

Vision par Ordinateur

OpenCV : prise ne main

1 Introduction

OpenCV (Open Source Computer Vision Library : <http://opencv.org>) est une bibliothèque multi-plateforme qui permet de développer des applications de vision par ordinateur en temps réel. Elle se concentre principalement sur le traitement d'images, la capture et l'analyse de vidéos, y compris des fonctions telles que la détection de visages et d'objets.

La vision par ordinateur peut être définie comme une discipline qui explique comment reconstruire, interrompre et comprendre une scène en 3D à partir de ses images en 2D, en termes de propriétés de la structure présente dans la scène. Elle traite de la modélisation et de la reproduction de la vision humaine à l'aide de logiciels et de matériel informatique.

La vision par ordinateur recoupe de manière significative les domaines suivants :

- Traitement d'images - Il se concentre sur la manipulation d'images.
- Reconnaissance des formes - Elle explique les différentes techniques de classification des formes.
- Photogrammétrie - Il s'agit d'obtenir des mesures précises à partir d'images, des nuages de points 3D des surfaces des objets de la scène, par exemple.

2 Installation : OpenCV-Python

Commencez par installer sur votre machine OpenCV-Python qui est l'API Python d'*OpenCV*.

Assurez vous d'avoir un environnement de travail Python correctement configuré. N'oubliez pas de créer un environnement virtuel spécifique pour *OpenCV*.

Afin de vérifier votre installation, exécuter le code suivant :

```
import cv2 as cv
print(cv.__version__)
print("OpenCV Version: {}".format(cv.__version__))
```

3 Fonctionnalités de base dans OpenCV

Dans cette séquence d'exercices simples vous allez apprendre comment charger, afficher et enregistrer des images et des vidéos, comment dessiner des formes simples, comment contrôler les événements de la souris et comment créer des trackbars.

3.1 Charger, afficher et enregistrer une image

Codez une application qui vous permet de charger, afficher et sauver l'image sous un autre nom.

```

import cv2
img = cv2.imread("Img/lena.png")
cv2.imshow("in", img)
cv2.imwrite('Img/lenaBis.png', img)
cv2.waitKey(0)

```

3.2 Lire et visionner une vidéo

Codez une application qui permet de lire une vidéo.

```

import cv2
cap = cv2.VideoCapture("Img/sample-mp4-file-small.mp4")
while True:
    success, img = cap.read()
    cv2.imshow("Result", img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

```

3.3 Lire et visionner le flux de la caméra

Codez une application qui permet de lire et afficher le flux de la caméra.

```

import cv2
#capturing the web cam num 0
cap = cv2.VideoCapture(0)
if not cap.isOpened():
    print("Cannot open camera")
    exit()

cap.set(3,640) #width = id 3
cap.set(4,480) #height = id 4
cap.set(10,20) #brightness = id 10

while True:
    success, img = cap.read()
    cv2.imshow("Result", img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

```

3.4 Fonctions de traitements d'images

Étudiez les différentes fonctions de traitements d'image fournies par la librairie.

1. Commencer par charger une image en couleur.
2. Transformez la en niveaux de gris.
3. Ajoutez lui du bruit Gaussian.
4. Calculer les contours à l'aide de l'opérateur de Canny.
5. Faites une dilatation de l'image des contours avec un masque 5×5
6. Faites une érosion de l'image dilatée avec le même masque.
7. Changer la taille de l'image avec la méthode `resize`
8. Découper directement la matrice de l'image avec l'instruction `img[debut.height:fin.height, debut.width:fin.width]`

```

...
imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
imgBlur = cv2.GaussianBlur(imgGray, (7, 7), 0)
imgCanny = cv2.Canny(img, 150, 200)
...
kernel = np.ones((5, 5), np.uint8)
imgDialation = cv2.dilate(imgCanny, kernel, iterations=1)
imgEroded = cv2.erode(imgDialation, kernel, iterations=1)
...
imgResize = cv2.resize(img, (1000, 500))
imgCropped = img[0:200, 100:300]

```

3.5 Dessiner des formes simples : cercle, ligne, rectangle, texte

Apprenez à dessiner différentes formes géométriques 2D fournis par la librairie.

1. Commencer par créer une image noire de dimension 512×512 avec 3 canaux pour les couleurs, de type *unsigned int* en utilisant une matrice de zéros de la librairie *numpy* comme suit

```
img = np.zeros((512, 512, 3), np.uint8)
```

2. Colorier une partie de cette image en bleu en utilisant un intervalle

```
img[debut.h:fin.h, debut.w:fin.w] = 255, 0, 0
```

3. Dessiner une ligne sur votre image en utilisant l'instruction suivante :

```
ligne : img, pointDepart, pointFin, couleur, épaisseur
cv2.line(img, (0, 0), (300, 200), (0, 0, 255), 3)
```

4. Dessiner un rectangle sur l'image avec l'instruction suivante :

```
rectangle : img, pointDepart, pointFin, couleur, épaisseur
cv2.rectangle(img, (0, 0), (250, 350), (0, 0, 255), 2)
```

5. Dessiner un cercle plein avec l'instruction suivante :

```
cercle : img, pointCentre, rayon, couleur, épaisseur
cv2.circle(img, (400, 50), 30, (255, 255, 0), cv2.FILLED)
```

6. Ajouter un texte sur votre image avec l'instruction suivante :

```
texte : img, string, pointDepart, font, fontScale, couleur, epaisseur
cv2.putText(img, "TEST", (30, 200), cv2.FONT_HERSHEY_COMPLEX, 1, (0, 150, 0), 3)
```

7. Voir dans la documentation de la librairie quelles sont les autres formes géométriques simples et les expérimenter.

3.6 Exo1 : LOGO

Réaliser un joli LOGO 2D pour le Département INFO en utilisant ce que vous venez d'apprendre.

4 Gestion des événements

4.1 Gérer les événements de la souris

Dans cette section, vous allez apprendre comment gérer les événements de la souris dans OpenCV.

Pour ce faire, commencez par afficher la liste des différents actions pouvant être prises en compte avec les instructions suivantes :

```
events = [i for i in dir(cv) if 'EVENT' in i]
print(events)
```

Maintenant, vous allez créer une application graphique qui permet de dessiner des cercles, en faisant un double-click, sur une image.

Tout d'abord, vous allez créer une fonction de rappel pour la souris qui est exécutée lorsqu'un événement de souris se produit. Un événement de souris peut être n'importe quoi lié à la souris, comme le bouton gauche enfoncé, le bouton gauche relâché, un double-clic sur le bouton gauche, etc. Cela nous donne les coordonnées (x, y) pour chaque événement de souris. Avec cet événement et cette position, nous pouvons faire ce que nous voulons.

Créer une fonction de rappel pour la souris a un format spécifique qui est le même partout. Elle ne diffère que par ce que la fonction accomplit. Dans notre cas, notre fonction de rappel pour la souris fait une seule chose : elle dessine un cercle là où nous double-cliquons. Voici donc le code ci-dessous :

```
# mouse callback function
def draw_circle(event,x,y,flags,param):
    if event == cv.EVENT_LBUTTONDBLCLK:
        cv.circle(img,(x,y),10,(255,0,0),-1)

# Create a black image, a window and
# bind the mouse callback function to this window
img = np.zeros((512,512,3), np.uint8)
cv.namedWindow('image')
cv.setMouseCallback('image',draw_circle)

while(1):
    cv.imshow('image',img)
    if cv.waitKey(1) & 0xFF == ord('q'):
        break
```

Pour aller plus loin, vous pouvez ajouter la possibilité de dessiner des cercles ou des rectangles, en fonction du mode choisi par l'utilisateur. Pour ce faire, il faut définir dans la boucle principale, un "binding" clavier pour la touche '*m*' afin de basculer entre le rectangle et le cercle.

4.2 Créer des trackbars

Dans cette section, vous allez apprendre comment faire des trackbar ou barre de défilement dans OpenCV.

Maintenant, vous allez créer une application simple qui montre la couleur que vous spécifiez avec des trackbars. Vous allez avoir une fenêtre qui affiche la couleur choisie et les trois barres de défilement pour spécifier chacune des couleurs *B*, *G*, *R*. En faisant glisser la barre de défilement, la couleur de la fenêtre change en conséquence. Par défaut, la couleur initiale sera définie sur noir.

```
# Create a black image, a window
img = np.zeros((300,512,3), np.uint8)
cv.namedWindow('image')
```

Pour la fonction `cv.createTrackbar()`, le premier argument est le nom de la barre de défilement, le deuxième est le nom de la fenêtre à laquelle elle est attachée, le troisième argument est la valeur par défaut, le quatrième est la valeur maximale et le cinquième est la fonction de rappel qui est exécutée chaque fois que la valeur de la barre de défilement change.

```
...
cv.createTrackbar('R','image',0,255,nothing)
...
```

La fonction de rappel (callback) a toujours un argument par défaut qui est la position de la barre de défilement. Dans notre cas, la fonction ne fait rien, donc vous allez passer simplement.

```
#callback function
def nothing(x):
    pass
```

Nota Bene : Une autre application importante de la barre de défilement est de l'utiliser comme un **bouton** ou un **interrupteur**. La librairie OpenCV, par défaut, n'a pas de fonctionnalité de bouton. On peut donc utiliser la barre de défilement pour obtenir une telle fonctionnalité.

Dans cette application, vous allez créer un tel interrupteur pour que l'application fonctionne uniquement si l'interrupteur est ACTIVÉ, sinon l'écran est toujours noir.

```
while(1):
    cv.imshow('image',img)
    if cv.waitKey(1) & 0xFF == ord('q'):
        break
    # get current positions of four trackbars
    r = cv.getTrackbarPos('R','image')
    g = cv.getTrackbarPos('G','image')
    b = cv.getTrackbarPos('B','image')
    s = cv.getTrackbarPos(switch,'image')
    if s == 0:
        img[:] = 0
    else:
        img[:] = [b,g,r]
```

4.3 Exo2 : Paint

Créez une application Paint avec des couleurs (RGB) et un rayon de pinceau réglables à l'aide de trackbars.

5 Détection par la couleur

Créer une application simple qui permet de segmenter un objet dans une image par sa couleur. La figure 1 montre un exemple où une voiture jaune a été sélectionnée. Vous devez utiliser des trackbars pour permettre à l'utilisateur de choisir la couleur de l'objet, dans l'espace de couleur HSV.

Voici un algorithme possible :

1. Convertir l'image d'entrée dans l'espace couleur HSV
-> `imgHSV = cv.cvtColor(img, cv.COLOR_BGR2HSV)`
2. A l'aide de 6 trackbars obtenir :
la couleur MIN (`lower = np.array([h_min, s_min, v_min])`)
et la couleur MAX (`upper = np.array([h_max, s_max, v_max])`)
3. Segmenter l'image d'entrée dans l'intervalle [MIN,MAX]
-> `mask = cv.inRange(imgHSV, lower, upper)`
4. Utiliser l'opérateur AND_Logique entre l'image d'entrée et le mask
-> `imgResult = cv.bitwise_and(img, img, mask=mask)`

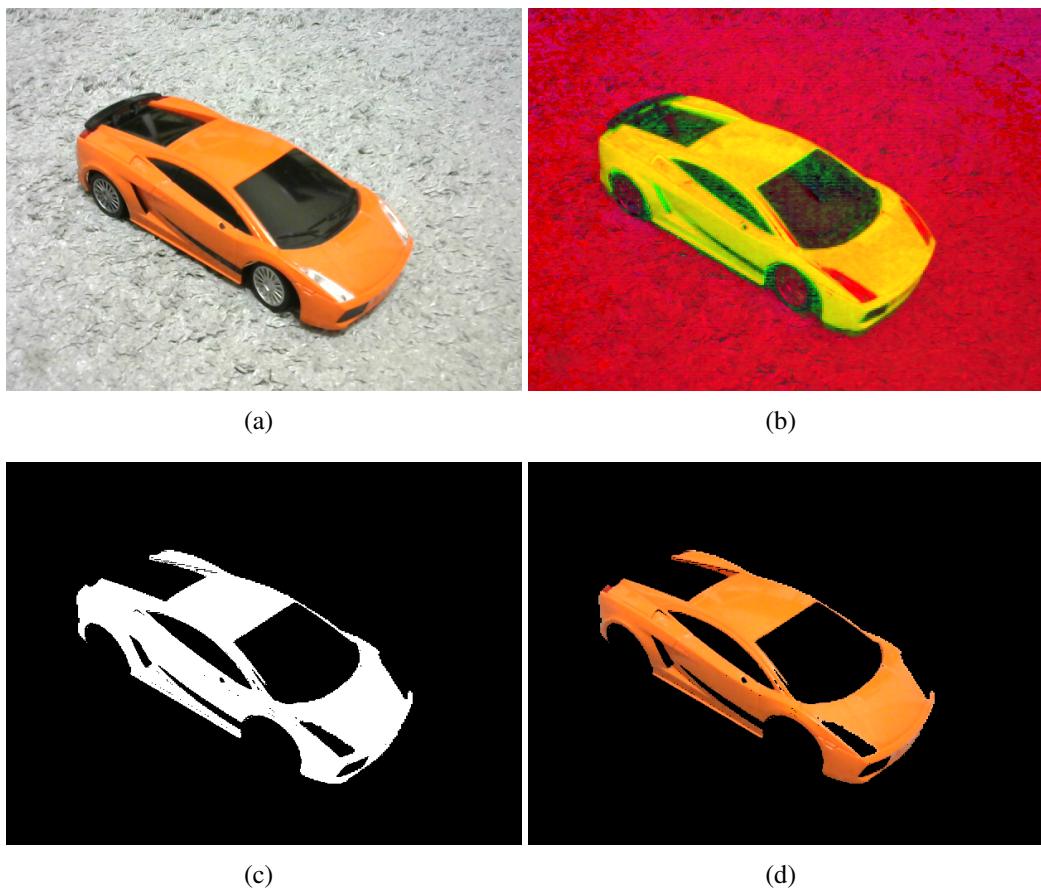


FIGURE 1 – (a) image input en RGB, (b) image input en HSV, (c) mask binaire, (d) résultat de la segmentation

5.1 Exo : Sélectionneur de couleur

Adapter l'application précédente afin d'obtenir un "SelectionneurDeCouleur" qui permet de sélectionner une couleur dans le flux de la vidéo de la caméra.

6 Détection d'objets géométriques

Créer une application simple dont le pipeline est illustré sur la figure 2.

Voici un algorithme possible :

1. Transformer l'image d'entrée en niveaux de gris
-> `imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)`
2. Réduire le bruit à l'aide d'un filtre Gaussian
-> `imgBlur = cv2.GaussianBlur(imgGray, (7, 7), 1)`
3. Extraire les pixels des contours avec un filtre de Canny
-> `cv2.Canny(imgBlur, 50, 50)`
4. Trouver les contours
-> `contours, hierarchy`
= `cv2.findContours(imgCanny, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)`
Rq : contours est une liste Python de tous les contours de l'image.
Chaque contour individuel est un tableau Numpy de coordonnées (x, y)
des points limites (sur le contour) de l'objet.
5. Afficher les contours trouvés
-> `for cnt in contours:`
 `cv2.drawContours(imgContour, cnt, -1, (255, 0, 0), 3)`
5. Calculer la boîte englobante et le cercle circonscrit
de chaque contour trouvé
-> `for cnt in contours:`
 `x, y, w, h = cv2.boundingRect(cnt)`
 `center, radius = cv2.minEnclosingCircle(approx)`
6. Calculer une approximation polygonale avec le minimum de sommets
-> `for cnt in contours:`
 `peri = cv2.arcLength(cnt, True) # longueur du contour`
 `approx = cv2.approxPolyDP(cnt, 0.02*peri, True)`
 # affiche le nombre de sommets dans l'approximation du contour
 `print(len(approx))`
7. Utiliser le nombre de coins pour étiqueter les formes
-> `cv2.putText(imgBBox, "objet", (x,y),`
 `cv2.FONT_HERSHEY_COMPLEX, 0.7, (0,0,0), 2)`

7 Appli 1 : Virtual Paint

Combiner les éléments appris précédemment afin de réaliser une application de "Virtual Paint" (voir Fig. 3). Le but est de détecter la couleur du bouchon d'un feutre dans le flux de la caméra et de lui associer un crayon virtuel qui sera utilisé pour dessiner sur les images en temps réel.

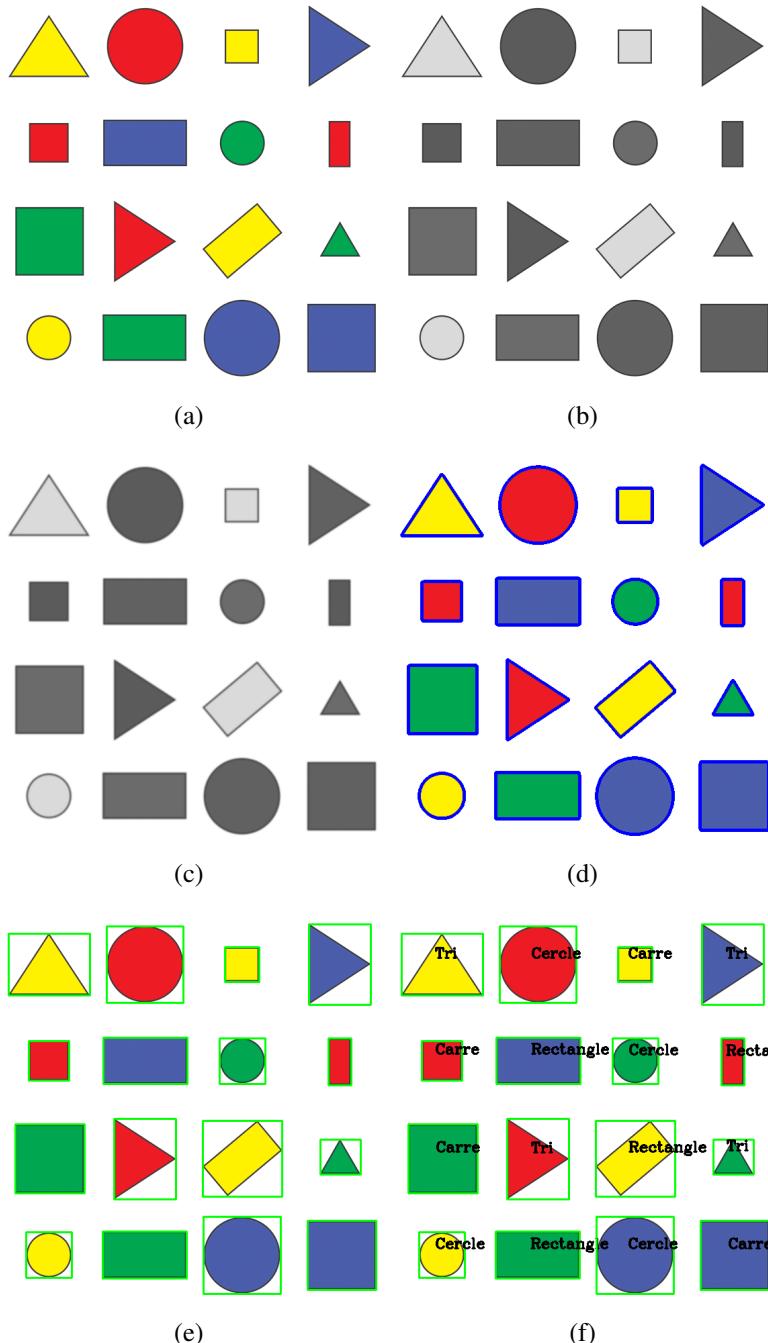


FIGURE 2 – (a) image input, (b) image input en niveaux de gris, (c) image lissée, (d) contours, (e) boîtes englobantes, (f) objets reconnus avec labels

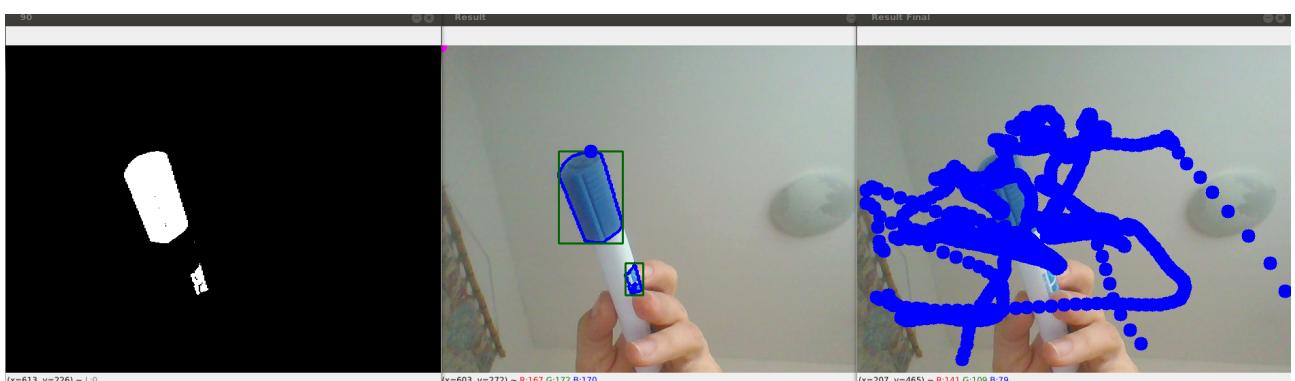


FIGURE 3 – Virtual Paint

8 Transformation de la perspective

Créer une application simple permettant de changer la perspective d'une image dont le pipeline est illustré sur la figure 4.

Pour la transformation de perspective, vous avez besoin d'une matrice de transformation 3×3 telle que les lignes droites resteront droites après la transformation. Pour trouver cette matrice, vous avez besoin de 4 points sur l'image d'entrée (points verts et bleus) et les points correspondants sur l'image de sortie (points rouges). Parmi ces points, au moins 3 d'entre eux ne doivent pas être colinéaires. La matrice de transformation peut être trouvée par la fonction `cv.getPerspectiveTransform`. Appliquez ensuite la méthode `cv.warpPerspective` avec cette matrice de transformation 3×3 .

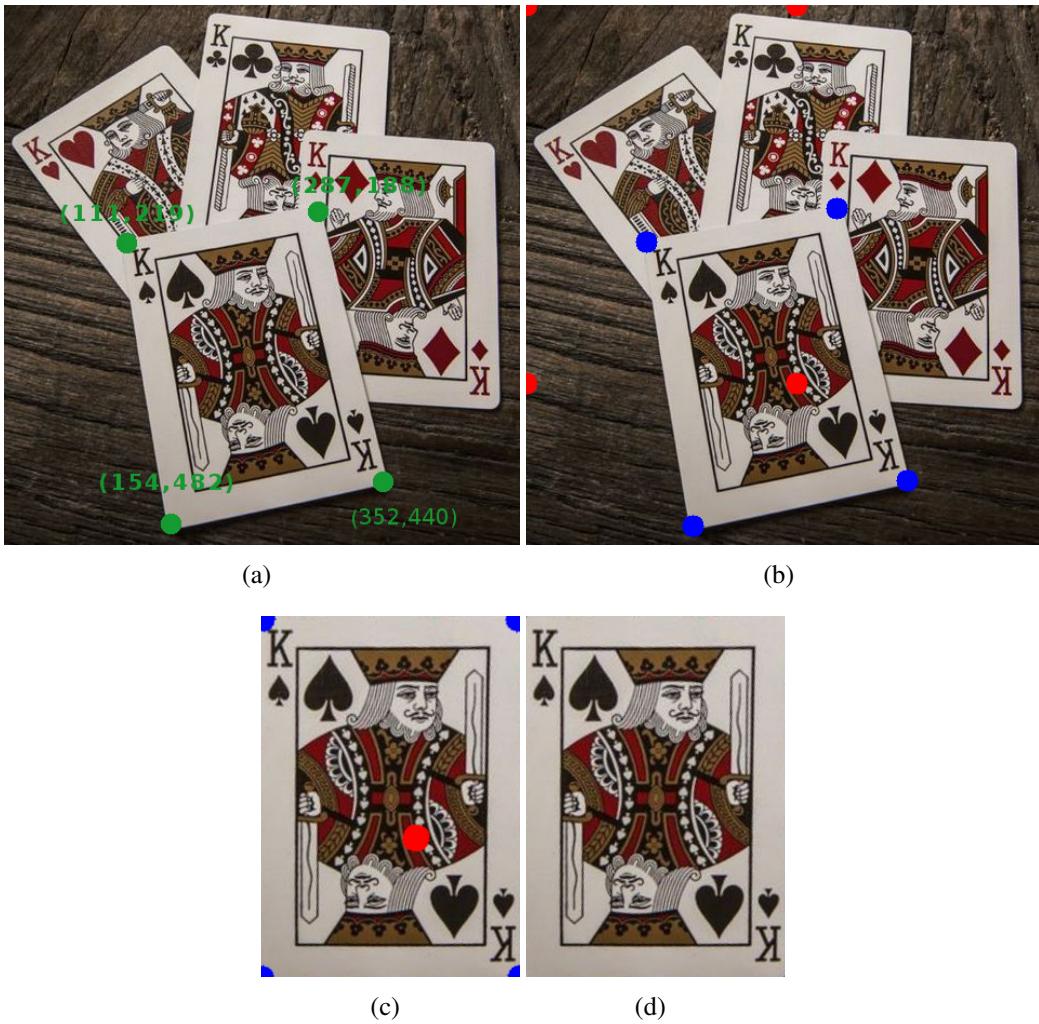


FIGURE 4 – (a) image input avec coins de la carte, (b) image input avec les 4 points initiaux (en bleu) et les 4 points après transformation (en rouge), (c) image après transformation, (d) résultat final

8.1 Qu'est-ce que la transformation de la perspective ?

Comme son nom l'indique, la transformation de perspective est associée au changement de point de vue. Ce type de transformation ne préserve pas le parallélisme, la longueur et l'angle. En revanche, il préserve la colinéarité et l'incidence. Cela signifie que les lignes droites resteront droites même après la transformation.

Sur la Figure 5, (x', y') sont les points transformés tandis que (x, y) sont les points d'entrée. La matrice de transformation (M) peut être considérée comme une combinaison d'une rotation, d'une translation et d'une projection. Pour la transformation affine, le vecteur de projection est égal à 0.

$$\begin{bmatrix} t_i x' \\ t_i y' \\ t_i \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \\ c_1 & c_2 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

↳ Scaling Factor ↳ Transformation Matrix (M)

(a)

$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \rightarrow$ defines transformations such as rotation, scaling etc

$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \rightarrow$ defines translation vector

$\begin{bmatrix} c_1 & c_2 \end{bmatrix} \rightarrow$ projection vector

(b)

FIGURE 5 – (a) matrice de transformation affine, (b) les sous-matrices

La transformation affine peut donc être considérée comme un cas particulier de transformation de la perspective.

La matrice de transformation (M) étant définie par 8 constantes (degré de liberté), pour trouver cette matrice, nous sélectionnons d'abord 4 points dans l'image d'entrée et nous mappons ces 4 points aux emplacements souhaités dans l'image de sortie. Ces points sont inconnus en avance et doivent être trouvés en fonction du cas d'utilisation (de cette manière, nous aurons 8 équations et 8 inconnues qui pourront être facilement résolues).

Une fois la matrice de transformation calculée, nous appliquons la transformation en perspective à l'ensemble de l'image d'entrée pour obtenir l'image transformée finale.

9 Appli : Scanner de documents

Combiner les éléments appris précédemment afin de réaliser une application "Scanner de documents" (voir Fig.6 et Fig.7).

Commencer par mettre en place le pipeline de transformation sur l'image fixe, fournie sur Arche (voir Fig.6).

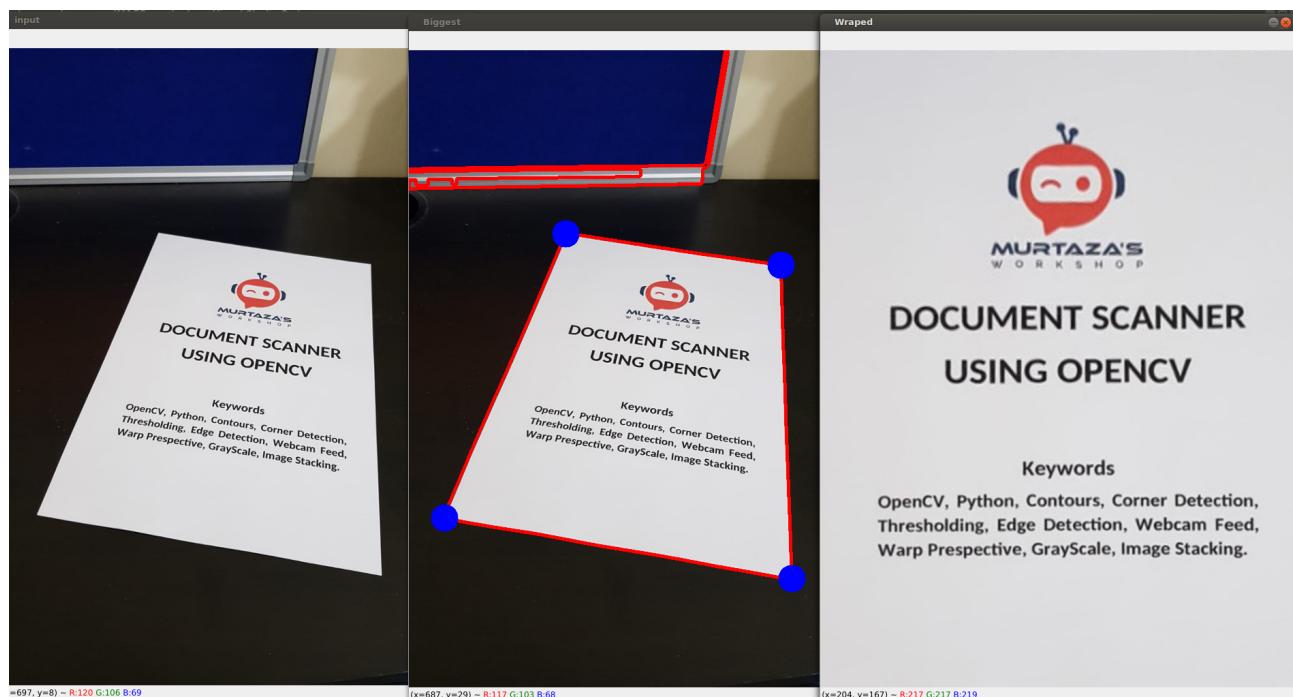


FIGURE 6 – Scanner de documents à partir d'une image fixe.

L'algorithme doit suivre les étapes suivantes :

1. Charger l'image
2. Pre-processing
 - convertir l'image en niveaux de gris
 - atténuer le bruit avec un filtre de flou
 - trouver les contours avec le filtre de Canny
 - si les contours sont trop fins, faire une passe de dilatation/érosion
3. Trouver le contour le plus long dont on récupère les 4 coins
4. Changer le point de vue à l'aide des 4 coins précédents.

Ensuite, faites une seconde version de l'application qui permet de scanner des documents à partir du flux de la caméra, comme montré sur la Fig.7.

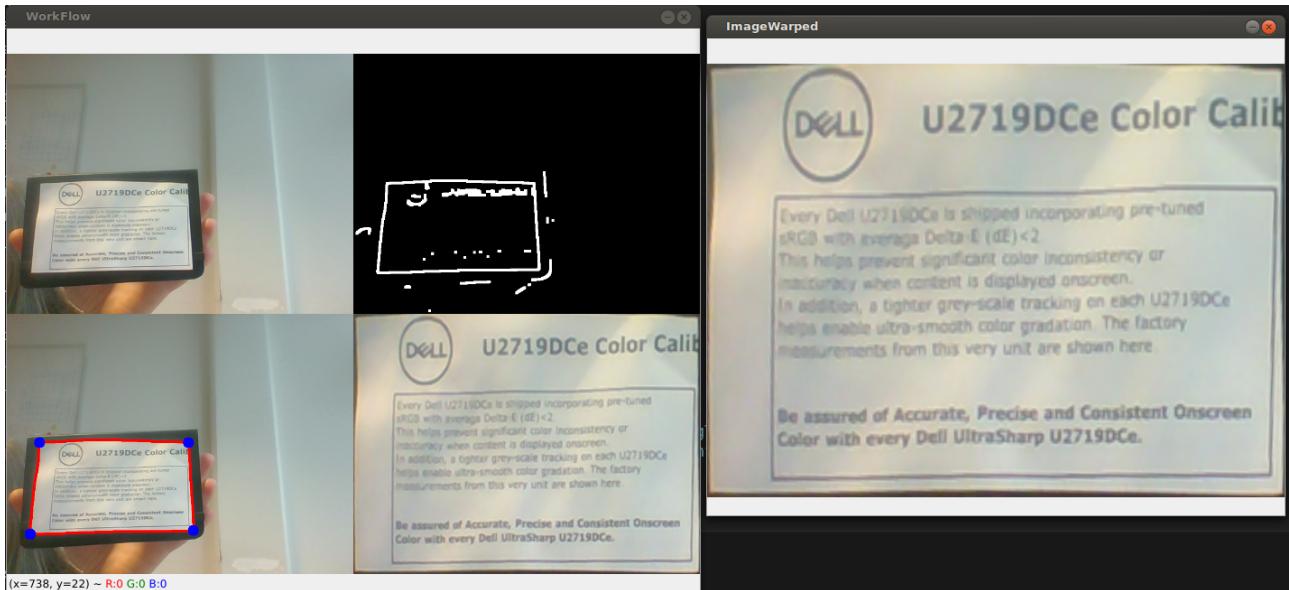


FIGURE 7 – Scanner de documents à partir de la caméra.

10 Détection de visages

Dans cette section, nous allons voir comment détecter les visages à l'aide d'un classificateur en cascade dans OpenCV Python. La détection des visages revêt une grande importance dans différents domaines du monde d'aujourd'hui. C'est une étape importante dans plusieurs applications, la reconnaissance faciale (également utilisée comme biométrie), la photographie (pour l'autofocus sur le visage), l'analyse du visage (âge, sexe, reconnaissance des émotions), la vidéosurveillance, etc.

L'un des algorithmes populaires pour la détection faciale est « haarcascade ». Il s'agit d'un algorithme moins coûteux en termes de calcul, rapide et offrant une grande précision.

Cela fonctionne en quatre étapes :

1. Sélection des caractéristiques Haar : une caractéristique de type Haar se compose de régions sombres et de régions claires. Il produit une valeur unique en prenant la différence entre la somme des intensités des régions sombres et la somme des intensités des régions claires. Il s'agit d'extraire les éléments utiles nécessaires à l'identification d'un objet. Les "features" proposées par Viola and Jones sont :
2. Cr?ation d'images intégrales : Un pixel donné dans l'image intégrale est la somme de tous les pixels à gauche et de tous les pixels au-dessus. Étant donné que le processus d'extraction des caractéristiques de type Haar implique le calcul de la différence entre les régions rectangulaires sombres et claires, l'introduction des images intégrales réduit considérablement le temps nécessaire pour accomplir cette tâche.

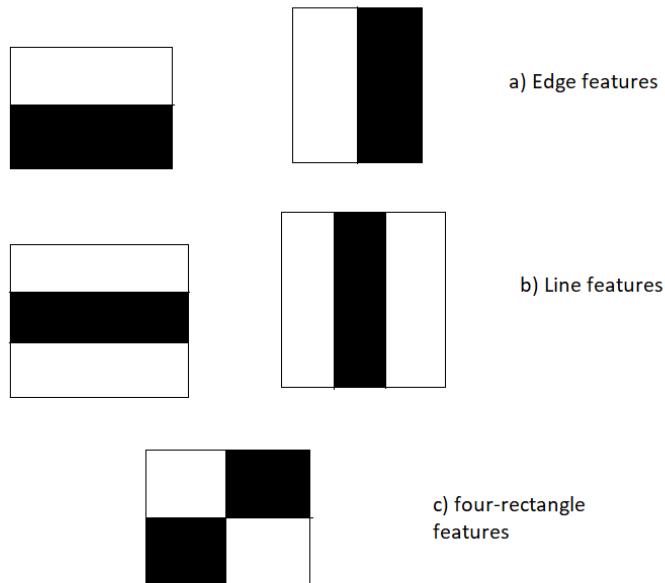


FIGURE 8 – Scanner de documents

3. Formation AdaBoost : cet algorithme sélectionne les meilleures "features" parmi tous les features détectés. Il combine plusieurs « classificateurs faibles » (meilleures fonctionnalités) en un seul « classificateur fort ». Le « classificateur fort » généré est essentiellement la combinaison linéaire de tous les « classificateurs faibles ».
4. Classificateur en cascade : il s'agit d'une méthode permettant de combiner des classificateurs de plus en plus complexes comme AdaBoost dans une cascade qui permet d'éliminer rapidement les entrées négatives (non-visage, si on cherche des visages) tout en dépensant plus de calculs sur des régions semblables à des visages prometteuses ou positives. Cela réduit considérablement le temps de calcul et rend le processus plus efficace.

OpenCV est livré avec de nombreux classificateurs pré-entraînés. Ces fichiers XML peuvent être chargés par la méthode `cv2.CascadeClassifier`. Ici, nous allons utiliser `haarcascade_frontalface_default.xml` pour détecter les visages.

Voici un algorithme possible :

1. Charger l'image et la convertir en niveaux de gris
-> `gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)`
2. Charger le fichier XML du classifieur Haar
-> `haar_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')`
3. Détecter les visages
-> `faces_rect = haar_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=9)`
4. Afficher un rectangle autour de chaque face
-> `for (x, y, w, h) in faces_rect:
 cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)`

La méthode `cv2::CascadeClassifier::detectMultiScale` renvoie les boîtes englobantes des faces détectées (c'est-à-dire x, y, w, h). Il lui faut deux paramètres, à savoir `scaleFactor` et `minNeighbors`. `ScaleFactor` détermine le facteur d'augmentation de la taille de la fenêtre qui

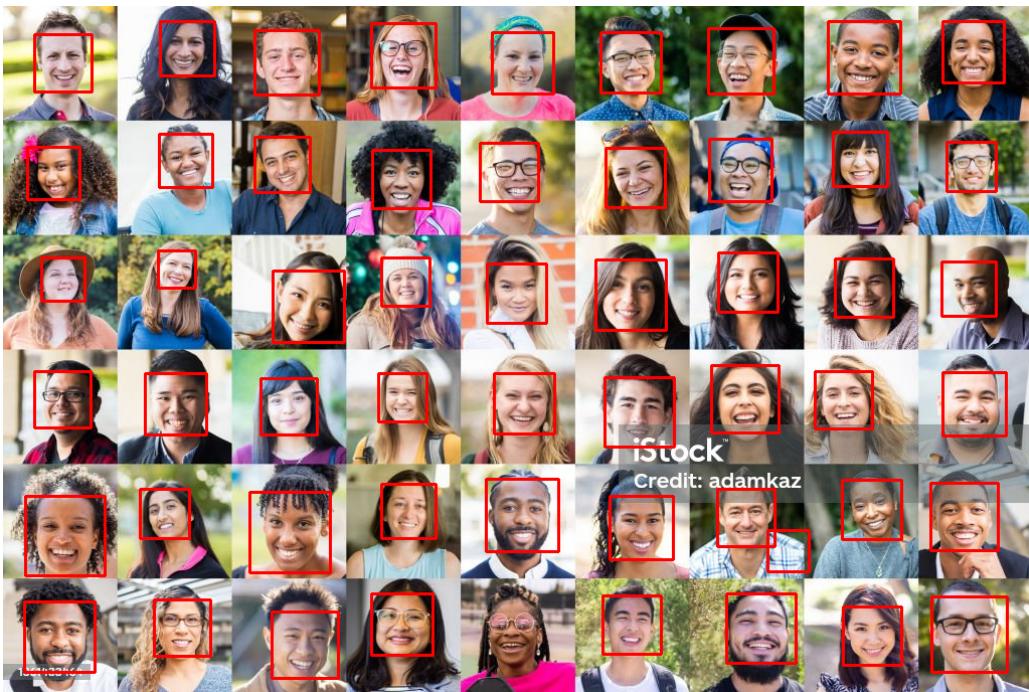


FIGURE 9 – Détection de visages.

commence initialement à la taille « minSize », et après avoir testé toutes les fenêtres de cette taille, la fenêtre est agrandie par « scaleFactor » et la taille de la fenêtre augmente jusqu'à « maxSize ». Si le « scaleFactor » est grand (par exemple, 2,0), il y aura moins d'étapes, donc la détection sera plus rapide, mais nous pourrions manquer des objets dont la taille se situe entre deux échelles testées. (le facteur d'échelle par défaut est 1,3). Plus les valeurs de « minNeighbors » sont élevées, moins il y aura de faux positifs et moins d'erreurs seront en termes de fausse détection de visages. Cependant, il est également possible que des traces floues du visage manquent.

11 Appli 3 : Radar

Combiner les éléments appris précédemment afin de réaliser une application "RADAR" (voir Fig. 10).

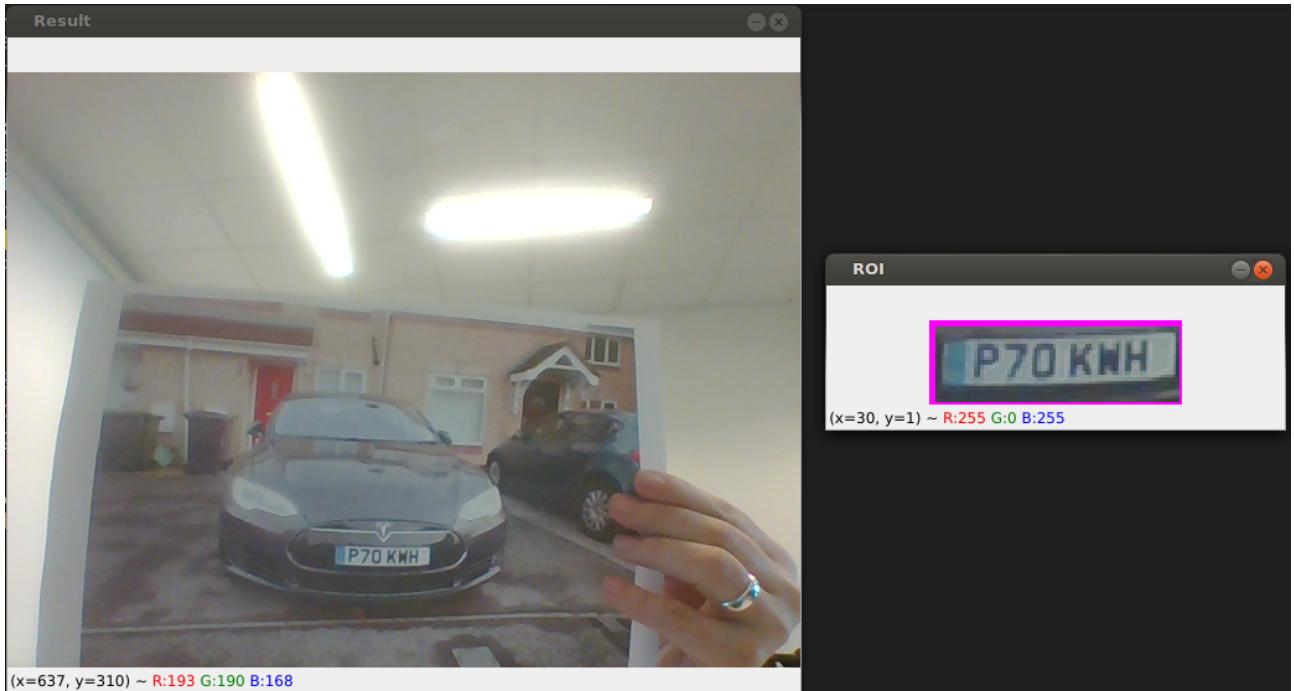


FIGURE 10 – Appli radar.