FACULTAD DE

INFORMÁTICA DE BARCELONA

TRABAJO DE FINAL DE GRADO

DETECCIÓN ROBUSTA DE LA ORIENTACIÓN DE LA CABEZA DEL USUARIO A PARTIR DE UNA CÁMARA RGBZ

Autor:
Martín Cristóbal Balasch
Especialidad Computación

Director:
Carlos Andújar Gran
Departamento de Ciencias de la Computación

Resumen

La localización de caras es una característica ampliamente utilizada actualmente en diferentes productos software. Además, con la aparición de sensores RGBZ (como la Kinect o la RealSense) se ha añadido la capacidad no sólo detectar dónde hay una cabeza, si no de obtener información tridimensional sobre la misma.

En este proyecto se diseña, desarrolla y analiza un software que permita obtener, mediante el uso de las cámaras RGBZ anteriormente mencionadas, la orientación 3D de la cabeza del usuario que esté delante de ellas, es decir, los ángulos que determinan hacia qué dirección está mirando el usuario. Para ello se ha diseñado un algoritmo basado en el método *Iterative Closest Point*, de forma que por cada imagen capturada por la cámara se detecte qué ángulos presenta la cabeza.

También se ha desarrollado una plataforma externa utilizando un servomotor y un microcontrolador Arduino, permitiendo realizar pruebas de los diferentes parámetros del algoritmo para validar sus resultados, mediante una plataforma giratoria sobre la que se puede orientar con precisión una reproducción a escala de una cabeza 3D.

Resum

La localització de cares és una característica ámpliament utilitzada en diferents productes software actualment. A més, amb l'aparició de sensors RGBZ (com la Kinect o la RealSense) s'ha afegit la capacitat de, no només detectar a on hi ha una cara, si no d'obtenir la informació tridimensional d'aquesta.

En aquest projecte es disenya, desenvolupa i s'analitza un software que permeti obtenir, mitjançant l'ús de les cámeres RGBZ anteriorment nombrades, la orientació del cap de l'usuari que es trobi davant d'elles, és a dir, dels angles que defineixen cap a quina direcció está mirant l'usuari. Per aconseguir-ho s'ha dissenyat un algoritme basat en el mètode *Iterative Closest Point*, de manera que per cada imatge capturada per la cámara es detecti quins angles presenta el cap.

També s'ha desenvolupat una plataforma externa utilitzant un motor i un microcontrolador Arduino, a on es poden realitzar proves dels diferents parámetres de l'algoritme per validar els resultats mitjançant una plataforma giratoria sobre la qual s'ha col·locat una reproducció a escala d'un cap en tres dimensions que es pot orientar amb precisió.

Abstract

Face localization has become a hugely demanded feature in many different sofware products. In addition, with the appearence of RGBZ sensors (such as the Kinect and the RealSense) the capacity of not only detecting where the face is located but also obtaining the 3D orientation of the face has been added.

In this project we aim to design, develop and test a software able to, using the RGBZ sensors, detect the pose of the head of a user in front of the camera, that is, extract the three angles that define the direction of the head. To do that, we developed an algorithm based on the *Iterative Closest Point* family. For each image provided by the camera, the angles are detected.

An external platform was also developed using a servomotor and an Arduino microcontroller, able to perform tests of the different parameters of the algorithm to validate the results using a rotating base that can turn precisely a reproduction of a real-size 3D printed head.

Agradecimientos

En primer lugar querría agradecer al profesor Carlos Andújar Gran por ofrecerme la oportunidad de realizar este proyecto. Fue el profesor Carlos Andújar el que me propuso realizar este trabajo cuando le pregunté sobre realizar un Trabajo de Final de Grado y además propuso el algoritmo que se presenta en este documento. También agradecerle por echarme una mano en los momentos de dificultad. Sin su atención y su dedicación nada habría sido posible.
También quiero agradecer a mis familiares y a mis amigos cercanos por apoyarme en todo momento y ayudarme a mantener la motivación y las energías.
Muchísimas gracias.

Índice

1. Introducción

1.1. Motivaciones

Las aplicaciones de computación gráfica cada vez tienen más importancia en muchísimos ámbitos. Desde la informática médica hasta los videojuegos, pasando por multitud de industrias que buscan aprovechar su potencial. Además, la aparición de cámaras capaces de captar profundidad ha abierto un nuevo abanico de posibilidades, haciendo accesible para el gran público hardware que de otra manera sería prohibitivo. En este proyecto buscamos diseñar, desarrollar y comparar diversos métodos de seguimiento facial utilizando diversas técnicas y cámaras. El objetivo es comprobar si es posible igualar o incluso mejorar el seguimiento facial que ya se ofrece en el mercado aplicando ciertas restricciones sobre la posición de la cabeza. Esto sería útil en muchas aplicaciones donde se tiene ya cierto conocimiento a priori sobre la posición del usuario con respecto a la cámara. La aplicación utilizaría diversas cámaras con captación de profundidad para poder comparar nuestros resultados con los diversos algoritmos que ya se ofrecen actualmente en el mercado. Este documento busca dar al lector una descripción detallada del proyecto. Se encuentra dividido en diversas secciones aportando información sobre el contexto, el alcance y la planificación tanto temporal como económica.

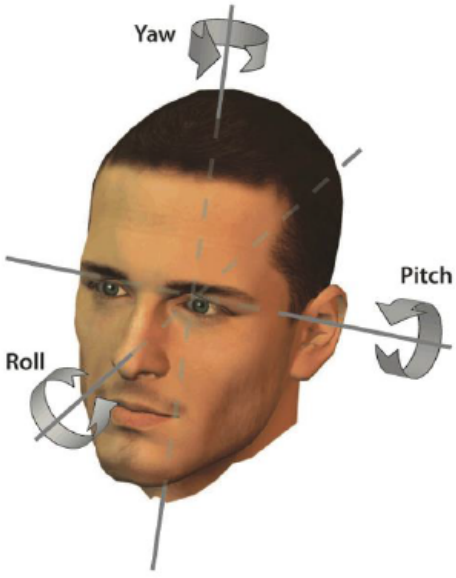


Figura 1.1: Orientación de una cabeza definida por 3 ángulos

1.2. Contextualización

En esta sección se explicará el contexto del proyecto. En la primera mitad se mostrará una breve descripción del área de interés del proyecto y los actores afectados, mientras que en la segunda mitad se da información sobre el estado del arte en las áreas que se explorarán durante el desarrollo del proyecto.

1.2.1. Áreas de interés

En el proyecto se observan claramente dos áreas de interés. La primera consiste en la detección automática de la orientación de la cara, que puede ser interesante en diversas aplicaciones informáticas, como por ejemplo en aplicaciones interactivas o juegos, o incluso para aplicaciones pensadas para usuarios con movilidad reducida, los cuales podrían realizar acciones simples con los movimientos de la cabeza. La segunda consiste en los aspectos técnicos del proyecto como el desarrollo del visor o la implementación de los algoritmos.

1.2.2. Actores implicados

- **Director del proyecto** El director del proyecto es Carlos Andújar, profesor del departamento de Ciencias de la Computación de la Universidad Politécnica de Catalunya. Su trabajo consistirá en guiar al desarrollador del proyecto en los momentos de dificultad.
- **Desarrollador del proyecto** Este proyecto dispone tan sólo de un desarrollador (el autor de este trabajo). Trabajará en la planificación del proyecto, la documentación, el código y las pruebas. También se dedicará a la investigación.
- **Usuarios** Nuestro objetivo es que el proyecto pueda ser utilizado por los usuarios de la forma que encuentren necesaria. Es posible que el proyecto pueda ser un componente que se pueda añadir a otros proyectos para así aprovechar la capacidad de detectar la orientación de la cabeza.

1.3. Objetivos

El objetivo principal del proyecto consiste no sólo en diseñar y desarrollar, si no en comparar los resultados de diferentes métodos de seguimiento facial (principalmente buscando la orientación de la cabeza del usuario). El proyecto busca obtener un reconocimiento más robusto de la orientación del

que ofrecen las cámaras 3D comerciales, ya que se sospecha que éstas no aprovechan o directamente no utilizan la información de profundidad que ofrece la cámara. Un objetivo secundario del proyecto consiste en desarrollar un juego de pruebas fiable sobre el que comparar todos los resultados, para así tener una seguridad rigurosa sobre la bondad de los resultados tanto de las cámaras comerciales como del propio algoritmo a desarrollar.

1.4. Estado del arte

El objetivo de este apartado consiste en dar información sobre el estado del arte de las diferentes partes del proyecto, buscando información sobre soluciones actuales ya disponibles al público.

1.4.1. Detección de la orientación de la cabeza

La detección robusta de la posición de la cabeza es un tema que, desde la aparición de las cámaras portátiles ha sido extensivamente estudiado. Aun y así, la mayoría de la literatura contempla la detección de la posición de la cara del usuario pero no tanto la detección del propio ángulo de la cabeza. Existe un estudio de la universidad de Ulm, Alemania, en el que se muestran experimentos para la detección de los ángulos para una aplicación de interacción humano-máquina [1], utilizando muestras de usuarios con imágenes de entre -90º a 90º. El hecho de que no utilicen la nube de puntos implica una diferencia con respecto al proyecto aquí presentado.
También es interesante el estudio realizado en la universidad de Hong Kong, que precisamente analiza el hecho de realizar un seguimiento de la orientación 3D de la cabeza de un usuario mediante el uso de cámaras RGBD [2]. La diferencia principal entre el estudio y este proyecto consiste en que, en el estudio realizado por la universidad de Hong Kong, se utilizaba el modelo tridimensional de una cara como nube de puntos inicial para realizar la detección de la orientación. Como se podrá comprobar más adelante en este documento, nuestro proyecto realiza un método ligeramente distinto para determinar la orientación de la cabeza del usuario, aún y utilizando variantes del mismo algoritmo.

1.4.2. Visores 3D

Actualmente la oferta de visores 3D en el mercador es muy amplia. Podriamos definir como visor 3D cualquier aplicación que muestre contenido en 3 dimensiones. Podemos comentar algunos ejemplos significativos:

- **MeshLab** *MeshLab* es un software de procesado de mallas 3D muy conocido en el ámbito técnico. Incorpora un visualizador avanzado que ofrece tanto herramientas de visualización 3D como herramientas para la modificación de la propia geometría. Se trata de una aplicación que se sale del ámbito del visualizador 3D requerido para el proyecto ya que incorpora muchísimas características innecesarias para nosotros.

- **OpenGL** *OpenGL* se introdujo en 1992 y se ha convertido en la plataforma más utilizada y más soportada para el desarrollo de aplicaciones con gráficos bidimensionales y tridimensionales. Se han liberado diferentes versiones de *OpenGL*, siendo la más reciente la versión 4.5 [3].

- **Direct3D** *Direct3D* es un *framework* de desarrollo de gráficos 3D creado por *Microsoft*. Se puede utilizar para dibujar primitivas tridimensionales o para realizar operaciones paralelas en la GPU [4]. Es un sistema utilizado en multitud de aplicaciones *Windows* y en videojuegos creados para la plataforma *Games For Windows*.

1.4.3. Algoritmos de registro para nubes de puntos

Existen muchos algoritmos que se basan en las nubes de puntos. De hecho, existe la librería Point Cloud Library, que se dedica precisamente a proporcionar algoritmos destinados al análisis de nubes de puntos. Entre ellos cabe destacar el algoritmo de Iterative Closest Point, que se trata de un algoritmo utilizado para alinear dos nubes de puntos.
El algoritmo *ICP* se ha convertido en el método dominante para el registro de modelos tridimensionales. Se basa en el uso de la geometría pero en algunas ocasiones se puede utilizar el color de los puntos. El ICP inicia con dos nubes de puntos y una aproximación inicial sobre la transformación relativa de una a otra. A partir de ahí, se va ajustando iterativamente la transformación escogiendo repetidamente pares de puntos correspondientes buscando minimizar alguna medida de error [5]. Este algoritmo ha sido extensivamente estudiado y supondrá la base sobre la cual se construirá el proyecto. Al tratarse de un algoritmo tan examinado existen multitud de variaciones, combinaciones de parámetros y extensiones posibles.

1.5. Alcance y metodología

En esta sección se busca dar una descripción clara del alcance del proyecto. Primero, se definirá el problema principal que el proyecto busca solventar. A continuación, se explicará el alcance del proyecto además de todos los po-

sibles obstáculos que pueden aparecer durante su desarrollo. Finalmente se definirá la metodología que se va a utilizar durante el proyecto.

1.5.1. Alcance

Para poder llevar a cabo el proyecto, se necesita un conjunto de programas capaces de hacer un seguimiento de la orientación de la cabeza. Para ello, se desarrollaron diversas versiones utilizando las herramientas de desarrollo de las diferentes cámaras 3D del mercado y otras utilizando los algoritmos que queremos comparar. Para poder visualizar de forma rápida los resultados, será necesario desarrollar un visor 3D sobre el que mostrar las mallas a tiempo real juntamente con el ángulo que el algoritmo está calculando. No se busca realizar un visor 3D con capacidades de renderizado avanzadas. Se requerirán diversos algoritmos para comparar los resultados. La mayoría de ellos se basarán en la misma familia de algoritmos y la diferencia básica será la selección de los diversos frames obtenidos por la cámara a la hora de calcular la orientación. Finalmente se necesitan métodos para obtener la malla 3D de la cara del usuario. Todas las cámaras con profundidad que se utilizarán ya incorporan herramientas para obtener la nube de puntos que la cámara puede detectar. Se propone utilizar tanto una Kinect como una Intel RealSense como cámaras iniciales. No se descarta añadir otras en un futuro.

1.5.2. Metodología

En las siguientes líneas se describirá la planificación que se seguirá a la hora de desarrollar en proyecto. Primeramente, queríamos asegurar que el proyecto era factible. Para ello se desarrolló una primera prueba que consistía en utilizar el algoritmo básico que se utilizaría durante el desarrollo del proyecto sobre una nube de puntos obtenida por la cámara RealSense y comprobar si el algoritmo daba resultado. La prueba fue satisfactoria. El desarrollo del proyecto se inició creando una versión básica del visor con capacidad de detección de la orientación de la cara del usuario. El programa debe ser capaz de visualizar la nube de puntos o la malla obtenida por la cámara 3D y de dar una primera aproximación de la orientación. Seguidamente, se estudiaron los diferentes parámetros del algoritmo y se empezaron a desarrollar los juego de pruebas sobre el cual se compararán las diferentes versiones. Una vez con la versión básica del programa, se añadieron mejoras en un proceso iterativo buscando siempre mejorar o bien la precisión de la orientación o bien la velocidad de cómputo del algoritmo.

1.5.3. Herramientas

El desarrollo del programa principal del proyecto utilizó Qt para la creación de la interfaz de usuario y OpenGL para la visualización de los objetos 3D. Los dos son open-source y multiplataforma por lo que no habrá problemas utilizando el programa en los sistemas operativos más utilizados (Windows, Linux y Mac). Se utilizó también la librería *Point Cloud Library* (PCL) sobre la cual se desarrollaron los algoritmos para la detección de la orientación. El lenguaje principal en el que se desarrollaró el proyecto fué C++. Se utilizó Git para llevar control sobre el desarrollo del proyecto y sobre los cambios realizados en el código.

1.5.4. Validación de resultados

Para comprobar que los métodos de detección son robustos y fiables necesitaremos un juego de pruebas sobre el que comparar los resultados. Existen diferentes opciones a la hora de diseñar un método para obtener datos que se puedan utilizar de *Ground Truth* para los diferentes métodos que se utilizarán. Entre ellos, existe la opción de pedir a un usuario que se sitúe frente a la cámara y realice movimientos con la cabeza, pero fue descartada a causa de la incomodidad que sufre el usuario y por la difícil reproducción de los resultados. Como alternativa, se ha pensado en utilizar un motor paso a paso para girar una reproducción a escala 1:1 de una cabeza humana, de plástico o goma, sobre una mesa giratoria. Así, la cámara escanearía la cabeza mientras, controlando el motor, asignamos ángulos de giro. Este método nos permitiría saber en todo momento el ángulo actual de la cabeza y además se obtendría la nube de puntos de cada instante, lo que permitiría reutilizar la misma secuencia con nuestros algoritmos o incluso repitiendo el experimento en las mismas condiciones que el juego de pruebas para comparar la bondad de la detección de la orientación de la cabeza. También sería interesante realizar pruebas con sujetos reales. Es muy simple ya que el único requisito es que el usuario se posicione frente a la cámara y gire la cabeza en diferentes direcciones.

1.6. Obstáculos y riesgos

Aunque los objetivos y el alcance del proyecto se encuentran bien definidos, probable que durante el desarrollo del mismo aparezcan problemas e imprevistos. En este apartado hemos previsto los posibles problemas que pueden aparecer durante la ejecución del proyecto:

1.6.1. Visualizador

Es muy poco probable que aparezcan problemas en el desarrollo del visor 3D, ya que existen muchos visores actualmente en el mercado (muchos de ellos open-source) y en el caso de nuestro proyecto no tiene requerimentos especiales al tratarse de una herramienta interna y de comprobación.

1.6.2. Algoritmos

Los principales problemas pueden aparecer a la hora de desarrollar los algoritmos de detección. Uno de los principales problemas es el tiempo de cómputo, ya que es muy posible que el algoritmo no pueda ejecutarse en tiempo real y sea necesario o bien ajustar parámetros o bien buscar un método de reducción para eliminar puntos innecesarios de la nube de puntos obtenida por la cámara 3D. Además, puede aparecer un problema de desfase en función de cómo se desarrolle el algoritmo, ya que la acumulación de pequeños errores puede ir incrementando el desfase del ángulo detectado con el ángulo real.

2. Planificación del proyecto

2.1. Planificación temporal

Esta sección incluye la información referente a la planificación temporal del proyecto. El objetivo consiste en describir las tareas que se realizarán durante el desarrollo, ofreciendo un plan de acción que servirá para llevar a cabo el proyecto en el lapso de tiempo estimado inicialmente. Cabe destacar que el plan de acción puede verse afectado durante el desarrollo del proyecto a causa de imprevistos o problemas. La duración aproximada del proyecto es de 5 meses, empezando el día 2 de febrero y con una fecha límite del 27 de junio.

2.1.1. Descripción de las tareas

En esta sección se describirán las diferentes tareas que se llevarán a cabo durante el desarrollo del proyecto. Las tareas que se realizarán durante el proyecto serán las siguientes:

1. Se realizará un test de viabilidad para asegurar que el proyecto es realizable.

2. Si el test anterior resulta satisfactorio, se planificará el trabajo a realizar durante las siguientes semanas.

3. Se preparará el entorno de desarrollo y el sistema sobre el cual se realizarán las pruebas.

4. Se desarrollarán los programas y algoritmos necesarios para llevar a cabo el proyecto.

5. Se realizarán los estudios y análisis necesarios para sacar conclusiones sobre los resultados obtenidos

2.1.2. Test de viabilidad

La primera tarea que se realizó en el proyecto fue un test de viabilidad para comprobar que el proyecto era factible de realizar. Consistió en el desarrollo de un pequeño programa que se encargaba de leer la información de profundidad recibida desde una cámara con sensor 3D, en concreto la cámara *Intel RealSense*, utilizando la librería de análisis de nubes de puntos *Point Cloud Library*, para después analizar diferentes conjuntos de puntos obtenidos en tiempos distintos y comprobar si era posible obtener una transformación

geométrica que alineara las dos nubes. Las pruebas fueron satisfactorias por lo que fue posible seguir adelante con el proyecto.

2.1.3. Planificación

La tarea de planificación del proyecto se encuentra cubierta durante el desarrollo del curso de Gestión de Proyectos (GEP). Se puede dividir en diferentes etapas que encajan con las diferentes entregas que se realizan durante el curso:

1. Definición del alcance del proyecto
2. Definición y descripción de la planificación temporal.
3. Descripción de los costes económicos y análisis de sostenibilidad.
4. Estado del arte

Esta tarea es imprescindible ya que tiene como objetivo planificar el orden y el tiempo que se le dedicará a cada una de las posteriores tareas, además de utilizarse para definir posibles alternativas ante problemas que puedan aparecer durante el desarrollo.

2.1.4. Preparación del sistema

Antes de adentrarnos en el desarrollo del proyecto, es necesaria una etapa de preparación. La preparación del sistema incluye la instalación y configuración de las herramientras de desarrollo necesarias para llevar a cabo del proyecto. En primer lugar, el proyecto será desarrollado utilizando *Visual Studio 2013*. Además, requerirá de la plataforma *Qt* para la creación de la interfaz de usuario del programa, además de los controladores de las diferentes cámaras 3D como *Kinect* y *Intel RealSense*. El proyecto será desarrollado de forma íntegra en *Windows* (ya que el soporte de los drivers de las cámaras 3D es mucho menos problemático en esta plataforma), por lo que sólo será necesario disponer de un sistema operativo. Una vez los diferentes elementos se encuentran configurados, es posible empezar a desarrollar el proyecto como tal.

2.1.5. Desarrollo

En esta tarea se incluye la mayor parte del peso de este proyecto. Incorpora toda la implementación y comprobación de los diferentes algoritmos que se utilizarán para el análisis de los movimientos de la cabeza del usuario. El

desarrollo se puede dividir en las siguientes etapas:

Desarrollo del visor

El visor será el método de interacción entre el usuario y el programa. Será la base sobre la cual se irán implementando los algoritmos y sobre la que trabajarán las diferentes cámaras. Se desarrollará una aplicación *Qt* que se encargará de mostrar la información obtenida por las diferentes cámaras 3D. Además, también será capaz de realizar las pruebas de precisión necesarias pudiendo cargar juegos de prueba desde disco.

Desarrollo de los juegos de prueba

Este apartado es importante ya que incluye el desarrollo del *Ground Truth* contra el cual realizaremos todos nuestros experimentos. Los juegos de prueba se desarrollarán de la forma más robusta posible, utilizando un motor paso a paso conectado a una placa programable, pudiendo así generar ciertos movimientos de algún obtego (como por ejemplo la cabeza de plástico que se puede observar en la figura) a medida que el controlador guarda el ángulo actual del motor. Esto significa que será necesario, primero, construir el sistema electrónico para controlar el motor y, a continuación, desarrollar un método para poder generar, cargar y guardar secuencias de movimiento para los juegos de prueba.

Figura 2.1: Cabeza de plástico impresa en 3D

Desarrollo de los algoritmos

Los algoritmos se desarrollarán basándose en el esquema del *Iterative Closest Point*. Se implementarán diversas versiones de éste, dedicando cierto tiempo del desarrollo a experimentos para encontrar la mejor combinación de parámetros, y se probará su bondad contra los diferentes juegos de prueba. Una vez implementados, serán accesibles desde el visor pudiendo así comparar resultados de forma cómoda. Cabe destacar que existirá un módulo

encargado de transformar los formatos de las salidas de las diferentes cámaras 3D a una entrada válida para los algoritmos, ya que no todas las cámaras ofrecen las mismas especificaciones.
Las dependencias de estas etapas son bastante lógicas, aún y así se puede observar que hay algunas tareas que se pueden realizar en paralelo. La etapa de desarrollo del visor será la primera en ser realizada, de forma más o menos paralela al desarrollo electrónico del mecanismo para las pruebas. Seguidamente, se desarrollarán los diferentes algoritmos necesarios, de forma más o menos lineal. Además, se irá documentando el proyecto a medida que éste es realizado, ya que así la documentación será menos propensa a errores y olvidos.

2.1.6. Comparativa y análisis

Una vez los algoritmos han sido implementados, se compararán las diferentes versiones contra las versiones ya incluidas en el *SDK*, utilizando los juegos de prueba desarrollados anteriormente. Se requerirá un tiempo para recoger los datos y hacer el análisis. Este apartado incluye la tarea final del proyecto y la parte dónde se obtienen y documentan las conclusiones del mismo.

2.1.7. Duración de las tareas

En la tabla 1 se resume de forma aproximada la cantidad de tiempo dedicado a las diferentes tareas anteriormente nombradas.

Tarea	**Tiempo (horas)**
Test de viabilidad	20
Planificación	80
Preparación del sistema	7
Desarrollo del visor	80
Desarrollo de los juegos de prueba	80
Desarrollo de los algoritmos	150
Comparación y análisis	40
Total	457

Tabla 1: Distribución del tiempo por cada tarea

2.1.8. Alternativas

Es más que probable que, durante la realización del proyecto, aparezcan imprevistos que dificulten el desarrollo del plan inicial. Al utilizar una metodología iterativa, es posible que alguna iteración lleve más tiempo que el planificado, pero ya que las tareas se encuentran subdivididas será posible cercionarse a tiempo de los problemas y solucionarlos. Además, en el caso concreto de la adquisición de los motores, al tratarse de maquinaria muy propensa a estropearse, se ha decidido comprar más de un motor para tener repuestos en caso de que algo ocurriera.

2.2. Recursos

Podemos dividir los recursos en recursos de software y recursos de hardware. A continuación se incluye una lista con los recursos necesarios y las tareas en las que serán utilizados:
Recursos software:

- Windows 10: todas las tareas
- Qt: programación del visor
- Visual Studio: todo el desarrollo
- Controladores cámaras 3D: todo el desarrollo. Comparativa y pruebas

Recursos hardware:

- PC(Intel Core i/ 4790@3.60 GHz, 8 GB RAM DDR3, nVidia GeForce GTX 780Ti): todas las tareas
- Portátil Asus UX303-LA: todas las tareas
- Kinect 2, Intel RealSense SR300: desarrollo de la aplicación, desarrollo de los juegos de prueba, comparativa y pruebas.
- Motor: desarrollo de los juegos de prueba
- Cabeza de plástico: desarrollo de los juegos de prueba

2.3. Diagrama de Gantt

El siguiente diagrama de Gantt representa de forma gráfica la planificación explicada en este apartado.

	Nombre	Duración	Inicio	Fin	Predecesoras
2	**Planificación del proyecto**	1 mes	**17/02/2016**	**01/04/2016**	**1**
3	Definición del alcance	1 semana	17/02/2016	25/02/2016	
4	Planificación temporal	1 semana	26/02/2016	07/03/2016	3
5	Costes económicos y sostenibilidad	1 semana	08/03/2016	14/03/2016	4
6	Estado del arte	1 semana	15/03/2016	23/03/2016	5
7	Presentacion	1 semana	24/03/2016	01/04/2016	6
8	Preparación del sistema	1 semana	04/04/2016	08/04/2016	2
9	**Desarrollo**	2 meses	**11/04/2016**	**17/06/2016**	**8**
10	Desarrollo del visor	2 semanas	11/04/2016	29/04/2016	8
11	Desarrollo de los juegos de prueba	2 semanas	11/04/2016	29/04/2016	8
12	Desarrollo de los algoritmos	1 mes	02/05/2016	17/06/2016	10
13	Comparación y análisis	1 semana	20/06/2016	01/07/2016	9

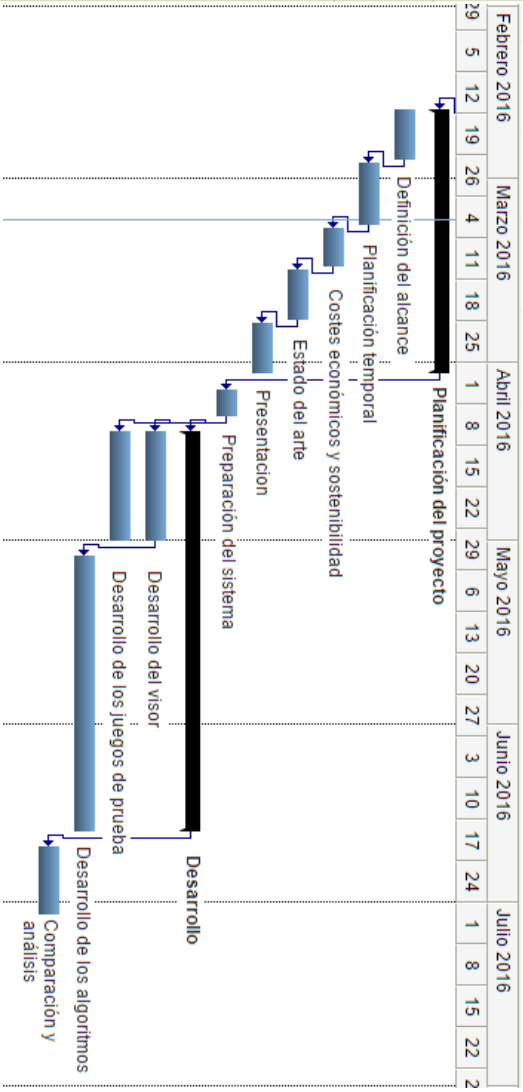


Figura 2.2: Diagrama de Gantt del proyecto

2.4. Gestión económica

En esta sección estimaremos el coste económico necesario para llevar adelante el proyecto. Dividiremos los costes en recursos hardware, recursos software y recursos humanos, ayudándonos a ver el impacto de cada tipo de recurso de forma separada. Se tendrá en cuenta la vida útil de los elementos y la duración del proyecto para calcular la amortización de los recursos.

2.4.1. Recursos humanos

La tabla 2 contiene todos los recursos humanos que serán necesarios durante el desarrollo del proyecto: Dividiendo las diferentes tareas, se podrá entender mejor el número de horas dadas a cada rol en la tabla 4. El director del proyecto se encargará de la planificación del proyecto. El diseñador de software se encargará de la programación, incluyendo la programación de los elementos del proyecto como el visor 3D como del desarrollo de los algoritmos. Lo mismo se puede decir del programador. El tester se encargará de ir realizando pruebas durante el desarrollo del proyecto, buscando errores en los diferentes módulos. Además, tendrá una extensa tarea final de comprobación del programa final. El analista se encargará de recoger los datos y realizar estudios y comparativas entre ellos.

Rol	**Precio por hora (Euros/hora)**	**Tiempo (horas)**	**Coste (horas)**
Director	50	100	5000
Diseñador	35	137	4795
Programador	35	120	4200
Tester	25	60	1500
Analista	30	40	1200
Total		457	16695

Tabla 2: Presupuesto de los recursos humanos

2.4.2. Recursos hardware

La tabla 3 contiene todo el hardware que se utilizará durante el desarrollo del proyecto:

2.4.3. Recursos software

La tabla 4 contiene todo el software que se utilizará durante el desarrollo del proyecto:

Recurso	**Precio (Euros)**	**Vida útil**	**Amortización**
PC	1200	4 años	125
Portátil	800	4 años	83.3
Cámara Kinect	99	4 años	10.3
Adaptador Kinect para Windows	49.99	4 años	5.1
Cámara RealSense	99	2 años	20.6
Motor	20	2 años	4.2
Microcontrolador Arduino	20	4 años	2
Total	2287.99		250.5

Tabla 3: Presupuesto de los recursos hardware

Recurso	**Precio (Euros)**	**Vida útil**	**Amortización**
Windos 10	135	3 años	18.75
Qt	0	3 años	0
Visual Studio 2013	0	3 años	0
Controladores	0	3 años	0
Total	135		18.75

Tabla 4: Presupuesto de los recursos hardware

2.4.4. Presupuesto final

Utilizando los datos de las tablas 2, 3 y 4 se describirá el coste total del proyecto en la tabla 5

Concepto	**Coste (Euros)**
Recursos Humanos	2287.99
Recursos Hardware	135
Recursos Software	16695
Total	19117.99

Tabla 5: Presupuesto total estimado

2.4.5. Control del presupuesto

Como se ha mencionado anteriormente, el presupuesto puede requerir modificaciones si aparecen obstáculos o problemas durante el desarrollo del proyecto. Aún y teniendo problemas con el proyecto, es difícil que aparezcan gastos extra de hardware exceptuando el caso en que algún elemento deje de funcionar, en cuya situación se deberá incrementar el coste comprándolo de nuevo. En cuanto a los costes de software, es muy poco probable que aumenten los gastos ya que existen multitud de alternativas gratuitas para realizar las tareas que serán necesarias durante el desarrollo. Resumiendo, el principal foco de imprevistos en cuanto al coste es el apartado de recursos humanos, dónde es mucho más difícil mantener un control de las horas y la dificultad del trabajo. Como se puede observar en el apartado de Planificación Temporal, la tarea que más tiempo ocupa, y por lo tanto, más recursos consume es la del desarrollo de los algoritmos. Ya que esta tarea involucra a diferentes perfiles como programador, diseñador y tester, es muy complicado predecir el tiempo que ocupará su desarrollo y por lo tanto es muy posible que su coste final sea diferente al nombrado en presente documento.

2.5. Sostenibilidad y responsabilidad social

En esta sección analizaremos la sostenibilidad del proyecto en sus diferentes áreas de acción: área económica, área social y área medioambiental.

2.5.1. Sostenibilidad económica

En el presente documento ya se puede encontrar una estimación de los costes del proyecto, tanto materiales como humanos. El coste que se puede

observar en el apartado de Gestión Económica debería ser el único coste del proyecto, ya que al consistir en un proyecto cuyo objetivo es el análisis y la comparación de diferentes tecnologías y métodos, no se espera que haya que realizar un mantenimiento ni un desarrollo posterior, a no ser que se requiera añadir elementos al estudio o, si éste resulta un éxito, se busque desarrollar una herramienta para incorporar la tecnología a futuros proyectos. Realizar un proyecto con un menor coste resultaría imposible, ya que se requiere de hardware especializado que actualmente no se encuentra a un precio más económico que el aquí nombrado. Además, se espera cierta calidad en los resultados obtenidos por el hardware por lo que opciones más baratas quedan fuera de posibles elecciones. Ya que el proyecto no espera salir al mercado si no utilizarse como una herramienta interna, no se espera obtener ingresos económicos con él. Aun así, es posible que si el resultado del estudio es muy satisfactorio, se pueda desarrollar software capaz de integrar la funcionalidad de seguimiento facial a diferentes proyectos, pudiendo vender una licencia permitiendo recuperar la inversión inicial del estudio. El proyecto recibirá un 9 en el área de viabilidad económica, ya que su precio no es exageradamente elevado y además es muy difícil conseguir gastar menos dinero. Aun así, sería posible utilizar software libre de forma íntegra, pero eso añadiría complicaciones al proyecto a causa de problemas de compatibilidad de los diferentes controladores.

2.5.2. Sostenibilidad social

La aplicación que se desarrollara en este proyecto se utilizará para un ámbito muy concreto de la informática. En concreto estudiará la posibilidad de disponer de detección facial con ángulo. Si el proyecto resulta en éxito, se podría aplicar la tecnología a diferentes ámbitos de la interacción humano-m áquina, ya que además el hardware necesario para su implementación no tiene un precio abusivo. La tecnología podría proponer sistemas de interacción tanto para videojuegos como para gente con problemas de movilidad en las extremidades, permitiendo controlar ciertas acciones de un ordenador con movimientos de cabeza. Ya que el proyecto no se cierra a un caso de uso y pretende ser una tecnología usable para diversos proyectos, las opciones que pueden aparecer son muy amplias. El proyecto recibirá un 7 en el área de mejora social, ya que es posible que finalmente no tenga mucho impacto si no resulta en un estudio satisfactorio o puede suponer una nueva forma de interacción en caso de resultar un éxito.

2.5.3. Sostenibilidad ambiental

Los recursos utilizados en el proyecto han sido detallados en la Gestión Económica y en la Planificación Temporal. Durante el desarrollo del proyecto, se dispondrá de o bien un ordenador de sobremesa o bien de un portátil encendido. Además de utilizar diferentes cámaras 3D conectadas a esos mismos ordenadores. Cabe destacar que durante las primeras etapas del proyecto también existirá la necesidad de conectar motores y microcontroladores a la corriente. Finalmente, las impresoras, la energía y el papel consumido durante la impresión de toda la documentación serán todos los recursos necesarios. Con toda la información, podemos realizar una estimación de la energía consumida durante el desarrollo del proyecto. Podemos asumir que un ordenador de sobremesa consume una media de 250W mientras se trabaja en el proyecto, y que un ordenador portátil consume unos 130W mientras se realiza la actividad. Además, cabe destacar que el consumo del microcontrolador junto con el motor paso a paso puede ser de unos 5W. Ya que un ordenador será necesario durante el desarrollo del proyecto (Tanto portátil como de sobremesa), asumiendo que se dedican las mismas horas en sobremesa que en portátil, y que durante las primeras etapas del desarrollo se requerirá del microcontrolador, podemos asumir que si la duración del proyecto es de 457 horas, la energía necesaria es de unos 87 kWh, o lo que es equivalente, 56.55 Kg de CO2. El proyecto recibirá un 9 en el área de sostenibilidad ambiental, ya que los únicos recursos agresivos con el medio ambiente son los propios ordenadores y el microcontrolador junto con la energía gastada al desarrollar el proyecto.

2.5.4. Puntuación final de sostenibilidad

A continuación se muestra una tabla resumen (tabla 6 con las diferentes puntuaciones de las áreas del proyecto analizadas en estos apartados:

¿Sostenible?	Económica	Social	Ambiental
Planificación	Viabilidad económica	Mejora calidad de vida	Viabilidad ambiental
Valoración(0:10	9	6	9
Resultados	Coste final versus previsión	Impacto social	Consumo de recursos
Valoración (-10:10)	0	6	9
Riesgos	Adaptación a cambios	Daños sociales	Daños ambientales
Valoración (-20:0)	0	0	0
Valoración total	**39**		

Tabla 6: Matriz de sostenibilidad

3. Registro de nubes de puntos

El proceso de registro de nubes de puntos es uno de los procesos más interesantes cuando se habla de aplicaciones que utilizan escáneres 3D. Aplicaciones de escaneo o de tracking necesitan de algún método para procesar las nubes de puntos resultantes. El registro de nubes de puntos es un problema que aparece en numerosas áreas de la informática. Entre ellas, es notable su aparición en aplicaciones relacionadas con la visión por computador, los gráficos por computador, la robótica, el escaneo de monumentos, la arqueología digital, la arquitectura y numerosas otras áreas. Debido a su gran impacto, la motivación para investigar métodos de registro ha sido muy alta en los últimos años, buscando obtener alineamientos muy fiables y automatizando el proceso.

3.1. Problema de registro

El problema de registro 3D consiste en alinear de forma consistente dos o más nubes de puntos, entendiendo una nube de puntos como un conjunto de puntos tridimensionales (es decir, con 3 coordenadas x, y, z). [6] El objetivo del registro es obtener la orientación 3D relativa (tanto translación como orientación) entre diferentes vistas, alineándolas bajo un mismo sistema de coordenadas, de tal manera que las partes comunes de las diferentes nubes de puntos se alineen de la mejor forma posible. La idea clave consiste en identificar pares de puntos (uno de cada nube de puntos) correspondientes y buscar la transformación que minimice la distancia entre ellos. El registro de parejas de puntos se consigue habitualmente utilizando una de las muchísimas variantes existentes del algoritmo *Iterative Closest Point* (ICP). El algoritmo

ICP requiere de una inicialización, buscando una transformación inicial para mejorar la calidad del alineamiento final y aumentar la velocidad de convergencia. [6] Una vez se ha estimado la transformación rígida relativa para las dos nubes de puntos, el algoritmo refina la transformación de forma iterativa generando pares de puntos en las dos nubes y minimizando una métrica de error. En la figura 3.2 se pueden apreciar dos caminos distintos para realizar el proceso de registro. El primero consiste en obtener puntos de interés para después estimar una transformación entre las nubes. Puede ser una buena etapa de inicialización del ICP, que se representa con el segundo camino.

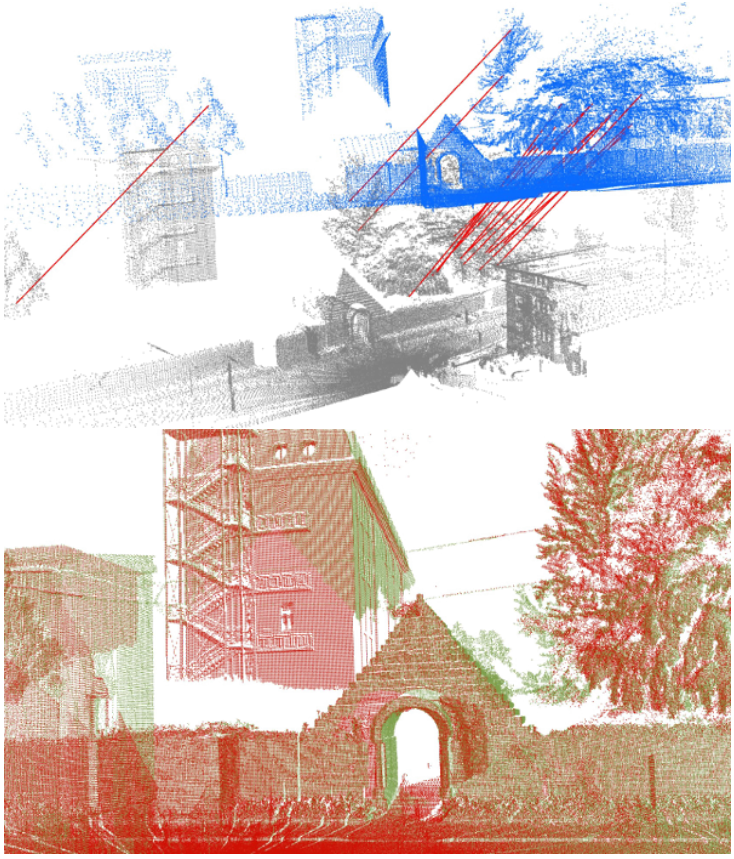


Figura 3.1: Resultado de aplicar la transformación obtenida de un registro sobre dos nubes de puntos

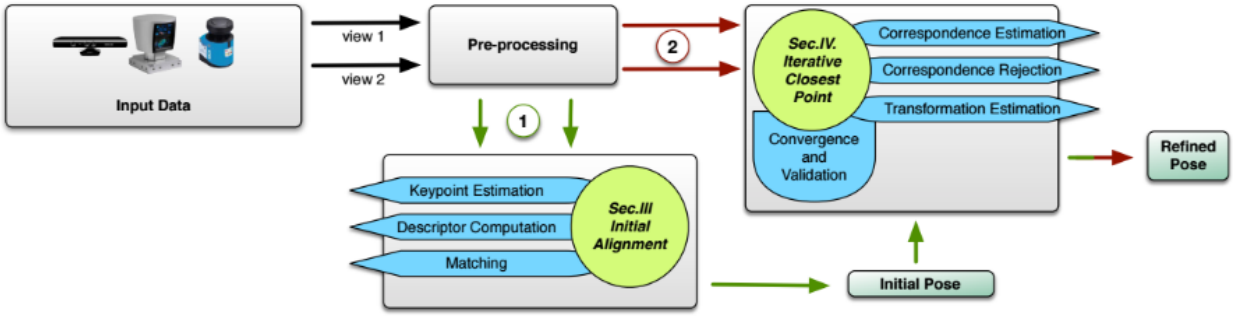


Figura 3.2: Resumen del proceso de registro de un par de nubes de puntos

3.1.1. Formulación matemática

Una nube de puntos consiste en una estructura de datos P utilizada para representar una colección de puntos $p \in R^n$. En un espacio tridimensional, los elementos de la nube de puntos se encuentran representados por las coordenadas geométricas X, Y y Z. Se acostumbra a aumentar la dimensión del espacio cuando existe más información, como por ejemplo información de color. Dado un punto p de una nube P y un punto q de una nube Q, el problema de registro busca encontrar correspondencias entre P y Q, estimando una transformación T que, al aplicarla a la nube P, alinea los pares de puntos correspondientes ($p_i \in P$, $q_j \in Q$). Unos de los principales problemas del registro es que las correspondencias son, normalmente, desconocidas.

3.2. Iterative Closest Point

El concepto del algoritmo de ICP se puede resumir en dos apartados:

- Cálculo de las correspondencias entre las dos nubes de puntos
- Cálculo de la transformación que minimiza la distancia entre las correspondencias

Repitiendo iterativamente los dos pasos se consigue una convergencia a la transformación buscada. En el caso concreto de nuestro proyecto, asumiendo que pueden existir puntos de una nube que no encuentren correspondencia en la otra, utilizaremos una distancia máxima d_{max}. En la mayoría de implementaciones del ICP, la selección de d_{max} representa un *trade-off* entre la convergencia y la bondad de la solución. [7] Un valor muy bajo puede resultar en una mala convergencia, mientras que un valor muy alto genera correspondencias erróneas. En el código 1 se puede observar el algoritmo básico, recibiendo como entrada dos nubes de puntos $A = a_i$ y $B = b_i$ y una transformación inicial T_0 y obteniendo como resultado la transformación T que alinea las dos nubes de puntos.

3.3. Componentes del ICP

Tal y como se ha comentado en el apartado anterior, existen multitud de variantes del *Iterative Closest Point*. Estas variantes del algoritmo se consiguen modificando las diferentes partes en que éste se puede dividir. Dividiremos el algoritmo en las siguientes fases: [5]

1. **Selección** de un conjunto de puntos.

```
T = T0 ;
while not converged do :
  for i = 1 to N do
    mi = FindClosestPointInA (T * bi ) ;
    if dist (transform (T, bi ) , mi ) <= dmax then
      wi = 1 ;
    else
      wi = 0 ;
    end
  end
  Update (T)
end
```

Código 1: Algoritmo ICP básico

2. **Relacionar** los puntos a puntos de la otra nube.
3. **Rechazar** algunos pares basándonos en la visión individual o considerando el conjunto entero.
4. Asignar una **métrica de error** basándonos en los pares de puntos.
5. **Minimizar** la métrica de error

El proceso de registro iterativo repite las fases de relación de los puntos y de alineamiento hasta que el resultado converge o bien se alcanza algún criterio de terminación secundario, como podría ser un número máximo de iteraciones. En caso de que las dos nubes de puntos tengan un alineamiento perfecto (por ejemplo, son la misma nube pero aplicando cierta transformación rígida) el algoritmo alcanza el mínimo global de la métrica de error utilizada, alcanzando lo que se denomina un alineamiento óptimo. [6] Cabe destacar también el hecho de que el algoritmo tiene la posibilidad de quedar atrapado en un mínimo local. Este caso se da cuando las nubes de puntos tan sólo encajan de forma parcial. Es por esta razón que se añade la fase que rechaza algunos pares de puntos. Además, también se puede intentar acelerar el alineamiento utilizando diferentes filtros en la fase de selección [2].

3.3.1. Selección

Las nubes de puntos se obtienen, normalmente, de sensores. Dependiendo del sensor utilizado, las nubes de puntos pueden ser conjuntos muy grandes, sobretodo con sensores de alta resolución. Además, a mayor tamaño de la nube de puntos, más costoso es realizar el registro, lo que lleva a la necesidad de la fase de selección. En muchos casos, la información que aportan los puntos

llega a ser redundante o innecesaria (por ejemplo, en superfícies planas), por lo que el hecho de realizar el registro con un subconjunto de los puntos puede ser suficiente para obtener una transformación decente, obteniendo a cambio un valioso tiempo de cálculo. Se pueden dividir los métodos para selección en dos grandes familias:

1. Detección **automática** de un pequeño conjunto de puntos únicos
2. **Muestreo** del conjunto de puntos inicial

Aplicar un muestreo a la nube de puntos puede ser interesante ya que se reducen el número de puntos a analizar. Aún y así, puede no ser sufificiente o incluso llevar al algoritmo a hacer registros erróneos por la pérdida de resolución general de la nube. El uso de detección de puntos clave en la nube de puntos puede determinar una gran mejora, ya que es más que probable que gran cantidad de puntos no aporten información o sean innecesarios, como por ejemplo puntos del fondo. En la sección 6.1 propondremos tanto un filtro de muestreo como un filtro que elimina los puntos innecesarios para la implementación.

3.3.2. Estimación de correspondencias

La estimación de la correspondencias es el proceso que relaciona puntos p_i de la nube de puntos P a los vecinos más cercanos q_j de la nube de puntos Q. Una forma sencilla pero ineficiente de realizar la búsqueda consiste en realizar una búsqueda exhaustiva en todos los puntos de la nube Q para encontrar el vecino de cada punto de la nube P. Como es de esperar, con el tamaño de las nubes de puntos pudiendo llegar fácilmente a una cardinalidad de millones, el método es prohibitivo. A causa de esto, se han utilizado diferentes estructuras de datos con gran capacidad de búsqueda, como por ejemplo *octrees* o *kd-trees*. Estas estructuras tienen un tiempo de búsqueda logarítmico, obteniendo una mejora sustancial al tiempo lineal del algoritmo inicial. Aún y así, inicializar este tipo de estructuras lleva más tiempo (concretamente, $O(NlogN)$ para los *kd-trees*).
En el caso particular que nos ocupa, en el que las nubes de puntos vienen de cámaras RGB-D, la estimación de correspondencia se puede acelerar utilizando la naturaleza de las nubes de puntos (que provienen provienen de una imagen, por lo tanto se asume una cierta organización en los puntos). Ésto permite poder dejar de lado las estructuras mencionadas anteriormente, que pueden ser muy costosas, y utilizar lo que se denomina *estimación de correspondencia utilizando proyecciones* [6].

3.3.3. Rechazo de correspondencias

Un factor que puede empeorar la solución del registro de nubes de puntos es el hecho de considerar correspondencias incorrectas. Por este motivo, la fase de rechazo es muy importante. Existen diversos métodos de rechazo de correspondencias, entre ellos:

1. **Método basado en distancia:** este método filtra puntos que se encuentran a una distancia determinada (Ver figura 3.3a).

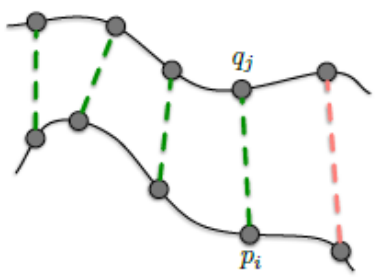
2. **Método basado en la mediana de las distancias:** este método es similar al anterior, pero a diferencia de utilizar un límite fijo, éste se calcula como la mediana de todos las distancias punto-a-punto del conjunto de correspondencias. Está comprobado que el uso de la mediana en vez de la media ayuda a ser más independiente a los *outliers*.

3. **Método basado en reciprocidad:** este método se basa en que, normalmente, cada punto de una nube se corresponde a un punto de la otra nube. Por lo tanto, es posible que ocurra que un punto de la nube inicial tenga más de una correspondencia en la otra nube. El método se queda con un solo par, el que dispone de la mínima distancia (Ver figura 3.3b).

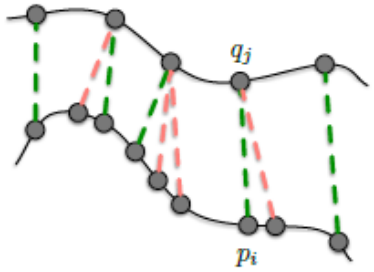
4. **Método basado en compatibilidad de normales:** este método utiliza información de normales de los puntos, rechazando los pares que tengan normales inconsistentes. Se entiende por normales inconsistentes aquellos pares de normales que, por ejemplo, crean un ángulo mayor a cierto valor (Ver figura 3.3c).

5. **Método basado en la frontera de la superfície:** Utilizar puntos de la frontera de una nube puede inducir a errores, sobretodo si estas nubes vienen de un sensor RGB-D y se encuentran parcialmente superpuestas. Utilizando la información de profundidad, se pueden detectar las fronteras utilizando una ventana alrededor de los puntos, buscando discontinuidades (Ver figura 3.3d).

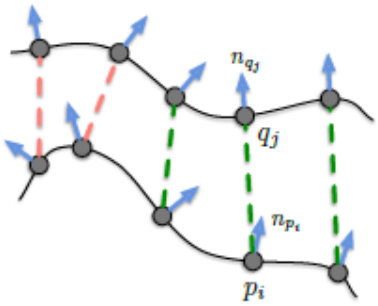
6. **Combinaciones de métodos:** normalmente se utiliza más de un método para rechazar correspondencias. Es posible diseñar un *pipeline* para rechazar correspondencias con los diferentes métodos, uno tras otro.



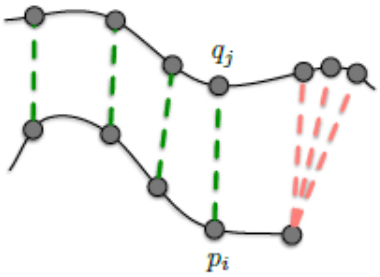
(a) Distancia punto a punto



(b) Correspondencias múltiples



(c) Normales



(d) Fronteras

Figura 3.3: Diferentes rechazadores de correspondencias. Las correspondencias verdes se mantienen y las rojas se rechazan.

3.3.4. Criterios de parada

El criterio de parada define qué condiciones se deben cumplir para que el algoritmo termine de iterar. Existen otros criterios de paradas a parte de definir un número máximo de iteraciones, pudiendo incluso combinar más de uno.

1. **Máximo número de iteraciones:** este criterio define el máximo número de iteraciones. Se considerará que el algoritmo termina cuando se sobrepase el valor impuesto.

2. **Máxima distancia entre transformaciones:** este criterio termina el algoritmo cuando la transformación estimada se encuentra muy lejos de la transformación inicial. Se utiliza cuando se conoce que las dos nubes de puntos se encuentran, como máximo, a una distancia determinada.

3. **Mínima transformación relativa:** este criterio define la mínima diferencia que debe existir entre la transformación de una iteración y la siguiente. Si se cumple el mínimo impuesto, se considera que el algoritmo ha convergido.

4. **Máximo número de iteraciones parecidas:** este criterio permite al algoritmo mantenerse durante un número determinado de iteraciones alrededor de un mínimo. Se acostumbra a utilizar juntamente con el criterio anterior para conseguir que el algoritmo escape de mínimos locales.

5. **Error cuadrático medio relativo:** este criterio es similar al criterio de la mínima transformación relativa. La diferencia consiste en utilizar el error cuadrático medio en lugar del incremento en la rotación/traslación.

6. **Error cuadrático medio absoluto:** este criterio termina de iterar cuando el error entre dos nubes de puntos se encuentra por debajo de cierto valor impuesto.

Una buena combinación y optimización de los diferentes criterios de parada es crítica a la hora de balancear la calidad y el tiempo de cálculo del registro. Una buena configuración determinará la capacidad del algoritmo para ser utilizado en entornos a tiempo real.

3.4. Resultado. Matriz de transformación

Una vez el algoritmo de ICP ha convergido, obtenemos como resultado la matriz de la transformación que, aplicada a una nube de puntos, nos alinea la nube con la nube objetivo. Para la mayoría de aplicaciones, conocer la transformación geométrica puede ser suficiente, pero para el desarrollo de este proyecto nos interesa tan sólo una parte muy concreta de la transformación: la rotación. Es por este motivo que, una vez obtenida la transformación en forma de matriz 4x4, esta se debe descomponer para obtener las diferentes transformaciones que representa [8]: traslaciones, rotaciones y escalados.
Para extraer los diferentes componentes de la transformación, existe un algoritmo que permite extraer transformaciones que, realizadas en la secuencia correcta, dan una transformación equivalente. A efectos prácticos del proyecto, tan sólo las **rotaciones** son interesantes ya que ignoramos tanto los escalados como las traslaciones, pero es interesante ver cómo se extraen también ésos parámetros.
El algoritmo funciona deshaciendo las transformaciones. Los pasos consisten en, primero, determinar el factor de perspectiva. A continuación, se extraen la traslación, el escalado y las rotaciones.
Extraer los componentes de perspectiva es la parte más compleja del algoritmo. Básicamente, se debe resolver la siguiente ecuación:

$$\begin{bmatrix} M_{1,1} & M_{1,2} & M_{1,3} & M_{1,4} \\ M_{2,1} & M_{2,2} & M_{2,3} & M_{2,4} \\ M_{3,1} & M_{3,2} & M_{3,3} & M_{3,4} \\ M_{4,1} & M_{4,2} & M_{4,3} & M_{4,4} \end{bmatrix} = \begin{bmatrix} M_{1,1} & M_{1,2} & M_{1,3} & 0 \\ M_{2,1} & M_{2,2} & M_{2,3} & 0 \\ M_{3,1} & M_{3,2} & M_{3,3} & 0 \\ M_{4,1} & M_{4,2} & M_{4,3} & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & p_w \end{bmatrix}$$

La ecuación anterior se reduce a:

$$\begin{bmatrix} M_{1,4} \\ M_{2,4} \\ M_{3,4} \\ M_{4,4} \end{bmatrix} = \begin{bmatrix} M_{1,1} & M_{1,2} & M_{1,3} & 0 \\ M_{2,1} & M_{2,2} & M_{2,3} & 0 \\ M_{3,1} & M_{3,2} & M_{3,3} & 0 \\ M_{4,1} & M_{4,2} & M_{4,3} & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ p_w \end{bmatrix}$$

Asumiendo que la partición 3x3 de la parte superior izquierda de la matriz M no es singular, el sistema se puede resolver fácilmente. Algunos de los siguientes pasos no funcionarían si la matriz no fuera singular, por lo que no representa un problema muy importante. [8]
Extraer las traslaciones es trivial. La traslación en X es el elemento $M_{4,1}$, la traslación en Y es el elemento $M_{4,2}$ y la traslación en Z es el elemento $M_{4,3}$.
En este punto nos encontramos con una matrix 3x3, que a partir de ahora

llamaremos $M' = M_{1.,3,1.,3}$.
A continuación se obtienen los escalados y los parámetros *shear*. El factor de escalado en X es la longitud de M'_1 y el factor de *shear* $s_{xy} = M'_1$. Entonces, la segunda fila de la matriz se transforma para ser ortogonal con la primera, realizando el siguiente cálculo: $M'_2 = M'_2 - s_{xy}M'_1$. El factor de escala en Y se calcula análogamente utilizando la segunda fila ya modificada, y lo mismo se realiza para el factor de escala en Z.
La matriz resultante es ahora una matriz que define una rotación, pero aún puede incluir un factor de escalado de -1. Para realizar la comprovación, se realiza el determinante de la matriz. Si éste es -1, se niega la matriz y los tres factores de escalado. Si definimos la matriz obtenida como R podemos decir que ésta es igual a:

$$\begin{bmatrix} cos(\beta)cos(\gamma) & cos(\beta)sin(\gamma) & -sin(\beta) \\ sin(\alpha)sin(\beta)cos(\gamma) - cos(\alpha)sin(\gamma) & sin(\alpha)sin(\beta)sin(\gamma) + cos(\alpha)cos(\gamma) & sin(\alpha)cos(\beta) \\ cos(\alpha)sin(\beta)cos(\gamma) + sin(\alpha)sin(\gamma) & cos(\alpha)sin(\beta)sin(\gamma) - sin(\alpha)cos(\gamma) & cos(\alpha)cos(\beta) \end{bmatrix}$$

Por lo tanto, $\beta = arcsin(-R_{1,3})$. Si $cos(\beta) \neq 0$, α se puede calcular fácilmente utilizando $R_{2,3}$ y $R_{3,3}$. Finalmente, γ se puede derivar de $R_{1,2}$ y $R_{1,1}$.
En la implementación del sistema, se ha utilizado la librería *glm* para la descomposición de la matriz 4x4.

4. Desarrollo del visualizador

El visualizador será la plataforma sobre la que se realizarán todas las pruebas, comunicándose con el hardware externo. El sistema tiene unos requerimentos básicos que debe cumplir para permitir experimentar de la forma más cómoda posible, por lo tanto es una parte importante del desarrollo del proyecto, ya que de ella dependen todas las tareas de experimentación y comparación de resultados.

4.1. Especificación del sistema

El visualizador debe permitir realizar las pruebas pertinentes de la forma más sencilla posible y, a ser posible, mostrar toda la información necesaria o que pueda suponer de interés durante el desarrollo del proyecto. Para ello, el sistema debe permitir seleccionar fácilmente que cámara y que algoritmo queremos utilizar para las pruebas, además de dar las opciones de iniciar o terminar el uso de la cámara. También debe ofrecer capacidades a la hora de modificar diferentes parámetros de los algoritmos, permitiendo realizar diferentes pruebas de forma continua sin necesidad de recompilar todo el proyecto. En cuanto a la visualización, debe ofrecer capacidades para observar gráficas que ayuden a interpretar los tres ángulos obtenidos a la hora de realizar el seguimiento de la cabeza del usuario. El sistema también dispondrá de una ventana donde el usuario será capaz de ver la imagen capturada por la cámara 3D, ofreciendo *feedback* sobre el buen funcionamiento de la aplicación y referencia visual de la inclinación de la cabeza. Además, se podrá cambiar la visualización para ver la nube de puntos que está siendo usada para los cálculos de los algoritmos.

4.2. Diseño

A continuación se presenta un diagrama de clases simplificado del sistema (Figura 4.1), de forma que se pueda apreciar la estructura que se utilizará en el proyecto. Se ha utilizado un diseño basado en herencia para facilitar la implementación de nuevas cámaras y nuevas tecnologías. La idea era desarrollar un sistema fácilmente expandible capaz de soportar multitud de algoritmos para así implementar una plataforma de pruebas reusable para diferentes proyectos. Una vez desarrollado el esquema propuesto, implementar la segunda cámara (la *Microsoft Kinect 2*) en el sistema supuso tan sólo un día de trabajo. El sistema completo ha sido desarrollado utilizando *Qt*, ya que es una librería muy útil a la hora de desarrollar interfaces gráficas en *C++*.

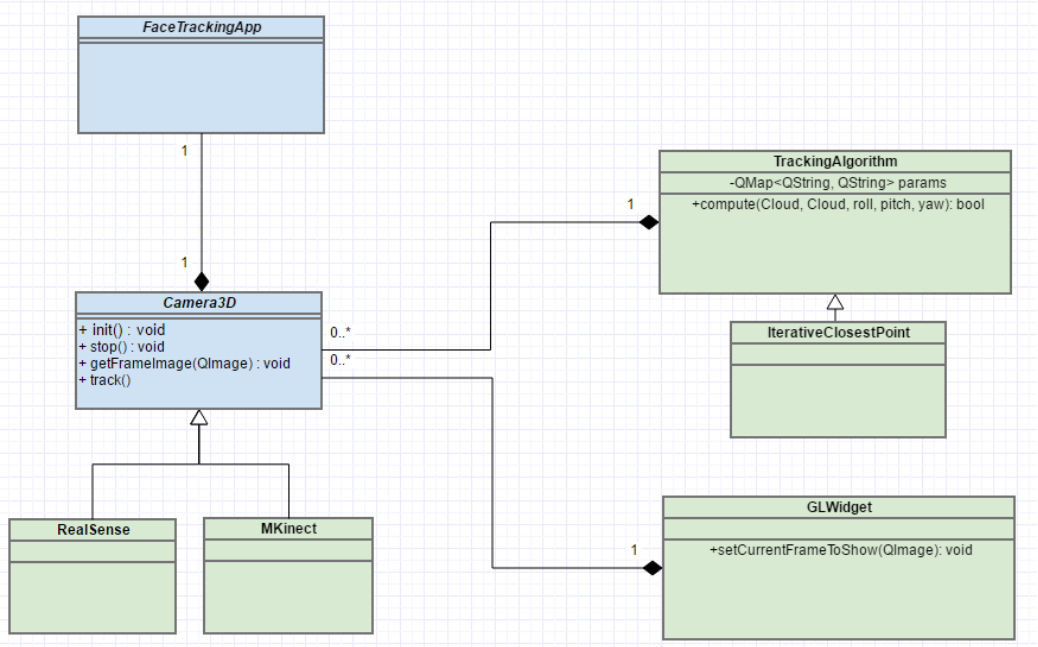


Figura 4.1: Diagrama de clases del visualizador (simplificado)

4.3. Implementación

Tal y como se ha comentado en el apartado de Diseño, se ha utilizado *Qt* para desarrollar el sistema completo del visualizador. *Qt* ofrece muchas extensiones al lenguaje C++, como por ejemplo el concepto de *Widget* o *Signal/Slot*. Se conoce por *Widget* en *Qt* a todo elemento de la interfaz de usuario. Es capaz de recibir la interacción del usuario y de dibujarse sobre la pantalla. Todo elemento de una interfaz hecha en *Qt* es un *Widget*, tanto la ventana principal como los botones. Todos los *Widgets* de *Qt* se representan como clases que heredan de la clase `QWidget`, como por ejemplo `QPushButton` para los botones o `QLabel` para los textos. Además, *Qt* dispone de métodos para determinar la forma de comunicación entre los diferentes elementos de la interfaz. Para ello se utilizan los *Signals/Slots*. Los Signals y los Slots son funciones especiales que *Qt* ofrece para determinar que *Widgets* se comunican y que información se envían entre ellos. En concreto, un signal es emitido por un objeto (bajo ciertas circunstancias, como la recepción de un evento) y un slot es una función que se ejecuta cuando cierto signal es recibido. Para las conexiones de los diferentes widgets de la aplicación se ha utilizado el software *Qt Designer*. También se pueden conectar signals y slots de forma programática utilizando la función `connect`.

Utilizando los elementos ya mencionados, se ha programado una interfaz

Figura 4.2: Signal *clicked()* conectado a slot *saveYawPlot()*

con las funcionalidades básicas necesarias para nuestro proyecto. Es decir, es capaz de seleccionar entre las diferentes cámaras y algoritmos, permite modificar los parámetros de estos últimos y ofrece sistemas de visualización para poder ver rápidamente los resultados de las pruebas. Como añadido, se ha implementado la capacidad de guardar directamente las gráficas desde el propio programa, para así facilitar el trabajo de documentación de resultados posterior. Además, se ha implementado de forma que los datos de la prueba se actualizan tan pronto como el algoritmo haya dado resultado. Es decir que en todo momento el usuario realizando la prueba es capaz de ver exactamente que resultados se están obteniendo a tiempo real. En la figura 4.3 se puede apreciar la interfaz desarrollada, marcando con los siguientes números sus diferentes partes:

1. Lista de cámaras disponibles
2. Lista de algoritmos disponibles
3. Visor de la cámara (tanto RGB como puntos)
4. Gráficas resultantes de la detección
5. Información de la detección
6. Ángulos detectados
7. Selector para el visor (RGB o puntos)
8. Botones de control: Iniciar cámara, Apagar cámara o iniciar test
9. Configurador del algoritmo
10. Selector de estimador de correspondencias
11. Selección de rechazadores
12. Parámetros de los rechazadores

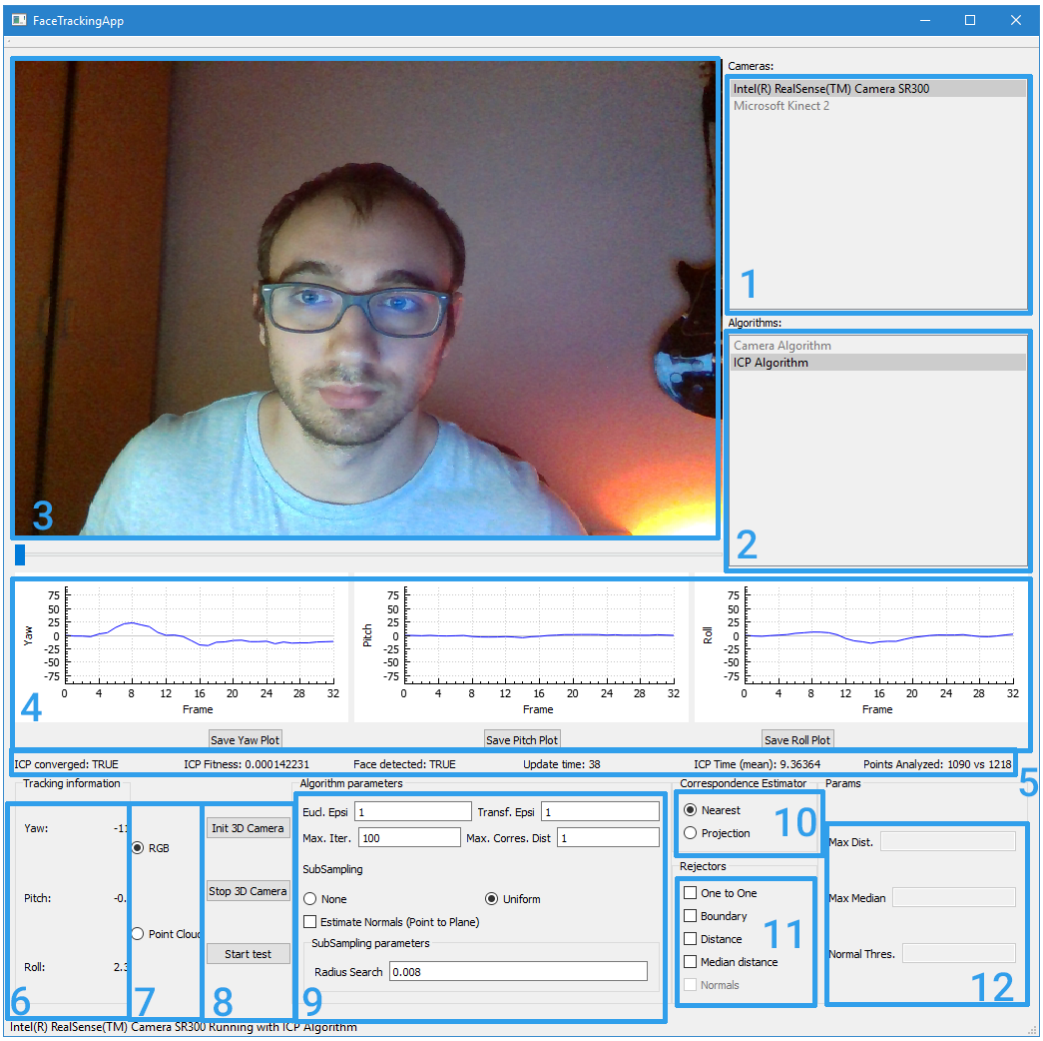


Figura 4.3: Interfaz del visualizador

5. Desarrollo de la plataforma de pruebas

Durante el desarrollo del proyecto será necesario realizar una serie de pruebas. Para realizar las pruebas con la máxima robustez posible, se ha desarrollado una plataforma externa que ofrecerá las mismas condiciones durante todos los test, ofreciendo un alto nivel de repetibilidad. La idea básica de la plataforma consiste en controlar el giro de una cabeza de plástico mediante el uso de un motor. Éste a su vez es controlado por un microcontrolador Arduino, permitiendo programar los movimientos libremente. Además, para añadir comodidad a la plataforma, se realiza una comunicación entre el programa principal y el microcontrolador, permitiendo realizar los test directamente desde el programa y comparar los valores recibidos por la cámara 3D con el valor informado al motor. De esta manera evitamos factores humanos durante la realización del test, realizando movimientos precisos y mediciones repetibles que serían inalcanzables si los test se realizaran con una persona. En la figura 5.1 se puede observar gráficamente la estructura de conexiones del sistema.

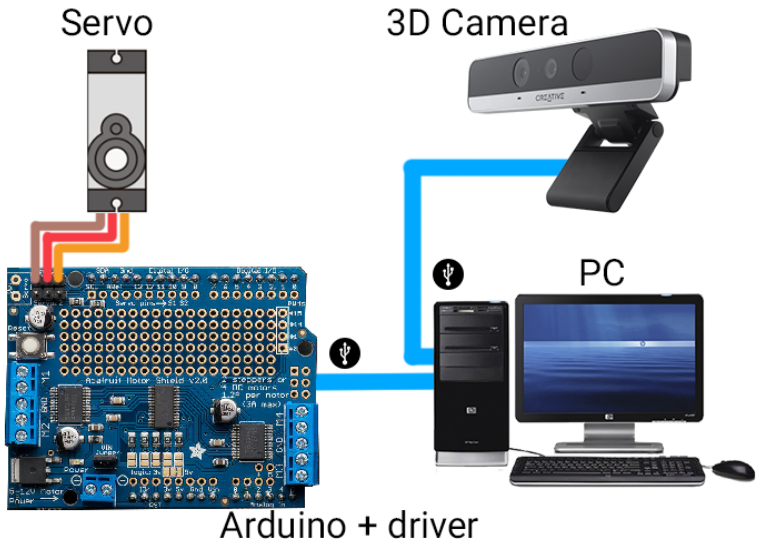


Figura 5.1: Esquema del sistema completo

5.1. Arduino y conexión serie

Arduino es una plataforma de prototipado de código libre que busca una gran facilidad en el uso tanto del hardware como del software. Las placas Arduino son capaces de leer entradas de sensores y de convertirlas en salidas, ya

sea activando un motor cuando se pulse un botón o encendiendo un led cuando se detecte un bajo nivel de iluminación. Para programar las placas Arduino se utiliza un lenguage implementado en *C/C++* basado en Wiring [9]. Wiring es un framework de código abierto para microcontroladores que permite escribir aplicaciones multiplataforma para controlar elementos conectados a un gran número de microcontroladores [10], de forma muy similar a Arduino. Un programa arduino consta básicamente de dos partes:

1. Una función de **setup**, en la que se inicializan todos los elementos del programa.

2. Una función **loop**, que se repite indefinidamente siempre que la placa se mantenga encendida.

En el código 2 se puede observar el código Arduino que hace parpadear un diodo LED.

```
1  // the setup function runs once when you press reset or power
       the board
2  void setup() {
3    // initialize digital pin 13 as an output.
4    pinMode(13, OUTPUT);
5  }
6
7  // the loop function runs over and over again forever
8  void loop() {
9    digitalWrite(13, HIGH);   // turn the LED on (HIGH is the
       voltage level)
10   delay(1000);              // wait for a second
11   digitalWrite(13, LOW);    // turn the LED off by making the
       voltage LOW
12   delay(1000);              // wait for a second
13 }
```

Código 2: Ejemplo de código Arduino

Se puede apreciar que programar en Arduino es realmente sencillo. Aún y así, es necesario un método de comunicación entre la placa y el ordenador, para que así podamos mandar órdenes directamente desde el programa principal. Para ello se utiliza la conexión serie. Implementar comunicación serie con Arduino es realmente sencillo, ya que la plataforma nos ofrece un sistema de comunicación ya implementado sobre la placa. [11] Para realizar la comunicación desde el programa principal a la placa, Arduino nos ofrece una librería escrita en *C++* para facilitar las llamadas a la línea serie del ordenador, compatible con plataformas *Windows*.

5.2. Montaje del controlador

Para facilitar el control del servomotor y eliminar preocupaciones sobre conexiones y alimentación, se ha utilizado un controlador para Arduino. El controlador proporciona una interfaz simple en la que conectar el servomotor además de un microchip completamente dedicado a las señales PWM. Para utilizarla, se ha tenido que realizar cierto trabajo de soldado para conectar los terminales a la placa, como se puede apreciar en la figura 5.3.

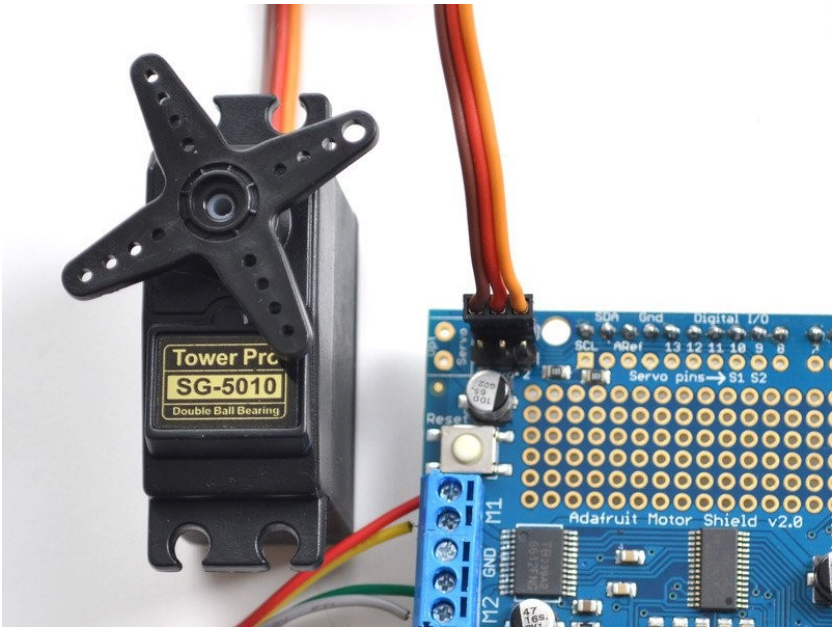


Figura 5.2: Servomotor conectado al controlador

El controlador soporta de serie el uso de 2 servomotores, 4 motores DC bidireccionales y hasta 2 motores paso a paso. Además, el controlador es compatible con diferentes placas Arduino, como por ejemplo los Arduino UNO, los Arduino Leonardo y muchos otros. [12] En la figura 5.2 se puede apreciar el servomotor conectado a la conexión `Servo 1` del controlador.

5.3. Plataforma giratoria

Con el motor funcionando, el siguiente paso es realizar el montaje de la plataforma giratoria. Es necesario dispone de una base lo más estable posible sobre la que colocar la cabeza, ya que el terminal giratorio del motor es muy pequeño.
Para el montaje, se han utilizado elementos caseros para desarrollar un prototipo rápido pero funcional, permitiendo llevar a cabo las pruebas necesa-

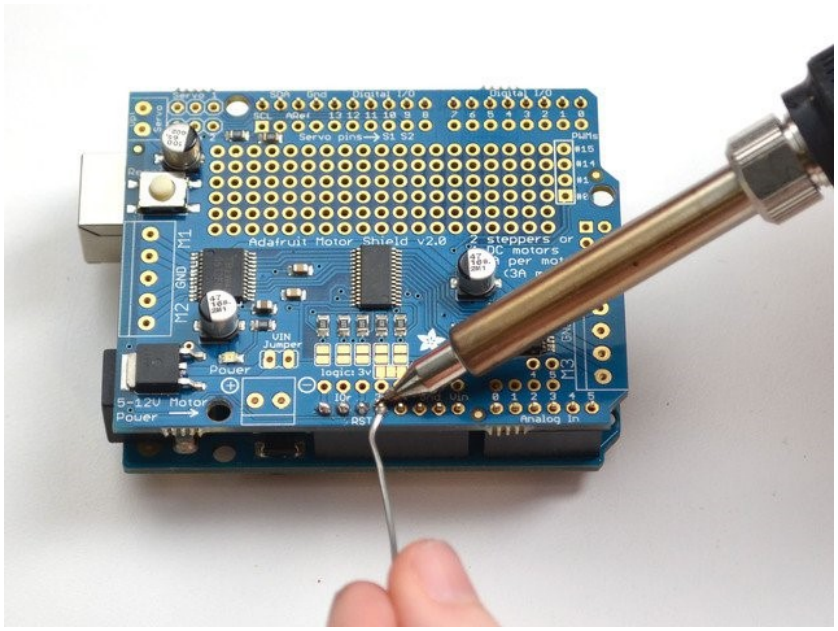


Figura 5.3: Soldado de los terminales al controlador

rias. Estos elementos caseros se componen básicamente de trozos de madera resultantes de desensamblar diferentes elementos de cocina. El montaje se realizó utilizando clavos para mantener las piezas estructurales juntas y para mantener el terminal del motor unido a la plataforma rotatoria. Además el motor se atornilló a la madera estructural para añadir estabilidad durante los movimientos.

Figura 5.4: Materiales utilizados para el prototipaje

Figura 5.5: Plataforma giratoria con una cabeza 3D impresa mediante estereolitografía

6. Desarrollo del algoritmo de detección de la orientación

El objetivo del proyecto consiste en comparar los resultados obtenidos por los algoritmos implementados en las cámaras con los resultados obtenidos por el *Iterative Closest Point*. para ello, se ha utilizado la libreria *Point Cloud Library*, también conocida como *PCL*. La librería dispone de su propia implementación del algoritmo de *Iterative Closest Point*, y de hecho dispone de diferentes variantes del mismo. Para su funcionamiento, el algoritmo requiere como entrada dos nubes de puntos: la primera nube siendo la que será transformada y la segunda nube siendo la nube de puntos objetivo. La variante principal del *ICP* se encuentra en la clase `IterativeClosestPoint`. Aún y disponiendo de diferentes variantes, en nuestro proyecto nos limitaremos a utilizar la clase básica del algoritmo y a modificar diferentes parámetros que nos ofrece la misma. A causa de que la librería *PCL* dispone de su propio tipo de datos para definir las nubes de puntos, nuestra aplicación requiere de un sistema de conversión. Necesitamos obtener la nube de puntos que detecta cada sensor y transformarlo a un formato compatible con la librería. En el caso de la nube de puntos obtenida por la *Intel RealSense*, es suficiente con transformar las unidades de los puntos de milímetros a metros. En el caso de la *Kinect 2*, los puntos ya vienen dados en metros.

Una vez las nubes de puntos se encuentran en un formato utilizable por la librería, existen diferentes formas de realizar el registro. El primer esquema de seguimiento que se planteó consistía en realizar, por cada frame, un registro entre la nube de puntos capturada en el frame actual con la nube de puntos capturada en el frame anterior. Esta opción fue descartada ya que se consideró demasiado propensa a errores, ya que se irían acumulando los errores de cada registro, provocando un desplazamiento en la detección que se iría aumentando a medida que se vayan realizando más registros. Como alternativa al esquema inicial, se planteó el hecho de comparar la nube de puntos capturada en cada frame con la nube de puntos inicial, para así no acumular errores progresivos. En la figura 6.1 se puede apreciar un diagrama de flujo del esquema propuesto.

El esquema propuesto añade una dificultad importante: es muy posible que, a medida que la cabeza del usuario se aleja de la posición inicial, el algoritmo de *ICP* empiece a tener problemas a causa de la dificultad en el registro de las dos mallas de puntos. Para intentar mejorar el registro, se decidió guardar la transformación obtenida en cada frame para así inicializar

el algoritmo con una transformación inicial que sabemos se encuentra cerca de la transformación actual, acelerando así la convergencia del algoritmo y mejorando la solución obtenida.

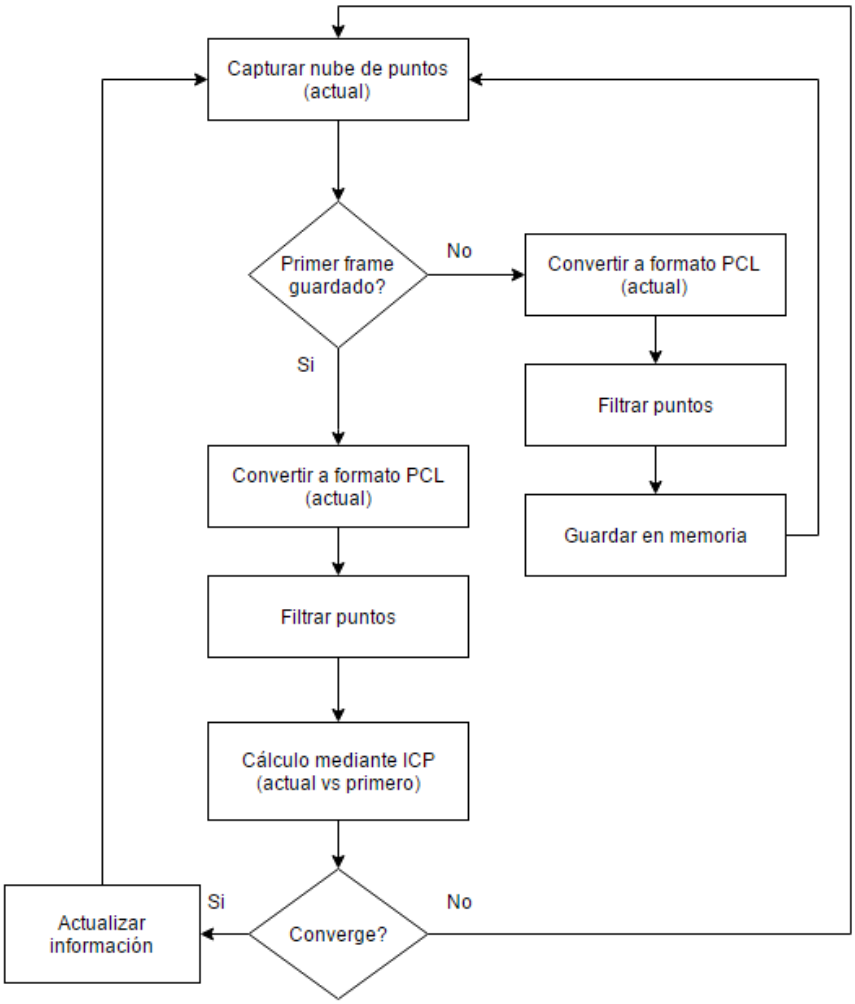


Figura 6.1: Proceso de tracking de nuestra aplicación

6.1. Filtrado de los puntos

A continuación presentaremos dos filtros propuestos durante el desarrollo del proyecto. El primer filtro presentado es el filtrado uniforme, un filtro de muestreo que no utiliza ningún tipo de información del dominio. A continuación, se presentará un filtro automático que utiliza información extra sobre la posición de la cabeza, ofreciendo ciertos datos que aportan un gran valor informativo al filtrado.

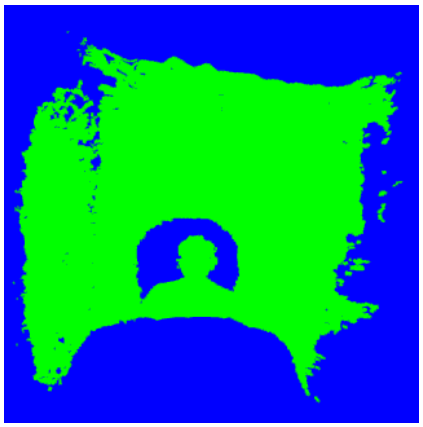
Filtrado Uniforme

El filtrado uniforme es el filtro más básico que trataremos en este trabajo. Tal y como se ha comentado anteriormente, es un filtrado genérico que no utiliza ningún tipo de información del dominio, es decir, todos los puntos de la nube tienen el mismo valor.

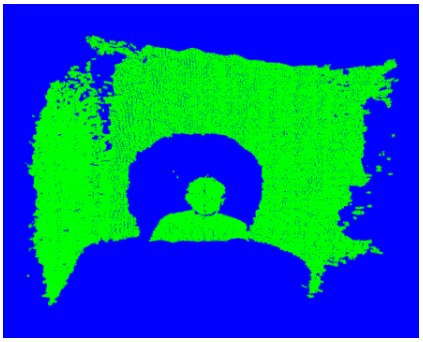
El filtrado uniforme construye una cuadrícula de vóxeles 3D por encima de la nube de puntos. A continuación, por cada vóxel, todos los puntos que se encuentran en su interior se aproximan con su centroide. Este algoritmo es un poco más lento que aproximar los puntos con el centro del voxel pero consigue representar la nube de puntos con más fidelidad. [13] Cómo se puede observar en la figura 6.2a, la nube de puntos presenta una gran densidad, por lo que el cálculo del *ICP* sería extremadamente lento. Se puede observar el resultado de aplicar un filtro uniforme con diferentes radios del voxel, concretamente con un radio de 0.005 mm en la figura 6.2b, un radio de 0.01 mm en la figura 6.2c y un radio de 0.05 mm en la figura 6.2d. Además, también se ha querido comparar el tiempo que se tarda en calcular las diferentes nubes de puntos filtradas en función del radio del vóxel. Para realizar el análisis, se han estudiado los datos de 20 frames, cada uno con su nube de puntos correspondiente, y se ha calculado el tiempo de cálculo medio. Los resultados se pueden observar en la tabla 7. Además del radio y el tiempo de cálculo se ha añadido, para efectos informativos, la media de las diferencias entre la cantidad de puntos inicial y la cantidad de puntos final. Como es lógico, a mayor tamaño del vóxel más puntos se quedan fuera de la nube final.

Radio del vóxel	**Diferencia**	**Tiempo**
0.005 mm	130237	3.9 s
0.01 mm	207296	1.9 s
0.05 mm	237036	1.9 s

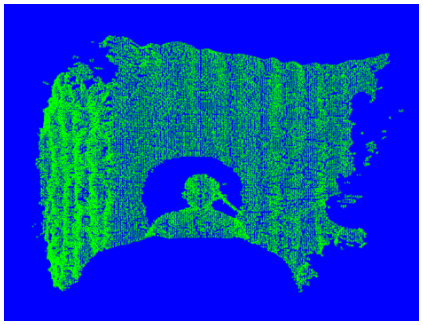
Tabla 7: Tiempo de filtrado con diferentes radios de vóxel



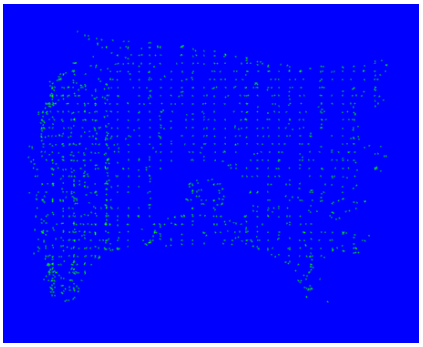
(a) Sin filtrar



(b) Radio = 0.005m



(c) Radio = 0.01m



(d) Radio = 0.05m

Figura 6.2: Nubes de puntos filtradas con diferentes radios utilizando el filtrado uniforme.

Eliminar puntos no pertenecientes a la cabeza

Un filtro más interesante para el objetivo del proyecto consiste en filtrar los puntos en función de su posición. Gracias a los datos obtenidos por la cámara 3D y a que sabemos que el usuario se encuentra a una distancia más o menos conocida al sensor, podemos realizar un filtrado muy simple eliminando los puntos que se encuentran a una profundidad superior a cierto valor. Se ha comprobado que un valor de 50 cm es suficiente para recortar la cabeza del usuario de una forma muy correcta (ver figura 6.3), siendo además un cálculo muy sencillo y rápido ya que es suficiente con rechazar todos los puntos con una coordenada Z superior a 0.5 metros.

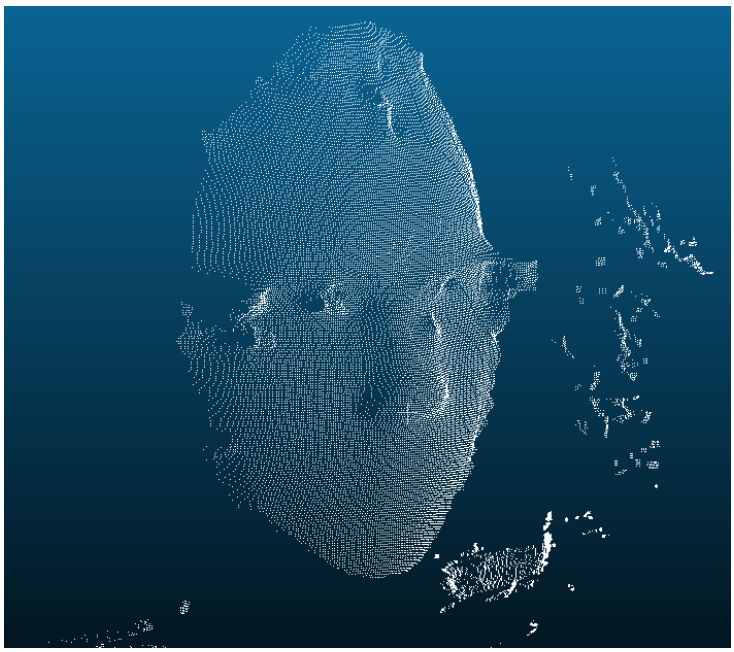


Figura 6.3: Puntos de la cabeza aislados

En la figura 6.4 se pueden apreciar las nubes de puntos capturadas en diferentes ángulos de giro. Se puede observar la nube de puntos inicial (derecha) junto con nubes de puntos detectadas durante una rotación hacia la derecha de la cabeza (centro e izquierda).

Combinación de filtros

Debido a la gran reducción en el número de puntos una vez se han eliminado los puntos no pertenecientes a la cabeza del usuario, el filtrado uniforme se acelera notablemente. A continuación se puede observar, en la tabla 8, una comparativa con los tiempos obtenidos aplicando filtrados uniformes de diferentes radios a la nueva nube de puntos. Se ha comprobado experimentalmente que, a medida que se aumenta el tamaño del vóxel y se reduce el número

Figura 6.4: Nubes de puntos de diferentes posturas

de puntos, la capacidad del algoritmo para hacer un seguimiento robusto disminuye drásticamente. Se han considerado los valores entre 0.008mm y 0.01mm como los más adecuados a la hora de mantener un tiempo de cálculo razonable y un seguimiento robusto.

Radio del vóxel	**Diferencia**	**Tiempo**
0.005 mm	16661	163 ms
0.008 mm	19247	171 ms
0.01 mm	19658	175 ms
0.03 mm	20661	179 ms
0.05 mm	25187	221 ms

Tabla 8: Tiempo de filtrado con diferentes radios de vóxel (cabeza filtrada)

6.2. Optimización de parámetros

El algoritmo *ICP* es altamente configurable. Desde los diferentes criterios de parada como el máximo número de iteraciones pasando por diferentes métodos para calcular y rechazar correspondencias. Debido a la importancia del tiempo de cálculo en nuestro proyecto (debido a que requerimos una

aplicación que trabaje prácticamente a tiempo real) debemos encontrar una configuración de parámetros que ofrezca un buen compromiso entre tiempo de cálculo y resultado obtenido.

6.2.1. Análisis inicial

En esta primera etapa de optimización de parámetros se realizarán las pruebas pertinentes para intentar encontrar los parámetros básicos que requiere el *ICP*. Nos centraremos en optimizar 4 parámetros clave del algoritmo. Muchos de ellos determinan los criterios de parada del mismo (para más detalles sobre los criterios de parada, ver apartado 3.3.4):

1. El máximo número de iteraciones del algoritmo. El valor se puede obtener con el método `getMaximumIterations`.

2. La distancia máxima que dos puntos correspondientes pueden tener. El valor se puede obtener con el método `getMaxCorrespondenceDistance`.

3. El error euclídeo máximo entre dos iteraciones consecutivas del algoritmo. El valor se puede obtener con el método `getEuclideanFitnessEpsilon`.

4. La diferencia máxima entre dos transformaciones consecutivas del algoritmo. El valor se puede obtener con el método `getTransformationEpsilon`.

Nos centraremos en buscar la combinación de parámetros que ofrezca tanto un tiempo de cálculo razonable, permitiendo realizar el cálculo a tiempo real como un valor de error mínimo. Para esta optimización inicial utilizaremos como métrica de error la suma de las distancias cuadradas entre las dos nubes de puntos, que se puede obtener utilizando la propia librería *PCL* mediante el método `getFitnessScore`. Se tomarán 100 muestras con cada configuración y se mostrará, en las siguientes gráficas, tanto el valor medio obtenido como la desviación representada en forma de barras verticales.
El primer parámetro a optimizar será el número de iteraciones del *ICP*. Se han tomado muestras utilizando valores de entre 10 a 100 máximas iteraciones y se ha comprobado tanto el *fitness* obtenido como el tiempo de cálculo. Se puede apreciar en la figura 6.5 tanto el valor medio del *fitness* como la desviación del mismo. Hay valores, como las 20 o las 80 iteraciones, que no presentan una robustez a la hora de obtener resultados fiables. El comportamiento esperado del algoritmo sería que, a más iteraciones, mejor fuera el *fitness*. Aún así a partir de cierto número de iteraciones empieza a mostrar resultados peores. Creemos que la razón detrás de este comportamiento es que llega un punto donde el algoritmo realiza transformaciones erróneas que parecen alejarse de la solución correcta.

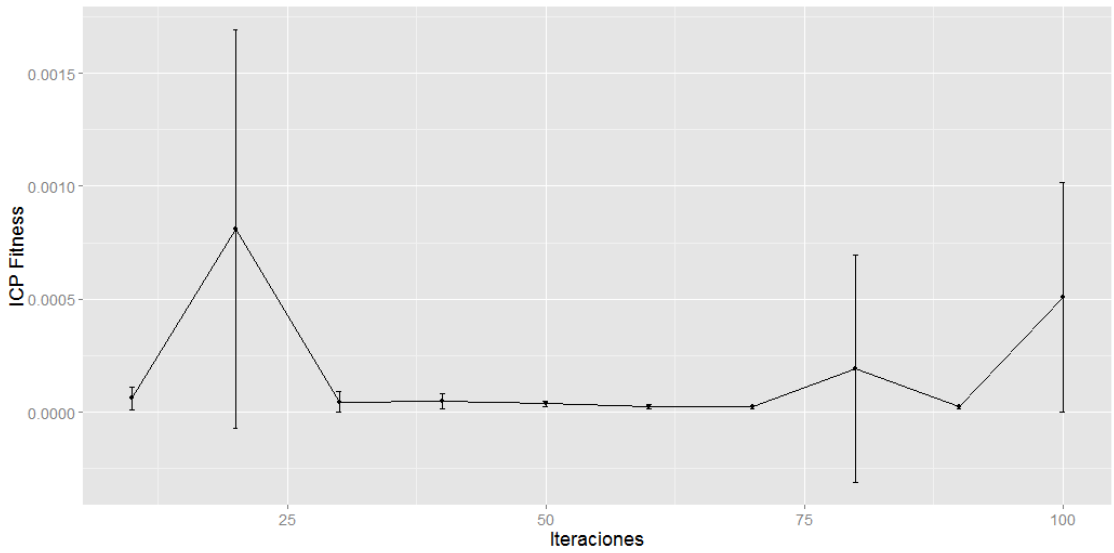


Figura 6.5: Número de iteraciones ICP vs fitness obtenido

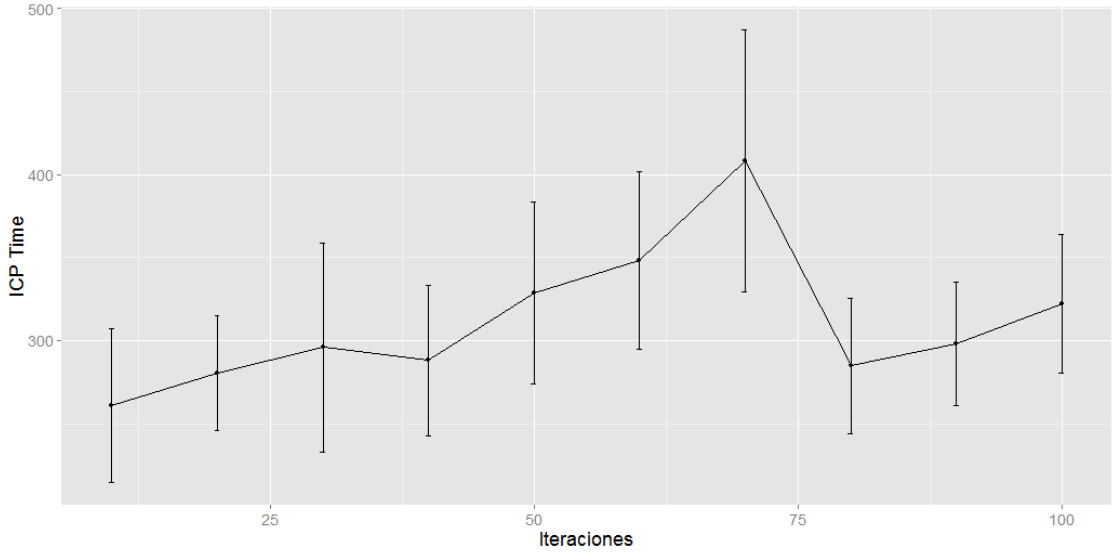


Figura 6.6: Número de iteraciones ICP vs Tiempo de cálculo (ms)

En la figura 6.6 se puede observar la tendencia del *ICP* a tardar más cuantas más iteraciones máximas se pueden realizar. Cabe destacar el hecho de que a partir de cierto número de iteraciones el tiempo cae drásticamente. Se cree que la razón puede ser el hecho de que, al tratarse de una cota superior de iteraciones, el algoritmo empieza a terminar por otras razones a partir de éste número. Es interesante observar que el fenómeno ocurre alrededor del mismo número de iteraciones que provocaba que el *fitness* empeorara en la figura 6.6. Al observar las dos gráficas, al final se decidió por utilizar **50 iteraciones** como valor definitivo, ya que presenta un buen compromiso entre tiempo de cálculo (alrededor de 325ms) y *fitness*.

El siguiente parámetro a optimizar será la distancia máxima que pueden tener dos puntos para ser correspondientes. Se documentarán tan sólo valores que generen convergencia, ya que experimentando se ha comprobado que valores menores a 1 hacen que el algoritmo nunca converja. Se puede apreciar

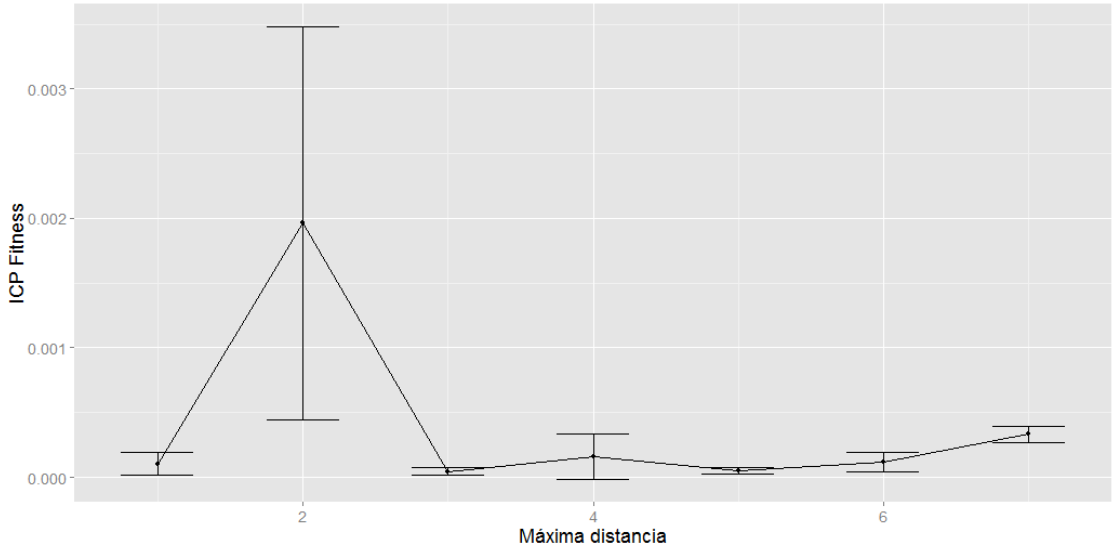


Figura 6.7: Máxima distancia vs fitness obtenido

que el algoritmo no marca mucha diferencia cambiando valores de distancia máxima para las correspondencias, siempre mostrando valores de *fitness* de entre 0.000 y 0.003 (ver figura 6.7), por lo tanto parece ser un parámetro que no afecta demasiado a la bondad de la solución. La causa es posible que sea que al tratarse de movimientos muy controlados las correspondencias nunca llegan a estar muy lejanas y por lo tanto el hecho de aumentar el límite no las cambia. Aún y así cabe destacar que parece tener cierto efecto en el tiempo de cálculo, siendo el valor de 4 mm para el parámetro el que marca mejores

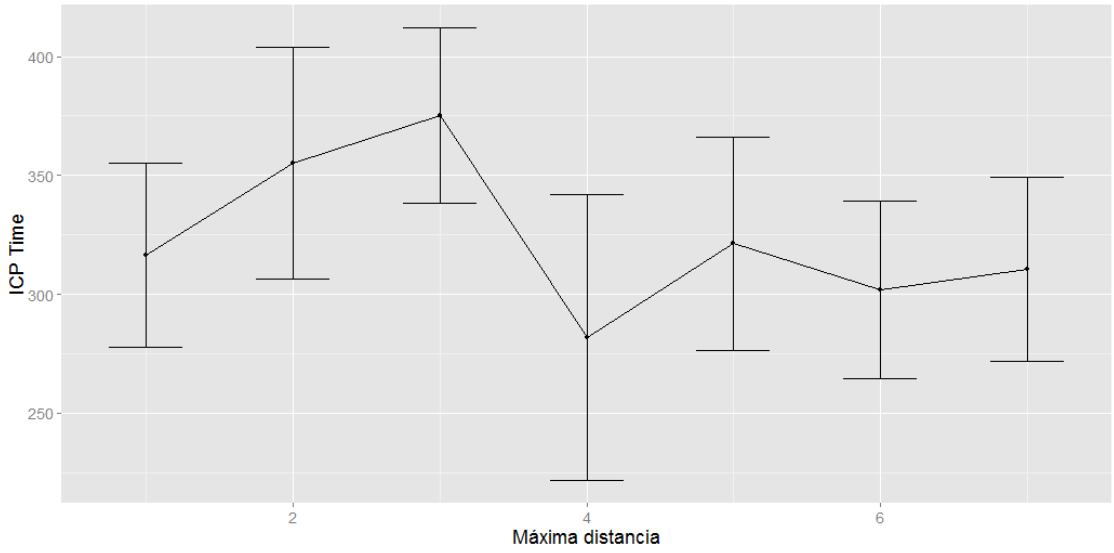


Figura 6.8: Máxima distancia vs Tiempo de cálculo (ms)

tiempos (ver figura 6.8).
Tomaremos el valor de **4 mm para la máxima distancia entre correspondencias** como valor definitivo para nuestro algoritmo.

A continuación el siguiente parámetro a optimizar es el error euclídeo máximo entre dos iteraciones. En la figura 6.9 vemos cierta mejora con los valores 0.01 y 0.1, aunque no llegan a ser mejoras realmente significativas. En la figura 6.10 se puede apreciar cierta tendencia a mejorar cuando el error máximo se sitúa sobre los valores 0.01 y 0.1. Por lo que al final el valor que utilizaremos para las pruebas finales será **0.1**.

Finalmente, el último parámetro a optimizar es el que determina la máxima diferencia que puede existir entre dos transformaciones consecutivas.

Analizando las figuras 6.11 y 6.12 podemos concluir que el valor de *fitness* tampoco es en este caso muy variable, existiendo muy poca diferencia entre unos valores u otros. El tiempo de cálculo sufre ligeramente más variación y podemos asumir que un valor alrededor de **0.01** parece ser razonable.

6.2.2. Refinamiento de los parámetros

Una vez analizados los parámetros de forma individual, teniendo en cuenta tan sólo la métrica de error que nos ofrece *PCL* para el algoritmo básico, analizaremos los resultados que se obtienen en una situación de tracking real. Es posible que la configuración de los parámetros dependa en cómo se com-

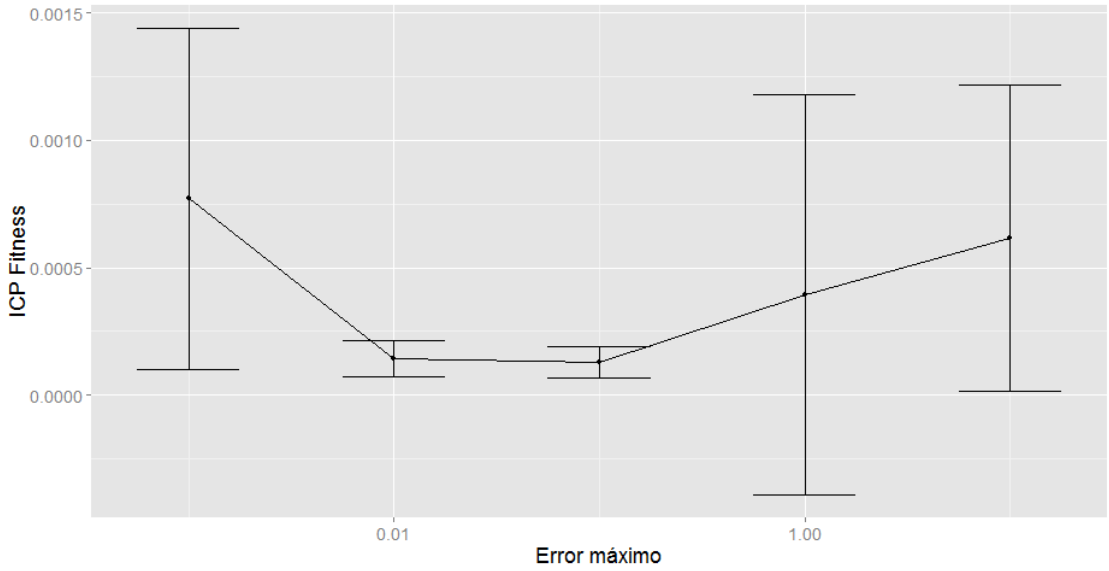


Figura 6.9: Máximo error vs fitness obtenido

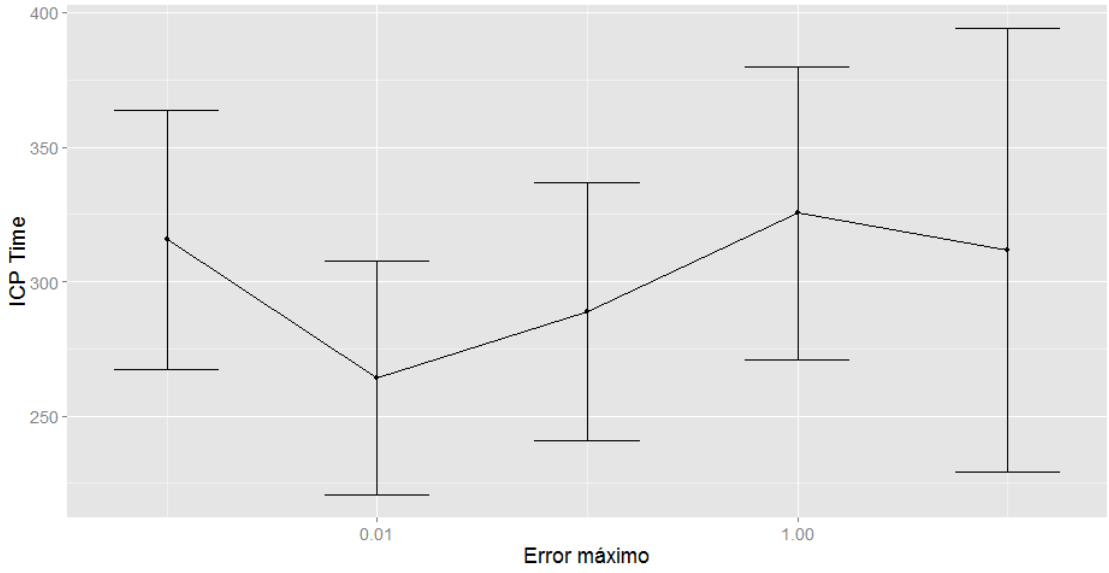


Figura 6.10: Máximo error vs Tiempo de cálculo (ms)

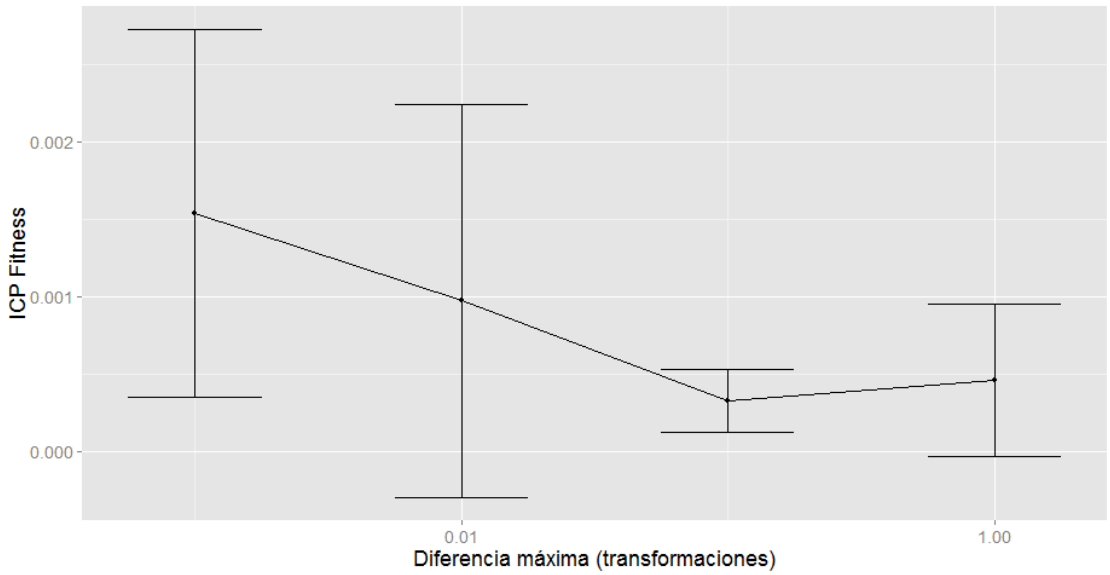


Figura 6.11: Máxima diferencia vs fitness obtenido

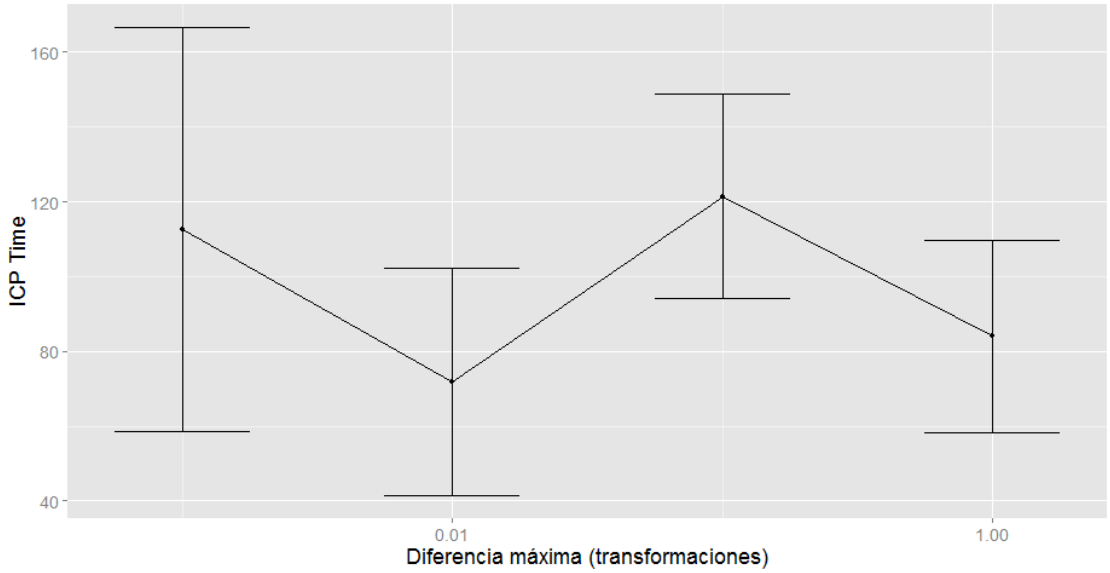


Figura 6.12: Máxima diferencia vs Tiempo de cálculo (ms)

porta a la hora de detectar la orientación de la cabeza, por lo que seguramente se deberán hacer reajustes para mantener un comportamiento estable.
Inicialmente empezaremos con los parámetros definidos anteriormente y analizaremos trazas de ángulos *yaw*, *pitch* y *roll* mientras se realizan movimientos de un lado a otro con la cabeza para ver qué detecta nuestro algoritmo.
El primer experimento se realizará con los parámetros obtenidos anteriormente, es decir, 50 iteraciones máximas, 0.1 como error máximo, una diferencia máxima entre transformaciones de 0.01 y una distancia máxima entre correspondencias de 4 mm. Los resultados se pueden observar en la figura 6.13, e indican que aún y detectándose de forma más o menos robusta, también se afectan los otros ángulos.

Debido a que las interferencias són bastante elevadas utilizando los parámetros mencionados, se realizaron una serie de pruebas cambiando parámetros. Para efectos prácticos, mostraremos tan sólo los resultados del cambio que efectuó una mejora, pero cabe mencionar que se realizaron experimentos con todos los parámetros aunque ninguno de ellos, a parte del parámetro que determina la máxima diferencia entre transformaciones consecutivas, mostró signos de mejora. En la figura 6.14 se puede apreciar como las señales de *pitch* y *roll* no se ve tan afectadas durante el movimiento de la cabeza de lado a lado. Es interesante recordar que el parámetro de diferencia máxima con valor 0.1 ofrecía un *fitness* ligeramente superior (ver figura 6.7). Por lo tanto, se cambiará el valor del parámetro a 0.1 en las pruebas definitivas.

6.2.3. Selección del método de estimación de correspondencias

Tal y como se comenta en la sección 3.3.2), el método por defecto del *ICP* para determinar correspondencias es utilizar el algoritmo de *nearest-to* o *más cercanos*. También se comenta la alternativa de utilizar estimación por proyecciones para, aún aproximando las correspondencias perdiendo un poco de precisión, se acelera notablemente el tiempo de cálculo. En este apartado se experimentará utilizando los dos métodos para decidir cuál de ellos utilizar de forma definitiva.
En la figura 6.15 se puede apreciar que el estimador utilizando proyecciones tiene cierta tendencia a acelerar el tiempo de cálculo aunque sin ser excesivamente notoria su participación. Aún así parece presentar menos variabilidad en el tiempo de cálculo que utilizar el estimador *Nearest*.

6.2.4. Selección de los rechazadores

El último conjunto de parámetros que analizaremos en este proyecto consiste en los rechazadores que utilizaremos para el algoritmo (para más in-

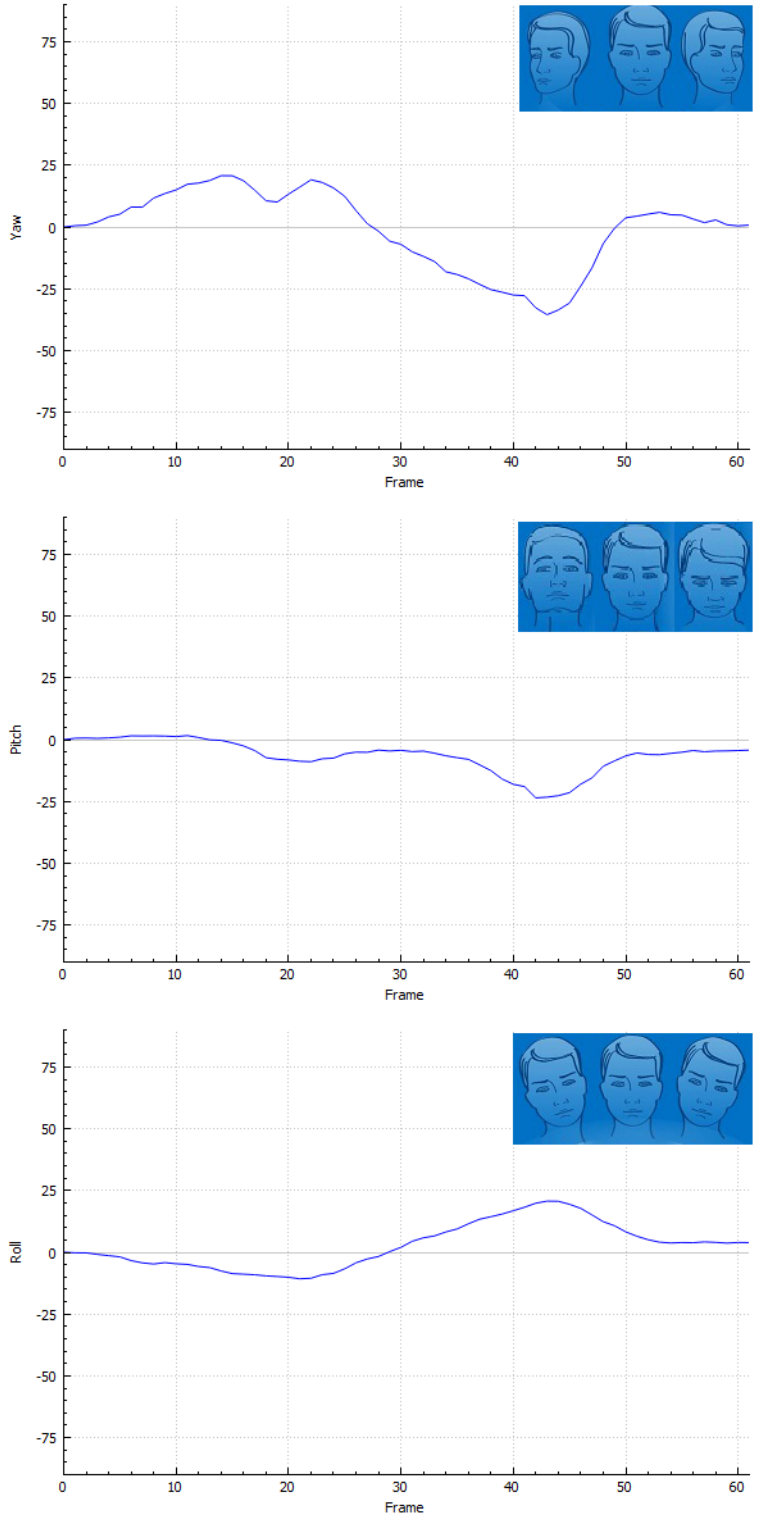


Figura 6.13: 50 iteraciones, max. error 0.1, max. dif. 0.01, distancia corresp 4

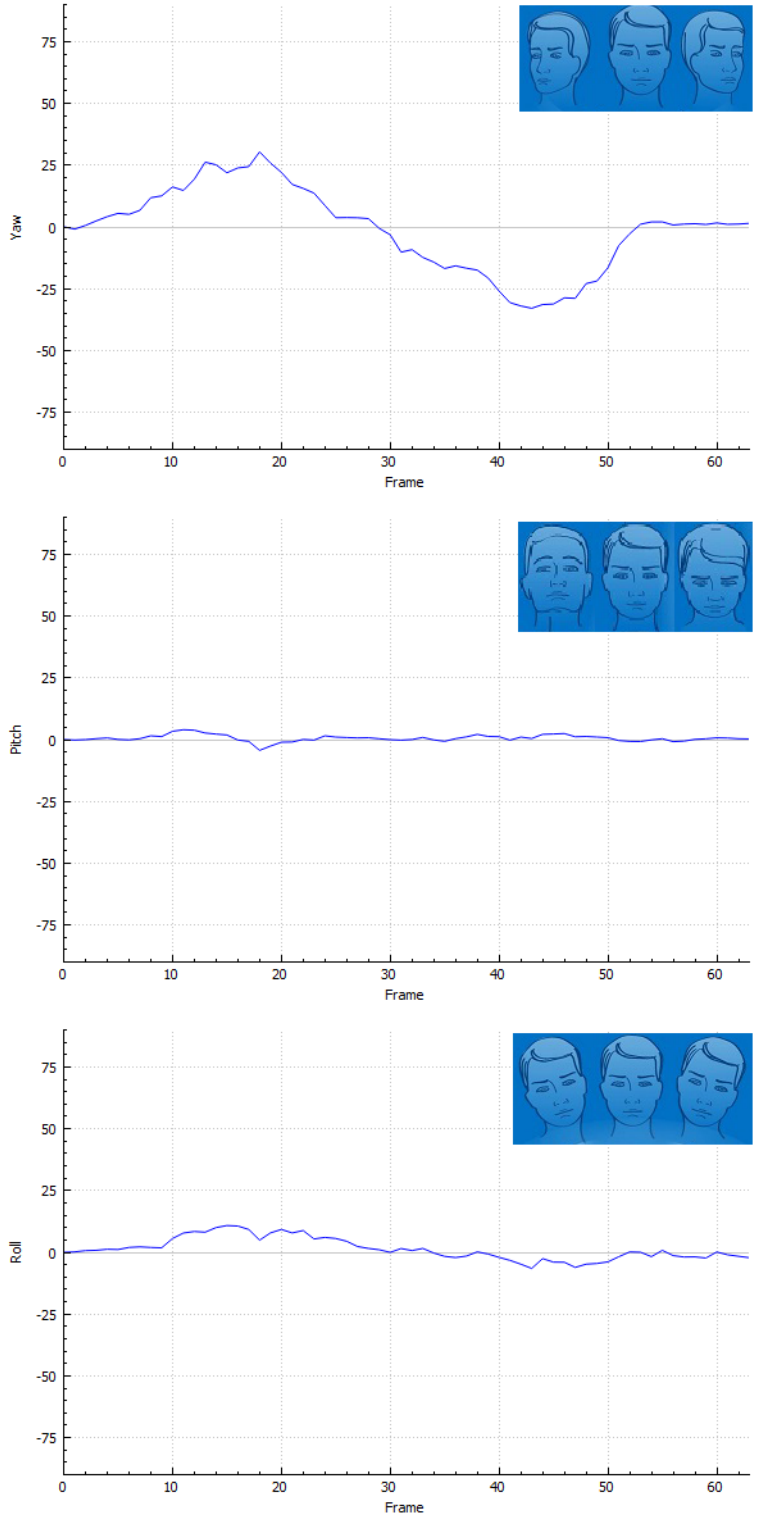


Figura 6.14: 50 iteraciones, max. error 1, max. dif. 0.1, distancia corresp 4mm

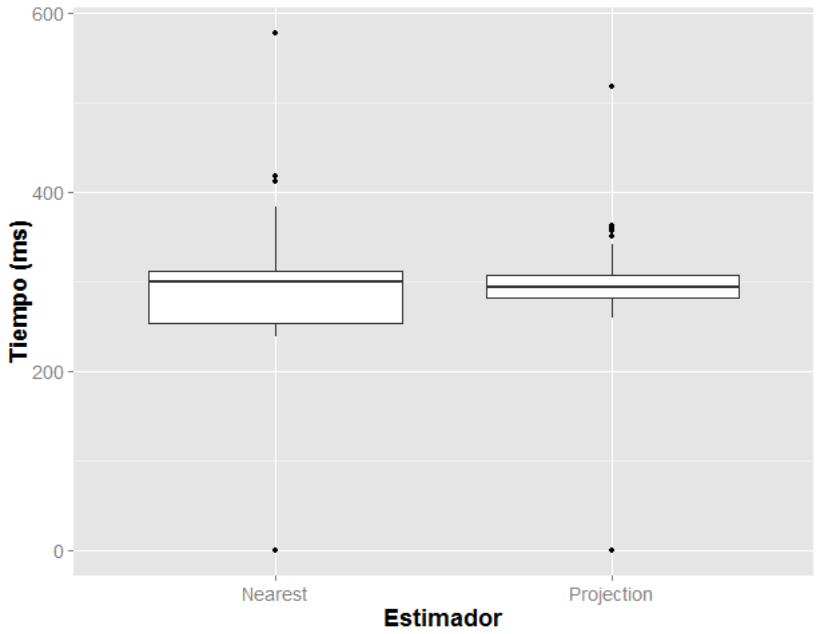


Figura 6.15: Boxplot del tiempo de cálculo del ICP con los dos estimadores

formación sobre los rechazadores consultar la sección 3.3.3). Se ha utilizado la librería *PCL* para implementar diferentes rechazadores y en esta sección se presentarán diferentes experimentos para decidir cuáles de ellos utilizar y cuáles no. La mayoría de los rechazadores no tienen un efecto muy notable en el tiempo de ejecución, por lo que nos limitaremos a comparar su capacidad para mejorar la solución ya obtenida en los apartados anteriores.
Cabe destacar que el rechazador basado en normales no será un rechazador que tengamos en cuenta. Después de realizar ciertos experimentos, se ha considerado inviable su uso ya que para utilizarlo es necesario un paso previo de estimación de normales. Se ha comprobado que es un paso muy costoso que reduce mucho la velocidad de cálculo sin llegar a añadir ningún tipo de mejora.
Dicho esto, el primer rechazador que consideramos es el rechazador recíproco, que rechaza múltiples correspondencias quedándose tan sólo con la pareja de puntos más cercana. En la figura 6.16 se puede apreciar que el hecho de añadir el rechazador recíproco no tiene un efecto realmente apreciable en la detección. Si bien sí parece añadir cierto ruido en la señal de *roll* con respecto a los resultados anteriores (figura 6.14.
Se ha realizado también un experimento utilizando el rechazador de correspondencias que se encuentren en la frontera de la nube de puntos. Aún así, el

experimento resultó ser un fracaso ya que al utilizar este rechazador el *ICP* nunca llegó a converger. Creemos que una de las posibles causas sea que la mayoría de correspondencias se encuentren en la zona de la frontera de la nube, ya que a medida que se gira la cabeza puntos como los ojos o la nariz pueden encontrarse en la frontera.
Por último, experimentaremos con el rechazador adaptable que utiliza la mediana de la distancia como límite. Se puede apreciar en la figura 6.17 que el hecho de aplicar el rechazador adaptable mejora notablemente la detección. Cabe destacar que el uso de éste rechazador añade otra limitación a las distancias de las correspondencias (hasta ahora existía el límite de 4 mm como distancia máxima) añadiendo además un máximo adaptable que depende no sólo de la distancia de un par de puntos, sino de la distancia de todas las correspondencias.

6.2.5. Parámetros finales

Una vez se han realizado todas las pruebas para escoger los diferentes parámetros del algoritmo, en la tabla 9 se puede apreciar un resumen con los parámetros finales escogidos con los cuales se realizarán las pruebas en la plataforma.

Parámetro	**Valor**
Máximas iteraciones	50
Máxima distancia	4 mm
Máximo error euclídeo	0.1
Diferencia máxima entre transformaciones	0.1
Estimación de correspondencias	Proyección
Rechazadores	Median distance (4 mm)

Tabla 9: Parámetros escogidos para el *ICP*

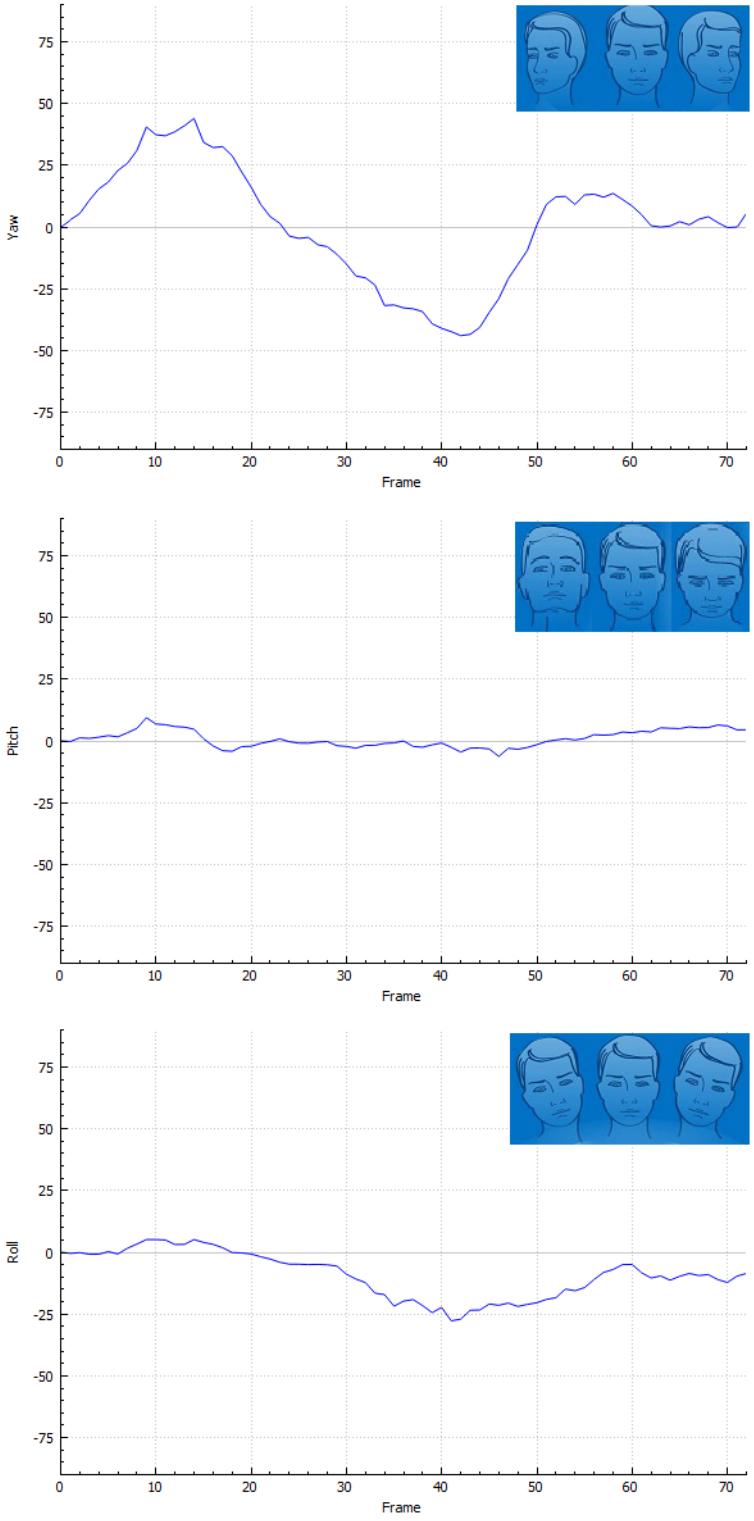


Figura 6.16: Resultados con rechazador recíproco

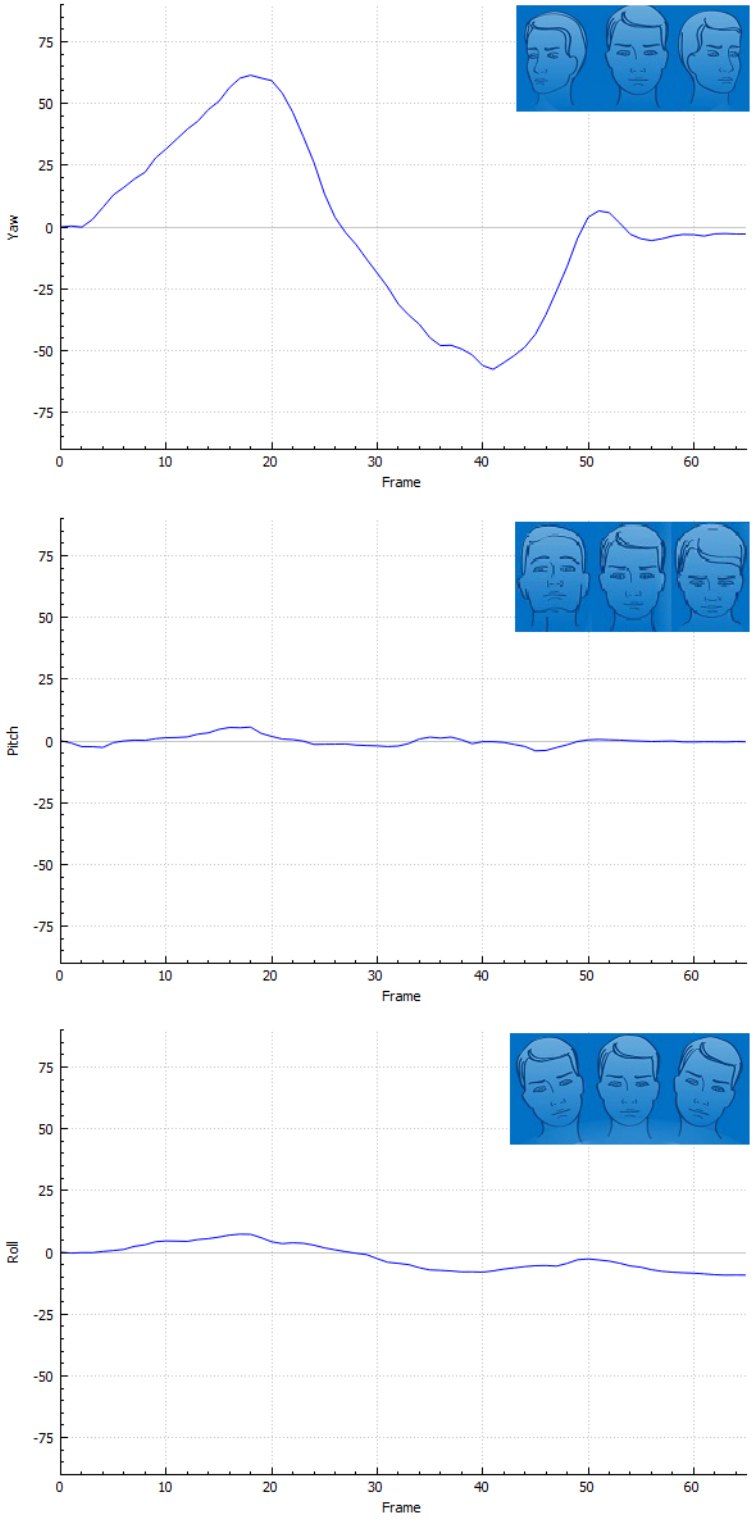


Figura 6.17: Resultados con rechazador de mediana de la distancia: 4 mm

7. Pruebas y resultados

En esta sección se comentarán los resultados obtenidos con las diferentes opciones. Por limitaciones del sistema de pruebas, tan sólo utilizaremos la plataforma externa para comparar el ángulo yaw de la cabeza. En la sección de trabajo futuro se ofrece una mejora en este aspecto. Cabe destacar que la plataforma Kinect 2 tiene una limitación muy importante que nos impedirá utilizar la plataforma externa para realizar las comparaciones con el algoritmo que viene incluido. La limitación consiste en que, para detectar la cabeza, la Kinect 2 primeramente intenta detectar el cuerpo del usuario. Por lo tanto, ya que nuestra plataforma externa no es nada más que un busto, la Kinect 2, al no reconocer extermidades superiores, asume que no se encuentra ante ninguna cabeza. Esta limitación no existe en la Intel Real Sense, que es capaz de detectar la cabeza independientemente de si el cuerpo del usuario es visible o no. Ante esta disruptiva, se decidió que para las comparaciones finales se compararía tan sólo el algoritmo de la *Intel RealSense* con su homólogo utilizando el *Iterative Closest Point*. Se incluirá además una comparativa entre las dos cámaras.

7.1. Algoritmos integrados

En esta sección se analizarán los resultados obtenidos con los algoritmos que nos ofrecen los controladores de las cámaras 3D. Cabe destacar que, por limitaciones de la plataforma Kinect 2 nombradas anteriormente, las comparaciones se han tenido que realizar sin utilizar la plataforma.

7.1.1. Resultados con Intel RealSense

A continuación se describirán los resultados obtenidos con el software integrado en la cámara de Intel, la Real Sense SR300. El movimiento realizado ha sido, primero, un giro de cabeza hacia la derecha. A continuación, se ha realizado un giro completo hacia la izquierda y después se ha vuelto a la posición neutra. Seguidamente, se han realizado los giros en orden inverso, es decir, primero a la izquierda y después a la derecha, para terminar la prueba en una posición neutra. En la figura 7.2 se puede observar la evolución del ángulo yaw mientras se realizaban los movimientos con la cabeza. Se pueden apreciar algunos momentos clave de la detección y fácilmente se pueden extraer diversas conclusiones con la prueba. Primeramente, se pueden comparar las últimas dos imágenes para ver que existe un error a la hora de detectar la cabeza a 0 grados, ya que parece existir un tiempo de retraso entre que la cabeza se encuentra mirando al frente y la detección de la cámara. Además,

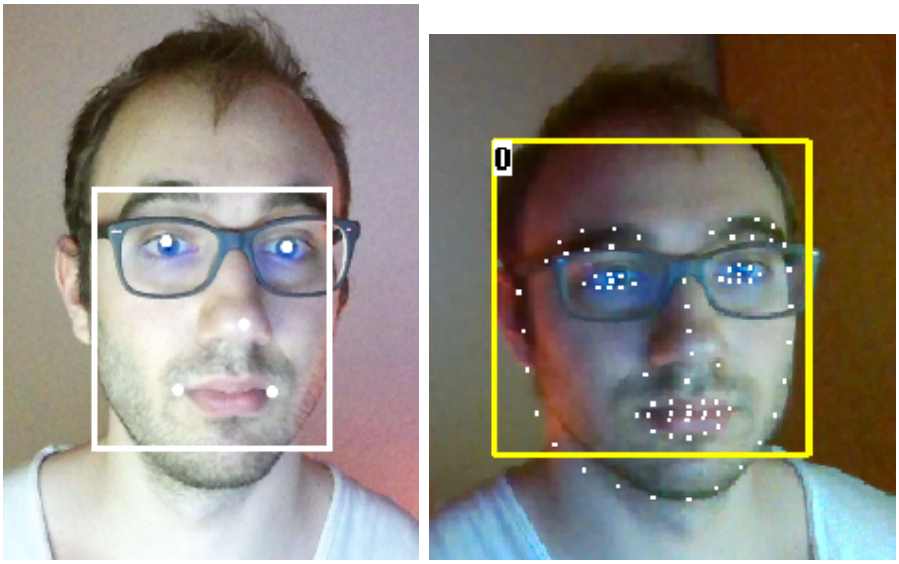


Figura 7.1: Cara detectada por Kinect 2 (izq.). Cara detectada por RealSense (der.)

existe un error de detección cuando se cambia rápidamente de dirección entre las imágenes 4 y 5, ya que aún y detectando que el usuario ha pasado por el estado de 0 grados, la cámara mantiene un ángulo positivo cuando claramente el ángulo es negativo en ese momento. Finalmente, comentar que el primer giro a la izquierda prácticamente no se detecta, dando como resultado ángulos entre los 5 y los 10 grados.

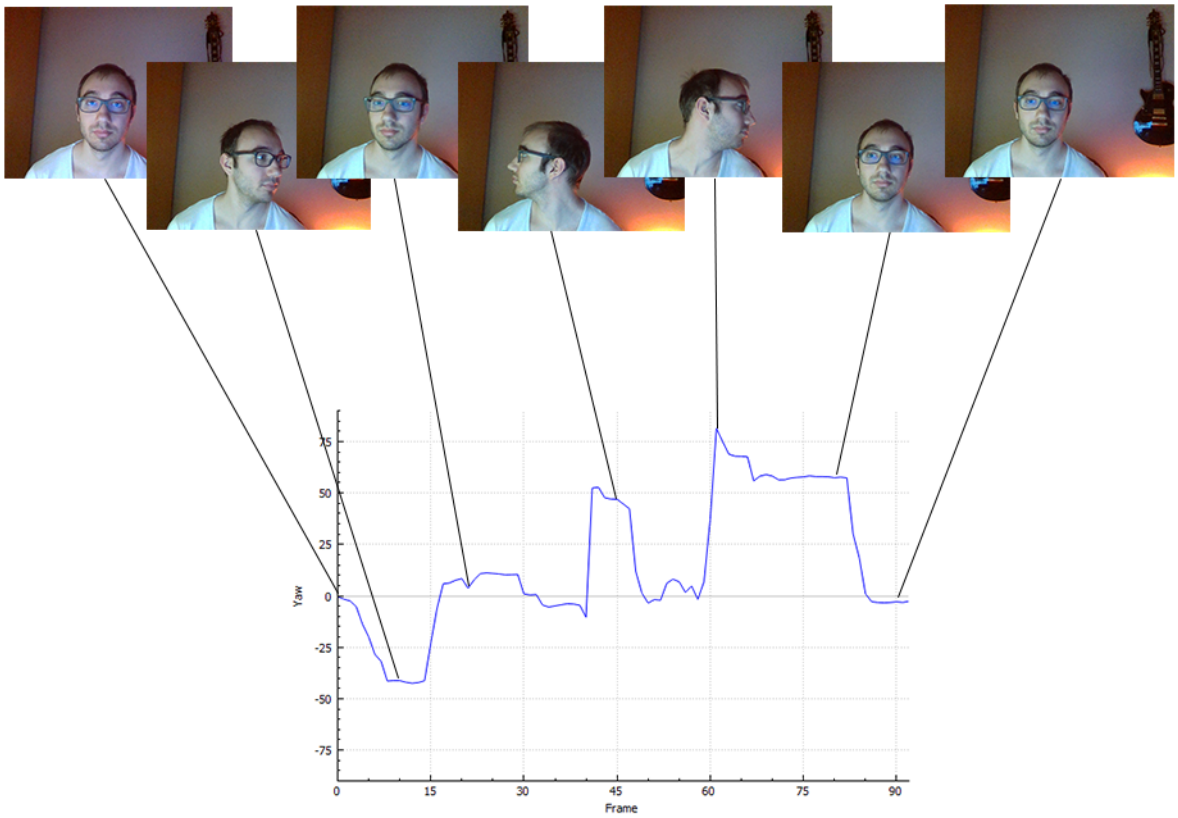


Figura 7.2: Evolución del ángulo yaw con Intel RealSense

7.1.2. Resultados con Kinect 2

A continuación se describirán los resultados obtenidos con el software integrado en la cámara de Microsoft, la Kinect 2. Los movimientos que se han realizado con la cabeza han sido los mismos que los realizados en la prueba de la Intel RealSense. En la figura 7.3 podemos apreciar bastantes diferencias con respecto a la figura 7.2 obtenida con la RealSense. Primeramente, apreciamos cierto retraso en la detección de la cabeza, debido a que primero debe detectar las articulaciones del tronco superior del usuario. A partir de ahí, podemos apreciar un mejor comportamiento con respeco a la Intel RealSense. En este caso, el cambio de signo del ángulo ha sido detectado satisfactoriamente. Creemos que es debido a la mayor cantidad de información que el algoritmo implementado por la Kinect 2 analiza, teniendo información sobre la posición de los hombros y de los brazos. También es interesante comentar el hecho de que la Kinect 2 ha detectado el segundo giro a la izquierda sin mayores

problemas. Es un hecho que en el gráfico obtenido por la Kinect 2 se pueden distinguir claramente los estadios de la prueba, cosa que no se observa en el gráfico obtenido por la Intel RealSense.

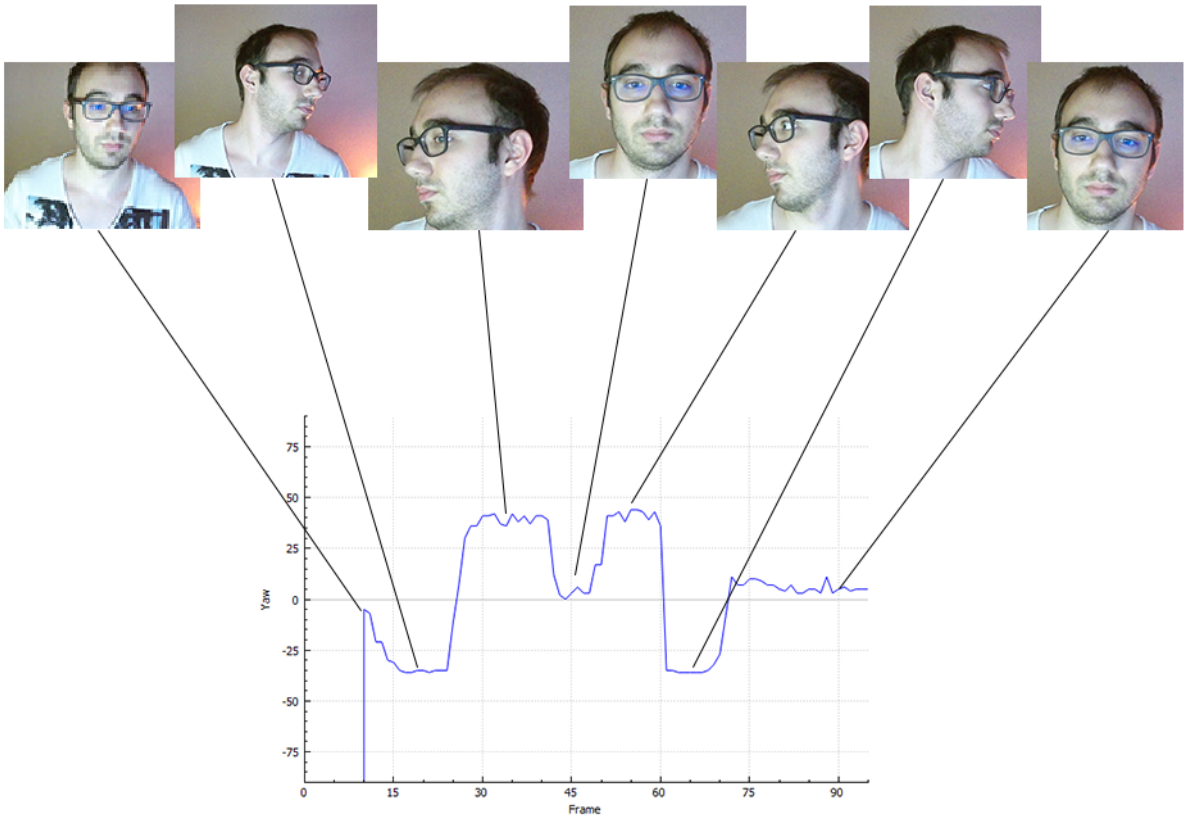


Figura 7.3: Evolución del ángulo yaw con Microsoft Kinect 2

7.1.3. Conclusiones de la comparativa

Para resumir la comparación realizada, podemos interpretar diversos puntos:

- La cámara Kinect 2 ofrece un mejor seguimiento de la cabeza, detectando cambios bruscos de sentido, a costa de requerir la detección del torso.
- La Intel RealSense es capaz de detectar una cabeza sin requerir ver el torso. Aún así, parece ser más propensa a errores cuando el usuario realiza movimientos bruscos, tardando un tiempo en reajustar su detección.

7.2. Comparativa entre ICP y algoritmos integrados

En esta sección se estudiarán los resultados obtenidos con el algoritmo implementado por la Intel RealSense y los obtenidos con el algoritmo de *Iterative Closest Point*. Se utilizará la plataforma externa para obtener unos resultados lo más repetibles posible. Se han realizado pruebas con diferentes amplitudes de movimiento, concretamente amplitudes de -20 a 20 grados y amplitudes de -40 a 40 grados. Para ver el comportamiento de los algoritmos en mayores amplitudes, se ha realizado un experimento utilizando movimiento tan sólo a un lado, llegando hasta los 60 grados.
En las siguientes gráficas, la línea azul representa la señal detectada por el algoritmo, mientras que la señal roja indica qué ángulo se le está indicando al motor.

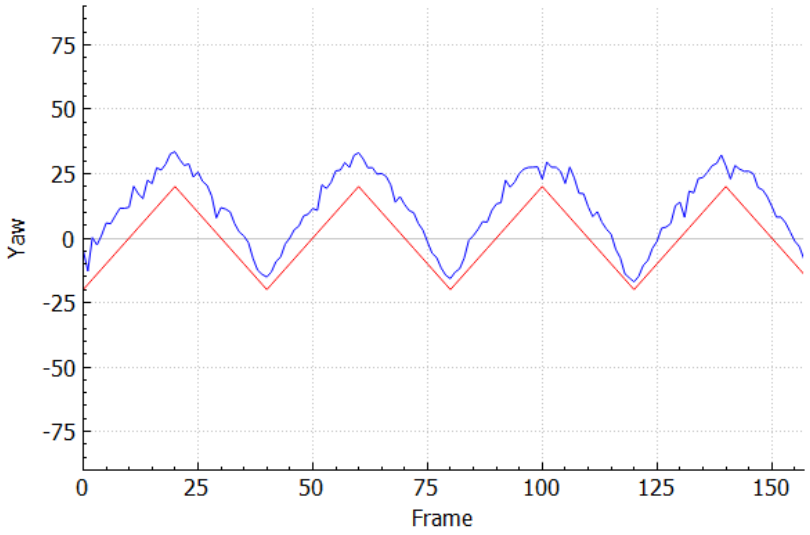


Figura 7.4: Resultados utilizando algoritmo integrado con Intel RealSense - 40 grados de amplitud

Comparando las figuras 7.4 y 7.5 observamos como cambia en los resultados el hecho de utilizar un algoritmo u otro. En el caso de los resultados obtenidos por la cámara, se puede apreciar cierto ruido durante la detección. Sin embargo, a pesar del ruido, la señal obtenida se asemeja bastante a la señal enviada al motor. En cambio, al utilizar el algoritmo *ICP* se puede apreciar que resulta en una señal muchísimo más suave. Nuestra teoría para explicar este efecto consiste en que el detector de la cámara de *Intel* utiliza un algoritmo que no tiene en cuenta los movimientos anteriores de la cabeza, realizando una detección por cada frame. En cambio, nuestro algoritmo

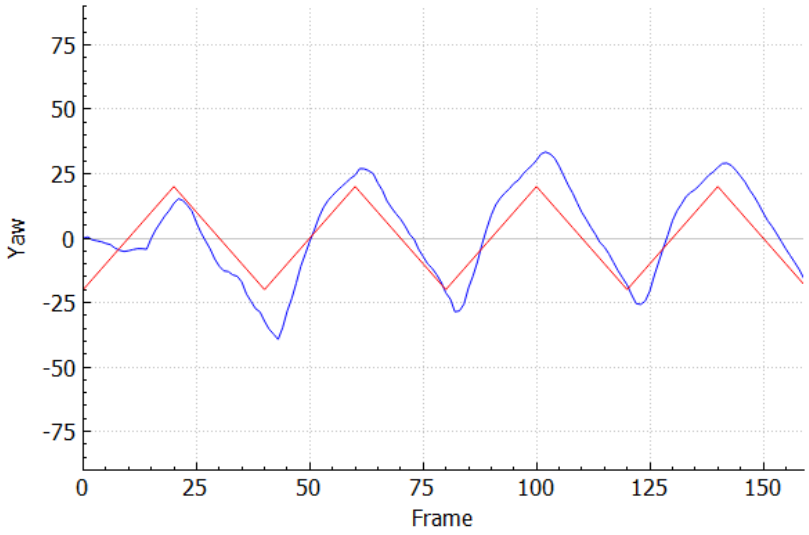


Figura 7.5: Resultados utilizando algoritmo ICP - 40 grados de amplitud

mantiene la transformación realizada hasta el frame anterior, por lo que la detección del ángulo es mucho más continua.
El siguiente experimento se realizó aumentando la amplitud del movimiento para comprobar como se comportaban los algoritmos cuando la cara se encontraba extremadamente inclinada. Los resultados se pueden observar en las figuras 7.6 y 7.7. En el caso del algoritmo de *Intel*, se puede apreciar que durante la primera etapa del experimento hacia un seguimiento muy cercano al real. En algún momento de la rotación, parece ser que el algoritmo pierde la detección de la cabeza y empieza a confundirse, y parece ser que el mismo efecto ocurre al final de la prueba. Sin embargo, durante el resto del experimento parece aproximar bastante bien la amplitud del movimiento, aún y habiéndose generado un desfase desde la pérdida de la cabeza. Parece ser que inclinaciones entre 30 y 50 grados empiezan a ser peligrosas para la detección. Utilizando el algoritmo de *ICP*, de la misma forma que en el experimento anterior parecía mostrar una traza mucho más suave, aquí parece actuar de la misma manera. En ningún momento parece haber un error de no detectar el movimiento pero sí es destacable el hecho de que a partir de los 20 o 25 grados parece tener problemas para realizar una detección robusta.

Se realizó un experimento más realizando movimientos hacia el lado izquierdo, buscando una amplitud más alta. En las figuras 7.8 y 7.9 se pueden observar los resultados. La *Intel RealSense* empieza realizando un seguimiento muy robusto, pero por alguna razón en el segundo pico empieza a fallar, de forma similar que ocurría en el experimento anterior. Nuestra teoría pa-

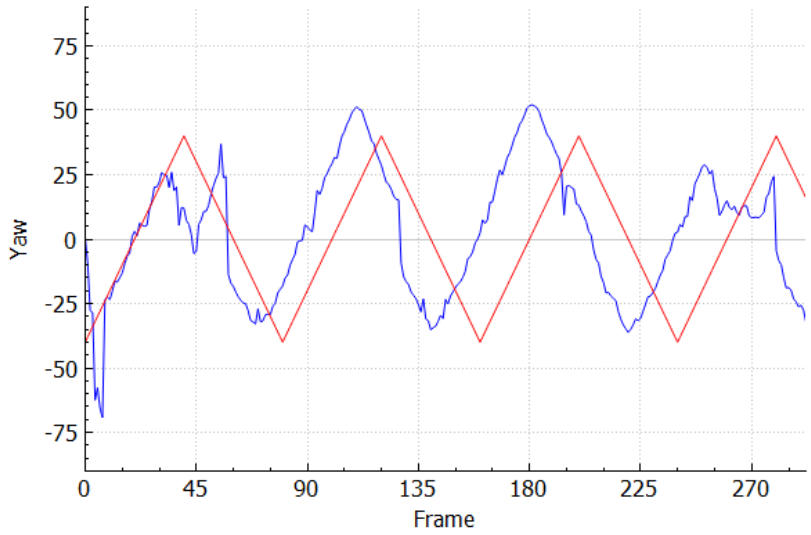


Figura 7.6: Resultados utilizando algoritmo integrado con Intel RealSense - 80 grados de amplitud

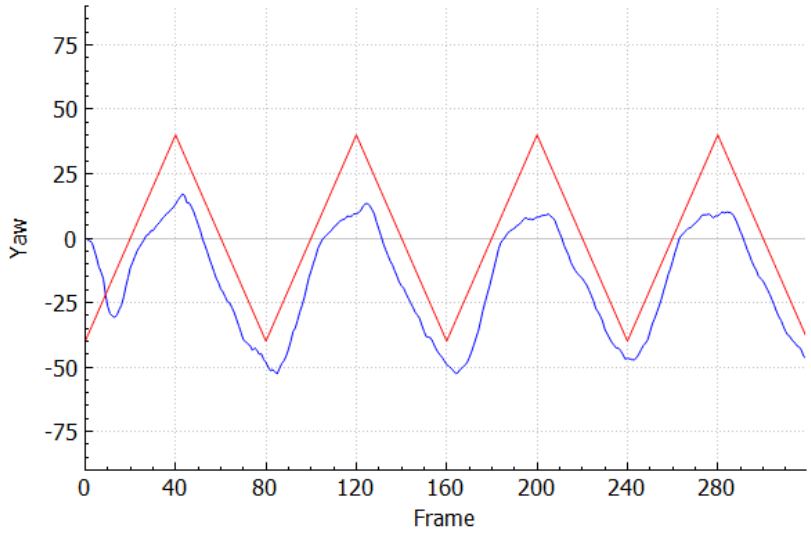


Figura 7.7: Resultados utilizando algoritmo ICP - 40 grados de amplitud

ra explicar este efecto es que para la cámara es imprescindible detectar la cabeza correctamente para estimar su pose. Cosas como sombras o cambios en la iluminación pueden afectar negativamente la detección y por lo tanto dar resultados equivocados. El algoritmo *ICP* es, sin embargo, resistente a condiciones con poca luz ya que se utiliza la información del sensor de profundidad de forma única. Aún y así, en los experimentos realizados con el *ICP* se puede apreciar el mismo problema que en el experimento anterior, ya que a partir de los 25 grados parece perder el seguimiento de la cabeza. La causa más probable es la oclusión parcial que sufre la cabeza cuando el giro es muy acusado. La nariz empieza a tapar una parte de la misma y, como se realiza siempre el registro con la nube de puntos inicial que consiste en la cabeza mirando hacia la cámara, es muy posible que se generen correspondencias erróneas.

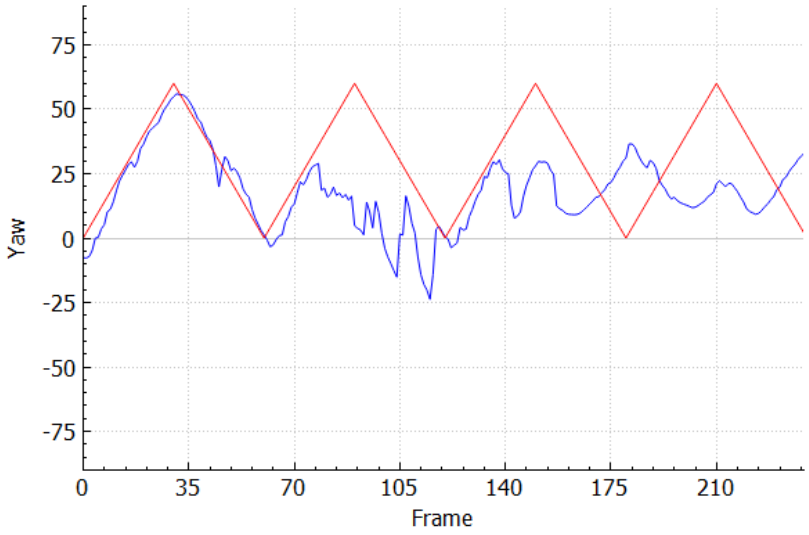


Figura 7.8: Resultados utilizando algoritmo integrado con Intel RealSense - 60 grados de amplitud

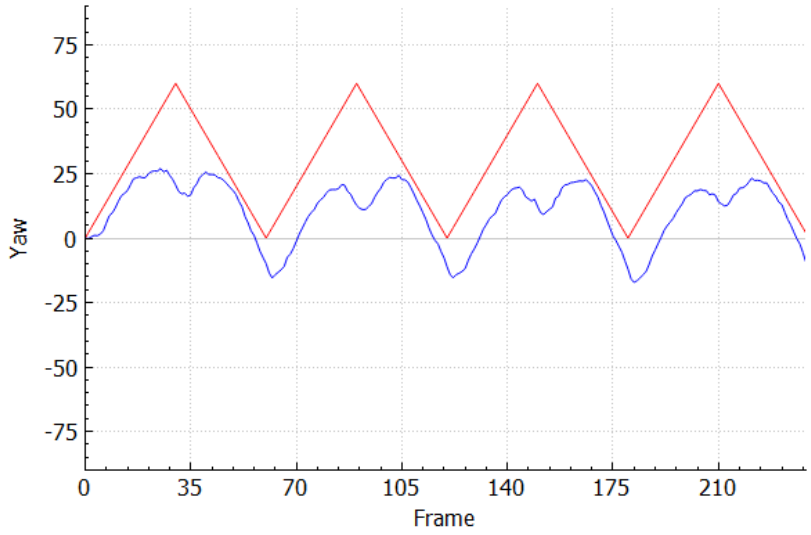


Figura 7.9: Resultados utilizando algoritmo ICP - 60 grados de amplitud

8. Conclusiones finales

Al inicio del documento se presentaron los objetivos del proyecto, que resumidamente eran el desarrollar un sistema de seguimiento facial (concretamente de la orientación de la cabeza) para compararlo a los algoritmos que se encuentran implementados en los kits de desarrollo de las cámaras 3D actualmente en el mercado. También existía el objetivo secundario de desarrollar una plataforma de pruebas sobre la que poder realizar experimentos de forma consistente
Se puede decir que el proyecto ha cumplido con los objetivos propuestos. El sistema que se ha diseñado es capaz de utilizar diferentes cámaras 3D y de realizar test de forma sencilla conectándose a la plataforma externa. Además, se ha realizado un estudio profundo del algoritmo *Iterative Closest Point* que puede permitir además aplicar conocimientos a otros proyectos.

El proceso de desarrollo no ha sido extremadamente complejo exceptuando por las etapas de adaptación a los *frameworks* de los controladores de las cámaras. La mayoría de código que podría haber sido extremadamente complejo y la implementación de estructuras de datos no estándar ha sido proporcionado por la librería *Point Cloud Library*, simplificando muchísimo el proceso de desarrollo y permitiendo implementar más opciones. El uso de la librería ha generado también la necesidad de realizar experimentos para determinar el mejor conjunto de parámetros y opciones para el algoritmo

utilizado.

Para resumir, el resultado del proyecto ha sido satisfactorio ya que se han conseguido cumplir tanto el objetivo primario como el objetivo secundario propuestos, obteniendo como resultado un algoritmo que realiza un seguimiento facial muy aceptable y una plataforma que puede extenderse fácilmente y reutilizarse para otros proyectos.

9. Trabajo futuro

Los objetivos del proyecto han sido alcanzados exitosamente. Sin embargo, aún existe un amplio margen de mejora del mismo. Se pueden comentar diversos aspectos del proyecto que, con más tiempo, se podrían mejorar para obtener aún mejores resultados:

- Mejoras en la plataforma de pruebas. Actualmente la plataforma externa consiste en un aparato muy básico, poco funcional y fácilmente mejorable. Sería interesante introducir técnicas de impresión 3D para diseñar una plataforma más robusta, adaptada a las necesidades del proyecto, y incorporando el hardware (controlador Arduino) en el propio cuerpo del aparato.

- Añadir grados de libertad a la plataforma de pruebas. Las pruebas se han realizado exclusivamente en ángulo *yaw* ya que la plataforma giratoria no es capaz de realizar más movimientos. Una posible idea para mejorar este aspecto sería el hecho de añadir más motores para construir un sistema con 3 ángulos de rotación.

- Realizar más experimentos. Aún y habiendo obtenido unos resultados razonables, creemos que aún existe un amplio margen de mejora en el algoritmo. Sería muy interesante seguir realizando pruebas añadiendo más características al *ICP*, experimentando con mayores rangos de valores.

- Añadir sistemas de detección o de predicción. Sería muy interesante añadir algún sistema de predicción para ayudar al algoritmo *ICP*. Un sistema capaz de, analizando los movimientos realizados por el usuario, predecir qué movimiento es más probable que realice durante los próximos frames para intentar aproximar la transformación que utilizará el algoritmo.

- Mejorar el filtrado de los puntos de forma aún más inteligente. No sería descabellado utilizar algoritmos de visión por computador para extraer puntos de la cabeza que podrían tanto acelerar como mejorar la detección de su orientación. Puntos como la posición de los ojos, la nariz o la barbilla podrían utilizarse para seleccionar qué puntos utilizaremos para el *ICP*.

Appendices

A. Especificaciones de dispositivo: Intel Real Sense

La cámara Intel RealSense SR300 es un dispositivo capaz de capturar imágenes tanto de color como de profundidad. Dispone de 2 micrófonos para capturar sonido.

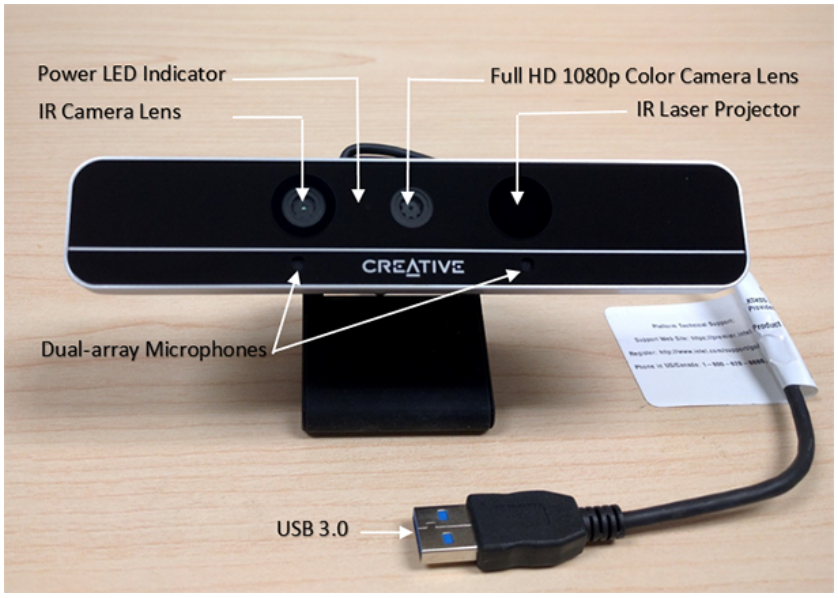


Figura A.1: Dispositivo Intel RealSense SR300

El sensor RGB es capaz de captar imágenes a color a una resolución de 1080p a 30 FPS o bien a una resolución de 720p a 60 FPS. El sensor de profundidad es capaz de capturar imágenes a una resolución de 640 x 480 píxels.

El Intel RealSense SDK for Windows contiene los controladores necesarios para utilizar el sensor Intel RealSense SR300. El soporte para este modelo se ofrece a partir de la versión 2016 R1 en adelante. El software ofrece capacidad para detectar las articulaciones de la mano y detectar puntos de interés en la cara del usuario, como las pupilas.

Característica	**Especificación**
Distancia	0.2m a 1.2m
Resolución Color	1920x1090 (30FPS) 640x480(60FPS)
Resolución Profundidad	640x480
Resolución IR	640x480
Audio	2-mic array
USB	3.0

Tabla 10: Especificaciones Intel Real Sense SR300

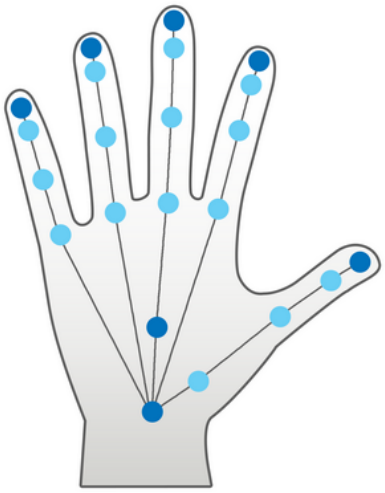


Figura A.2: Articulaciones de la mano

B. Especificaciones de dispositivo: Microsoft Kinect 2

La cámara Kinect es un dispositivo de captura que permite obtener información tanto de color como de profundidad. Además también dispone de 4 micrófonos para capturar sonido, con capacidad de localizar el origen del mismo.

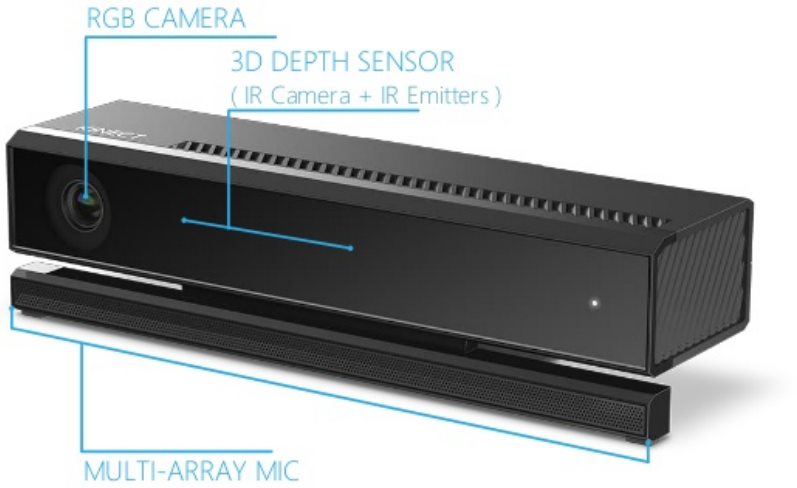


Figura B.1: Dispositivo Kinect 2

El sensor RGB es capaz de captar imágenes a color a una resolución de 1080p, con una frecuencia de 30Hz en condiciones de iluminación óptimas. Además el sensor de profundidad es capaz de captar imágenes a una resolución de 512 x 424 píxels.

Característica	**Especificación**
Distancia	0.4m a 4.5m
Resolución Color	1920x1090
Resolución Profundidad	512x424
Resolución IR	512x424
Audio	4-mic array
USB	3.0

Tabla 11: Especificaciones Kinect 2

El Microsoft Kinect SDK contiene los controladores necesarios para utilizar el sensor Kinect 2 en Windows. El software ofrece la capacidad de detectar hasta 6 personas y 25 articulaciones por persona, incluyendo articulaciones para los dedos de las manos. A diferencia de su versión anterior, la Kinect, el SDK ya no distingue el uso del sensor entre un uso sentado y un uso de cuerpo completo.

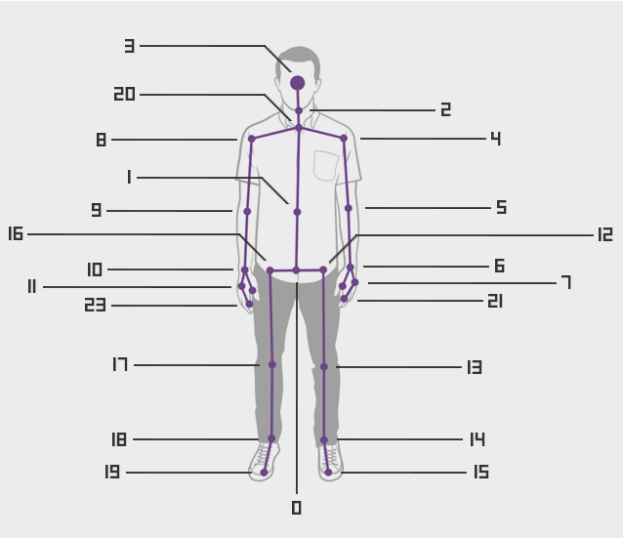


Figura B.2: Articulaciones detectadas por la Kinect 2

C. Leyes y regulaciones

En esta sección se comentarán las leyes aplicables al proyecto y las regulaciones que se deben tener en cuenta para el desarrollo del mismo.

C.1. Licencias de software

Durante el desarrollo del proyecto se han utilizado diferentes licencias de software:

- **Microsoft Visual Studio 2013:** Licencia gratuita para estudiantes de la UPC siempre que sea para uso no comercial.
- **Intel Real Sense SDK:** Derecho de uso del SDK con los dispositivos de Intel.
- **Kinect SDK:** Derecho de uso del SDK con los dispositivos de Microsoft.
- **Qt:** GNU General Public License versión 3 y GNU Lesser General Public License 2.1
- **OpenGL:** Librería OpenSource. Se trata de una licencia de software libre B.
- **Arduino (Hardware):** Licencia Creative Commons Attribution Share-Alike. Permite uso tanto commercial como personal mientras haya crédito.
- **Arduino (Software):** Software OpenSource. Licenciado bajo GPL para el código fuente escrito en Java y licenciado bajo LGPL para las librerías de C/C++.
- **LaTeX:** Project Public License (version 1.3c).

C.2. Legislación

La parte más importante del proyecto consiste en la detección de la pose de la cabeza del usuario. Actualmente no existen leyes que impidan que un software pueda detectar ésta información siempre con el consentimiento del usuario. Lo más parecido que existe son leyes de protección de datos sobre software de reconocimiento facial, pero éste es un tema que en nuestro caso no se aplica ya que nuestro software en ningún momento reconoce la persona que se encuentra frente a la cámara, tan sólo reconoce la existencia de una cara.

Además, por la forma en la que se ha planteado el proyecto, no es necesaria la colaboración de personas para realizar la validación y la verificación del software, ya que se dispone de la plataforma implementada en *Arduino* para esa finalidad.

D. Glosario de abreviaturas

- **3D:** 3 Dimensiones
- **RGB:** Red Green Blue
- **RGBD:** Red Green Blue Depth
- **ICP:** Iterative Closest Point
- **PCL:** Point Cloud Library
- **SDK:** Software Developtment Kit
- **GPU:** Graphics Processing Unit
- **CPU:** Central Processing Unit

Referencias

[1] U. Weidenbacher, G. Layher, P. Bayerl, and H. Neumann, *Perception and Interactive Technologies: International Tutorial and Research Workshop, PIT 2006 Kloster Irsee, Germany, June 19-21, 2006. Proceedings*, ch. Detection of Head Pose and Gaze Direction for Human-Computer Interaction, pp. 9–19. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.

[2] S. Li, K. N. Ngan, and L. Sheng, "A head pose tracking system using rgb-d camera," in *Proceedings of the 9th International Conference on Computer Vision Systems*, ICVS'13, (Berlin, Heidelberg), pp. 153–162, Springer-Verlag, 2013.

[3] OpenGL, "The industry's foundation for high performance graphics." URL: `https://www.opengl.org/documentation/current_version/`.

[4] Microsoft, "Direct3d." URL: `https://msdn.microsoft.com/en-us/library/windows/desktop/hh309466(v=vs.85).aspx`.

[5] S. Rusinkiewicz and M. Levoy, "Efficient variants of the icp algorithm," in *3-D Digital Imaging and Modeling*, pp. 145–152, IEEE, 2001.

[6] D. Holz, A. E. Ichim, F. Tombari, R. B. Rusu, and S. Behnke, "Registration with the point cloud library: A modular framework for aligning in 3-d," *IEEE Robotics Automation Magazine*, vol. 22, pp. 110–124, Dec 2015.

[7] A. Segal, D. Haehnel, and S. Thrun, "Generalized-icp.," in *Robotics: Science and Systems*, vol. 2, 2009.

[8] S. W. Thomas, "Vii.1 - decomposing a matrix into simple transformations," in *Graphics Gems II* (J. Arvo, ed.), pp. 320 – 323, San Diego: Morgan Kaufmann, 1991.

[9] Arduino, "Arduino - introduction." URL: `https://www.arduino.cc/en/Guide/Introduction`.

[10] Wiring, "Wiring homepage." URL: `http://wiring.org.co/`.

[11] Arduino, "Arduino - serial." URL: `https://www.arduino.cc/en/Reference/Serial`.

[12] Adafruit, "Adafruit motor/stepper/servo shield for arduino v2 kit - v2.3." URL: `https://www.adafruit.com/product/1438`.

[13] PCL, "Point cloud library (pcl): pcl::uniformsampling class template reference." URL: http://docs.pointclouds.org/1.7.1/classpcl_1_1_uniform_sampling.html#details.

[14] P. J. Besl and H. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, pp. 239–256, Feb 1992.

[15] J. L. Blanco, "Iterative Closest Point (ICP) and other matching algorithms | MRPT." URL: http://www.mrpt.org/Iterative_Closest_Point_(ICP)_and_other_matching_algorithms.

[16] PCL, "Documentation - point cloud library (pcl)." URL: http://pointclouds.org/documentation/.

[17] Arduino, "Arduino - reference." URL: https://www.arduino.cc/en/Reference/HomePage.

Índice de tablas

Índice de figuras

Índice de códigos