---------------------TUTORIEL D’INSTALLATION OPEN CV -----------

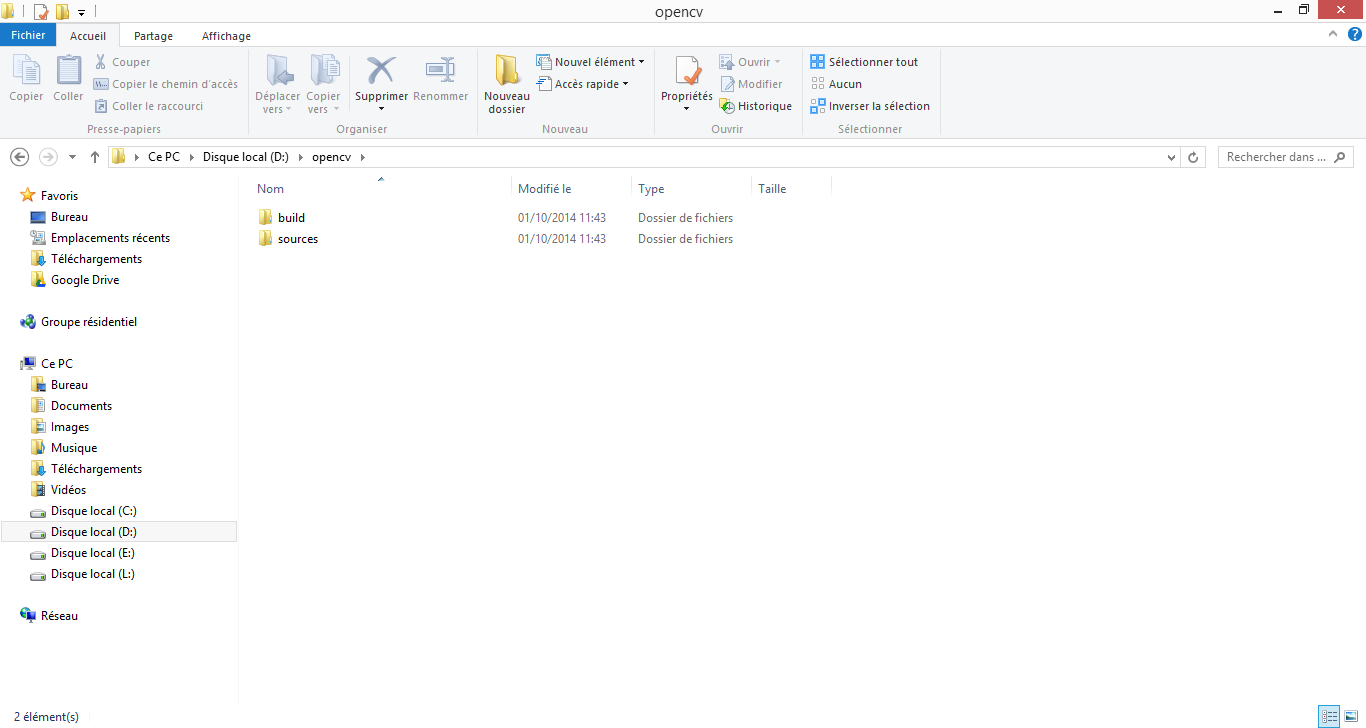
-----------------------------------------WINDOWS 7 / 8 / 8.1---------------------------------

-------------------------------------------64 Bit / 32 Bit OS---------------------------------------

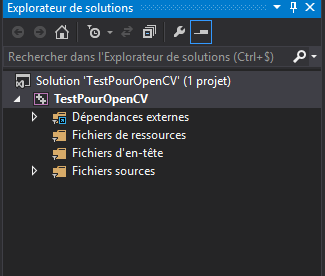
---------------------------------------OPENCV 2.4.10---------------------------------

------------------------------------VISUAL STUDIO 2013---------------------------

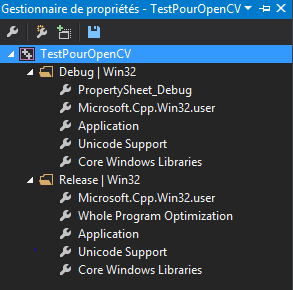
* Installer Visual Studio 2013 Version Ultimate Pro ou Premium
* Télécharger Open CV pour Windows → [ici](http://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.4.10/opencv-2.4.10.exe/download) ← (l’enregistrer sur son PC)
* Extraire les fichiers à la racine de votre disque C:\ ou D:\
  1. L’architecture doit ressembler à ça :



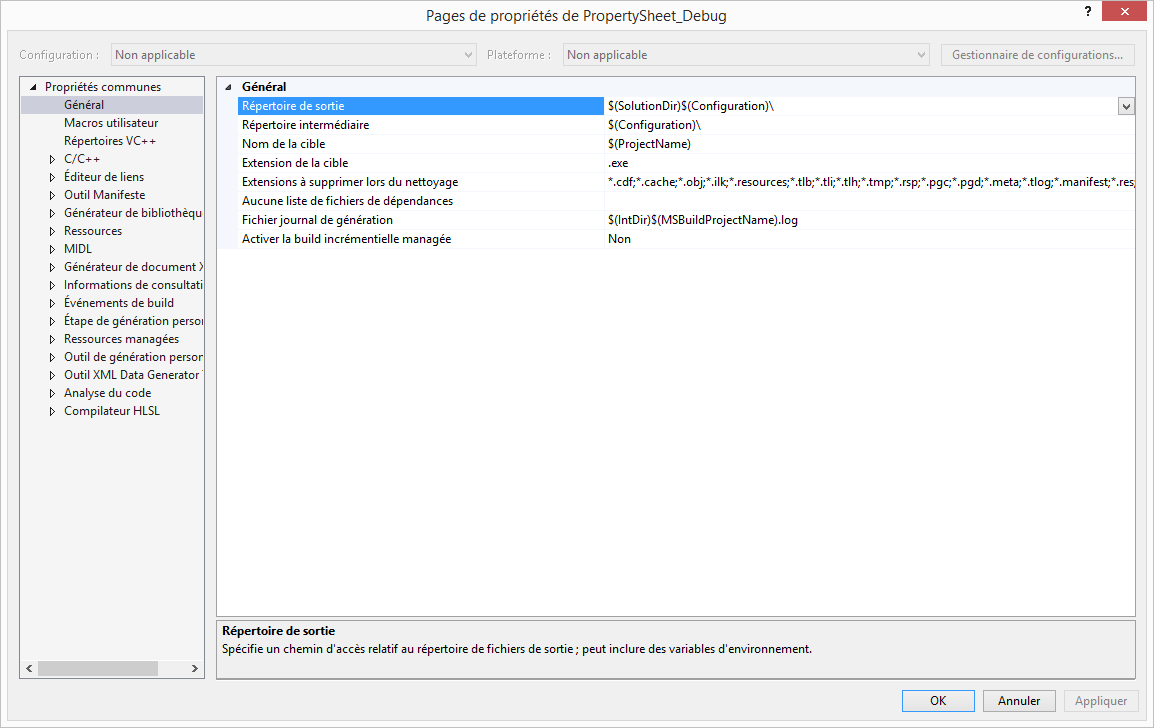
* Créer un nouveau projet Visual studio -> C++ -> Applications console
  1. Sélectionner projet vide
  2. Votre solution ressemble à ca



* Cliquer sur Affichage -> Autres fenêtres -> Gestionnaire de propriétés



* Clic droit sur le dossier Debug -> Ajouter une nouvelle feuille de propriétés de projet.
* Nommer la PropertySheet\_Debug
* Double cliquer sur cette nouvelle feuille de propriété



* Dans Répertoire VC++ -> répertoire d’exécutable -> Ajouter : D:\opencv\build\x86\vc12\bin;

(Chemin à modifier en fonction de l’endroit où est extrait Open CV)

* Dans Répertoire VC++ -> Répertoire de bibliothèques -> Ajouter : D:\opencv\build\x86\vc12\lib; (x86 est la plateforme d’exécution, vc12 pour Visual Studio 2013)
* Dans C/C++ (Général) -> Autres répertoire d’Include -> Ajouter : D:\opencv\build\include
* Dans Editeur de liens (Général) -> répertoire de bibliothèques supplémentaire -> Ajouter : D:\opencv\build\x86\vc12\lib
* Dans Editeur de liens -> Entrée -> Dépendance supplémentaire -> Ajouter l’ensemble des librairies:

opencv\_calib3d2410d.lib

opencv\_contrib2410d.lib

opencv\_core2410d.lib

opencv\_features2d2410d.lib

opencv\_flann2410d.lib

opencv\_gpu2410d.lib

opencv\_highgui2410d.lib

opencv\_imgproc2410d.lib

opencv\_legacy2410d.lib

opencv\_ml2410d.lib

opencv\_nonfree2410d.lib

opencv\_objdetect2410d.lib

opencv\_ocl2410d.lib

opencv\_photo2410d.lib

opencv\_stitching2410d.lib

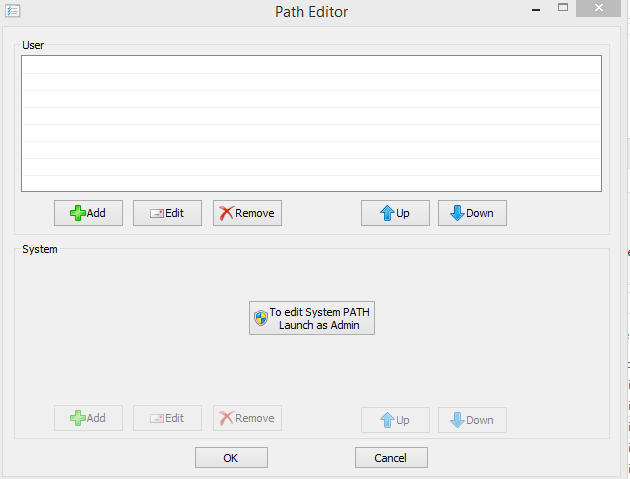
opencv\_superres2410d.lib

opencv\_ts2410d.lib

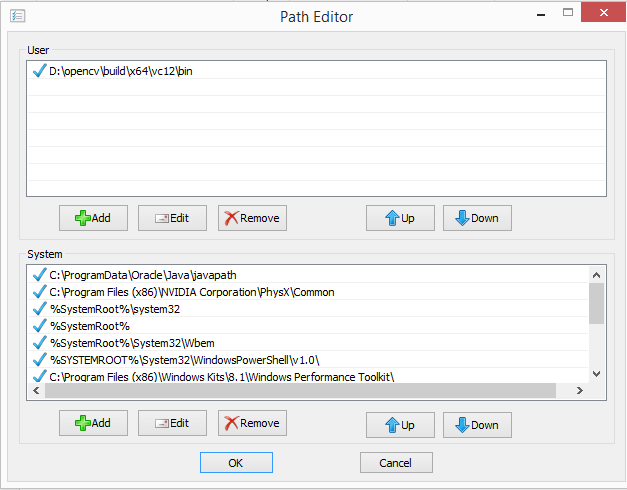
opencv\_video2410d.lib

opencv\_videostab2410d.lib

* Cliquer sur Appliquer puis OK
* Télécharger le logiciel Path Editor → [Ici](https://patheditor2.codeplex.com/) ←
* Une fois le téléchargement terminer -> Exécuter le .exe



* Donner les droits Administrateurs



* Dans la partie du bas (System)
  + Cliquer sur + Add et ajouter : D:\opencv\build\x86\vc12\bin
* Dans votre projet Visual Studio coller ce code de test pour vérifier que tous fonctionne

#include "opencv2/highgui/highgui.hpp"

#include "opencv2/imgproc/imgproc.hpp"

#include "opencv2/objdetect/objdetect.hpp"

#include <iostream>

using namespace cv;

using namespace std;

CascadeClassifier face\_cascade;

CascadeClassifier eye\_cascade;

/\*\*

\* Function to detect human face and the eyes from an image.

\*

\* @param im The source image

\* @param tpl Will be filled with the eye template, if detection success.

\* @param rect Will be filled with the bounding box of the eye

\* @return zero=failed, nonzero=success

\*/

int detectEye(Mat& im, Mat& tpl, Rect& rect)

{

std::vector<Rect> faces, eyes;

face\_cascade.detectMultiScale(im, faces, 1.1, 2, 0 | CV\_HAAR\_SCALE\_IMAGE, Size(30, 30));

for (int i = 0; i < faces.size(); i++)

{

Mat face = im(faces[i]);

eye\_cascade.detectMultiScale(face, eyes, 1.1, 2, 0 | CV\_HAAR\_SCALE\_IMAGE, Size(20, 20));

if (eyes.size())

{

rect = eyes[0] + Point(faces[i].x, faces[i].y);

tpl = im(rect);

}

}

return eyes.size();

}

/\*\*

\* Perform template matching to search the user's eye in the given image.

\*

\* @param im The source image

\* @param tpl The eye template

\* @param rect The eye bounding box, will be updated with the new location of the eye

\*/

void trackEye(Mat& im, Mat& tpl, Rect& rect)

{

Size size(rect.width \* 2, rect.height \* 2);

Rect window(rect + size - Point(size.width / 2, size.height / 2));

window &= Rect(0, 0, im.cols, im.rows);

Mat dst(window.width - tpl.rows + 1, window.height - tpl.cols + 1, CV\_32FC1);

matchTemplate(im(window), tpl, dst, CV\_TM\_SQDIFF\_NORMED);

double minval, maxval;

Point minloc, maxloc;

minMaxLoc(dst, &minval, &maxval, &minloc, &maxloc);

if (minval <= 0.2)

{

rect.x = window.x + minloc.x;

rect.y = window.y + minloc.y;

}

else

rect.x = rect.y = rect.width = rect.height = 0;

}

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

{

// Load the cascade classifiers

// Make sure you point the XML files to the right path, or

// just copy the files from [OPENCV\_DIR]/data/haarcascades directory

face\_cascade.load("D:/opencv/sources/data/haarcascades/haarcascade\_frontalface\_alt2.xml");

eye\_cascade.load("D:/opencv/sources/data/haarcascades/haarcascade\_eye.xml");

VideoCapture cap(0); // open the video camera no. 0

if (face\_cascade.empty() || eye\_cascade.empty() || !cap.isOpened()) // if not success, exit program

{

cout << "Cannot open the video cam" << endl;

return -1;

}

cap.set(CV\_CAP\_PROP\_FRAME\_WIDTH, 320);

cap.set(CV\_CAP\_PROP\_FRAME\_HEIGHT, 240);

double dWidth = cap.get(CV\_CAP\_PROP\_FRAME\_WIDTH); //get the width of frames of the video

double dHeight = cap.get(CV\_CAP\_PROP\_FRAME\_HEIGHT); //get the height of frames of the video

cout << "Frame size : " << dWidth << " x " << dHeight << endl;

namedWindow("MyVideo", CV\_WINDOW\_AUTOSIZE); //create a window called "MyVideo"

Mat frame, eye\_tpl;

Rect eye\_bb;

while (waitKey(15) != 'q')

{

cap >> frame;

if (frame.empty())

break;

// Flip the frame horizontally, Windows users might need this

flip(frame, frame, 1);

// Convert to grayscale and

// adjust the image contrast using histogram equalization

Mat gray;

cvtColor(frame, gray, CV\_BGR2GRAY);

if (eye\_bb.width == 0 && eye\_bb.height == 0)

{

// Detection stage

// Try to detect the face and the eye of the user

detectEye(gray, eye\_tpl, eye\_bb);

}

else

{

// Tracking stage with template matching

trackEye(gray, eye\_tpl, eye\_bb);

// Draw bounding rectangle for the eye

rectangle(frame, eye\_bb, CV\_RGB(255, 100, 125));

}

// Display video

cv::imshow("MyVideo", frame);

}

return 0;

}