponible à la bibliothèque de l'Institut d'Optique.

1 Introduction

On se propose dans ce projet d'analyser une image contenant un ColorChecker SG, extraire les valeurs de couleur de chaque carré de la mire, et utiliser ces informations pour corriger les couleurs de l'image. Les valeurs de références fournies par le constructeur de la ColorChecker SG nous serviront de guide pour cette correction.

Ci-dessous une représentation d'une mire ColorChecker SG. Les patches colorés sont de qualité colorimétrique, c'est-à-dire que leur couleur est très bien calibrée et connue. On note que cette mire possède un aspect *glossy* (*a contrario* d'une mire matte), il faut donc faire attention à ce que les sources ne reflètent de manière spéculaire lors de la prise d'image. Par ailleurs, ce type de mire est particulièrement onéreuse (de l'ordre de 500€), il ne faut donc en **aucun cas toucher les patches colorés. Merci de manipuler la mire avec un grande précaution**.



2 Objectifs

L'objectif est d'extraire les couleurs de chaque patch de la mire à partir d'une image. On va ensuite déterminer numériquement la transformation qu'il faut appliquer aux couleurs de l'image (à l'aide d'une matrice de transfert) pour que les couleurs soient fidèles à celle que l'on observe dans la réalité.

Vous pouvez demander au LEnsE à utiliser la mire pour prendre vos propres photos, mais pour que l'effet de la correction soit perceptible, il faut que les conditions soient assez extrêmes (les smartphones font majoritairement de bonnes photos actuellement).

Ci-dessous, une illustration du résultat qui peut être obtenu.

Image détériorée

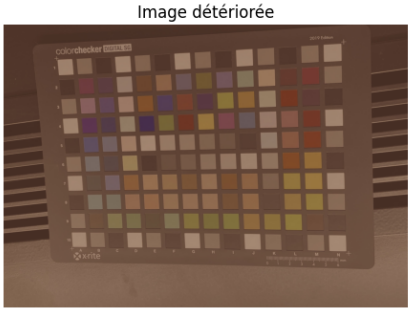
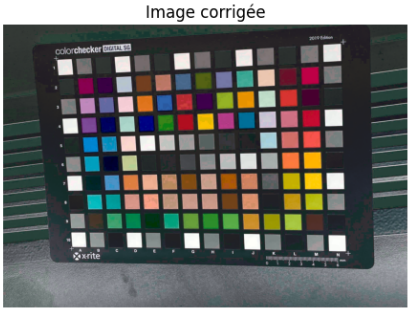


Image corrigée



3 Grandes étapes

- Définir une classe `Couleur` (qui stocke une couleur et permet d'effectuer des conversions entre les différents espaces de couleurs (sRGB, CIELAB, XYZ))
- Définir une classe pour représenter un patch de couleur (avec sa position, son nom et une `Couleur`)
- Définir un système de géométries (point, quadrilatère, rectangle, grille de rectangles...).
- Définir une classe de base pour représenter une ColorChecker générique : `ColorCheckerBaseClass` (qui contiendra l'ensemble des rectangles colorés)
- Définir deux sous classes (qui héritent `ColorCheckerBaseClass`) :
  - Une pour représenter une ColorChecker de référence (en lisant les valeurs Lab constructeur)
  - Une pour représenter une ColorChecker à mesurer (en analysant une image)

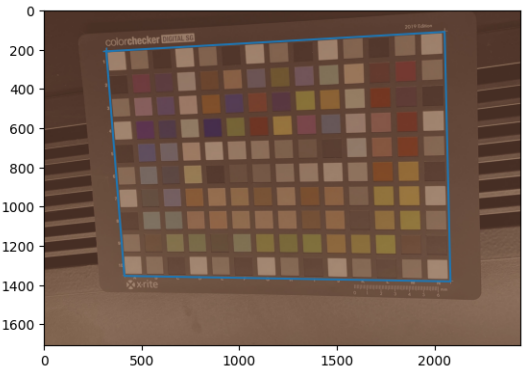
- Définir une classe pour corriger une image à partir de deux instances de ColorChecker (une référence et une de mesure)

Il est recommandé de monitorer visuellement l'avancée de votre projet à chaque étape. Pour cela, vous pouvez implémenter des méthodes (`.plot(self, ax = None)`) dans vos classes.

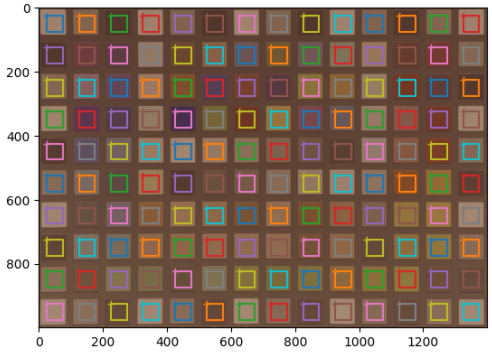
3.1 Focus sur l'extraction des couleurs

L'extraction de la couleur de chaque patch peut être particulièrement fastidieuse. On propose ici une marche à suivre générale pour simplifier le problème.

Lors de la prise d'image, il n'y a aucune raison que la mire soit parfaitement alignée sur la grille de pixels. On peut donc penser transformer la géométrie de l'image pour que la grilles de patches colorés soit alignée sur la grille de pixels : cela simplifiera le processus de d'extraction. Les figures (a) et (b) de la figure 1 montrent un exemple sur une image réelle.



(a) Représentation de la zone d'intérêt définie par les coins de la mire (sur les croix)



(b) Image alignée avec la une grille de pixels. Les zones d'intérêt de chaque patch sont illustrées par des rectangles.

Figure 1. Illustration de la transformation géométrique à appliquer sur l'image pour l'aligner

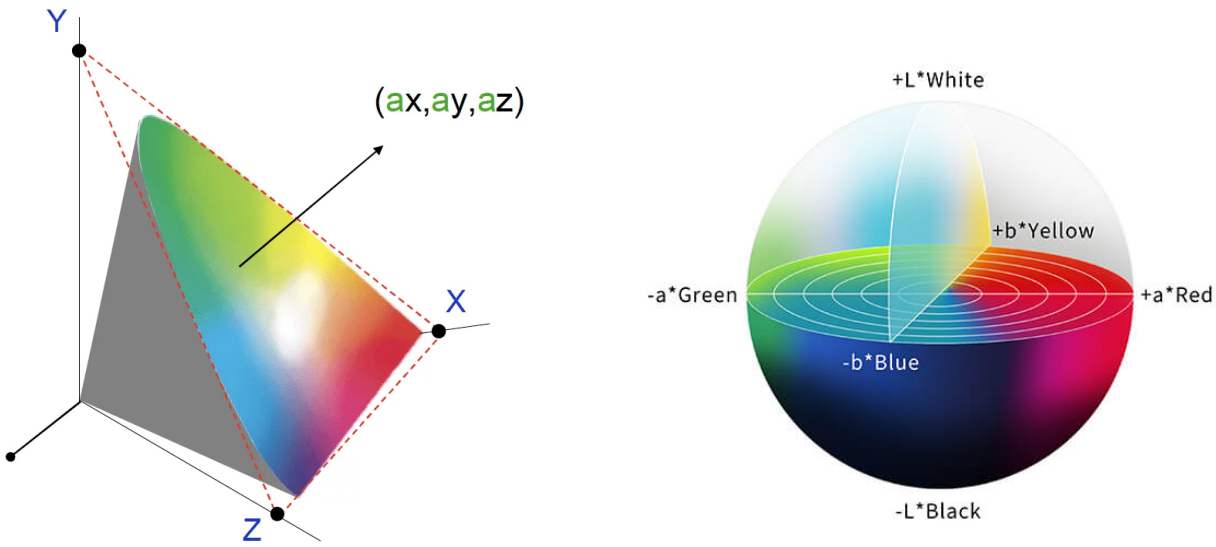
En pratique, on peut utiliser la bibliothèque `open-cv` pour réaliser cette opération, en particulier avec les fonctions `warpPerspective` et `getPerspectiveTransform`. Une fois l'image corrigée de sa perspective, on peut définir une grille de rectangle uniformément repartie. *A fortiori*, il est intéressant d'implémenter une méthode dans les géométries afin de recadrer une image selon un rectangle. La couleur moyenne de cette sous-image peut être déterminée avec `np.mean(cropped_img, axis = 2)`

4 Notions utiles de colorimétrie

4.1 Espaces de couleur

Les espaces colorimétriques sont des systèmes de représentation des couleurs visibles par l'œil humain. Il en existe plusieurs, selon le contexte et le but de la manipulation des couleurs. Dans ce projet, on va se focaliser principalement sur 4 notions importantes :

- **sRGB** : c'est un système de couleurs (défini par la norme Rec.709) pour les écrans, les scanners et les imprimantes. Il est utilisé pour coder la grande majorité des images numérique (au format jpeg).
- **RGB** : c'est un espace basé sur les trois couleurs primaires : rouge, vert et bleu. Il existe une infinité de variantes de RGB, selon le blanc de référence et les coordonnées chromatiques des trois couleurs utilisées comme primaires pour le système. Il existe l'espace CIERGB qui est un espace RGB défini par un point blanc et trois valeurs particulière de couleurs primaires.
  - Il est important de ne pas faire de confusion entre le sRGB et RGB. En particulier, le lien entre les deux n'est pas linéaire. Le sRGB est encodé avec une non-linéarité par rapport aux valeurs de luminances affichées (pour reproduire le comportement de l'œil et optimisé le stockage) que l'on appelle courbe γ.
- **CIELAB** : c'est un espace perceptuel, qui tient compte de la façon dont l'œil humain perçoit les couleurs. Il se compose de trois axes : L pour la luminosité, a pour la teinte entre rouge et vert, et b pour la teinte entre jaune et bleu. Aussi appelé $L^*a^*b^*$ CIE 1976. Voir figure 2b.
- **XYZ** : c'est un espace normalisé par la Commission Internationale de l'Eclairage (CIE), qui englobe toutes les couleurs visibles par l'œil humain. Il se base sur les fonctions colorimétriques $\bar{x}(\lambda)$, $\bar{y}(\lambda)$ et $\bar{z}(\lambda)$, qui mesurent la réponse des cônes de la rétine à différentes longueurs d'onde. Voir figure 2a.



(a) CIE 1931 XYZ **(b)** $L^*a^*b^*$ CIE 1976

Figure 2. Représentations graphiques d'espaces colorimétriques

4.1.1 Conversion d'espaces de couleur

La conversion d'un espace de couleur à un autre repose sur des transformations mathématiques standardisées, ces opérations sont assez pénibles à implémenter (et ce n'est pas l'idée du travail à effectuer). Il existe des bibliothèques Python qui permettent de convertir les espaces de couleurs, on peut recommander :

- `skimage.color` : bien documenté et très simple d'utilisation. La figure 3, illustre les fonctions utiles dans notre cas.
- `colour-science` : Très bien documenté et très complet. Mais il y a énormément de fonctionnalités très avancées qui ne sont pas utiles, on peut vite s'y perdre.
- `open-cv` : Bibliothèque très riche avec une grande communauté d'utilisateurs, mais les documentations est parfois assez abrupte (car adaptée du C++).

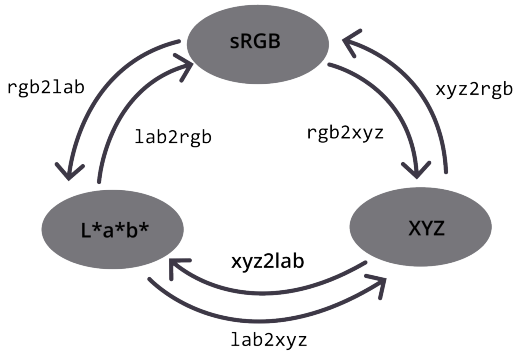


Figure 3. Diagramme des fonctions de conversion disponibles dans `skimage.color`. Notez que pour cette bibliothèque `rgb` représente le sRGB.

4.2 Correction colorimétrique

4.2.1 Calcule de la matrice de correction

Pour chaque patch coloré (ici indexé par $j \in [\![1, N]\!]$) de la mire, on cherche à résoudre l'équation matricielle :

$$\begin{bmatrix} X_j^{\text{ref}} & Y_j^{\text{ref}} & Z_j^{\text{ref}} \end{bmatrix} = w_j^T \cdot U \tag{1}$$

Le vecteur $[X_j^{\text{ref}}\, Y_j^{\text{ref}}\, Z_j^{\text{ref}}]$ représente les coordonnées colorimétriques fournies par le constructeur de la mire dans l'espace CIE 1931 XYZ pour le patch j.

Le vecteur w_j est une combinaison polynômes des valeurs (X_j, Y_j, Z_j) mesurées sur chaque patch de l'image. Selon l'ordre de correction choisi, les valeurs de w_j sont les suivantes :

Ordre 1	$w_j = (1, X_j, Y_j, Z_j)$
Ordre 2	$w_j = (1, X_j, Y_j, Z_j, X_j^2, X_jY_j, X_jZ_j, Y_j^2, Y_jZ_j, Z_j^2)$
Ordre 3	$w_j = (1, X_j, Y_j, Z_j, X_j^2, X_jY_j, X_jZ_j, Y_j^2, Y_jZ_j, Z_j^2,$ $X_j^3, X_j^2Y_j, X_j^2Z_j, X_jY_j^2, X_jY_jZ_j, X_j^2Z_j, Y_j^3, Y_j^2Z_j, Y_jZ_j^2, Z_j^3)$

On nomme k la taille du vecteur w_j (cette valeur dépend dont de l'ordre de correction choisi). La taille de la matrice U sera alors $[k, 3]$.

L'idée est donc de chercher une matrice U qui transforme les valeurs (X_j, Y_j, Z_j) erronées en $(X_j^{\text{ref}}\, Y_j^{\text{ref}}\, Z_j^{\text{ref}})$. L'usage de w_j permet de d'ajouter de la surdétermination dans le système linéaire avec des termes croisés et ainsi de facilité la résolution numérique.

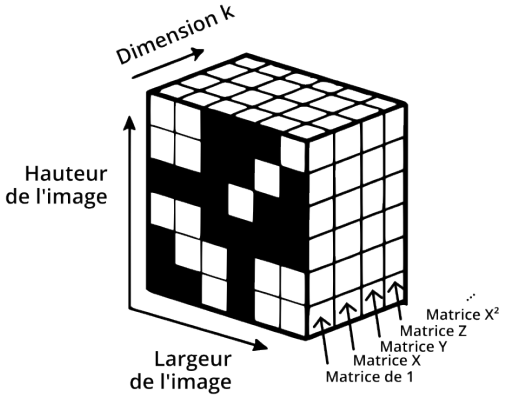
Dans notre cas, il y a $N = 140$ patchs sur la mire. On peut écrire de manière plus condensée le système des 140 équations matricielles 1 de la manière suivante :

$$\begin{bmatrix} X_1^{\text{ref}} & Y_1^{\text{ref}} & Z_1^{\text{ref}} \\ \vdots & \vdots & \vdots \\ X_N^{\text{ref}} & Y_N^{\text{ref}} & Z_N^{\text{ref}} \end{bmatrix} = \begin{bmatrix} w_1^T \\ \vdots \\ w_N^T \end{bmatrix} \cdot U \tag{2}$$

On peut alors trouver la matrice U avec une résolution numérique basée sur la méthode des moindres carrés. La bibliothèque `scipy.linalg` dispose d'une fonction `lstsq` qui peut résoudre l'équation sous la forme de l'équation 2. C'est-à-dire avec une matrice de gauche de dimension $[140, 3]$, une matrice contenant les w_k de taille $[140, k]$ et toujours une matrice U de dimension $[k, 3]$.

4.2.2 Correction de l'image

Après avoir calculé la matrice de correction U, il est possible de corriger le rendu colorimétrique de l'image. On doit disposer de l'image convertie en coordonnées XYZ, cette image est représentée par une matrice de taille $[H, W, 3]$, ou H et W sont respectivement la hauteur et largeur de l'image. L'idée est d'utiliser cette image pour créer une matrice $\tilde{M}_{\text{img}}$ de taille $[H, W, k]$ (où k est la taille du vecteur $w + j$). Sur sa troisième dimension, cette matrice aura la même valeur que le polynôme qui représente w_j. Ci-dessous une image pour illustrer la matrice que l'on veut obtenir. En pratique, pour construire cette matrice avec Python, il peut être intéressant d'utiliser les fonctions `stack`, `concatenate` et `squeeze` de la bibliothèque `numpy`.



Pour appliquer la correction, il est possible de faire le produit matriciel entre $\tilde{M}_{\text{img}}$ et U sans passer par une boucle. En effet, la fonction `dot` de `numpy` gère la matrice à trois dimensions, le résultat de cette opération sera une image corrigée en XYZ de taille $[H, W, 3]$. Pour afficher l'image, il faut finalement convertir l'image en sRGB.

5 Quelques pistes d'ouvertures possibles

Vous devrez choisir et réaliser au moins l'une des ouvertures suivantes dans le cadre de ce projet.

Ouverture A Impact de l'ordre de correction

- Il est possible d'étudier visuellement l'impact de l'ordre choisi sur w_j sur la qualité de correction.

Ouverture B Graphes de modification de la chromaticité xy

- Il est possible de réaliser un graphe dans le diagramme de chromaticité CIE 1931 xy; pour chaque patch on peut tracer une flèche qui illustre la modification de chromaticité entre la référence et l'image, avant et après correction.

Ouverture C Detection automatique de la mire

- Il existe des librairies Python qui permettent de détecter automatiquement les mires type ColorChecker dans une image. Cela évite de devoir renseigner manuellement les coordonnées des 4 coins de la mire. Voir : Site de colour-science pour la détection de ColorCheckers

6 Critères d'évaluation

Grille à simplifier (bilan Semestre 5)

- **METHODES NUMERIQUE**
  - **Ecriture Matricielle / Vectorielle**
    - utilisation des méthodes liées aux vecteurs/matrices (Numpy)
    - aucune boucle **for** inutile
  - **Organisation en actions élémentaires**
    - les étapes sont découpées en fonctionnalité plus simple à tester
  - **Description des tests de validation**
    - chaque fonction a été testée
    - chaque étape a été validée
  - **Organisation des informations à traiter**
    - les données sont rangées dans des objets bien identifiés
- **PROGRAMMATION**
  - **Ecriture globale du code et commentaires (PEP 8)**
    - variables et fonctions respectant les conventions d'écriture standard
    - commentaires utiles
  - **Utilisation, écriture de fonctions**
    - paramètres et retours pertinents des fonctions
  - **Documentation des fonctions (PEP257)**
    - paramètres et retours des fonctions sont documentés
  - Création de classes et d'objets
    - classe contenant des attributs et méthodes pertinents
    - aucune fonction n'est appelée en dehors d'un objet
- **INGENIEUR.E PHYSIQUE**
  - **Graphiques pertinents et légendés**
    - graphiques scientifiques (axes, titre...)
    - axes des graphiques légendés (passage temps/fréquence)
  - **Organisation en actions élémentaires**
    - les étapes sont découpées en fonctionnalité plus simple à tester
  - **Génération de données pertinentes de tests**
    - choix de la position des sources pertinent
  - **Analyse des données et validation modèle**
    - comparaison avec la théorie
    - analyse pertinente des cartes obtenues
- **AVANCEMENT**
  - Étapes 1 et 2 : x 0.5
  - Étapes 1, 2 et 3 : x 0.7
  - Une des ouvertures : x 1.0
