TSHALA
BENJAMIN

M1 TSI/EEA

Utilisation de la bibliothèque OPENCV

Objectif

Le but est de comprendre la bibliothèques Opencv et son utilisation

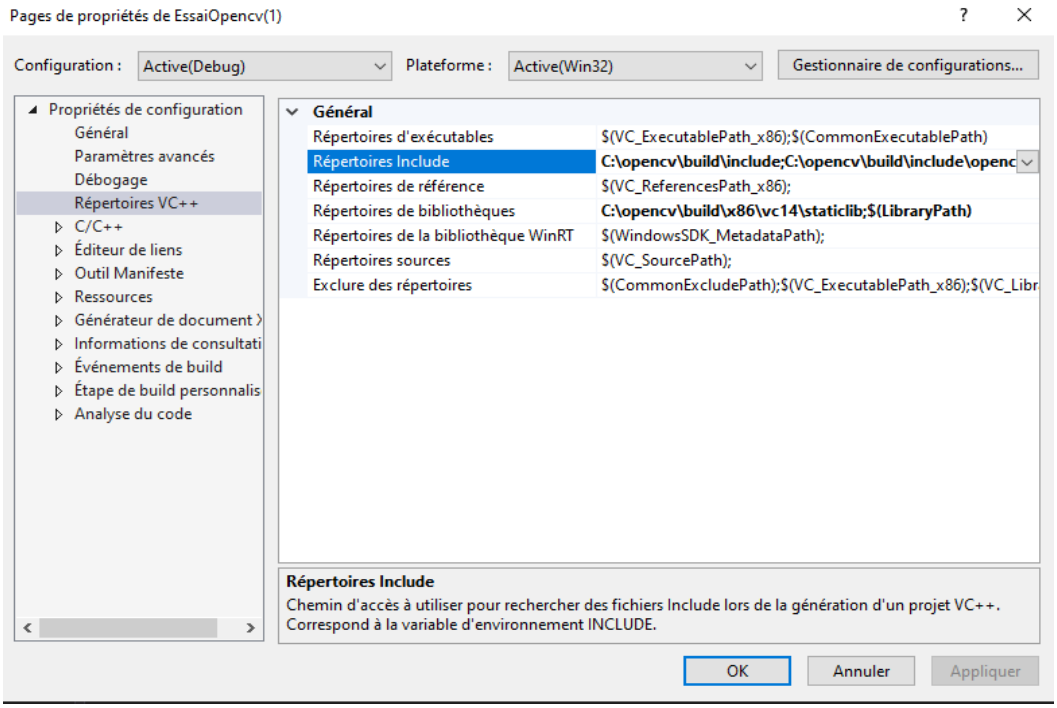
Analyse d'un projet Opencv

Après avoir Téléchargez, décompresser et ouvert le projet « DemoOpencvStaticLib » On se rend compte de l'existence de 2 classes qui sont :

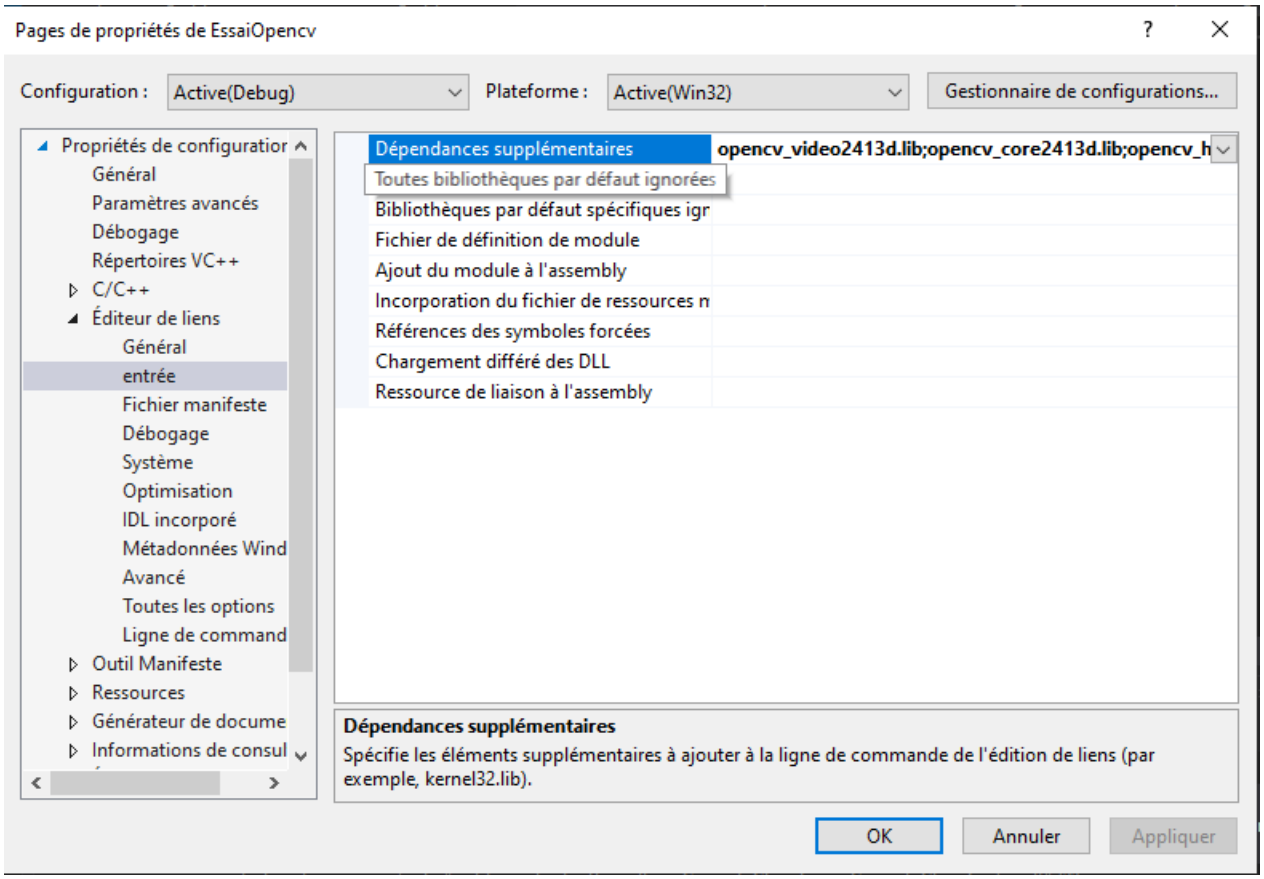
1. EssaiOpencv
2. EssaiOpencvDlg

L'ouverture des propriétés du projet nous permet de voir les Include et bibliothèque ajoutée pour l'utilisation d'Opencv :

1. Include C:\opencv\build\include ; C:\opencv\build\include\opencv ; $(IncludePath) »
2. Bibliothèques C:\opencv\build\x86\vc14\staticlib ; $(LibraryPath)



Dans l'éditeur de lien, sous menu entrée, des dépendances supplémentaires ont été ajoutées « opencv_core2413d.lib;opencv_highgui2413d.lib;opencv_imgproc2413d.lib;IlmImfd.lib;libtiffd.lib;libpngd.lib;libjpegd.lib;libjasperd.lib;zlibd.lib;vfw32.lib;%(AdditionalDependencies) »



Les dépendances en mode debug ne sont pas les mêmes que pour le mode release

Using namespace cv : nous permet d'inclure les fonctions qui se trouve dans cv.

Apres l'exécution du programme on se rend compte qu'il permet de détecter les bords

Figure 1: détection des bords

1. Application de détection des cercles sur l'mage grâce à la transformé de Hough

1. La Transformation de Hough

La transformation de Hough est une technique utilisée pour détecter des formes géométriques, telles que des lignes, des cercles ou des ellipses, dans une image. Elle est particulièrement efficace pour détecter des objets dont la forme est définie par une équation mathématique.

Dans le cadre de la détection de cercles, la transformation de Hough fonctionne en transformant l'espace des pixels de l'image en un espace paramétrique, où chaque point dans cet espace représente un cercle potentiel. Les cercles dans l'image originale sont détectés en identifiant des intersections dans cet espace paramétrique.

Étapes de la Transformation de Hough pour les Cercles :

1. **Prétraitement** : L'image est souvent prétraitée pour améliorer la détection des bords, par exemple avec un filtre gaussien pour réduire le bruit.
2. **Détection des bords** : La détection des bords est généralement effectuée avec un algorithme comme Canny, qui met en évidence les contours dans l'image.

3. **Transformation de Hough** : Chaque pixel du bord détecté est transformé en une courbe dans l'espace des paramètres, représentant tous les cercles possibles qui pourraient passer par ce point.
4. **Accumulateur** : Un accumulateur est utilisé pour enregistrer combien de courbes passent par chaque point dans l'espace des paramètres. Les points où le plus grand nombre de courbes se croisent indiquent la présence d'un cercle.
5. **Détection des cercles** : Les centres des cercles sont déterminés à partir des intersections maximales dans l'accumulateur, et les rayons sont calculés en fonction de la distance des points de bord au centre.

Cette technique est robuste pour détecter des cercles dans des images bruitées ou partiellement obstruées.

Dans cette partie du programme on ouvre la caméra et on déclare un « frame » qui va recevoir des trames de la caméra, puis on applique un filtre gaussien qui va flouter notre image et rendre la détection de cercle plus précise.

```
void CEssaiOpencvDlg::OnBnClickedButtonFiltre()
{

VideoCapture cap(0); // ouvrir la camera
	if (!cap.isOpened())  // vérifier si la camera est ouverte
		return;


	for (;;)
	{
		Mat frame;
		cap >> frame; // nouvelle trame de la camera
		GaussianBlur(frame, frame, Size(9, 9), 2, 2); //application
	du filtre gaussien
```

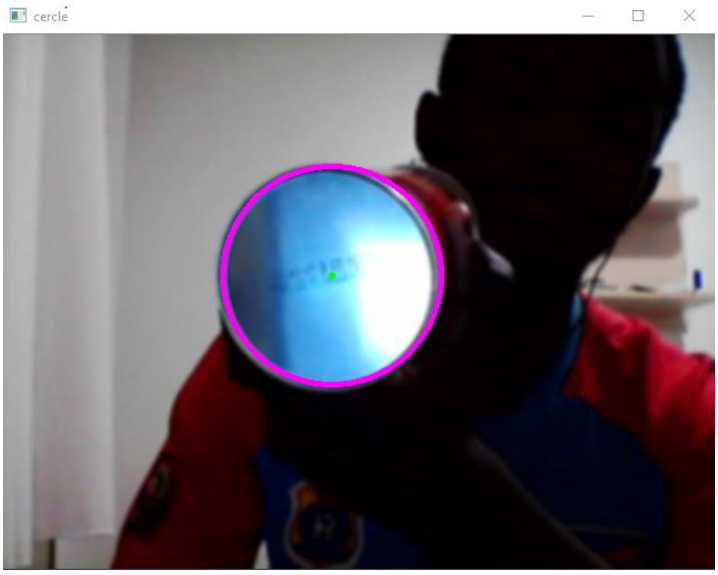
Dans cette partie du programme on applique la transformé de Hough pour détecter les cercles sur la trame, puis dans la boucle for on dessine les cercles détecter.

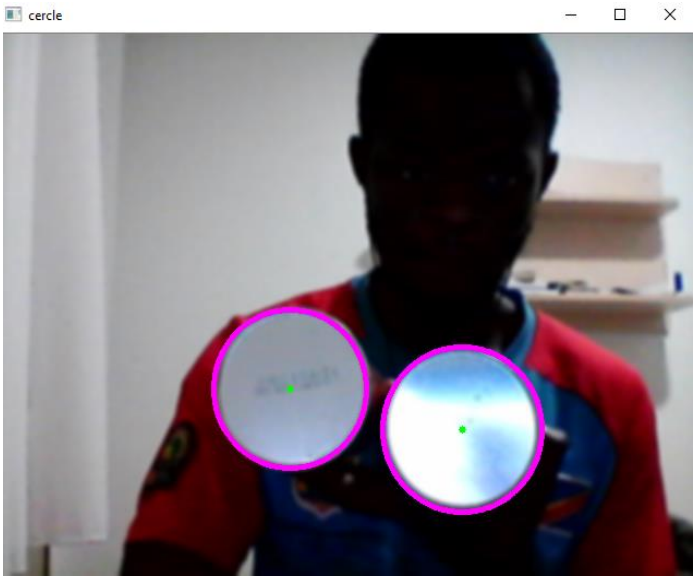
```
vector<Vec3f> circles;

		/// on applique sur la trame la transformé de hough pour trouver les cercles
		HoughCircles(frame, circles, CV_HOUGH_GRADIENT, 1, frame.rows / 8, 200, 100, 0, 0);

		/// dessiner les cercles detecter
		for (int i = 0; i < circles.size(); i++)
		{
			printf("are you um?\n");
			Point center(cvRound(circles[i][0]),cvRound(circles[i][1]));
			int radius = cvRound(circles[i][2]);
			// centre du cercle, point couleur vert
			circle(frame, center, 3, Scalar(0, 255, 0), -1, 8, 0);
			// cercle contour couleur rose
			circle(frame, center, radius, Scalar(255, 0, 255), 3, 8, 0);
		}
```

Finalement on affiche et on voit le résultat.

```
imshow("cercle", frame); //affichage de la fenêtre


		if (waitKey(0x1B) >= 0) break;
	}

}
```

Et pour désactiver l'affichage on clique sur le bouton « ESC ».





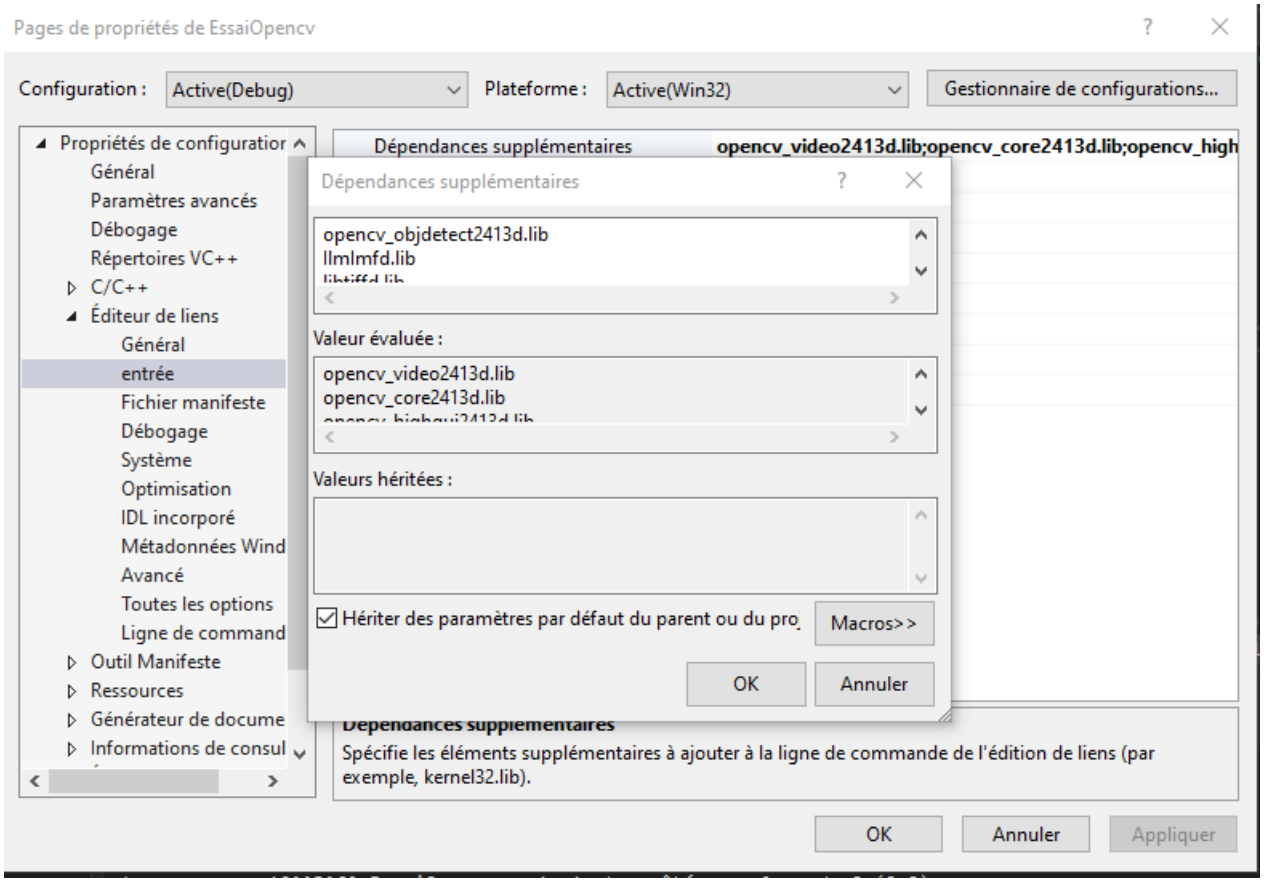
Comme on peut le voir sur les images on est capable de détecter avec la caméra les cercles existants.

2. Implémentions d'une localisation de visages (cascade de Haar).

Dans cette partie on va utilise la cascade de haar pour effectuer la détection des visages, tout d'abord on commence par faire appel au bibliothèques nécessaire pour effectuer cette fonction.

```
#include<opencv2\highgui\highgui.hpp>
#include<opencv2\imgproc\imgproc.hpp>
#include<opencv2\objdetect\objdetect.hpp>
```

Puis on ajoute les dépendances supplémentaires nécessaire.



Pour cet exercice on ajoute la dépendance opencv_objdetect2413.lib.

Dans notre classe « face.h » on déclare une méthode de type « void » qui prend en paramètre une variable pour contenir l'image et une variable de type cascadeClassifier d'objdetect.

```
void detection(Mat img);

CascadeClassifier faceDetection;
```

Dans la fonction « detection » on déclare un vecteur qu'on nomme face qui va prendre la liste d'objet détecter de différentes tailles dans l'image d'entré sous forme de rectangle lors de l'utilisation de « detectMultiScale » puis on lance une boucle « for » pour pouvoir tracer un rectangle de couleur rouge sur notre image.

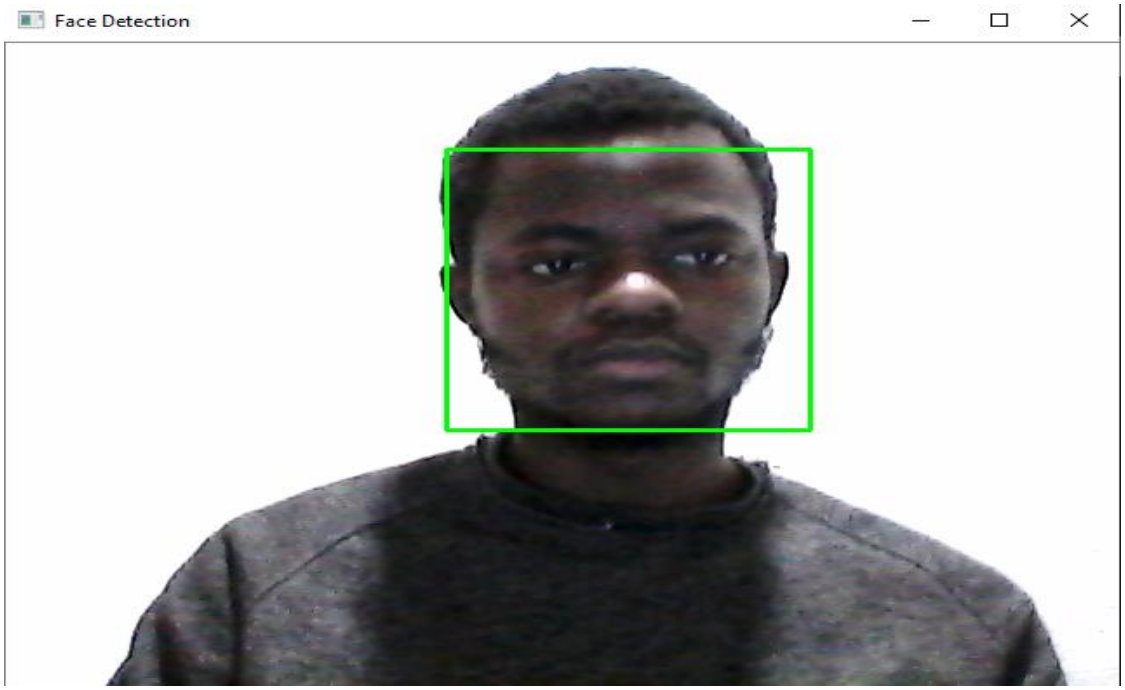
```
void  detection(Mat img )
{
	vector<Rect> faces;

	faceDetection.detectMultiScale(img, faces);

	for (unsigned int i = 0; i < faces.size(); i++) {
		Point pt1(faces[i].x, faces[i].y);
		Point pt2((faces[i].x + faces[i].height), (faces[i].y +
faces[i].width));

		rectangle(img, pt1, pt2, Scalar(0, 255, 0), 2, 8, 0);

	}
	imshow("Face Detection", img);

}
```

Dans notre main on charge tout d'abord « `haarcascade_frontalface_default.xml` » pour la détection du visage uniquement

```
if
(!faceDetection.load("C:\\Opencv\\sources\\data\\haarcascades\\haarcascad
e_frontalface_default.xml")) {
            cout << "\n XML File not found";
            exit(0);
      }

      VideoCapture cap(0);
      if (!cap.isOpened())
            return 1;
```

Puis finalement on fait appel a notre fonction detection dans la « main »

```
Mat edges;
      for (;;)
      {

            Mat frame;
            cap >> frame;
            cvtColor(frame, edges, CV_RGB2GRAY);

            vector<Rect> faces, faces2;
            Mat gray, smallImg;

            detectFaces(frame);


            if (waitKey(0x1B) >= 0) break;
      }

      return 1;
}
```



Finalement lors de l’execution denotre programme on peut voir un carré de couleur verte autour du visage

Conclusion

Ce rapport a exploré l'utilisation de la bibliothèque OpenCV pour la détection de cercles et de visages dans des images. À travers l'analyse et la mise en œuvre de deux projets spécifiques, nous avons pu démontrer l'efficacité de certaines fonctionnalités d'OpenCV, notamment la transformation de Hough pour la détection de cercles et la cascade de Haar pour la reconnaissance faciale.

L'approche méthodique suivie dans ce rapport a permis de comprendre comment intégrer les différentes bibliothèques et dépendances nécessaires pour ces tâches, tout en fournissant des exemples de code pratiques. Ces exemples ont été essentiels pour illustrer la manière dont ces

algorithmes peuvent être appliqués à des données en temps réel, comme celles provenant d'une caméra.

Toutefois, pour compléter cette étude, il aurait été bénéfique d'inclure des visualisations des résultats obtenus, telles que des captures d'écran montrant les cercles détectés et les visages encadrés. De plus, une réflexion plus approfondie sur les limites et les améliorations possibles aurait permis d'approfondir l'analyse.

En conclusion, ce rapport offre une bonne introduction à l'utilisation d'OpenCV pour des tâches de vision par ordinateur. Les exemples présentés peuvent servir de base solide pour des applications plus complexes, tout en laissant place à des améliorations et à une exploration plus avancée des capacités d'OpenCV.