Universitat de Vic
Escola Politècnica Superior

Treball Final de Màster

Reconeixement de gestos de la mà amb el sensor Kinect

Elisabet Faja Grau

Màster en Tecnologies Aplicades de la Informació
Directors: Ramon Reig i Bolaño, Pere Martí i Puig
Vic, Setembre de 2012

Universitat de Vic
Escola Politècnica Superior

Resum del Treball Final de Màster
Màster en Tecnologies Aplicades de la Informació

Títol: Reconeixement de gestos de la mà amb el sensor Kinect
Paraules clau: Kinect, reconeixement, imatges, signes, dactilologia, profunditat, segmentació, llindars, contorns, gestos, envolupant, mà, dits
Autor: Elisabet Faja Grau
Direcció: Ramon Reig i Bolaño i Pere Martí i Puig
Data: Setembre de 2012

Resum

El reconeixement dels gestos de la mà (HGR, Hand Gesture Recognition) és actualment un camp important de recerca degut a la varietat de situacions en les quals és necessari comunicar-se mitjançant signes, com pot ser la comunicació entre persones que utilitzen la llengua de signes i les que no.

En aquest projecte es presenta un mètode de reconeixement de gestos de la mà a temps real utilitzant el sensor Kinect per Microsoft Xbox, implementat en un entorn Linux (Ubuntu) amb llenguatge de programació Python i utilitzant la llibreria de visió artifical OpenCV per a processar les dades sobre un ordinador portàtil convencional.

Gràcies a la capacitat del sensor Kinect de capturar dades de profunditat d'una escena es poden determinar les posicions i trajectòries dels objectes en 3 dimensions, el que implica poder realitzar una anàlisi complerta a temps real d'una imatge o d'una seqüencia d'imatges.

El procediment de reconeixement que es planteja es basa en la segmentació de la imatge per poder treballar únicament amb la mà, en la detecció dels contorns, per després obtenir l'envolupant convexa i els defectes convexos, que finalment han de servir per determinar el nombre de dits i concloure en la interpretació del gest; el resultat final és la transcripció del seu significat en una finestra que serveix d'interfície amb l'interlocutor.

L'aplicació permet reconèixer els números del 0 al 5, ja que s'analitza únicament una mà, alguns gestos populars i algunes de les lletres de l'alfabet dactilològic de la llengua de signes catalana.

El projecte és doncs, la porta d'entrada al camp del reconeixement de gestos i la base d'un futur sistema de reconeixement de la llengua de signes capaç de transcriure tant els signes dinàmics com l'alfabet dactilològic.

Universitat de Vic
Escola Politècnica Superior

Abstract fo Final work of
Màster of Applied Information Tecnology

Title: Hand Gesture Recognition using Kinect Sensor
Keywords: Kinect, Recognition, images, signs, sign language, depth, segmentation, threshold, contours, gestures, convex hull, hand, fingers
Author: Elisabet Faja Grau
Directors: Ramon Reig i Bolaño and Pere Martí i Puig
Date: September de 2012

Abstract

Nowadays, Hand Gesture Recognition (HGR) is an important research field since there are many situations in which it is necessary to communicate through signs, for example communication between deaf and non-deaf people.

In this project, a Real-Time Hand Gesture Recognition method with Kinect sensor for Xbox is presented. This system is implemented with Python under Linux using OpenCV computer vision library for data processing.

Kinect sensor capacity to capture depth data enables 3D objects motion tracking, which allows the users to carry out complete real-time analysis of images or images sequences.

Recognition process is based on image segmentation to remove background, which provides hand-processing only, contours detecting, convex hull and convex defects in order to determine the number of fingers and recognize hand gestures. Once the gesture has been recognized, the system will provide the correspondent output number or letter.

The application can recognize numbers from 0 to 5, since only one hand is processed, many popular gestures and some letters of Catalan sign language alphabet.

This project is, therefore, an introduction of Hand Gesture Recognition and the cradle of a future sign language recognition system able to transcribe both dynamic signs and sign language alphabet.

Agraïments

Aquest projecte no hagués estat possible sense el suport de totes les persones que han estat al meu costat des del primer moment.

Agraeixo a en Ramon Reig la seva atenció, paciència i el suport que mà demostrat.

A l'Enric, per donar-me la mà i caminar amb mi en tot aquest procés.

Per altra banda, agrair el suport moral que m'han donat els meus amics i companys, sobretot a la M.Angels, la Merci, l'Eli, la Sandra i en Jacint.

Finalment, donar gràcies a la meva família, als meus pares Albert i Montse i a la meva germana Mari per estar sempre al meu costat.

Dedico aquest treball a la meva neboda Carla, per fer-me somriure cada dia.

Índex

Índex de figures

Índex de taules

Capítol 1: Introducció

"La ciència més útil és aquella que el seu fruit és el més comunicable"
– Leonardo da Vinci

1.1.- Introducció

El reconeixement de gestos de les mans és una de les àrees que més interès estan generant ja que permeten desenvolupar aplicacions pel control de dispositius o per la transmissió d'informació a temps real.

Existeixen moltes situacions on és preferible o necessari comunicar-se mitjançant la llengua de signes. La majoria d'essers humans estan dotats de la capacitat de veu que els permet comunicar-se entre si, però per desgracia, no tothom té aquesta capacitat degut a la manca del sentit de l'oïda i és gràcies a la llengua de signes que poden comunicar-se.

La majoria de persones no estan familiaritzades amb la llengua de signes i això els hi fa difícil comunicar-se sense un intèrpret. Aquesta necessitat obre les portes al camp del reconeixement de signes i la seva transcripció per tal de facilitar aquesta comunicació a temps real.

Fins a dia d'avui han aparegut molts sistemes de reconeixement de gestos, la majoria basats en imatges estereoscòpiques, el que implica l'ús d'algoritmes complicats i càmeres molt cares. És per això que sensors com Kinect aporten una solució econòmica per treballar amb el reconeixement i interpretació de moviments i gestos d'usuaris a temps real.

Kinect és un sensor de moviment desenvolupat per PrimeSense i comercialitzat per Microsoft com a perifèric d'entrada per a la seva videoconsola Xbox 360 llançat al mercat durant el novembre de 2010. Aquest perifèric de l'estil d'una web-cam permet a l'usuari controlar i interactuar amb la Xbox 360 sense haver de tenir cap contacte físic amb un comandament de control tradicional mitjançant una interfície natural d'usuari que reconeix gestos, ordres de veu, imatges o objectes.

El dispositiu està format per una càmera RGB, un sensor de profunditat i un micròfon multi-array que funcionen mitjançant el seu software propietari i que proporcionen la captura de moviment 3D, el reconeixement facial i de veu.

Tot i estar dissenyada inicialment per a utilitzar-la amb la videoconsola Xbox, de seguida es van desenvolupar controladors (drivers) per a poder-la utilitzar amb ordinadors o altres dispositius.

Els primers controladors Kinect van ser de codi obert per GNU/Linux i permetien l'ús de la càmera RGB i les funcions de profunditat. Més tard, durant el segon trimestre de 2011 Microsoft va llançar el seu paquet per a desenvolupadors (SDK – Software Development Kit) en fase Beta (de proves) per utilitzar-la sota el sistema operatiu Windows 7. D'aquesta manera les capacitats que té la Kinect arriben als ordinadors i amb això s'obren les portes als desenvolupadors per a crear aplicacions.

Desenvolupadors d'arreu del món estan investigant possibles aplicacions de Kinect que van més enllà dels jocs, com per exemple la resposta de robots a gestos humans, el control de software mitjançant el moviment de les mans, sistemes de vídeo-vigilància amb seguiment i control de presència, aplicacions en medicina per detectar alteracions de comportament en nens i detectar malalties com l'autisme o el reconeixement i interpretació de les llengües de signes.

D'aquí neix l'objectiu principal d'aquest treball, que és crear una plataforma oberta al desenvolupament d'aplicacions pel reconeixement de gestos de la mà, basats en la llengua de signes, mitjançant les dades de profunditat capturades amb el sensor Kinect.

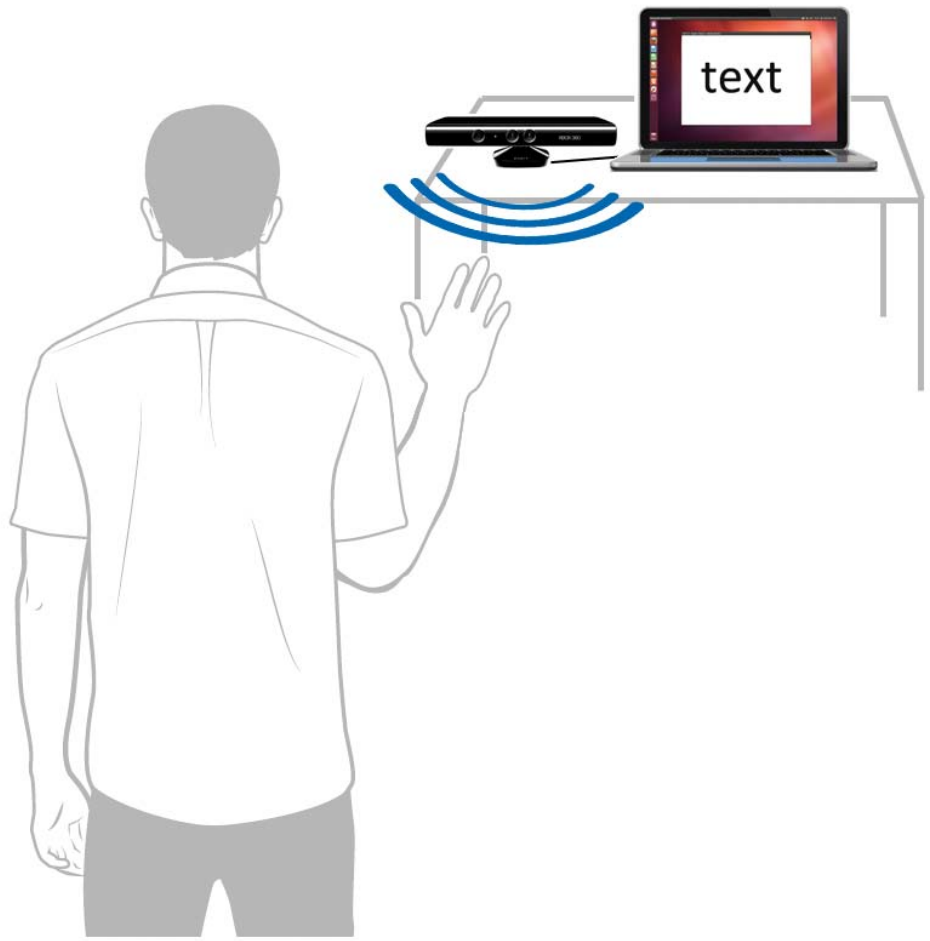


Figura 1.1: Esquema del sistema de reconeixement amb el sensor Kinect

1.2.- Objectius

Kinect obre un món de possibilitats als desenvolupadors d'arreu del món per a crear aplicacions orientades a diferents àmbits de recerca. El potencial dels algoritmes de detecció de moviment ja sigui per a control de maquinari o per al diagnòstic d'alteracions de comportament marquen clarament l'objectiu d'aquest treball final de màster; crear una plataforma de desenvolupament per a Kinect que permeti capturar i reconèixer gestos de les mans basats en la llengua de signes i el seu abecedari dactilològic.

El sistema de reconeixement que es planteja és una introducció al reconeixement de gestos, una plataforma d'experimentació i un sistema obert i totalment ampliable.

Mitjançant el sensor Kinect es capturen les dades de profunditat a temps real que seran posteriorment processades per tal de fer el reconeixement del gest de la mà de l'usuari i mostrar per pantalla la transcripció.

1.3.- Metodologia

La primera fase del treball consisteix en la recerca d'informació i l'estudi de la documentació per tal de conèixer detalladament tant el sensor Kinect com les característiques de les imatges i dades que captura, les possibilitats que ofereix i la valoració de les alternatives de software per tal d'escollir el més adient.

La segona etapa correspon a la captura d'imatges i a l'estudi detallat de formats i característiques per tal de poder-les processar.

La darrera part del treball inclou el processament de les imatges i el desenvolupament d'algoritmes per l'extracció i interpretació de les dades de profunditat que ens proporciona Kinect per tal de poder reconèixer els gestos de les mans.

Capítol 2: Fonaments i materials

"La ciència més útil és aquella que el seu fruit és el més comunicable"
– Leonardo da Vinci

2.1.- Llengua de signes

La llengua de signes és el llenguatge natural de les persones sordes que utilitzen per comunicar-se amb el seu entorn social mitjançant el canal visual i espacial. La seva estructura gramatical es caracteritza bàsicament per la configuració de les mans i dels moviments, tenint en compte la seva orientació i ubicació en l'espai.

Tot i tenir un canal de transmissió diferent, l'estructura és la mateixa que la de qualsevol llengua oral i per tant, es pot analitzar la seva dimensió fonològica, morfològica, sintàctica i discursiva.

A dia d'avui existeixen un gran nombre de llengües de signes, gairebé cada idioma parlat té la seva corresponent llengua de signes. A l'estat espanyol existeix dues llengües de signes reconegudes, la llengua de signes espanyola (LSE) i la catalana (LSC). Per a poder tenir una visió general de les característiques d'una llengua de signes, es prendrà com a referència la catalana.

2.1.1.- Els signes

Totes les llengües estan formades per signes arbitraris (paraules) que permeten la convenció que estableix una relació fixa entre significant i significat, pròpia de les llengües humanes. El signe o senya és el nom que rep aquesta convenció en les llengües de signes.

Els articuladors de la llengua de signes a més de les mans que són fonamentals, també ho són el cap, la cara i el tronc.

Segons l'estudi de la llengua de signes de l'investigador Josep Quer Villanueva [1] hi ha 5 paràmetres que componen un signe, els quals es divideixen en components manuals i components no manuals. Entre els components manuals trobem les quatre categories següents:

- ✓ **La configuració** fa referència a la forma que adopta la mà com a conseqüència de la posició dels dits.

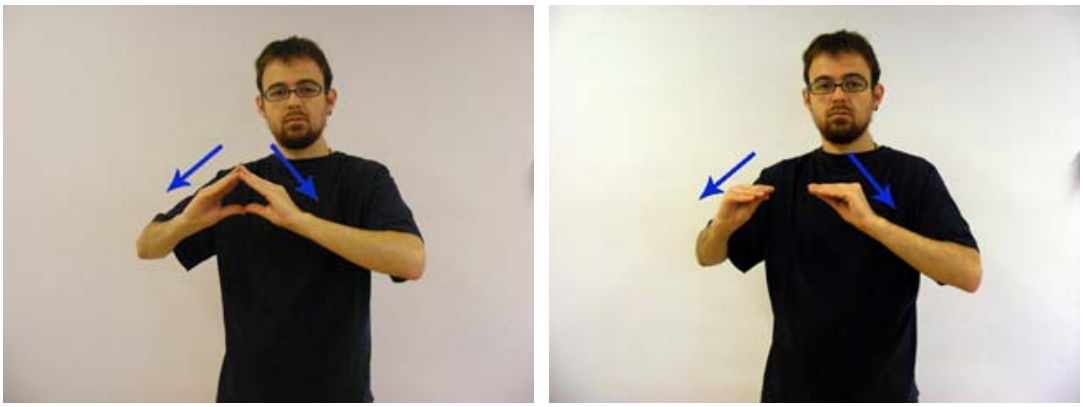


Figura 2.1: Configuració de les mans per el signe “Universitat” [2]

- ✓ El **Punt o lloc d’articulació** indica on es produeix el signe, que pot ser a la part superior del cos fent-hi contacte o no, en l’espai sígnic, que és l’espai que trobem enfront del signant, delimitat per l’extensió màxima dels braços cap a dalt per damunt del cap o cap endavant.

Figura 2.2: Punts d’articulació, (esquerra) amb contacte i (dreta) sense contacte [2]

- ✓ **Moviment,** la major part dels signes impliquen un moviment entre dos punts d’articulació.

Figura 2.3: Moviment de les mans pel signe “Examen” [2]

- ✓ **Orientació,** és el pla al qual està orientat el palmell de la mà.

Figura 2.4: Orientació del palmell de la mà pels signes (esquerra) tarda i (dreta) mare [2]

El paràmetre no manual i últim inclou totes aquelles marques necessàries per a la configuració del signe però que no són manuals.

- ✓ **Component bucal,** són els gestos realitzats amb la boca i les galtes que formen part de la descripció fonològica del signe.
- ✓ **Component parlat o oral,** gest labial que està relacionat amb el mot oral corresponent al signe.
- ✓ **Posició de les celles i del front**
- ✓ **Direcció de la mirada**
- ✓ **Posició del cos**
- ✓ **Posició i/o moviment del cap**
- ✓ **Expressió facial**

2.1.2.- Alfabet dactilològic

L'alfabet dactilològic és un tipus abecedari que s'utilitza a la llengua de signes i que consisteix en representar manualment el sistema d'escriptura corresponent a la llengua parlada. S'hi representen les grafies que corresponen als sons, a més, també s'hi poden representar accents, dos punts, etc.

L'origen de l'alfabet dactilològic està relacionat amb el domini de les llengües parlades sobre les llengües de signes. Aquests alfabets han estat inventats bàsicament perquè les persones sordes puguin accedir a les llengües parlades.

Les configuracions de la mà són anatòmicament més complicades que les que s'utilitzen per parlar en llengua de signes ja que aquestes intenten imitar les grafies de les lletres.

A la figura de a continuació podem veure l'alfabet dactilològic de la llengua de signes catalana.

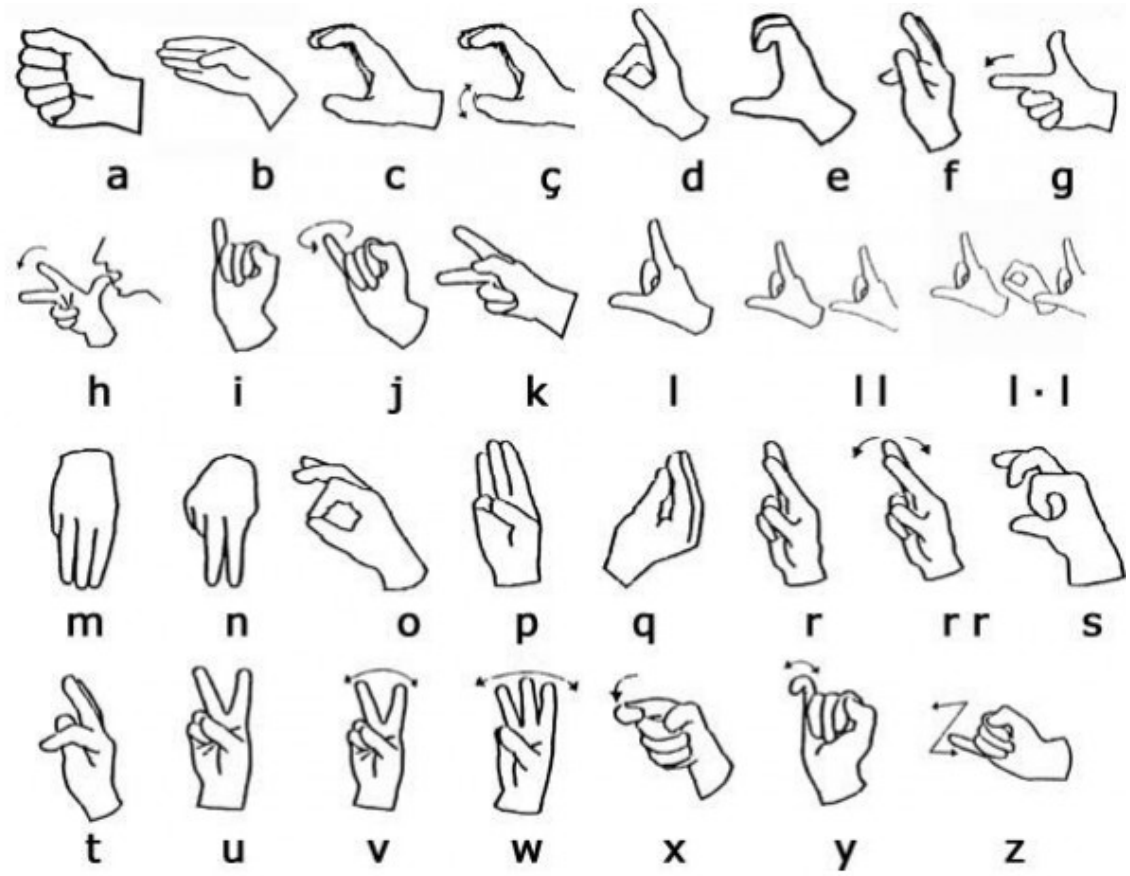


Figura 2.5: Alfabet dactilològic català

Aquest projecte realitza el reconeixement de certs gestos estàtics de la mà, prenent com a referència l'alfabet dactilològic.

2.2.- Microsoft Kinect

Kinect (figura 2.6) és un sensor de moviment desenvolupat per PrimeSense com a perifèric per a la videoconsola Microsoft Xbox 360 i llançat al mercat per Microsoft durant el novembre de 2010. Aquest dispositiu de l'estil d'una web-cam disposa de tres sensors (càmera RGB, sensor de profunditat i micròfon) que li permeten detectar moviment, de manera que l'usuari pot controlar i interactuar amb la Xbox 360 sense haver de tenir cap contacte físic amb un comandament de control tradicional mitjançant una interfície natural d'usuari que reconeix gestos, ordres de veu, imatges o objectes.

Kinect va obrir la porta al desenvolupament d'aplicacions en els camp de la Visió per Ordinadors, del Reconeixement de moviments o gestos, la Realitat Virtual o la Robòtica.

Figura 2.6: Vista frontal del sensor Microsoft Kinect

2.2.1.- Components i característiques

Kinect és un sensor format per una barra horitzontal d'uns 23 cm connectada a una base amb un eix motoritzat dissenyat per ser col·locat longitudinalment per sobre o sota del monitor o pantalla de vídeo.

El dispositiu està format per una càmera RGB, un sensor de profunditat i un micròfon multi-array que funcionen mitjançant el seu software propietari i que proporcionen la captura de moviment 3D, el reconeixement facial i de veu (figura 2.7).

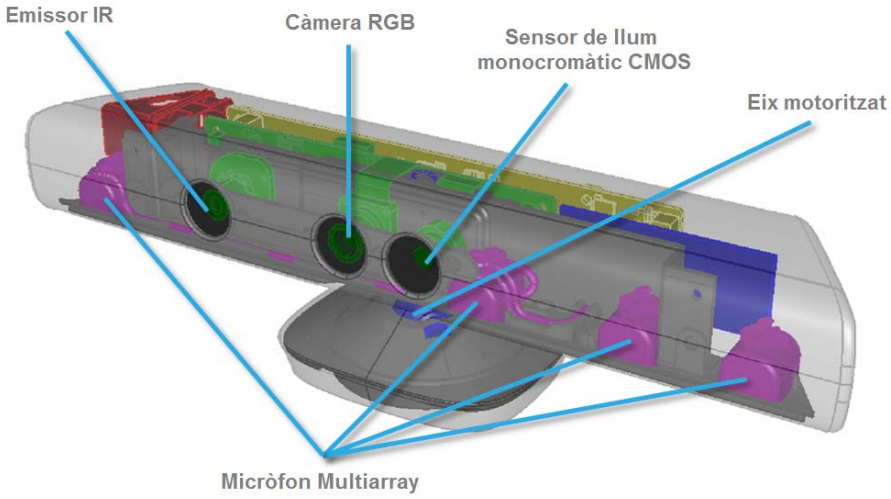


Figura 2.7: Components / Sensors de Microsoft Kinect

- ✓ **El Sensor 3D de profunditat** està format per un emissor de rajos infrarojos (IR) i un sensor de llum monocromàtic CMOS (Complimentary Metal-Oxide Semiconductor) que permeten captar l'espai en tres dimensions sota qualsevol condició de llum. El sensor de profunditat pot operar a les resolucions VGA de 80x60, 320x240, 640x480 píxels a 30fps i 11 bits de profunditat, és a dir, proporciona 2048 nivells de sensibilitat.

- ✓ **La Càmera RGB** proporciona vídeo 2D a color i disposa de resolucions de 1280x960 píxels a 12fps RGB, de 640x480 píxels a 30fps i de 640x480 píxels a 15fps Raw YUB.
- ✓ **El Micròfon Multiarray** està compost per 4 micròfons ubicats a llarg de la part inferior del sensor que permeten el reconeixement de la veu amb localització de la font acústica, la supressió dels soroll i la cancel·lació de l'eco. El flux de dades és de 16bits a 16kHz.
- ✓ **L'Eix motoritzat** ens permet inclinar el sensor verticalment en un interval de ± 27º per tal de poder canviar el camp de visió.

El camp de visió del sensor tant per vídeo RGB com per les dades de profunditat és de 57,5 graus horitzontalment i 43,5 graus verticalment.

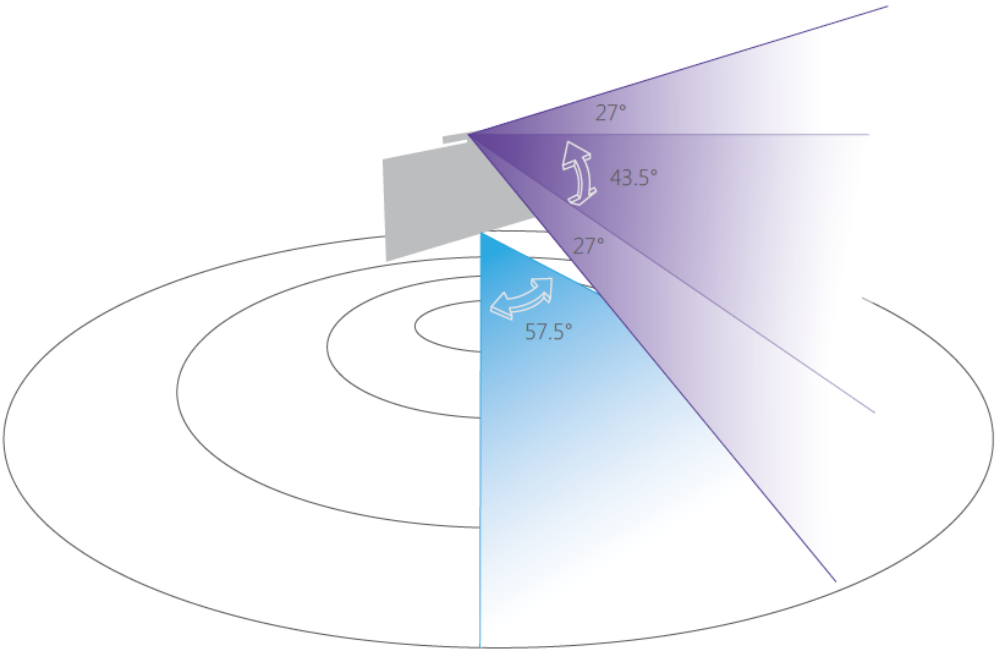


Figura 2.8: Camp de visió del sensor Kinect

2.2.2.- Controladors i SDKs

Microsoft, originalment, va comercialitzar el sensor Kinect per a ser utilitzat únicament amb la seva videoconsola Microsoft Xbox 360. El port USB 2.0 del que disposa per la connexió amb la consola es va deixar obert, de manera que s'ampliaven les seves funcionalitats si s'utilitzava juntament amb un ordinador. Des d'aleshores van començar a aparèixer diferents controladors de codi obert, SDKs i APIs per a poder desenvolupar aplicacions per a altres dispositius fins que a l'any 2011 Microsoft va llançar el seu propi paquet per a desenvolupadors (SDK – Software Development Kit).

2.2.2.1.- Controladors Open Souce i SDKs

A Internet podem trobar una gran varietat de controladors per a Kinect, la taula següent (Taula 2.1) mostra una comparativa dels controladors més utilitzats de codi obert:

Nom	Llenguatge	Plataformes	Característiques	Llicència
OpenKinect - libfreenect [6]	Python C Synchronous, Actionscript C++, C# Java (JNA i JNI), Javascript ComonLisp Glib	Linux (Ubuntu...) Windows Mac OS X	* Imatges RGB i de profunditat * Dades d'acceleròmetre * Control de LED i Motor * Utilitat Simulator: Simulador Fakenet Kinect * Utilitat Record: gravació en arxiu de les dades RGB, profunditat i d'acceleròmetre	Apache 2.0 GPL v2
CL NUI Platform - SDK i Driver [7]	C C++ WPF/C# Java Actionscript	Windows	* Imatges RGB i de profunditat * Dades d'acceleròmetre * Control de LED i Motor	GNU GPL
OpenNI - NITE Middleware [8]	C C++	Windows Linux Ubuntu	* Imatges RGB i de profunditat * Gravació en arxiu de les dades RGB i de profunditat * Identificació d'usuari * Funció de detecció * Reconeixement de gestos * Dades de l'esquelet	GNU LGPL
Robot Operating System (ROS) - Kinect [9]	Python, C++	UNIX	* Imatges RGB i de profunditat * Control de LED i Motor	BSD

Taula 2.1: Comparativa de controladors de codi obert per a Kinect

- ✓ **OpenKinect [6]** és una comunitat de persones i desenvolupadors interessats en l'ús de la càmera Kinect amb ordinadors i altres dispositius que treballen en el desenvolupament de llibreries lliures de codi obert que permeten el seu ús amb els sistemes operatius Windows, Linux i Mac OS. L'objectiu principal de la comunitat és la seva llibreria **libfreenect** que permet adquirir imatges RGB i de profunditat, el control de l'eix motoritzat, del LED i de l'acceleròmetre.
- ✓ **CL NUI Platform [7]** és un paquet de controlador i SDK de codi obert autoinstal·lable i de fàcil ús desenvolupat per Code Laboratories per a dispositius Windows.
- ✓ **OpenNI [8]** és un *framework* que ha estat desenvolupat per l'empresa encarregada de fabricar el sensor Kinect, PrimeSense, i que permet, com els controladors anteriors, el control de la majoria de components de la Kinect. Una de les característiques més interessants d'aquest paquet és la compatibilitat amb la llibreria **NITE** també de PrimeSense que conté un conjunt d'algoritmes pel processament d'imatges, detecció i seguiment d'individus.

- ✓ **Robot Operating System (ROS) Kinect [9],** ROS és un meta-sistema operatiu de codi obert per a robots. La comunitat de desenvolupadors responsable del projecte han desenvolupat *ROS Kinect*, projecte enfocat a la integració del sensor amb la seva plataforma i que conté controladors de baix nivell basats en OpenKinect i OpenNI.

L'algoritme per a la detecció i reconeixement de gestos de les mans descrit en aquest projecte està basat en l'anàlisi de les dades de profunditat que ens ofereix el sensor. Qualsevol dels controladors i llibreries anteriors ens permeten l'accés a aquesta informació, de manera que, amb la voluntat de que el programa fos de codi obert i ampliable en un futur, s'ha decidit utilitzar la llibreria de OpenKinect libfreenect corrent sota Ubuntu i amb el llenguatge de programació orientat a objectes Python.

2.2.2.2.- OpenKinect – Libfreenect

El software ***libfreenect*** inclou totes les eines necessàries per activar, inicialitzar i comunicar-nos amb el sensor Kinect. Inclou els controladors i una API multi plataforma que funciona amb Windows, Linux i Mac OSX. *Libfreenect* té extensions per als següents llenguatges de programació:

- ✓ Python
- ✓ C Synchronous
- ✓ Actionscript
- ✓ C++
- ✓ C#
- ✓ Java (JNA i JNI)
- ✓ Javascript
- ✓ Common Lisp
- ✓ GFreenect (GLib)

Actualment la llibreria permet l'accés a les següents dades que ens proporciona el sensor Kinect:

- ✓ Imatges RGB i de profunditat
- ✓ Motors
- ✓ Acceleròmetre
- ✓ LED

Pel que fa a l'àudio, actualment *libfreenect* no permet l'adquisició d'aquestes dades, tot i que la comunitat està treballant perquè això sigui possible en un futur proper.

El projecte *libfreenect* de codi obert està sota dues llicències, de manera que els usuaris poden optar per treballar sota la llicència Apache 2.0 o la GPL v2.

Tal i com s'ha comentat en paràgrafs anteriors, aquest paquet ens permet adquirir vídeo RGB i dades de profunditat, tal i com es pot comprovar en la figura següent, que correspon a l'execució d'un dels exemples que ens proporciona el paquet.

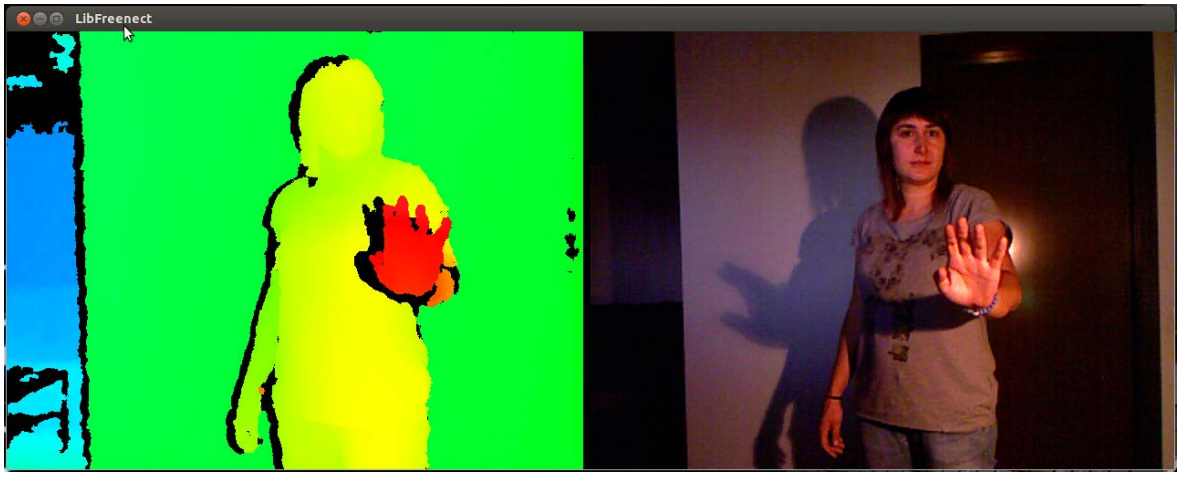


Figura 2.9: Captura RGB i de profunditat mitjançant el paquet libfreenect de Openkinect

2.2.2.3.- Microsoft Kinect SDK

Durant el segon trimestre de 2011 Microsoft va llançar el seu propi paquet per a desenvolupadors (SDK – Software Development Kit) en fase de proves (Beta) per utilitzar-la sota Windows [10].

El paquet per desenvolupadors Kinect for Windows SDK proporciona les eines i APIs necessàries per a desenvolupar aplicacions de reconeixement de gestos i veu en C++, C# o Visual Basic utilitzant Microsoft Visual Studio 2010.

L'arquitectura del paquet per a desenvolupadors de Microsoft SDK està formada per 5 capes, (1) Hardware, (2) Controladors, (3) Components d'àudio i vídeo, (4) DirectX Media Object (DMO) pel micròfon i (5) les APIs estàndards de Windows 7. Al nivell superior hi hauria l'aplicació desenvolupada amb Visual Studio (C++, C# o Visual Basic).

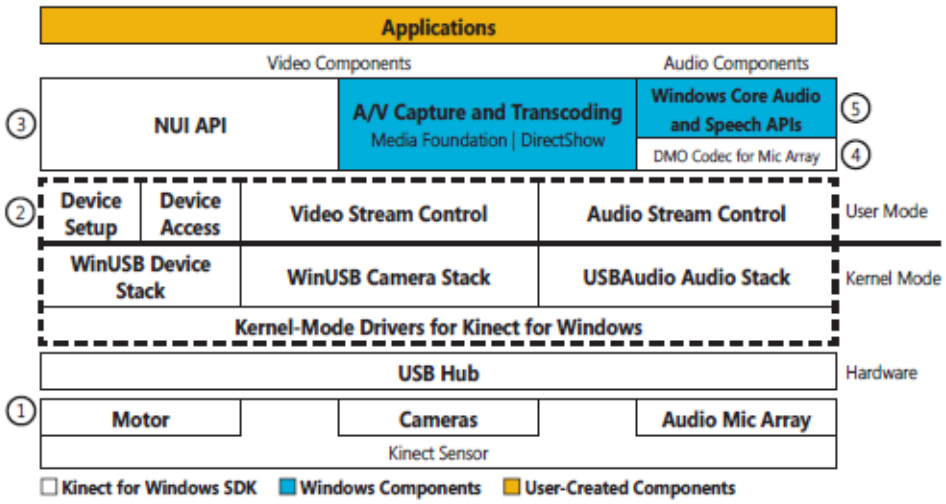


Figura 2.10: Arquitectura de l'SDK Microsoft Kinect

La darrera versió del paquet llançada durant el maig de 2012 (Microsoft Kinect SDK v1.5) ofereix eines i capacitats avançades de reconeixement superiors a les que ens proporcionen els paquets de codi obert descrits a l'apartat anterior, com pot ser el seguiment de l'esquelet d'una persona asseguda.

A la imatge següent es mostra l'execució d'un dels exemples demostratius que incorpora el paquet de desenvolupament de Microsoft, on es poden testejar el vídeo RGB i les dades de profunditat a diferents resolucions i freqüències en fotogrames per segon, l'àudio, el control de l'eix motoritzat i la funció de reconeixement de l'esquelet.

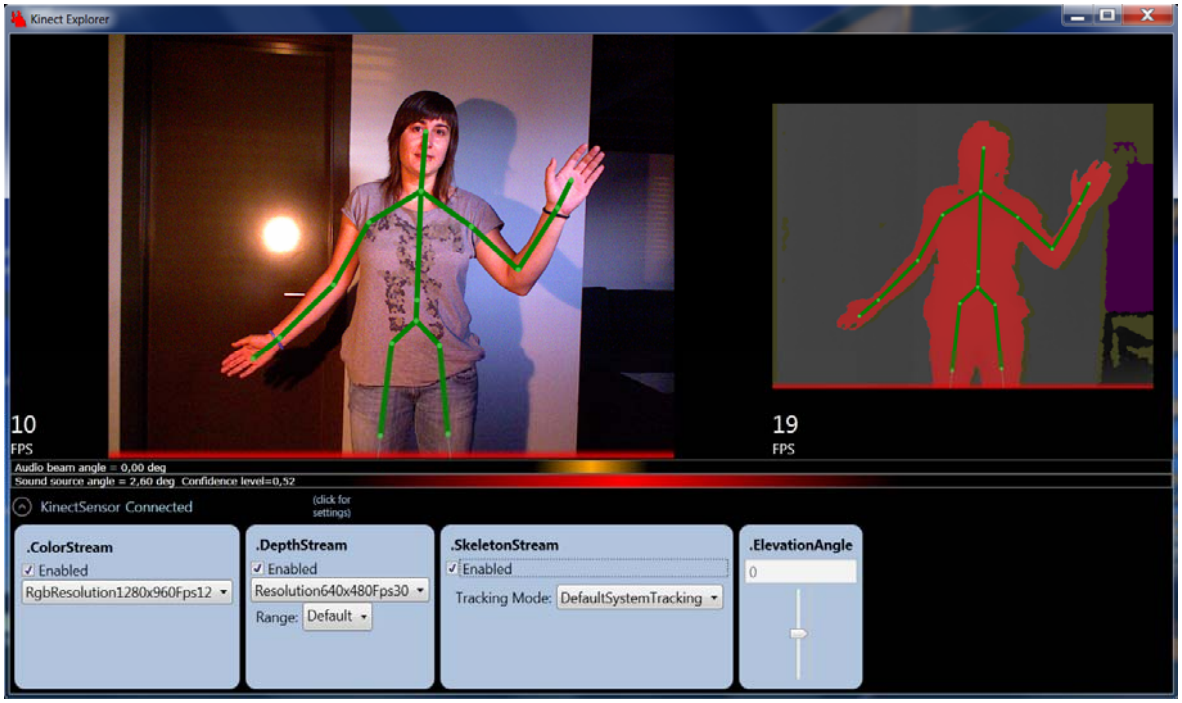


Figura 2.11: Aplicació demostrativa de les funcionalitats del paquet Microsoft Kinect SDK

2.2.3.- Dades de profunditat

La percepció de profunditat és un dels grans avantatges del sensor Kinect respecte altres dispositius de la seva família.

Kinect ens proporciona totes les dades necessàries per fer un anàlisi detallat de les captures, amb la càmera RGB es pot captar la imatge en dues dimensions per obtenir una realitat precisa de l'entorn, i amb el sensor de profunditat la tercera dimensió.

La captura de la informació de profunditat es realitza mitjançant la projecció d'un patró de punts infrarojos gràcies a l'emissor IR i la posterior captura de la llum infraroja projectada pel sensor de llum monocromàtic CMOS, que serà processada pel microprocessador de la càmera per tal de poder veure la imatge de profunditat de l'escena (figura 2.12).

Figura 2.12: Captura de profunditat del sensor PS1080 de PrimeSense [11]

Les dades de profunditat ens proporcionen la distància en metres i l'usuari (Player) per a cada píxel de la imatge de l'escena. La distància d'un objecte es representa amb la següent escala de colors (figura 2.13) segons el mode de detecció configurat (modes disponibles quan es treballa amb el paquet per a desenvolupadors de Microsoft o amb la vídeo consola Xbox).

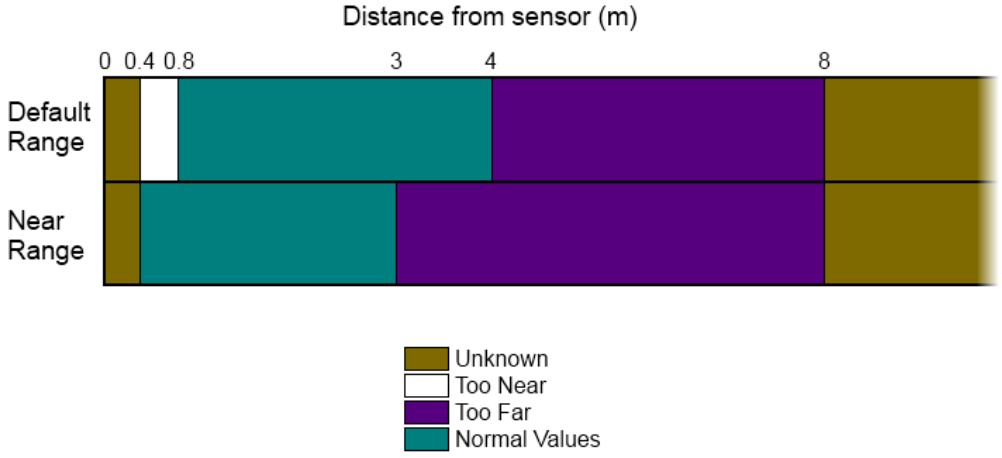


Figura 2.13: Escala de colors segons la distància d'un objecte [5]

El sensor Kinect, tot i tenir un interval límit recomanat de 1.2 a 3.5 metres utilitzat en el mode per defecte pot mantenir el seguiment fora d'aquests marges sempre que no es superin els seus límits físics.

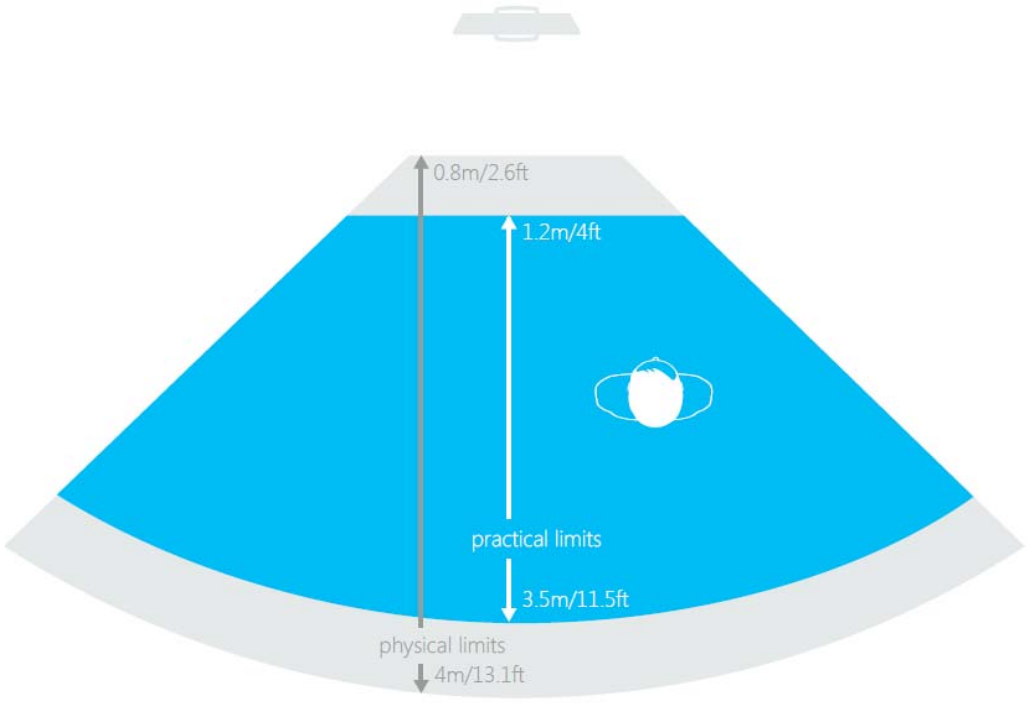


Figura 2.14: Marge de distàncies de profunditat

La percepció de profunditat permet focalitzar l'objecte del que es vol fer el seguiment. Permet una correcte identificació de l'objecte respecte altres elements de l'escena que estan situats a una distància diferent i que per tant, poden ser omesos. A la imatge següent es pot veure la representació de la profunditat capturada pel sensor segons l'escala de colors anteriorment descrita.

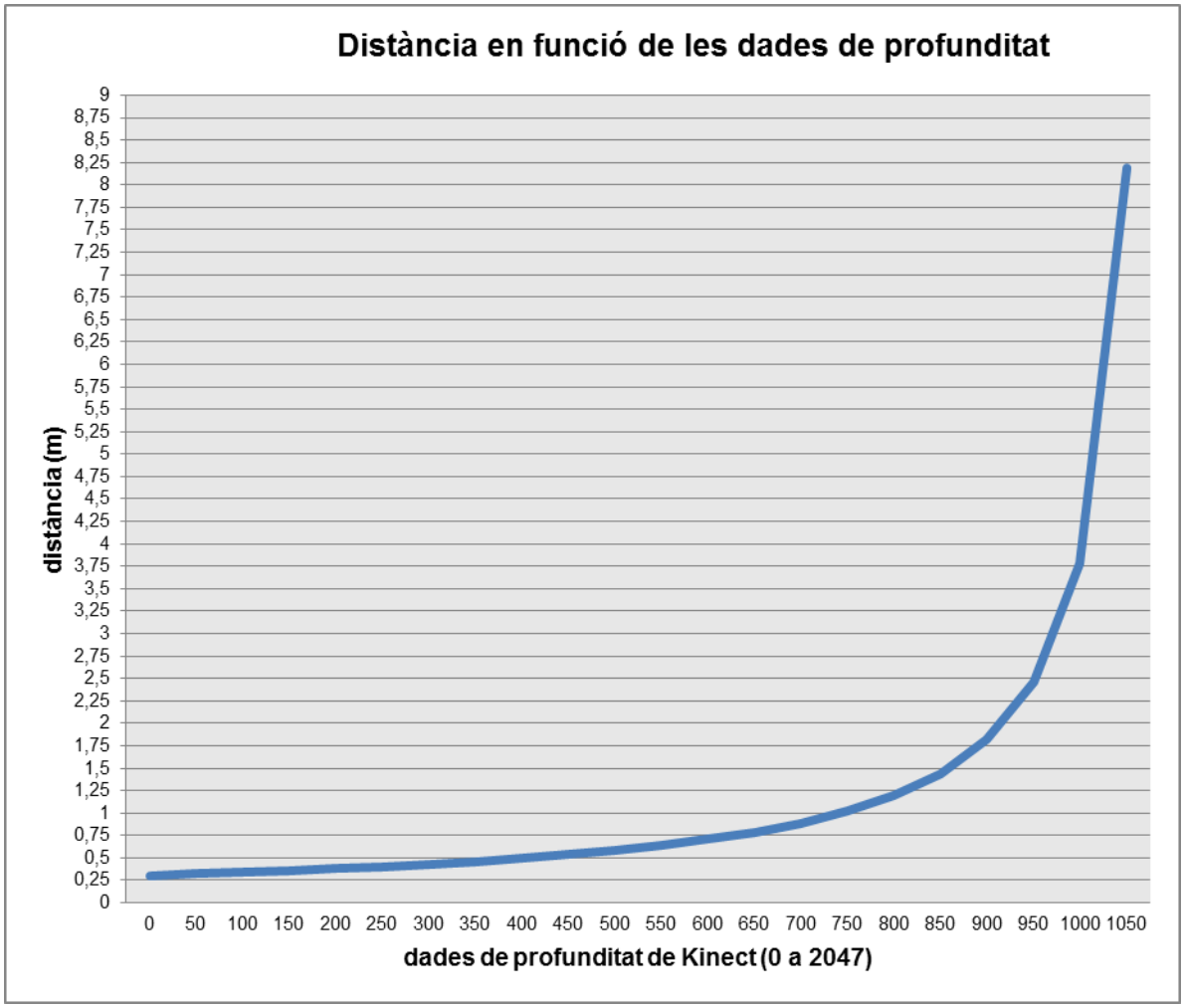
Figura 2.15: Representació de profunditat mitjançant Microsoft Kinect SDK

Com ja s'ha definit en l'apartat 2.2.1 el sensor ens proporciona les dades de profunditat a una resolució de fins a 640x480 píxels a 30 fotogrames per segon i a 11 bits de precisió el que significa que disposem de 2048 nivells de sensibilitat. Per tant, els elements de la matriu 640x480 amb la que es poden representar les dades de profunditat tindran valors en l'interval de 0 al 2047. Quan el valor d'un element és 2047 significa que el sensor no veu la reflexió del raig infraroig o no hi ha informació de profunditat.

Per a poder calcular la distància d'un objecte existeixen varies aproximacions. A la pàgina web oficial de la comunitat OpenKinect proposen el següent mètode d'aproximació de distàncies [6]

$$distància = 0.1236 * \tan\left(\frac{rawDisparity}{2842.5} + 1.1863\right) \quad en\ metres \qquad (1)$$

De la fórmula anterior en podem aproximar una corba representant la distància d'un objecte en funció de les dades de profunditat capturades pel sensor.



A la gràfica anterior es pot veure que com més llunyà és un objecte més difícil és determinar la distància que hi ha entre ell i el sensor.

La profunditat doncs, és una característica que permet aconseguir l'extracció de la mà com a objecte de seguiment, però per obtenir més informació i poder fer el reconeixement de gestos és necessari processar aquestes dades.

2.3.- Llibreries de processament d'imatges

2.3.1.- Llibreries Open Source

Existeixen varies llibreries de codi obert que ofereixen gran quantitat de funcions per al processament d'imatges i facilitar-ne l'extracció d'informació, entre elles, destaquen les següents:

- ✓ **OpenCV [12]** (Open Source Computer Vision) és una llibreria de funcions per a visió artificial a temps real de codi obert, multi-plataforma i desenvolupada en C.
- ✓ **OpenNI - NITE [13][14]** és un middleware multi-plataforma desenvolupat per treballar juntament amb el framework OpenNI de PrimeSense que conté un conjunt d'algoritmes pel processament d'imatges, detecció i seguiment d'individus.
- ✓ **Java Advanced Imaging (JAI) [15]** proporciona un conjunt d'eines per a la programació orientada a objectes que permeten la manipulació d'imatges.

A la taula següent es fa una comparativa de les llibreries, compatibilitat amb diferents sistemes operatius, llenguatges de programació i tipus de llicència.

Nom	Llenguatge	Plataformes	Llicència
OpenCV	Python C C++ Java	Linux Windows Mac OS X Android	BSD
OpenNI - NITE	C# C++ Java	Linux Windows Mac OS X Android	GNU GPL
Java Advanced Imaging (JAI)	Java	Linux Windows Mac OS X Solaris (UNIX)	GNU GPL

Taula 2.2: Comparativa de llibreries de processament d'imatges

Com ja s'ha comentat en l'apartat anterior, el projecte està desenvolupat amb el llenguatge de programació Python, de manera, que vista la taula anterior, OpenCV és l'única llibreria compatible però que proporciona totes les eines necessàries pel processat de les imatges capturades amb Kinect.

2.3.2.- OpenCV

OpenCV [12] (Open Source Computer Vision) és una llibreria de funcions per a visió artificial a temps real. Aquest *framework* de processament d'imatges de codi obert està sota llicència BSD i és gratuït tant per ús acadèmic com comercial.

La llibreria originalment es va desenvolupar en C, però més tard i amb l'objectiu d'arribar a un major nombre de desenvolupadors, es va adaptar per poder treballar amb C++, Python i Java, a més, és multi plataforma, existeixen versions per a Windows, Linux, Mac i Android.

OpenCV és una de les llibreries més populars ja que compte amb més de 500 algoritmes optimitzats (figura 2.3) i més de 2.5 milions de descàrregues. Les seves funcions engloben tots els camps de la visió artificial, com el reconeixement d'objectes, facial o de gestos, interacció Home-Computadora (HCI), la visió estereoscòpica, visió robòtica i calibratge de càmeres, entre d'altres.

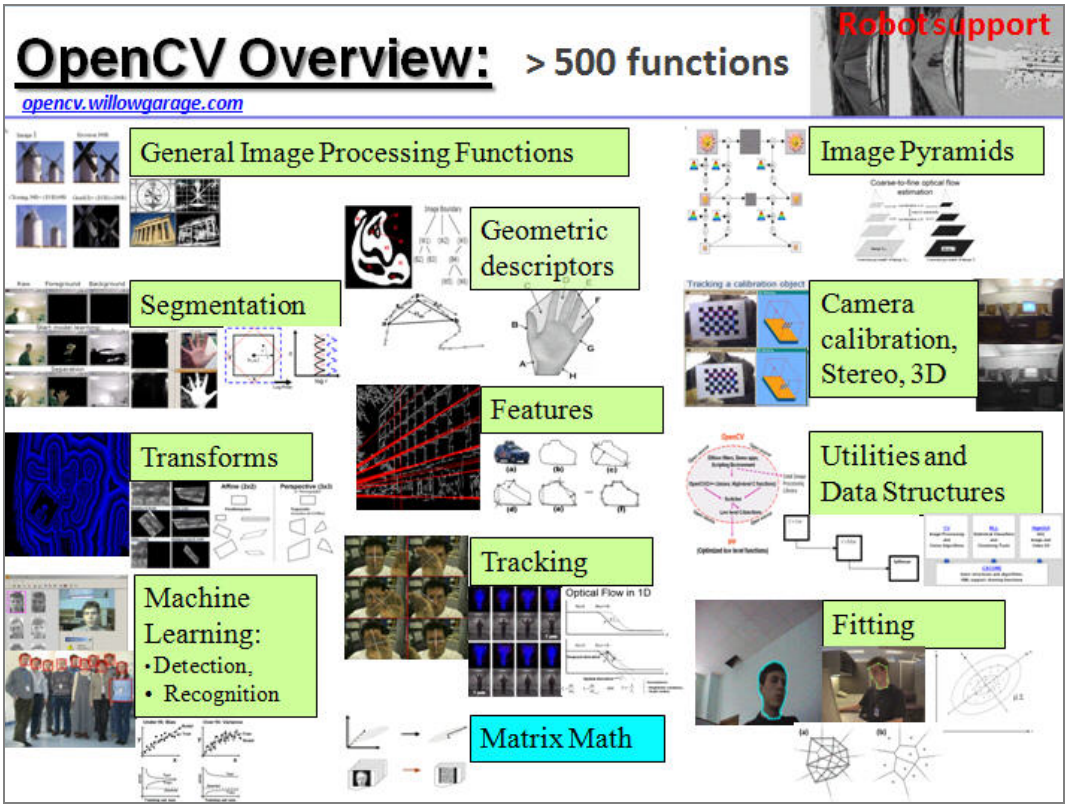


Figura 2.16: Grups de funcions de la llibreria OpenCV [12]

OpenCV té una estructura modular, el que vol dir que el paquet està format per diverses llibreries o mòduls, que són els següents:

- ✓ **core,** mòdul que defineix les estructures de dades bàsiques i les funcions bàsiques que utilitzen tots els mòduls.

- ✓ **imgproc,** mòdul de processament d'imatge filtratge, transformacions geomètriques, conversió de formats i histogrames entre d'altres.
- ✓ **video,** mòdul d'anàlisi de vídeo que inclou estimació de moviment, subtracció de fons i algoritmes de seguiment d'objectes.
- ✓ **calib3d,** algoritmes bàsics de geometria multi-visió, calibratge de càmeres simples o estereoscòpiques, estimació de la posició d'objectes i reconstrucció 3d entre d'altres.
- ✓ **features2d,** detectors de característiques, descriptors i comparadors
- ✓ **objdetect,** detecció d'objectes de classes predefinides (cares, ulls, gerres, gent, cotxes...)
- ✓ **highgui,** una interfície de fàcil ús per la captura de vídeo amb capacitats UI senzilles.
- ✓ **gpu,** algoritmes d'acceleració de GPU dels diferents mòduls de OpenCV
- ✓ .... altres mòduls auxiliars com Flann (Fast Library for Approximate Nearest Neighbors) i Google, els enllaços a Python i d'altres.

En aquest projecte s'utilitzen gran part dels mòduls que ens proporciona la llibreria, entre ells destacar els mòduls *core, imgproc i highgui.*

2.4.- Reconeixement de gestos

2.4.1.- Captura de moviment

La captura de moviment (CM) o *Motion tracking* en anglès, és el procés que consisteix en traduir moviments reals en representacions digitals mitjançant el seguiment d'un conjunt de punts d'interès en una escena per un temps determinat. [16]

La CM pot realitzar-se sobre qualsevol objecte o persona que tingui moviment. Els punts d'interès normalment corresponen a connexions entre parts rígides de l'actor, que en el cas d'una persona podrien ser articulacions del cos, tals com, genolls, colzes, espatlles entre d'altres.

Existeixen diferents tècniques per realitzar la captura de moviment, entre elles, els sistemes òptics són dels més utilitzats. Aquests sistemes utilitzen marcadors reflectius o constituïts per LEDs col·locats sobre l'actor per destacar els punts d'interès dels que se'n vol fer el seguiment.

Actualment, gràcies al desenvolupament de tècniques en el camp de la visió artificial s'han desenvolupat sistemes que no necessiten l'ús de marcadors, ja que s'utilitzen algoritmes que mitjançant l'entrada de dades proporcionades per les càmeres poden identificar figures humanes i realitzar-ne el seguiment. Alguns sistemes utilitzen múltiples càmeres que digitalitzen diferents vistes de l'escena, que mitjançant tècniques d'estereoscòpia són utilitzades per determinar-ne els punts.

Com ja s'ha vist en l'apartat 2.2 del present capítol, la càmera Kinect permet la percepció de profunditat o tercera dimensió mitjançant el sensor d'infrarojos. Aquestes dades s'utilitzen per realitzar la captura de moviment en videojocs i gràcies a l'alliberament del port USB del que disposa, també es poden utilitzar en altres camps mitjançant l'ús del PC.

Dispositius com la Kinect, faciliten, a dia d'avui, el desenvolupament en el camp de la visió artificial i la captura de moviment, degut al seu reduït cost i la seva compatibilitat amb els ordinadors.

Les llibreries ja descrites en apartats anteriors ens proporcionen una sèrie d'algoritmes per a facilitar el processament de les imatges capturades pel sensor i detectar els objectes o punts d'interès per tal de poder-ne fer el reconeixement.

En el camp del reconeixement de gestos, hi ha dues grans arees, la del reconeixement de gestos dinàmics i la dels gestos estàtics. Aquestes ens determinen quins algoritmes fonamentals utilitzar.

Pel que fa al reconeixement de gestos dinàmics, és a dir, d'una seqüència d'imatges que formen un moviment, és habitual treballar amb la detecció i seguiment de l'esquelet, que proporciona informació relativa als punts d'interès del cos humà, com poden ser el cap, les articulacions i les extremitats.

En el cas del reconeixement de gestos estàtics amb les mans no és necessàri l'utilització de les dades de l'esquelet, ja que a conseqüència de la resolució del sensor de profunditat, l'algoritme no pot proporcionar informació precisa dels punts que determinen la postura o gest de la mà. En aquest cas, s'utilitzaren tècniques de segmentació per tal de d'eliminar tota la informació no relativa a les mans i mitjançant la detecció de contorns s'analitza el perfil de la mà per determinar els punts d'interès i així poder reconeixer els gestos.

2.4.2.- Seguiment de l'esquelet

Simplificar la forma d'un objecte o reduir la quantitat de dades que aquest conté ha estat sempre un camp d'interès en el processament d'imatges.

Una de les tècniques més utilitzades és l'esqueletització de les imatges. Obtenir l'esquelet d'una imatge significa trobar un patró a partir de la forma de l'objecte que té menor quantitat de píxels però que conserva continuïtat.

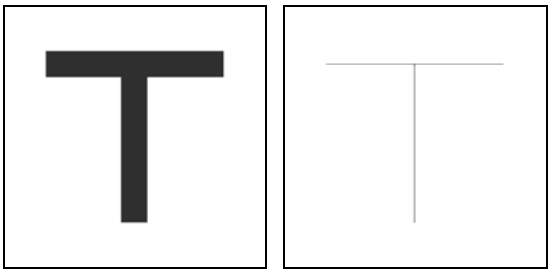


Figura 2.17: Patró en forma de T i el seu respectiu esquelet

També es pot veure l'esquelet com el lloc on convergeixen els centres dels cercles bitangencials que caben de manera complerta dins de la regió considerada.

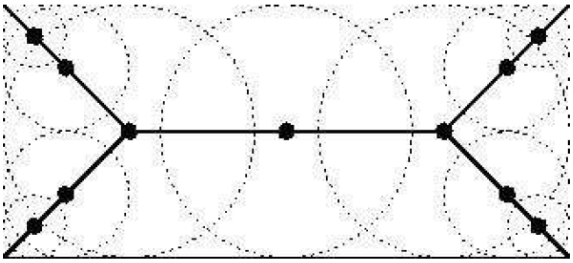


Figura 2.18: Esquelet d'un rectangle definit pels cercles bitangencials

Existeixen varis algoritmes que s'utilitzen per a la determinació de l'esquelet d'un objecte, a [29] Fredy Carranza ens descriu els més utilitzats.

- ✓ **Diagrama de Voronoi** és una estructura geomètrica que representa informació de proximitat sobre un conjunt de punts o objectes, és a dir, el diagrama de Vornoi d'un conjunt d'objectes geomètrics és una partició de l'espai en cel·les cada una de les quals conté una contigüitat amb els seus punts més propers
- ✓ **Algoritme de Zhang – Suen** requereix desplaçar una màscara o plantilla 3x3 per tota la imatge, de tal manera que els píxels coberts per la plantilla seran analitzats per certes condicions per definir si han de ser eliminats o no. En aquest algoritme s'utilitzen dues subiteracions, cada una d'elles s'encarrega d'eliminar píxels de zones determinades de la plantilla.
- ✓ **Algoritme de Holt,** sorgeix de canviar les condicions que imposa l'algoritme de Zhang-Suen en expressions lògiques, de manera que els patrons i mascares passen a ser equacions.

En el camp del visió artificial i captura de moviment l'esquelet consisteix en un conjunt de punts en l'espai 3D que representen les articulacions de la persona.

Les llibreries de processament d'imatges i visió artificial ofereixen algoritmes de detecció i seguiment de l'esquelet i les articulacions per a poder fer el reconeixement de moviment i de postures de les persones.

El paquet per desenvolupadors de Microsoft permet reconèixer fins a sis usuaris dins del camp de visió del sensor, i d'aquests, en dos se'ls hi pot fer el seguiment de l'esquelet. [5]

Aquest algoritme de seguiment de l'esquelet proporciona les coordenades X, Y i Z de la posició de fins a vint articulacions del cos de l'usuari, tant si està dret com assegut.

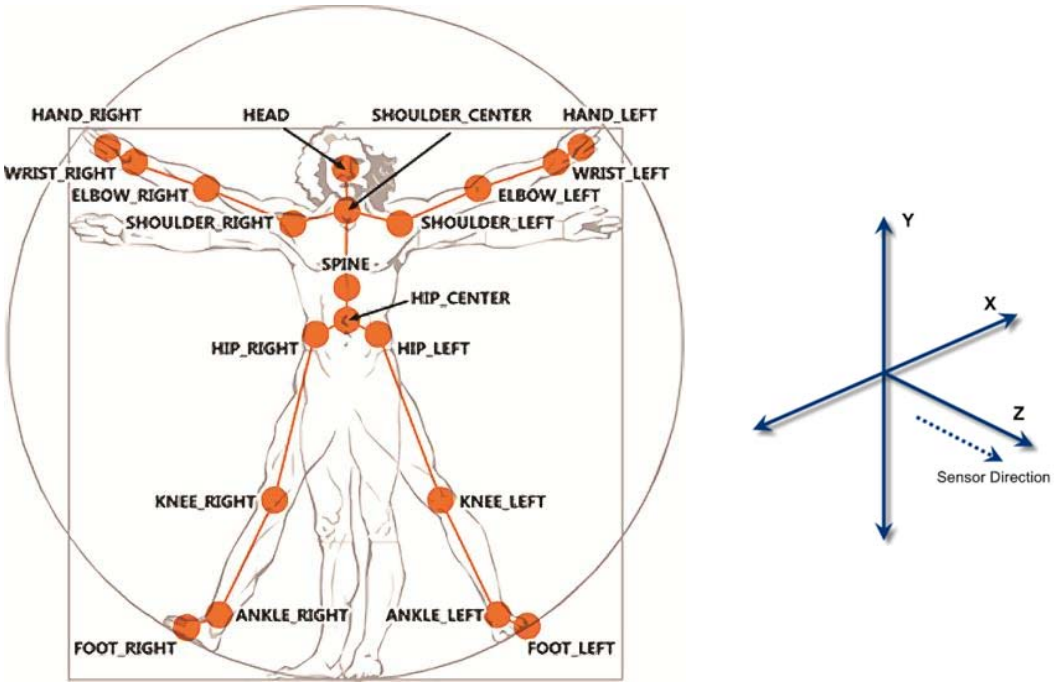


Figura 2.19: Articulacions de l'esquelet de Microsoft Kinect SDK i sistema de coordenades

La funció ens retorna una matriu amb la postura de l'esquelet formada per vectors de 4 elements (x, y, z i w) que representen cadascuna de les posicions de les articulacions. Els elements x, y i z són les coordenades expressades en metres i w fa referència a la distància del pla del terra a l'origen, el seu valor sempre és 1.

2.4.3.- Segmentació

La segmentació és una tècnica molt comuna en el processament d'imatges. Generalment s'utilitza per extreure d'una imatge aquells elements d'interès per a poder ser analitzats de manera aïllada.

La segmentació consisteix en la partició d'una imatge en regions homogènies segons una o més característiques, com poden ser la forma, el color, el brillo, la textura o el moviment. Aquest procés normalment té lloc després d'una etapa de preprocessat, com pot ser l'eliminació de soroll (filtratge), i és un procés fonamental per l'anàlisi d'imatges.

Per a la segmentació d'imatges s'utilitzen tres conceptes bàsics, a Carlos Platero [17]:

- ✓ **Semblança:** els píxels agrupats han de ser semblants respecte algun criteri (nivells de gris, color, contorn, textura...)
- ✓ **Connectivitat:** els objectes corresponen amb regions contínues de píxels.
- ✓ **Discontinuïtat:** els objectes tenen formes geomètriques que defineixen contorns. Aquests contorns delimiten un objecte dels altres.

A la pràctica però, aquestes condicions resulten gairebé impossibles d'aplicar degut a factors com són l'aparició de sorolls, la falta d'il·luminació sobre l'escenari o les ombres

dels objectes i en fan molt difícil l'acotació i resultant criteris ineficients en escenes complexes. De totes maneres, l'ús del coneixement de la imatge (color, forma, dimensions de l'objecte) redueix la complexitat del procés, i en altres casos, es poden utilitzar sensors que realcin els objectes d'interès.

Degut a la complexitat del procés, la segmentació continua essent un camp de recerca i investigació que a dia d'avui combinen tècniques basades en la localització de regions uniformes i contorns d'objectes.

A continuació s'enumeren cinc de les tècniques més utilitzades [18]:

- ✓ **Segmentació basada en llindars,** és una tècnica molt utilitzada que s'aplica quan hi ha una clara diferència entre els objectes a extreure i el fons de l'escena. Es regeix bàsicament amb la semblança entre els píxels que formen un objecte i la diferència amb la resta. En aquest cas és important que l'escena tingui un fons uniforme i objectes semblants.

  Quan s'aplica un llindar T la imatge a escales de grisos F(m,n) quedarà binaritzada, etiquetant amb "1" els píxels corresponents a l'objecte i amb "0" aquells que siguin el fons.

- ✓ El **Mètode Ostu (de Nobuyuki Otsu)** és un dels mètodes més populars per la segmentació a partir d'un llindar que s'obté de forma no paramètrica, maximitzant la variància entre classes (between-class-variance) mitjançant una cerca exhaustiva.

- ✓ **Segmentació basada en projeccions o histogrames laterals,** es basa en les projeccions dels nivells de gris sobre els eixos de coordenades, anomenades també histogrames laterals, histograma vertical sumant les aportacions per cada fila i horitzontal per cada columna

- ✓ **Segmentació basada en creixement de regions,** aquest tipus de segmentació consisteix en realitzar una partició de la imatge en K regions que tinguin les següents propietats:

  - o Les regions obtingudes de la partició han de ser disjuntes
  - o La seva unió ha de ser la imatge completa
  - o Els píxels de cada regió han d'estar connectats (regió connexa)
  - o S'ha de verificar que **P(Ri) = VERDADER I P(Ri∪Rj) = FALS**, per regions adjacents **Ri i Rj**, essent **P** el predicat que proporciona el test de homogeneïtat de la regió.

  El creixement de regions consisteix en prendre un píxel o conjunt de píxels com a regió inicial "llavor" (almenys una llavor per regió) i a partir d'aquesta fer créixer les regions incorporant píxels segons els criteris predefinits.

- ✓ **Segmentació basada en divisió i fusió (Split & merge, quad-tree),** a l'invers que els cas anterior que es parteix d'una llavor, aquests comencen amb la imatge sencera la que s'anirà dividint per quatre formant unitats més petites, així

successivament fins a ser homogènies. Aquest procediment recursiu es pot representar amb un arbre quaternari (quad-tree).

En el camp del reconeixement de gestos amb les mans hi ha diferents mètodes per segmentar les imatges o eliminar-ne el fons, són habituals la segmentació basada en la detecció de la pell pel color (Skin Segmentation) i, la basada en la distància en que es troba l'objecte utilitzant les dades de profunditat proporcionades pel sensor i aplicant la segmentació per llindar, aquest darrer mètode és el que s'ha utilitzat en el projecte.

2.4.3.1.- Segmentació de la pell per color

La segmentació de la pell té per objectiu detectar les regions de la pell humana en una imatge. És una tècnica popular en el camp del seguiment o detecció de parts del cos humà, sobretot les cares i les mans.

Les tècniques de segmentació de la pell impliquen la classificació dels píxels de la imatge per la seva informació de color. El fonament d'aquest enfocament és que el color de la pell humana és particular i diferent del de molts altres objectes.

Existeixen diferents mètodes i algoritmes de segmentació per colors, basats en classificadors lineals, bayesians, gaussians i també, en perceptrons multicapa, aquests són comparats per *Son Lam Phung, Abdesselam Bouzerdoum i Douglas Chai en el seu article Skin Segmentation Using Color Pixel Classification: Analysis and Comparison* [22].

L'ús de la segmentació per color fa difícil separar les mans de la cara ja que tenen un color molt similar, a més el seu rendiment canvia si les condicions d'il·luminació varien o el fons no és uniforme. Una solució als inconvenients anteriors seria utilitzar guants que facilitessin aquesta segmentació, però que implicaria una mala experiència d'usuari.

Els enfocaments basats en les dades de profunditat permeten evitar els problemes anteriors, és per aquest motiu que s'ha escollit aquesta opció per a la segmentació de les mans.

2.4.3.2.- Segmentació per profunditat

A dia d'avui existeixen diferents sistemes de reconeixement de gestos que utilitzen les dades de profunditat, sistemes bastats en imatges estereoscòpiques, sistemes que utilitzen sensors TOF (Time Of Flight) que estimen la distància calculant el temps transcorregut entre l'emissió i la recepció d'un raig infraroig i recentment sistemes basats en sensors de l'estil de Kinect.

Les dades de profunditat captades per aquests dispositius han de passar per un procés de segmentació per tal de poder identificar els objectes que es volen processar.

Tal i com s'introdueix en l'apartat anterior hi ha varis algoritmes de segmentació d'imatges que generalment es basen en discontinuïtats i semblances dels píxels.

En el cas de la segmentació quan s'utilitzen dades de profunditat s'utilitza en molts casos el mètode de ***Segmentació basada en llindars***, que consisteix en definir un llindar que permet seleccionar els píxels de l'objecte i eliminar la resta.

El càlcul de la imatge resultat de la segmentació és molt simple, ja que només consisteix en una comparació numèrica segons el valor llindar *t* preestablert. Quan s'aplica un llindar *t* la imatge original *im(x, y)* quedarà binaritzada *bin(x,y)* etiquetant amb el valor "1" els píxels corresponents a l'objecte i amb "0" la resta que són el fons.

$$bin(x,y) = \begin{cases} 1 & si\ im(x,y) < t \\ 0 & si\ im(x,y) \geq t \end{cases} \qquad (2)$$

El llindar pot dependre de la imatge *im(x,y),* d'alguna propietat local del píxel *p(m,n)*, i fins i tot de la seva pròpia posició.

Quan parlem de dades de profunditat capturades amb el sensor Kinect, els elements que formen la matriu de la imatge tenen valors de 0 a 2047, vist en l'apartat 2.2.3, 2048 nivells que ajuden a la determinació de la distància de l'objecte.

En el cas de la segmentació de la mà, per tal de poder establir un llindar òptim i eficient, es requereix que la mà de l'usuari sigui l'objecte més proper al sensor i així, definir la distància de treball.

Tot i la facilitat de creació d'un algoritme de segmentació d'aquest tipus, les llibreries de processament d'imatges anteriorment descrites ja disposen d'algoritmes de segmentació on només cal indicar el llindar de tall.

2.4.4.- Detecció de contorns

La detecció de contorns és una eina bàsica en el processament d'imatges en el camp de la visió artificial i essencial en moltes aplicacions d'interpretació d'imatges. Molt utilitzada en tasques de segmentació i de reconeixement d'objectes.

Els contorns d'una imatge digital es poden definir com la frontera entre dues regions de nivells de gris significativament diferents. La seva causa principal és originada per la intersecció de varis objectes amb diferents nivells de reflectància, que al ser projectats a la càmera generen discontinuïtats d'intensitat en els píxels. Per a la seva detecció s'ha de posar èmfasi en els canvis bruscos dels nivells de gris dels píxels veïns i suprimir aquelles àrees amb valors de gris constants.

Un contorn local és un píxel que el seu nivell de gris difereix significativament del nivell de gris amb alguns dels seus píxels veïns, és a dir, hi ha diferència de contrast local. Això es deu bàsicament a dues situacions:

- ✓ El píxel forma part del contorn entre dues regions diferents d'una imatge
- ✓ El píxel forma part d'un arc molt fi sobre un fons de diferent nivell de gris.

Per detectar els contorns es comença per la detecció dels contorns locals que es realitza mesurant la taxa de canvi dels tons de gris del seu entorn. El fonament per a la seva detecció es basa en l'aplicació de l'operador derivada en un entorn de veïnatge. En aquest cas s'utilitza l'operador gradient (com a operador de derivada de primer ordre) i l'operador Laplacià (com a operador de segon ordre).

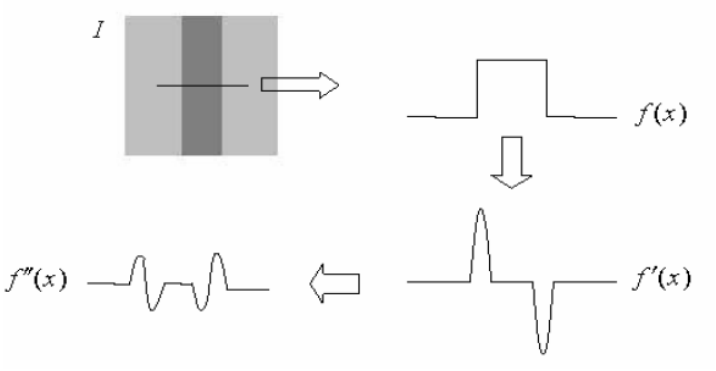


Figura 2.20: Salt brusc i corresponents derivades de primer i segon ordre

Es poden detectar les transicions brusques a partir dels extrems de la primera derivada o dels passos per zero de la segona.

Els mètodes basats en el gradient són més adequats quan la transició és brusca ja que són poc sensibles en canvis graduals de nivells de grisos, on és més adequat utilitzar operadors basats en derivades de segon ordre.

Existeixen nombroses tècniques basades en els operadors de primer i segon ordre, a continuació es fa una breu introducció a quatre de les tècniques més utilitzades [17]:

- ✓ **Operador basats en el gradient (operador de primer ordre)**
  L'aproximació discreta al gradient té dues components, el gradient horitzontal i el vertical, que es poden definir a partir de la diferència amb els píxels veïns. Aquests gradients es poden redefinir amb màscares que marquen les pautes de l'operació a realitzar. Aquestes poden representar-se en forma de matrius i permeten determinar els gradients, primer en direcció de les files (horitzontal) i després en la de les columnes (vertical).

  - o **Roberts**
  $$M_V = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \qquad M_H = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3}$$
  - o **Prewitt**
  $$M_V = 1/3 \cdot \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad M_H = 1/3 \cdot \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{4}$$
  - o **Sobel**
  $$M_V = 1/4 \cdot \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \qquad M_H = 1/4 \cdot \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{5}$$

- **Fre-Chen** $M_V = 1/(2+\sqrt{2})\begin{bmatrix} -1 & -\sqrt{2} & -1 \\ 0 & 0 & 0 \\ 1 & \sqrt{2} & 1 \end{bmatrix}$ $M_H = 1/(2+\sqrt{2})\begin{bmatrix} -1 & 0 & 1 \\ -\sqrt{2} & 0 & \sqrt{2} \\ -1 & 0 & 1 \end{bmatrix}$ (6)

✓ **Operador basats en el Laplacià (operador de segon ordre)**

De manera anàloga es pot definir l'operador laplacià aplicant successivament dues operacions gradient. Es pot calcular mitjançant els següents filtres o mascares:

$$M = 1/8 \cdot \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \qquad M = 1/4 \cdot \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{7}$$

La plantilla Laplaciana no s'aplica directament, ja que al ser una diferència de diferències és molt sensible al soroll.

- La **Laplaciana de la Gaussiana (LoG, Marr-Hildred),** filtre prèviament la imatge per reduir la sensibilitat al soroll amb un filtre Gaussià.
- **L'Algoritme de Canny** és àmpliament utilitzat, i es basa en el procés d'optimització de la imatge. Es suavitza la imatge amb un filtre gaussià, s'estimen les magnituds del gradient i les direccions dels contorns locals, es determinen els màxims locals i es realitza una supressió no maximal amb dos llindars.

2.5.- Treballs relacionats

Després del llançament del sensor Kinect l'any 2010, es van començar a desenvolupar diversos sistemes de reconeixement que l'utilitzaven com a dispositiu de captura de les imatges i dades de profunditat.

El reconeixement de la mà és un procés molt més complex si es compara amb el seguiment de les articulacions del cos degut a les petites dimensions d'aquesta, la qual cosa complica el procés de segmentació i anàlisi.

Zhou Ren a [19] presenta un robust sistema de Reconeixement de gestos del mà utilitzant les dades de profunditat de Kinect. Proposa una mesura de distàncies de dissimilitud de la mà anomenada Finger-Earth Mover's Distance (FEMD) basada en la Earth Mover's Distance (EMD), que és una mesura de distàncies entre dues distribucions de probabilitat. Amb la forma de la mà i la representació de la seva sèrie temporal (corba de la distància relativa entre cada vèrtex del contorn i el centre de la mà) FEMD mesura la distancia mínima de dissimilitud entre la mà i una plantilla de reconeixement,

$$c = \arg\min FEMD(H, Tc) \tag{8}$$

on H és l'entrada i Tc la plantilla. Aquesta mesura està orientada als dits, de manera que no és necessari tenir una representació perfecte de tota la mà. Zhou Ren a [27] demostra

l'eficiència del sistema en dues aplicacions a temps real, una aplicació d'aritmètica tipus calculadora i en el joc de pedra-paper-tisores.

Yi Li a [20] ens presenta un altre sistema de reconeixement de la mà a temps real, que permet identificar els dits i reconèixer el significat de nou gestos diferents. El procediment de identificació dels dits es basa en la mesura de les seves distàncies relatives calculades a partir de l'envolupant convexa i el contorn. El reconeixement dels gestos es realitza filtrant els vectors que representen els dits amb un conjunt de classificadors segons les seves característiques. A diferència del cas anterior, aquest sistema permet fer el reconeixement de dues mans simultàniament.

Un sistema de reconeixement complert de la llengua de signes ha de ser capaç de reconèixer els gestos de la mà, però és essencial que pugui reconèixer els signes dinàmics de les persones signants, ja que és tal i com es comuniquen a la vida real. Hi ha molts sistemes basats en el reconeixement de l'esquelet que permeten fer-ne la interpretació.

Recentment, Daniel Martínez Capilla a [21] descriu un Traductor de la llengua de signes basat en el seguiment de l'esquelet d'una persona. El sistema realitza el seguiment temporal de vuit punts de l'esquelet que són determinants per a la identificació del gest que posteriorment es poden classificar mitjançant algun dels dos algoritmes següents, el Nearest Neighbor DTW i el Nearest Group DTW. Aquesta aplicació és capaç de reconèixer 14 signes diferents amb una precisió del 95.238%.

Per altra banda, en el camp de reconeixement, ja sigui de la parla o de la llengua de signes, habitualment s'utilitzen models estocàstics com els Models Ocults de Markov (HMM – Hidden Markov Models) per realitzar el reconeixement dels signes, un exemple n'és el treball de Simon Lang a [28], un sistema que ofereix una precisió superior al 97% per a vuit o nou signes diferents.

Capítol 3: Metodologia

"La ciència més útil és aquella que el seu fruit és el més comunicable"
– Leonardo da Vinci

3.1.- Esquema general del sistema desenvolupat

El sistema de reconeixement d'aquest projecte es pot dividir en quatre grans blocs, la configuració de l'entorn de treball, la detecció de la mà, la identificació dels dits i el reconeixement de gestos, tal i com es pot veure en el següent diagrama:

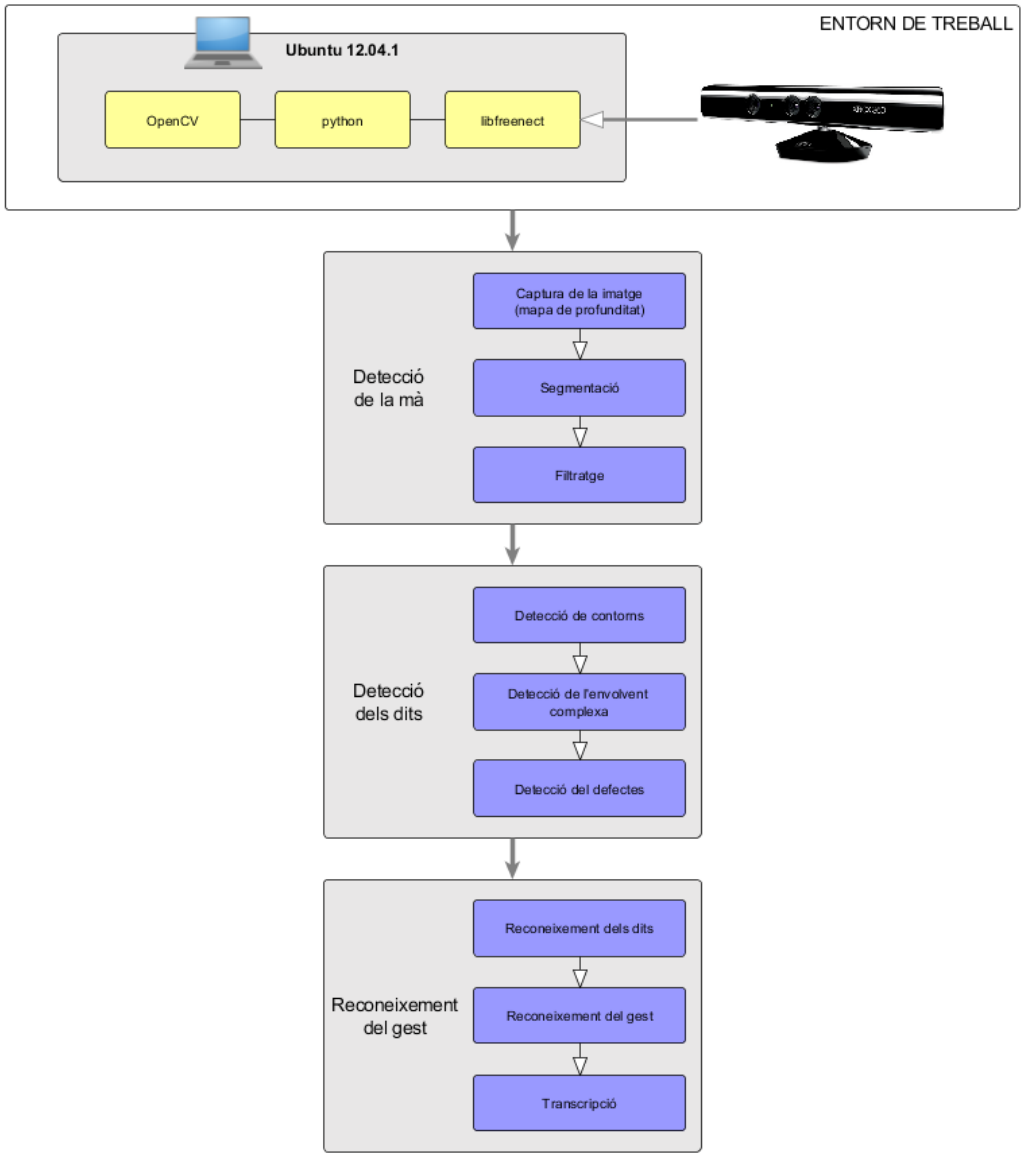


Figura 3.1: Diagrama de flux del projecte

L'entorn de treball tot de codi obert i lliure està configurat sota Ubuntu i programat amb el llenguatge de programació python. Els drivers libfreenect de OpenKinect s'utilitzaran per a la captura de les dades de profunditat del sensor Kinect. Les dades capturades es processaran amb la llibreria de visió artifical OpenCV.

El primer pas és segmentar les dades capturades per tal d'eliminar el fons i poder treballar únicament amb les mans de l'usuari.

Una vegada segmentada la imatge es processa per tal de poder determinar el contorn i el l'envolupant convexa de la mà que ens permetran detectar i identificar els dits.

Finalment el reconeixement de gestos es fa classificant segons el nombre de dits i els angles entre ells.

3.2.- Entorn de treball

En aquesta secció es descriu el hardware i software utilitzat per dur a terme la realització del projecte i el motiu pel qual han estat escollits.

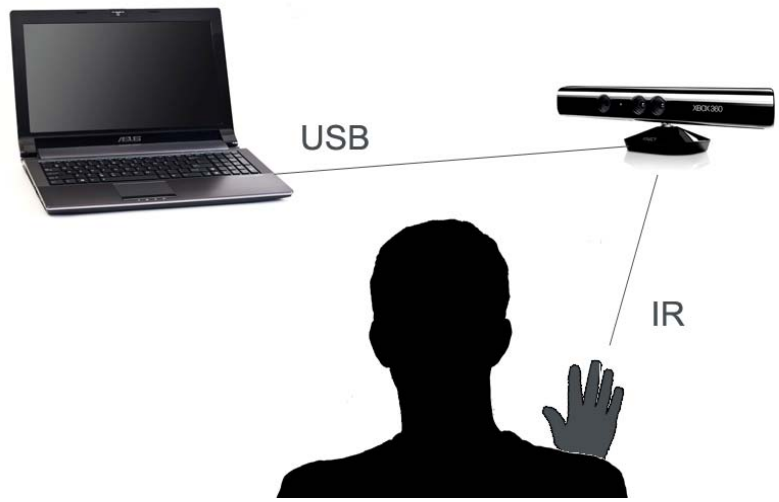


Figura 3.2: Entorn de treball

3.2.1.- Hardware

Per al desenvolupament d'aquest projecte s'ha utilitzat juntament amb el sensor Kinect un ordinador portàtil Asus N53SN Series amb les següents característiques:

- ✓ Processador Intel® Core™ i7 2630QM CPU (2.00-2.9GHz)
- ✓ Memòria RAM DDR3 de 8GB
- ✓ Targeta Gràfica NVIDIA® GeForce® GT 550M con 2GB DDR3 VRAM

3.2.2.- Software

El projecte ha estat desenvolupat en Python, ja que és un llenguatge de programació orientat a objectes potent i de codi obert.

Pel que fa al processament de les dades, s'han utilitzat la llibreria OpenCV, ja descrita en el capítol anterior i que proporciona una gran varietat d'algoritmes de processament d'imatges en el camp de la visió artificial.

El sistema operatiu sobre el que s'ha treballat és Ubuntu degut a que es pretenia crear un sistema obert i lliure de llicències, i com que les llibreries i controladors són multi-plataforma no hi havia cap inconvenient en aquest aspecte.

Les versions instal·lades de cada component són:

- ✓ Sistema Operatiu: Ubuntu 12.04 LTS
- ✓ Entorn de programació: Python 2.7.3
- ✓ Llibreria de visió artificial: Open CV 2.1
- ✓ Llibreria de visió artificial: Open CV 2.4
- ✓ Controladors per Kinect: Libfreenect de OpenCV

A l'annex II d'aquest projecte es detalla el procés d'instal·lació i configuració de l'entorn de treball. Perquè els programes, llibreries i controladors funcionin correctament és necessària la instal·lació de certes dependències que s'enumeren en el mateix annex.

3.3.- Detecció de la mà

El primer pas pel reconeixement dels gestos correspon al procés de detecció de la mà, que es pot dividir en tres etapes, la primera fa referència a la captura de les dades de profunditat mitjançant el sensor Kinect, la segona a la segmentació de la imatge per tal d'eliminar el fons i poder treballar unicament amb l'objecte d'interès, la mà, i la darrera referent a l'etapa de prefiltratge prèvia al processat i anàlisi del contorn.

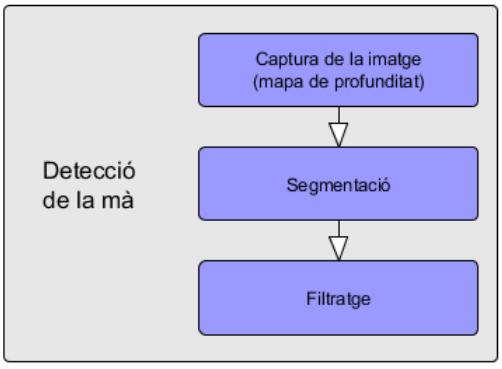


Figura 3.3: Etapes de la detecció de la mà

3.3.1.- Captura de dades de profunditat

Com ja s'ha especificat en apartats anteriors, la captura de les dades s'ha realitzat mitjançant els controladors *libfreenect* proporcionats per la comunitat OpenKinect.

```
from freenect import sync_get_depth as get_depth
(depth)= get_depth()
```

Les dades de profunditat que es capturen amb el sensor de profunditat 3D, poden ser de resolucions 80x60, 320x240 i 640x480 píxels, per les petites dimensions de les mans en relació amb l'escena, és imprescindible escollir la màxima resolució, tot i això, les dimensions de les mans són reduïdes, aproximadament d'uns 120x120 píxels.

El primer pas doncs, és la captura de les dades de profunditat 640x480 a 30fps mitjançant *libfreenect*. Cada fotograma és representa com una matriu de dimensions 640x480 píxels a 11 bits de precisió, el que implica tenir valors enters en l'interval de [0,2047].

```
In [4]: depth
Out[4]:
array([[2047, 2047,  967, ..., 2047, 2047, 2047],
       [2047,  966,  967, ..., 2047, 2047, 2047],
       [2047,  967,  967, ..., 2047, 2047, 2047],
       ...,
       [2047, 2047, 2047, ..., 2047, 2047, 2047],
       [2047, 2047, 2047, ..., 2047, 2047, 2047],
       [2047, 2047, 2047, ..., 2047, 2047, 2047]], dtype=uint16)
```

Figura 3.4: Matriu de les dades de profunditat

Figura 3.5: Representació de la imatge de les dades de profunditat

Les dades de profunditat en el seu estat original no ens proporcionen cap imatge nítida, això canvia si les convertim a un format de 8 bits on tindrem valors en l'interval [0,255] a escala de grisos .

Figura 3.6: Imatge de la profunditat convertida a format 8 bits escala de grisos

Una vegada capturades les dades, ja es pot procedir a la segmentació de la imatge per eliminar el fons i poder treballar únicament amb la mà.

3.3.2.- Segmentació

La segmentació és el procés bàsic per tal de poder seleccionar l'objecte a analitzar, que en aquest cas correspon a les mans de l'usuari.

El mètode que s'utilitza per a la segmentació de les dades és el de l'aplicació d'un llindar que s'ajusta a la distància a la que hi ha situada la mà segons les dades proporcionades pel mapa de profunditat capturat.

Perquè la segmentació sigui eficient cal que l'usuari s'asseguri que la mà és l'objecte més proper al sensor, ja que el llindar es relatiu al punt més proper, com es veu a la figura.

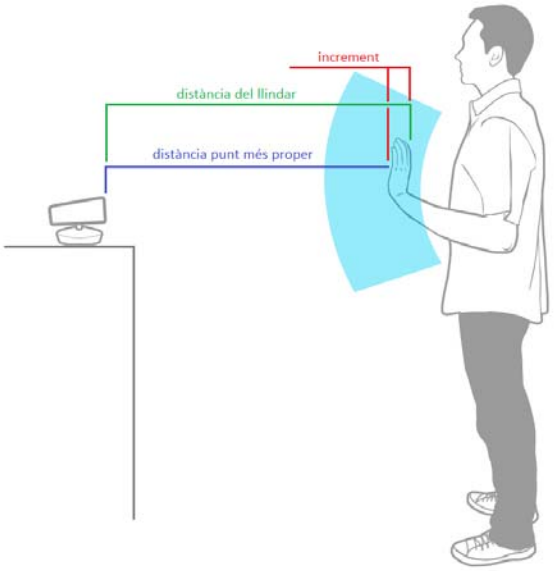


Figura 3.7: Càlcul del llindar de segmentació

Per tant, el llindar es calcularà incrementant en X nivells, dels 2047 que ens proporciona la captura, el valor de la distància fins a l'objecte més proper.

```
import opencv
cv.Threshold(src, dst, threshold, maxValue, thresholdType)
```

La distància òptima de treball, tot i que la segmentació estigui basada en funció de la posició de la mà, és d'entre 0,5 i 0,8 metres. En el gràfic de distàncies en funció de les dades de profunditat calculat a l'apartat 2.2.3, es pot veure que estem aproximadament entre els nivells de profunditat de 400 a 650.

Per tant, analitzant la corba (figura 3.8) es pot extreure una possible aproximació de l'increment que s'ha d'aplicar a la mínima distància calculada. Dins d'aquest marge (400-650) el creixement de la distància és lent i gairebé recte, ens movem entorn a 30 centímetres en 250 nivells de profunditat diferents, el que podríem aproximar a 0.12 centímetres per nivell. Si es considera que tots els dits de la mà no estan a la mateixa distància que el punt més proper, hem de prendre com a llindar un punt més llunyà, i tenint en compte que la mà no sol fer més de 5 centímetres de profunditat en la majoria de les seves postures, es pot aproximar, segons la hipòtesi anterior que l'increment pot ser de 40 punts, el que serien uns 5 centímetres.

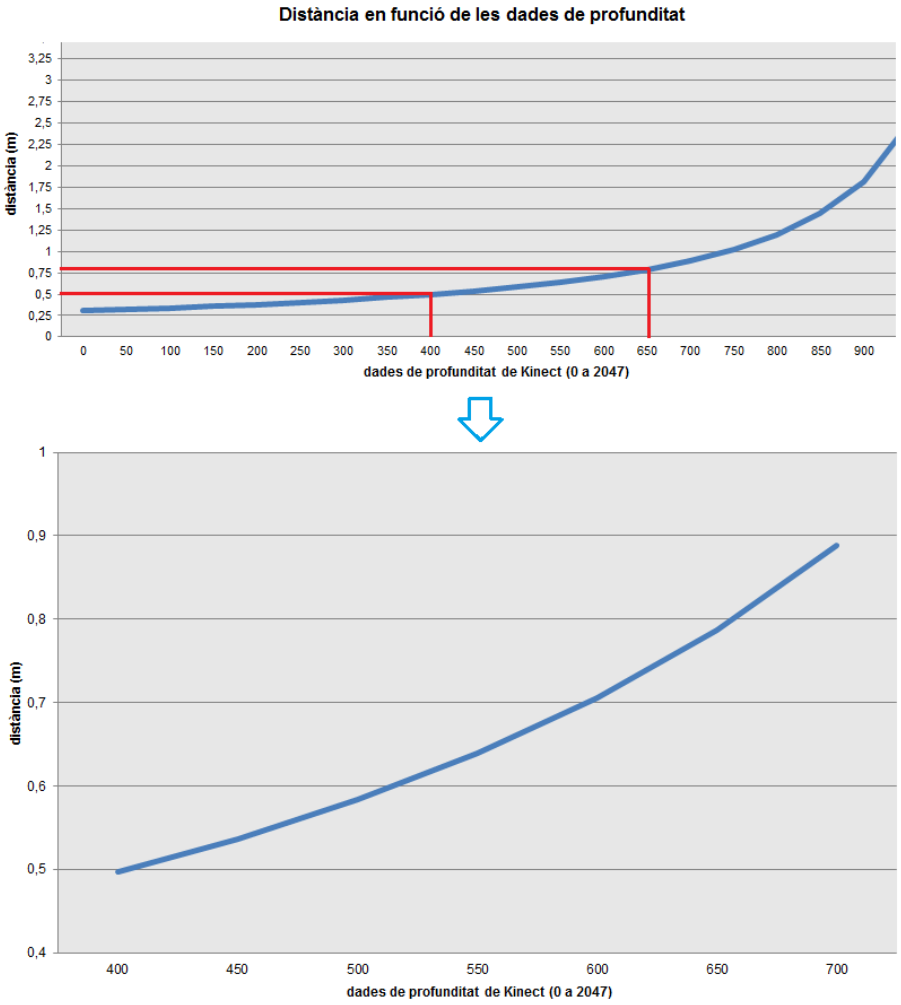


Figura 3.8: Distàncies de treball en funció de la profunditat

Aplicant doncs aquest llindar en funció del punt més proper de la mà podem segmentar les dades i aïllar la mà de la resta d'objectes de l'escena. (figura 3.9).

Figura 3.9: Representació de la imatge segmentada i binaritzada

Com ja s'ha documentat en la teoria, apartat 2.4.3, la segmentació a dos nivells implica la binarització de la imatge.

3.3.3.- Filtratge

Per tal de suavitzar els contorns de la imatge extreta i poder-ne eliminar el soroll es procedeix a una etapa de filtratge mitjançant un filtre *Median*, una solució molt utilitzada com a pas previ per millorar els resultats del posterior processat, per exemple, quan es volen detectar contorns d'imatges.

El filtre Median és un dels filtres no lineals més utilitzats. En aquest filtre s'ordenen els valors de la plantilla de N veïns per cada píxel i la sortida s'obté del valor central de l'ordenació. Normalment la plantilla (kernel) tenen un número imparell de valors (3x3, 5x5...). Es pot expressar de la manera següent:

$$XS(m,n) = median(X_N(m,n)) \qquad (9)$$

$$X_N(m,n): N \ \ píxeles \ \ vecinos \ \ de \ \ X(m,n)$$

$$median(N \ pixeles) = I \qquad \forall \begin{cases} num(I \leq I_i) & es \quad (N-1)/2, \\ num(I \geq I_i) & es \quad (N-1)/2 \end{cases}$$

En el cas que pertoca, apliquem a la imatge binaritzada de la mà un filtre *Median* amb una plantilla de 9x9, d'aquesta manera aconseguim suavitzar el contorn de la imatge tal i com es pot comprovar a la figura següent.

Figura 3.10: Imatge filtrada amb un filtre *Median*

```
import opencv
cv.medianBlur(src, dst, ksize)
```

3.3.4.- Normalització

Pel bon funcionament de l'algoritme de detecció és imprescindible que l'usuari s'asseguri que la mà és l'objecte més proper al sensor i a aproximadament entre 0.5 i 0.8 metres de distància.

L'algoritme de reconeixement està basat en l'anàlisi del contorn de la mà, i per tant, perquè el procés es dugui a terme correctament el sensor Kinect ha de ser capaç de veure tot el contorn, per tant, la mà ha d'estar encarada al sensor.

El sistema s'ha dissenyat de manera que l'orientació de la mà no sigui significativa, de manera que la detecció dels dits es realitza tant si la mà està en posició vertical com horitzontal.

3.4.- Deteccio dels dits

Una vegada es disposa de la imatge de la mà, segmentada i filtrada, ja es pot procedir a la detecció dels dits, que són elements imprescindibles per tal de poder fer el reconeixement d'un gest.

En l'etapa de detecció dels dits podem destacar tres processos, l'extracció de contorns, la detecció de l'envolvent convexa i finalment la detecció dels defectes, que en permetran extreure tota la informació necessària per a poder fer el reconeixement.

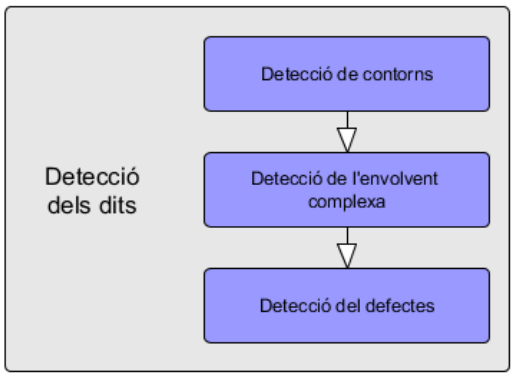


Figura 3.11: : Etapes de la detecció dels dits

3.4.1.- Contorns

La detecció de contorns d'una imatge és un eina molt útil quan es vol fer l'anàlisi de les formes i la detecció d'objectes pel posterior reconeixement.

Extreure el contorn d'un objecte redueix la complexitat computacional de les etapes de processament posteriors degut a que la resta de càlculs es realitzaran únicament amb els punts del contorn i no de tota la imatge.

Tot i haver vist en el capítol 2 els principals algoritmes de detecció de contorn, la llibreria OpenCV disposa d'una funció de detecció de contorns basada en l'algoritme *Suzuki85* de *Satoshi Suzuki and Keiichi Abe* [23] que és capaç de determinar els contorns externs i interns d'una imatge binària.

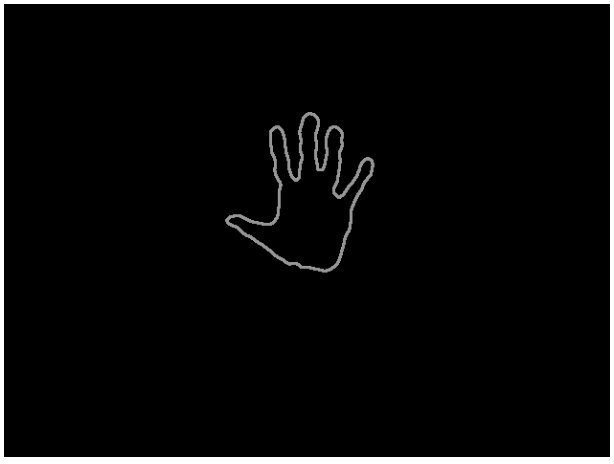


Figura 3.12: Contorn de la mà.

```
import opencv
cv.findContours(image, contours, hierarchy, mode, method)
cv.DrawContours(image, contours, external_color, hole_color,
               max_level,thickness=1, lineType=8, offset=(0, 0))
```

Una vegada extret el contorn de la mà i per a poder procedir a l'anàlisi de la forma és important simplificar-lo, és a dir, esquematitzar-lo perquè mantingui tota la informació postural, però en un nombre reduït de punts.

Per a realitzar aquest procés, la llibreria OpenCV disposa d'una funció que permet aproximar un contorn a una forma poligonal (ApproxPoly), basada en l'algoritme de Douglas-Peucker [24] que té com a objectiu trobar una corba similar a la corba donada però amb un nombre menor de punts . El resultat d'aquesta aproximació es pot visualitzar a la següent figura.

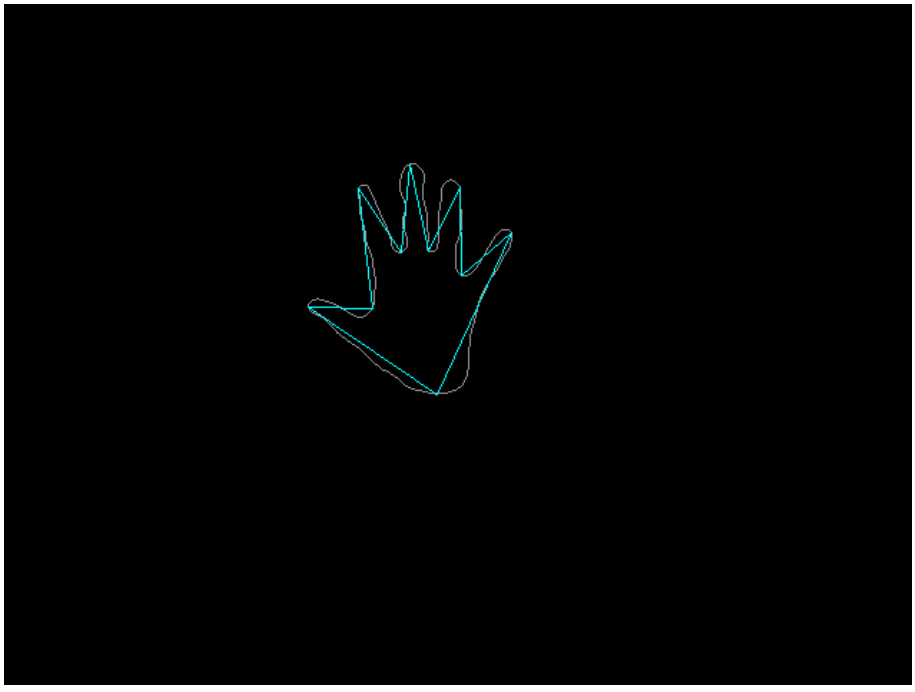


Figura 3.13: Aproximació poligonal de la mà

```
import opencv
ApproxCs = cv. ApproxPoly(contours, storage, method,
                                parameter=0, parameter2=0)
```

El contorn poligonal calculat simplificarà l'anàlisi posterior del contorn. El nombre de punts convexos i còncaus d'aquest contorn és més reduït, el que implica una facilitat a l'hora de determinar el nombre de dits de la mà.

3.4.2.- Detecció de l'envolupant convexa

L'envolupant convexa d'un conjunt de punts X es defineix com la intersecció de tots els conjunts convexos que contenen X. És a dir, és un polígon convex que conté tots els punts del conjunt i que els seus vèrtexs són punts del mateix conjunt.

Donats K punts X_1, X_2, ... X_n , la seva envolupant convexa C ve donada per l'expressió:

$$C(x) = \left\{ \sum_{i=1}^{n} \alpha_i x_i \middle| x_i \in X, \alpha_i \in \Re, \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i = 1 \right\} \qquad (10)$$

Hi ha varis algoritmes de càlcul de l'envolupant convexa, entre ells, destacar el de Graham [25] que és molt utilitzat i el de Sklansky [26] del que està basada la funció que ens proporciona la llibreria OpenCV. A la imatge següent podem veure el resultat d'aplicar la funció *ConvexHull.*

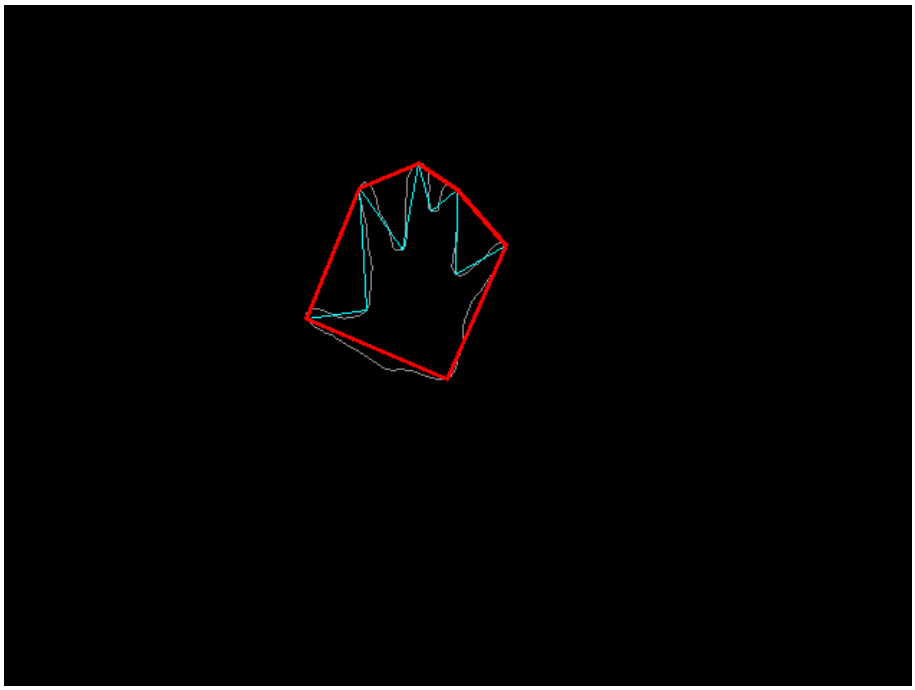


Figura 3.14: Envolupant convexa de la mà

```
import opencv
cv.ConvexHull2(ApproxCs,cv.CreateMemStorage(),return_points=True)
```

3.4.3.- Detecció de defectes de convexitat

Mitjançant el càlcul de l'envolupant convexa i el contorn de l'objecte es poden determinar el que podríem anomenar com "defectes de convexitat" (Convexity Defects), que són les ares buides entre els vèrtexs convexes i còncaus que formen el contorn de l'objecte.

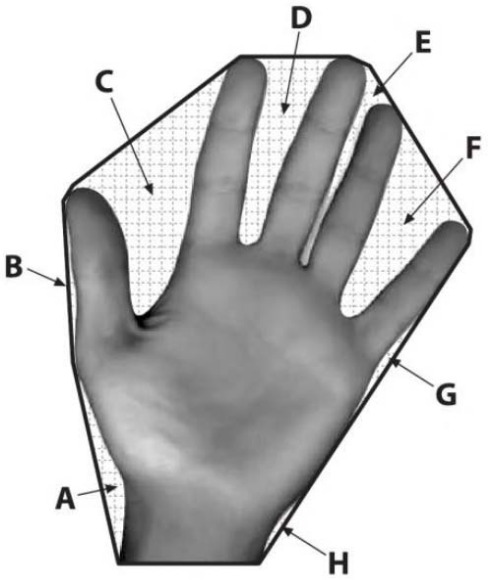


Figura 3.15: Defectes de convexitat i envolupant convexa

Com es pot interpretar en la figura, aquests defectes ens permeten interpretar la forma de la mà, el que és útil per determinar-ne la postura.

Quan s'aplica la funció *ConvexityDefects* que ofereix OpenCV es retorna un vector de quatre elements que està format pels punts que tanquen cada un dels defectes i la distància entre el punt més llunyà i l'envolupant convexa.

```
Vector :(punt_inici, punt_final, punt_llunyà, distància)
```

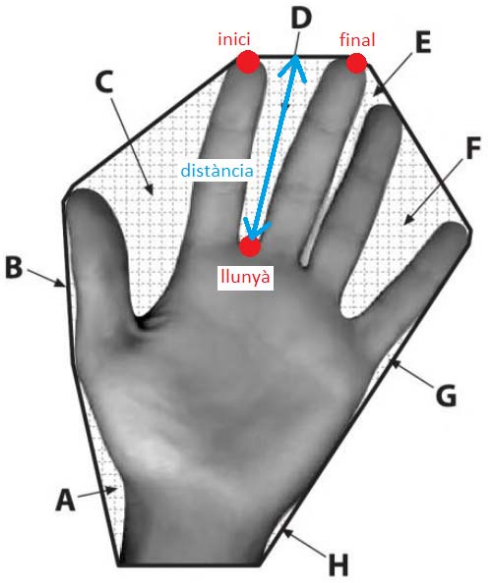


Figura 3.16: Punts que determinen un defecte convex

Aplicant la funció obtenim la representació següent (figura 3.17), on es marquen tots els punts convexos (punts inici i fi) i còncaus (punts llunyans) que determinen els defectes.

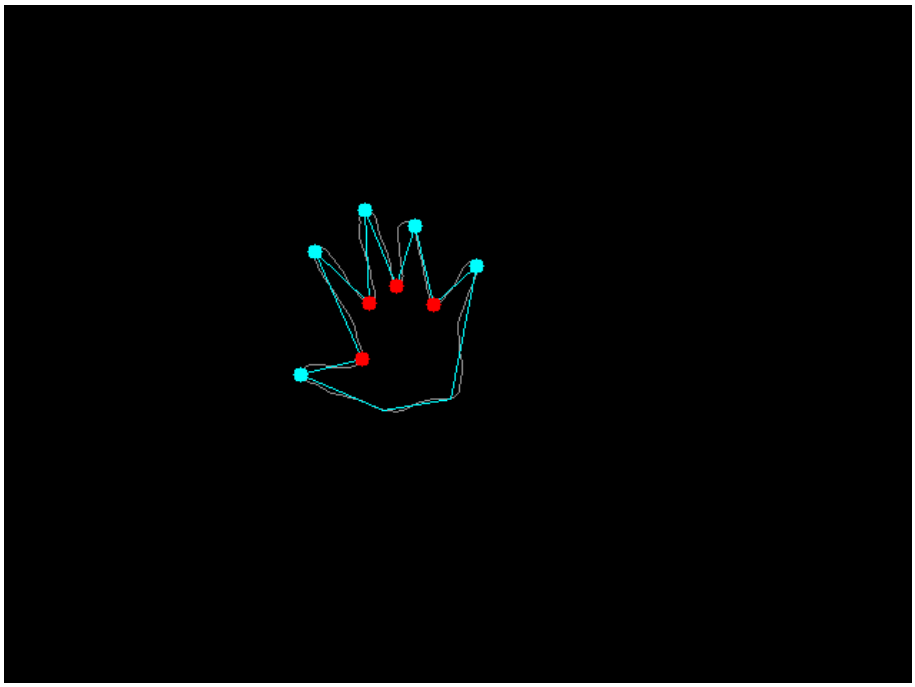


Figura 3.17: Punts convexos i còncaus de la mà

```
import opencv
cv.ConvexityDefects(contour, convexhull, storage)
```

3.5.- Reconeixement de gestos estàtics

El procediment de reconeixement de gestos té en compte bàsicament dos factors, el nombre de dits que composen el gest i l'angle que hi ha entre ells.

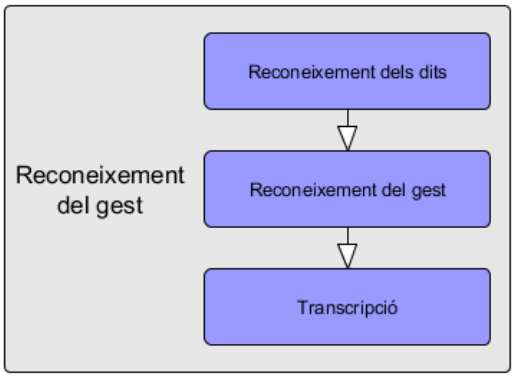


Figura 3.18: : Etapes del reconeixement

3.5.1.- Reconeixement del nombre de dits

Determinar el nombre de dits de la mà és un procés bàsic per realitzar el reconeixement de la postura de la mà que es proposa en aquest projecte.

Mitjançant els defectes convexos determinats en l'etapa anterior podem determinar el nombre de dits en funció del nombre de defectes i l'angle format pels vectors que van des

del centre de la mà (centroide) fins als punts inici i final que determinen el defecte. El procediment seria el següent (exemple de 5 dits):

1) Cal calcular la posició del centre de la mà. OpenCV permet determinar els moments d'una forma poligonal, els quals ens permetran determinar el seu centroide (centre de massa) de la manera següent:

```
m = cv.Moments(contour)
m10 = int(cv.GetSpatialMoment(m,1,0))
m00 = int(cv.GetSpatialMoment(m,0,0))
m01 = int(cv.GetSpatialMoment(m,0,1))
centroid = (int(m10/m00), int(m01/m00)
```

El que ens dona el resultat següent:

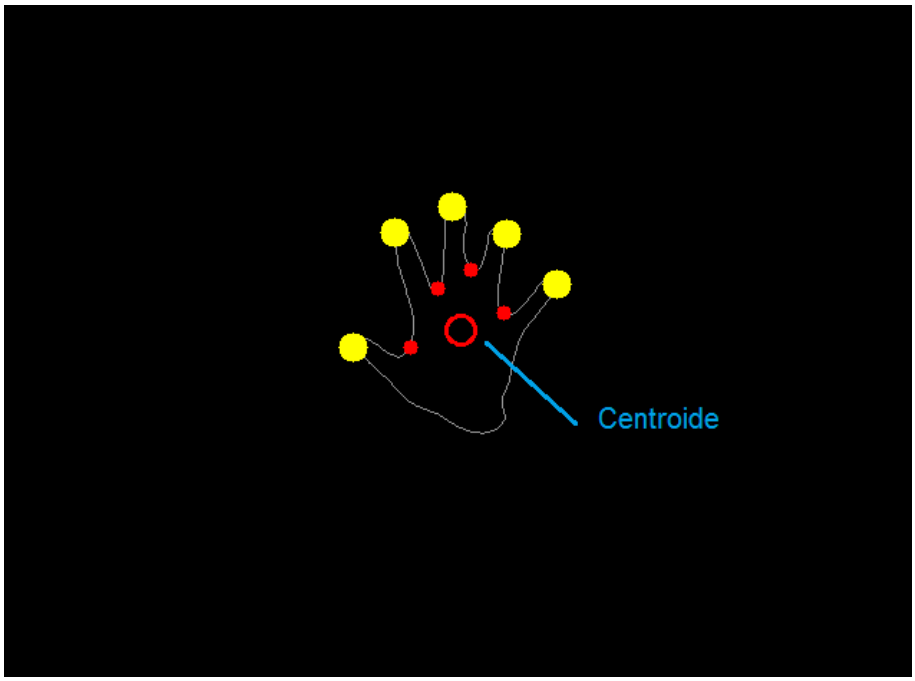


Figura 3.19: Centroide de la mà

2) Per altra banda es disposa de la llista de defectes amb una longitud de 4, quatre vectors que contenen els punts que el determinen. que té la forma següent:

```
In [32]: defects
Out[32]:
[((226, 148), (188, 246), (240, 236), 44.86750793457031),
 ((338, 179), (297, 142), (300, 206), 45.503265380859375),
 ((297, 142), (261, 128), (279, 177), 39.144195556640625),
 ((261, 128), (226, 148), (259, 190), 52.83879852294922)]
```

2) Si es considera que cada defecte es pot definir per dos vectors (punt inici – centroide i centroide – punt final) podrem calcular l'angle entre ells de la manera següent:

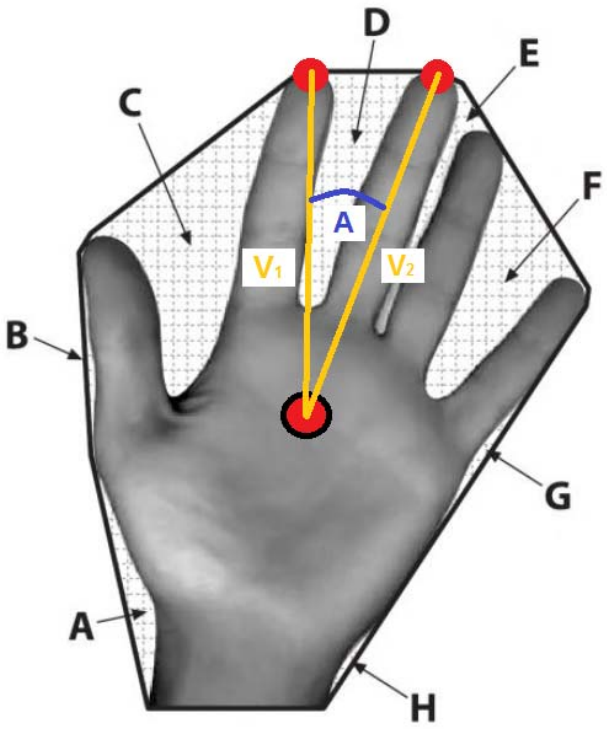


Figura 3.20: Vectors i angle entre ells

Per tant, si definim els dos vectors com $\vec{V}_1(x_1, y_1)$ i $\vec{V}_2(x_2, y_2)$ l'angle entre ells és:

$$A = \arccos \frac{\vec{V}_1 \cdot \vec{V}_2}{\|\vec{V}_1\| \cdot \|\vec{V}_2\|} \qquad (11)$$

Aconseguint un vector format pels angles interns de cada un dels defectes:

```
In [36]: Angles
Out[36]: [91.84604425130362, 57.288979314481935, 47.38676496915788, 40.00483685336366]
```

3) Una vegada determinats els angles, s'interpreta que si aquests són inferiors a 90 graus ens indiquen que és un defecte entre dits, per tant, quatre defectes dins d'aquests paràmetres ens indiquen que hi ha 5 dits a la mà, tres defectes quatre dits, i així successivament excepte quan existeix únicament un defecte que pot coincidir en el cas d'un o dos dits.

   Per solucionar aquesta confusió la restricció de l'angle inferior a 90 graus ens és suficient, degut a que l'angle del defecte que crea el dit índex estirat és superior, de manera que podem identificar-los separadament.

Els resultats obtinguts d'aquest procediment de reconeixement són els que es mostren a la pàgina següent.

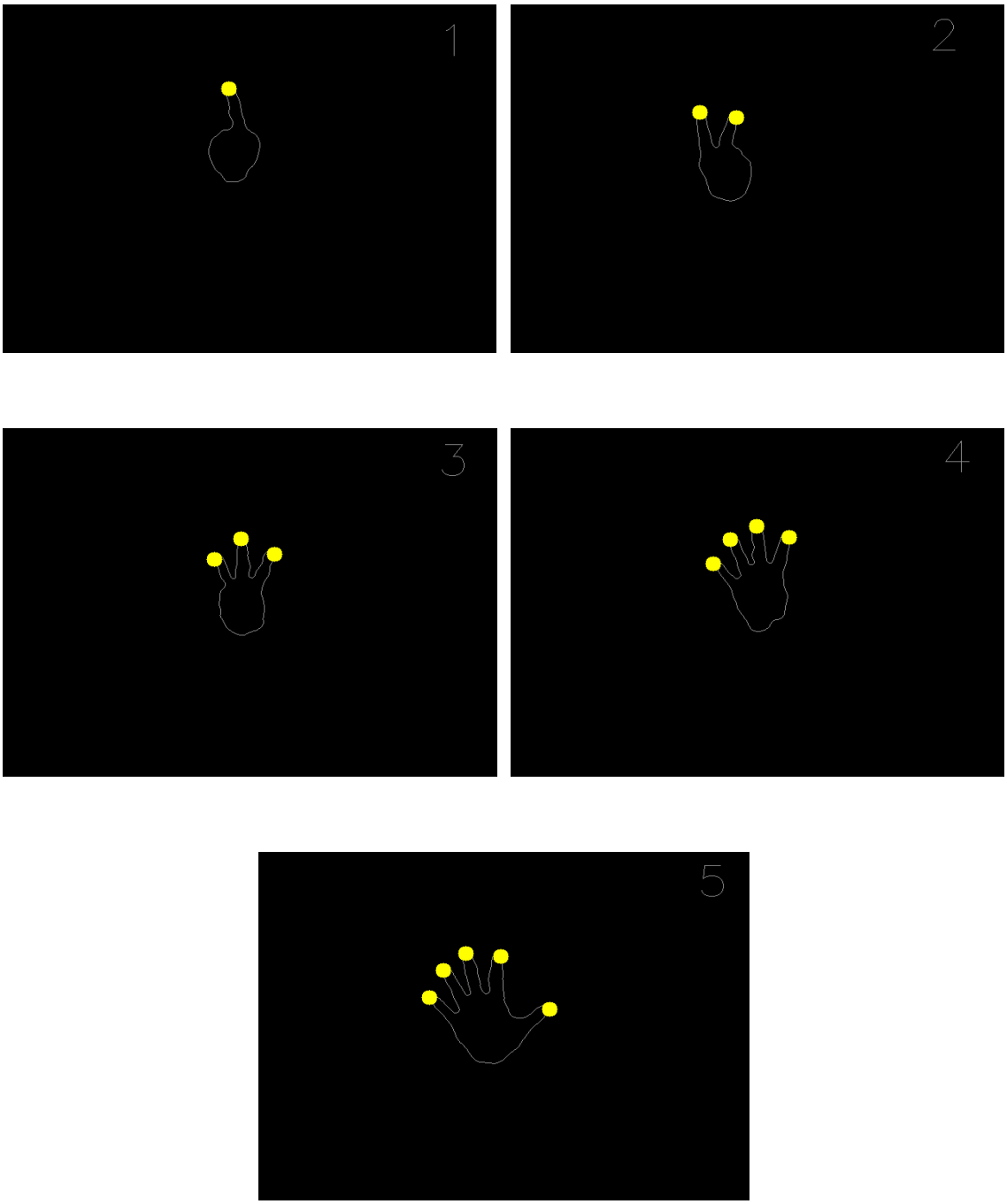


Figura 3.21: Resultat del reconeixement dels dits de la mà

Una vegada reconeguts el nombre de dits que hi ha a la mà, ja es pot procedir a la classificació dels gestos.

3.5.2.- Reconeixement de gestos

Una vegada determinat el nombre de dits estesos de la mà ja es pot procedir al reconeixement del gest.

Els gestos són reconeguts mitjançant un classificador de dos nivells; primer filtre la postura segons el nombre de defectes convexos que la formen i després la reconeix tenint en compte els angles entre els dits.

En el cas que siguin postures on no hi ha cap dit estes o la mà està tancada, el reconeixement es realitza tenint en compte l'àrea del contorn.

El sistema és capaç de reconèixer els números del 0 al 5, els símbols populars com l'OK i Victòria, i les lletres A, B i W de l'alfabet dactilològic.

El classificador desenvolupat pels gestos que és capaç de reconèixer tindria l'estructura següent:

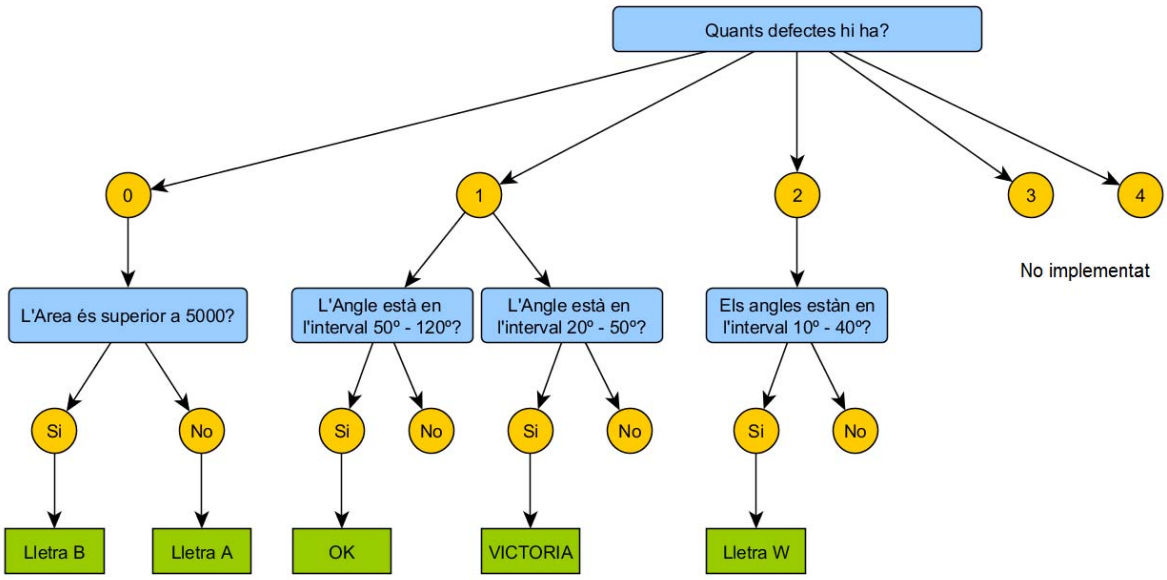


Figura 3.22: Estructura del classificador de gestos

A la pàgina següent es poden veure els resultats d'aplicar el classificador anterior.

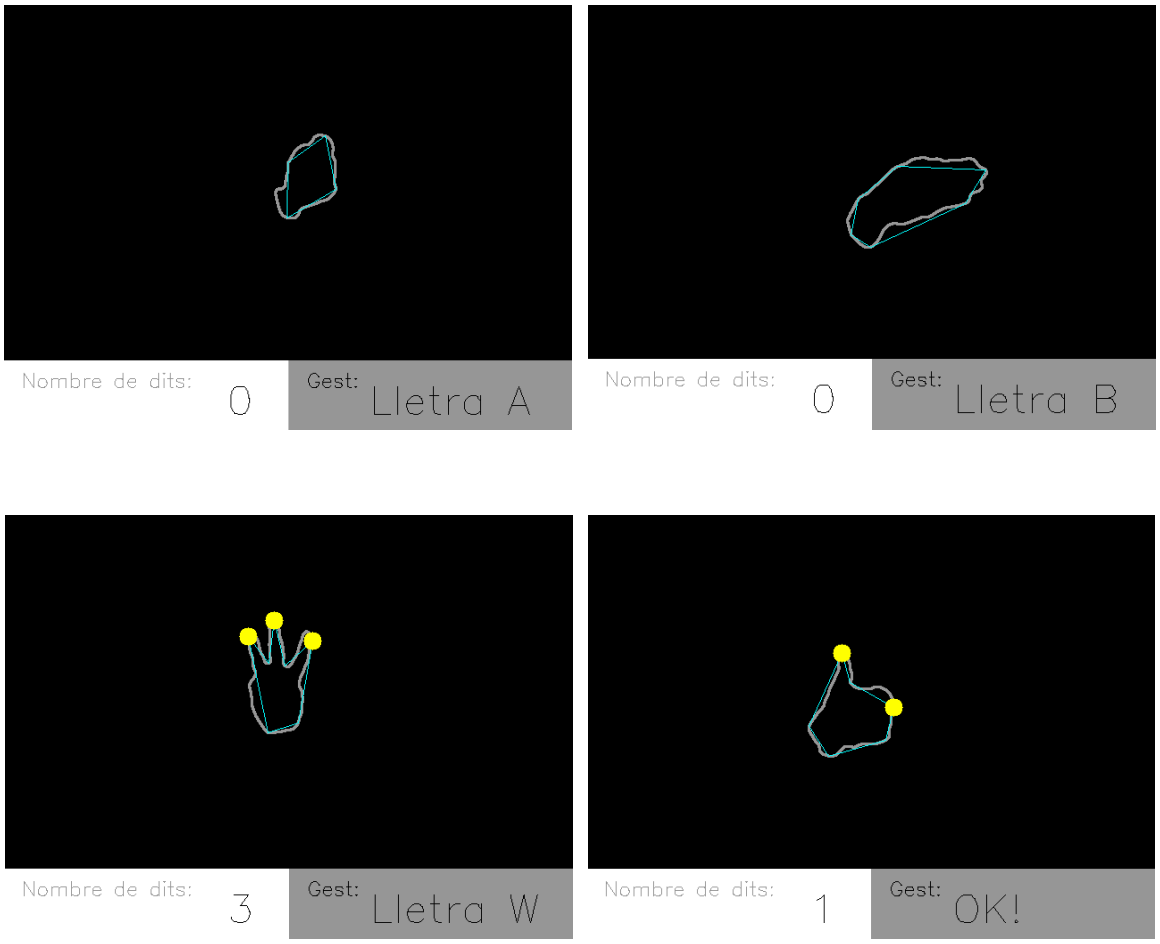


Figura 3.23: Resultats del reconeixement del gest de la mà

Capítol 4: Conclusions

"La ciència més útil és aquella que el seu fruit és el més comunicable"
– Leonardo da Vinci

4.1.- Conclusions

En aquest projecte s'ha presentat un sistema de reconeixement de gestos de la mà a temps real mitjançant l'ús del sensor Kinect capaç de determinar el nombre de dits, dos gestos populars ("OK" i "Victoria") i tres lletres de l'alfabet dactilològic.

L'estudi i recerca de treballs relacionats consciencien de la dificultat de realitzar un reconeixement acurat de la postura de la mà, degut a la seves petites dimensions i la importància que té fer una bona segmentació de la imatge. És per aquest motiu que s'utilitza una segmentació basada en les dades de profunditat i no per color de la pell. D'aquesta manera s'aconsegueix un sistema robust capaç de treballar amb qualsevol condició de llum, qualsevol color de pell i qualsevol fons d'escena.

El marge òptim de treball del sistema és de 0.5 a 0.8 metres, tot i això, el llindar de segmentació és dinàmic, es col·loca en funció del punt més proper al sensor, aconseguint així més flexibilitat a l'hora d'interactuar amb l'aplicació.

Els càlculs de contorns, de l'envolupant i del centroide convexa de la mà són fonamentals per tal de poder determinar el nombre de dits mitjançant el càlcul d'angles interiors.

El reconeixement de gestos es realitza amb un classificador que filtre per nombre de dits de la postura, angles entre ells i àrea del contorn.

Al considerar-se un sistema d'experimentació en el camp del reconeixement de gestos de la mà i que és totalment obert i ampliable no s'han realitzat mesures d'eficiència.

4.2.- Treball futur

El sistema de reconeixement que s'ha presentat és totalment obert, de manera que en un futur s'ha de treballar per poder aconseguir millorar-lo per tal de que sigui molt més robust i eficient.

Aconseguir un sistema de reconeixement molt més acurat permetria distingir entre molts més signes i gestos, el que portaria a poder reconèixer totes les lletres i signes de puntuació que formen part de l'alfabet dactilològic.

Per altra banda, la incorporació de Models Ocults de Markov (HMM – Hidden Markov Models) [30] permetria la implementació d'una nova etapa de reconeixement i seguiment de l'esquelet per interpretar els signes dinàmics de la llengua de signes que formen les paraules i frases, d'aquesta manera es completaria el sistema.

Aquest projecte és doncs, la porta d'entrada al camp del reconeixement de gestos i la base d'un futur sistema de reconeixement de la llengua de signes capaç de transcriure tant els signes dinàmics com l'alfabet dactilològic.

Bibliografia

[1] *Mira què dic, Diccionari Múltimedia de Signes de Catalunya* [en línia]. Barcelona: Generalitat de Catalunya, Departament d'educació. <www.edu365.cat/mqd/>.

[2] QUER VILLANUEVA, Josep. *"Les llengües de signes com a llengües naturals"*. Barcelona: ICREA i Universitat de Barcelona, 2004.

[3] *SIGNEM, Guia bàsica de comunicació en la llengua de signes* [en línia]. Barcelona: Universitat Autònoma de Barcelona. <http://philipjfry.cephis.uab.cat/signem/index.php>.

[4] *Kinect for Windows Quickstart Series* [en línia]. Microsoft Channel 9. <http://channel9.msdn.com/Series/KinectQuickstart>.

[5] *Kinect for Windows SDK* [en línia]. Microsoft MSDN Library. <http://msdn.microsoft.com/en-us/library/hh855347.aspx>.

[6] *OpenKinect project* [en línia]. OpenKinect Community. <http://openkinect.org/wiki/Main_Page>.

[7] *CL NUI Platform - Kinect Preview* [en línia]. Code Laboratories. <http://codelaboratories.com/nui>.

[8] *OpenNI Documentation* [en línia]. OpenNI Organization. <http://openni.org/>.

[9] *ROS Kinect* [en línia]. ROS Community. <http://www.ros.org/wiki/kinect>.

[10] *Kinect for Windows, Developer Downloads* [en línia]. Microsoft. <http://www.microsoft.com/en-us/kinectforwindows/develop/developer-downloads.aspx>.

[11] *Our Full 3D Sensing Solution* [en línia]. PrimeSense Natural Interaction. <http://www.primesense.com/en/technology/115-the-primesense-3d-sensing-solution>.

[12] *OpenCV Wiki* [en línia]. OpenCV. <http://opencv.willowgarage.com/wiki/>.

[13] *PrimeSense Natural Interaction* [en línia]. PrimeSense <http://www.primesense.com/nite>.

[14] "*NiTE™ Middleware − the Natural Interaction Engine*"· [data sheet]. PrimeSense, 2012.

[15] *Java Advancing Image API v1.1*. [en línia]. Oracle. <http://www.oracle.com/technetwork/java/readme-1-1-137504.html>.

[16] ROS, Alfonso; MENDON, Ismael. "*Captura de movimiento de personas con múltiples sensores Kinect*" [Projecte de grau]. Universidad Simón Bolívar, 2012

[17] PLATERO, Carlos. *"Apuntes de Vision Artificial"* [Apunts]. Universidad Politécnica de Madrid, Departamento de Electronica, Automatica e Informatica Industrial, 2009.

[18] REIG BOLAÑO, *"Modulo 4. Analisis y Reconocimiento"* [Apunts]. Universitat de Vic, Assignatura Processat de Dades Multidimensionals, Màster TAI, 2010.

[19] REN, Zhou; YUAN, Junsong; ZHANG, Zhengyou. *"Robust hand gesture recognition based on Finger-earth mover's distance with a commodity depth càmera". Proceedings of ACM MM*, 2011

[20] LI, Yi. *"Hand gesture recognition using Kinect"*. A Software Engineering and Service Science (ICSESS), 2012 IEEE 3rd International Conference pàgines 196-199, 2012.

[21] MARTINEZ CAPILLA, David. *"Sign Language Translator"* [MSc Thesis]. University of Tennessee, 2012.

[22] LAM PHUNG, Son; BOUZERDOUM, Abdesselam; CHAI, Douglas. *"Skin Segmentation Using Color Pixel Classification: Analysis and Comparison"*. University of Wollongong, Faculty of Informatics. A Pattern Analysis and Machine Intelligence, IEEE Transactions pàgines 148-154, 2005.

[23] SUZUKI, Satoshi; ABE, Keiichi. *"Topological structural analysis of digitized binary images by border following"*. A Computer vision, graphics, and image processing, pàgines 32-46 (1985)

[24] HERSHBERGER, John; SNOEYINK, Jack. "*Speeding Up the Douglas-Peucker Line-Simplication Algorithm"*

[25] GRAHAM, L.R. *"An efficient algorithm for determining the convex hull of a finite planar set"*. A Information Processing Letters 1, pàgines 132-133, 1972.

[26] ALOUPIS, Greg; KALUZNY, Bohdan. "*The Three-Coins Algorithm for Convex Hulls of Polygons"* [en línia]. McGill University, School of Computer Science <http://cgm.cs.mcgill.ca/~beezer/cs507/3coins.html>.

[27] REN Zhou; MENG, Jingjing; YUAN, Junsong; ZHANG, Zhengyou. "Robust hand gesture recognition with kinect sensor". *Proceedings of ACM MM*, 2011.

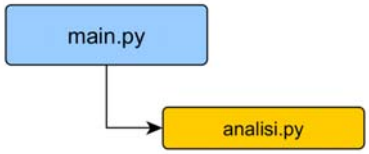
[28] LANG, Simon. *"Sign Language Recognition with Kinect"* [Projecte Final]. Freie Universität Berlin, 2011.

[29] CARRANZA ATHÓ, Fredy, "Esqueletización". Universitat Nacional de Trujillo, 2006.

[30] RABINER, Lawrence R. "*A tutorial on Hidden Markov Models ans selected Applications in speech recognition". Proceedings of the IEEE 77(2):257-286,* Febrer 1989.

Annex I: Codi font

El sistema està compost per dos arxius, el programa principal i la funció d'anàlisi de la forma de la mà, amb la jerarquia següent:



✓ *main.py*

```
#!/usr/bin/env python
from freenect import sync_get_depth as get_depth, sync_get_video as get_video
import numpy as np #Importa la llibreria NumPy
import cv,cv2 #Importa les llibreries OpenCV

import Analisi #Importa la funcio d'Analisi de la ma

global depth, rgb

while True:

        # Capturar dades de profunditat i RGB
        (depth,_), (rgb,_)= get_depth(), get_video()
        cv.ShowImage("Dades de profunditat Originals", cv.fromarray(depth))

        depth32 = depth.astype(np.float32)

        # Calcul del punt mes proper al sensor
        minim = np.min(depth32)

        # Establir el llindar de segmentacio en funcio del punt mes proper + 50 unitats
        llindar = minim + 50

        # Segmentar segons el llindar i conversio a imatge binaria.
        _,depthThresh = cv2.threshold(depth32, llindar, 255, cv2.THRESH_BINARY_INV)
        cv.ShowImage("Imatge segmentada", cv.fromarray(depthThresh))

        #Filtrem amb un filtre Median amb finestra 9 x 9
        depthThresh=depthThresh.astype(np.uint8)
        cv2.medianBlur(depthThresh, 9, depthThresh)
        cv.ShowImage("Imatge filtrada", cv.fromarray(depthThresh))

        # Crida a la funcio d'Analisis de la ma
        Analisi.TR(depthThresh)

        cv.WaitKey(5)
```

✓ *analisi.py*

```
#!/usr/bin/env python
import numpy as np #Importa la llibreria NumPy
import cv,cv2 #Importa les llibreries OpenCV
import math

def TR(depthThresh):
        cs = cv.FindContours(cv.fromarray(depthThresh),cv.CreateMemStorage(),mode = cv.CV_RETR_TREE,
method = cv.CV_CHAIN_APPROX_SIMPLE) #Finds the contours

        imatge = cv.CreateImage((640,480), 8, 3)
        cv.DrawContours(imatge, cs, (150,150,150), (255,0,0), 1, 2, 8, (0,0))

        cont = 0
        centroid = list()
       cHull = list()
       contourArea = list()
       cHullArea = list()
        centroidList = list()
        defects = list()
       convexHull = list()

        while cs: #Iterate through the CvSeq, cs.

                # aproximacio de contorns
                ApproxCs = cv.ApproxPoly(cs, cv.CreateMemStorage(),cv.CV_POLY_APPROX_DP, 15, 1)
                cv.DrawContours(imatge, ApproxCs, (255,255,0), (255,0,0), 1, 1, 8, (0,0))

                contourArea.append(cv.ContourArea(ApproxCs)) #Llista l'area
                m = cv.Moments(ApproxCs) #busca els moments del contorn

                try:
                        m10 = int(cv.GetSpatialMoment(m,1,0)) #moment m10
                        m00 = int(cv.GetSpatialMoment(m,0,0)) #moment m00
                        m01 = int(cv.GetSpatialMoment(m,0,1)) #moment m01
                        #Llista les coordenades del centroid del contorn
                        centroid.append((int(m10/m00), int(m01/m00)))
                        #Busca envolupant convexa
                        convexHull = cv.ConvexHull2(ApproxCs,cv.CreateMemStorage(),return_points=False)
                        #Busca els defectes de lenvolupant convexa
                        defectes = cv.ConvexityDefects(ApproxCs,convexHull,cv.CreateMemStorage())
                        cont += 1 #comptador

                except:
                        pass
                cs = cs.h_next() #passa a la seguent sequencia de contorn

        for i in range(cont): #Itera segons el comptador
                font = cv.InitFont(cv.CV_FONT_HERSHEY_SIMPLEX,0.5,0.5,1)
                c = centroid[i]
#               cv.Circle(imatge, centroid[i], 10, (0,0,255), thickness=2, lineType=8, shift=0)

                sc = list()
                ec = list()
                fc = list()
                ld=list()
                Angles=list()

                for j in range(len(defectes)):
                        s,e,f,d = defectes[j]
#                       cv.Line(imatge,e,c,[0,255,255],1)
#                       cv.Line(imatge,s,c,[0,255,255],1)
#                       cv.Circle(imatge,f,5,[0,0,255],-1)
#                       cv.Circle(imatge,s,5,[255,255,0],-1)
#                       cv.Circle(imatge,e,5,[255,255,0],-1)
                        sc.append((s[0],s[1]))
                        ec.append((e[0],e[1]))
                        fc.append((f[0],f[1]))
                        ld.append(d)

                        # Busco angle entre els vectors que formen el defecte
                        sx=s[0]
                        sy=s[1]
                        ex=e[0]
                        ey=e[1]
                        cx=c[0]
                        cy=c[1]
                        s1=sx-cx
                        s2=sy-cy
                        e1=ex-cx
                        e2=ey-cy

                        cosA=((s1*e1)+(s2*e2))/(math.sqrt((s1*s1)+(s2*s2))*math.sqrt((e1*e1)+(e2*e2)))
                        Angle = math.acos(cosA)
                        Angle = math.degrees(Angle)
                        Angles.append(Angle)
```

```
                st=list()
                en=list()
                punts=list()
                for x in range(len(Angles)):
                        if Angles[x]<90: # si l'angle es inferior a 90 graus s'interpreta que es un dit
                                st.append(sc[x])
                                en.append(ec[x])
                                pss = set(st)
                                pes = set(en)
                                punts = list(pss.union(pes))
                                if (len(defectes))==1 and Angles[x]>40:
                                        punts.append(sc[x])
                        elif (len(defectes))==1: # cas que nomes hi hagi 1 defecte (1 dit)
                                punts.append(sc[x])

                dits = len(punts)

                for x in range(dits):
                        cv.Circle(imatge,punts[x],10,[0,255,255],-1)

                # Interficie grafica
                cv.Rectangle(imatge, (0,400), (320,480), (255,255,255), cv.CV_FILLED, lineType=8,
shift=0)
                font = cv.InitFont(cv.CV_FONT_HERSHEY_SIMPLEX,1,0.5,1)
                cv.PutText(imatge, "Nombre de dits:" , (20,430), font, (150,150,150))
                cv.Rectangle(imatge, (320,400), (640,480), (150,150,150), cv.CV_FILLED, lineType=8,
shift=0)
                cv.PutText(imatge, "Gest:" , (340,430), font, (0,0,0))

                font2 = cv.InitFont(cv.CV_FONT_HERSHEY_SIMPLEX,1.5,1.5,2)

                area = cv.ContourArea(ApproxCs)
                if defectes==[]:
                        cv.PutText(imatge, "%d" %(dits), (250,460), font2, (0,0,0))
                        if area < 5000:
                                cv.PutText(imatge, "Lletra A", (410,460), font2, (0,0,0))
                        else:
                                cv.PutText(imatge, "Lletra B", (410,460), font2, (0,0,0))

                elif len(defectes)==1:
                        if 50<Angles[0]<120:
                                cv.PutText(imatge, "%d" %(len(defectes)), (250,460), font2, (0,0,0))
                                cv.PutText(imatge, "OK!", (410,460), font2, (0,0,0))
                        elif 20<Angles[0]<50:
                                cv.PutText(imatge, "%d" %(dits), (250,460), font2, (0,0,0))
                                cv.PutText(imatge, "VICTORIA!", (410,460), font2, (0,0,0))

                elif len(defectes)==2:
                        cv.PutText(imatge, "%d" %(dits), (250,460), font2, (0,0,0))
                        cv.PutText(imatge, "Lletra W", (410,460), font2, (0,0,0))

                elif len(defectes)==3:
                        cv.PutText(imatge, "%d" %(dits), (250,460), font2, (0,0,0))


                elif len(defectes)==4:
                        cv.PutText(imatge, "%d" %(dits), (250,460), font2, (0,0,0))

        cv.ShowImage("Reconeixement de gestos de la ma", imatge)
```

Annex II: Configuració de l'entorn de trebal Kinect / Ubuntu / Python / OpenCV

✓ Instal·lació dels drivers libfreenect

Instruccions d'instal·lació dels controladors libfreenect de OpenKinect per Ubuntu a la pàgina oficial de la comunitat :
www.openkinect.org/wiki/Getting_Started#Manual_Build_on_Linux

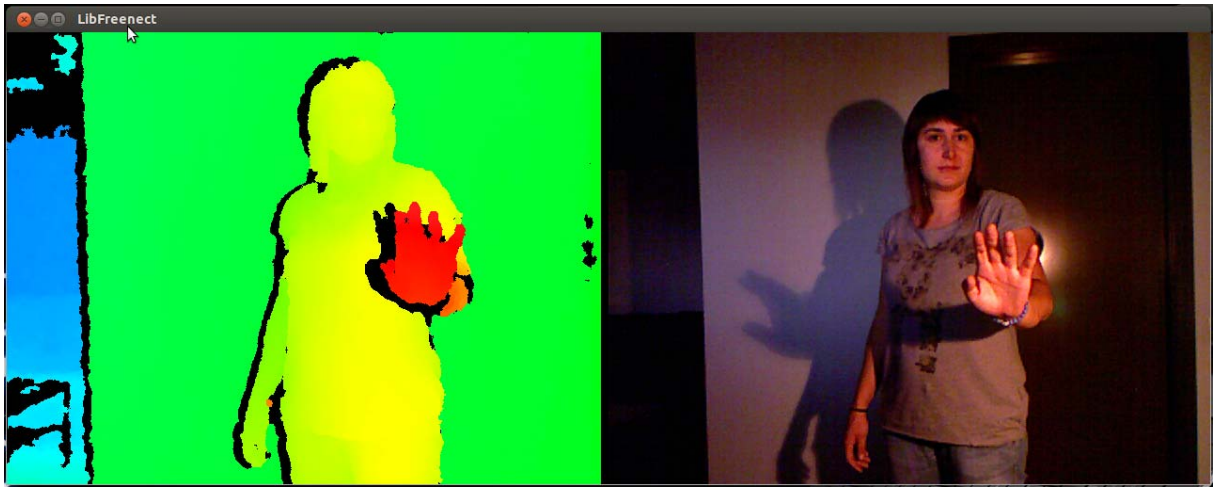
Perquè el controlador funcioni correctament cal instal·lar una sèrie de dependències:

```
* cmake
* libusb-1.0-0
* libusb-1.0-0-dev
* pkg-config
* libglut3
* libglut3-dev
```

Per tant, per instal·lar les dependències anterior i el controlador, cal obrir un terminal amb permisos d'administrador (root) i executar les següents línies:

```
sudo apt-get install git-core cmake libglut3-dev pkg-config build-essential
libxmu-dev libxi-dev libusb-1.0-0-dev
git clone git://github.com/OpenKinect/libfreenect.git
cd libfreenect
mkdir build
cd build
cmake ..
make
sudo make install
sudo ldconfig /usr/local/lib64/
sudo glview
```

Si la instal·lació s'ha efectuat correctament executant glview s'hauria d'activar el sensor Kinect i obrir la finestra següent:



✓ *Configuració de python*

Informació de configuració de l'entorn libfreenect per treballar amb el llenguatge de programació Python: http://openkinect.org/wiki/Python_Wrapper

La llibreria libfreenect té la seva adaptació per Python programada amb Cython, això implica haver d'instal·lar una sèrie de dependències per tal de que pugui ser compilada:

```
* Cython
* Python-dev
* Python-numpy
```

Les dependències anteriors es poden instal·lar directament des del centre de programari de Ubuntu.

Per instal·lar python al sistema si que ho haurem de fer des d'un terminal amb privilegis d'administrador, l'ordre és la següent:

```
sudo python setup.py install
```

Una vegada instal·lat python des de la comunitat proposen testejar una sèrie d'exemples per comprovar el correcte funcionament. https://github.com/amiller/libfreenect-goodies

En aquest cas s'han d'instal·lar les llibreries següents:

```
* IPython
* Matplotlib
* OpenCV 2.1
* PyOpengl
* wxPython
```

IPython es pot instal·lar igualment des del centre de programari, la resta haurem de fer-ho des d'un terminal:

```
sudo apt-get install python-matplotlib
sudo apt-get install python-opengl
sudo apt-get install python-wxgtk2.8 python-wxtools wx2.8-i18n
                          python-wxversion libwxgtk2.8-dev libgtk2.0-dev
```

Per instal·lar la versió de OpenCV requerida pel paquet, versió 2.1, cal:

```
sudo apt-get install build-essential libcv-dev libcv2.1
        libcvaux-dev libcvaux2.1 libhighgui-dev libhighgui2.1
        opencv-doc
```

La instal·lació d'OpenCV 2.4 implica haver d'instal·lar vàries dependències, de manera que el procediment es detalla a continuació.

✓ *Instal·lació de la llibreria OpenCV 2.4*

Per instal·lar la darrera versió de OpenCV cal instal·lar una sèrie de dependències que són essencials pel seu bon funcionament:

```
sudo apt-get install build-essential checkinstall cmake
pkg-config yasm
```

Llibreries d'Imatge:

```
sudo apt-get install libtiff4-dev libjpeg-dev libjasper-dev
```

Llibreries de Vídeo:

```
sudo apt-get install libavcodec-dev libavformat-dev
       libswscale-dev libdc1394-22-dev libxine-dev
       libgstreamer0.10-dev libgstreamer-plugins-base0.10-dev libv4l-dev
```

Ara ja es pot procedir a descarregar l'última versió de OpenCV, versió 2.4, compilar-la i instal·lar-la. (Descàrrega a Sourceforge)

```
tar -xvf OpenCV-2.4.0.tar.bz2
cd OpenCV-2.4.0/
mkdir build
cd build
```

Configuració utilitzant CMake:

```
cmake -D WITH_QT=ON -D WITH_XINE=ON -D WITH_OPENGL=ON -D
       WITH_TBB=ON -D BUILD_EXAMPLES=ON ..

sudo make install
```

Finalment, perquè els programes puguin utilitzar la llibreria cal afegir la ruta a l'arxiu /etc/ld.so.conf:

```
usr/local/lib
sudo ldconfig
```

Per comprovar si el procés d'instal·lació s'ha realitzat correctament:

```
./opencv_test_core
```

```
[==========] Running 109 tests from 85 test cases.
[----------] Global test environment set-up.
[----------] 1 test from Core_PCA
[ RUN      ] Core_PCA.accuracy
[       OK ] Core_PCA.accuracy (1950 ms)
[----------] 1 test from Core_PCA (1950 ms total)

[----------] 1 test from Core_Reduce
[ RUN      ] Core_Reduce.accuracy
[       OK ] Core_Reduce.accuracy (167 ms)
[----------] 1 test from Core_Reduce (167 ms total)
```

Si el procediment d'instal·lació s'ha realitzat correctament podem testejar-lo exectutant alguns dels exemples python que ens proporciona libfreenect de la manera següent:

```
python demo_freenect.py
```

I ha d'aparèixer la següent finestra: