Trabajo de Fin de Grado

GRADO DE INGENIERÍA INFORMÁTICA

Facultad de Matemáticas
Universidad de Barcelona

Detección automática de manos con caminos geodésicos en datos multi-modales

---

Vitaliy Konovalov

Directores: Sergio Escalera y Albert Clapés
Realizado en: Departamento de Matemática Aplicada y Análisis. UB
Barcelona, 19 de junio de 2013

Agradecimientos

A pesar de mi ausencia por motivo de Erasmus quiero dar un agradecimiento especial a Sergio Escalera por haber aceptado las reuniones por Skype. A Albert Clapés por guiarme a lo largo del desarrollo. A los dos les agradezco el tiempo que han invertido en los correos que tanto me han ayudado. También a la comunidad de pcl-users.org por resolver algunas dudas.

Índice

Abstract

In the last years, with the appearance of the multi-modal RGB-Depth information provided by the low cost Kinect™ sensor, new ways of solving Computer Vision challenges have come and new strategies have been proposed. In this work the main problem is automatic hand detection in multi-modal RGB-Depth visual data. This task involves several difficulties due to the changes in illumination, viewport variations and articulated nature of the human body as well as its high flexibility. In order to solve it the present work proposes an accurate and efficient method based on hypothesis that the hand landmarks remain at a nearly constant geodesic distance from automatically located anatomical reference point. In a given frame, the human body is segmented first in the depth image. Then, a graph representation of the body is built in which the geodesic paths are computed from the reference point. The dense optical flow vectors on the corresponding RGB image are used to reduce ambiguities of the geodesic paths' connectivity, allowing to eliminate false edges interconnecting different body parts. Finally, as the result, exact coordinates of both hands are obtained without involving costly learning procedures.

Resumen

En los últimos años, con la aparición de la información RGB-Depth multi-modal proporcionada por el sensor de la cámara Kinect™ de coste bajo, nuevas formas de resolver problemas de Visión por Computador han aparecido y se han propuesto nuevas estrategias. En este trabajo, el problema principal es la detección automática de las manos en datos RGB-depth multi-modales. Esta tarea involucra varias dificultades debidas a los cambios en la iluminación, las variaciones de posición y la naturaleza articulada del cuerpo humano, así como su alta flexibilidad. Con el fin de resolver este problema se propone aquí un método preciso y eficiente, basado en la hipótesis de que las regiones de las manos se mantienen a una distancia geodésica casi constante al punto de referencia del centro del torso detectado automáticamente. Dado un frame determinado, el cuerpo del sujeto es segmentado del fondo con la ayuda de la información de profundidad. Entonces, se construye una representación en forma de grafo en la que los caminos geodésicos se calculan a partir del punto de referencia. Se utilizan los vectores de flujo óptico denso en la imagen RGB correspondiente para reducir las ambigüedades de la conectividad de los caminos geodésicos, permitiendo eliminar falsas aristas que conectan diferentes partes del cuerpo. Finalmente, como resultado, las coordenadas exactas de ambas manos se obtienen sin recurrir a métodos de aprendizaje costosos.

Resúm

En els últims anys, amb l'aparició de la informació RGB-Depth multi-modal proporcionada pel sensor de la càmera Kinect™ de baix cost, noves formes de resoldre problemes de Visió per Computador han aparegut i s'han proposat noves estratègies. En aquest treball, el problema principal és la detecció automàtica de les mans en dades RGB-depth multi-modals. Aquesta tasca involucra diverses dificultats degudes als canvis en la il·luminació, les variacions de posició i la naturalesa articulada del cos humà, així com la seva alta flexibilitat. Per tal de resoldre aquest problema es proposa aquí un mètode precís i eficient, basat en la hipòtesi que les regions de les mans es mantenen a una distància geodèsica gairebé constant al punt de referència del centre del tors detectat automàticament. Donat un frame determinat, el cos humà és segmentat del fons amb l'ajuda de la informació de profunditat. Llavors, es construeix una representació en forma de graf on els camins geodèsics es calculen a partir del punt de referència. S'utilitzen els vectors de flux òptic dens a la imatge RGB corresponent per reduir les ambigüitats de la connectivitat dels camins geodèsics, permetent eliminar arestes falses que connecten diferentes parts del cos. Finalment, com a resultat, les coordenades exactes de les dues mans s'obtenen sense necessitat de mètodes d'aprenentatge costosos.

1. Introducción

La constante evolución de las tecnologías abre caminos a distintos estudios en los campos de *Human Computer Interaction(HCI)* y *Computer Vision*. La detección automática de las manos ha sido siempre un reto para este último ya que tiene numerosas aplicaciones de interés. Es una tarea difícil porque presenta varias dificultades que tenemos que afrontar incluyendo cambios en la iluminación, la alta flexibilidad y la deformidad de las articulaciones del cuerpo humano. Los ordenadores hoy en día se usan en casi todos los campos de la humanidad y cada vez el uso aumenta. Y es que presentan muchas ventajas a la hora de hacer cálculos o automatizar tareas repetitivas. Aparte de ser una herramienta que añade potencial al hombre, los ordenadores también sustituyen al hombre en determinadas ocasiones. La detección automática de las manos en concreto puede servir de gran utilidad en medicina, vigilancia, industria, juegos, aplicaciones de interacción hombre-máquina, biometría, asistencia al diagnóstico, monitorización de gente con necesidades, robótica afectiva, etc. El ordenador es una herramienta que puede ser equipada con sensores que le comunican con el mundo exterior. Pero una vez se tengan los datos, hay que procesarlos y hacer determinadas tareas para cumplir un objetivo. Este es el propósito de este proyecto. Al disponer de un ordenador y varios sensores se requiere programar una aplicación que permita detectar las manos de la persona y posteriormente hacer un seguimiento u otras tareas con esas coordenadas.

1.1. Ámbito del problema

Las áreas de investigación principales de este proyecto son *Computer Vision, Machine Learning* y *Human Computer Interaction*. El estudio y el diseño de interacción entre hombre y maquina es cada vez más importante dada la constante evolución de las tecnologías así como el creciente uso de éstas. Sin embargo, este campo es relativamente reciente con lo cual las herramientas que dan resultados de alta calidad escasean. Este proyecto plantea una solución que permitirá llevar a cabo el seguimiento de las manos completamente automático por parte de un ordenador lo cual tendrá un impacto elevado y multitud de usos prácticos.

En concreto, los datos del mundo real se recogerán con una cámara especial y se analizarán con el ordenador para localizar las dos manos. Dada la posibilidad de capturar datos de profundidad aparte de los tres canales de imagen pasaremos de dos dimensiones a tres lo cual permitirá aplicar el algoritmo de búsqueda de caminos geodésicos en grafos. Estos caminos permitirán verificar la hipótesis de que las manos se encuentran a distancias geodésicas constantes desde un punto de referencia en el cuerpo. Este punto se localizará de forma automática también.

Una vez el ordenador sea capaz de procesar los datos de la cámara, analizar y detectar las manos, se podrá utilizar este método en distintas aplicaciones cotidianas donde las máquinas pueden sustituir al hombre o al menos facilitarle el trabajo.

1.2. Antecedentes

Los temas de principal interés de *Computer Vision* y *Machine Learning* han sido la recuperación automática de pose y el análisis de comportamiento humano desde los principios. Aunque se empezó a trabajar con secuencias de imágenes estáticas, los recientes estudios presentan estrategias que usan tecnologías de captura de movimiento (conocido como MOCAP [1]) con Time-of-Flight cámaras [2](ver sección 3.1). Consisten en la sincronización de múltiples cámaras en entornos controlados y el análisis de los mapas de profundidad que estas permiten obtener. Las últimas tecnologías, en concreto Kinect™, han planteado una nueva manera de ver a través de computador. Aparte de los tres canales RGB ahora también se puede obtener la información de profundidad formando así un nuevo tipo de datos RGB-Depth. Con esto han aparecido nuevas estrategias para afrontar los problemas planteados.

La detección automática de las manos desde datos visuales se podría ver como un caso particular del problema de la recuperación de la pose humana. Como muchos otros problemas de *Computer Vision*, este se puede ver desde dos perspectivas diferentes. Por un lado están los enfoques [4,5,10,17] basados en el aprendizaje a partir de los datos a priori. Por el otro existen métodos [6,7,11,12,20] que estiman diferentes parámetros a partir de características observadas sin la necesidad de la fase de aprendizaje. Con relación a los primeros, el trabajo de Jaeggli et al. [5] muestra como predecir la pose de siluetas en dos dimensiones con un estimador entrenado en actividades como correr o andar. Por contra, en

[10], la predicción de la pose es estimada mediante datos estructurados con *vóxels*[1] en tres dimensiones. La mayor desventaja del enfoque basado en el aprendizaje está en la limitación de la cantidad de datos entrenados y en la probable imperfección del algoritmo usado para entrenarlos.

Han habido distintas propuestas de métodos que no incluyen conocimientos previos pero como contrapartida, se han mostrado muy dependientes de la fase de extracción de las características. Otros investigadores como Kehl y Van Gool en [6] han intentado resolver el problema montando una configuración de múltiples cámaras para adquirir de manera exacta la reconstrucción del volumen humano en tres dimensiones.

Para afrontar las limitaciones causadas por la adquisición de los datos visuales capturados por las cámaras monoculares y las complejidades de las configuraciones multi-cámara se han considerado las cámaras Time-of-Flight [3,13]. Estas cámaras incluyen la posibilidad de obtener el valor de profundidad[2] de un pixel determinado. Lo cual permite múltiples ventajas a un precio relativamente bajo. Más bajo incluso con el lanzamiento por parte de Microsoft® de la nueva tecnología Kinect™ [16]. Un dispositivo multi-sensor basado en la tecnología de luz estructurada capaz de capturar la información visual de profundidad para a continuación generar mapas de profundidad en tiempo real. Mapas que contienen rangos discretos de medidas del espacio físico. Aunque este dispositivo era destinado únicamente para complementar las consolas Xbox, dada su portabilidad y utilidad así como facilidad de instalación y uso, se ha convertido en una herramienta para muchos investigadores. En particular, Shotton et al. [17] ha presentado uno de los avances más importantes en la extracción de la pose del cuerpo humano a partir de los mapas de profundidad.

Los trabajos más recientes se han basado en esta propuesta entre otras para mejorar la recuperación de la pose humana. Los autores de [22] mejoran significativamente la estimación de la pose a partir de Random Forest[3] con la posterior optimización mediante Graph Cuts[4]. Este tipo de representación de la pose humana ha demostrado cierta mejora en los enfoques de reconocimiento gestual estándar para sistemas HCI [19]. En el otro importante trabajo Plagemann et al. [9] proponen detectar e identificar algunas partes del

1 Vóxel: del inglés volumetric pixel. Es la unidad cúbica que compone un objeto tridimensional.

2 Profundidad: la distancia desde la cámara hasta el primer objeto que se encuentra en la recta

3 Random Forest: una combinación de árboles predictores tal que cada árbol depende de los valores de un vector aleatorio probado independientemente y con la misma distribución para cada uno de estos.

4 Graph Cuts: un método de segmentación de imágenes

cuerpo (cabeza, manos y pies) en imágenes de profundidad a partir de la detección de puntos de interés. Estos se basan en la identificación de extremos geodésicos junto con su orientación. Posteriormente clasifican las partes del cuerpo usando el descriptor de formas locales (local shape descriptor) normalizado por la orientación.

La siguiente mejora de este trabajo incorpora el método de la estimación de la pose en el marco de seguimiento Bayesiano [8]. El uso de flujo óptico se ha visto útil en la segmentación de personas mientras haya objetos móviles en la secuencia de imágenes (frames) continua. En [14], el flujo óptico se usa para mejorar la substracción del fondo en la aplicación de seguimiento monocular de peatones.

En [15] Okada et al. se propone una aplicación de seguimiento de peatones en una configuración estéreo en la cual la información de profundidad y el flujo óptico son combinados para estimar el estado de la región de interés (posición 3D, postura y movimiento) a partir de una secuencia de seguimiento. Vea [18] para una revisión más detallada de enfoques más recientes.

1.3. Motivación

El mundo tiende hacia sistemas de análisis de comportamiento automáticos que sirven para comunicarnos con nuestro entorno. Igual que con nuestro entorno, con máquinas. La motivación de este proyecto viene dada por la existencia de problemas de análisis de imagen y detección de las manos. Con la aparición del dispositivo que proporciona varias modalidades visuales y es de poder más discriminatorio, se abren nuevas maneras de afrontar estos problemas.

Una nueva manera de ver el mundo por ordenador a través de la cámara conocida como Kinect permite pasar de la representación 2D a 3D. Queriendo aprovechar esta interesante ventaja he querido orientar mi investigación a un campo innovador que involucra nuevas tecnologías para encontrar soluciones a problemas que aún no las tienen así como explorar los diferentes enfoques de las investigaciones relacionadas.

1.4. Objetivos

Es de vital importancia observar el estado del arte antes de proceder con una propuesta propia. Dicha importancia es debida a la posibilidad de cometer fallos que otros ya han cometido y han solucionado. Lo que en este trabajo se pretende es encontrar la mejor solución y para saber cuál es la mejor se deben evaluar las demás y comparar los resultados. Por ello, teniendo en cuenta todos los trabajos realizados expuestos en la sección anterior, así como las limitaciones y dificultades a las que ya se han enfrentado varios investigadores, el objetivo de este trabajo es combinar las mejores soluciones en una sola y solventar el problema de la detección automática de manos en entornos controlados. Buscar solución a un problema completo que sería para escenarios con entornos abiertos haciéndola universal para todos los casos se extiende fuera del campo de este proyecto. La manera en que se va a afrontar el problema en este proyecto es primero intentar buscar soluciones a subproblemas y una vez comprobada la calidad de la solución y asegurando que cumple los objetivos establecidos pasar al siguiente de nivel más superior.

Entre otros, los objetivos más importantes son:

- Conseguir que el método funcione de la mejor forma cubriendo el mayor número de casos posible. En concreto las extremidades deben ser detectadas indiferentemente de su posición.
- Evitar fases de entrenamiento complejas previas al método para reducir el coste del procesamiento y su tiempo.
- Conseguir que el método funcione sin necesidad de condiciones iniciales como una pose específica.

1.5. Propuesta

Una vez estudiados los métodos y los enfoques existentes para afrontar el problema, sobretodo en [20], se propone en este trabajo un sistema de detección de manos completamente automático en datos RGB-depth multi-modales. Para desarrollar este sistema se han de explicar una serie de pasos:

1. Se parte de la segmentación de la parte superior del sujeto que se encuentra en frente de la cámara (dispositivo especial que proporciona información de profundidad además de RGB) del fondo (background). Dicha segmentación se lleva a cabo mediante thresholding de Otsu [23].
2. Se crea un nube de puntos (pointcloud) con los datos resultados de la segmentación (foreground) y se hace downsampling proceso por el cual se reduce la cantidad de información a tratar.
3. Se construye un grafo a partir del resultado para facilitar la posterior aplicación de algoritmos.
4. Se estima un punto de referencia invariante del torso mediante mapa de distancias que se computa a partir de la proyección 2D del sujeto. Finalmente el algoritmo de búsqueda de caminos geodésicos se aplica para calcular las distancias geodésicas desde el punto de referencia hasta todos los demás.
5. Se aplica flujo óptico denso como una restricción de frontera entre los caminos geodésicos que pueden formar partes de cuerpo que se han juntado. De esta manera se soluciona el problema de cuando por ejemplo la mano se junta al torso y los caminos geodésicos la ignoran. Así se consigue una detección precisa de las regiones de las manos.

Esta solución no requiere fases de entrenamiento costosas y complejas. Es más, al contrario de [20], el enfoque presentado en este trabajo no requiere ni tan siquiera una pose de calibración. A partir del análisis de histogramas geodésicos se reconocen las extremidades humanas a distancias geodésicas estables y cercanas. Con esto tenemos una detección robusta y completamente automática.

1.6. Aplicaciones

Una vez el ordenador sea capaz de saber donde están las manos de una persona se podrá usar esta información para conseguir diferentes objetivos o realizar tareas que el hombre tardaría más o bien se necesitarían muchas personas para llevar a cabo. Existen juegos, en los que se puede controlar al personaje haciendo movimientos con las manos. Movimientos que se repetirían por el personaje. Desde la salida de la Kinect se han desarrollado una gran cantidad de programas cuyo propósito es controlar el ordenador haciendo gestos con las manos. En la industria existen tareas manuales que se repiten por una serie de personas en cadena. Se podría aplicar nuestro proyecto en ese ámbito con tal de o bien controlar la calidad o contar las tareas completadas y saber el número de piezas. Existen otros campos donde seria muy útil conocer la posición de las manos. Vigilancia, *Human Computer Interaction*, biometría, asistencia al diagnóstico, la motorización de gente con necesidades, robótica afectiva, son otros ejemplos de posibles aplicaciones.

1.7. Organización de la memoria

En los apartados siguientes se explicará de forma más detallada el proceso de desarrollo de la solución así como los métodos empleados. El resto de la memoria se organizará de la siguiente forma:

Análisis

Se verán en detalle las distintas etapas del proyecto así como las herramientas usadas para el desarrollo. Se expondrán los motivos por los que se ha escogido una determinada plataforma y no otra, un lenguaje de programación y no otro. Se verán las librerías que se usan y también se justificará porque son convenientes para este proyecto. En la planificación se intentará explicar cómo se ha ido planificando el trabajo y los tiempos requeridos para cada tarea. En los requerimientos funcionales se describirán las funciones que se espera que tenga el programa.

Diseño

En esta parte se explicará de que manera se ha estructurado el proyecto. Con la ayuda de lenguaje UML(*Unified Modeling Language*) se mostrará el diagrama de clases que dará a entender cómo se enlazan distintos componentes del proyecto. Es un diagrama estático que

describe la estructura del sistema mostrando clases y relaciones entre ellas.

Desarrollo

Aquí se verá una explicación de cómo preparar todo el entorno de programación para comenzar a desarrollar. Se explicará cómo ha ido evolucionando el código y que problemas han habido. También se explicarán las herramientas y metodologías que se usaron para tanto facilitar como llevar un cierto control y evaluar la calidad del código.

Resultados

En esta sección se presentarán los resultados conseguidos en este proyecto. Se podrá apreciar a través de capturas de pantalla la calidad del método y algunos detalles. Adicionalmente se explicará una ampliación del proyecto que se ha producido a causa de una leve variación de requisitos.

Conclusiones

Se hará un análisis crítico de los puntos fuertes, flojos y líneas futuras del método, del escenario, de lo que ha funcionado y de lo que no. Se comentará en qué ha mejorado y qué métodos nuevos se han añadido respecto a trabajos anteriores. Finalmente se propondrán las posibles mejoras del método futuro y escenarios de aplicación futuros.

2. Análisis

En este apartado se explicará de que manera se ha llevado a cabo la programación de la aplicación para conseguir los objetivos y las funcionalidades propuestas al inicio. Se verán los motivos por los que se ha optado por utilizar unos métodos en lugar de otros. Haciendo uso del hardware disponible se justificará el uso de C++ como lenguaje principal así como las librerías que ofrecen esos métodos. Se analizará que datos son necesarios, como se obtienen y como se procesan.

No obstante, se avanza aquí que las principales razones por hacer estas elecciones son:

- la eficiencia y el rendimiento que ofrecen
- la compatibilidad entre las librerías
- reutilización del código
- el soporte y la complejidad de uso

2.1. Hardware - Kinect

A nivel de hardware, aparte de un ordenador, se utilizará un dispositivo capaz de tomar imágenes del mundo real y transferirlas al ordenador en forma digital. Básicamente se trata de una cámara dotada de ciertas capacidades útiles para este proyecto más allá de las cámaras normales. Aparte de la información de color y sonido, es capaz de proporcionar información de distancia entre la cámara y los objetos de la escena que estén a su alcance capturada por el sensor.

La cámara mencionada se denomina Kinect. Fue desarrollada por Microsoft y lanzada en 2010 para videoconsolas Xbox. Permite a los usuarios controlar e interactuar con la consola sin necesidad de tener contacto físico con un controlador de videojuegos tradicional, mediante una interfaz natural de usuario que reconoce gestos y comandos de voz.

Figura 1: El dispositivo principal usado en este proyecto - Kinect™

2.1.1. Especificaciones técnicas

- Frames por segundo (FPS): 9-30 en función de la resolución.
- Por defecto se usa un canal de 8-bit VGA con resolución (640x480 píxeles) aunque el hardware permite capturar hasta 1280x1024 con 8 FPS y otro formato de color como UYVY.
- El rango de distancias del sensor oscila entre 1.2–3.5 m.
- Visión angular del sensor 57° horizontal y 43° vertical
- El conjunto de cuatro micrófonos opera en un canal de 16-bit y 16 kHz.
- Aunque fue originalmente desarrollada para jugar, es también muy útil en investigaciones ya que permite la técnica Time-of-flight. Ésta consiste en el proceso de determinación de profundidad de la escena a partir de los cambios que la señal infrarroja emitida cuando rebote de los objeto de la escena hacia la cámara.

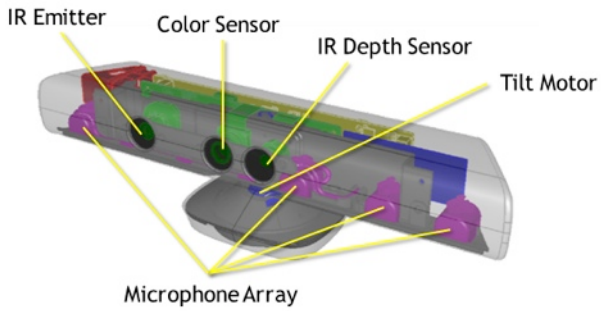

Figura 2: Constitución interna del dispositivo

2.1.2. Ventajas

Este dispositivo presenta una serie de ventajas:

- Se requiere solamente una cámara
- No se necesitan cálculos manuales de profundidad
- Adquisición de la geometría 3D de la escena en tiempo real
- Dependencia de la iluminación reducida
- Poca dependencia de la textura de la superficie
- SDK para desarrolladores

2.1.1. Funcionamiento

La Kinect se basa en el principio de ToF y funciona de la siguiente manera. Se lanzan miles de haces de luz mediante el dispositivo de infrarrojos en un entorno. La luz invisible para el ojo humano rebota en dicho entorno así como de los objetos que lo forman. El sensor CMOS capta la luz rebotada y el procesador interno de la Kinect calcula el tiempo que ha tardado la luz en rebotar y volver. Con eso se crea un mapa de profundidad. Tal y como se refleja en la figura 3.4b, para cada punto en las coordenadas XY se ve una distancia en el eje Z. Esto permite crear una tercera dimensión en la que se ven las distancias de profundidad imposibles de saber hasta ahora con las cámaras convencionales.

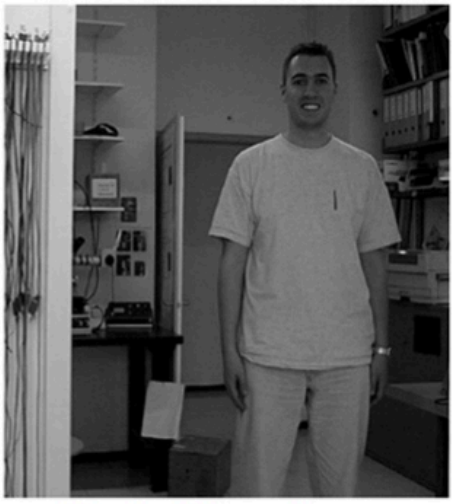

Figura 3a: imagen original en escala de grises

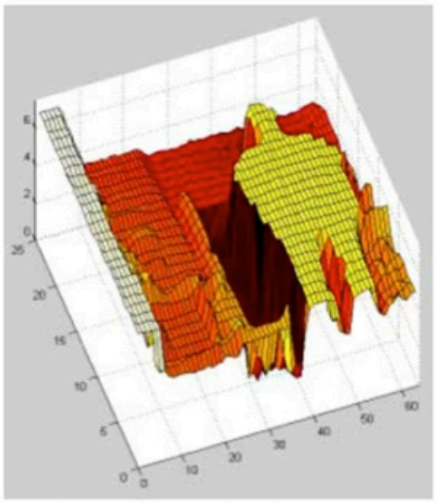

Figura 3b: Mapa de profundidad de la figura 3a

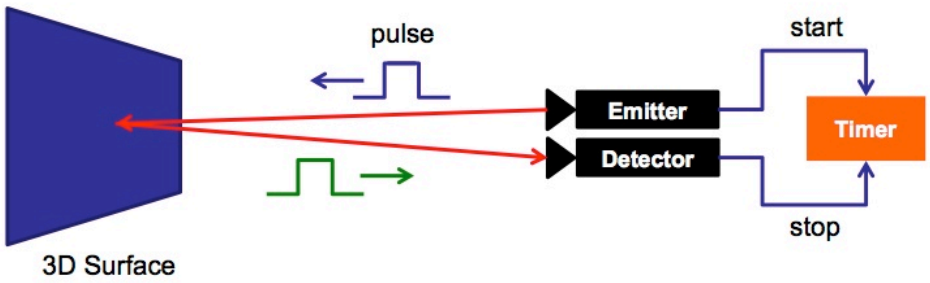

Figure 4: Imagen de [24] donde se ve el funcionamiento de ToF

2.2. Software

Para poder explicar el software y las versiones utilizadas se ha de definir primero la plataforma sobre la cual se va a trabajar. En este trabajo se ha elegido OS X como plataforma de desarrollo debido a las ventajas que presenta en cuanto a la velocidad, comodidad y la semejanza con Linux, Sistema Operativo con el cual he tenido mucha experiencia durante mis años de carrera.

Una vez instalado todo el software así como el controlador para el hardware (ver Sección 4.1 Configuración del proyecto), podemos ver varias aplicaciones de muestra que son útiles para entender los conceptos básicos y empezar a programar funciones más avanzadas.

2.2.1. Controlador

Uno de los puntos más importantes de la Kinect es que ofrece la posibilidad de conectarla a un ordenador aparte de videoconsola. Además Microsoft pone a disposición de los desarrolladores un kit de herramientas(SDK) que permite acceder a las funciones de Kinect por código. Es más, no solamente existe esta SDK que solamente está preparada para trabajar en la plataforma de Windows, sino que también hay alternativas para Linux y OS X.

En el caso particular de este trabajo, se ha usado el último controlador disponible para la plataforma OS X para marzo de 2012:

PrimeSense SensorKinect versión 5.1.2.1

2.2.2. Arquitectura (Framework)

Para explicar el Framework[5] usaré el esquema de la figura 5 encontrado en [25]. En la capa inferior (Depth Physical Layer) se encuentran los sensores (infrarrojos, profundidad, color, CMOS) y micrófonos. La siguiente capa (PrimeSense SoC) opera el hardware que hay en la capa anterior. Permite interactuar con los sensores mediante diferentes funciones como adquisición de la profundidad, correspondencia entre RGB y depth, mirroring, etc. En el

5 Framework se traduce como el marco de trabajo y define un conjunto estandarizado de conceptos y criterios para enfocar un tipo de problemática concreta.

siguiente nivel se encuentra el OpenNI. Otro framework de código abierto y soportado por miles de desarrolladores a nivel mundial. Trabaja simultáneamente con el sensor ejecutándose en el lado de host. Proporciona una API muy simple que permite al host manejar el sensor y acceder a todos los datos que éste recoge. En la penúltima capa están los diferentes algoritmos que facilitan diferentes funciones al usuario final. Utilizan los datos provenientes de capas inferiores. Por último en la capa superior se ven las aplicaciones terceras que proporcionan interfaces de usuario que manejan los algoritmos de la capa inferior.

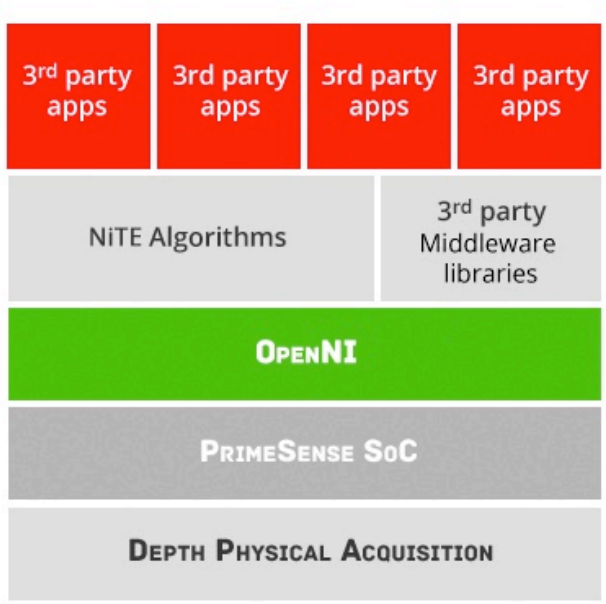

Figura 5: Framework usado en este proyecto

2.2.2.1.OpenNI

Como ya se introdujo anteriormente, OpenNI es un framework estándar de "sensing" 3D. Permite la comunicación entre una aplicación y los sensores de la cámara Kinect. Ofrece una gran variedad de funciones que facilitan el desarrollo de aplicaciones a necesidad del usuario.

Los sensores y componentes intermedios se distribuyen en módulos de manera que se activan a medida que se necesitan y solo aquellos que se necesitan. Para poder usar dichos módulos se han de crear *Production-Nodes* (nodos de producción). Son los elementos esenciales en la interfaz de OpenNI. Serán los elementos que producirán los datos a partir de los sensores. Cada nodo de producción encapsula la funcionalidad relacionada con la

generación del tipo de datos específico.

Los nodos más comunes y usados en este proyecto son:

Relativos a sensores:

- *Depth Generator*: capaz de crear un mapa de profundidad del entorno a partir de los sensores de profundidad de un dispositivo compatible y soportado por OpenNI.
- *Image Generator*: captura la imagen en color del entorno a partir de una camera RGB compatible.

Relativos a funcionalidades intermedias:

- *User Generator*: este nodo de producción crea la representación de un cuerpo detectado por los algoritmos de OpenNI en un escenario tridimensional. Proporciona una estructura de datos que marcan con puntos en forma de números, en función del número de la persona, la zona donde está la forma de esa persona. Esto facilita mucho el trabajo y ayuda en la substracción del fondo en la etapa de segmentación.

Aparte de proporcionar una API bien estructurada e intuitiva, contiene una gran colección de "samples" (pequeños programas – ejemplos que ayudan a entender el funcionamiento de los métodos de la API). Es prácticamente una herramienta básica y fundamental en los primeros pasos en el desarrollo sensorial 3D.

En este proyecto se usa la versión 1.5.4

Por otra parte existe otra colección de algoritmos y "samples". NITE™ (Natural Interface Technology for End-User) es la visión 3D por computador más avanzada y robusta disponible hoy en día. Tiene soporte multi-plataforma y está extremadamente optimizada para mínimo uso de CPU.

La versión usada en este proyecto es NITE-Dev-MacOSX-v1.5.2.21

SceneAnalysis es el único sample que se ha usado de NITE. Ha ayudado a entender como se ha de configurar los nodos y cuáles para que se pueda detectar al usuario y grabar una secuencia .oni[6]. Cabe destacar que no se ha cogido el código completo de este sample para

[6] Extensión ".oni": OpenNI proporciona este formato de almacenamiento de datos que se pueden tanto grabar como reproducir. Están formados por frames que contienen aparte de RGB, información como profundidad, número de usuarios, etc.

la los cimientos del proyecto sino que se han aprovechado funciones sueltas. La representación gráfica de lo que capta SceneAnalysis de la Kinect la hace en openGL, cosa que no nos es necesaria ya que usaremos OpenCV.

2.2.2.2.OpenCV

Se trata de una librería de código abierto que trata los campos de *Computer Vision* y *Machine Learning*. Ofrece una infraestructura común para aplicaciones de *Computer Vision*. Contiene más de 2500 algoritmos optimizados para detectar y reconocer caras, identificar objetos, clasificar las acciones humanas en vídeos, seguimiento de objetos en movimiento, extraer modelos 3D de objetos, producir nubes de puntos 3D de cámaras estéreo, unir imágenes para producir una imagen de alta resolución de una escena completa, etc.

Se ha recurrido a esta librería en casi todas las etapas de desarrollo para visualizar los resultados y sobre todo para la detección automática del punto de referencia.

La versión usada en este proyecto es OpenCV-2.4.2

2.2.2.3.PCL

PCL de Point Cloud Library es un proyecto a gran escala abierto e independiente para el procesamiento de imágenes y nubes de puntos 2D/3D. Es también multiplataforma y utiliza C++ como lenguaje de programación. Como las otras librerías contiene una serie de tutoriales e ejemplos que explican muy bien el funcionamiento tanto para principiantes como para usuarios más avanzados. El gran abanico de métodos se adapta muy bien a este proyecto. Una vez la imagen de profundidad se pasa al formato *point cloud*(nube de puntos) se pueden aplicar diferentes métodos de esta librería. Sobre todo se ha usado para visualizar la nube de puntos en tres dimensiones y hacer diferentes manipulaciones con esos puntos.

La versión usada en este proyecto es PCL 1.7

Para empezar a familiarizarse con la librería se han estudiado los siguientes tutoriales:

- Getting Started / Basic Structures
- Using PCL in your own project
- Filtering a PointCloud using a PassThrough filter

- Downsampling a PointCloud using a VoxelGrid filter
- Extracting indices from a PointCloud
- The PCD (Point Cloud Data) file format
- Reading Point Cloud data from PCD files
- Writing Point Cloud data to PCD files
- The CloudViewer
- PCLVisualizer
- Cluster Recognition and 6DOF Pose Estimation using VFH descriptors

2.2.2.4.Matlab

Se ha usado Matlab en las etapas de obtención de histogramas y evaluación de resultados finales por varias razones. Constituye una herramienta muy completa y útil en *Computer Vision*. Por un lado es un software muy completo y optimizado para las tareas que se necesitan cumplir. Presenta muchas funciones útiles y una forma rápida y comprensible de trabajar. Por el otro, es un programa que se ha ido conociendo en los laboratorios durante la carrera y se ha querido poner en práctica lo que se ha aprendido.

En este proyecto se ha usado la versión 2012 de Matlab.

Como entorno de desarrollo se ha usado Xcode 4.6.2 y SublimeText 2 por la rapidez y comodidad de programación que ofrecen. Combinando scripts escritos en bash y el código, se han hecho múltiples tests para ahorrar tiempo y cambios de parámetros manualmente.

Todas estas librerías están preparadas y escritas en C++ mayoritariamente. Aparte de ser un lenguaje muy eficiente y flexible proporciona la máxima compatibilidad entre las librerías escogidas y esta es la principal razón de usar C++ como lenguaje de programación.

2.2.3. Algoritmos

2.2.3.1.Dijkstra

Algoritmo muy popular en la comunidad de *Computer Science* sobretodo en la categoría de algoritmos sobre grafos. Basado en la búsqueda en grafos, consiste en solucionar el problema de encontrar los caminos más cortos desde un punto concreto en un grafo con pesos positivos en las aristas. Dado un vértice en el grafo, el algoritmo encuentra el camino más corto entre ése vértice y cualquier otro del mismo grafo. Puede ser usado para solucionar el subproblema de encontrar el camino más corto entre el punto origen y punto destino si se añade la condición de parada diferente. La estructura de datos que usaba el algoritmo originalmente era una simple cola con la que la complejidad era $O(V^2)$. Dicha complejidad puede ser reducida significativamente si en lugar de una cola se usa una cola con prioridades. Entonces pasa a ser O(|E|), es decir el número de aristas. Con esta optimización pasa a ser conocido como el algoritmo más rápido en este tipo de problemas.

```
procedure dijkstra(G,l,s)
Input:  Graph G = (V, E), directed or undirected;
        positive edge lengths {le : e ∈ E}; vertex s ∈ V
Output: For all vertices u reachable from s, dist(u) is set
to
        the distance from s to u.

for all u ∈ V :
   dist(u) = ∞
   prev(u) = nil
dist(s) = 0

H = makequeue (V) (using dist-values as keys)
while H is not empty:
   u = deletemin(H)
   for all edges (u, v) ∈ E:
      if dist(v) > dist(u) + l(u, v):
         dist(v) = dist(u) + l(u, v)
         prev(v) = u
         decreasekey(H, v)
```

Figura 6: Pseudocódigo del algoritmo Dijkstra. [29]

En el caso de presente trabajo, se usa este algoritmo para encontrar el camino de coste mínimo entre el punto de referencia y cualquier otro. En la figura 7 se pueden apreciar varios caminos desde el punto de la cabeza hasta los demás puntos.

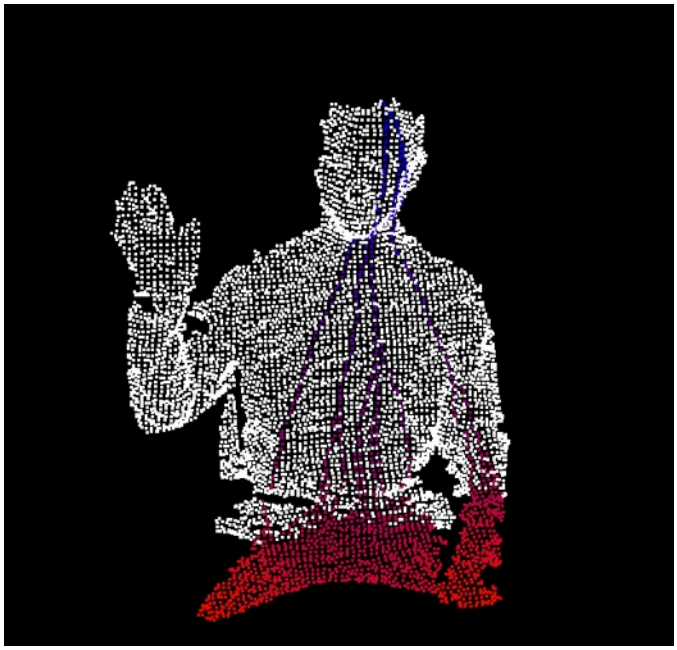

Figura 7: Ejemplo de aplicación de dijkstra sobre la nube de puntos desde la cabeza como punto de referencia.

2.2.3.2.Sort

Otro algoritmo muy útil y de uso frecuente en *Computer Science*. Proporciona una manera eficiente de ordenar los elementos de una lista según un criterio dado. Existen muchas variaciones y implementaciones que se diferencian en la rapidez y el coste de ejecución. En el caso concreto de este trabajo se ha usado la implementación de la librería estándar *algorithms* de C++. Utiliza iteradores de acceso aleatorio y se basa en un algoritmo híbrido que combina *introsort* que se ejecuta primero y el *insertion sort* sobre el resultado. La complejidad de esta implementación es N*log2(N) lo cuál es muy eficiente y es por eso que se usa. Cabe destacar que aparte del uso explícito en determinados puntos del código, este algoritmo también se usa de manera implícita en los momentos de manipulación de listas o vectores.

2.2.3.3.Optical Flow Farneback

El flujo óptico es el patrón del movimiento aparente de los objetos, superficies y contornos en una escena causado por el movimiento relativo entre un observador (un ojo o una cámara) y la escena [26].

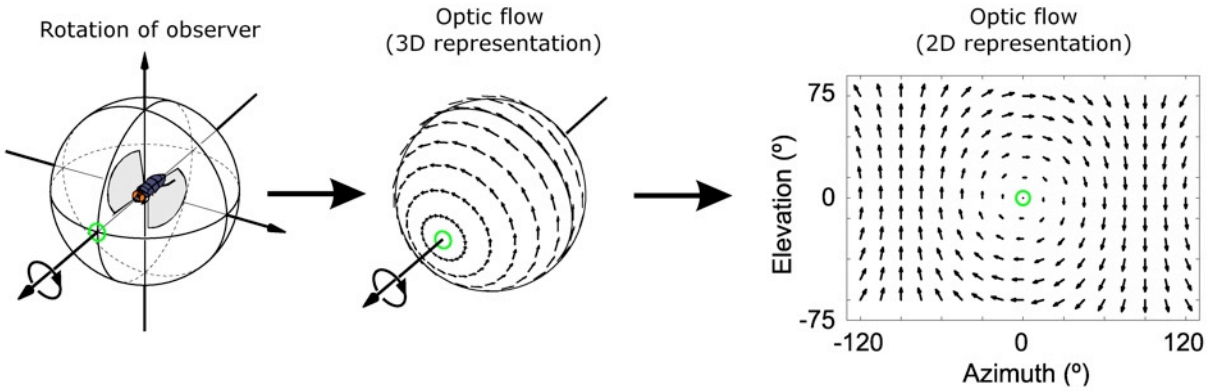

Figura 8: El flujo óptico experimentado por un observador que gira (en este caso, una mosca). La dirección y la magnitud del flujo óptico en cada posición está representado por la dirección y la longitud de cada flecha. Imagen tomada de [27].

La implementación más extendida entre la comunidad de *Computer Vision* es la de *Lucas-Kanade* pero en el caso de este proyecto se ha optado por la variación de Gunnar Farneback. Ésta consiste en estimar el flujo óptico denso, es decir, de todos los pixeles, a diferencia de L-K que procesa solo los píxeles en una vecindad de la imagen. Es una técnica más lenta pero más precisa. Dado que en primer lugar se ha centrado en la calidad de los resultados prevalece la precisión que la rapidez. Esa es la razón de escoger Farneback. Además se ha aprovechado la ventaja de que en C++ ya está implementado de manera eficiente este algoritmo. Ejemplos de aplicación del algoritmo se pueden ver en las figuras 9a, 9b.

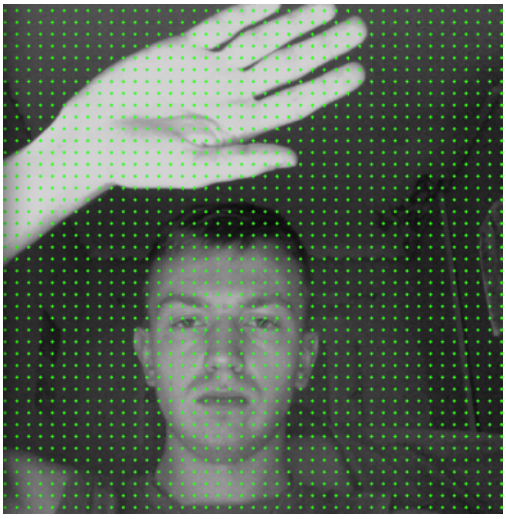

FIgura 9a: flujo óptico denso en instante t

Figura 9b: flujo óptico denso en el instante t+1

2.2.3.4.Canny

El algoritmo de Canny consiste en detectar una amplia gama de contornos en una imagen a partir de múltiples etapas. En este caso se ha usado la implementación de OpenCV que consiste en satisfacer tres criterios básicos:

- *Baja tasa de error:* la detección será buena solo si los contornos existen
- *Buena localización:* la distancia entre los píxeles del borde detectados y los reales ha de ser la mínima
- *Respuesta mínima:* una única respuesta del detector por borde

FIgura 10a: imagen original antes de aplicar el filtro

FIgura 10b: la imagen de la figura 10a una vez aplicado Canny

2.2.3.5.Operaciones Morfológicas

Se han usado dos operaciones de morfología matemática, **dilatación** y **erosión**, para quitar ruidos en la imagen resultante después de aplicar canny. La morfología matemática se basa en la teoría de conjuntos y es una técnica para el análisis y tratamiento de estructuras geométricas. Se aplica con frecuencia en imágenes digitales y *Computer Vision*. Originalmente se desarrolló para imágenes binarias pero después se extendió a funciones e imágenes en escala de grises. Erosión y dilatación son operaciones básicas y consisten en decrementar o incrementar la anchura de las líneas respectivamente.

Figura 11: Ejemplo de erosión(en amarillo) y dilatación(en verde) de una forma(en azul)

2.2.3.6.Distance map

Consiste en un mapa de distancias desde cualquier punto hasta el punto más cercano del mismo tipo. Generalmente se usa para datos en cuadrícula como píxeles o vóxels. Son algoritmos eficientes de computar, lineales en número de píxeles y rápidos en la práctica.

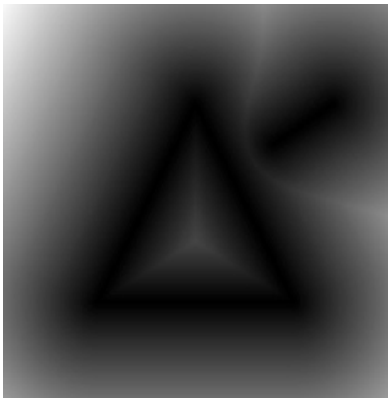

Figura 12: Ejemplo de distance map

Los usos de mapas de distancias son varios. Desde correspondencias de imágenes y reconocimiento de objetos, hasta planificación de rutas y escape de obstáculos. La fórmula para el cálculo del mapa de distancias es la siguiente:

Para un conjunto de puntos P : $DT(P)[x] = \min y \in P\ dist(x,y)$

Para cada localización x, la distancia hasta el punto y más cercano en P.

2.3.Análisis de métodos

El objetivo de este proyecto es desarrollar un método de detección automática de las manos en los datos RGB-depth multi-modales. Preciso y lo suficientemente robusto como para ser usado en muchas aplicaciones de HCI. En este trabajo, se limita el procedimiento solo para la parte superior delantera del cuerpo. No obstante, el sistema en sí es de carácter general y puede ser aplicado a otros escenarios de casos de uso ya que es capaz de detectar diferentes miembros del cuerpo. Al igual que en el enfoque presentado en [20], el método actual se basa en la suposición de que todos los puntos de referencia anatómicos (manos en este caso) se mantienen a una distancia geodésica casi constante para un punto de referencia anatómica estimado. Sin embargo, a diferencia del enfoque de [20], nuestro sistema no requiere ningún tipo de calibración para la inicialización. Los diferentes pasos del sistema se muestran en la Figura 13. A continuación, se describe cada etapa del sistema en detalle.

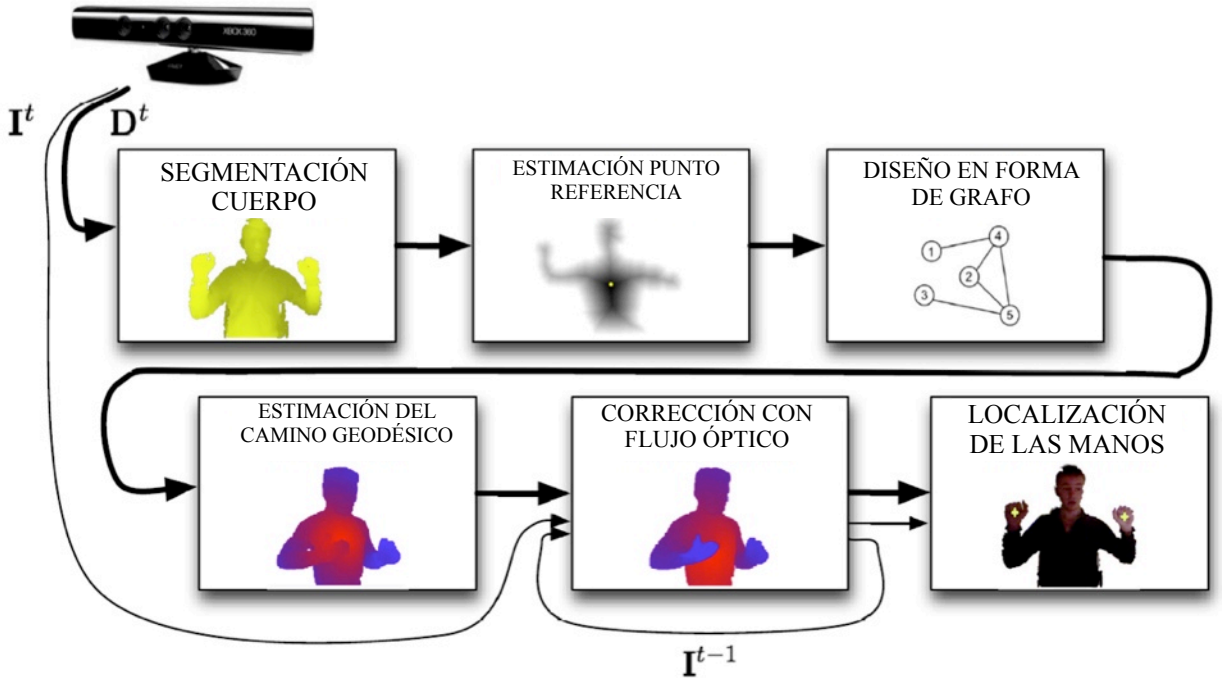

Figura 13: El procedimiento completo del sistema de la detección automatica de las manos en secuencias de datos RGB-depth multi-modales.

Segmentación del cuerpo humano

Dada una secuencia de datos RGB-depth multimodal en el instante *t*, el primer paso consiste en segmentar al usuario del resto de la escena. Para esta tarea, y dada la naturaleza de los escenarios, se ha de realizar una segmentación del primer plano en el frame de profundidad D^t basada en el valor de la profundidad de cada pixel. Por lo tanto, suponiendo una distribución bimodal de profundidad, en la imagen de profundidad D^t se aplica el método de Otsu [23] para encontrar automáticamente el valor de umbral $\alpha(D^t)$ adecuado. Adecuado será aquel que mejor separe las dos modalidades que corresponden a los objetos del primer plano y el fondo. En este caso, los objetos del primer plano son los usuarios.

El primer plano segmentado se transforma en una nube de puntos usando parámetros intrínsecos de la Kinect y se hace el llamado downsampling[7] usando voxelgrid. Este paso de downsampling consiste en la partición del espacio de la nube de puntos en algo similar a una cuadrícula de vóxels de tamaño *s*, conservando sólo un centroide[8] de todos los puntos contenidos originalmente en cada vóxel. Supongamos $P^t = \{p_{ijk}\}$ denota la nube de puntos filtrada que representa la región segmentada del primer plano. La notación indica que el punto p_{ijk} corresponde al punto resultante del proceso de downsampling en la celda de la cuadrícula (i, j, k). Este paso de downsampling acelerará enormemente las etapas posteriores ya que se reduce la cantidad de puntos que forman la nube de puntos.

Representación grafo del cuerpo

Aunque en la figura 13 este paso aparece después del que se explicará a continuación, no tiene importancia el orden en este caso. Se construye un grafo $G^t = (V^t, E^t)$, donde $V^t = B^t$ son los vértices y $E^t \subseteq V^t \times V^t$ son las aristas. Dos vértices están conectados en el grafo con una arista en base a su proximidad en la nube de puntos 3D. El conjunto de aristas se define como:

$$E^t = \{(p_{ijk}, p_{i'j'k'}) \in V^t \times V^t : || (i, j, k)^T - (i', j', k')^T ||_\infty < 1\}, \qquad (1)$$

donde $|| \cdot ||_\infty$ es la máxima norma y $(i, j, k)^T - (i', j', k')^T$ son las coordenadas 3D de un par de puntos p_{ijk} y $p_{i'j'k'}$ en B^t . En otras palabras, dos puntos p y p' están conectados por una arista en el grafo si son vecinos en un espacio 3D de 27 vóxels. Por otra parte, para cada arista e =

7 Downsampling: proceso de reducción de datos de entrada

8 Centroide: el centro geométrico

$(p, p') \in E^t$, se guarda un peso $w(e)=|| p-p' ||_2$, usado posteriormente en el cálculo de caminos geodésicos.

Estimación del punto de referencia para calcular las trayectorias geodésicas

Después de haber definido el grafo del cuerpo G^t a partir de la nube de puntos del sujeto B^t, se vuelve a procesar esta nube de puntos con el fin de detectar un punto de referencia para comenzar el cálculo del mapa geodésico. El punto de referencia $p^t_{ref} \in B^t$ corresponde a aquel punto del torso que tiene su correspondiente proyección al pixel x^t_{ref} que es el punto más alejado a todos los píxeles de la silueta del sujeto proyectado. Una estimación de pixel del torso se muestra gráficamente en la Figura 14. Se ha visto que este punto de referencia es más estable en nuestro escenario que otros puntos como la cabeza o el cuello.

Figura 14: Estimación del mapa de distancias a partir de la imagen del sujeto segmentado del fondo.

Para calcular el punto de referencia del torso p^t_{ref}, proyectamos la nube de puntos segmentada del cuerpo humano B^t en una imagen en 2D y calculamos el contorno exterior. Este mapa de contornos C corresponde a la silueta externa del cuerpo y no se ve afectada por los contornos internos del cuerpo. Para obtener una región cerrada para su uso futuro más fiable, el contorno necesita ser procesado mediante morfología matemática. Por lo tanto, se asigna a cada punto dentro de la silueta en la imagen del contorno 2D la distancia euclidiana mínima al punto del mapa de contornos C más cercano. Se le asigna al pixel de referencia del torso aquel punto cuyo valor sea el más alto dentro del mapa de distancias calculado:

$$x^t_{ref} = argmax_x \, min_{xc} \in {}_C d(x, x_C)$$

donde la x toma valores de los píxeles dentro de la silueta, x_C son los píxeles del contorno, y x^t_{ref} es el pixel de referencia del torso. Este cálculo se puede llevar a cabo de manera eficiente en tiempo lineal. Finalmente el punto x^t_{ref} se vuelve a proyectar a una nube de

puntos 3D para calcular el mapa geodésico con punto inicial definido por p^t_{ref}.

Estimación del camino geodésico

Usando G^t, se puede medir distancias geodésicas entre diferentes partes del cuerpo. La distancia geodésica $d_G(p, p')$ entre dos puntos $p, p' \in V^t$ está dada por:

$$d_G(p,p') = \sum_{e \in EP(p,p')} w(e),$$

donde $EP(p, p')$ contiene todas las aristas a lo largo de la ruta más corta entre p y p' usando el algoritmo de Dijkstra. Intuitivamente, la distancia geodésica entre dos localizaciones del cuerpo es la longitud de la ruta más corta sobre toda la superficie del cuerpo. En este punto se puede realizar la detección de las manos aunque la precisión se puede aumentar incluyendo algunas restricciones en el grafo de conectividad de los nodos en base a la información de movimiento estimada con flujo óptico. Un ejemplo del mapa geodésico calculado sin flujo óptico se puede observar en la figura 15.

Figura 15: Estimación del mapa geodésico sin el uso de las restricciones de flujo óptico. El color rojo representa valores geodésicos bajos y azul, altos.

Incluyendo las restricciones basadas en flujo óptico

La alta deformidad del cuerpo humano, y, en particular, los miembros superiores, conduce a problemas desafiantes de desambiguación. Dado el caso más sencillo de tener los brazos lo suficientemente separados uno del otro y también del torso, podemos detectar ambos puntos de las manos directamente con el procedimiento descrito hasta ahora. Sin embargo, en los casos en que los brazos se pegan a otra parte del cuerpo, es decir, que tiene al menos dos puntos $p, p' \in V^t$ que pertenecen a dos partes diferentes del cuerpo que satisfacen la condición (1) y por lo tanto pueden establecer una arista $(p, p') \in E^t$, podemos tener conexiones de grafo no deseadas entre algunas partes, y por lo tanto una estimación incorrecta de los caminos geodésicos. La figura 15 da un ejemplo donde un brazo delante del torso se conecta a la parte superior del cuerpo y la distancia geodésica del centro del

cuerpo a la mano es subestimada. En algunos casos las coordenadas de las manos pueden no detectarse correctamente. Por tanto, se propone aquí un enfoque de desambiguación, similar al propuesto en [20], que hace uso de la información de movimiento que se produce entre cada par de frames. Suponiendo que partes distintas del cuerpo se mueven por separado, este enfoque nos permite desconectar puntos (retirar la arista que los conecta) pertenecientes a diferentes partes del cuerpo.

En cada paso de tiempo, los vectores de flujo óptico denso se calculan entre pares de imágenes de intensidad I^{t-1} y I^t. Estas imágenes están en escala de grises y han sido previamente alineadas y sincronizadas con las de profundidad en el mismo instante de su adquisición. A partir de este cálculo, se obtiene $F^t = (F^t_X, F^t_Y)$, siendo $F^t_X(i, j)$ y $F^t_Y(i, j)$ el movimiento del pixel (i, j) de intensidad entre las dos imágenes en las direcciones x e y respectivamente. La Figura 16 muestra un ejemplo del mapa de flujo óptico denso estimado, remarcando los píxeles que contienen la magnitud del flujo óptico más alta calculada entre frames consecutivos. Utilizando el mapa actualizado y corregido, podemos eliminar las aristas no deseados en el grafo G^t que conectan puntos superpuestos de distintas partes del cuerpo. Por lo tanto, el conjunto de aristas se actualiza como $E^t = E^t - F^t$, con:

Figura 16: Ejemplo de mapa de flujo óptico denso remarcando en azul los puntos con mayor magnitud de movimiento

$$F^t = \{(x_{ij}, x_{kl}) \in E^t : || \, | F^t(i, j) | - | F^t(k, l) | \, ||_2 > \beta\}$$

donde β es un valor de umbral de la magnitud del gradiente de flujo óptico y |. | contiene la magnitud del vector. La magnitud de un vector de flujo óptico en la ubicación de pixel (i, j) en el instante t es:

$$|\mathbf{F}^t(i,j)| = \sqrt{\mathbf{F}^t_x(i,j)^2 + \mathbf{F}^t_y(i,j)^2}$$

La Figura 17 muestra el ejemplo de la Figura 15 una vez retiradas las conexiones del grafo basado en los valores más altos del flujo óptico. Hay que tener en cuenta que en este caso, el mapa geodésico se estima con éxito pudiendo obtener los valores geodésicas similares para ambas manos del usuario.

Localización de las manos

Figura 17: La misma imagen de figura 16 aplicando restricciones de flujo óptico esta vez para separar partes de cuerpo superpuestas.

Figura 18: Otro ejemplo de aplicación exitosa de nuestro sistema y el cálculo del mapa geodésico desde el punto central del torso como referencia.

Una vez calculado el mapa geodésico con el punto inicial en p^t_{ref}, basado en un grafo y considerando las restricciones de flujo óptico en las aristas, se va a construir un histograma de los valores geodésicos H_G . A partir del cual se podrá identificar el rango de valores H_G de distancias geodésicas que relacionan las localizaciones de cada extremidad del cuerpo humano. El histograma H_G codifica la distribución de distancias geodésicas $d_G\ (p, p')$ para todos los nodos del grafo. Un ejemplo del histograma H_G que codifica la distribución de distancias geodésicas se muestra en la Figura 19. Los valores más altos de H_G corresponden a las regiones de las manos ya que los caminos geodésicos empiezan en el punto central del torso. En la medida que se van reduciendo los valores más altos, es posible encontrar la región de la cabeza así como regiones internas del torso. En este sentido, a partir de una serie de experimentos se ha determinado que restringiendo a 1% los valores más altos del histograma, nuestro método es capaz de capturar las dos regiones de las manos y también

evita la detección de falsos positivos[9].

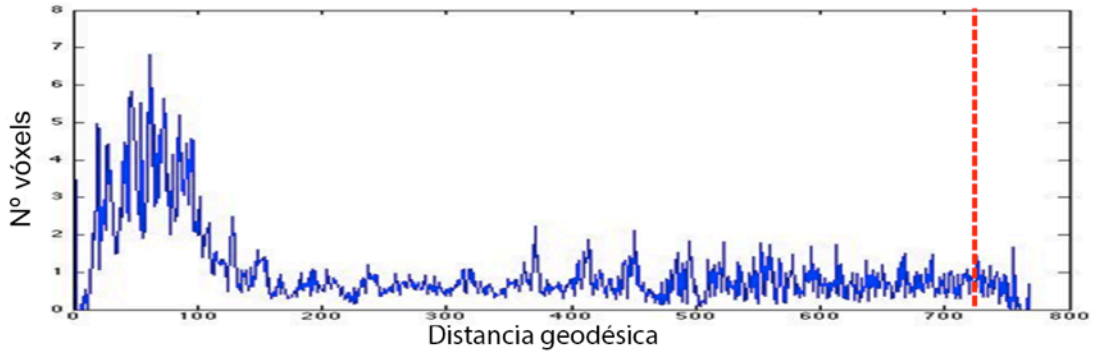

Figura 19: Histograma geodésico H_G codifica la distribución de las distancias geodésicas para una secuencia de imágenes como datos de entrada. La línea roja representa la zona que corresponde a las regiones de las manos.

Un ejemplo de la detección de los vóxels de la mano se muestra en la figura 20. Además, con el fin de evitar algunas detecciones ruidosas, después de la detección de puntos de la mano, vamos a filtrar los puntos segmentados por los operadores de morfología matemática, mantenimiento los dos componentes mejor conectados y guardando el centro promedio de las coordenadas para cada uno. Adicionalmente, la coherencia temporal se toma en cuenta para verificar la detección de una región de la mano entorno a la detección anterior por un umbral de distancia definido por γ. Esto también permite no detectar manos si se ocluyen y asegura unas trayectorias suaves. En la Figura 20, la nube verde alrededor de las manos corresponde a sus detecciones en frames consecutivos lo cual permite apreciar una trayectoria bien definida.

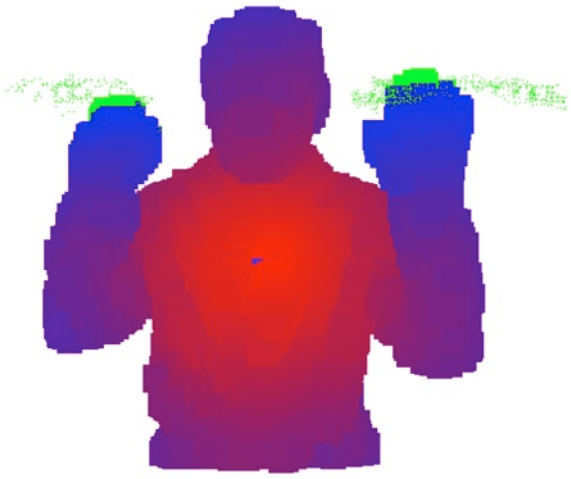

Figura 20: Trayectoria de la detección de las manos con los valores altos de geodésico pintados en verde.

9 Falso positivo: un resultado negativo es detectado como positivo

2.4.Planificación:

Para el desarrollo de este trabajo se ha seguido principalmente las indicaciones de los tutores fruto de las reuniones con ellos.

2.4.1. Plan de trabajo

Como ya se ha dicho en objetivos, la manera de afrontar el problema es escalar y progresiva. Se han desarrollado pequeñas etapas intermedias antes de llegar a la solución final.

	Name	Begin date	End date
1.	Leer y comprender antecedentes y enfoques similares	9/28/12	10/5/12
2.	Instalar y preparar el entorno	10/5/12	10/10/12
3.	Familiarizarse con el Framework	10/10/12	10/12/12
4.	Grabar una secuencia en RGBD	10/12/12	10/15/12
5.	Detectar persona	10/15/12	11/2/12
6.	Detección caras con OpenCV	11/2/12	11/30/12
7.	Aplicar dijkstra y obtener módulo Geodésico	12/3/12	1/11/13
8.	Condición frontera: Flujo Óptico	1/11/13	1/18/13
9.	Alinear RGB con Depth	1/21/13	1/25/13
10.	Reestructurar todos los módulos en una aplicación	1/25/13	2/15/13
11.	Etiquetaje - Groundtruth	2/20/13	3/1/13
12.	Testeo sobre Groundtruth	3/4/13	3/29/13
13.	Optimización código	4/1/13	4/17/13

Tabla 1: Planificación de las tareas. (Data en formato mes/dia/año)

Estimación inicial de tiempo total: 6 meses = 183 días (110 laborables)

Duración real final: 7 meses = 202 días (144 laborables)

Para llevar un cierto control de los cambios y versiones del código se ha hecho servir Git.

Figura 21: Se muestra el programa SourceTree y los proyectos activos. Geodesic es el proyecto actual.

El log completo se puede ver en la sección 8.1 de apéndice.

2.4.2. Diagrama de Gantt y costes

En el diagrama de la figura 22, se puede ver el diagrama de Gantt que representa las distintas tareas y la planificación de éstas. En términos de días permite ver el coste de dichas tareas.

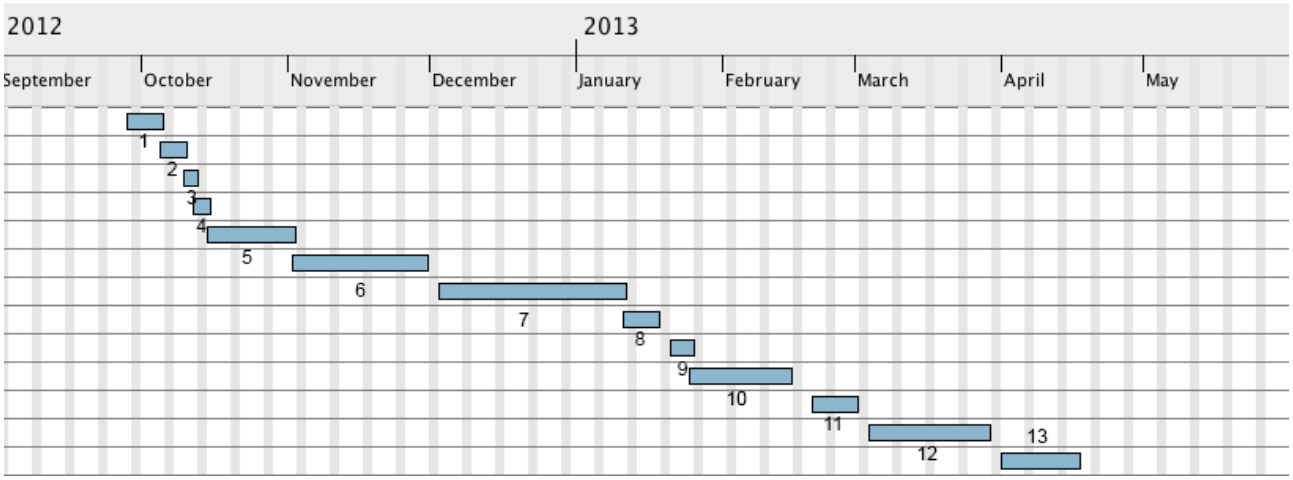

Figura 22: Diagrama Gantt

Mediante la tabla de costes (tabla 2) se ha hecho una estimación de los recursos necesarios para llevar a cabo el proyecto actual. Dada la planificación de la tabla 1, los recursos humanos y materiales que necesitamos y suponiendo una ejecución como parte de un modelo de empresa se han calculado los costes aproximados para los 7 meses. Adicionalmente dado que las licencias de software utilizado son libres se calcula coste cero. Aun así se ha de añadir el coste para propósitos académicos de Matlab ya que se ha empleado en este proyecto. Suponiendo un precio de un ingeniero 22€/h trabajando 6h/dia y teniendo en cuenta solo los días laborables 144 x 6 x 22=19008€

Ingeniero por hora	19008 €
Software (Matlab Version R2013a)	500 €
Hardware(ordenador, Kinect)	780 €
Total	**20288 €**

Tabla 2: Tabla de costes

2.5.Requerimientos funcionales

Dada una secuencia de imágenes en el formato .oni como fuente de datos de entrada, estos se deben procesar y analizar para finalmente producir unos resultados concretos.

En concreto y más en detalle el programa debe ser capaz de:

- Estar preparado para leer ficheros .oni como fuente de datos de entrada.
- Procesar los datos contenidos frame por frame y detectar tanto la persona en sí como su localización y de esta manera segmentar la persona del fondo.
- Construir una nube de puntos a partir de la segmentación anterior.
- Aplicar el algoritmo para determinar los caminos geodésicos guardando en una estructura de datos la localización y la distancia geodésica.
- Con la ayuda de flujo óptico determinar si ha habido movimiento entre el frame anterior y el actual y en qué zona concreta. Eliminar los puntos de mayor movimiento según unos parámetros dados por usuario y de esta manera evitar las ambigüedades.
- Guardar los datos en un formato determinado para un postproceso con Matlab
- Determinar las coordenadas exactas de las dos manos

2.5.1. Diagrama de casos de uso

En el siguiente diagrama de casos de uso (figura 23) se pueden ver las diferentes acciones o casos de uso que un usuario puede hacer con esta aplicación. Esto permitirá ver cuales son los principales módulos y funcionalidades que tiene que tener el sistema.

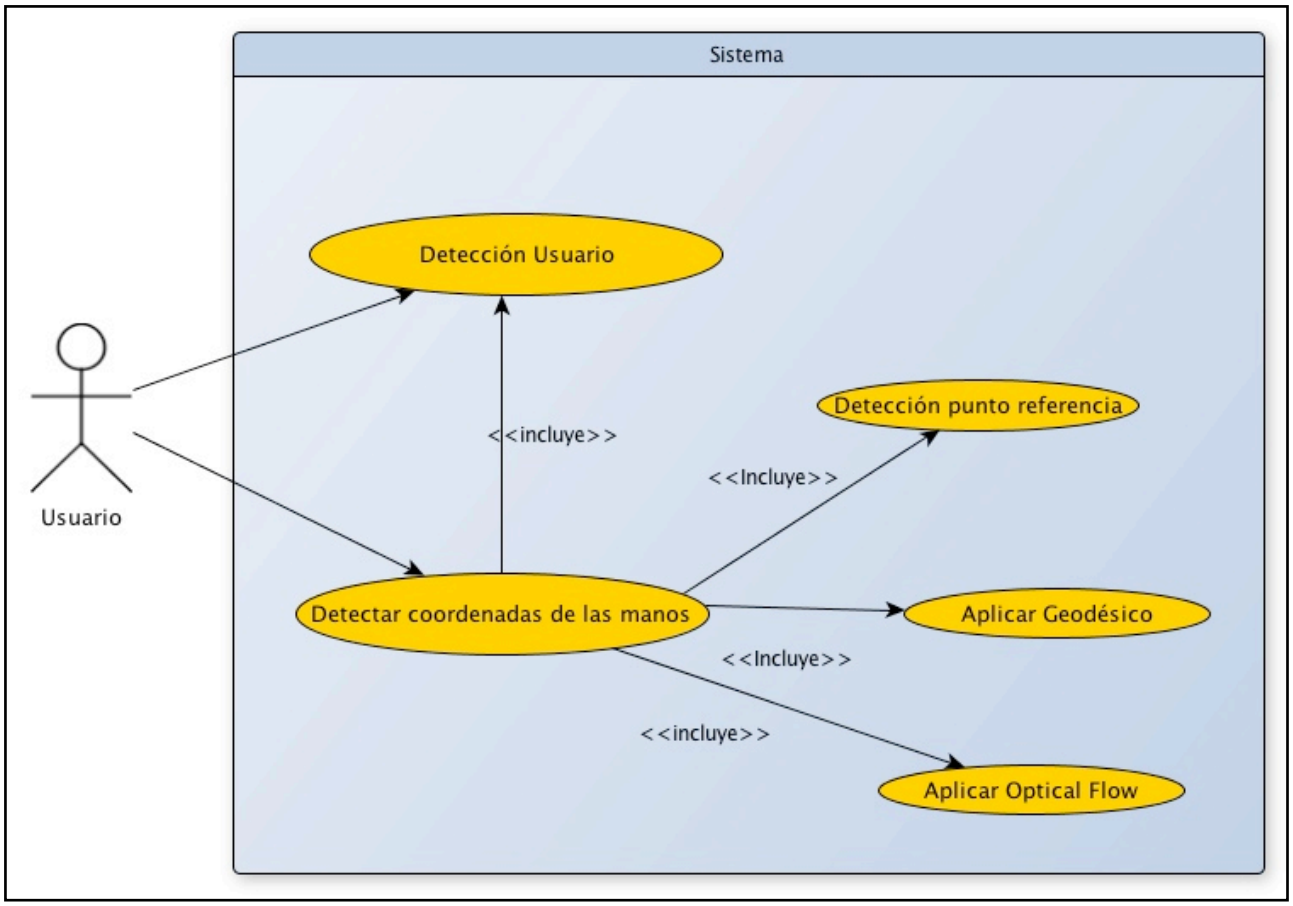

Figura 23: Diagrama de casos de uso

2.5.1.1.Caso de uso 1: Detección de usuario

UC1: Detección del usuario
Descripción: Dada una fuente de datos RGB-depth el sistema debe ser capaz de detectar al usuario marcándolo de un determinado color o mostrando mensajes por consola.
Actores: La persona que hay delante de la cámara.
Precondiciones: 1. La cámara debe estar conectada y detectada por el ordenador. 2. El sujeto debe estar en frente manteniendo una postura que le permite caber en la apertura focal de la cámara. 3. No debe haber más que una única persona en el campo de mira de la cámara. 4. En caso de que la fuente de datos fuera una grabación .oni, la ruta debe de ser correcta.
Flujo básico: 1. El usuario ejecuta el programa con la grabación como primer parámetro y "d" como segundo indicando que se quiere realizar una detección de persona. También indica el número del frame que quiere procesar. 2. El sistema comienza la ejecución y detecta la persona. 3. El sistema muestra un mensaje textual.
Flujo alternativo: 1.1 El usuario no ha pasado por parámetro la secuencia .oni. 1.2 El sistema intenta capturar los datos desde la cámara. 2.1 El sistema no detecta al usuario mostrando un mensaje pertinente.
Postcondiciones: El sistema construye una nube de puntos con la persona segmentada sin fondo.

2.5.1.2.Caso de uso 2: Detección de las coordenadas de las manos

UC2: Detectar las coordenadas de las manos <incluye> Casos de uso 1, 3, 4, 5.
Descripción: Procesar los datos de entrada y determinar la posición de las manos del sujeto que se encuentra en frente de la cámara si éste fue detectado.
Actores: La persona que hay delante de la cámara .
Precondiciones: 1. En caso de que la fuente de datos fuera una grabación .oni, la ruta debe de ser correcta y los datos de entrada deben contener la persona segmentada.
Flujo básico: 1. El usuario ejecuta el programa con la grabación como primer parámetro y "m" como segundo indicando que se quiere realizar una detección de las manos. El último parámetro debe especificar el frame específico. 2. El sistema ejecuta UC1, UC3, UC4, UC5. 3. El sistema muestra las coordenadas de las manos.
Flujo alternativo: 1.1 El usuario no ha pasado por parámetro la secuencia .oni. 1.2 El sistema intenta capturar los datos desde la cámara.

2.5.1.3.Caso de uso 3: Detección del punto de referencia

UC3: Detección del punto de referencia
Descripción: Dada una fuente de datos RGB el programa debe aplicar algoritmos de procesamiento de imagen para determinar el mapa de distancias y presentar como salida las coordenadas de su centro.
Actores: El usuario.
Flujo básico: 1. El usuario ejecuta el UC2 y el sistema llama a este módulo con una imagen RGB. 2. El sistema procesa los datos pasando la imagen a escala de grises y aplicando distintos algoritmos (Canny, Dilate, Distance Map). 3. El sistema, una vez determinada, muestra la posición del centro de referencia.

2.5.1.4.Caso de uso 4: Aplicación del Geodésico

UC4: Aplicación del Geodésico
Descripción: Cuando se ejecuta este módulo, con distintos umbrales como parámetros, se construye un grafo a partir del voxelgrid y se aplica dijkstra para obtener los caminos geodésicos. Como resultado se obtiene una estructura de datos que contiene los valores geodésicos referenciando los puntos de la nube de puntos.
Actores: El usuario.
Precondiciones: 1. Se ha de tener la nube de puntos conteniendo la persona detectada y segmentada
Flujo básico: 1. El usuario ejecuta UC2 y el sistema llama a este módulo 2. El sistema construye un grafo a partir de la nube de puntos del usuario 3. El sistema aplica dijkstra para obtener los valores geodésicos 4. El sistema da como salida la estructura de datos conteniendo dichos valores

2.5.1.5.Caso de uso 5: Aplicación de flujo óptico

UC5: Aplicación del flujo óptico
Descripción: Se determina el flujo óptico entre la imagen anterior y la actual. Se eliminan los puntos que se encuentran en la zona de flujo óptico alto.
Actores: El usuario.
Precondiciones: 1. Debe haberse procesado la imagen anterior para que entre la anterior y la actual se forme una pareja. 2. Se ha obtenido el mapa geodésico
Flujo básico: 1. El usuario ejecuta el UC2 para obtener las coordenadas de las manos con parámetro "f" que requiere la ejecución de éste módulo. 2. El sistema determina el flujo óptico entre la imagen anterior y la actual mediante la aplicación del algoritmo de Farneback. 3. El sistema elimina de la nube de puntos los puntos que pasan por la zona de flujo óptico alto desconectando de este modo distintas partes de cuerpo previniendo que se junten y produzcan falsos positivos.

3. Diseño

El propósito de esta sección es describir la solución conceptual que satisface los requisitos detallados en la sección anterior. En esta fase se detallan diagramas en el lenguaje UML de interacciones y de clase.

3.1.Diagramas de secuencia

Los siguientes diagramas muestran la interacción entre los diferentes módulos del sistema. Estos diagramas se corresponden con los casos de uso explicados en la sección anterior. Las implementaciones de algunas partes del sistema son demasiado extensas y por cuestiones de legibilidad algunos diagramas se encuentran en versión simplificada.

3.1.1.Diagrama de secuencia 1

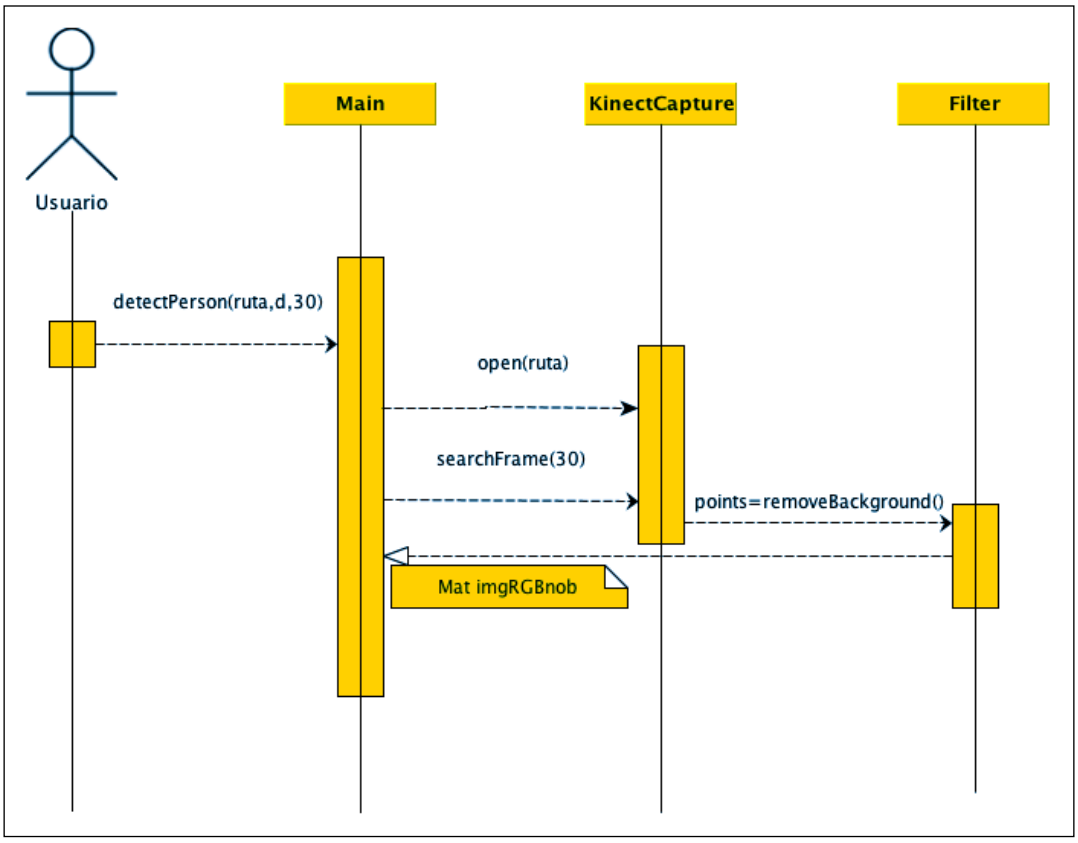

Figura 24: DS1 - Detección de usuario

3.2.Diagrama de secuencia 2

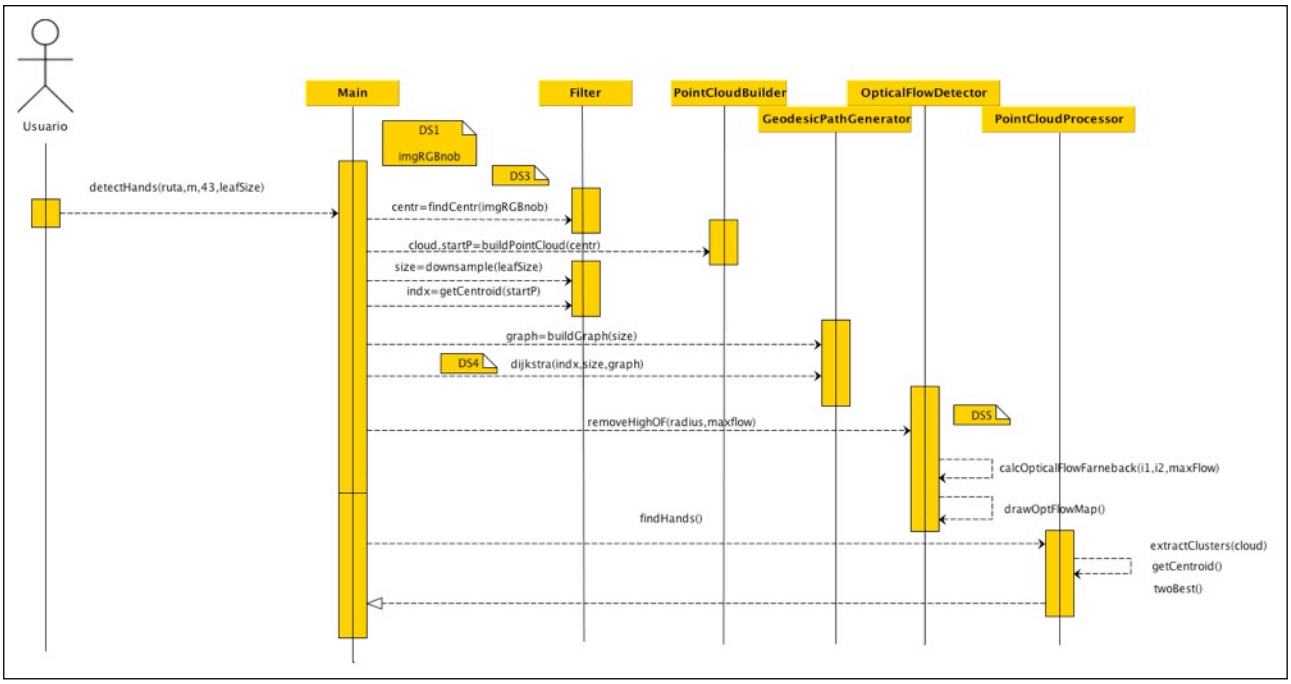

Figura 25: DS2 - Detección de las coordenadas de las manos

3.3.Diagrama de clases

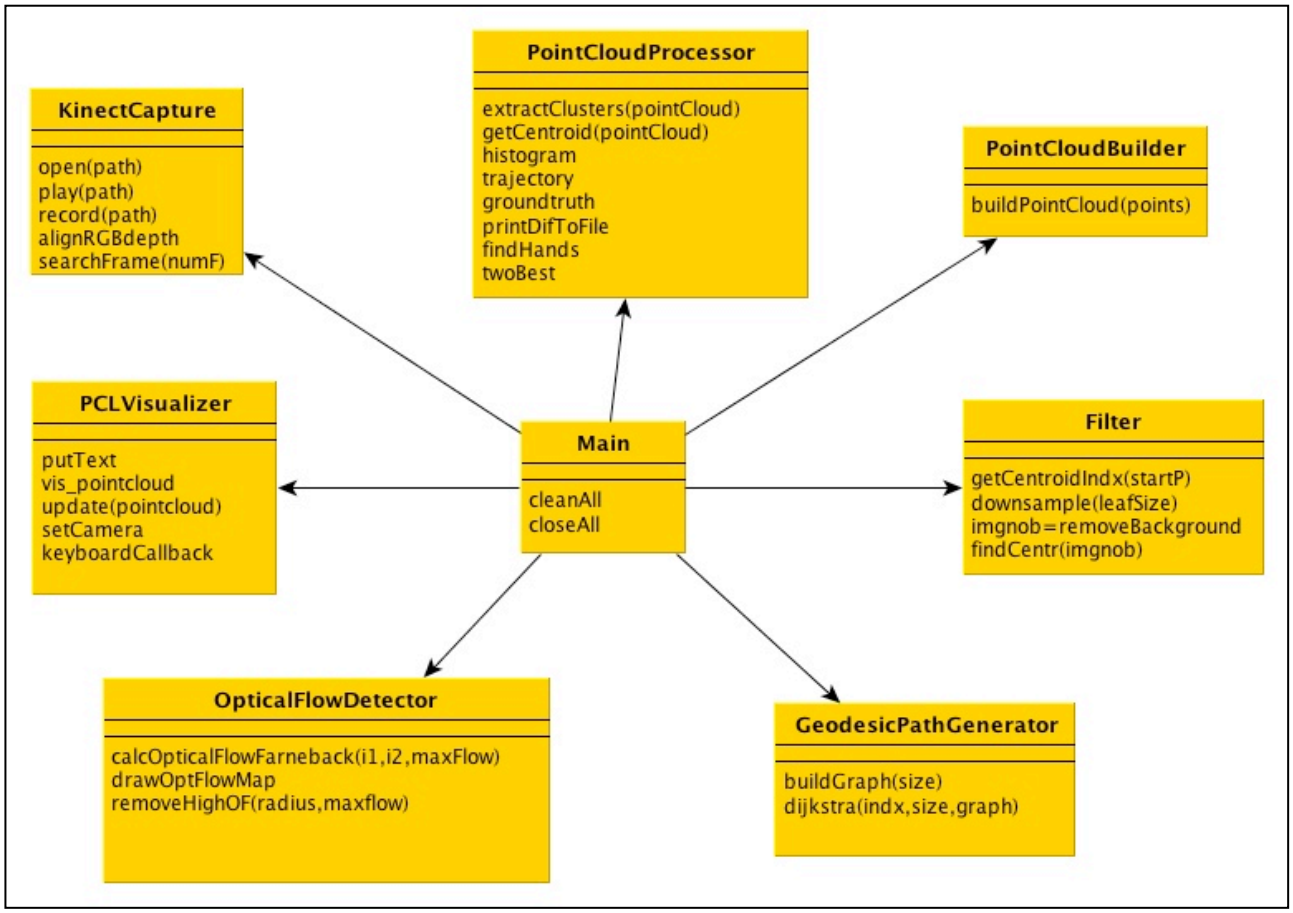

Figura 26: Diagrama de clases del sistema

4. Desarrollo

4.1. Configuración del proyecto

Al empezar este proyecto el primer paso fue leer y comprender los antecedentes, el estado de arte y los enfoques similares. Después de una larga investigación en diferentes publicaciones científicas y proyectos similares, se ha empezado con el desarrollo.

Se empezó con *Windows* y el entorno de programación *Visual Studio 2008*. Después de instalar los controladores y varias librerías necesarias como *OpenNI* y *OpenCV* se procedió al desarrollo. Pero al encontrarse con múltiples errores extraños en el sentido de la búsqueda por *Google*, se ha procedido a otro sistema operativo basado en *UNIX*. No solamente los errores fueron la razón sino que también influye la experiencia y los años de práctica en este entorno. Una vez se ha instalado todo y se ha comprobado que se puede programar sin incompatibilidades de controladores o errores extraños se ha procedido a la configuración del proyecto.

Pre-requisitos:

- disponer de *Xcode* 4.5 o superior
- tener instalado el componente "*Command Line Tools*"
- tener instalado *Xquartz* 2.7.3 o superior

Pasos para la correcta instalación y configuración del proyecto:

1. Descargar e instalar *MacPorts* 2.1.2 o superior. Una vez instalado sin errores, debe ser actualizado con el comando *sudo port -v selfupdate* en la terminal.
2. Con la ayuda de *MacPorts* se puede descargar e instalar *Cmake* necesario para la compilación del proyecto. Introduciendo el comando *sudo port install cmake* en la terminal se va a proceder a la instalación en el sistema.
3. Descargar e instalar las librerías necesarias:

- boost-1.50.0.mpkg
- eigen3-3.1.1.pkg
- flann-1.7.1.dmg
- libusb-devel-1.0.8.20101017-2.dmg
- OpenNI-MacOSX-v1.3.2.1.pkg
- QHull_2009.1.pkg
- qt4-mac-4.8.2.dmg
- Sensor-MacOSX-v5.0.3.3.pkg
- vtk5-5.10.0+qt-4.8.2.mpkg

4. Una vez instalados los componentes se va a proceder a la descarga de *PCL* desde *svn* de la página oficial. Introducir el siguiente comando en la terminal y esperar a que acabe la descarga:

```
svn co http://svn.pointclouds.org/pcl/trunk pcl-trunk
```

5. Compilar con

```
cd pcl-trunk && mkdir build && cd build
cmake -DCMAKE_BUILD_TYPE-Releas ..
make
sudo make install
```

Compilación

La mejor manera de conectar tantas librerías fue a través de *CMakeLists*. Es un archivo de configuración que permite a *cmake* encontrar las dependencias, las cabeceras y otros archivos necesarios para la compilación. La configuración usada para este proyecto se puede ver en detalle en la sección 8.2 de apéndices.

Cabe destacar que este método se ha usado para compilar muchos códigos de ejemplo de la librería PCL y las líneas importantes son estas en concreto:

```
find_package(PCL 1.3 REQUIRED)
include_directories(${PCL_INCLUDE_DIRS})
link_directories(${PCL_LIBRARY_DIRS})
add_definitions(${PCL_DEFINITIONS})
add_executable (main MACOSX_BUNDLE main.cpp)
target_link_libraries (main ${OPENNI_LIBRARY} ${OpenCV_LIBS} ${PCL_LIBRARIES})
```

En la primera línea se busca la librería en el sistema operativo. Se referencian los directorios de cabeceras y otras librerías necesarias. Finalmente se especifica como se llama el archivo que contiene el código fuente y el ejecutable del programa. En la última línea se relacionan las librerías con este archivo ejecutable.

Una vez se complete la compilación y la instalación, se va a disponer en el sistema de todo

lo necesario para empezar con la programación. Ahora se puede compilar código o bien por consola o bien con Xcode. El único requisito es disponer de un CmakeLists.txt que especifique las librerías necesarias. Con la ayuda de cmake y este fichero se genera el directorio con el proyecto preparado para Xcode.

Nota:
Dada la dificultad de seguir un orden específico y unas versiones determinadas, se ha grabado un vídeo[10] que explica todo el proceso. Después de ser publicado en la comunidad de pcl.users.org ha sido visto por 225 personas.

4.2. Implementación

A continuación se darán a conocer los detalles de implementación, problemas principales y medios usados para resolverlos.

Grabación secuencias RGB-depth alineadas

Con tal de ahorrar tiempo procesando los datos directamente desde la Kinect, se ha optado por grabar secuencias en formato .oni que permiten almacenar información RGB-depth aparte de otros datos como apertura de la cámara, resolución de la imagen, etc. Se ha usado el sample de *OpenNI* llamado *NiSimpleViewer*. Se trata de un pequeño programa capaz de grabar y reproducir archivos .oni. No obstante este programa era demasiado simple y se ha añadido la funcionalidad de alinear información RGB con la de profundidad. Para conseguirlo hay que decir al nodo generador de profundidad que tome la posición de referencia desde el nodo generador de imagen.

Manejo de la nube de puntos

En cuanto a PCL se ha estudiado varios ejemplos básicos antes de implementar la primera nube de puntos a partir de los datos propios. Se va a trabajar a lo largo de todo el proyecto

10 http://www.youtube.com/watch?v=M8x977x08Q4

con puntos de tipo *PointXYZRGB*. Permiten almacenar la información de posición 3D y el color en los tres canales.

Para construir una nube de puntos se ha de crear una estructura de tipo *PointCloud* que contiene un vector de puntos internamente. Una vez creada y inicialiada, se pueden añadir puntos.

Detección de usuario y segmentación del fondo

La siguiente etapa y la primera importante es la de detección de usuario junto a segmentación del fondo. Una vez se abre la secuencia de entrada, la información va fluyendo constantemente hasta que acabe. Intuitivamente se puede pensar en una especie de reproductor de video. Por tanto los frames singulares se capturan y se procesan mientras la secuencia se reproduce. Mediante el uso del nodo generador de usuarios de *OpenNI* se construye una estructura de datos "*sceneMetaData*". Dentro de ésta se guarda la imagen en forma de un mapa de números. Permite consultar cuantos usuarios(0, 1,…, n) se han detectado en cada pixel singular de la imagen capturada. Cabe destacar que esto se realiza de manera instantánea y eficiente ya que son implementaciones muy optimizadas.

Ahora una vez sabemos donde está el usuario, se puede construir la nube de puntos solamente con aquellos puntos que tengan valor igual a 1 (un usuario detectado en esas coordenadas). La figura 27 muestra un ejemplo de la nube de puntos del usuario sin fondo.

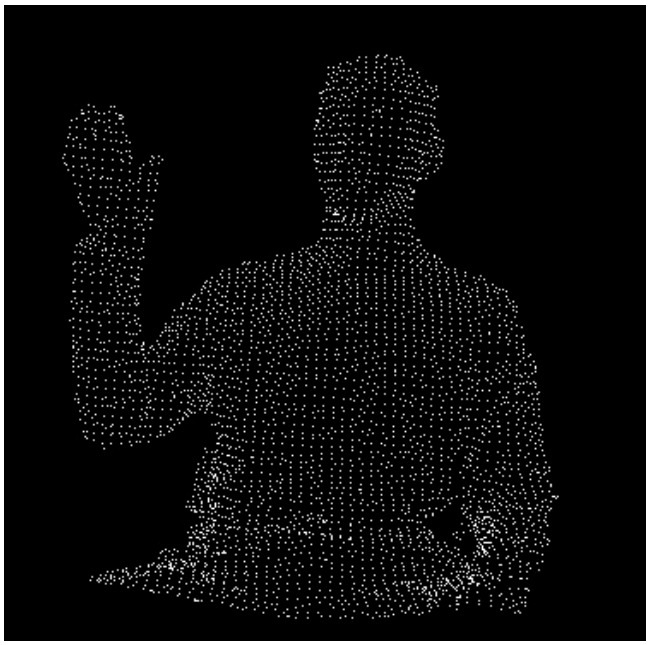

Figura 27: Ejemplo de una nube de puntos formada solamente por el usuario segmentado del fondo.

Comprensión del manejo de PointCloud

El siguiente paso será la estimación de los caminos geodésicos. Pero antes se ha de entender la manera de fijar un cierto color en cada punto de la nube de puntos con tal de poder colorear los caminos y ver de manera visual si está funcionando el método. Un recorrido por

todos los puntos y asignación de un valor 0-255 a cada canal del punto en los campos correspondientes, permiten de manera fácil y rápida colorear distintas zonas de la nube de puntos. En la figura 28 se muestra un ejemplo de una asignación de color satisfactoria.

Downsampling

Una nube de puntos puede estar formada por más de 32000 puntos. Con tal de reducir los cómputos y ahorrar tiempo se va a hacer un filtrado de esta nube mediante voxelgrid[11]. El filtro de voxelgrid está implementado en PCL y consiste en crear una cuadrícula 3D de vóxels (intuitivamente imaginar los vóxels como pequeñas cajas 3D en el espacio) sobre la nube de puntos de entrada. Como resultado, aquellos puntos que caerán dentro de un determinado vóxel serán aproximados por el *centroide* de éste. En la figura 29 se puede comprobar que las propiedades de los puntos como el color se conservan.

Figura 28: Ejemplo de una nube de puntos con dos colores

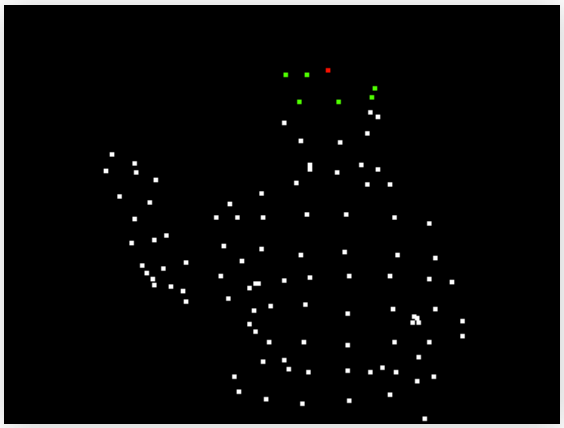

Figura 29: Ejemplo de una nube de puntos con dos colores después de filtraral con voxelgrid

Geodésico

Ahora estamos en condiciones para proceder a calcular los caminos geodésicos. Para poder determinar los caminos más cortos desde un cierto punto se va a aplicar el algoritmo descrito en la sección 2.2.3.1 de Análisis, el pseudocódigo del cual se puede ver en la figura 6. No obstante, para poder emplearlo se ha de transformar la nube de puntos en un grafo. Para llevar a cabo dicha tarea en C++ se va a usar un vector de vectores como estructura de datos. Después de aplicar voxelgrid es posible utilizar métodos que permiten encontrar los vecinos de un cierto vóxel. Para entender el funcionamiento de estos métodos se ha recurrido a *Blender* (un programa que permite modelar objetos 3D), y se ha modelado un

11 http://pointclouds.org/documentation/tutorials/voxel_grid.php

pequeño cubo que se puede ver en la figura 30. Esto ha permitido visualizar el problema y plantear una solución válida. Mediante tres bucles anidados se ha creado en C++ una estructura que simula este cubo y permite crear una vecindad de 27 vóxels. A continuación aplicando métodos de PCL y haciendo uso de distancia euclídea se ha podido transformar una nube de puntos en un grafo. La aplicación de dijkstra ha sido inmediata. Lo único que hay que tener en cuenta es que el grafo debe tener pesos. En este caso mientras se creaba el grafo se han tomado las distancias euclídeas entre vóxels como peso de las aristas.

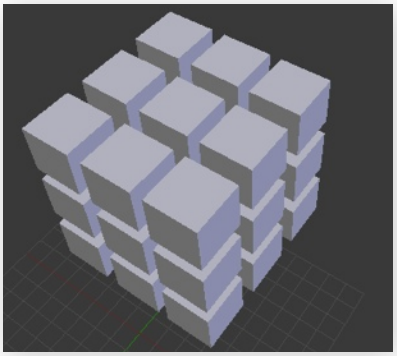

Figura 30: Cubo 3D formado por 27 cubos. Representa la vecindad de vóxels en el espacio.

En las primeras iteraciones del desarrollo se ha usado el punto más elevado como referencia y punto de partida del dijkstra. Pero en la siguiente etapa se determinará el centro del torso por mapa de distancias.

La figura 31 proporciona un ejemplo del mapa geodésico resultante de los pasos explicados anteriormente.

Determinación del punto de referencia

El proceso para encontrar el punto se basa en el mapa de distancias explicado en la sección 2.2.3.6 de Análisis. En las figuras 32a y 32b se observan los pasos siguiendo un orden natural.

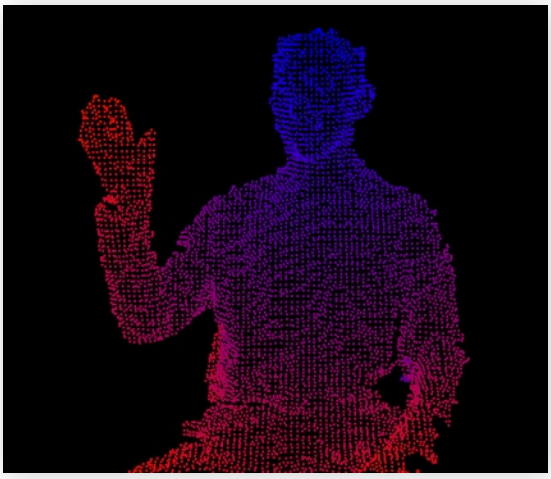

Figura 31: Ejemplo de mapa geodésico con el punto más elevado como punto de referencia.

Figura 32a: Se observan 5 pasos: Dilatación, Canny, Inversión, Máscara binaria, Mapa de distancias

Primero se aplica una serie de filtros explicados en la sección 2.2.3. En la figura 32a primero se aplica dilatación para juntar los píxeles que faltan para rellenar la figura blanca. A continuación se aplica el *canny*. Permite obtener los contornos de una imagen. Vemos que contiene algunos ruidos y el contorno que rodea el sujeto no está completamente cerrado. Aplicando morfología matemática se consigue cerrar el contorno. Como último paso se estima el mapa de distancias que a partir del contorno cerrado va adentrando hacia el centro marcando con una intensidad superior a medida que aumenta la distancia. De este modo el punto más alejado de todo el contorno corresponde al centro del torso.

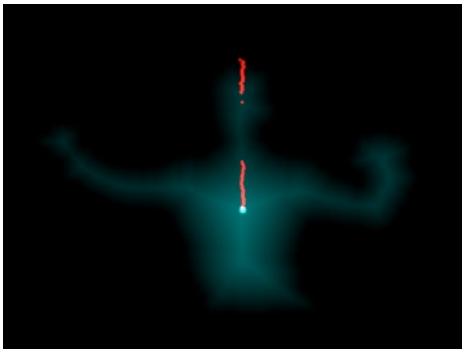

Figura 32b: Proceso de búsqueda del punto de referencia dentro del mapa de distancias.

Estimación de flujo óptico

Ahora que se tiene el centro del torso como punto de referencia para el geodésico se han de añadir las restricciones de frontera basadas en flujo óptico. En la sección 2.2.3.3 de Análisis y las figuras 8, 9a, 9b se explicó como se determina. Con tal de eliminar las regiones donde la magnitud de los vectores de flujo óptico era superior a un cierto umbral, se ha intentado primero marcar de color las regiones a eliminar.

Figura 33: Muestra la detección de las regiones de mayor movimiento entre dos frames consecutivos pintadas con líneas diagonales blancas.

En la figura 33 se puede apreciar como mediante recorridos simples a partir de bucles anidados se ha cambiado el color de los puntos. La zona de las manos presenta especial interés para este proyecto y se ha de poder manipular sin dificultad.

La figura 35 presenta los clusters que forman los puntos pintados en negro que representan las manos de la figura 34. Una función útil que se puede aplicar a los clusters es la determinación del centroide (el punto más céntrico en el espacio 3D de los puntos del cluster). A

partir de esta información se ha podido valorar la calidad del método ya que sabiendo la distancia entre los puntos de las manos reales (groundtruth) y los centroides de los clusters detectados por el método se puede estimar cuanto error se produce.

Finalmente el sistema completo se demuestra en la figura 36.

Figura 34: Las regiones en negro corresponden a las manos detectadas automáticamente por el sistema.

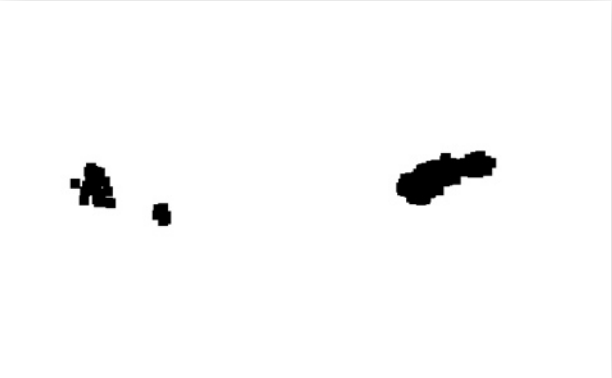

Figura 35: Clusters de puntos correspondientes a las regiones de las manos de la figura 34

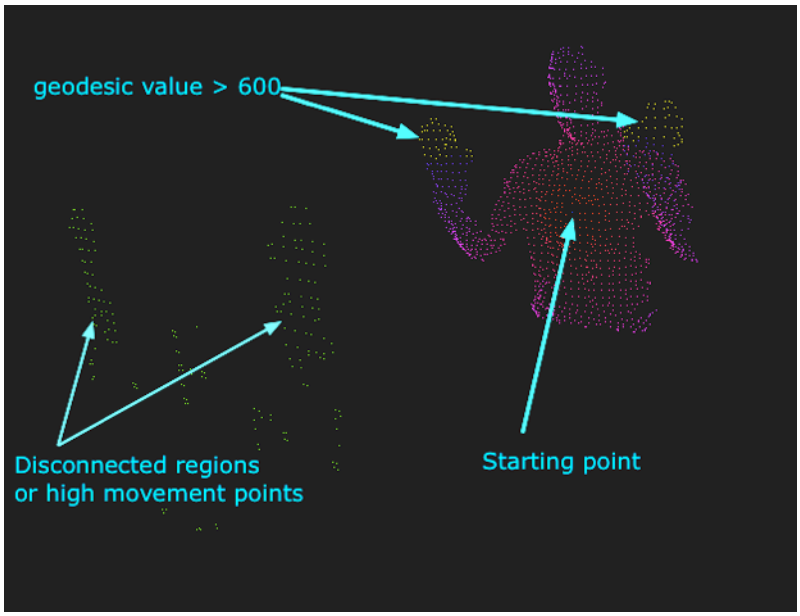

Figura 36: Ejemplo de ejecución de todos los módulos del sistema

Problemas y soluciones:

Durante todas las fases del desarrollo y sus iteraciones se ha recurrido a *Debugging*. Cada vez que se obtenía un resultado no esperado a partir de la visualización, se ha parado la ejecución del programa para ver el estado de las variables y determinar el posible fallo. Ha sido una funcionalidad altamente necesaria para poder programar más rápido. En la figura 37 se puede ver un ejemplo de depuración en Xcode.

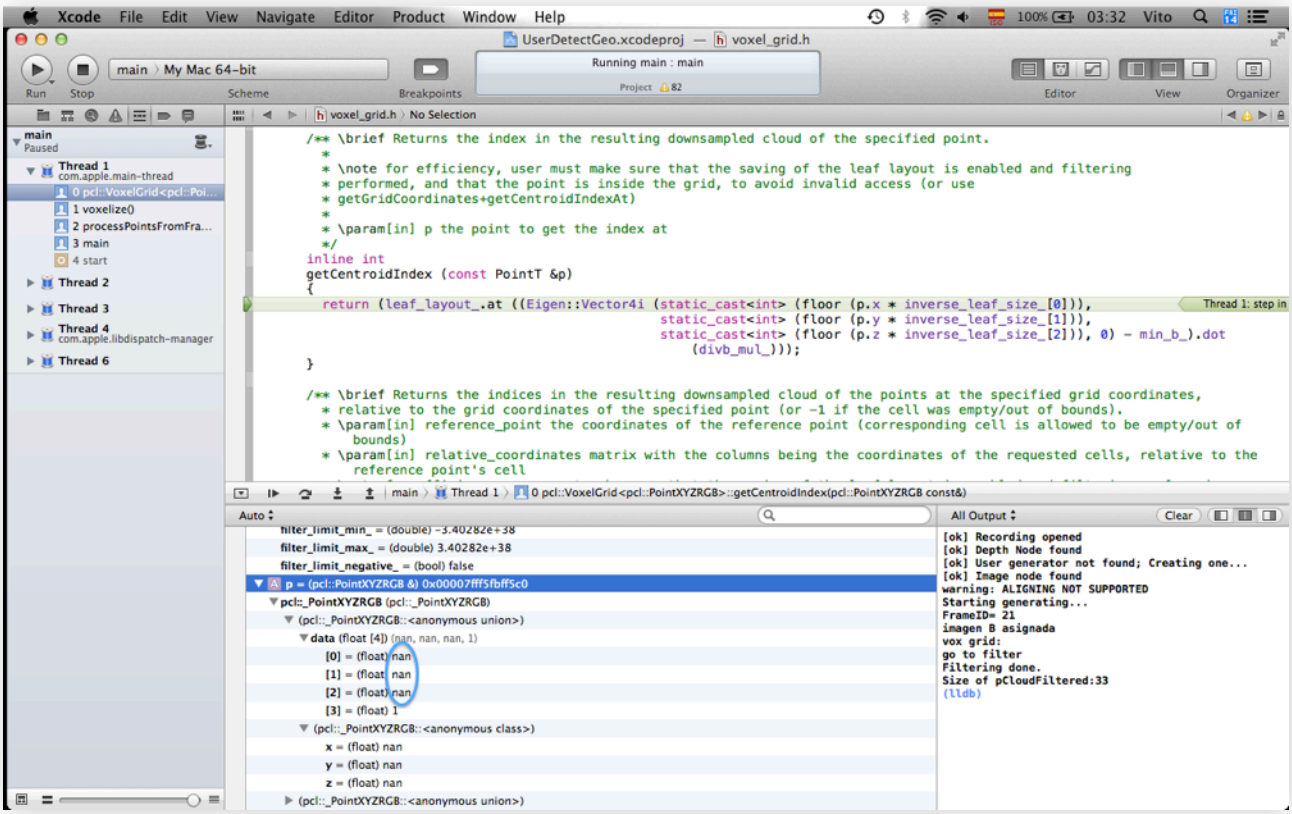

Figura 37: Ejemplo de depuración del programa en el entorno de programación Xcode

Aparte de fallos técnicos también han habido problemáticas relacionadas con las limitaciones del dispositivo. Un ejemplo de cuando sistema puede fallar es cuando se ocluyen partes del cuerpo. Por ejemplo en la figura 38 se ve como debido a la inclinación de la cabeza del sujeto, se ocluye el cuello. Internamente conlleva al programa a considerar dos regiones separadas. Para solucionar este problema se ha elegido un ángulo de visión de la cámara que permita ver todas las partes sin oclusiones.

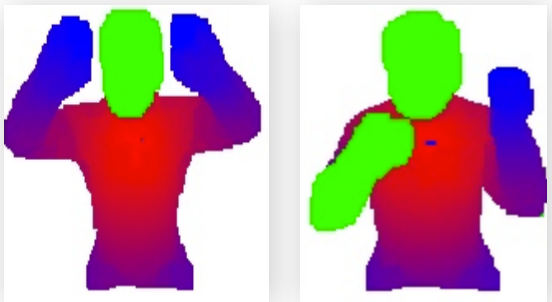

Figura 38: Ejemplo de oclusión. La región verde se considera como región desconectada

5. Resultados

A continuación se exponen los resultados conseguidos durante este trabajo. Cabe destacar que para un mejor entendimiento primero se explica qué datos se han empleado, la configuración que se ha empleado y cómo se han validado esos resultados mediante los experimentos.

5.1.Construcción de groundtruth

Se ha grabado un conjunto de seis secuencias diferentes. Cada una consiste en seis personas distintas escogidas de manera que permitan ver la independencia de nuestro método en cuanto a la forma y color de la piel. En dichas secuencias cada persona realizará una serie de gestos naturales y incontrolados en frente de una cámara Kinect. La única condición es que este trabajo se centrará en la parte superior del cuerpo. Por tanto se recogerán datos multi-modales conteniendo datos RGB y depth a partir de las secuencias grabadas. Los datos consisten en un total de 3000 frames tanto RGB como depth de una resolución 640x480 pixeles. En aquellos frames donde las manos eran visibles(un total de 2171), sus regiones han sido etiquetadas usando coordenadas 3D del mundo real. En la figura 39 se pueden apreciar algunos ejemplos:

Figura 39: Muestra los tres usuarios que se han escogido para validar el sistema de este proyecto. Se trata de una pequeña base de datos de groundtruth. La primera columna representa la imagen original con las manos etiquetadas manualmente. La segunda muestra la información de profundidad recogida por la Kinect. La última columna muestra el resultado de nuestro sistema despues de haber procesado los datos.

5.2.Métricas

Para todos los experimentos se han usado los siguientes parámetros(umbrales o thresholds):

$$\beta = 0.2, \gamma = 5\ mm$$

Con tal de validar los resultados de nuestro sistema, hemos usado los datos de groundtruth que contienen las manos etiquetadas manualmente. Adicionalmente para una validación mejor, se han variado las tolerancias de umbrales en las coordenadas 3D. El rendimiento final del sistema se muestra como el porcentaje de las manos correctamente detectadas.

A parte de estos thresholds importantes, también hay que tener en cuenta otros que aunque son menos relevantes no dejan de ser necesarios.

Para la realización del downsampling mediante voxelgrid hay que definir un tamaño de hoja (leaf size) que al fin y al cabo es lo que influye en una reducción mayor o menor de puntos.

Un leaf size de 1cm permite conseguir un 91% de reducción ya que de 460400 puntos se pasa a tener 41045 puntos.

Existen otros dos parámetros en cuanto a los clusters. Se ha de fijar un tamaño mínimo y otro máximo del número de puntos que pueden formar un cluster. En nuestro caso se ha usado 2 y 1083 respectivamente.

5.3.Testeo sobre groundtruth

Para probar la exactitud del método de detección automática de las manos, se ha probado el método presentado en este trabajo sobre el conjunto de datos disponible. Para comparar el groundtruth con ubicaciones de las manos detectadas automáticamente por el sistema, se estima el centro de masa de cada componente conexo y se compara con los datos de groundtruth. En cuanto al número de manos detectados correctamente, se introduce un umbral de distancia λ, parámetro expresado en milímetros, y se prueba la exactitud de reconocimiento para diferentes valores de este umbral λ. En los experimentos, se ha fijado un valor máximo de 25mm para estimar detecciones precisas útiles en sistemas de HCI. También se ha probado diferentes porcentajes de puntos geodésicos más altos en H_G para buscar los valores de sensibilidad y mejorar este parámetro.

Para determinar la precisión del método se aplican únicamente los cálculos geodésicos desde el centro del torso como punto de partida, y se incluye la restricción de flujo óptico con $\beta = 0{,}2$. Los resultados se muestran en la Figura 40. En la Figura 40 (a), se puede ver que se consigue el mejor rendimiento para 25 mm de umbral distancia y 1% de distancias máximas geodésicas. Como es de esperar, si aumentamos el umbral distancia se conseguirá mayor precisión. Sin embargo, sólo queremos conseguir una detección lo suficientemente precisa como para adaptarse a los requisitos de HCI. Cuando se incrementa el porcentaje de valores geodésicos, se detecta un mayor número de puntos de la mano, y en consecuencia, el centro de masa de la región detectada se desplaza en relación con groundtruth, lo que resulta en una reducción del número de manos detectadas. En la Figura 40 (b) se pueden ver los casi los mismos resultados, incluyendo las restricciones de flujo óptico. Como se muestra, se consigue el mejor rendimiento para valores similares a cuando no se usa flujo óptico. En este caso pero, la precisión se incrementa en un rango entre 5%-10% en relación con los resultados proporcionados en la Figura 40 (a). Los mejores resultados para ambas estrategias se muestran numéricamente en la Tabla 3.

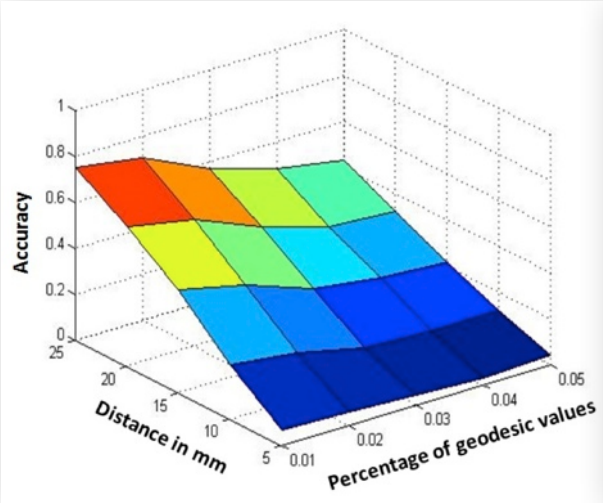

Figura 40a: Resultados obtenidos una vez se ha testeado el groundtruth

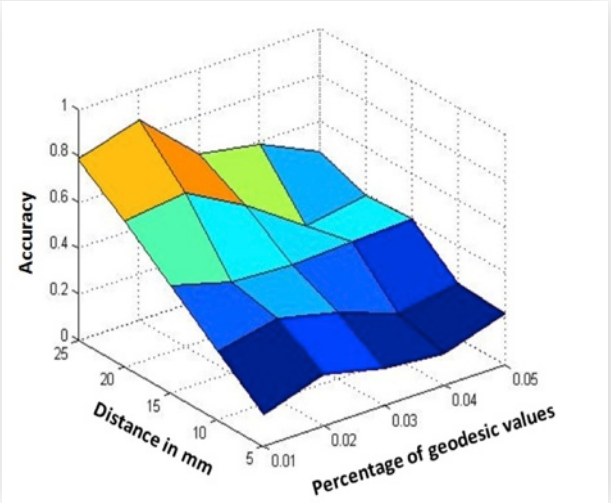

Figura 40b: Resultados obtenidos una vez se ha testeado el groundtruth incluyendo flujo óptico

	Camino geodésico	**Camino geodésico con flujo óptico**
Precisión de la clasificación	74,12%	84,15%

Table 3: La mejor precisión de la detección automática mediante caminos geodésicos con/sin flujo óptico.

5.4. Ampliación

A mitad de desarrollo el proyecto tuvo una ampliación causa de una variación de los datos de entrada en un momento determinado de desarrollo. Se han añadido unos nuevos requisitos. En concreto se ha intentado adaptar el código para que tomase como datos de entrada unas imágenes que solo presentan información de profundidad. En lugar de procesar una secuencia .oni formada por datos obtenidos de la Kinect(RGB-depth), se ha incluido la posibilidad de entrar datos formados únicamente por la información de profundidad. En las figuras 41a y 41b se puede ver la naturaleza de los nuevos datos de entrada.

Figura 41a: Se muestra un ejemplo de los datos de entrada nuevos conteniendo únicamente la información de profundidad.

Figura 41b: Imagen inmediatamente posterior a la 41a.

Como se puede comprobar siguen siendo pares de imágenes lo cuál permite estimar flujo óptico como en el primer escenario presentado más arriba. No obstante primero hay que segmentar al usuario del fondo. Para ello el primer problema surgido es la detección de persona. Ahora al no disponer de la posibilidad de usar funciones de OpenNI para la segmentación del fondo y detección de persona, se ha empleado una nueva técnica. Dado que estas imágenes se pueden ver en forma de capas según la profundidad(figuras 42-44), se ha seleccionado un valor determinado de profundidad que contiene justamente la forma del usuario. De esta manera se empezó a trabajar con estas imágenes.

A continuación se mostrará a base de capturas de pantalla y sin volver a explicar todo el proceso, los resultados conseguidos. Se puede ver que son los mismos que los que se consiguen a partir de una secuencia .oni.

Figura 42: Se ve la descomposición en capas de las imágenes de entrada según el valor de profundidad.

Figura 43: La imágen de la figura 42 vista desde frente con una cierta inclinación.

Figura 44: La misma imágen vista desde frente.

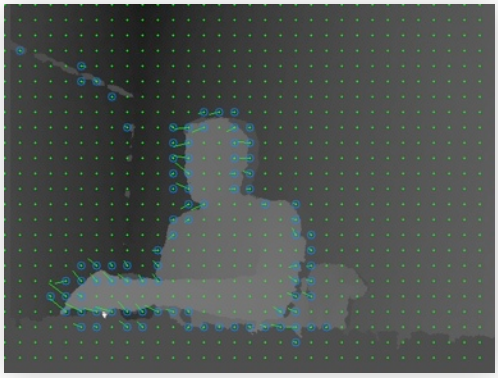

Figura 45: Mapa de flujo óptico denso estimado a partir de un par de imágenes de profundidad

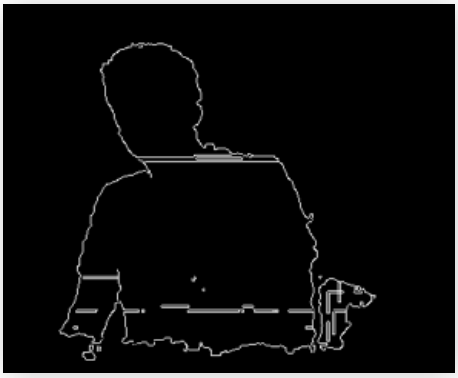

Figura 46: Contornos extraidos a partir de una imagen de profundidad

Figura 47: Mapa de distancias conseguido a partir de una imagen del nuevo escenario

6. Conclusiones

Se ha propuesto un sistema para la detección automática de las manos en las secuencias de datos multi-modales. El método se basa en la segmentación y la representación del cuerpo humano como grafo a partir de mapas de profundidad. A través de la información RGB se determinan los vectores de flujo óptico que se utilizan para eliminar las ambigüedades de aristas en el grafo. Los caminos geodésicos se calculan para obtener las distancias geodésicas desde un punto de referencia del torso estimado hasta todos los demás puntos que forman la superficie del cuerpo. Se usan distintos valores de distancia geodésica para detectar automáticamente ambas regiones de las manos.

El método es simple, robusto, eficiente y completamente automático, sin necesidad de una fase de entrenamiento o protocolos de inicialización fijos. Se ha trabajado sobre datos reales de tipo RGB-depth multi-modales en distintas condiciones ambientales y con los comportamientos arbitrarios de los sujetos. Los resultados muestran una alta precisión y adecuación del método para ser aplicado en escenarios reales HCI. Los principales casos en los que el sistema presentado falla se deben a dos razones. En primer lugar, a pesar de que es capaz de detectar las manos, no siempre es el 1% de los puntos geodésicos máximos el número óptimo para un subconjunto de imágenes. En consecuencia se desplaza el punto medio de la mano, sin satisfacer la restricción de la máxima distancia espacial de 25 mm. Y en segundo lugar, algunas configuraciones de brazo/mano en frente de la superficie del dispositivo ocultan información sobre la conectividad de las regiones. Como resultado, el camino geodésico no conecta diferentes partes del brazo/mano y finalmente se pierden. Una posible solución para ampliar el sistema incluye un dispositivo de calibración adicional que haga la reconstrucción de la nube de puntos para permitir la conexión de todos los puntos de la superficie y por lo tanto reduciendo el porcentaje de puntos ocultos. Este sistema está diseñado en escenarios controlados suponiendo que es para un tema de interacción hombre máquina lo cual supone que pueden haber unas partes visibles y otras no. No obstante, hay muchos gestos que pueden producir más oclusiones de las esperadas. Una posible propuesta para trabajos futuros que se planteen aplicar este método en entornos no controlados, hay que combinar muchas cámaras para evitar tantas oclusiones. También hay que recordar que en este trabajo se ha centrado en la parte superior del cuerpo. En otros escenarios habría que cambiar ciertos parámetros porque habrían otras partes del cuerpo involucradas.

7. Referencias

[1] https://en.wikipedia.org/wiki/Motion_capture

[2] http://www.tof-cv.org

[3] S. Soutschek, J. Penne, J. Hornegger, and J. Kornhuber, 3-d gesture based scene navigation in medical imaging applications using time-of-flight cameras, CVPR Workshops, 2008.

[4] R. Urtasun and T. Darrell, Sparse probabilistic regression for activity independent human pose infer- ence, CVPR, 2008.

[5] T. Jaeggli, E. Koller-Meier, and L. V. Gool, Learning generative models for multi-activity body pose estimation, IJCV, vol. 83, no. 2, pp. 121-134, 2009.

[6] R. Kehl and L. Gool, Markerless tracking of complex human motions from multiple views, CVIU, 2006.

[7] J. Bandouch, F. Engstler, and M. Beetz, Accurate human motion capture using an ergonomics-based anthropometric human model, AMDO, 2008.

[8] V. Ganapathi, C. Plagemann, D. Koller, and S. Thrun, Real time motion capture using a single time-of-flight camera, CVPR, 2010.

[9] C. Plagemann, V. Ganapathi, and D. Koller, Real-time identification and localization of body parts from depth images, ICRA, 2010.

[10] Y. Sun, M. Bray, A. Thayananthan, B. Yuan, and P. Torr, Regression based human motion capture from voxel data, BMVC, 2006.

[11] Y. Zhu, B. Dariush, and K. Fujimura, Controlled human pose estimation from depth image streams, CVPR Workshops, 2008.

[12] G. Pons-Moll, A. Baak, T. Helten, M. Muller, H.-P. Seidel, and B. Rosenhahn, Multisensor-fusion for 3d full-body human motion capture, CVPR, pp. 1-8, 2010.

[13] R. Jensen, R. Paulsen, and R. Larsen, Analyzing gait using a timeof- flight camera, Scandinavian Conference on Image Analysis, pp. 21-30, 2009.

[14] S. Denman, V. Chandran, and S. Sridharan, An adaptive optical flow technique for person tracking systems, PRL, vol. 28, no. 10, pp. 1232-1239, 2007.

[15] R. Okada, Y. Shirai, and J. Miura, Tracking a person with 3-d motion by integrating optical flow and depth, FG, pp. 1-6, 2000.

[16] Microsoft® Kinect™ for Windows SDK beta programming guide beta 1 draft version 1.1. 2012.

[17] J. Shotton, A. W. Fitzgibbon, M. Cook, and T. Sharp, Real-time human pose recognition in parts from single depth images, CVPR, pp. 1297-1304, 2011.

[18] Sergio Escalera, Human Behavior Analysis from Depth Maps, AMDO, pp. 282-292, 2012.

[19] Miguel Reyes, Gabriel Domnguez, and Sergio Escalera, Feature Weighting in Dynamic Time Warping for Gesture Recognition in Depth Data, 1st IEEE Workshop on Consumer Depth Camera cámaras for Computer Vision, ICCV, 2011.

[20] L.A. Schwarz, A. Mkhitaryan, D. Mateus, N. Navab, Estimating human 3D pose from Time-of-Flight images based on geodesic distances and optical flow, FG, pp 700-706, 2011.

[21] C. Plagemann, V. Ganapathi, and D. Koller, Real-time identification and localization of body parts from depth images, ICRA, 2010.

[22] Antonio Hernndez-Vela, Nadezhda Zlateva, Alexander Marinov, Miguel Reyes, Petia Radeva, Dimo Di-mov, and Sergio Escalera, Human Limb Segmentation in Depth Maps based on Spatio-Temporal Graph Cuts Optimization, JAISE, 2012.

[23] Nobuyuki Otsu (1979). "A threshold selection method from gray-level histograms". IEEE Trans. Sys., Man., Cyber. 9 (1): 62–66.

[24] R.Lange: 3D Time-of-flight distance measurement with custom solid-state image sensors in CMOS/CCD-technology. PhD thesis, University of Siegen, 2000

[25] http://www.primesense.com/solutions/nite-middleware/

[26] http://es.wikipedia.org/wiki/Flujo_óptico

[27] Huston SJ, Krapp HG (2008). «Visuomotor Transformation in the Fly Gaze Stabilization System». PLoS Biology 6 (7): pp. E173.

[28] Dan Huttenlocher: Distance Transforms, Computer Vision. http://www.cs.cornell.edu/courses/cs664/2008sp/handouts/cs664-7-dtrans.pdf

[29] S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani p.121

8. Apéndice

8.1. Log de git

```
* commit 5742ff5e10b839ef5c4c0a07c7dd223623bfb850
| Author: slinzex
| Date:   Thu May 2 20:22:45 2013 +0200
|
|     before splitting and structuring
|
* commit 17e33495f44830a391f479b944642e348ea5d3e6
| Author: slinzex
| Date:   Wed Apr 24 22:50:13 2013 +0200
|
|       skipping untagged frames (missing groundtruth) ; testing done; gonna prepare
optflow testing
|
* commit 47747130c9cdba3a77674207dbe0d173773624f1
| Author: slinzex
| Date:   Tue Apr 23 20:39:30 2013 +0200
|
|     kinect globals moved to main and struct; polishing parameters before final test
|
* commit df69ec83407c6e6d30f0b7b089cdc8c0aabdb51d
| Author: slinzex
| Date:   Tue Apr 23 01:45:21 2013 +0200
|
|      global paths to files; cross comparison well implemented; comparators,  custom
sorts...
|
* commit a2622d2591f9f2be9df67abf099fab38797d79c0
| Author: slinzex
| Date:   Sun Apr 21 00:43:46 2013 +0200
|
|     segmenting clusters moreless well; threshold % points implemented!
|
* commit 44d3bff67b690284366eb7e1bdb4b89f319c8cec
| Author: slinzex
| Date:   Fri Apr 19 22:44:38 2013 +0200
|
|      paths changed to read from defines.h; getting center of pointcloud of geodesic
high points;
|
* commit fed8988fb371102cf6c40639c618e6dd749cc7f9
| Author: slinzex
| Date:   Fri Apr 12 20:56:39 2013 +0200
|
|     before determining error rate based on groundtruth
|
* commit e127685063d2d3beb1a74331f9ecbe7c48fb7013
| Author: slinzex
```

```
| Date:   Thu Apr 11 17:43:05 2013 +0200
|
|     labelling
|
* commit adaad9589f0ea644bdb0cb03bf7911e038840a67
| Author: slinzex
| Date:   Thu Apr 4 19:08:07 2013 +0200
|
|     solved point selection and printing its x,y,z
|
* commit 5e34955eaa8942124de9d11e49a795209da17c55
| Author: slinzex
| Date:   Thu Apr 4 18:06:29 2013 +0200
|
|     before making changes on mouse picking
|
* commit cd6fd04947ebb973f7e9c99e152e3f112a9ccde7
| Author: slinzex
| Date:   Mon Mar 25 18:41:35 2013 +0100
|
|     successful separation where big movement!
|
* commit a9da8501892f79beb651bb91917545bb44fa645e
| Author: slinzex
| Date:   Mon Mar 25 16:53:58 2013 +0100
|
|     playing with optical flow _ cropping..
|
* commit dde1d75ee7d247125f0d850d4e048f1002943792
| Author: slinzex
| Date:   Sat Mar 2 17:43:17 2013 +0100
|
|     hands detection with trajectory
|
* commit 4c5084616b5c1e96659f9cdd32041af3af3f6bab
| Author: slinzex
| Date:   Wed Feb 20 00:53:43 2013 +0100
|
|     selecting hands with geodesic value!
|
* commit 1b2f7ec5157aef35cde1e5b8af88ef46118ab707
| Author: slinzex
| Date:   Sat Feb 16 23:36:26 2013 +0100
|
|     order error corrected : sort problem
|
* commit fb16beb9dd3ef02730df4b2cd7c41e595ade679f
| Author: slinzex
| Date:   Fri Feb 15 20:58:14 2013 +0100
|
|     final mean vector, works for sequence of images
|
* commit 10e7332d63d5eb27acb7d9890e36891f4bae2f55
| Author: slinzex
| Date:   Wed Feb 6 03:03:43 2013 +0100
```

```
|
|     first vectors , matlab plots, code comments
|
* commit d4ff12482c804e462205ca2dcf20f8d4168daa2b
| Author: slinzex
| Date:   Mon Feb 4 02:15:11 2013 +0100
|
|     reading and processing .oni  or depth images (fusion)
|
* commit 07075e07ab46e054067516a5ce64b9d1ac778b1d
| Author: slinzex
| Date:   Sun Feb 3 02:56:29 2013 +0100
|
|     unified oni+image+kinect
|
* commit 0fdaed2403d1e3ca8bdc93ed4c064734fc58e461
| Author: slinzex
| Date:   Sat Feb 2 17:09:09 2013 +0100
|
|     preparing for kinect adaptation
|
| *   commit 0a9e482d0611f5e2af1bb023ed1f003ab4db6d9b
| |\  Merge: f98025f 4946d59
| | | Author: slinzex
| | | Date:   Fri Feb 1 14:22:23 2013 +0100
| | |
| | |     returned to unstructured project
| | |
| | *   commit 4946d59920a3416bbd22fdd0971b65651b7d5e58
| | |\  Merge: 036f8f7 408898f
| | | | Author: slinzex
| | | | Date:   Fri Feb 1 13:44:29 2013 +0100
| | | |
| | | |       Merge branch 'betaRelease' of https://bitbucket.org/vkron/geodesic into
betaRelease
| | | |
| | | |     Conflicts:
| | | |      xCodeBuild/Debug/main.app/Contents/MacOS/main
| | | |      xCodeBuild/UserDetectGeo.build/Debug/ZERO_CHECK.build/build-state.dat
| | | |      xCodeBuild/UserDetectGeo.build/Debug/main.build/Objects-normal/x86_64/
main.o
| | | |      xCodeBuild/UserDetectGeo.build/Debug/main.build/build-state.dat
| | | |      xCodeBuild/UserDetectGeo.build/Debug/main.build/main.dep
| | | |      xCodeBuild/UserDetectGeo.xcodeproj/project.xcworkspace/xcuserdata/
vito.xcuserdatad/UserInterfaceState.xcuserstate
| | | |
| | | * commit 408898f78cbcc9d9b74e5013b04d312d14bf915c
| | | | Author: slinzex
| | | | Date:   Fri Feb 1 03:34:38 2013 +0100
| | | |
| | | |     crash pcl
| | | |
| | | * commit 24ecc4230d7771200509eb872364c1c92f8aa75f
| | |_|/  Author: slinzex
|/| |   Date:   Fri Feb 1 03:15:35 2013 +0100
```

```
| | |
| | |       project structured
| | |
| | * commit 036f8f775da43152b8a2ef1ee3e2bc413b6ac607
| | | Author: slinzex
| | | Date:   Fri Feb 1 13:43:55 2013 +0100
| | |
| | |     returned to unstructured project
| | |
| * | commit f98025fe71460e603b2f942f903a71857357cbfc
| |/  Author: slinzex
| |   Date:   Fri Feb 1 14:16:55 2013 +0100
| |
| |       returned to unstructured project
| |
| * commit 866960fb3efce246cd86b9cb5fe1b0a508298d03
|/  Author: slinzex
|   Date:   Fri Feb 1 13:40:33 2013 +0100
|
|       returned to unstructured project
|
* commit 1c3d98cad0f529bb4545144c4cc5a4134399f5dc
| Author: slinzex
| Date:   Thu Jan 31 23:33:28 2013 +0100
|
|     before restructuring project
|
* commit fad974a81fb4ab715e6dc40b782cb64f35a0a0ec
| Author: slinzex
| Date:   Thu Jan 31 19:34:15 2013 +0100
|
|     finding center of distance map success!
|
* commit dea091b80fbbd4f036da16cd63a1c8fa8ac9fdd8
| Author: slinzex
| Date:   Fri Jan 11 14:25:44 2013 +0100
|
|     - geodesic center from distance map
|     - mean vector of mean flows
|
* commit b5262a0846eca14939e3a5c60cbbffc55514235d
| Author: slinzex
| Date:   Mon Jan 7 01:42:56 2013 +0100
|
|     mean optiflow after geodesic
|
* commit abec6938f3aa365a998681060e38dedd947f2ef7
| Author: slinzex
| Date:   Mon Jan 7 00:12:23 2013 +0100
|
|     calc mean flow inside optical_flow
|
* commit 11aaed28597f674f017f087c7efad32d59fd98ce
| Author: slinzex
| Date:   Sat Jan 5 17:24:45 2013 +0100
```

```
|
|     -sorted vector of geovalues filtering INFs
|
* commit 5eb4b6fcd733953ab197d62ff8f04efcf23f635a
| Author: slinzex
| Date:   Wed Jan 2 17:26:04 2013 +0100
|
|     -beta release prepared
|     -funtions commented
|
| * commit 8778c5c8f588b6c718245cd9b867dd7a5cfcb0ee
|/  Author: slinzex
|   Date:   Mon Jan 7 01:57:02 2013 +0100
|
|       screenshots and quick codes opencv
|
* commit e29224f34e95c2ddc9050ea159a97a1ce4ad9dbc
| Author: slinzex
| Date:   Wed Jan 2 15:00:15 2013 +0100
|
|     starting from depth images OK
|
* commit f2c3375ccf816b5c700541f75fca5175e0b70217
| Author: slinzex
| Date:   Wed Jan 2 13:23:42 2013 +0100
|
|     input depth image to pcloud solved +/- with unsigned char
|
* commit d5fbfc9f3954f23a47b5b539738e3954591ebd73
| Author: slinzex
| Date:   Mon Dec 31 16:10:23 2012 +0100
|
|     input depth image to pcloud problem!
|
* commit 6ca85df9cc021263700faaab631e25234d90b811
| Author: slinzex
| Date:   Mon Dec 24 12:42:40 2012 +0100
|
|     reinstalled pcl
|     incorpored image_grabber
|     incorpored depth images to pcloud with BUG
|
* commit 841b9225558302fdd3413cb66cc9c4b357b586ae
| Author: slinzex
| Date:   Sun Dec 23 00:57:21 2012 +0100
|
|     code cleaning
|
* commit 3e181d598cf223a2167409582e1d6f413aa2cbd7
| Author: slinzex
| Date:   Sun Dec 23 00:14:23 2012 +0100
|
|     imshow bug SOLVED by creating Mat zeros
|
* commit 508302eb74c19216d6eb0a31f1a316d61b7660a5
```

```
| Author: slinzex
| Date:   Sun Dec 23 00:12:21 2012 +0100
|
|     imshow bug? random pixel intensity values.
|
* commit fc905f9e1d96cbd71c6b9c5a8ac22669b9734af2
| Author: slinzex
| Date:   Fri Dec 21 13:51:29 2012 +0100
|
|     Solved voxel grid indexing by creating new cloud_out organized
|
| * commit 8bce3efbbd28c0f82675918145868c9928e1a1f1
|/  Author: slinzex
|   Date:   Fri Dec 21 01:24:06 2012 +0100
|
|       includes moved to main.h
|
* commit fb12d3da9cd454fd5e99ffaef64d9007dbf16b1d
| Author: slinzex
| Date:   Fri Dec 21 00:54:19 2012 +0100
|
|     try to separate (not done)
|
* commit fb7455efb78c49cee6fcf0ef36b3f32927cd9857
| Author: slinzex
| Date:   Fri Dec 21 00:00:38 2012 +0100
|
|     before removing prints that iterate and show
|
* commit 0772e053c9025dc18c4f22504000971117bc3302
| Author: slinzex
| Date:   Thu Dec 20 23:54:13 2012 +0100
|
|     fixed optical flow issue: abs of fxy
|
* commit 33b89764bfe15ccc4c2fabcebeee416f06b5eb6d
| Author: slinzex
| Date:   Thu Dec 20 15:38:22 2012 +0100
|
|     filtering code
|
* commit 2d9967b15c198b46fab185971b1262659462e006
| Author: slinzex
| Date:   Thu Dec 20 05:04:38 2012 +0100
|
|     -removing hands from pCloud.
|
* commit bb03b5739695cfd4bd0c00716ec441bb431f3666
| Author: slinzex
| Date:   Tue Dec 18 15:49:05 2012 +0100
|
|     - possible solution to bug with separated parts on some frames. If Else INF
|
* commit d6e80b55370b2658c1aa7100e54c74bede9ffe25
| Author: slinzex
```

```
| Date:   Mon Dec 17 03:37:36 2012 +0100
|
|     problems duplicate background…???
|
* commit bbb87b9e6b620064615525f82bdc82dca6d576b0
| Author: slinzex
| Date:   Sun Dec 16 21:25:41 2012 +0100
|
|     -matching pcloud with optical flow region
|     -method at()
|     -problem with erasing from filtered voxelgrid
|
* commit 8dbc2b6abf1517bfd336d352db4207bcbef78d00
| Author: slinzex
| Date:   Fri Dec 14 04:26:17 2012 +0100
|
|      - voxelgrid repaired. Fail caused by headIndex change for organized cloud . Now
is 68244 . Before was 0
|
* commit de1b31818714fdbf779a401a28c95d82a569313d
| Author: slinzex
| Date:   Fri Dec 14 02:00:21 2012 +0100
|
|     - comments for drawOpticalFlow
|     - pointing big optical flows!
|     - BUILD FOR XCODE!!!
|
* commit 757095f956b6684d98211416f8cc55e88ca7b180
| Author: slinzex
| Date:   Thu Dec 13 16:05:13 2012 +0100
|
|     indented
|
* commit 295d7311383771fda12fd68ae64045b2e9296e63
| Author: slinzex
| Date:   Mon Dec 10 02:51:58 2012 +0100
|
|     organized PointCloud
|     -saving PNG on backgound to disk
|
* commit 2f6ff3bd1cda16a6cdba3e3ae8ba592a8fb2fa22
| Author: slinzex
| Date:   Mon Dec 10 02:50:14 2012 +0100
|
|     -solved problem with visualizing - zoom out…  solved by : resetCamera()
|     -visualizing and saving screenshots moreless good.
|
* commit 0a64b18aff7da7adb0e6e67eb85823773aee8b4f
| Author: slinzex
| Date:   Sun Dec 9 22:28:38 2012 +0100
|
|     visualizing RGB pointcloud and opencv RGB
|
* commit 864c800b234e2eb79976f9b579adf6f200e7d202
| Author: slinzex
```

```
| Date:   Sun Dec 9 19:22:53 2012 +0100
|
|     removed some comments
|
* commit 7754dcd13971aaaca734c6d93d6c9244d6012eeb
| Author: slinzex
| Date:   Sun Dec 9 15:11:04 2012 +0100
|
|     right button closes.
|     cleaned prints
|
* commit d5e4807acd11321cf47123a4376937fb9346a3e4
| Author: slinzex
| Date:   Sun Dec 9 15:06:09 2012 +0100
|
|     -solved BUG overcoloring. all structures are created locally.
|
* commit ae729fa9fe9b02f6a5d0d979f4939631b6829fc8
| Author: slinzex
| Date:   Sun Dec 9 14:38:20 2012 +0100
|
|     -solved annoying MAX warning
|     -solved cleanUpDijkstra which caused segmentation fault  (nodes*nodes) -> nodes
|     -detected BUG with overcolorring.
|
* commit 47e0ebddd85ed60f040784275fd0497a4c8f5b99
| Author: slinzex
| Date:   Sat Dec 8 20:21:11 2012 +0100
|
|      -solved BUG with overcolorring. Problem was with arrays in C++ and memory acces.
Solved by converting all to std::vector
|       -solved PCL:visualizer issue. viewer->spin() has loop. Don't use with spinOnce
method. One or other.
|     -added mouse control to close visualizer runned from terminal
|
| * commit 0db42d315679f918d5480e90f59fc68ceed46410
| | Author: slinzex
| | Date:   Sat Dec 8 20:09:17 2012 +0100
| |
| |     solved but with overcolorring.
| |       Problem was with arrays in C++ and memory acces. Solved by converting all to
std::vector
| |
| * commit 8a82a97f5a44029c6a947a3551cb5270d32248ba
|/  Author: slinzex
|   Date:   Thu Dec 6 01:56:33 2012 +0100
|
|       testing branching
|
* commit a93d720b68ecfb50e30c2a7c9d43ac5d3e92166a
| Author: slinzex
| Date:   Thu Dec 6 01:52:14 2012 +0100
|
|     main successfully copied after run.sh
|
```

```
* commit f17a963fe0c3510c5c1b003315b9f66224a3f0d9
| Author: slinzex
| Date:   Thu Dec 6 01:47:37 2012 +0100
|
|     First make on GIT
|
* commit 3b3783c0bd0e0138d643d9b74cc2e30afc4a45da
  Author: slinzex
  Date:   Thu Dec 6 01:25:36 2012 +0100

      Moving project to GIT
```

8.2. Configuración CMakeLists.txt usada en este proyecto

```
cmake_minimum_required(VERSION 2.8 FATAL_ERROR)
#Based on the Solution from Nicolas Burrus
project(UserDetectGeo)

## Add current Source Dir
#INCLUDE_DIRECTORIES(${CMAKE_CURRENT_SOURCE_DIR})
## Add folder /include
#INCLUDE_DIRECTORIES(${CMAKE_CURRENT_SOURCE_DIR}/include)
###############################################################################
# Find OpenNI
#
# This sets the following variables:
# OPENNI_FOUND - True if OPENNI was found.
# OPENNI_INCLUDE_DIRS - Directories containing the OPENNI include files.
# OPENNI_LIBRARIES - Libraries needed to use OPENNI.
# OPENNI_DEFINITIONS - Compiler flags for OPENNI.
find_package(PkgConfig)
# if(${CMAKE_VERSION} VERSION_LESS 2.8.2)
#   pkg_check_modules(PC_OPENNI openni-dev)
# else()
#   pkg_check_modules(PC_OPENNI QUIET openni-dev)
# endif()

set(OPENNI_DEFINITIONS ${PC_OPENNI_CFLAGS_OTHER})

#add a hint so that it can find it without the pkg-config
find_path(OPENNI_INCLUDE_DIR XnStatus.h
          HINTS ${NESTK_ROOT_DIRS_HINTS} ${PC_OPENNI_INCLUDEDIR} ${PC_OPENNI_INCLUDE_DIRS} /usr/include/
openni /usr/include/ni
          PATHS "$ENV{PROGRAMFILES}/OpenNI/Include" "$ENV{PROGRAMW6432}/OpenNI/Include"
          PATH_SUFFIXES openni)
#add a hint so that it can find it without the pkg-config
find_library(OPENNI_LIBRARY
             NAMES OpenNI64 OpenNI
             HINTS ${NESTK_ROOT_DIRS_HINTS} ${PC_OPENNI_LIBDIR} ${PC_OPENNI_LIBRARY_DIRS} /usr/lib
             PATHS "$ENV{PROGRAMFILES}/OpenNI/Lib${OPENNI_SUFFIX}" "$ENV{PROGRAMW6432}/OpenNI/Lib$
{OPENNI_SUFFIX}"
             PATH_SUFFIXES lib
)
find_library(NITE_LIBRARY
             NAMES XnVNite XnVNITE_1_3_1 XnVNITE_1_4_0 XnVNite_1_4_2 XnVNite_1_5_2
             HINTS ${NESTK_ROOT_DIRS_HINTS} ${PC_OPENNI_LIBDIR} ${PC_OPENNI_LIBRARY_DIRS} /usr/lib
             PATHS "$ENV{PROGRAMFILES}/PrimeSense/NITE/Lib${OPENNI_SUFFIX}" "$ENV{PROGRAMW6432}/PrimeSense/
NITE/Lib${OPENNI_SUFFIX}"
             PATH_SUFFIXES lib
)

find_path(NITE_INCLUDE_DIR XnVSessionManager.h
```

```
          HINTS ${NESTK_ROOT_DIRS_HINTS} ${PC_OPENNI_INCLUDEDIR} ${PC_OPENNI_INCLUDE_DIRS} /usr/include/
openni /usr/include/nite
          PATHS "$ENV{PROGRAMFILES}/PrimeSense/NITE/Include" "$ENV{PROGRAMW6432}/PrimeSense/NITE/Include"
          PATH_SUFFIXES openni)

set(OPENNI_INCLUDE_DIRS ${OPENNI_INCLUDE_DIR} ${NITE_INCLUDE_DIR})
if(APPLE)
  set(OPENNI_LIBRARIES ${OPENNI_LIBRARY} ${NITE_LIBRARY} usb)
else()
  set(OPENNI_LIBRARIES ${OPENNI_LIBRARY} ${NITE_LIBRARY})
endif()

include(FindPackageHandleStandardArgs)
find_package_handle_standard_args(OpenNI DEFAULT_MSG
    OPENNI_LIBRARY OPENNI_INCLUDE_DIR)

mark_as_advanced(OPENNI_LIBRARY OPENNI_INCLUDE_DIR)
if(OPENNI_FOUND)
  include_directories(${OPENNI_INCLUDE_DIRS})
  link_directories(${OPENNI_LIBRARY})
  add_definitions(${OPENNI_DEFINITIONS})
  message(STATUS "OpenNI found (include: ${OPENNI_INCLUDE_DIR}, lib: ${OPENNI_LIBRARY})")
endif(OPENNI_FOUND)

########################################## OPEN GL
# INCLUDE(FindOpenGL REQUIRED)
# INCLUDE(FindGLUT REQUIRED)
# INCLUDE_DIRECTORIES(${OPENGL_INCLUDE_DIR})
# INCLUDE_DIRECTORIES(${GLUT_INCLUDE_DIR})

# message(${OPENGL_LIBRARIES})
# message(${GLUT_INCLUDE_DIR})
# message(${GLUT_LIBRARIES})


########################################## OPEN CV
# set(OpenCV_DIR /opt/local/lib/cmake)
find_package( OpenCV REQUIRED )

########################################## PCL

find_package(PCL 1.3 REQUIRED)
include_directories(${PCL_INCLUDE_DIRS})
link_directories(${PCL_LIBRARY_DIRS})
add_definitions(${PCL_DEFINITIONS})


add_executable (main MACOSX_BUNDLE main.cpp)
target_link_libraries (main ${OPENNI_LIBRARY} ${OpenCV_LIBS} ${PCL_LIBRARIES})
```