# Etudes des algorithmes

Sylvain Ard 26/05/2017

Master RTMA Année 1 Université de Poitiers

## Introduction

Pour notre projet de M1 RTMA à l’Université de Poitiers, nous devions détecter et suivre une silhouette humaine sur une vidéo. Le code doit être porté sur Rapsberry.

On a sélectionné 4 codes sur Internet (certains comme celui-ci : <https://bitbucket.org/rodrigob/doppia> étant plus efficace mais nécessite Cuda qui n’est biensûr pas disponible sur Rapsberry.

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’algorithme 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.

## 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. »

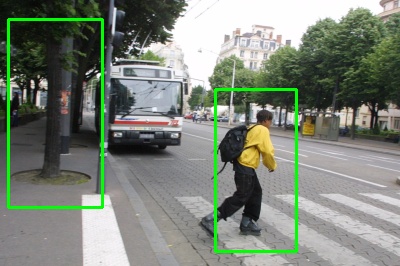
C’est particulièrement conseillé pour les silhouettes humaines.

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 différentes 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.

Code :

# utilisation

# python detect.py --images images

# importe les paquets necessaires

**from** \_\_future\_\_ **import** print\_function

**from** imutils**.**object\_detection **import** non\_max\_suppression

**from** imutils **import** paths

**import** numpy **as** np

**import** argparse

**import** imutils

**import** cv2

# construit le parseur d'arguments et parse les arguments

ap **=** argparse**.**ArgumentParser**()**

ap**.**add\_argument**(**"-i"**,** "--images"**,** required**=True,** help**=**"path to images directory"**)**

args **=** vars**(**ap**.**parse\_args**())**

# initialise le descripteur HOG

hog **=** cv2**.**HOGDescriptor**()**

hog**.**setSVMDetector**(**cv2**.**HOGDescriptor\_getDefaultPeopleDetector**())**

# liste les images du repertoire

imagePaths **=** list**(**paths**.**list\_images**(**args**[**"images"**]))**

**for** imagePath **in** imagePaths**:**#pour chaque image

# charge une image et la redimensionne pour augmenter la detection et diminuer le temps de calcul

image **=** cv2**.**imread**(**imagePath**)**

image **=** imutils**.**resize**(**image**,** width**=**min**(**400**,** image**.**shape**[**1**]))**

orig **=** image**.**copy**()**

# detection des humains

**(**rects**,** weights**)** **=** hog**.**detectMultiScale**(**image**,** winStride**=(**4**,** 4**),**

padding**=(**8**,** 8**),** scale**=**1.05**)**

# filtre les resultats

rects **=** np**.**array**([[**x**,** y**,** x **+** w**,** y **+** h**]** **for** **(**x**,** y**,** w**,** h**)** **in** rects**])**

pick **=** non\_max\_suppression**(**rects**,** probs**=None,** overlapThresh**=**0.65**)**

# dessine le rectangle autour des humains

**for** **(**xA**,** yA**,** xB**,** yB**)** **in** pick**:**

cv2**.**rectangle**(**image**,** **(**xA**,** yA**),** **(**xB**,** yB**),** **(**0**,** 255**,** 0**),** 2**)**

#montre l'image finale

cv2**.**imshow**(**"After NMS"**,** image**)**

cv2**.**waitKey**(**0**)**

## 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.

Code :

//importe les paquets nécessaires

#include "opencv2/imgproc/imgproc.hpp"

#include "opencv2/objdetect/objdetect.hpp"

#include "opencv2/highgui/highgui.hpp"

#include <stdio.h>

#include <fstream>

#include <iostream>

#include <ctype.h>

#include <iostream>

#include <dirent.h>

#include <sys/types.h>

#include <string>

**using** **namespace** cv**;**

**using** **namespace** std**;**

int main**(**int argc**,** char**\*\*** argv**)**

**{**

DIR **\***dir **=** opendir**(**"../images/"**);**

struct dirent **\***entry **;**

**while(**entry **=** readdir**(**dir**))** **{** //pour chaque image du répertoire ../images

char filename**[**300**];**

string s**(**filename**);**

**if** **((**strcmp**(**entry**->**d\_name**,**"."**)==**0**)||(**strcmp**(**entry**->**d\_name**,**".."**)==**0**))**

**continue;**

Mat img**=**imread**(**s**,**CV\_LOAD\_IMAGE\_COLOR**);**//lit l'image

vector**<**Rect**>** found**,** found\_filtered**;**

HOGDescriptor hog**;**

hog**.**setSVMDetector**(**HOGDescriptor**::**getDefaultPeopleDetector**());/X**

namedWindow**(**"people detector"**,** 1**);**

hog**.**detectMultiScale**(**img**,** found**,** 0**,** Size**(**8**,** 8**),** Size**(**32**,** 32**),** 1.05**,** 2**);**//détecte la personne

//filtre complexe des résultats que je ne me suis pas fatigué à comprendre étant donné que le résultat n'est pas satisfaisant

size\_t i**,** j**;**

**for** **(**i **=** 0**;** i **<** found**.**size**();** i**++)**

**{**

Rect r **=** found**[**i**];**

**for** **(**j **=** 0**;** j **<** found**.**size**();** j**++)**

**if** **(**j **!=** i **&&** **(**r **&** found**[**j**])** **==** r**)**

**break;**

**if** **(**j **==** found**.**size**())**

found\_filtered**.**push\_back**(**r**);**

**}**

//dessine des rectangles autour des personnes

**for** **(**i **=** 0**;** i **<** found\_filtered**.**size**();** i**++)**

**{**

Rect r **=** found\_filtered**[**i**];**

// le descripteur HOG mettant des rectangles un peu trop grands on les réduit

r**.**x **+=** cvRound**(**r**.**width**\***0.1**);**

r**.**width **=** cvRound**(**r**.**width**\***0.8**);**

r**.**y **+=** cvRound**(**r**.**height**\***0.07**);**

r**.**height **=** cvRound**(**r**.**height**\***0.8**);**

rectangle**(**img**,** r**.**tl**(),** r**.**br**(),** cv**::**Scalar**(**0**,** 255**,** 0**),** 3**);**

**}**

imshow**(**"people detector"**,** img**);**

imwrite**(**"detected\_ppl.jpg"**,** img**);**

waitKey**(**0**);**

**}**

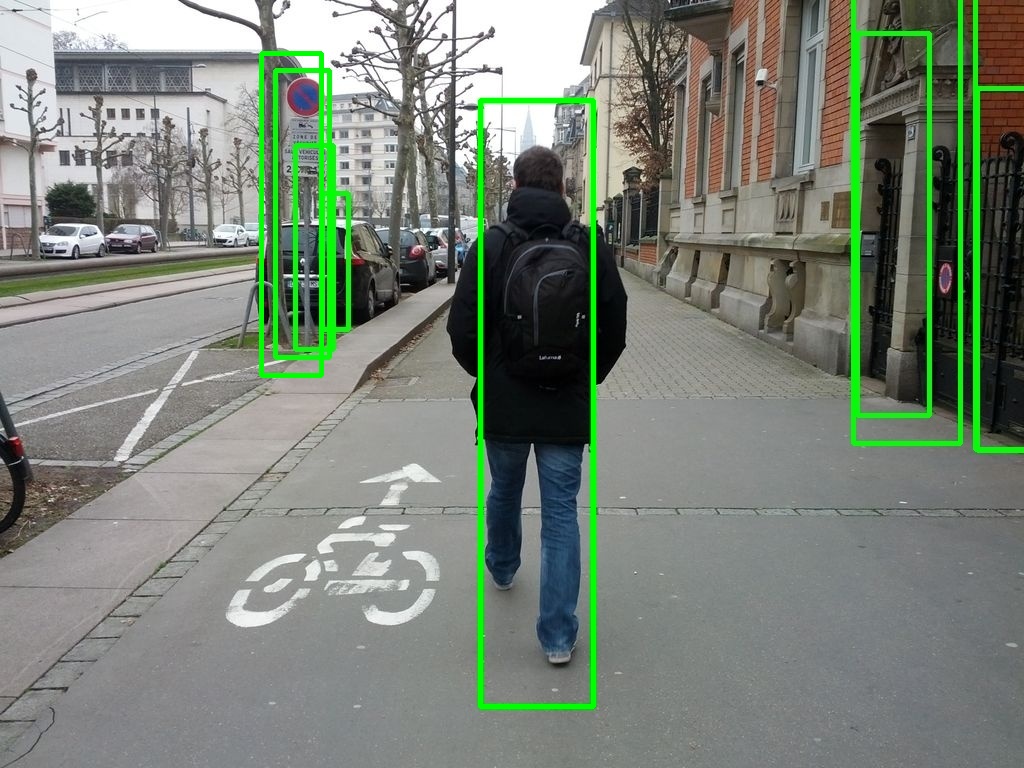
**return** 0**;**

**}**

## 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



Code :

#include "opencv2/imgproc/imgproc.hpp"

#include "opencv2/objdetect/objdetect.hpp"

#include "opencv2/highgui/highgui.hpp"

#include <fstream>

#include <iostream>

#include <ctype.h>

#include <iostream>

#include <dirent.h>

#include <sys/types.h>

#include <stdio.h>

#include <string.h>

//#include "openCVUtils.h"

**using** **namespace** cv**;**

**using** **namespace** std**;**

static Mat \_inImage**;**

static Mat \_outImage**;**

static HOGDescriptor \_hog**;**

static const char**\*** \_windowName **=** "people detector"**;**

static float \_hitThreshold **=** 0**;**

static float \_scaleFactor **=** 1.059**;**

static float \_groupThreshold **=** 0.0**;**

static int MAX\_TRACKBAR **=** 200**;**

static int \_trackbar1 **=** \_scaleFactor **\*** MAX\_TRACKBAR**/**2.0**;**

static int imshowWithCaption**(**const char**\*** windowName**,** Mat inImage**,** const char**\*** caption **)**

**{**

putText**(** inImage**,** caption**,**

Point**(** MIN**(**10**,**inImage**.**cols**),** MIN**(**20**,**inImage**.**rows**)),**

FONT\_HERSHEY\_PLAIN**,** 1**,** Scalar**(**0**,** 0**,** 255**)** **);**

imshow**(** windowName**,** inImage **);**

**return** 0**;**

**}**

static void hogDetect **()** **{**

\_inImage**.**copyTo**(**\_outImage**);**

vector**<**Rect**>** found**;**

vector **<**Rect**>** found\_filtered**;**

\_hog**.**detectMultiScale**(**

\_inImage //const Mat&

//Matrix of the type CV\_8U containing an image where objects are detected.

**,** found // vector<Rect>& objects

//Vector of rectangles where each rectangle contains the detected object.

**,** \_hitThreshold // hit\_threshold=0

//hit\_threshold – Threshold for the distance between features and SVM classifying plane. Usually it is 0 and should be specfied in the detector coefficients (as the last free coefficient). But if the free coefficient is omitted (which is allowed), you can specify it manually here.

**,** Size**(**8**,**8**)**//winSize

//win\_stride – Window stride. It must be a multiple of block stride

**,** Size**(**32**,**32**)**//padding

//padding – Mock parameter to keep the CPU interface compatibility. It must be (0,0).

**,** \_scaleFactor // double scale = 1.05

//scale0 – Coefficient of the detection window increase.

//scaleFactor - Parameter specifying how much the image size is reduced at each image scale.

**,** \_groupThreshold //group\_threshold=2

//group\_threshold – Coefficient to regulate the similarity threshold. When detected, some objects can be covered by many rectangles. 0 means not to perform grouping. See groupRectangles() .

**);**

size\_t i**,** j**;**

**for(** i **=** 0**;** i **<** found**.**size**();** i**++** **)**

**{**

Rect r **=** found**[**i**];**

**for(** j **=** 0**;** j **<** found**.**size**();** j**++** **)** **{**

//filter out overlapping rectangles

**if** **(** j**!=**i **)** **{**

Rect iRect **=** r**;**

Rect jRect **=** found**[**j**];**

Rect intersectRect **=** **(**iRect **&** jRect**);**

**if** **(**intersectRect**.**area**()>=**iRect**.**area**()\***0.9**)** **break;**

**}**

**}**

**if(** j **==** found**.**size**()** **)**

found\_filtered**.**push\_back**(**r**);**

**}**

**for(** i **=** 0**;** i **<** found\_filtered**.**size**();** i**++** **)**

**{**

Rect r **=** found\_filtered**[**i**];**

// the HOG detector returns slightly larger rectangles than the real objects.

// so we slightly shrink the rectangles to get a nicer output.

r**.**x **+=** cvRound**(**r**.**width**\***0.5**);**

// hacky shift right by 40px - rects seem to be shifted consistently

r**.**x **+=-**40**;**

r**.**width **=** cvRound**(**r**.**width**\***0.3**);**

r**.**y **+=** cvRound**(**r**.**height**\***0.07**);**

r**.**height **=** cvRound**(**r**.**height**\***0.8**);**

rectangle**(**\_outImage**,** r**.**tl**(),** r**.**br**(),** cv**::**Scalar**(**0**,**255**,**0**),** 3**);**

**}**

//eliminate overlaps

//imshow(\_windowName, \_outImage);

imshow**(**\_windowName**,** \_outImage**);**

**}**

static void hogTrackbarCallback1 **(**int trackbarPos**,** void**\*)** **{**

\_trackbar1 **=** trackbarPos**;**

\_scaleFactor **=** trackbarPos**/**100.0**;**

hogDetect**();**

**}**

static void setupHogWindow**()** **{**

namedWindow**(** \_windowName**,** WINDOW\_AUTOSIZE **);**

createTrackbar**(** "scaleFactor" //– Name of the created trackbar.

**,** \_windowName //– Name of the window that will be used as a parent of the created trackbar.

**,** **&**\_trackbar1 //– Optional pointer to an integer variable whose value reflects the position of the slider. Upon creation, the slider position is defined by this variable.

**,** MAX\_TRACKBAR //– Maximal position of the slider. The minimal position is always 0.

**,** hogTrackbarCallback1 //– Pointer to the function to be called every time the slider changes position. This function should be prototyped as void Foo(int,void\*); , where the first parameter is the trackbar position and the second parameter is the user data (see the next parameter). If the callback is the NULL pointer, no callbacks are called, but only value is updated.

**,** **NULL**// – User data that is passed as is to the callback. It can be used to handle trackbar events without using global variables.

**);**

**}**

int main**(**int argc**,** char**\*\*** argv**)** **{**

DIR **\***dir **=** opendir**(**"../images/"**);**

struct dirent **\***entry **;**

**while(**entry **=** readdir**(**dir**))** **{**

char filename**[**300**];**

strcpy**(**filename**,**"../images/"**);**

cout**<<**filename**<<**endl**;**

strcat**(**filename**,**entry**->**d\_name**);**

cout **<<**filename**<<**endl**;**

string s**(**filename**);**

**if** **((**strcmp**(**entry**->**d\_name**,**"."**)==**0**)||(**strcmp**(**entry**->**d\_name**,**".."**)==**0**))**

**continue;**

\_inImage**=**imread**(**s**,**CV\_LOAD\_IMAGE\_COLOR**);**

//\_hog = HOGDescriptor::HOGDescriptor();

\_hog**.**setSVMDetector**(**HOGDescriptor**::**getDefaultPeopleDetector**());**

namedWindow**(**\_windowName**,** 1**);**

setupHogWindow**();**

hogDetect**();**

waitKey**(**0**);**

**}**

**return** 0**;**

**}**

Code :

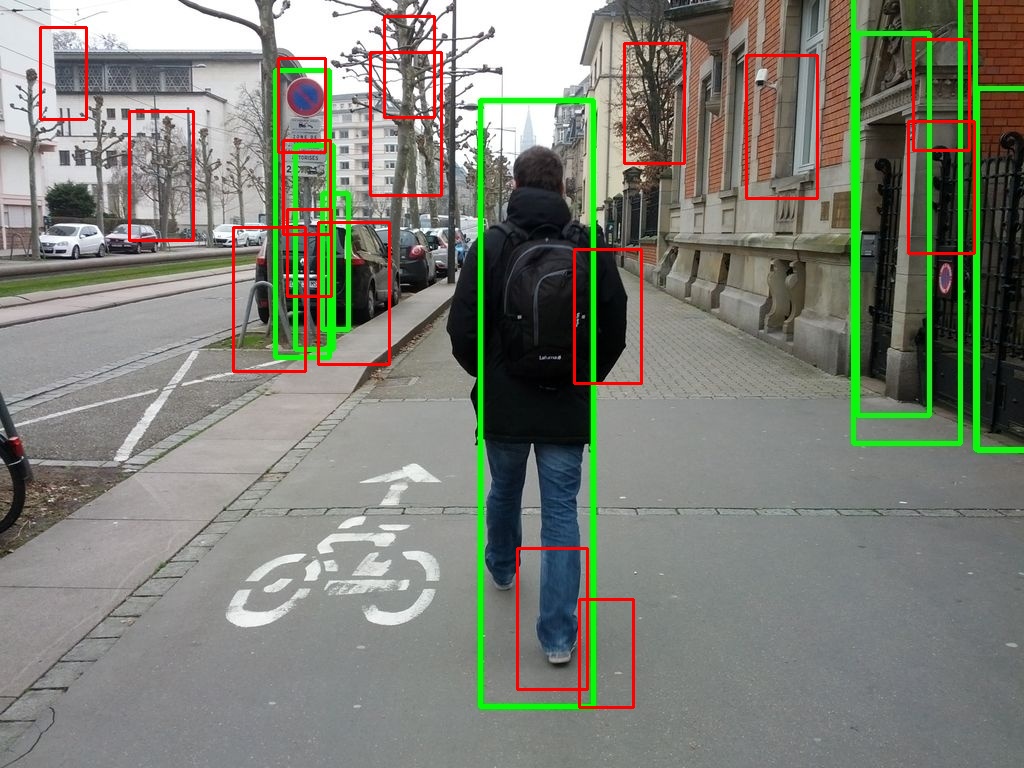
## Conclusion sur les algorithmes HOG

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 :

Code :

#include "opencv2/imgproc/imgproc.hpp"

#include "opencv2/objdetect/objdetect.hpp"

#include "opencv2/highgui/highgui.hpp"

#include <fstream>

#include <iostream>

#include <ctype.h>

#include <iostream>

#include <dirent.h>

#include <sys/types.h>

#include <stdio.h>

#include <string.h>

//#include "openCVUtils.h"

**using** **namespace** cv**;**

**using** **namespace** std**;**

struct MatchPathSeparator

**{**

bool **operator()(** char ch **)** const

**{**

**return** ch **==** '/'**;**

**}**

**};**

std**::**string

gnu\_basename**(** std**::**string const**&** pathname **)**//renvoit le nom d'un fichier par rapport au chemin spécifié

**{**

**return** std**::**string**(**

std**::**find\_if**(** pathname**.**rbegin**(),** pathname**.**rend**(),**

MatchPathSeparator**()** **).**base**(),**

pathname**.**end**()** **);**

**}**

int main**(**int argc**,** char**\*\*** argv**)** **{**

// Name of the downloaded my cascades..

//string cascadeHead = "cascadeH5.xml";

string cascadeName **=** "cascadG.xml"**;**

//string cascadeName="cas.xml";

// Load the cascade

CascadeClassifier detectorBody**;**

bool loaded1 **=** detectorBody**.**load**(**cascadeName**);**

/\* CascadeClassifier detectorHead;

bool loaded2 = detectorHead.load(cascadeHead);\*/

DIR **\***dir **=** opendir**(**"../images/"**);**

struct dirent **\***entry **;**

**while(**entry **=** readdir**(**dir**))** **{**

char filename**[**300**];**

string s**(**filename**);**

**if** **((**strcmp**(**entry**->**d\_name**,**"."**)==**0**)||(**strcmp**(**entry**->**d\_name**,**".."**)==**0**))**

**continue;**

Mat img**=**imread**(**s**,**CV\_LOAD\_IMAGE\_COLOR**);**

// save original make img gray

// draw rectangle back to the original colored sample

Mat original**;**

img**.**copyTo**(**original**);**

// Prepare vector for results

vector**<**Rect**>** human**;**

vector**<**Rect**>** head**;**

// Prepare gray image

cvtColor**(**img**,** img**,** CV\_BGR2GRAY**);**

// equalize Histogram

equalizeHist**(**img**,** img**);**

// detect body and head in the img

// Set the proper min and max size for your case

detectorBody**.**detectMultiScale**(**img**,** human**,** 1.04**,** 4**,** 0 **|** 1**,** Size**(**30**,** 80**),** Size**(**80**,**200**));**

**if** **(**human**.**size**()** **>** 0**)** **{**

**for** **(**int gg **=** 0**;** gg **<** human**.**size**();** gg**++)** **{**

rectangle**(**original**,** human**[**gg**].**tl**(),** human**[**gg**].**br**(),** Scalar**(**0**,** 0**,** 255**),** 2**,** 8**,** 0**);**

**}**

**}**

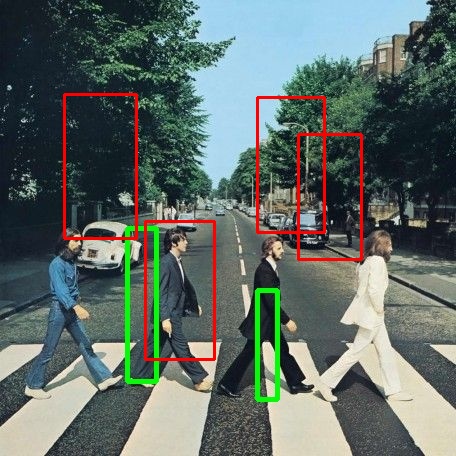
imshow**(**"image"**,** original**);**

imwrite**(**"images\_cascade/"**+**gnu\_basename**(**s**),**original**);**

waitKey**(**0**);**

**}**

**}**



Enfin j’ai réalisé un code détectant une silhouette puis détectant les points d’intérêt dedans puis les suivant et quand ils ont disparus redétectant la silhouette. Pour cela je me suis inspiré du code lkdemo.py disponible ici : <https://github.com/npinto/opencv/blob/master/samples/python/lkdemo.py>, j’y ai intégré le code choisi de reconnaissance de silhouettes, j’ai repris le mécanisme de suivi des points d’intérêt vu en cours avec François Lecellier et le reste est de moi.

En voici le code :

**from** \_\_future\_\_ **import** print\_function

**from** imutils**.**object\_detection **import** non\_max\_suppression

**from** imutils **import** paths

**import** imutils

**import** numpy **as** np

**import** time

**import** cv2

**import** math

**def** silhouette**(**image**):**

# initialize the HOG descriptor/person detector

hog **=** cv2**.**HOGDescriptor**()**

hog**.**setSVMDetector**(**cv2**.**HOGDescriptor\_getDefaultPeopleDetector**())**

image **=** imutils**.**resize**(**image**,** width**=**min**(**400**,** image**.**shape**[**1**]))**

orig **=** image**.**copy**()**

# detect people in the image

**(**rects**,** weights**)** **=** hog**.**detectMultiScale**(**image**,** winStride**=(**4**,** 4**),**padding**=(**8**,** 8**),** scale**=**1.05**)**

# draw the original bounding boxes

**for** **(**x**,** y**,** w**,** h**)** **in** rects**:**

cv2**.**rectangle**(**orig**,** **(**x**,** y**),** **(**x **+** w**,** y **+** h**),** **(**0**,** 0**,** 255**),** 2**)**

# apply non-maxima suppression to the bounding boxes using a

# fairly large overlap threshold to try to maintain overlapping

# boxes that are still people

rects **=** np**.**array**([[**x**,** y**,** x **+** w**,** y **+** h**]** **for** **(**x**,** y**,** w**,** h**)** **in** rects**])**

pick **=** non\_max\_suppression**(**rects**,** probs**=None,** overlapThresh**=**0.65**)**

**return** pick**,** image

**def** carre**(**x**):**

**return** x**\***x**;**

**def** afficheImage**(**img**):**

img **=** imutils**.**resize**(**img**,** width**=**min**(**400**,** img**.**shape**[**1**]))**

cv2**.**imshow**(**"Video"**,** img**)**

cv2**.**waitKey**(**1**)**

cap **=** cv2**.**VideoCapture**(**'test2.mov'**)**

# params for ShiTomasi corner detection

feature\_params **=** dict**(** maxCorners **=** 500**,**

qualityLevel **=** 0.01**,**

minDistance **=** 10**,**

blockSize **=** 3 **)**

# Parameters for lucas kanade optical flow

lk\_params **=** dict**(** winSize **=** **(**31**,**31**),**

maxLevel **=** 3**,**

criteria **=** **(**cv2**.**TERM\_CRITERIA\_EPS **|** cv2**.**TERM\_CRITERIA\_COUNT**,** 20**,** 0.03**))**

# Create some random colors

color **=** np**.**random**.**randint**(**0**,**255**,(**100**,**3**))**

**for** i **in** range**(**1**,**100**):**#passe les 100 premières frames du fim le tempq que la personne arrive

ret**,** frame **=** cap**.**read**()**

ret**,** frame **=** cap**.**read**()**#on capte la premiere frame

frame **=** imutils**.**resize**(**frame**,** width**=**min**(**400**,** frame**.**shape**[**1**]))**#retaille la frame a 400

gray **=** cv2**.**cvtColor**(**frame**,** cv2**.**COLOR\_BGR2GRAY**)**#on calcule son image en niveaux de gris

a\_silhouette**=**0#indique si on a trouve une silhouette

dr\_image\_avec\_silhouette**=**gray**.**copy**()**#derniere image contenant une silhouette que l'on initialise arbitrairement a gray

ii**=**0

**while(**1**):**#boucle infiniment

ii**=**ii**+**1

k **=** cv2**.**waitKey**(**30**)** **&** 0xff

**if** k **==** 27**:**#si on a pressé Echap

**break**#quitte

**print(**"frame "**+**str**(**ii**))**

#on stocke l'ancienne frame en niveaux de gris dans old\_frame et l'ancienne frame dans old\_frame

old\_frame**=**frame**.**copy**()**

old\_gray**=**gray**.**copy**()**

ret**,** frame **=** cap**.**read**()**#lit une frame de la vidéo

frame **=** imutils**.**resize**(**frame**,** width**=**min**(**400**,** frame**.**shape**[**1**]))**#redimensionne la frame à 400 pour augmenter la rapidité et la précision de la détection

img**=**frame

gray **=** cv2**.**cvtColor**(**frame**,** cv2**.**COLOR\_BGR2GRAY**)**

**if** **not** a\_silhouette**:**#si pas de silhouette on en recherche une

pick**,** image\_**=**silhouette**(**frame**)**#on recherche une silhouette

**if** **(**len**(**pick**)==**0**):**#si on en trouve toujours pas

afficheImage**(**img**)**

**continue;**#on passe a la frame suivante

**else:**#sinon

#on dessine le rectangle de la silhouette

**for** **(**xA**,** yA**,** xB**,** yB**)** **in** pick**:**

cv2**.**rectangle**(**img**,** **(**xA**,** yA**),** **(**xB**,** yB**),** **(**0**,** 255**,** 0**),** 2**)**

a\_silhouette**=**1

dr\_image\_avec\_silhouette**=**gray**.**copy**()**

#s'il ya une silhouette on cherche des points d'interet sur la silhouette

#on cree un masque de 0 pour tout refuser

mask\_**=**np**.**zeros\_like**(**gray**)**#zeros\_like prend la meme largeur, hauteur et type que frame et met des 0 dedans

#on dessine des rectangles blancs sur les silhouettes pour autoriser la recherche de silhouette dessus

**for** **(**xA**,** yA**,** xB**,** yB**)** **in** pick**:**

cv2**.**rectangle**(**mask\_**,** **(**xA**,** yA**),** **(**xB**,** yB**),** 255**,** 2**)**

#on calcule les points d'interet que sur la silhouette grace au masque

p0 **=** cv2**.**goodFeaturesToTrack**(**gray**,** mask **=** mask\_**,** **\*\***feature\_params**)**

afficheImage**(**img**)**

**print(**"silhouette trouvee !"**)**

**continue**#force a calculer la frame suivante

**if** a\_silhouette**:**

#calcule le flot optique

p1**,** st**,** err **=** cv2**.**calcOpticalFlowPyrLK**(**old\_gray**,** gray**,** p0**,** **None,** **\*\***lk\_params**)**

#on enleve les points d'interets qui n'ont pas beaucoup bouges

**for** i **in** range**(**len**(**p0**)):**

**if** st**[**i**]:**

**if** math**.**sqrt**(**carre**(**p1**[**i**][**0**][**0**]-**p0**[**i**][**0**][**0**])+**carre**(**p1**[**i**][**0**][**1**]-**p0**[**i**][**0**][**1**]))<**0.5**:**#le 0.5 peut être modifié

st**[**i**]=**0

#cree un masque pour cacher les nouveaux points d'interets

mask\_**=**np**.**zeros\_like**(**gray**)**

mask\_**[:]=**255

centre**=(**0**,**0**)**#le centre des points d'interets

**for** i **in** range**(**len**(**p1**)):**#nouveaux points d'interets

cv2**.**circle**(**mask\_**,(**np**.**int32**(**p1**[**i**][**0**][**0**]),**np**.**int32**(**p1**[**i**][**0**][**1**])),**5**,**0**,-**1**)**

centre**=(**centre**[**0**]+**np**.**int32**(**p1**[**i**][**0**][**0**]),**centre**[**1**]+**np**.**int32**(**p1**[**i**][**0**][**1**]))**

centre**=(**centre**[**0**]/**len**(**p1**),**centre**[**1**]/**len**(**p1**))**

**if** centre**[**0**]>**270**:**

**print(** "tourner a droite"**)**

**if** centre**[**0**]<**130**:**

**print** **(**"tourner a gauche"**)**

p0 **=** cv2**.**goodFeaturesToTrack**(**gray**,** mask **=** mask\_**,** **\*\***feature\_params**)**#recalcule es points d'interets

#on calcule la somme des status (qui valent 0 dans le cas des points d'interets disparus et 1 dans le cas des points d'interet suivis)

somme**=**np**.**ndarray**.**sum**(**st**)**

**if** **(**somme**==**0**):**#si tous les points d'interet ont disparus

a\_silhouette**=**0

afficheImage**(**img**)**

**print(**"tous les points d'interet nt disparus !"**)**

**continue**#on force la calcul d'une autre frame pour rechercher une silhouette

# Select good points

good\_new **=** p1**[**st**==**1**]**

good\_old **=** p0**[**st**==**1**]**

# Create a mask image for drawing purposes

mask **=** np**.**zeros\_like**(**old\_frame**)**

# draw the tracks

**for** i**,(**new**,**old**)** **in** enumerate**(**zip**(**good\_new**,**good\_old**)):**

a**,**b **=** new**.**ravel**()**

c**,**d **=** old**.**ravel**()**

cv2**.**circle**(**img**,(**a**,**b**),**5**,**color**[**i**].**tolist**(),-**1**)**#on affiche les points d'interets suivis

afficheImage**(**img**)**

p0 **=** good\_new**.**reshape**(-**1**,**1**,**2**)**

cv2**.**destroyAllWindows**()**

cap**.**release**()**

## Bibliographie succinte conseillée :

 Yannick Benezeth, Détection de la présence humaine par vision : Thèse, [Université d'Orléans](https://fr.wikipedia.org/wiki/Universit%C3%A9_d%27Orl%C3%A9ans), 2009 ([lire en ligne](ftp://ftp.univ-orleans.fr/theses/yannick.benezeth_1908.pdf) [[archive](http://archive.wikiwix.com/cache/?url=ftp%3A%2F%2Fftp.univ-orleans.fr%2Ftheses%2Fyannick.benezeth_1908.pdf)])

 [*"Histograms of Oriented Gradients for Human Detection"*](http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf) (PDF). p. 2.

 [*"Histograms of Oriented Gradients for Human Detection"*](http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf) (PDF). p. 4.

 [*"Histograms of Oriented Gradients for Human Detection"*](http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf) (PDF). p. 5.

 [*"Histograms of Oriented Gradients for Human Detection"*](http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf) (PDF). p. 6.

 [*"Fast Human Detection Using a Cascade of Histograms of Oriented Gradients"*](http://seaboy.tistory.com/attachment/cfile25.uf@116008124A989DCB0E0C23.pdf) (PDF).