Projet de fin de 2ème année

REALISATION D'UN SYSTEME DE RECONNAISSANCE DU LANGAGE DES SIGNES

Réalisé par :
Yao Yann Vianney BOHOUSSOU
Idriss ABDOU MALAM

Sous la direction de :
M. OULAD HAJ THAMI

Année universitaire
2015-2016

Remerciements

Nos remerciements vont, en premier lieu, à l' endroit de notre encadrant de projet Pr. Oulad Haj Thami sans qui ce travail n'aurait pas vu le jour. Ses directives conjuguées à ses multiples conseils et encouragements nous ont permis de parvenir au terme de ce travail. Nous adressons également toute notre reconnaissance envers tout le corps professoral de l'ENSIAS. L'enseignement reçu de la part de celui- ci nous a permis de mener à bien la réalisation de ce projet.

Enfin, nous remercions tous ceux qui, de près ou de loin, nous ont aidé directement ou indirectement dans la réalisation ce projet.

Liste des abréviations

SIFT : Scale Invariant Feature Transform
RANSAC : Random Sample Consensus

Table des figures

Resumé

Dans le cadre du projet de fin d'année du semestre 4 de la formation d'ingénieur en informatique à l'ENSIAS, nous avons été emmenés à réaliser un système de reconnaissance du langage des signes. Cette reconnaissance s'opère en deux étapes principales qui sont : la détection de la main et la reconnaissance du signe. La détection permet de distinguer dans le flux vidéo la mains des autres objets et facilite ainsi la reconnaissance dans la mesure ou l'image utilisée ne contient que le signe réalisé. Elle utilise principalement une méthode de segmentation colorimétrique sur la couleur de la peau et le codage de freeman .La reconnaissance quand à elle permet de reconnaitre le signe réalisé par l'utilisateur du système et se fait en utilisant les descripteurs SIFT.

Mots clés :
détection, segmentation colorimétrique, couleur de la peau, reconnaissance, SIFT

Abstract

As part of the project year-end the half 4 of the computer engineer training ENSIAS, we were taken to achieve a sign language recognition system. This recognition takes place in two main stages : detection of the hand and the recognition of the sign. Detection can distinguish in the video stream the hands of other objects and facilitates the recognition to the extent that the image used only contains the sign made. It mainly uses a method of color segmentation on skin color and freeman coding.The recognition allows to recognize the signs carried out by the user and is achieved by the using of SIFT descriptors.

Key word :
detection, color segmentation, skin color, recognition, SIFT

Table des matières

Introduction

Aujourd'hui, la communication homme-machine s'effectue à travers trois principaux canaux à savoir les dispositifs de transmission tels que la souris et le clavier,la voix humaine et le geste.Le dernier canal présente beaucoup plus d'avantages par rapport aux premiers que ce soit en terme de naturalité, de concision,d'efficacité ou encore du caractère direct de l'interaction entre l'homme et la machine. En effet,les signes utilisés pour interagir avec une machine sont plus ou moins universels,ce qui permet l'utilisation de la machine à tout individu.En plus, la prise en compte de la dynamique gestuelle élargit la bande passante du canal en ce sens qu'elle permet de véhiculer plusieurs actions au moyen d'un seul geste.La communication gestuelle peut aussi donner lieu à la suppression des périphériques (souris,clavier..) dans les machines ; ces dernières n'analyseront alors que le mouvement de la main.

Dans ce projet,nous proposons un système de reconnaissance de gestes.Il capture une image et produit en sortie son interprétation. Cela passe par plusieurs phases. D'abord,dans la phase initiale,on acquiert des informations caractérisant la main de l'utilisateur. Ensuite ces informations sont utilisées pour segmenter la main du reste de l'image et la suivre dans une séquence vidéo.Enfin, l'image segmentée est comparée itérativement aux les images pré-enregistrées afin de produire le résultat. Ce système est entièrement créer en langage c++ avec la bibliothèque de traitement de données multimédia opencv.

Dans ce rapport,nous contextualisons dans un premier temps le projet en posant la problématique,en définissant les concepts qui tournent autour du sujet et en présentant les domaines d'applications la communication gestuelle,puis nous présenterons la bibliothèque de traitement de données multimédia opencv et après nous détaillons toutes les étapes de la mise en place du système réalisé,de l'acquisition de l'image à son interprétation en passant la détection de la main et la reconnaissance du signe capturé.

Chapitre 1

Contexte du projet

1.1 Problematique

La communication homme-machine exige une maitrise ou du moins une familiarisation avec les technologies. Cette dernière s'effectue exclusivement via le canal vocal ou des dispositifs réservés à cet effet (souris, clavier, etc ...).Ce qui a pour conséquence la restriction du nombre d'utilisateurs.

D'autre part, ces canaux de de transmission n'assurent pas une communication efficiente pour certains types d'applications comme ceux pour lesquels les informations spatio-temporelles doivent être exprimées. Aussi, ces canaux de transmission pénalisent les personnes handicapées (sourdes, muettes) au vu de leur incapacité de parler et d'entendre. Ils ne disposent que des classiques claviers et souris, sans aucune interaction gestuelle. Telles sont quelques-unes des limites de la communication homme-machine existante. L'intégration de système de communication gestuelle s'impose pour améliorer cette communication.

Commençons d'abord par comprendre de quoi il s'agit à travers ces pertinentes questions : Qu'est-ce qu'un geste ? Existe-t-il des catégories de gestes ? Quels sont les domaines d'applications de la communication gestuelle ? Avec quels dispositifs capture-t-on les gestes ? Quelles les étapes de mise en place d'un système de reconnaissance de gestes ?

1.2 La communication gestuelle

1.2.1 Définition de geste

Un geste est un signe manuel ou corporel qui permet d'illustrer les mots du langage, de les compléter ou de les appuyer. On classe généralement les gestes en fonction des parties du corps impliquées. On distingue trois types de gestes :

- les gestes de la main et du bras : Ils forment la principale catégorie de gestes. La main permet de réaliser des gestes précis et complexes. Les recherches autour de ces gestes concernent principalement la reconnaissance de positions de la main, l'interprétation du langage des signes et le développement d'interface homme-machine.
- les gestes de la tête et du visage : Peu et ont une signification spécifique. L'orientation de la tête est très utile pour la détection du champs de vision. Les recherches dans ce domaine s'intéressent à la reconnaissance faciale comme moyen d'authentification biométrique, comme soutien à d'autres systèmes de reconnaissance tels que la reconnaissance de la parole
- les gestes impliquant tout le corps : Les recherches dans ce domaine s'intéressent a tout le corps en interaction avec son environnement (analyse des gestes d'un danseur afin de générer de la musique idoine, analyse des gestes d'un athlète pour améliorer ces performances).

On différencie également les gestes dynamiques des gestes statiques. Un geste statique, également appelé posture, concerne la configuration du corps ou d'une partie du corps a un moment fixe dans le temps alors que le geste dynamique désigne une succession continue de postures. Par la suite, nous nous focaliserons que sur les gestes impliquant la main.

1.2.2 La fonction sémiotique de la main

[B4] La main a plusieurs fonctions dont la plus importante est la fonction sémiotique. Dans ce cas, la main joue le rôle d'organe émetteur d'information à destination de son environnement. Elle s'adresse à la perception des interlocuteurs de la personne qui effectue le geste.

La main permet de communiquer une grande diversité d'informations. Grace à celle-ci, on peut transmettre de messages au moyen de gestes simples tels que les gestes de salutation. Elle sert également de moyen d'expression de gestes des langues de signes, qui forment une véritable langue dotée d'une syntaxe et d'une sémantique assez riches. En outre, la main permet de véhiculer les gestes co-verbaux, qui s'effectuent simultanément avec la parole permettant d'illustrer ou de compléter le

message vocal.

1.2.3 Domaines d'applications en communication homme-machine

Les domaines d'applications sont très tributaires de l'évolution des technologies de capture ou de production de gestes. On distingue entre autres :

— La robotique : on crée des organes moteurs permettant de manipuler des objets par simulation des fonctions de la main.
— La réalité virtuelle : l'objectif est d'utiliser le geste de la main pour déplacer ou modifier des objets virtuels. La réalité virtuelle permet alors de doter un système d'une très bonne ergonomie et convivialité. D'où la nécessité de technologies de capture et de production de gestes sophistiquées.
— Domaine de l'animation de personnages virtuels : le but est de produire des représentations graphiques sur écran de personnages réalisant de gestes. L'application la plus illustratif est le système de création automatique de conversations animées entre des personnages virtuels. Les conversations sont créées par un planificateur de dialogue qui produit le texte et les intonations des phrases. Les expressions faciales, le mouvement des lèvres, le mouvement de la tête et des bras sont engendrés et synchronisés à partir du texte, de l'intonation et des relations entre le locuteur et son interlocuteur. Les gestes des bras et des mains apportent des informations supplémentaires.

En somme on peut séparer les applications de la communication gestuelle en trois catégories :

— les interfaces gestuelles utilisées pour décrire des formes ou des commandes
— les applications multimodales des fonctions de la main
— les applications de reconnaissance de geste des langues de signes.

Notre travail porte sur la dernière catégorie. Elle nécessite de dispositifs de capture de gestes exécutés par l'utilisateur.

1.2.4 Les systèmes de capture de gestes

Pour la capture de gestes de la main, on distingue deux types de dispositifs :

— Les dispositifs à caméra vidéo .Ils ne permettent que la capture de gestes en deux dimensions. Si l'on souhaite donc mesurer les gestes dans l'espace, il nécessaire d'utiliser une deuxième caméra pour capturer les informations issues de la troisième dimension. Cela nécessite également le développement d'algorithmes pour la reconstituer de du geste à partir des images. Un autre problème peut se poser lors de la capture des mouvements des doigts qui peuvent se cacher les uns les autres.
— Les gants numériques : Un gant numérique est associé à un système de capture de l'emplacement de de l'orientation. Il permet de mesurer le mouvement de la main dans l'espace. Son seul inconvénient est qu'il contraint les gestes de l'utilisateur par sa liaison filaire avec l'ordinateur. Dans ce travail, nous utilisons la caméra vidéo. De ce fait, nous ne capturons que les gestes en deux dimensions.

1.3 Structure générale d'un système de reconnaissance de gestes

[B5] La structure générale d'un système d'analyse de mouvements se décompose en quatre parties indépendantes et dont certaines sont facultatives.

1.3.1 Initialisation

Elle désigne le preprocessing de données. Elle s'assure qu'un certain nombre d'hypothèses soient vérifiées au démarrage du système. Les principales hypothèses émises par un système d'analyse de mouvement sont :

— La camera ne bouge pas ou alors de manière constante,
— le sujet fait face à la camera,
— les mouvements sont lents et continus,
— la lumière est constante,
— l'arrière -plan est statique et uniforme,
— les vêtements du sujet ont une couleur spécifique

1.3.2 Tracking

Le tracking est une tâche indépendante et constitue une préparation à la reconnaissance de gestes. Cette phase est caractérisée par la segmentation de l'image, la transformation de l'image pour convenir à un algorithme particulier et la méthode de suivi d'objet par exemple la main de frames en frames. La segmentation des objets d'intérêts (tête, main...) du reste de l'image peut se faire avec plusieurs solutions. La première solution est d'analyser les données temporelles (frames successives),de les comparer pixel par pixel et de détecter les différences. Bref, chercher les objets qui ont bougé. On considère généralement qu'il y'a un seul objet mouvant. La deuxième solution est seuillage qui s'effectue sous hypothèse d'arrière-plan monochrome et de vêtements unicolores du sujet. Il existe une approche statistique qui utilise les caractéristiques (couleur, contours,...) des pixels pour extraire les formes de l'arrière- plan. La transformation de l'image consiste à représenter des données utiles telles que des zones d'intérêts, des contours, etc en vue de faciliter les prochaines étapes. Quant à la méthode de suivi d'objet, elle consiste à trouver des objets similaires dans les frames consécutives. On procède généralement par prédiction. Cette dernière consiste à délimiter les zones d'intérêts en fonction des objets précédemment détectes et des informations sur ces objets.

1.3.3 Estimation de pose

C'est l'étape d'identification de la configuration des objets détectés. Au cours de cette phase, on extrait des informations sur les objets détectés telles que le centre de masse, la position ou encore l'orientation.

1.3.4 Reconnaissance

La dernière étape d'un système de reconnaissance de geste est la reconnaissance proprement dite. Le but de cette phase est d'identifier les gestes capturés. On distingue deux types de reconnaissance : la reconnaissance statique ou dynamique respectivement basée sur une ou plusieurs frames. La reconnaissance statique compare les données d'une frame avec des images pré-enregistrées. Ces images peuvent être des templates, des silhouettes ou des postures. S'agissant de la reconnaissance dynamique, elle se base sur un ensemble de frames consécutives pour déterminer l'action correspondante. Elle permet ainsi, de faire la différence entre marcher et courir, s'asseoir et se lever, etc.

Chapitre 2

Opencv

2.1 Vision par ordinateur

L'espace qui nous entoure a une structure tri-dimensionnelle (3D). Lorsque l'on demande à une personne de décrire ce qu'elle voit , elle n'éprouve aucune dificulté à nommer les objets qui l'entourent : téléphone, table, livre : Et pourtant l'information qui est réellement disponible sur la rétine des ses yeux n'est, ni plus ni moins, une collection de points. En chaque point ou pixel (picture element) il y a tout simplement une information qui donne une indication quant à la quantité de lumière et la couleur qui proviennent de l'espace environnant et qui ont été projetées à cet endroit de la rétine. Le téléphone, la table ou le livre n'existent pas sur la rétine. Guidé a la fois par l'information codée dans l'image (ou la rétine) et par ses propres connaissances, le processus visuel construit des percepts. Le téléphone ou le livre sont le réponses finales, résultant d'un processus d'interprétation qui fait partie intégrante du système de vision. De plus, il n'y a pas de correspondance terme à terme entre l'information sensorielle (la lumière et la couleur) et la réponse finale (des objets 3D). Le système de vision doit fournir les connaissances nécessaires afin de permettre une interprétation non ambigue.

Avec la naissance de machines de calcul de plus en plus sophistiquées, un certain nombre de scientifiques se sont attaqués au problème de la vision d'unpoint de vue quantitatif : est-il possible de construire un modèle computationnel pour la perception visuelle ? C'est à la suite de ce questionneent qu'est née le concept de vision par ordinateur. On entend par vision par ordinateur l'ensemble des méthodes utilisées pour capturer une scène du monde réel, la représenter sous un format numérique et interpreter son contenu afin d'en extraire les données nécéssaire à notre activité.
Pour ce faire, un tel système de vision a besoin d'un sous système capable de fournir une numérisation du monde réel (typiquement une caméra numérique) et d'un autre sous système en esure de dérouler des algorithmes permettant d'interpreter les données capturées par le système de vision. La vision par

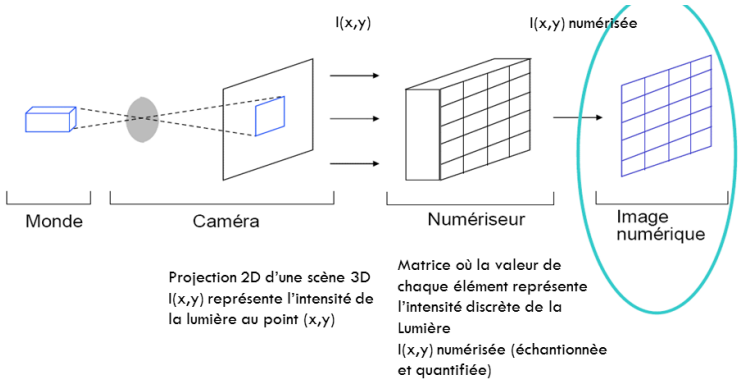

FIGURE 2.1 – Acquisition d'une image numérique

ordinateur est utilisée dans plusieurs domaines tels que la vidéosurveillance, le médical, la robotique ...

2.2 Historique

Opencv est une bibliothèque de vision par ordinateur disponible sur http ://source-Forge.net/projects/opencvlibrary. Elle est écrite en C et C++ optimisé et est disponible pour les machines sous Linux, windows, Mac OS X. Des API sont disponibles pour d'autres langages comme python, Matlab, java ...
Le projet OpenCV est une initiative Intel Research afin d'améliorer les performances des applications gourmandes en temps processeur avec un focus particulier sur les applications de vision en temps réel. Opencv est optimisé afin de pouvoir tirer partie des machines multicore et d'utiliser la librairie IPP (Integrated Performance Primitives) développée par Intel. Les principaux objectifs du projet opencv

sont :

- Faire progresser la recherche en vision par ordinateur en fournissant du code non seulement open source, mais aussi optimisé pour les infrastructures de vision de base.
- Diffuser les connaissances de vision en fournissant une infrastructure commune dont les développeurs pourraient tirer parti, de sorte que le code soit plus facilement lisible et réutilisable.
- Faire évoluer les applications commerciales basées sur la vision en produisant du code avec des performances optimisées, portable disponible gratuitement .

Pour atteindre ces objectifs, Opencv fournit plus de 500 fonctions de vision par ordinateur mais aussi de machine learning étant donné le lien étroit qui existe entre les deux domaines. La première version alpha de OpenCV a été rendue publique lors de la conférence IEEE sur la vision par ordinateur et reconnaissance de formes en 2000, et cinq bêtas ont été publiées entre 2001 et 2005. La première version 1.0 a été publiée en 2006. À la mi-2008, OpenCV obtenu le soutien des entreprises de Willow Garage, et est maintenant À nouveau en cours de développement. Une version 1.1 "pre-release" a été publiée en Octobre 2008.

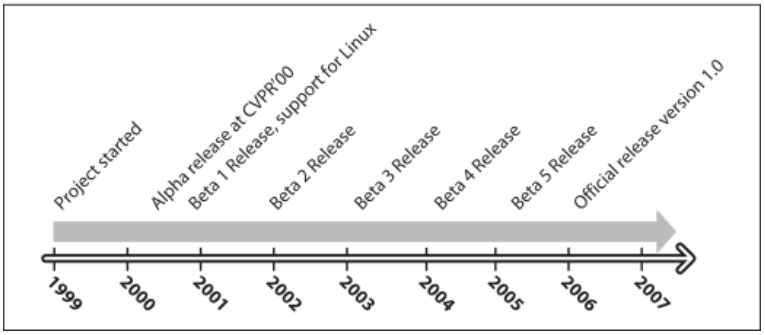

FIGURE 2.2 – Historique opencv

2.3 Les modules principaux

- Core : Ce module contient les fonctionnalités de base. Il permet de manipuler les structures de base, réaliser des opérations sur des matrices, dessiner sur des images, sauvegarder et charger des données dans des fichiers
- imgproc : Ce module contient les fonctionnalités de traitement d'images. Les fonctions et structures de ce module ont trait aux transformations d'images, au filtrage, à la détection de contours, de points d'intérêt...
- features2d : Ce module concerne principalement l'extraction de descripteurs.
- objdetect : la détection des objets et des instances des classes prédéfinies (par exemple, des visages, des yeux, des tasses, des personnes, voitures, etc.).
- video :traitement de flux vidéo. Ces fonctions servent à segmenter et suivre les objets en mouvement dans une vidéo.
- highgui : entrées-sorties et interface utilisateur. OpenCV intègre sa propre bibliothèque haut-niveau pour ouvrir, enregistrer et afficher des images et des flux vidéo. Celle- ci contient aussi un certain nombre de fonctions permettant de réaliser des interfaces graphiques très simples.
- calib3d : calibration, estimation de pose et stéréovision. Ce module contient des fonctions permettant de reconstruire une scène en 3D à partir d'images acquises avec plusieurs caméras simultanément.

2.4 Installation

2.4.1 Dépendances

— GCC
— CMake
— Git
— GTK+2.x , incluant headers (libgtk2.0-dev)
— pkg-config
— Python et Numpy avec les paquets (python-dev, python-numpy)
— ffmpeg or libav development packages : libavcodec-dev, libavformat-dev, libswscale-dev
— (optionel) libtbb2 libtbb-dev
— (optionel) libdc1394 2.x
— (optionel) libjpeg-dev, libpng-dev, libtiff-dev, libjasper-dev, libdc1394-22-dev

Ces dépendances peuvent être installées en utilisant ces commandes dans un terminal

(compiler) sudo apt-get install build-essential
(required) sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev libswscale-dev
(optional) sudo apt-get install python-dev python-numpy libtbb2 libtbb-dev libjpeg-dev libpng-dev libtiff-dev libjasper-dev libdc1394-22-dev

2.4.2 Téléchargement

La bibliothèque est disponible sur le lien **https ://sourceforge.net/projects/opencvlibrary/**. Le téléchargement peut aussi être éffectué en tapant ces commandes dans un terminal

git clone https ://github.com/Itseez/opencv.git

2.4.3 Compilation et installation de opencv

— créer un dossier opencv_build

mkdir opencv_build

— se déplacer dans ce dossier (cd opencv_build) et lancer la commande suivante :

cmake [<some optional parameters>] <chemin du dossier contenant les fichiers cource de opencv>

— lancer enfin les commandes suivantes :

make
sudo make install

Chapitre 3

Détection de la main

3.1 Principe général

Le système à développer étant censé interpreter le langage des signes, une étape importante est de détecter la main, les signes étant éssentiellement réalisés avec la main. Il existe plusieurs approches possibles pour réaliser cette détection de la main a savoir celles basées sur la couleur de la peau, sur la forme de la main, sur le mouvement ou encore sur la distribution spatiale des parties du corps. Nous avons opté pour une combinaison entre la segmentation colorimétrique et la description de la forme de la main. Notre détection s'opère ainsi selon les étapes suivantes :

- calibrage du système sur la couleur de la main de l'utilisateur par récupération d'échantillons de pixels de la main au démarage du système à partir desquels des intervalles de couleurs de la main sont récupérés.
- segmentation de la frame courante pour récupérer uniquement les pixels dont la couleur est comprise dans les intervalles de couleurs obtenus après la phase de calibrage.
- détermination des contours par l'application d'un filtre canny à l'image segmentée.
- reconnaissance du contour de la main en utilisant le codage de freeman.

3.2 Espace de couleur

Il existe plusieurs espaces de couleurs RGB, YCbCr, HSV, YIQ, YUV ... L'espace de couleur RGB correspondant aux couleurs primaires (Red, Green, Blue) est le plus utilisé et même celui fournit par défaut pour la majorité des formats d'images et de vidéo. Cependant, celui ci présente une certaine redondance dans les différentes composantes. Afin de réduire cette redondances, nous avons opté pour les espaces de couleurs HSV et YCbCr qui sont obtenus par une transformation linéaire de l'espace RGB et qui possèdent une redondance moindre dans leur différentes composantes.

3.3 Calibrage du système

Cette étape s'éffectue au lancement du système et a pour objectif de déterminer les couleurs qui seront utilisées pour la segmentation colorimétrique des frames capturées.Pour ce faire, des zones sont définies dans la frame de calibrage ou doit être positionné l'objet qui devra être recherché par la suite.

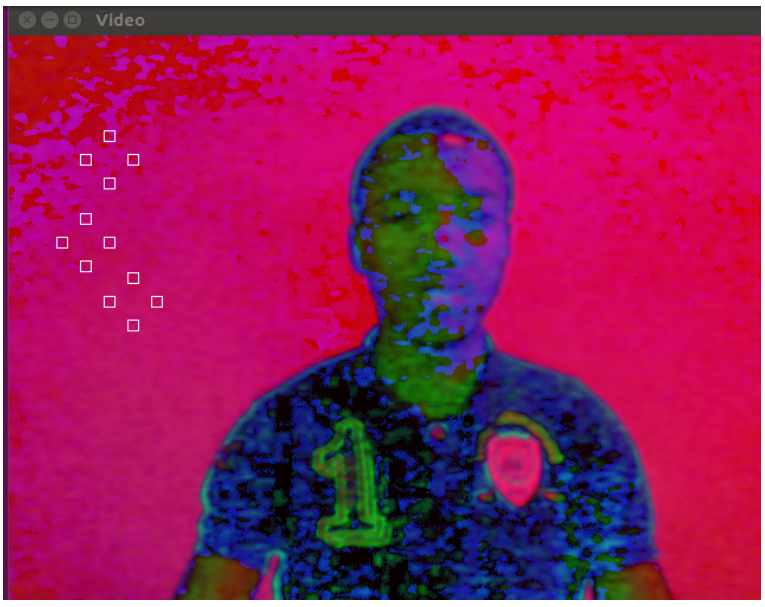

FIGURE 3.1 – calibrage HSV

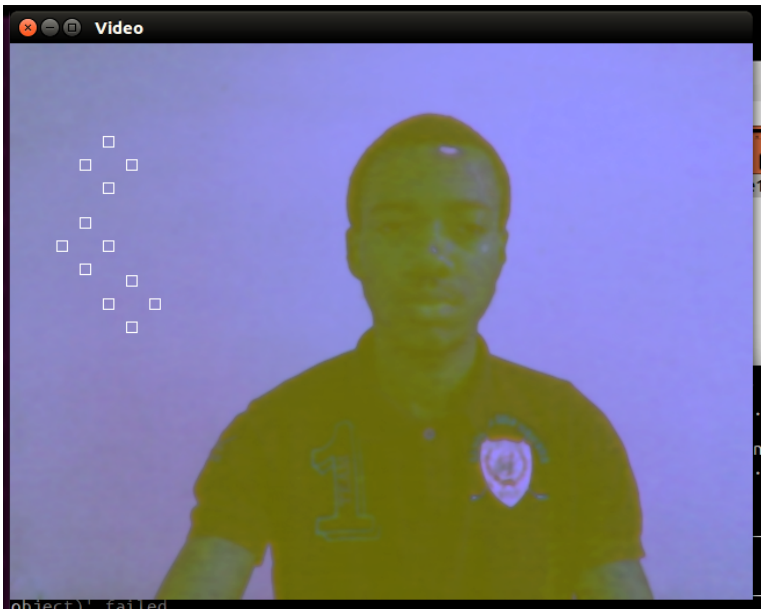

FIGURE 3.2 – calibrage YCbCr

Ensuite pour chaque zone, on détermine la valeur maximale et la valeur minimale de chaque composante de l'espace de couleur.

```
vector<Scalar> getColor(Mat image){
        Scalar scM,scm;
        vector<Scalar> listScalar;
        int B = 0,G = 0,R = 0;
        int MB=0,mB=0,MG=0,mG=0,MR=0,mR=0;
        for(int i= 2; i<9;i++)
        {
                for(int j=2; j<9; j++)
                {
                        Vec3b pixel = image.at<Vec3b>(i,j);
                        B = pixel.val[0];
                        if(B>MB)MB=B;
                        G = pixel.val[1];
                        if(G>MG)MG=G;
                        R = pixel.val[2];
                        if(R>MR)MR=R;
                }
        }
        mB=MB;mG=MG;mR=MR;
        for(int i= 2; i<9;i++)
        {
                for(int j=2; j<9; j++)
                {
                        Vec3b pixel = image.at<Vec3b>(i,j);
                        B = pixel.val[0];
                        if(B<mB)mB=B;
                        G = pixel.val[1];
                        if(G<mG)mG=G;
                        R = pixel.val[2];
                        if(R<mG)mR=R;
                }
        }
        scM = Scalar(MB,MG,MR);
        scm = Scalar(mB,mG,mR);
        listScalar.push_back (scM);
        listScalar.push_back (scm);
        return listScalar;}
```

FIGURE 3.3 – récupération intervalle de couleur en c++

La méthode présentée est appliquée pour chaque zone de calibrage. On obtient ainsi en sortie un ensemble d'intervalles de couleurs possibles dans lesquels la couleur de l'objet qui sera recherché peut être comprise.

3.4 Segmentation colorimétrique

Dans cette étape, nous cherchons à détecter dans une image uniquement les objets dont la couleur appartient à l'un des intervalles obtenus lors de la phase de calibrage.

Entrées : une image ($image$) et une liste d'intervalles de couleurs $list$
Output : une image ($resultat$) segmentée
$t \leftarrow taille(list)$
$reslt$ ensemble fini $reslt = \{im_1, im_2, \ldots, im_t\}$ d'images
$i \leftarrow 0$
pour $it \leftarrow debutlist$ ***to*** $finlist$ **faire**
  $color \leftarrow *it$
  $reslt[i] \leftarrow segmentationCouleur(image, color)$
  $i \leftarrow i + 1$
fin
$i \leftarrow 0$
$resultat \leftarrow rest[i]$ **pour** $i \leftarrow 1$ ***to*** t **faire**
  $resultat \leftarrow resultat + reslt[i]$
fin
retourner $resultat$

Algorithme 1 : PROCESSCOLOR segmentation colorimétrique de l'image

```
Mat processColor(Mat image, vector<vector<Scalar> > interColor)
{
        Mat resultat;
        int nbrIntervalles = (int)interColor.size();
        Mat *reslt = new Mat [ nbrIntervalles ];
        vector<Scalar> color;
        int i = 0;//itérateur pour le tableau de Mat --reslt--
        for (vector<vector<Scalar> >::iterator it = interColor.begin() ; it != interColor.end(); ++it)
        {
                color = *it;
                inRange(image,color[1],color[0],reslt[i]);
                i++;
        }

        i=0;
        resultat = reslt[0];
        for(i=1;i<nbrIntervalles;i++)
        {
                resultat = resultat + reslt[i];
        }

        return resultat;
}
```

FIGURE 3.4 – segmentation colorimétique en c++

3.5 Détermination des contours

En traitement des images, plusieurs filtres permettent de mettre en évidence les contours. Nous avons par exemple le filtre canny, le filtre gradient, le filtre laplacien. Ces filtres atténuent les basse fréquences et permettent ainsi l'accentuation des détails et du contraste. Leur application à une image consiste en une convolution de l'image par le noyau du filtre qui peut être de taille (3,3), (7,7) etc.

- Faire une rotation de π du noyau par rapport à son centre
- Centrer le filtre sur le point P(x, y) en le superposant à l'image
- Effectuer la somme pondérée entre les pixels de l'image et les coefficients du filtre
- Le pixel P dans l'image filtrée aura comme valeur cette somme pondérée

$$g(m,n) = \sum\sum h(i,j) f(m-i, n-j)$$

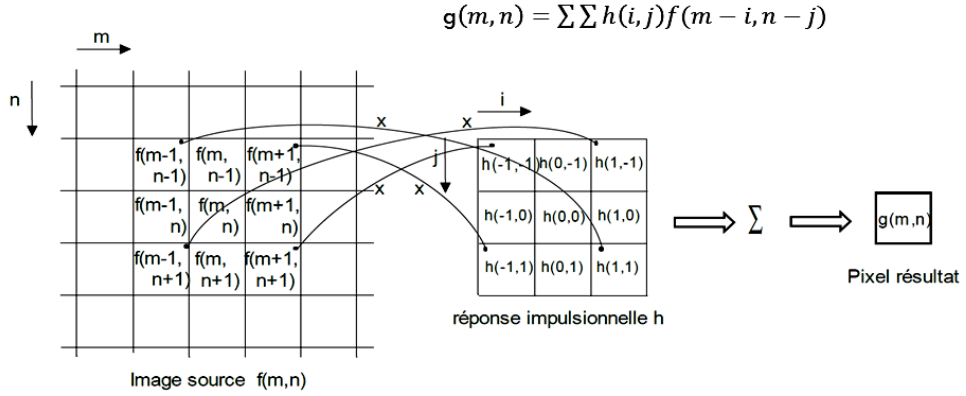

FIGURE 3.5 – Principe de la convolution

Nous avons opté pour un filtre canny qui est implémenté en opencv par la fonction

void Canny(InputArray image, OutputArray edges, double threshold1, double threshold2, int apertureSize=3, bool L2gradient=false).

```
Mat getContours_dessinerContours (Mat frame)
{
        Mat cont1,cont2;
        vector<vector<Point> > contours;
        vector<Vec4i> hierarchy;
        Scalar color = Scalar( 255, 255, 255 );

        Canny( frame, cont1, 100, 255, 3 );
        dilate( cont1, cont1, Mat(), Point(-1,-1),1, BORDER_CONSTANT, morphologyDefaultBorderValue() );
        findContours( cont1, contours, hierarchy, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE, Point(0, 0) );
        cont2 = Mat::zeros( frame.size(), CV_8UC3 );
        for( size_t i = 0; i< contours.size(); i++ )
        {
                drawContours( cont2, contours, (int)i, color, CV_FILLED, 8, hierarchy, 0, Point() );
        }
        return cont2;
}
```

FIGURE 3.6 – Détermination des contours

3.6 Reconnaissance du contour de la main

Une fois les contours des objets dont la couleur correspond à la couleur pour laquelle le système a été calibré au départ, il faut reconnaitre parmi tout les contours déterminés celui de la main. Afin de réaliser cette reconnaissance, nous avons opté pour un codage de freeman des contours.Le codage de Freeman, permet de traduire les orientations prises par la suite des vecteurs, chacun obtenu entre deux points successifs du contour.

Entrées : un contour (*contour*)
Output : codage de freeman du contour ($freemanCode$)
$i \leftarrow 0$
pour $pt \leftarrow debutcontour$ ***to*** $fincontour$ **faire**
  si $i = 0$ **alors**
    $pt_prec = pt$
    $i \leftarrow i + 1$;
  fin
  sinon
    si $pt.x = pt\_prec.x + 1 et pt.y = pt\_prec.y$ **alors**
      freemanCode.ajouter(0)
    fin
    sinon si $pt.x = pt\_prec.x + 1 et pt.y = pt\_prec.y + 1$ **alors**
      freemanCode.ajouter(1)
    fin
    sinon si $pt.x = pt\_prec.x et pt.y = pt\_prec.y + 1$ **alors**
      freemanCode.ajouter(2)
    fin
    sinon si $pt.x = pt\_prec.x - 1 et pt.y = pt\_prec.y + 1$ **alors**
      freemanCode.ajouter(3)
    fin
    sinon si $pt.x = pt\_prec.x - 1 et pt.y = pt\_prec.y$ **alors**
      freemanCode.ajouter(4)
    fin
    sinon si $pt.x = pt\_prec.x - 1 et pt.y = pt\_prec.y - 1$ **alors**
      freemanCode.ajouter(5)
    fin
    sinon si $pt.x = pt\_prec.x et pt.y = pt\_prec.y - 1$ **alors**
      freemanCode.ajouter(6)
    fin
    sinon si $pt.x = pt\_prec.x + 1 et pt.y = pt\_prec.y - 1$ **alors**
      freemanCode.ajouter(7)
    fin
    $pt\_prec = pt$
  fin
fin
retourner $freemanCode$

Algorithme 2 : getFreemanCodage Codage de freeman d'un contour

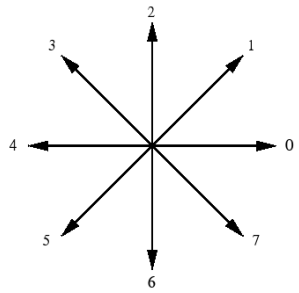

FIGURE 3.7 – Vecteurs de parcours du codage de freeman

```
vector<int> getFreemanCodage (vector<Point> contour)
{
        vector<int> freemanCode;
        Point prec,pt;
        int pt_x, pt_y, prec_x, prec_y;
        int i = 0;
        for (vector<Point>::iterator it = contour.begin() ; it != contour.end(); ++it)
        {
                if(i == 0){prec = *it;i+=1;}
                else
                {
                        pt = *it;
                        pt_x = pt.x;
                        pt_y = pt.y;
                        prec_x = prec.x;
                        prec_y = prec.y;
                        if(pt_x == prec_x+1 && pt_y == prec_y)
                                freemanCode.push_back(0);
                        else if(pt_x == prec_x+1 && pt_y == prec_y+1)
                                freemanCode.push_back(1);
                        else if(pt_x == prec_x && pt_y == prec_y+1)
                                freemanCode.push_back(2);
                        else if(pt_x == prec_x-1 && pt_y == prec_y+1)
                                freemanCode.push_back(3);
                        else if(pt_x == prec_x-1 && pt_y == prec_y)
                                freemanCode.push_back(4);
                        else if(pt_x == prec_x-1 && pt_y == prec_y-1)
                                freemanCode.push_back(5);
                        else if(pt_x == prec_x && pt_y == prec_y-1)
                                freemanCode.push_back(6);
                        else if(pt_x == prec_x+1 && pt_y == prec_y-1)
                                freemanCode.push_back(7);

                        prec = pt;
                }
        }
        return freemanCode;
}
```

FIGURE 3.8 – getFreemanCodage en c++

On compare ensuite les codes de freeman pour déterminer celui posédant le plus de forts points de courbures. Un point de courbure est défini dans le code de freeman par deux entiers successifs dont la différence est différente de 0. Si cettedifférence est égale à 1, il s'agit d'un point de courbure simple si il es supérieur ou égal à 2, il s'agit d'un fort point de courbure.

Entrées : code de freeman sous la forme d'une liste d'entiers ($freeman$)
Output : nombre de forts points de courbure ($courbure$)
$courbure \leftarrow 0$
$prec \leftarrow -1$
pour $it \leftarrow debutfreeman$ ***to*** $finfreeman$ **faire**
  si $prec = -1$ **alors**
    prec = *it ;
  fin
  sinon
    si $abs(*it - prec) >= 2$ **alors**
      $courbure = courbure + 1$
    fin
    $prec = it$
  fin
fin
retourner $courbure$

Algorithme 3 : GETFREEMANCOURBURE nombre de forts points de courbure du code de freeman

```
int calcFreemanZigZag(vector<int> freeman)
{
        int zigzag=0;
        int prec = -1;
        for (vector<int>::iterator it_int = freeman.begin() ; it_int != freeman.end(); ++it_int)
        {
                if(prec == -1)
                {
                        prec = *it_int;
                }
                else
                {
                        if(abs(*it_int - prec) >= 2)
                                zigzag+=1;
                        prec = *it_int;
                }
        }

        return zigzag;
}
```

FIGURE 3.9 – calcul du nombre de cpoint de courbure en c++

Ensuite, le contour possédant le plus de points de courbures est le contour le la main.

FIGURE 3.10 – Resultat de la détection de la main

Chapitre 4

Reconnaissance du signe

La reconnaissance que nous effectuons est de type statique en ce sens que nous comparons chaque frame capturée et segmentée avec des postures stockées dans une base de données d'images. La comparaison s'effectue par hybridation de deux méthodes pour améliorer les résultats.

4.1 Méthode de comptage du nombre de doigts

Cette méthode vise à dénombrer les doigts de l'image courante afin d'éliminer un certain nombre d'images. Cette phase permet également d'améliorer la reconnaissance en terme de temps de réponse ; plus on élimine des images, plus la complexité de l'algorithme de reconnaissance diminue. Pour mettre en œuvre cette méthode, il faut d'abord calculer l'enveloppe convexe de l'image segmentée, ensuite déterminer les sommets et les interstices et enfin déduire le nombre de doigts.

4.1.1 L'enveloppe convexe

L'enveloppe convexe d'un objet est l'ensemble convexe le plus petit parmi ceux qui le contiennent. Dans un plan, elle est comparable à un élastique englobant tous les points d'un objet qu'on relâche jusqu'à ce qu'il se contracte au maximum.

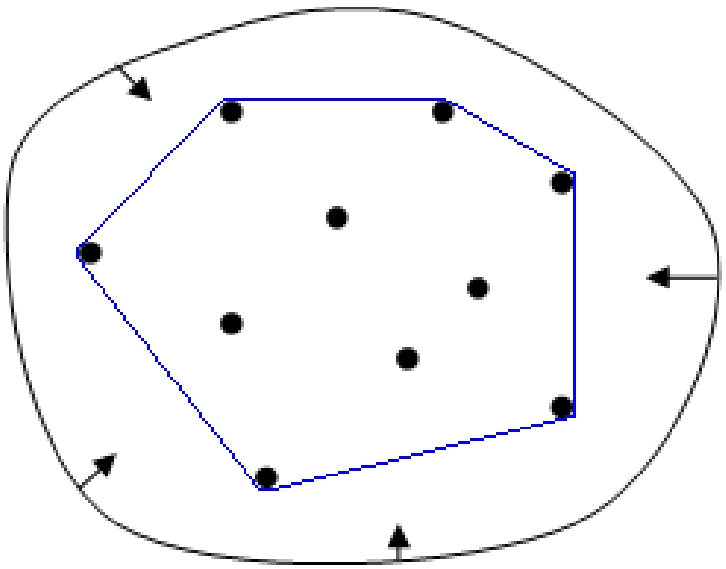

FIGURE 4.1 – Enveloppe convexe

En algorithmique, le calcul de l'enveloppe convexe est un problème classique abordé et résolu par plusieurs chercheurs. Parmi les solutions, on peut citer :

— marche Jarvis [w6]
— algorithme de Chann [w7]
— parcours de Graham [w8]
— diagramme de Voronoi [w9]

Opencv dispose d'une implémentation de cet algorithme. C'est celle-ci que nous avons utilisé .Le résultat est le suivant :

FIGURE 4.2 – enveloppe convexe de la main

4.1.2 Les sommets et les interstices des doigts

Les sommets et les interstices sont déterminés à partir de l'enveloppe convexe et du contour de la main. Ils représentent les extrema du contour de l'image. Les sommets sont des points qui appartiennent au contour et à l'enveloppe connexe. Ils correspondent aux points au niveau desquels la dérivée seconde du contour est nulle et la courbe change de sens de variation en décroissant. Quant aux interstices, ils appartiennent seulement au contour et correspondent aux points au niveau desquels la dérivée seconde s'annule et la courbe du contour change de sens de variation en croissant. Il existe une fonction dans opencv qui détermine ces points.

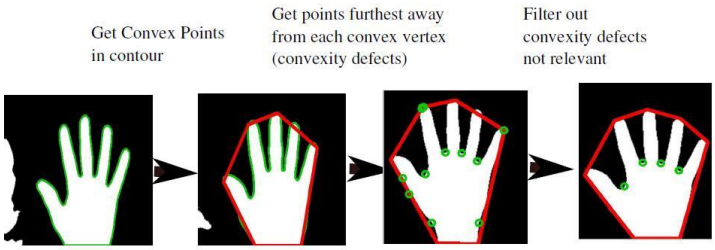

FIGURE 4.3 – sommets et interstices de la main

4.1.3 Déduction du nombre de doigts

A partir des sommets ou des interstices, on peut facilement déduire le nombre de doigts. Il suffit simplement de compter le nombre de sommets ou le nombre d'interstices augmente d'une unité. Par cette étape, on réduit alors le périmètre de recherche et on se focalisera désormais que sur les images comptant le même nombre de doigts.

4.2 La détection et la correspondance de points-clés par SIFT

[w10] La scale-invariant feature transform (SIFT) est un algorithme de détection et d'identification des éléments similaires entre différentes images numériques et ce quelle que soit l'échelle de représentation des images. L'étape primordiale est le calcul des descripteurs SIFT des images à étudier. Ce sont de données numériques issues de l'analyse de l'image et qui résument son contenu indépendamment de l'échelle, du cadrage et de la luminosité. Ces invariances confèrent à cette méthode une robustesse à laquelle elle doit sa popularité. Les étapes de calcul des descripteurs SIFT se déclinent comme suit :

— Détection de points-clés :
Un point-clé est défini par ses coordonnées (x, y) et par son facteur d'échelle caractéristique (σ).On appelle gaussienne de facteur $\sigma, L(x, y, \sigma) = I(x, y) * G(\sigma)$, l'image convoluée par un filtre gaussien de paramètre σ. On obtient dans un premier temps des points-clés candidats qui sont les extrema locaux de la fonction Dk telle que : $D_k(x, y) = L_k(x, y) - L_k - 1(x, y)$ avec L_k gaussienne de facteur $k * \sigma$. Un traitement supplémentaire est effectué pour éliminer les points-clés candidats instables, de faible contraste ou situés sur des arêtes de contour à faible courbure.

FIGURE 4.4 – points d'intérêts d'une image

— Assignation d'orientation : Cette étape consiste à attribuer à chaque point-clé une ou plusieurs orientations calculées à partir des gradients un voisinage du point. Elle est donnée par la formule suivante :

$$H[\theta(x,y)] = arctan(I_y(x,y)/I_x(x,y))$$

avec $I_x et I_y$ les matrices d'orientations suivant respectivement x et y.

— Descripteurs SIFT : A cette étape, on détermine les descripteurs associés aux points d'intérêts qui sont des histogrammes d'orientations de gradients autour des ceux-ci. Pour ce faire, on divise l'espace autour des points d'intérêts en 4*4 blocs de 4*4 pixels chacun. Pour chaque carre 4*4, on détermine un histogramme d'orientations divisé en 8 intervalles par la formule suivante :

$H[\theta(x,y)] = H[\theta(x,y)] + \| I_g(x,y) \| * G(x,y,\sigma)$ où $\theta(x,y)$ est l'orientation du gradient discrétisée au pixel (x,y), $\| I_g(x,y) \|$ est le module du gradient au pixel (x,y) et $G(x,y,\sigma)$ la pondération obtenue au pixel (x,y) en appliquant un masque gaussien centré sur le point d'intérêt.

Le vecteur SIFT, de taille 4*4*8 est obtenu en concaténant chacun des histogrammes calcules.

— Mise en correspondance des points : A ce niveau, on met en correspondance les points d'intérêts de l'image requête avec ceux des images déterminés avec la première méthode. D'abord, on détermine les deux plus petites distances deux points d'intérêts de sorte à toujours minimiser la somme des distances entre les couples des points. Si le rapport de ces distances est inférieur à 0.8, alors on conserve cette correspondance. Sinon, cette correspondance est tout simplement rejetée. Ensuite, on procède à l'élimination des points aberrants. Ce sont des points isolés dont on peut s'en passer lors de la reconstitution de l'image. La méthode, la plus utilisée est la méthode RANSAC [w11].C'est une méthode itérative, non déterministe dans le sens où elle produit un résultat correct avec une certaine probabilité, celle-ci augmentant à mesure que le nombre d'itérations augmente. Son principe est le suivant :
  — sélectionner d'un nombre aléatoire de données (points)
  — Dégager un modèle provisoire en déterminant ses paramètres
  — Tester les autres données au modèle précédemment estimé. Tout point qui correspond au modèle estimé est considéré comme une donnée pertinente candidate.
  — Le modèle estimé est considéré comme correct si suffisamment de points ont été classés comme données pertinentes candidates.
  — Les points non pertinents sont simplement rejetés.

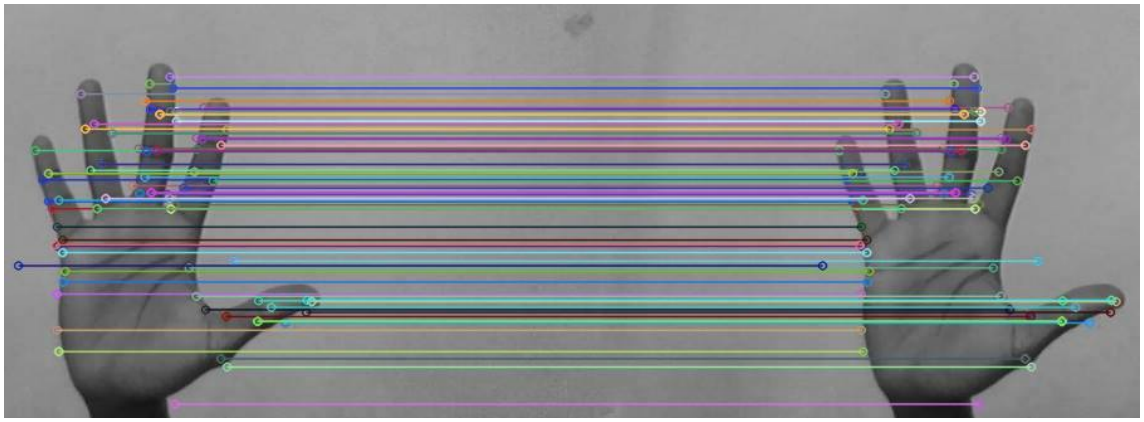

FIGURE 4.5 – mise en correspondance des points d'interets

— Enfin, on détermine le ratio qui rapport les points issus de l'algorithme RANSAC et ceux considérés a l'étape précédente. L'image dont le ratio est le plus grand est considérée comme étant la plus similaire à l'image requête. Elle est considérée comme identique à l'image si et seulement si le ratio est supérieur ou égal 0.8

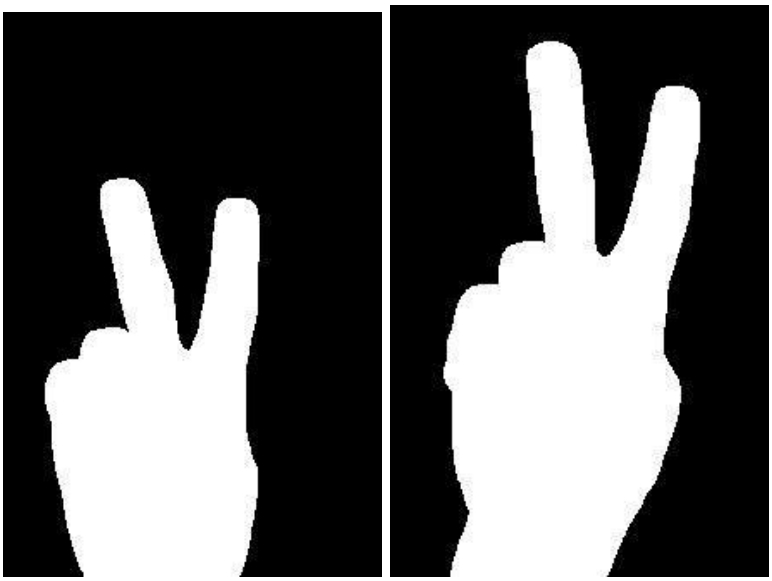

FIGURE 4.6 – resultat de la reconnaissance

Conclusion

Ce projet avait pour principal objectif la réalisation d'un système de reconnaissance du langage des signes.Cette reconnaissance se fait en deux étapes principales : la détection de la main réalisée à partir d'une détectide la couleur de la peau et d'un codage de freeman, la reconnaissance du signe réalisée en utilisant la méthode SIFT. Bien que notre système fournisse des resultats acceptables, il peut encore être amélioré d'un point de vue vitesse et efficacité de la reconnaissance. Ces améliorations pouront constituer des évolutions futures du système Ce projet a été une belle expérience pour nous. Il nous en effet a permis de nous initier au domaine de la vision par ordinateur, d'acquérir de nouvelles connaissances concernant le traitement des images et de perfectionner notre maitrise l'utilisation de La bibliothèque Opencv, compétences qui nous seront sans aucun doute utilis pour nos futures carrières d'ingénieur informaticiens.

Bibliographie

[B1] Radu HORAUD et Olivier MONGA. *CVision par ordinateur : outils fondamentaux Deuxième edition Editions Hermès.*

[B2] Samaa M. Shohieb, Hamdy K. Elminir, A.M. Riad. *SignsWorld Atlas ; a benchmark Arabic Sign Language database.*

[B3] Djamila Dahmani, Slimane Larabi. *User-independent system for sign language finger spelling recognition.*

[B4] Annelies BRAFFORT. *Reconnaissance et compréhension de gestes, application à la langue des signes.*

[B5] Julien Thomet Département d'informatique Université de Fribourg. *Une vue d'ensemble de la reconnaissance de gestes.*

[W1] *http ://www.bogotobogo.com/cplusplus/files/OReilly%20Learning%20OpenCV.pdf.*

[W2] *https ://openclassrooms.com/courses/introduction-a-la-vision-par-ordinateur.*

[W3] *http ://docs.opencv.org/2.4/doc/tutorials/introduction/linux_install/linux_install.html.*

[W4] *http ://milq.github.io/install-opencv-ubuntu-debian/.*

[W5] *https ://en.wikipedia.org/wiki/OpenCV.*

[W6] *https ://fr.wikipedia.org/wiki/Marche_de_Jarvis.*

[W7] *https ://fr.wikipedia.org/wiki/Algorithme_de_Chan.*

[W8] *https ://fr.wikipedia.org/wiki/Parcours_de_Graham.*

[W9] *https ://fr.wikipedia.org/wiki/Diagramme_de_Vorono%C3%AF.*

[W10] *https ://fr.wikipedia.org/wiki/Scale-invariant_feature_transform.*

[W11] *https ://fr.wikipedia.org/wiki/RANSAC#L.27algorithme.*