Outils Numériques pour l'Ingénieur·e en Physique

2024-2025

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

"And I'll see your true colors", Cyndi Lauper

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. 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** pour stocker et manipuler des données numériques.
2. **Définir et documenter les méthodes et attributs** de chaque classe
3. **Produire des figures** claires et légendées à partir de signaux numériques (image, couleur) - 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 **Pycharm**. Des tutoriels Python (et sur les bibliothèques classiques : Numpy, Matplotlib or Scipy) sont disponibles à l'adresse : http://lense.institutoptique.fr/python/.

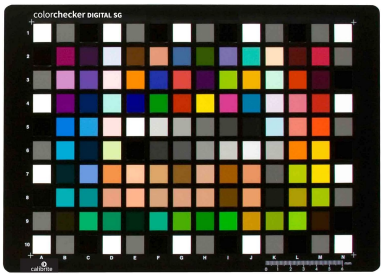
Sujet C - Calibration d'image numérique

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 de corriger des images, via une image de référence contenant ColorChecker SG. Le but est d'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 (via un fichier texte).

Ci-dessous une représentation d'une mire ColorChecker SG. Les patches colorés sont de qualité colorimétrique, c'est-à-dire que leur couleur est 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. Si vous êtes amené à vous en servir, merci de manipuler la mire avec une 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. Une fois cette étape réalisée, on pourra appliquer cette correction à d'autres images prisent dans les mêmes conditions.

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). Nous vous fournissons un catalogue d'image de mire ColorChecker SG pour réaliser ce projet.

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

Image détériorée

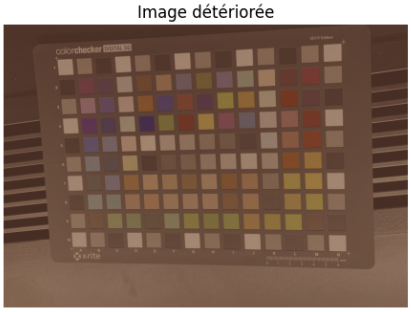
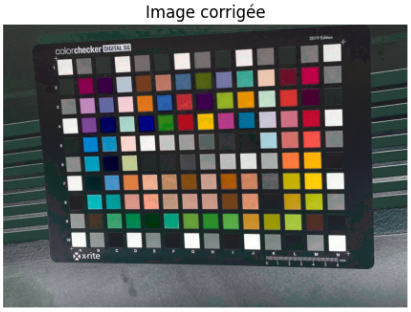


Image corrigée



3 Grandes étapes

- Définir une classe pour représenter les **patchs** de couleur de la mire, celle-ci contiendra les valeurs de couleur ainsi que la position du patch sur la mire. Cette classe pourra être utilisée pour représenter les patchs de la mire de référence et ceux de la mire à corriger.
- Définir une classe pour représenter une **ColorChecker de référence**, celui-ci contiendra les valeurs de couleur de chaque patch en CIELAB, d'après les données fournies par le constructeur.
- Définir une classe pour représenter une ColorChecker réel, pour extraire les couleurs de chaque patch de la mire à partir d'une image.

- Définir une classe pour corriger une image à partir de deux instances de ColorChecker (une référence et une de mesure). Cette classe une fois initialisée doit être capable de corriger une image en utilisant les valeurs de couleur des deux mires.

Il est recommandé de surveiller visuellement l'avancée de votre projet à chaque étape. Pour cela, vous pouvez implémenter des méthodes (`plot`) dans vos classes.

3.1 Code de démarrage

Afin de faciliter la réalisation de ce projet, un code de démarrage est fourni. Il n'est pas obligatoire de l'utiliser, mais cela est recommandé. Celui-ci contient des classes de base, telles que :

- `Color` : classe pour représenter une couleur et effectuer des conversions. Cela peut être utile manipuler les couleurs des patches.
- `PerspectiveRemover` : classe pour retirer la perspective d'une image à partir des quatre coins de la mire. Le but est d'obtenir une image où les patches de la mire sont alignés avec les pixels, ce qui facilitera l'extraction des couleurs.
- `Rectangle` : classe pour représenter un rectangle (avec des méthodes pour le dessiner, et en faire une homothétie, recadrer une image...).

La documentation de ces classes est disponible dans le code de démarrage. Celle-ci est en anglais. Libre à vous de compléter ces classes au besoin.

3.2 Déroulé des séances

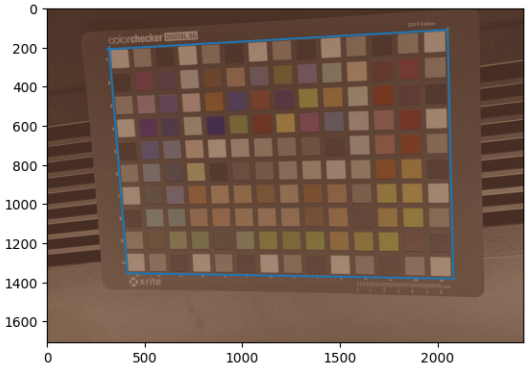
Vous trouverez ici un déroulé indicatif des séances pour vous aider à avancer dans le projet.

- **Séance 1** : Découverte de l'orienté objet dans python.
- **Séance 2** : Découverte des classes de du fichier de démarrage. Réflexion sur la structure du code via un schéma de classes. Validation en fin de séance.
- **Séance 3** : Implémentation et débogage de la classe `ReferenceColorChecker` et `Patch`.
- **Séance 4** : Implémentation et débogage des classes `RealColorChecker` et `ColorCorrection`.
- **Séance 5** : Finalisation du projet. Préparation de la présentation finale.
- **Séance 6** : Évaluation du projet.

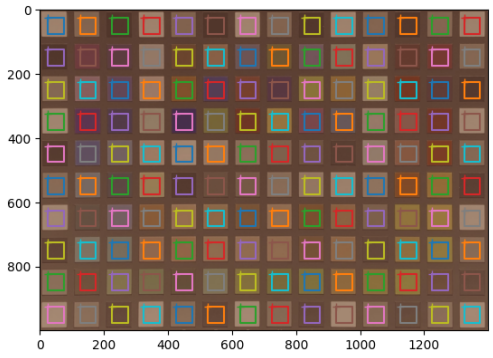
3.3 Focus sur l'extraction des couleurs

L'extraction de la couleur de chaque patch peut être particulièrement fastidieuse. On propose ici une marche à suivre 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 grille colorée 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`. La classe `PerspectiveRemover` fournie dans le code de démarrage permet de réaliser cette opération.

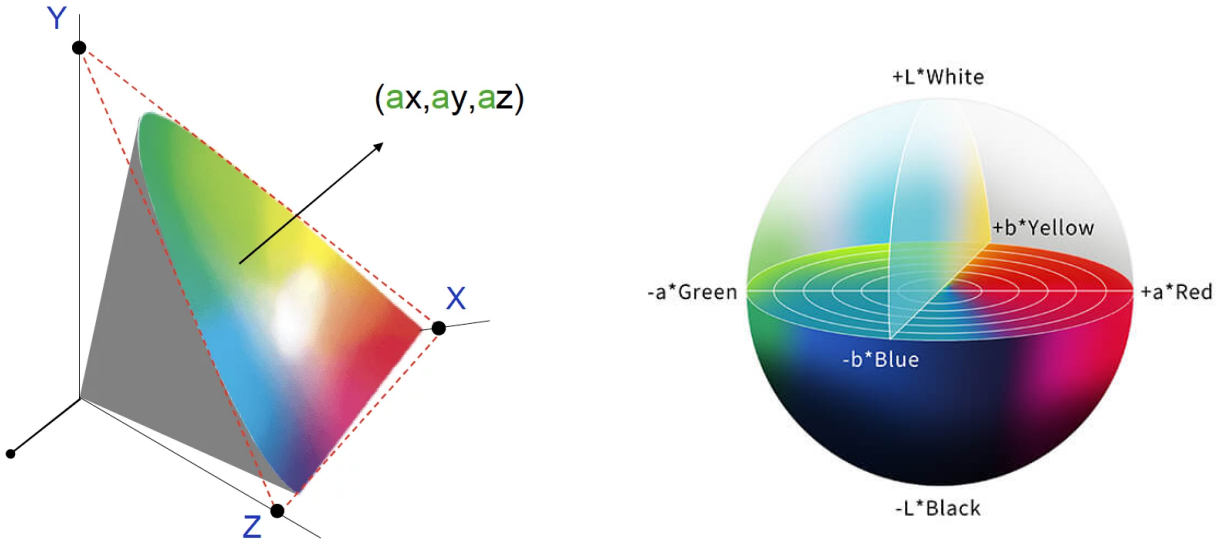
Pour extraire les couleurs, il est possible de sélectionner un pixel au centre de chaque patch. Mais cette méthode est imprécise, car à cause du bruit numérique présent sur l'image, ce pixel ne représente pas forcément la couleur du patch. Pour améliorer la précision, on peut réaliser une moyenne des couleurs sur une zone autour du pixel central. La taille de cette zone est à définir, mais elle doit être suffisamment grande pour contenir une grande partie du patch, mais pas trop grande pour ne pas inclure les couleurs des patches voisins.

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). Dans le code, on utilisera souvent par abus de langage le terme `rgb` pour désigner cet espace.
- **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, pour ce projet on se propose d'utiliser `skimage.color`. La classe `Color` fournie dans le code de démarrage permet de réaliser ces conversions sur un pixel. Pour convertir une image entière, il est possible d'utiliser directement les fonctions adéquates de `skimage.color` sur une matrice de taille $[h, w, 3]$.

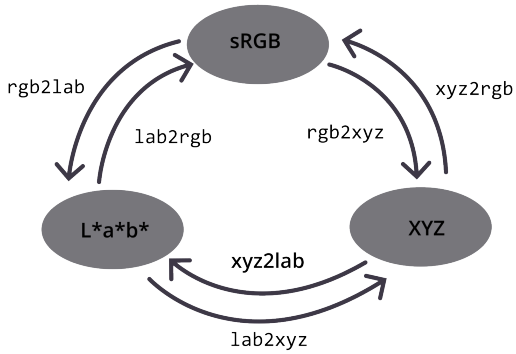


Figure 3. Diagramme des fonctions de conversion disponibles dans `skimage.color`. Notez que pour cette bibliothèque `rgb` représente le sRGB.

4.2 Correction colorimétrique

Les calculs de correction colorimétrique sont basés sur des calculs matriciels, il est fortement recommandé de construire manupiler les matrices avec `NumPy` **matricielle (sans utiliser de boucle)**. Cela rend le code plus efficient et plus lisible. En pratique, pour construire la plupart des matrices utiles pour ce projet, il est intéressant d'utiliser les fonctions `stack`, `concatenate` et `squeeze` de la bibliothèque `numpy`.

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 :

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

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 (converties dans l'espace XYZ) pour le patch j. On se place dans l'espace XYZ pour effectuer les calculs, les espaces sRGB et CIELAB sont moins adaptés pour les calculs de correction (car ils présentent des non-linéarités).

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

Ordre 1	$w_j = (1, X_j, Y_j, X_j)$
Ordre 2	$w_j = \left(1, X_j, Y_j, X_j, {X_j}^2, X_jY_j, X_jX_j, {Y_j}^2, Y_jX_j, {X_j}^2\right)$
Ordre 3	$w_j = \begin{array}{l}(1, X_j, Y_j, X_j,\\ {X_j}^2, X_jY_j, X_jX_j, {Y_j}^2, Y_jX_j, {X_j}^2,\\ {X_j}^3, {X_j}^2Y_j, {X_j}^2X_j, X_j{Y_j}^2, X_jY_jX_j, {X_j}^2X_j, {Y_j}^3, {Y_j}^2X_j, Y_j{X_j}^2, {X_j}^3)\end{array}$

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 :

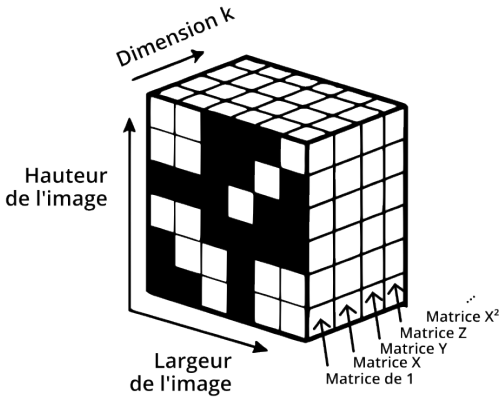
$$\begin{bmatrix} X_1^{\text{ref}} & Y_1^{\text{ref}} & Z_1^{\text{ref}} \\ \vdots & \vdots & \vdots \\ X_N^{\text{ref}} & Y_N^{\text{ref}} & Z_N^{\text{ref}} \end{bmatrix} = \begin{bmatrix} w_1 \\ \vdots \\ w_N \end{bmatrix} \cdot U \tag{2}$$

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 $[k, 3]$.

Il est recommandé de commencer l'inversion de la matrice U avec l'ordre 1, puis de tester les ordres supérieurs pour voir si la correction est améliorée. (cf. partie 5 : Ouverture A).

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.



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 Traitement en *batch* d'images

- À partir de l'exploitation d'une image avec un mire, il est possible de réaliser un traitement en batch d'un ensemble d'images, à la condition que les images soient prises dans les mêmes conditions.

Ouverture C 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. Une illustration est donnée en figure 4.

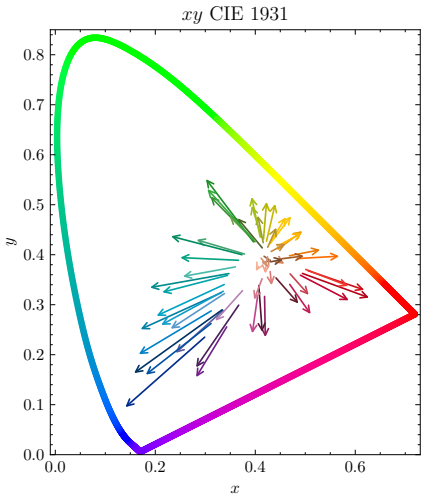


Figure 4. Diagramme de chromaticité CIE 1931 xy, le debut de la flèche correspond à la couleur mesurée et la fin à la couleur de référence.

Ouverture D 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

---