TP2

Partie 1 : Bases des Espaces de Couleurs – Théorie & Exercices Pratiques

1 Présentation Rapide des Espaces de Couleurs

Dans un premier temps, on rapelle que :
- RGB/BGR : OpenCV charge les images au format BGR (Bleu, Vert, Rouge).
- HSV : Sépare la Teinte (Hue), la Saturation et la Valeur (Value, ou luminosité). Un espace utile pour la segmentation par couleur.
- CIELab : Conçu pour être perceptuellement uniforme. Les distances entre deux couleurs dans cet espace correspondent mieux aux différences perçues par l'œil humain.

1.2 Exercice 1 : Conversion et Visualisation des Espaces de Couleurs

Consignes:

1. Chargez une image (format jpg ou png) au format BGR à l'aide d'OpenCV.
2. Convertissez l'image en deux autres espaces de couleur : HSV et CIELab.
3. Affichez, côte à côte, l'image originale (en RGB pour l'affichage avec matplotlib), l'image convertie en HSV et l'image en CIELab.
4. Commentez les résultats obtenus pour chaque espace de couleur.

5. On va maintenant manipuler l'espace de couleur HSV pour modifier certains paramètres visuels d'une image, ici la saturation, et observer le résultat de cette modification après conversion en RGB pour un affichage correct avec matplotlib.

- Dans cet exercice, nous allons réduire la saturation de l'image. La saturation correspond à l'intensité de la couleur. Une saturation plus faible donnera des couleurs plus atténuées, voire proches du gris.

Étapes du code :

a) Copie de l'image en espace HSV :

- On crée une copie de l'image convertie en HSV dans la variable img_hsv_mod pour ne pas altérer l'image originale.

b) Diminution de la saturation :

- On diminue la composante S (saturation) de 90 unités en utilisant la fonction cv2.subtract.

- Attention : il faut s'assurer que la nouvelle valeur ne soit pas négative (cv2.subtract gère ce cas en limitant à zéro).

  Code :
  img_hsv_mod[:, :, 1] = cv2.subtract(img_hsv_mod[:, :, 1], 90)

c) Reconversion de l'image en BGR puis RGB

d) Affichage avec matplotlib :

Exercice 2 : Manipulation des Composantes d'un Espace de Couleurs

Consignes

1. Utilisez l'image convertie en HSV.
2. Isolez la composante Hue en extrayant le premier canal de l'image HSV.
3. Affichez cette composante en niveaux de gris.
4. Expérimentez en modifiant la valeur de saturation globale (dans tout l'image HSV), puis reconvertissez l'image modifiée vers l'espace BGR et affichez le résultat.
5. Comparez l'image d'origine et l'image après modification de la saturation. Discutez de l'influence de la saturation sur la perception des couleurs.

Questions

- Comment la composante Hue représente-t-elle la teinte des pixels ?
- Quelle est l'influence de la saturation sur l'intensité et la vivacité des couleurs ?
- Peut-on imaginer des applications pratiques de la modification de la saturation (exemples : amélioration de certaines images, effets artistiques, etc.) ?

Partie 2 : Exercice – Classification de Couleur d'un T-shirt via Distance CIELab

Contexte

Vous disposerez d'images de t-shirts et vous devez automatiser leur classification en fonction de leur couleur. On considérera 5 couleurs différentes (par exemple : rouge, bleu, vert, jaune, noir). Pour ce faire, on utilisera la distance entre la couleur moyenne d'un t-shirt en espace CIELab et des couleurs de référence.

Étapes de l'Exercice

1. **Extraction de la région correspondant au t-shirt**
   On peut fournir aux étudiants une image simple où le t-shirt est clairement visible. La région peut être délimitée manuellement (en utilisant une ROI via cv2.selectROI) ou en utilisant un masque pré-défini.
2. **Calcul de la couleur moyenne du t-shirt en CIELab**
   Convertir la région extraite en CIELab puis calculer la moyenne des canaux a et b (attention, L pourrait être utile selon le contexte, mais souvent on se focalise sur a et b pour la teinte).
3. **Définition des couleurs de référence**
   Proposez aux étudiants de définir, en CIELab, 5 vecteurs représentant le rouge, bleu, vert, jaune et noir. Vous pouvez donner des valeurs approximatives. Par exemple :

```
ref_colors = {
   "rouge": [135, 208, 195],
   "bleu": [82, 207, 21],
   "vert": [222, 42, 211],
  "jaune": [247, 107, 222],
   "noir": [26, 128, 128]
}
```

4. **Calcul de la distance entre la couleur moyenne et chaque couleur de référence**
   Utilisez par exemple la distance Euclidienne en espace a, b ou en L, a, b.
5. **Attribution de la classe couleur**
   La couleur de référence ayant la plus petite distance à la couleur moyenne sera la classe attribuée au t-shirt.

Exemple de code pour calculer la distance et affecter la couleur :

```
python

# Exemple d'une fonction pour calculer la distance euclidienne
def euclidean_distance(color1, color2):
    return np.linalg.norm(np.array(color1) - np.array(color2))
```

Partie 3 : Exercice – Quantification d'une Image par Uniformisation des Valeurs

Objectif

La quantification dans ce contexte consiste à réduire le nombre de niveaux de gris ou de couleur disponibles par canal afin de simplifier l'image tout en conservant une représentation acceptable à l'œil. Par exemple, vous pouvez réduire chacun des 3 canaux (R, G, B) de 256 niveaux à seulement 16 ou 32 niveaux.

Consignes

1. Chargez une image en couleur.
2. Choisissez un nombre de niveaux désiré par canal (par exemple, 16 niveaux).

3. Transformez l'image en utilisant une quantification uniforme : pour chaque canal, chaque pixel sera mappé sur l'un des N niveaux disponibles.
4. Affichez côte à côte l'image originale et l'image quantifiée.
5. Réfléchissez à l'impact visuel de cette quantification.

Explications

- Pour quantifier un canal, on peut appliquer la formule suivante :
  quantized_value = floor(original_value / (256 / N)) * (256 / N)
  où N est le nombre de niveaux souhaités.
- Cette méthode réduit directement le nombre de valeurs possibles par pixel sans utiliser d'algorithme de clustering.