# Etudes des algorithmes

## Introduction

On a exécuté chaque code sur un panel de 10 images trouvées sur Internet représentant plusieurs possibilités de piétons et on a observé les résultats.

A partir de ces images on a choisi l’algo qui a le meilleur taux de réussite.

Une fois vérifié l’exécution correcte de notre algorithme sur Rapsberry on a choisi la meilleure taille d’image possible parmi celles de la caméra.

On a trouvé que la meilleure résolution est 352x288.

Après on calcule le temps d’exécution de l’algorithme sur une image.

On voit que l’algorithme met environ 0.4 secondes pour s’exécuter ce qui est dû à 99% au calcul du HOG.

## Algorithme 1

Source : <http://www.pyimagesearch.com/2015/11/09/pedestrian-detection-opencv/>

Ce code après test est le meilleur. C’est un code Python qui se lance comme cela : detect.py -i images où images est un dossier d’images.

Il utilise OpenCV et imutils. Pour installer OpenCV on a utilisé le tutoriel suivant : <http://www.pyimagesearch.com/2015/10/26/how-to-install-opencv-3-on-raspbian-jessie/>

Pour installer imutils on fait la commande pip install imutils.

Il utilise les descripteurs HOG (histogrammes de gradient orientés) cf <https://fr.wikipedia.org/wiki/Histogramme_de_gradient_orient%C3%A9>

Selon Wikipédia « La technique calcule des [histogrammes](https://fr.wikipedia.org/wiki/Histogramme) locaux de l'orientation du [gradient](https://fr.wikipedia.org/wiki/Gradient) sur une grille dense, c'est-à-dire sur des zones régulièrement réparties sur l'image. »

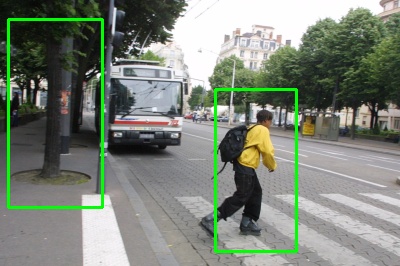
C’est particulièrement conseillé pour les silhouettes humaines.

Puis une « suppression des non-maxima » est réalisée par la fonction « non\_max\_suppression » de imutils pour tenter de supprimer les enveloppes en double sur les personne tout en essayant de conserver des enveloppes dfférentes pour des personnes proches (ce qui ne marche pas à tous les coups).

Nos tests ont été mis dans un fichier Excel.

Nous montrons les images les plus significatives :

Nous voyons des faux-positifs :



Des non-détections exemple :



Ou celle-ci :



Qui n’est d’ailleurs détectée par aucun des algorithmes car la personne est courbée.

## Algorithme 2

Le deuxième algorithme a été pris sur <http://stackoverflow.com/questions/31660271/improving-people-detection-with-opencv> dans la question de [dev\_nut](http://stackoverflow.com/users/692867/dev-nut).

Il est codé en C++ et utilise les technologies HOG et OpenCV

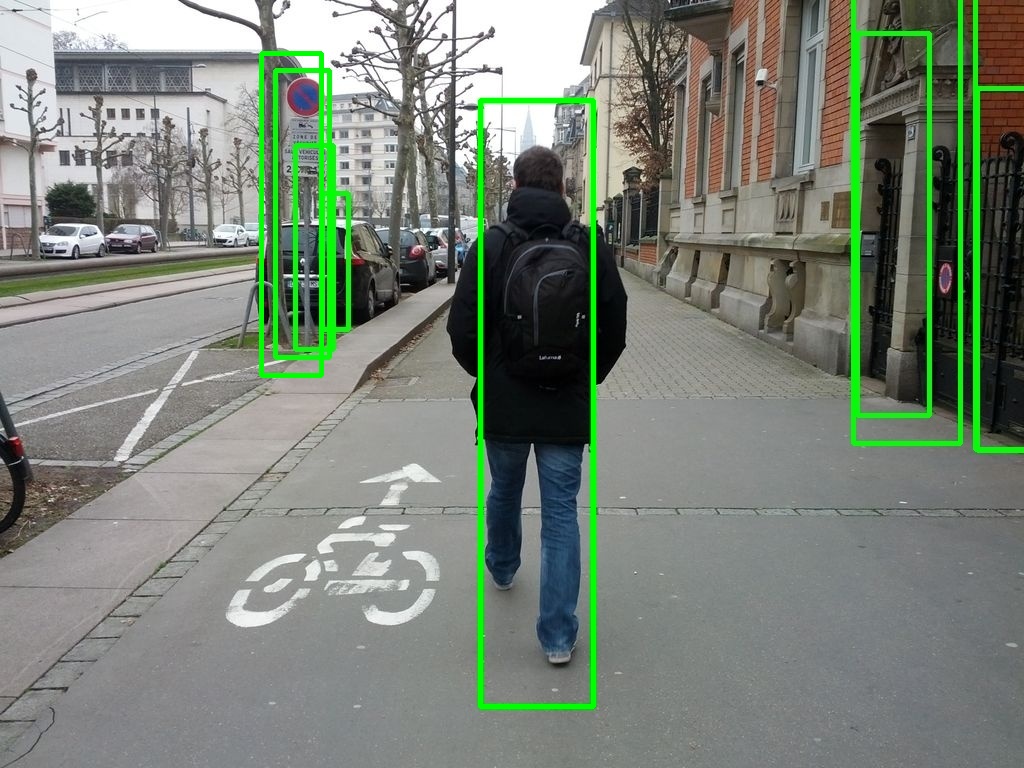


On voit que certaines images n’ont pas été détectées contrairement à l’algorithme 1.

## Algorithme 3

Le troisième et dernier algorithme testé est l’algorithme de la page <http://stackoverflow.com/questions/31660271/improving-people-detection-with-opencv> réponse de [foundry](http://stackoverflow.com/users/1375695/foundry)

On voit qu’il y a plein de faux-positifs



## Conclusion sur les algorithmes HOG

Ces trois algorithmes fonctionnent sur le principe du HOG mais filtrent les rectangles de façon différente mais différemment efficace. Le plus efficace étant le premier.

## Algorithme 4

Ce code a été pris sur la page : <http://funvision.blogspot.fr/2016/03/opencv-31-pedestrian-people-detection.html>

Il utilise une autre technique que le HOG les classifieurs Cascade. J’ai pris les classifieurs du site pour mes tests car je ne dispose pas d’une base de données d’humains assez grande.

Le résultat est néanmoins catastrophique, plein de faux-positifs et de non-détections on l’élimine donc d’office. Le classifieur HOG semble le plus robuste pour la détection d’humains. Exemple de résultat :