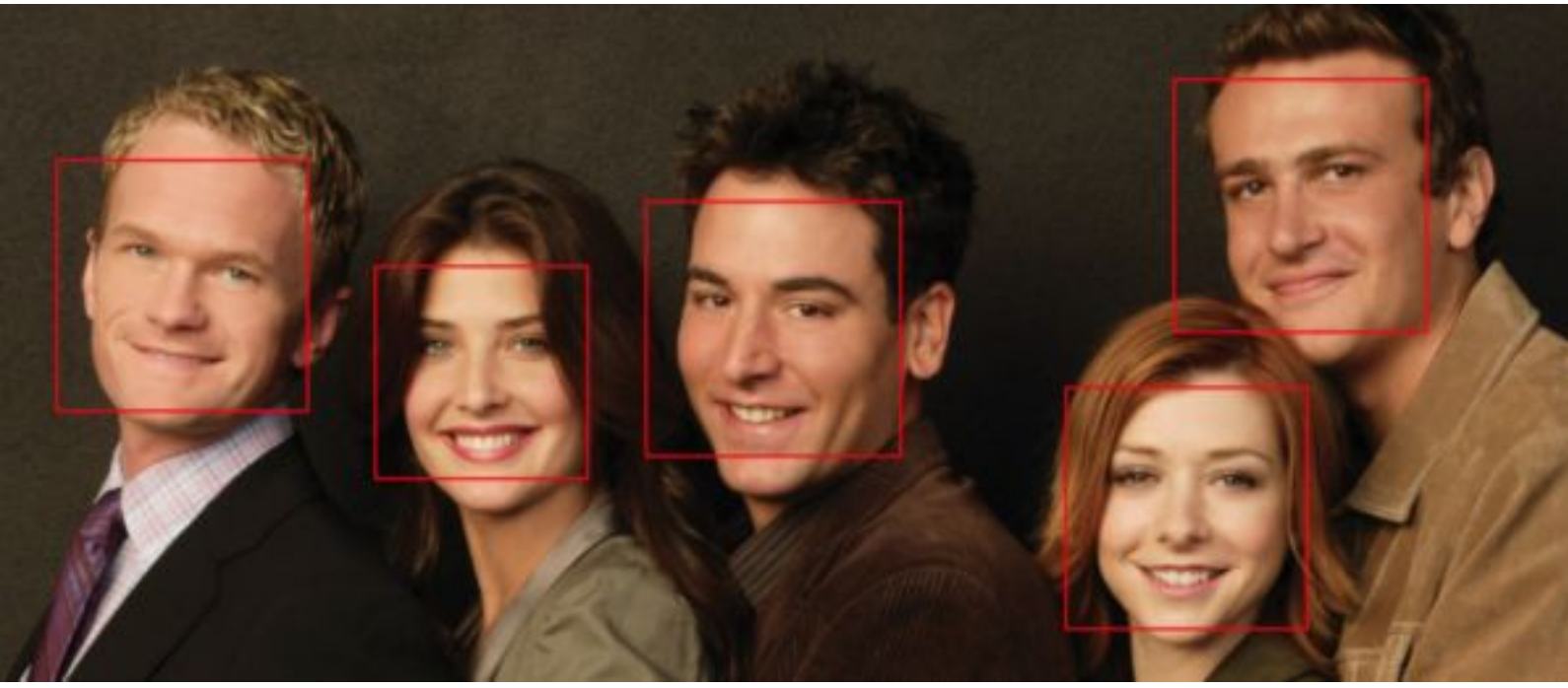
VISION

-

CASCADE DE HAAR



Florent Muret

Jean-Baptiste Rambaud

Introduction

Ce projet s'intéresse à l'usage de solution d'analyse d'images appliquées à la robotique. En effet, la détection d'image est très utile pour de la détection de bords, l'analyse des émotions ou l'évitement d'obstacles par exemple.

Au cours de ce projet, nous allons nous familiariser avec OpenCV qui est la solution la plus utilisée en traitement d'image.

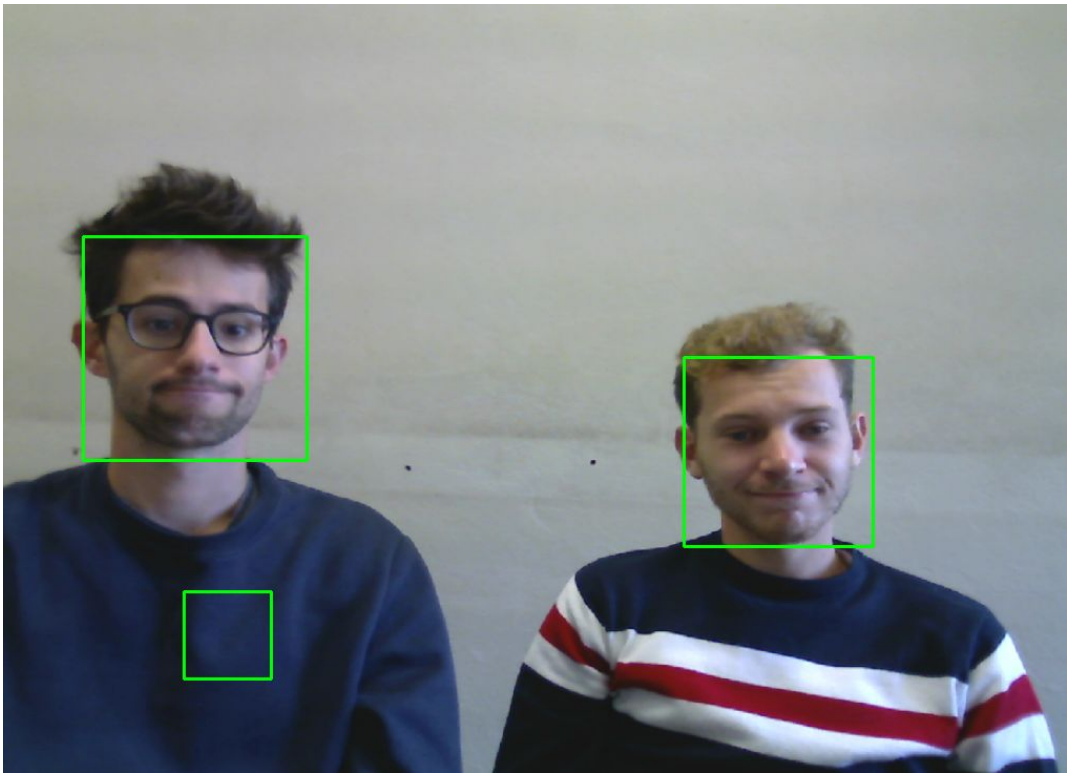
Nous étudierons également une cascade de haar, son principe et sa mise en oeuvre pour de la détection de visage.

Enfin, nous créerons notre propre classifier à l'aide d'une base de données que nous aurons nous-mêmes établis.

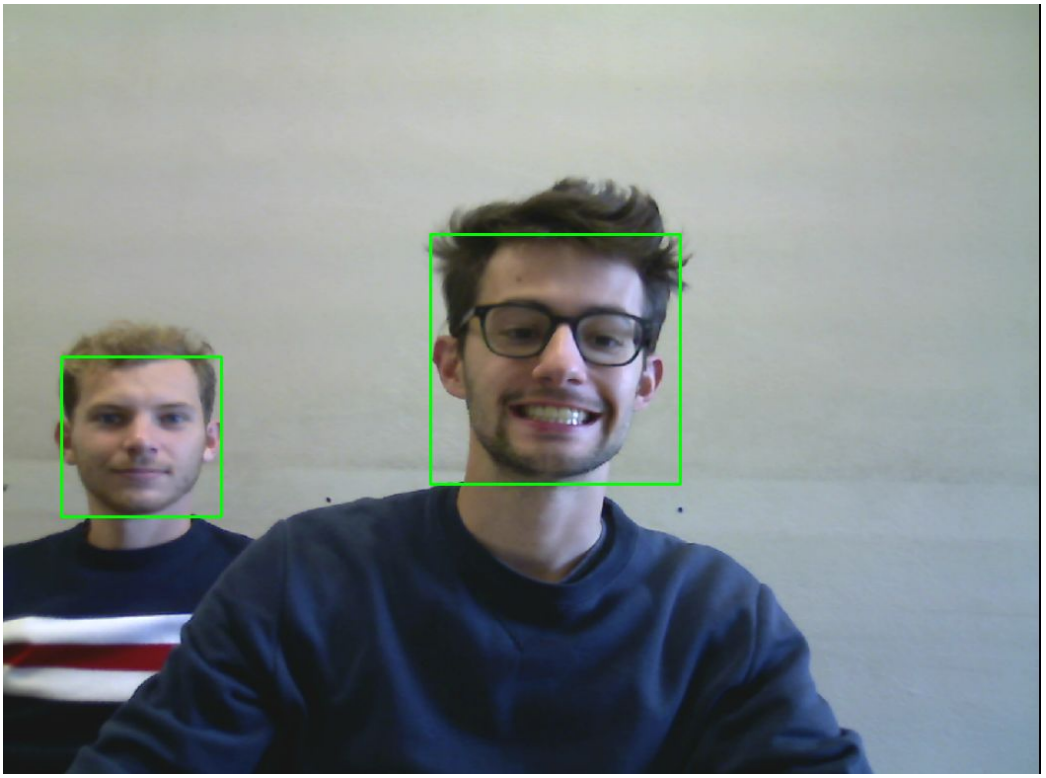
Partie 1 : prise en main de l'article de blog

Une cascade de Haar est une technique utilisé en vision pour la détection d'objet dans une image. Une Cascade Haar est fondamentalement un classificateur qui est utilisé pour détecter l'objet pour lequel elle a été entraînée. Le principe de la Cascade de Haar consiste à superposer l'image positive à un ensemble d'images négatives.

Dans l'optique d'utiliser les techniques de Haar cascade avec OpenCV, on commence par mettre en place un code permettant de détecter les visages. Pour cela, nous utilisons les classifiers fournis avec OpenCV.



On observe que le pull de Florent est détecté comme étant un visage ce qui n'est pas le résultat désiré. Nous augmentons alors légèrement le scaleFactor.



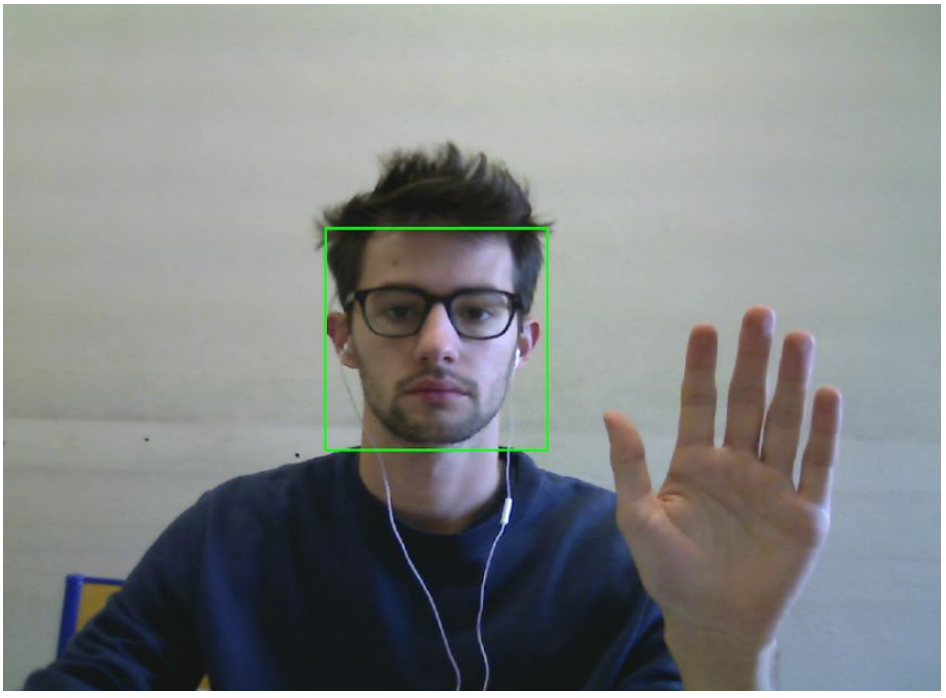
La résultat obtenu correspond mieux à ce que l'on souhaitait avoir. La fonction detectMultiScale() qui permet de faire une détection de type haar cascade prend pour paramètre en entrée:

- **factorScale** : échelle à laquelle chaque cascade doit faire évoluer le traitement de l'image
- **minNeighbors** : nombre minimum de voisin qui doivent participer à la détection de l'objet choisi
- **minSize** : taille minimum de l'objet
- **maxSize** : taille maximum de l'objet

Partie 2 : traitement en temps réel - caméra sous opencv

On commence par analyser le code décrit dans l'article suivant:
https://realpython.com/face-detection-in-python-using-a-webcam/

Ce code permet de faire de la détection de visage à l'aide d'une webcam et d'openCV.



On implémente suite un programme qui effectue une détection de haar directement sur le flux webcam.

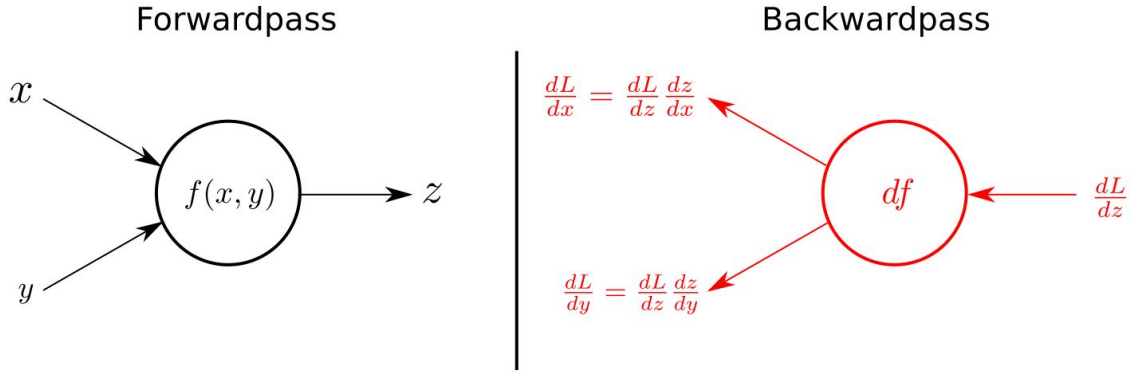
Si jamais on branchait deux caméras, il serait possible de sélectionner la webcam à utiliser en changeant le numéro en paramètre de l'objet VideoCapture d'openCV. En effet, VideoCapture(0) correspondant à la première caméra trouvée sur l'ordinateur.

```
import cv2
import sys

cascPath = sys.argv[1]
faceCascade = cv2.CascadeClassifier(cascPath)

video_capture = cv2.VideoCapture(0)

while True:
    # Capture frame-by-frame
    ret, frame = video_capture.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.1,
        minNeighbors=5,
        minSize=(30, 30),
        flags=cv2.cv.CV_HAAR_SCALE_IMAGE
    )

    # Draw a rectangle around the faces
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

    # Display the resulting frame
    cv2.imshow('Video', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# When everything is done, release the capture
video_capture.release()
cv2.destroyAllWindows()
```

Partie 3: création d'un classifieur

Dans cette partie on cherche à créer nos propres classifieurs pour des objets spécifiques que l'on souhaite pouvoir détecter. Pour nous aider dans cette démarche, de nombreuses ressources sont disponibles comme ImageNet qui est un site qui réunit de nombreuses bases de données contenant des images d'objets. Le but de ce site est de faciliter la création de classifieurs en mettant les ressources en images à disposition.

Le principe pour créer un classifier consiste dans un premier temps à récupérer un nombre conséquent d'images d'intérêt. Ensuite ces images sont découpés à l'aide d'un logiciel d'image. Une base de données d'images négatives que l'on ne désire pas détecter est constitué. La base de donnée finale d'images est alors divisé en deux: 70% des images est affectée à l'entraînement, les 30% restantes sont dédiés à la phase de test. En effet, on ne désire pas vérifié le bon fonctionnement du classifieur sur des images sur lesquelles ce classifieur s'est entraîné est a appris.



Ces images sont alors transmis à un programme d'entraînement qui est basé sur le principe de backpropagation. Grâce à ce principe qui utilise les dérivés, à chaque itération, notre réseau de neurones va être soumis à une nouvelle entrée (une nouvelle image) et en comparant la sortie obtenue avec la sortie désirée, le classifieur va pouvoir apprendre par lui même à mieux détecter ou non l'image désirée. Lorsque la phase d'apprentissage est terminée, on soumet alors ce

classifieur à une base de donnée de test afin de s'assurer du bon fonctionnement de ce classifieur.

Cette partie du TP n'a pas pu être traité dans son intégralité car nous n'avons pas eu le temps d'entraîner notre classifieur car cela aurait nécessité un temps conséquent.

Partie 4: détection de visage sous ROS

Ici on cherche à appliquer cette technique avec ROS. Pour cela on met en place un noeud ROS utilisant le package usb_cam qui permet de:

- lire le flux de la caméra
- détecter les visages dans ce flux
- publier sur un topic */faces_detected* les bounding boxes des visages trouvés.

On a choisi de renvoyer directement les coordonnées des boxes autour des visages sous la forme (x1,y1,x2,y2) plutôt que de renvoyer un entier donnant le nombre de visages détectés.

On crée alors deux types de messages:

- BoundingBox qui contient 4 entiers: x1,x2,y1,y2
- FacesDetected:contient un tableau de bouding box

Ainsi, en tapant la commande *rostopic echo /faces_detected*, nous obtenons la sortie à droite à chaque fois qu'un visage est détecté :

```
bboxes:
  -
    x1: 294
    y1: 184
    x2: 521
    y2: 521
---
bboxes:
  -
    x1: 295
    y1: 184
    x2: 520
    y2: 520
---
bboxes:
  -
    x1: 297
    y1: 186
    x2: 520
    y2: 520
---
bboxes:
  -
    x1: 295
    y1: 183
    x2: 521
    y2: 521
---
```

Nous créons ensuite un launchfile afin de faciliter le lancement du nœud ros. Ce launchfile permet de lancer *usb_cam* ainsi que le nœud que nous venons de créer.

Enfin nous créons un package ros permettant d'appliquer les techniques de haar cascade.

Conclusion - Difficultés rencontrées

Dans ce TP nous nous sommes familiariser avec OpenCV à travers l'étude d'une cascade de haar pour de la détection de visage.

Nous n'avons malheureusement pas eu le temps de créer notre propre classifier par manque de temps. En effet, l'entraînement d'un réseau de neurones est un long processus car il nécessite une grande base de donnée afin d'être réellement efficace.