07 DECEMBRE 2023

RAPPORT APPLICATION GRAPHIQE

GREGORY WITTMANN

IUT Saint-Dié
UNIVERSITÉ DE LORRAINE

Table des matières

INTRODUCTION

Le présent rapport documente le développement d'une application Python axée sur le traitement d'image et la reconnaissance d'image, en utilisant la bibliothèque OpenCV. Ce projet vise à exploiter les fonctionnalités puissantes d'OpenCV pour créer une interface graphique interactive permettant d'effectuer diverses opérations de traitement d'image.

L'objectif principal de cette application est de démontrer la polyvalence d'OpenCV dans le domaine du traitement d'image, tout en offrant une expérience utilisateur intuitive à travers une interface graphique. Nous explorerons divers aspects de la bibliothèque, mettant l'accent sur des fonctionnalités telles que la capture d'image, le filtrage, la détection d'objets, et la reconnaissance d'image.

Ce rapport détaillera le développement et de mise en œuvre de l'application. Des captures d'écran seront fournies pour illustrer les différentes étapes du projet.

Détection par couleur

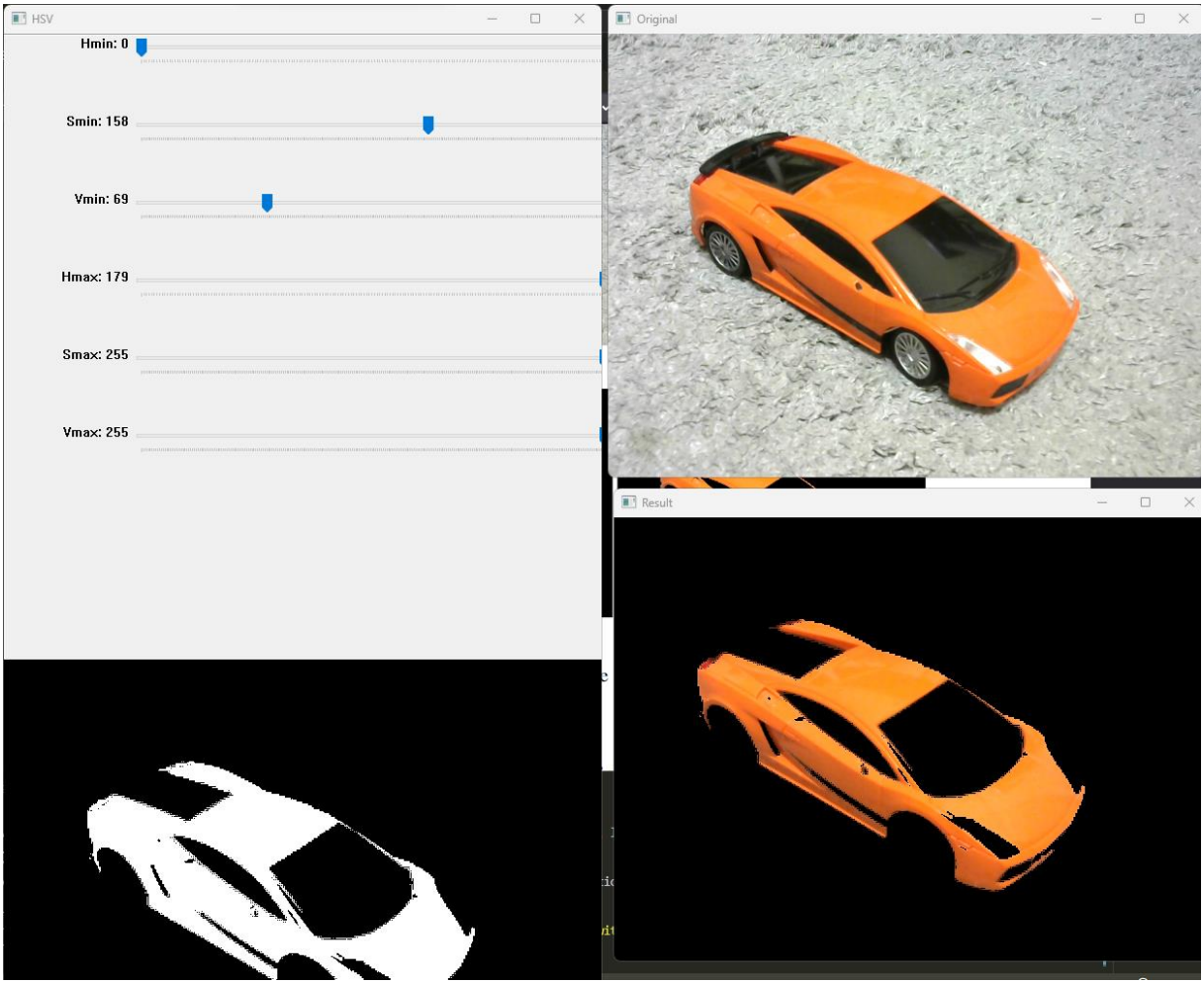
loadImage : Charge une image à partir du fichier 'car.png' en utilisant la fonction `cv.imread`.

convertToHSV : Convertit l'image en espace de couleur HSV en utilisant `cv.cvtColor`. Affiche ensuite l'image HSV dans une fenêtre.

maskHSV : Convertit l'image en espace de couleur HSV, crée une fenêtre avec des trackbars pour ajuster les valeurs HSV et génère un masque HSV interactif. Applique ensuite le masque à l'image de base et affiche l'image originale, le masque et le résultat de l'application du masque.

if __name__ == "__main__": : Appelle la fonction `maskHSV` avec une image chargée à partir de la fonction `loadImage` lorsque le script est exécuté directement.

Capture d'écran :



Détection d’objets géométrique

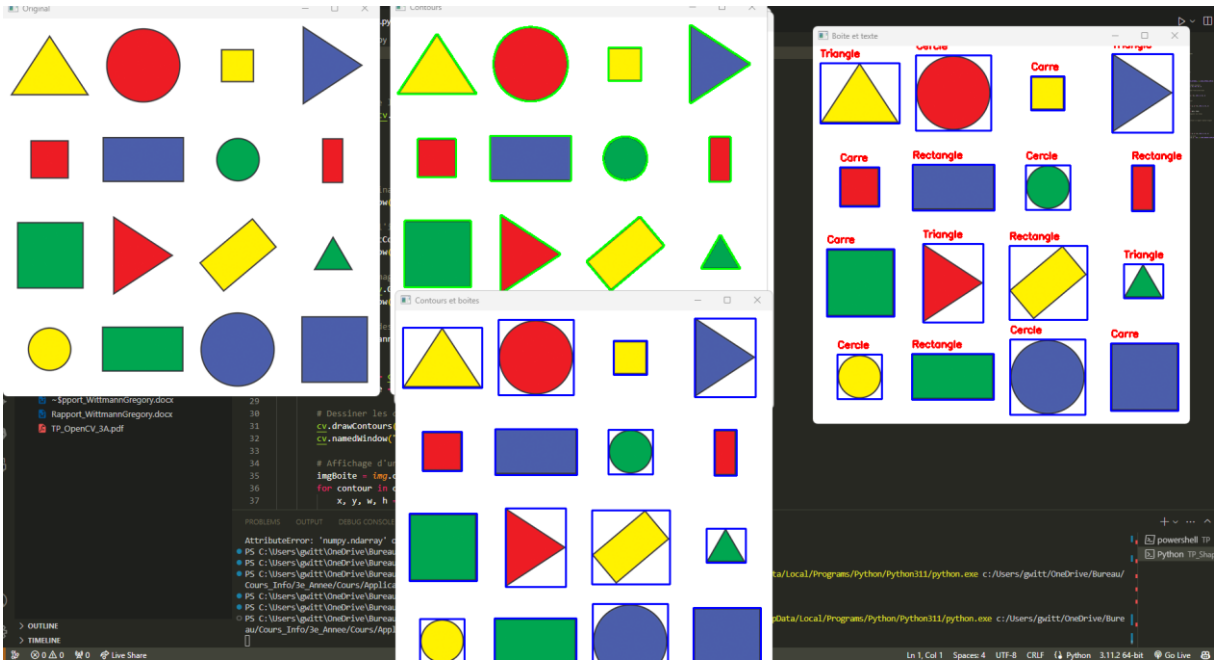
loadImage : Charge une image à partir du fichier 'shapes.png' en utilisant la fonction `cv.imread`.

main : Cette fonction effectue plusieurs étapes de traitement d'image :

- Convertit l'image en niveau de gris.
- Applique un lissage (Gaussian Blur) sur l'image en niveaux de gris.
- Utilise l'algorithme de détection de contours Canny.
- Trouve les contours dans l'image et dessine ces contours sur une copie de l'image originale.
- Dessine des boîtes englobantes autour des contours détectés.
- Ajoute du texte à chaque boîte englobante pour identifier la forme géométrique.

if __name__ == "__main__": : Appelle la fonction `main` avec une image chargée à partir de la fonction `loadImage` lorsque le script est exécuté directement.

Capture d’écran :



Virtual Paint

main : Cette fonction effectue plusieurs étapes :

Initialise le flux vidéo à partir de la caméra (capture vidéo) avec `cv.VideoCapture`.

Crée une fenêtre pour ajuster les valeurs HSV en temps réel à l'aide de trackbars.

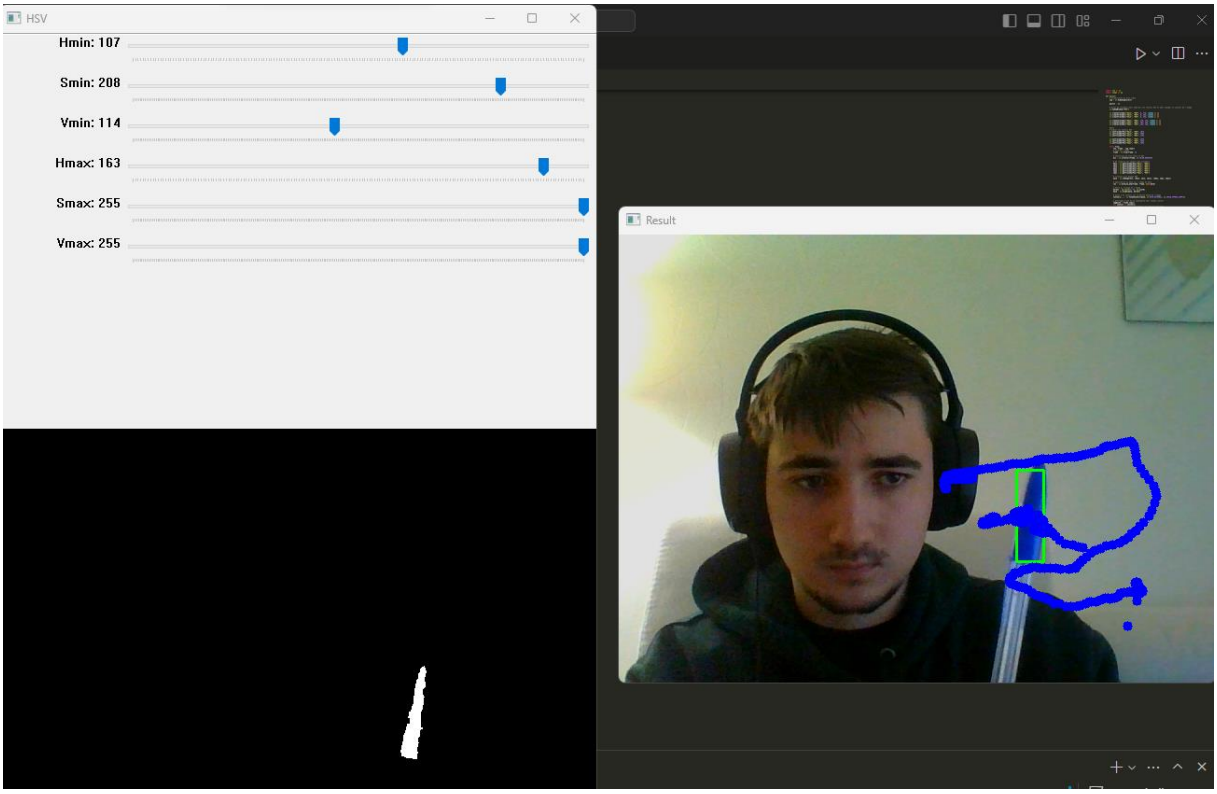
Configure initialement les valeurs HSV pour détecter la couleur bleue.

Dans une boucle infinie, capture chaque image du flux vidéo, effectue les opérations suivantes :

- Convertit l'image en espace de couleur HSV.
- Crée un masque HSV pour détecter la plage de couleurs spécifiée.
- Applique le masque sur l'image originale.
- Effectue un lissage sur le masque.
- Identifie les contours dans le masque.
- Dessine des boîtes englobantes autour des contours identifiés.
- Affiche l'image résultante avec les boîtes englobantes et les points centraux.
- Permet à l'utilisateur de quitter en appuyant sur 'q' ou de réinitialiser la liste des points en appuyant sur 'r'.

if __name__ == "__main__": : Appelle la fonction `main` lorsque le script est exécuté directement.

Capture d'écran :



Transformation de la perspective

main : Cette fonction effectue plusieurs étapes : Charge une image à partir du fichier 'cards.jpg'. Copie cette image dans deux autres variables (`imgB` et `imgC`).

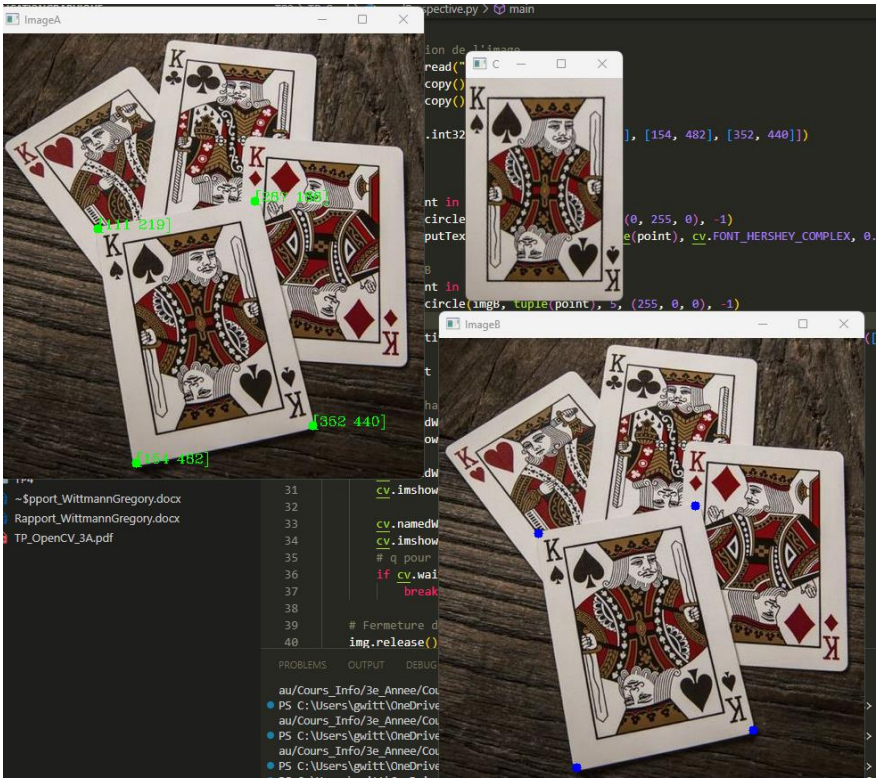
Définit manuellement quatre points (`points`) qui définissent une zone rectangulaire sur l'image. Dans une boucle, dessine des cercles et des textes aux positions des points sur l'image principale (`img`). Dessine des cercles aux mêmes positions sur une copie de l'image (`imgB`) avec une couleur différente.
Utilise `cv.getPerspectiveTransform` pour obtenir la matrice de transformation de perspective à partir des points définis (`points`) vers une nouvelle position rectangulaire.

Applique cette transformation à une copie de l'image (`imgC`) pour obtenir une nouvelle image (`goOutput`).

Affiche les images originales, les points marqués et l'image résultante après la transformation de perspective.

if __name__ == "__main__": : Appelle la fonction `main` lorsque le script est exécuté directement.

Capture d'écran :



Scanner de documents

preprocess_image : Convertit l'image en niveaux de gris, applique un flou gaussien, puis dilate l'image pour améliorer la détection des contours.

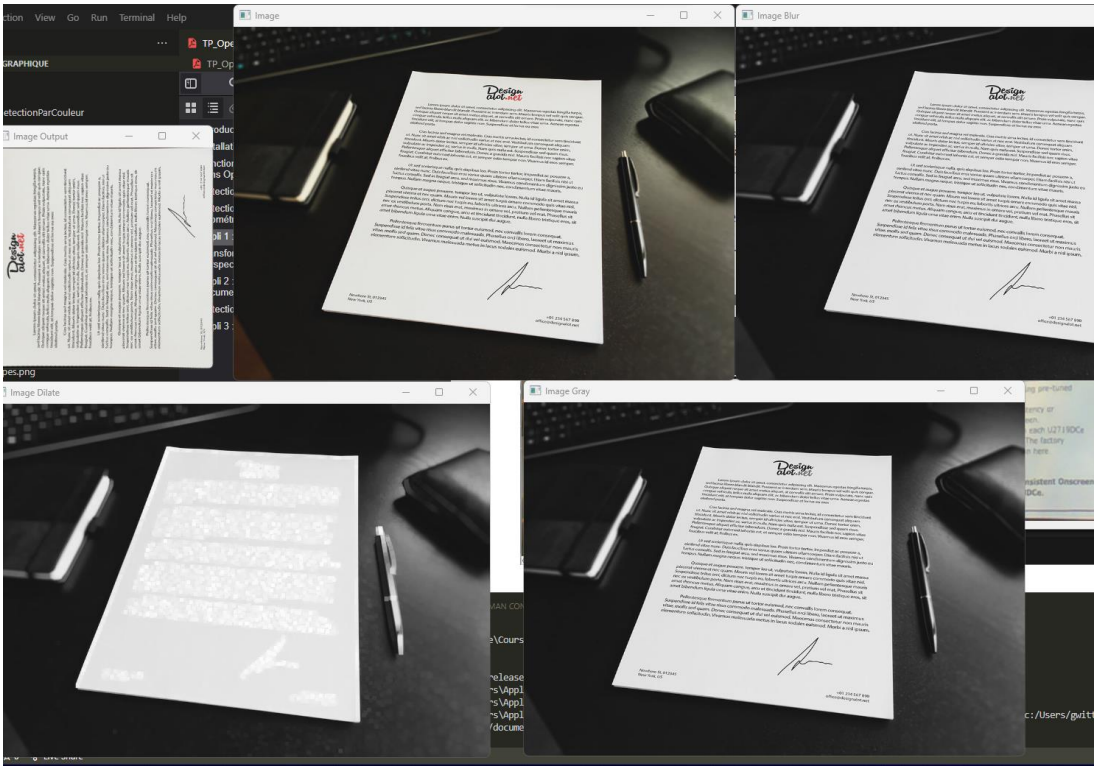
find_largest_contour : Applique un seuil à l'image dilatée, trouve les contours, puis identifie le plus grand contour.

find_corner_points : Approxime le contour pour obtenir les coins, puis récupère les points de ces coins.

apply_perspective_transform : Définit des points de destination pour la transformation de perspective, calcule la matrice de transformation de perspective, puis applique la transformation à l'image d'origine.

main : Charge une image, redimensionne l'image, prétraite l'image, trouve le plus grand contour, trouve les coins du contour, applique la transformation de perspective, et affiche les différentes étapes du processus.

Capture d'écran :



Scanner de documents (vidéo)

preprocess_image_video : Convertit chaque trame en niveaux de gris, applique un flou gaussien, puis dilate l'image pour améliorer la détection des contours.

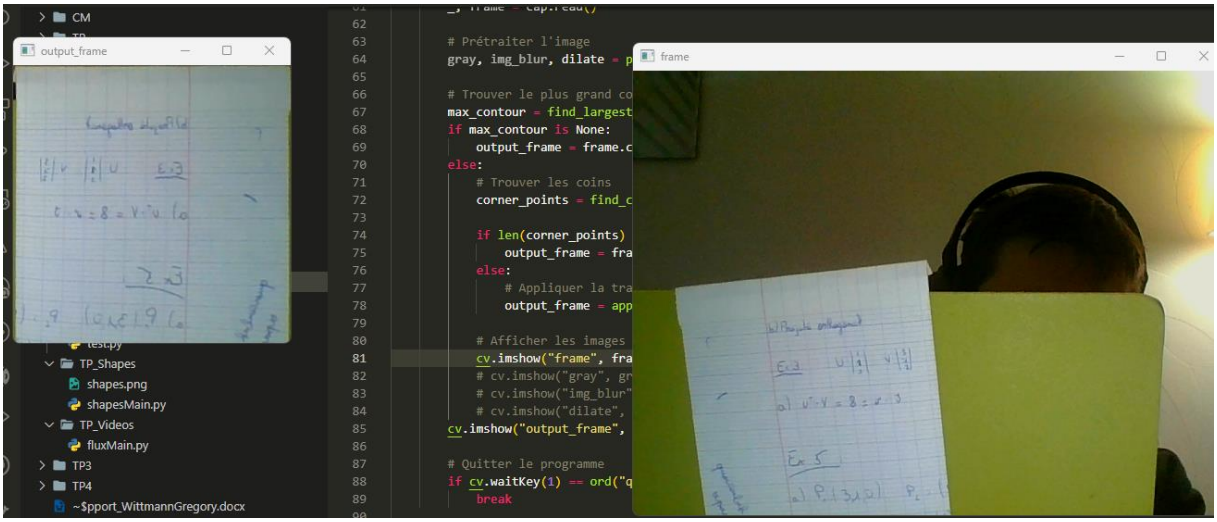
find_largest_contour_video : Applique un seuil à l'image dilatée, trouve les contours, puis identifie le plus grand contour.

find_corner_points_video : Approximation du contour pour obtenir les coins de la feuille.

apply_perspective_transform_video : Définit des points de destination pour la transformation de perspective, calcule la matrice de transformation de perspective, puis applique la transformation à chaque trame de la vidéo.

main : Capture la vidéo à partir de la caméra en temps réel, prétraite chaque trame, trouve le plus grand contour, trouve les coins, et applique la transformation de perspective en continu.

Capture d'écran :



Détection des visages

Chargement de l'image : L'image est chargée à partir du fichier "visages.jpg".

Conversion en niveaux de gris : L'image est convertie en niveaux de gris pour simplifier le processus de détection des visages.

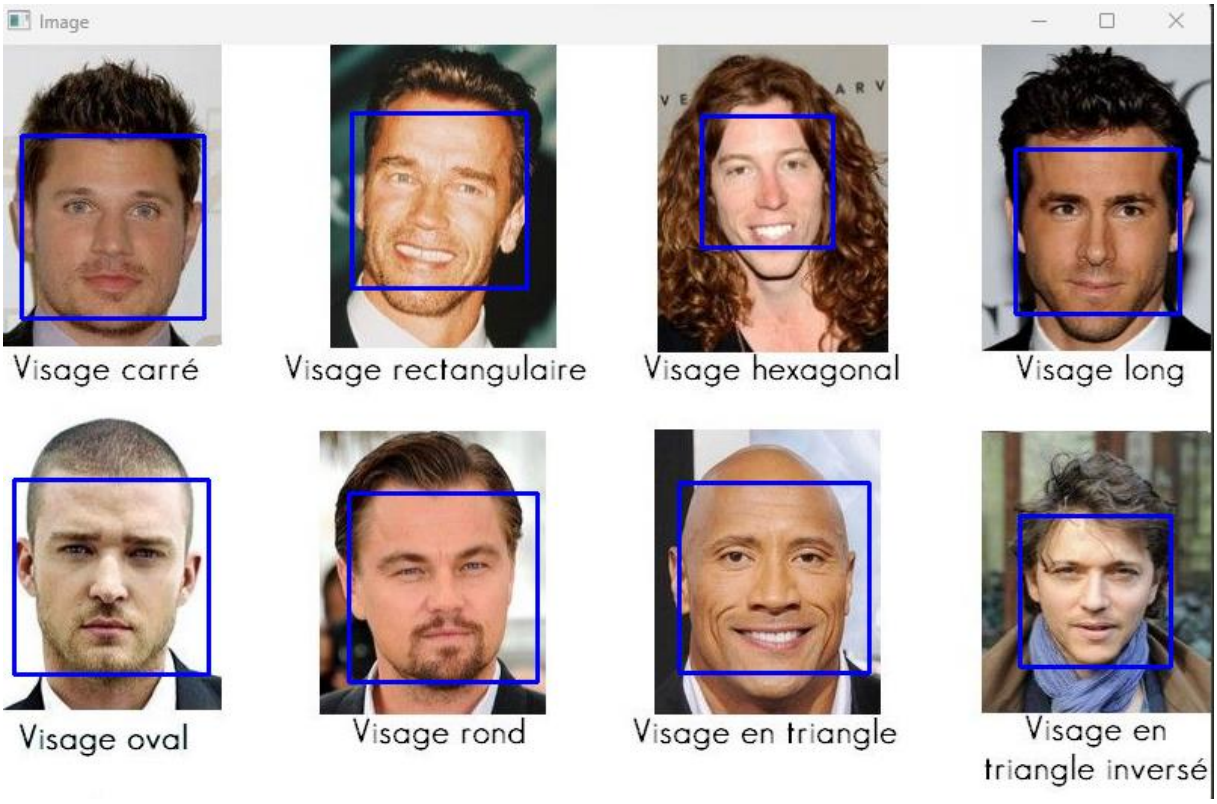
Chargement du classificateur Haar : Le classificateur Haar pour la détection de visages (`haarcascade_frontalface_default.xml`) est chargé. Ce fichier XML contient les informations nécessaires pour le détecteur de visage.

Détection des visages : La méthode `detectMultiScale` est utilisée pour détecter les visages dans l'image en niveaux de gris. Elle prend en compte le facteur d'échelle (`scaleFactor`) et le nombre minimum de voisins (`minNeighbors`).

Affichage des rectangles autour des visages détectés : Pour chaque rectangle représentant un visage détecté, un rectangle bleu est dessiné autour de lui à l'aide de la fonction `cv2.rectangle`.

Affichage de l'image : L'image résultante avec les rectangles autour des visages est affichée.

Capture d'écran :



Détection des plaques d’immatriculation

Chargement de l'image : L'image est chargée à partir du fichier "russian2.jpg".

Conversion en niveaux de gris : L'image est convertie en niveaux de gris pour simplifier le processus de détection des plaques d'immatriculation.

Chargement du classificateur Haar spécifique pour les plaques d'immatriculation russes : Le classificateur Haar spécifique (`haarcascade_licence_plate_rus_16stages.xml`) est chargé. Ce fichier XML contient les informations nécessaires pour le détecteur de plaques d'immatriculation.

Détection des plaques d'immatriculation : La méthode `detectMultiScale` est utilisée pour détecter les plaques d'immatriculation dans l'image en niveaux de gris. Elle prend en compte le facteur d'échelle (`scaleFactor`) et le nombre minimum de voisins (`minNeighbors`).

Affichage des rectangles autour des plaques d'immatriculation détectées : Pour chaque rectangle représentant une plaque d'immatriculation détectée, un rectangle vert est dessiné autour d'elle à l'aide de la fonction `cv2.rectangle`.

Capture d’écran :

Détection des plaques d’immatriculation (vidéo)

Ouverture de la caméra : La caméra est ouverte en utilisant `cv2.VideoCapture(0)`.

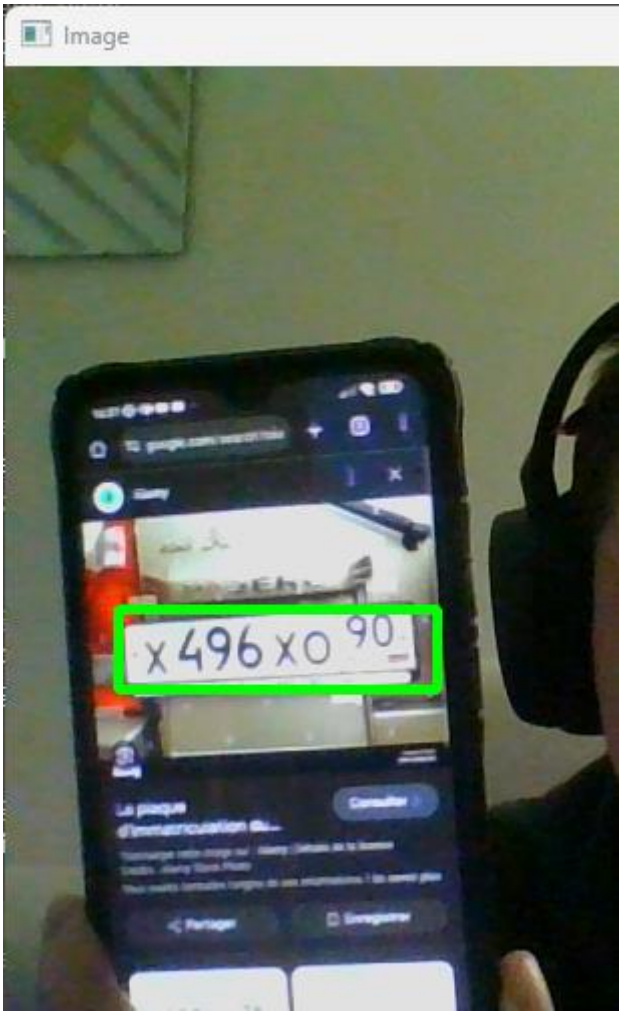
Chargement du classificateur Haar spécifique pour les plaques d'immatriculation russes : Le classificateur Haar spécifique (`haarcascade_licence_plate_rus_16stages.xml`) est chargé.

Boucle sur le flux vidéo : Le programme entre dans une boucle où chaque itération capture une image de la caméra.

Conversion en niveaux de gris : L'image capturée est convertie en niveaux de gris.

Détection des plaques d'immatriculation : La méthode `detectMultiScale` est utilisée pour détecter les plaques d'immatriculation dans l'image en niveaux de gris. Les rectangles entourant les plaques d'immatriculation détectées sont dessinés sur l'image originale.

Capture d’écran :



Conclusion

En conclusion, cette série d'expérimentations a démontré la polyvalence et l'applicabilité de diverses techniques de traitement d'image et de vision par ordinateur. Des approches telles que la détection par couleur, la reconnaissance d'objets géométriques, le "virtual paint", la correction de perspective, le scanner de documents, ainsi que la détection de visages et de plaques d'immatriculation ont été explorées avec succès.

Cependant, il est important de noter que ces résultats peuvent varier en fonction des conditions d'éclairage, de la qualité de l'image et d'autres facteurs environnementaux. Des ajustements supplémentaires pourraient être nécessaires pour une utilisation dans des scénarios plus complexes.