

Université de Toulon

IUT de Toulon

Département Génie Electrique et Informatique Industrielle (GEII)

# Électronique Spécialisé

## OpenCV

écrit le 14 janvier 2025

par

Bruno HANNA

*Encadrant universitaire :* Valentin GIES

---



# Table des matières

<b>Introduction</b>	<b>1</b>
<b>1 Programme Principal</b>	<b>2</b>
1.1 Définition des Plages de Couleurs et Création des Masques . . . . .	2
1.1.1 Définition des Plages de Couleurs . . . . .	2
1.1.2 Création des Masques de Couleurs . . . . .	2
1.1.3 Création d'une Image Verte pour l'Application des Masques . . . . .	3
1.2 Définition des Noyaux et Itérations pour l'Érosion et la Dilatation . . . . .	3
1.2.1 Combinaison des Masques Nettoyés . . . . .	4
1.2.2 Conversion en Image Binaire et Détection des Contours . . . . .	4
1.2.3 Annotation des Contours sur l'Image Originale . . . . .	5
1.2.4 Redimensionnement et Affichage de l'Image Finale . . . . .	5
<b>2 Programme Bibliothèque</b>	<b>6</b>
2.1 Module <code>bruno.py</code> . . . . .	6
2.1.1 Fonction <code>erosion_dilatation</code> . . . . .	6
2.2 Fonction <code>detect_contours</code> . . . . .	6
2.2.1 Initialisation des Compteurs et Variables de Suivi . . . . .	6
2.2.2 Boucle de Traitement des Contours . . . . .	7
2.2.3 Annotation des Contours et Ajout des Informations Visuelles . . . . .	7
2.2.4 Identification du Plus Petit Fruit . . . . .	8
2.2.5 Annotation du Plus Petit Fruit . . . . .	8
2.2.6 Affichage des Comptages des Fruits . . . . .	9
2.2.7 Affichage des Résultats dans la Console . . . . .	9
<b>Conclusion</b>	<b>10</b>

# Introduction

Ce document présente une analyse détaillée d'un programme Python conçu pour détecter et identifier différents fruits dans une image en utilisant la bibliothèque OpenCV. Le programme traite l'image, applique des filtres de couleur, détecte les contours des fruits, et les annoter avec des informations pertinentes telles que le type de fruit et sa surface. De plus, il identifie le fruit de plus petite taille dans l'image.

# Programme Principal

## 1.1 Définition des Plages de Couleurs et Création des Masques

Pour identifier les différents fruits, le programme définit des plages de couleurs spécifiques en HSV et crée des masques binaires pour chaque type de fruit.

### 1.1.1 Définition des Plages de Couleurs

```
1 lower_nectarine1 = np.array([0, 25, 25])
2 upper_nectarine1 = np.array([10, 255, 255])
3 lower_nectarine2 = np.array([170, 25, 25])
4 upper_nectarine2 = np.array([180, 255, 255])
5
6 lower_jaune = np.array([10, 100, 100])
7 upper_jaune = np.array([23, 255, 255])
8
9 lower_vert = np.array([30, 0, 50])
10 upper_vert = np.array([80, 255, 160])
```

Listing 1.1 – Définition des plages de couleurs pour les nectarines, jaunes et verts

- **Nectarines** : Deux plages de teintes sont définies pour gérer l'enroulement des valeurs de teinte en HSV.
- **Jaunes** : Plage de teinte pour détecter les fruits jaunes.
- **Verts** : Plage de teinte pour détecter les fruits verts.

### 1.1.2 Création des Masques de Couleurs

```
1 maskjaune = cv2.inRange(imagehsv, lower_jaune, upper_jaune)
2 maskvert = cv2.inRange(imagehsv, lower_vert, upper_vert)
3 mask1 = cv2.inRange(imagehsv, lower_nectarine1, upper_nectarine1)
4 mask2 = cv2.inRange(imagehsv, lower_nectarine2, upper_nectarine2)
5 masknectarine = cv2.bitwise_or(mask1, mask2)
```

Listing 1.2 – Création des masques binaires pour chaque couleur

`cv2.inRange` crée des masques binaires où les pixels dans la plage spécifiée sont blancs (255) et les autres sont noirs (0). Pour les nectarines, les deux masques `mask1` et `mask2` sont combinés avec `cv2.bitwise_or`.

### 1.1.3 Création d'une Image Verte pour l'Application des Masques

```

1 full_green = np.zeros_like(img)
2 full_green[:] = [0, 255, 0]
3 vert_img = cv2.bitwise_and(full_green, full_green, mask=maskvert)
4 jaune_img = cv2.bitwise_and(full_green, full_green, mask=maskjaune)

```

Listing 1.3 – Création d'une image entièrement verte

Ces lignes créent une image verte pure et appliquent les masques vert et jaune pour isoler les régions d'intérêt.

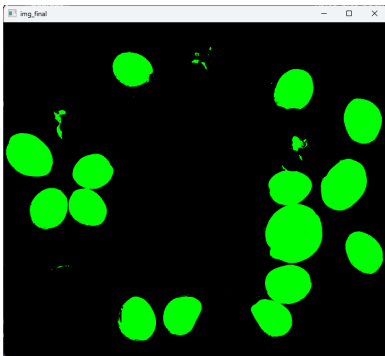


Figure 1.1 – Masque Jaune

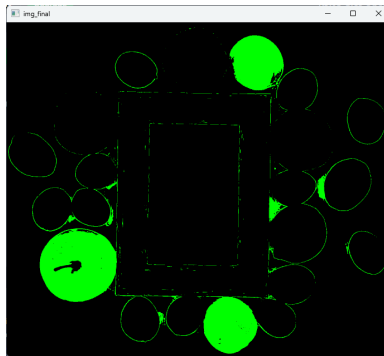


Figure 1.2 – Masque Vert

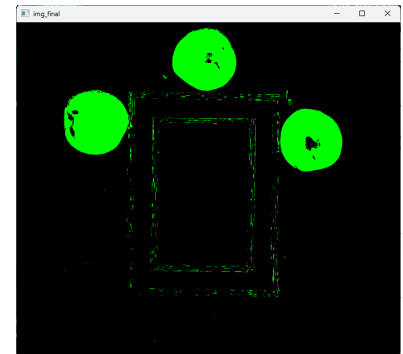


Figure 1.3 – Masque Nectarine

## 1.2 Définition des Noyaux et Itérations pour l'Érosion et la Dilatation

```

1 kernel_nectarine1 = [3,3]
2 kernel_nectarine2 = [3,3]
3 iteration_nectarine = [1,1]
4 mask_nectarine = erosion_dilatation(masknectarine, kernel_nectarine1,
    kernel_nectarine2, iteration_nectarine)
5
6 kernel_jaune1 = [9,9]
7 kernel_jaune2 = [7,9]
8 iteration_jaune = [5,5]
9 mask_jaune = erosion_dilatation(maskjaune, kernel_jaune1, kernel_jaune2,
    iteration_jaune)

```

Listing 1.4 – Définition des noyaux et itérations pour les différentes couleurs

Pour chaque type de masque (nectarine, vert (identique à celui de la nectarine) et jaune), des noyaux de taille spécifique et des itérations sont définis et appliqués via la fonction ero-

sion\_dilatation. Pour la couleur jaune, un ajustement sera fait de la matrice afin d'allonger les formes. Cela permettra d'éviter la fusion des abricots entre eux ainsi qu'avec le citron.

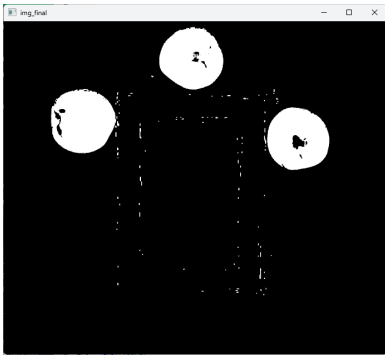


Figure 1.4 – Érosion Dilatation Jaune

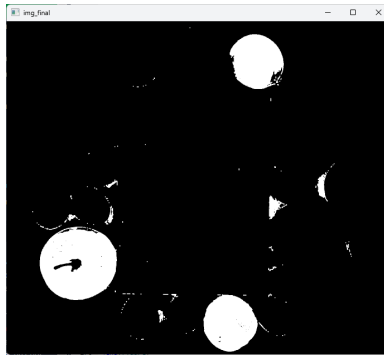


Figure 1.5 – Érosion Dilatation Vert

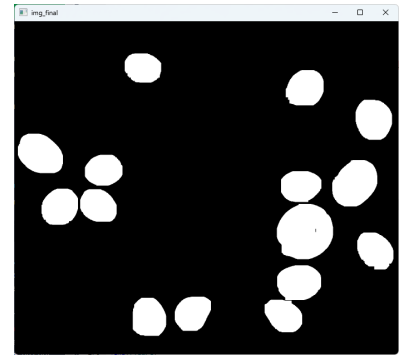


Figure 1.6 – Érosion Dilatation Nectarine

### 1.2.1 Combinaison des Masques Nettoyés

```
1 mask_jaunevert = cv2.bitwise_or(mask_vert, mask_jaune)
2 mask_final = cv2.bitwise_or(mask_jaunevert, mask_nectarine)
```

Listing 1.5 – Combinaison des différents masques

Les masques nettoyés pour le vert et le jaune sont combinés, puis fusionnés avec le masque nectarine pour obtenir un masque final représentant tous les fruits détectés.

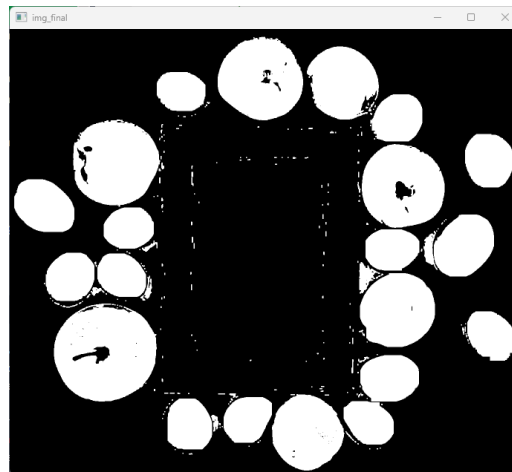


Figure 1.7 – Masque Final

### 1.2.2 Conversion en Image Binaire et Détection des Contours

```
1 _, img_gray = cv2.threshold(mask_final, 1, 255, cv2.THRESH_BINARY)
```

```
2 contours, _ = cv2.findContours(img_gray, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

Listing 1.6 – Seuil binaire et détection des contours

`cv2.threshold` convertit le masque final en une image binaire, facilitant la détection des contours avec `cv2.findContours`.

### 1.2.3 Annotation des Contours sur l'Image Originale

```
1 image_final = detect_contours(img, contours)
```

Listing 1.7 – Détection et annotation des contours

La fonction `detect_contours`, définie dans le module `bruno.py`, analyse chaque contour détecté, identifie le type de fruit, et ajoute des annotations visuelles telles que des cercles et du texte.

### 1.2.4 Redimensionnement et Affichage de l'Image Finale

```
1 img_final = cv2.resize(image_final, dim, interpolation=cv2.INTER_AREA)
2 cv2.imshow('img_final', img_final)
3 cv2.waitKey(0)
```

Listing 1.8 – Redimensionnement et affichage de l'image finale

L'image annotée est redimensionnée (car sur mon pc elle apparaissait trop grande) et affichée dans une fenêtre.



Figure 1.8 – Image Finale

# Programme Bibliothèque

## 2.1 Module `bruno.py`

Le module `bruno.py` contient des fonctions utilitaires utilisées dans le script principal pour effectuer des opérations morphologiques et annoter les contours.

### 2.1.1 Fonction `erosion_dilatation`

```
1 def erosion_dilatation(img, kernel1, kernel2, iteration):
2     kernel = np.ones((kernel1[0], kernel1[1]), np.uint8)
3     img_erosion = cv2.erode(img, kernel, iterations=iteration[0])
4     kernel = np.ones((kernel2[0], kernel2[1]), np.uint8)
5     img_dilation = cv2.dilate(img_erosion, kernel, iterations=iteration
6                               [1])
7     cv2.waitKey(0)
8     return img_dilation
```

Listing 2.1 – Fonction d'érosion et de dilatation

— **Paramètres :**

- `img` : Image binaire à traiter.
- `kernel1, kernel2` : Tailles des noyaux pour l'érosion et la dilatation.
- `iteration` : Nombre d'itérations pour chaque opération.

- **Description :** Applique une érosion suivie d'une dilatation sur l'image binaire, ce qui aide à éliminer le bruit et à combler les trous dans les masques.

## 2.2 Fonction `detect_contours`

La fonction `detect_contours` traite les contours détectés dans une image binaire, identifie les types de fruits basés sur l'aire des contours, et ajoute des annotations telles que des cercles et des textes pour visualiser les résultats.

### 2.2.1 Initialisation des Compteurs et Variables de Suivi

Après la définition de la fonction, plusieurs variables sont initialisées pour suivre le nombre de chaque type de fruit détecté et pour identifier le plus petit fruit dans l'image.



```
1 fruit_counts = {
2     "abricot": 0,
3     "citron vert": 0,
4     "citron": 0,
5     "pomme": 0,
6     "nectarine": 0
7 }
8
9 smallest_area = float('inf') # Surface du plus petit fruit
10 smallest_contour = None      # Contour du plus petit fruit
11 smallest_center = (0, 0)     # Coordonnées du plus petit fruit
```

`fruit_counts` est un dictionnaire qui maintient un compteur pour chaque type de fruit identifié. Les variables `smallest_area`, `smallest_contour`, et `smallest_center` sont utilisées pour déterminer et stocker les informations du fruit ayant la plus petite aire dans l'image.

### 2.2.2 Boucle de Traitement des Contours

La fonction parcourt ensuite chaque contour détecté pour déterminer le type de fruit et mettre à jour les compteurs correspondants.

```
1 for contour in contours:
2     area = cv2.contourArea(contour)
3     if 100 < area < 100000: # Filtrage par aire
4         fruit_props = type_fruits(area)
5         if fruit_props is not None:
6             nom = fruit_props["nom"]
7             text_color = fruit_props["text_color"]
8             circle_color = fruit_props["circle_color"]
9
10            # Mettre à jour le compteur du fruit détecté
11            fruit_counts[nom] += 1
```

Pour chaque contour, la fonction calcule son aire à l'aide de `cv2.contourArea`. Seuls les contours dont l'aire se situe entre 100 et 100000 pixels carrés sont considérés, ce qui permet de filtrer les petits bruits et les contours trop grands. La fonction `type_fruits` est ensuite appelée pour déterminer le type de fruit basé sur l'aire. Si un type de fruit est identifié, le compteur correspondant dans `fruit_counts` est incrémenté.

### 2.2.3 Annotation des Contours et Ajout des Informations Visuelles

Une fois le type de fruit identifié, la fonction procède à l'annotation des contours et à l'ajout de cercles et de textes sur l'image de sortie.

```

1 cv2.drawContours(output_img, [contour], -1, (200, 200, 200), 3)
2
3 (x, y), radius = cv2.minEnclosingCircle(contour)
4 center = (int(x), int(y))
5 radius = int(radius)
6 cv2.circle(output_img, center, radius, circle_color, 2)
7
8 cv2.putText(output_img, f"{nom}", (center[0] - 50, center[1] - 10),
9             cv2.FONT_HERSHEY_SIMPLEX, 0.8, text_color, 2)
10 cv2.putText(output_img, f"{int(area)}px2", (center[0] - 60, center[1] +
11         30),
12             cv2.FONT_HERSHEY_SIMPLEX, 0.8, text_color, 2)

```

`cv2.drawContours` dessine le contour détecté sur `output_img` en gris avec une épaisseur de 3 pixels. Ensuite, `cv2.minEnclosingCircle` est utilisé pour approximer un cercle englobant le contour, dont le centre et le rayon sont calculés et dessinés avec `cv2.circle`. Les informations textuelles, telles que le nom du fruit et son aire, sont ajoutées à l'image à l'aide de `cv2.putText`, positionnées relativement au centre du cercle englobant.

### 2.2.4 Identification du Plus Petit Fruit

La fonction met également à jour les informations concernant le plus petit fruit détecté dans l'image.

```

1 if area < smallest_area:
2     smallest_area = area
3     smallest_contour = contour
4     smallest_center = center

```

Si l'aire du contour actuel est inférieure à `smallest_area`, les variables `smallest_area`, `smallest_contour`, et `smallest_center` sont mises à jour pour refléter ce nouveau plus petit fruit.

### 2.2.5 Annotation du Plus Petit Fruit

Après avoir parcouru tous les contours, la fonction vérifie si un plus petit fruit a été identifié et l'annote en conséquence.

```

1 if smallest_contour is not None:
2     cv2.drawContours(output_img, [smallest_contour], -1, (0, 255, 0), 3)
3     # Vert fluo
4     cv2.putText(output_img, f"Coord. petit fruit: Y={smallest_center[0]},
5         X={-smallest_center[1]}",
6         (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

```

Si un contour a été identifié comme le plus petit, celui-ci est dessiné en vert fluo sur l'image de sortie, et ses coordonnées sont affichées en haut à gauche de l'image.

### 2.2.6 Affichage des Comptages des Fruits

La fonction ajoute également un résumé des comptages des différents types de fruits détectés à la partie inférieure de l'image.

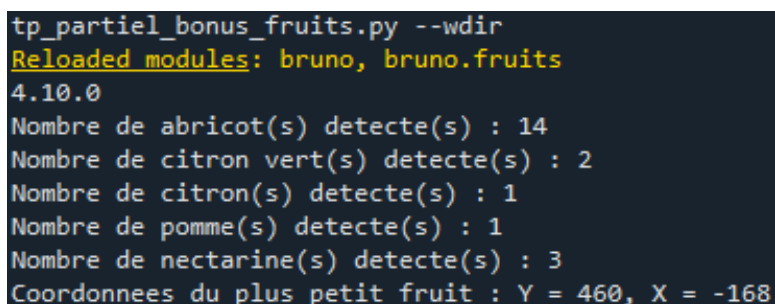
```
1 decal_bas = 50
2 for fruit, count in fruit_counts.items():
3     if count > 0:
4         text = f"{fruit}(s) detecte(s) : {count}"
5         cv2.putText(output_img, text, (500, output_img.shape[0] - (420 +
6             decal_bas)),
7             cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 255), 2)
8         decal_bas += 50
```

### 2.2.7 Affichage des Résultats dans la Console

Enfin, la fonction imprime les résultats des détections dans la console.

```
1 for fruit, count in fruit_counts.items():
2     print(f"Nombre de {fruit}(s) detecte(s) : {count}")
3 if smallest_contour is not None:
4     print(f"Coordonnees du plus petit fruit : Y = {smallest_center[0]}, X
5         = {-smallest_center[1]}")
```

Les comptages des fruits sont affichés dans la console, ainsi que les coordonnées du plus petit fruit si un tel fruit a été identifié.



```
tp_partiel_bonus_fruits.py --wdir
Reloaded modules: bruno, bruno.fruits
4.10.0
Nombre de abricot(s) detecte(s) : 14
Nombre de citron vert(s) detecte(s) : 2
Nombre de citron(s) detecte(s) : 1
Nombre de pomme(s) detecte(s) : 1
Nombre de nectarine(s) detecte(s) : 3
Coordonnees du plus petit fruit : Y = 460, X = -168
```

Figure 2.1 – Affichage Console

# Conclusion

Ce programme illustre l'utilisation efficace de Python et OpenCV pour le traitement et l'analyse d'images. En combinant des techniques de filtrage de couleurs, d'opérations morphologiques, et de détection de contours, il permet d'identifier et de classier différents fruits dans une image.