Aplicación basada en imágenes RGBD para obtención de medidas de ajuste para ciclistas

Proyecto Fin de Carrera

Daniel Arbelo Cabrera

Tutor: José Javier Lorenzo Navarro

Tutor: Modesto Fernando Castrillón Santana

Las Palmas de Gran Canaria, julio de 2017

Agradecimientos

Quizás la parte más personal y menos importante de la memoria. Tengo tanta gente a la que agradecer haber llegado hasta este punto que no sé por dónde empezar. Por supuesto estaría mal no mentar primero a mis tutores: Javier con su infinita paciencia y buen humor y Modesto, que siempre ha encontrado gracioso que sea de Teror.

Por otro lado me sentiría mal si no agradeciese a los compañeros que han estado siempre ahí: Aitor, Alberto (al que siempre me referiré como mi "esposa"), a esa pareja inseparable Mayca y Edu, Yazaida (no te debería decir cuándo leo), a David por ser mi pato de goma personal y a Geco, aunque a él no tengo claro por qué, pero gracias Geco.

Y a ti Ale, porque somos idiotas y lo dejamos todo para el final.

También agradecer a Fer y a Cris, que pese a no ser parte de la carrera han mostrado siempre su apoyo y han intentado (con escaso efecto) que mi proyecto tenga un aspecto visual más agradable.

Me gustaría agradecer a Pedro Marín por el interés que ha mostrado con mi proyecto y la ayuda que ha prestado al mismo (siempre he bromeado que él es co-tutor).

A mi primo y a la gente que entrena conmigo, que siempre me han mirado como un loco cuando hablo de mi proyecto, pero que me escuchasen me ayudaba.

Por último, y especialmente, a mi familia: mis padres y mi hermana, que me han apoyado en todo momento y, de forma que no entiendo, aún no me han matado.

Índice general

Índice de figuras

Capítulo 1

Introducción

En los últimos años el aumento de deportistas en nuestra sociedad es un hecho más que evidente, las competiciones tanto para profesionales como para aficionados llegan a varias cada fin de semana. A su vez implica que las personas aumentan su nivel de entrenamiento a varias sesiones a la semana.

Dentro de este grupo de deportistas una gran sector se han centrado en buscar la eficiencia dentro de sus ejercicios, buscando el rendimiento óptimo en cada sesión de entreno y además, evitar posibles lesiones que puede conllevar el entrenamiento.

Aprovechando esta situación las aplicaciones de biomecánica han sufrido un aumento de la demanda y del interés dentro de este sector. En este ámbito se centra este proyecto, en el que se trata de procesar imágenes en RGB-D para hacer un estudio de biomecánica con gente pedaleando.

1.1. Objetivos del proyecto

Con este proyecto se pretende desarrollar una aplicación que sea capaz de llevar a cabo la grabación de una persona pedaleando y el seguimiento de los movimientos de dicha persona. Todo esto con el fin de poder sacar los ángulos que forman las articulaciones mientras se una persona pedalea, la información que dichos ángulos transmiten a un usuario experto permiten discernir si la

posición del sillín (tanto en altura como en distancia con respecto al manillar) es la correcta para el paciente sobre el cual en ese momento se está realizando el estudio de biomecánica.

Más adelante se verá con detalle el proceso, pero por lo pronto se puede descomponer en diferentes objetivos a cumplir:

- *Elección de las herramientas de desarrollo.* El punto de partida deberá ser, ante el problema propuesto, la elección de las herramientas con las que se llevará a cabo la resolución del mismo.
- *Familiarización con el entorno.* Una vez elegidas las herramientas (tal y como se verá en el capítulo 3) es necesario empezar a tener contacto con las mismas, familiarizarse con ellas y prepararlas para el transcurso del desarrollo.
- *Recopilación de secuencias de test.* El siguiente paso, una vez configurado todo el entorno de desarrollo, es la recopilación de una secuencia para su procesado.
- *Tratamiento de imágenes.* En este punto se trata el procesado inicial de las imágenes: primero se hará una segmentación de las mismas basada en la profundidad para luego hacer una detección y seguimiento de las marcas. Por último se obtendrán las estadísticas pertinentes.
- *Diseño e implementación de la interfaz* Se realizará una interfaz para que el usuario final (un experto en biomecánica) pueda usar la aplicación.
- *Pruebas* Se llevarán a cabo diferentes tests que prueben el correcto funcionamiento de la aplicación desarrollada.

Capítulo 2

Estado del arte

En este apartado se procederá a buscar las distintas herramientas existentes que llevan a cabo una función similar a la que pretende alcanzar este proyecto. Para cada una de ellas se tendrá en cuenta sus características y sus puntos débiles. Así mismo se ha decidido separar en dos grupos:

1. Métodos de longitud de piernas.
2. Métodos basados en el ángulo de la rodilla

Aunque este proyecto está dentro del segundo grupo, debido a su importancia y su extendido uso se enumerarán varios métodos del primero.

2.1. Métodos de longitud de piernas

La búsqueda de una altura óptima a la hora de pedalear no es ninguna novedad, ya sea por razones de comodidad, evitar lesiones o aprovechamiento máximo de la fuerza al realizar el ejercicio. Hace ya bastantes años existen diferentes métodos para la búsqueda de la posición correcta.

Existen varios métodos clásicos que indican una aproximación más o menos fiable a la hora de elegir la posición correcta del sillín, entre estos métodos se encuentra el de la *sabiduría popular* que indica como punto óptimo que

Cuadro 2.1: Tabla método Greg Lemond

Entrepierna	**Distancia**	**Entrepierna**	**Distancia**	**Entrepierna**	**Distancia**
55	48.565	71	62.693	87	76.821
56	49.448	72	63,576	88	77.704
57	50.331	73	64,459	89	78.587
58	51,214	74	65.342	90	79.47
59	52,097	75	66.225	91	80.353
60	52.98	76	66.225	92	81.236
61	53.863	77	67.108	93	82.119
62	54.746	78	67.991	94	83.002
63	55.629	79	68.874	95	83.885
64	56.512	80	69.757	96	84.768
65	57.395	81	70.64	97	85.651
66	58.278	82	71.523	98	86.534
67	59.161	83	72.406	99	87.417
68	60.044	84	73.289	100	88.3
69	60.927	85	74.172	101	89.183
70	61.81	86	75.938	102	90.066

la posición en altura sea dos dedos por debajo de la cadera y que la distancia entre manillar y sillín sea suficiente como para que quepa tu mano y tu antebrazo entre ambos. Es evidente que este método es poco fiable y quizás sea solamente válido para uso ocasional de la bicicleta, aquellas personas que deseen dedicarse de un modo más profesional o que vayan a darle un uso con una intensidad media-alta necesitan un método más fiable.

Dentro de los métodos que podríamos considerar tradicionales, hay una pequeña variedad que se centran en determinar la altura en base a la altura de las extremidades inferiores[1]. Dentro de este subgrupo se encuentra *el método Hamley y Thomas* en el cual se determina que la altura del asiento viene determinada tras medir la altura de la entrepierna y multiplicar dicho valor por un factor 1.09.

También forma parte de este subconjunto *el método Greg LeMond* (en honor al célebre ciclista), bastante similar al anteriormente descrito y que en este caso se basa en medir la longitud de las piernas, desde la entrepierna al suelo tal y como se ve en la figura 2.1. A continuación se multiplicará este valor por un factor de 0.883 como muestra la tabla 2.1. Este método está basado en la experiencia empírica del corredor, e implica cierto nivel de riesgo lesivo en las rodillas ya que el ángulo de flexión de la articulación será mayor.

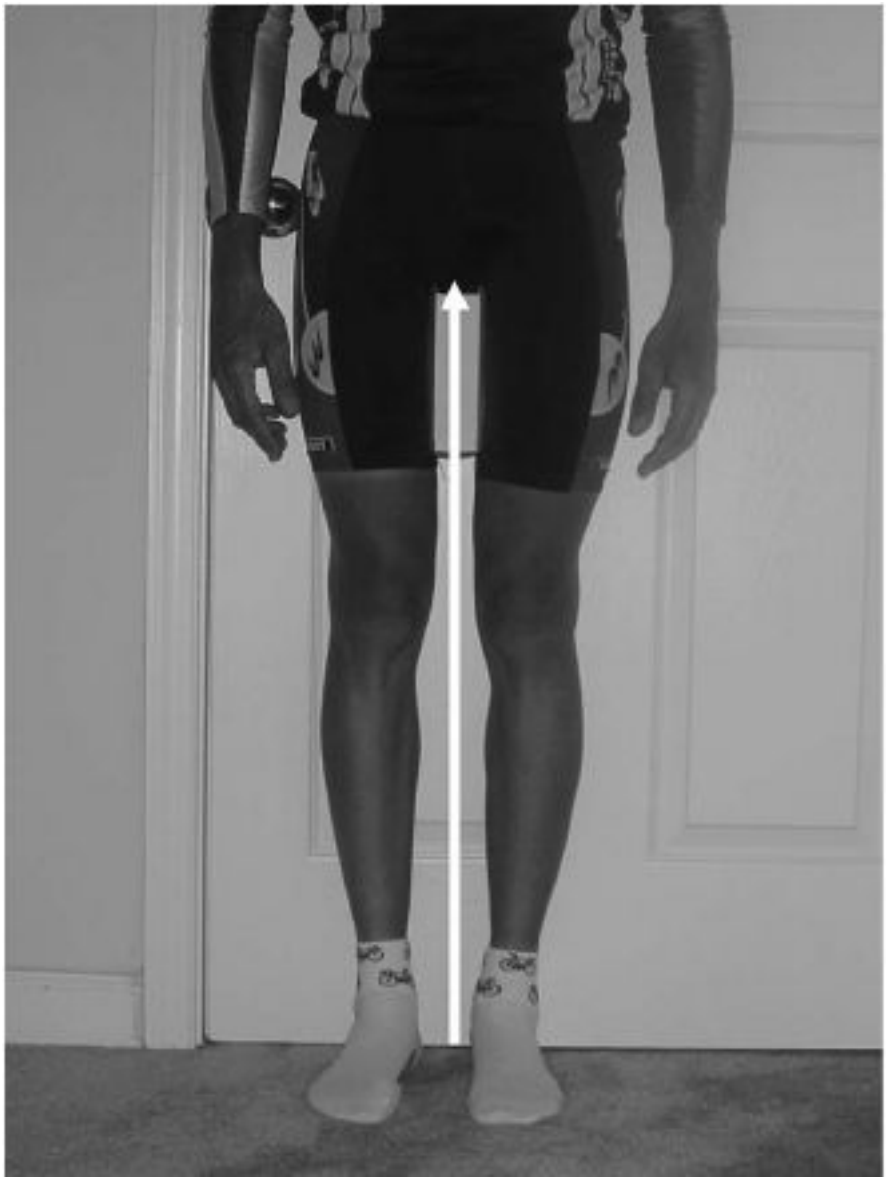

Figura 2.1: Toma de medidas del método Lemond. Fuente ResearhGate

El *método del talón* uno de los más comunes y simples, cuando el ciclista está sentado sobre el sillín, la rodilla debe extenderse y con el talón en el pedal, la biela debe estar alineada con el tubo del sillín.

El *método de la longitud trocantérica* se basa en la distancia del hueso más prominente de la cadera (el trocánter mayor) con respecto al suelo, se establece una relación 1:1.

2.2. Métodos basados en el ángulo de la rodilla

Los métodos basados en ángulos son la base sobre la que se sustentan la mayoría de las aplicaciones de biomecánica para ciclistas, todos parten del *métodos de Holmes*, que se basa en ubicar la altura del sillín con el ciclista su pie en el punto muerto inferior (entiéndase a las 6 en un reloj), estableciendo un rango de 25 a 35 grados. Un ángulo mayor de 35º, facilitará la ocurrencia de lesiones con dolor anterior de rodilla y un ángulo menor a 25º facilitará la ocurrencia del Síndrome de fricción de la banda iliotibial y la tendinitis bicipital.

Dentro de este tipo subconjunto de métodos se encuentran las aplicaciones de biomecánica que usan la tecnología para obtener la información necesaria para que cada ciclista tenga una pedalada óptima. A continuación analizaremos algunas de las disponibles.

2.2.1. Bike Fast Fit

Se trata de una aplicación disponible en exclusiva de la *App Store* para *IPhone e IPad* que permite el cálculo de los ángulos entre articulaciones. Para ello, tal y cómo se puede apreciar en la figura 2.2 se necesita grabar un vídeo del ciclista en un rodillo mientras con unas marcas adhesivas en en los puntos críticos para la medición.

El seguimiento de los puntos no es automático, es necesario que el experto, tras grabar la sesión, una las articulaciones usando las marcas como guía visual para hacerlo de forma correcta.

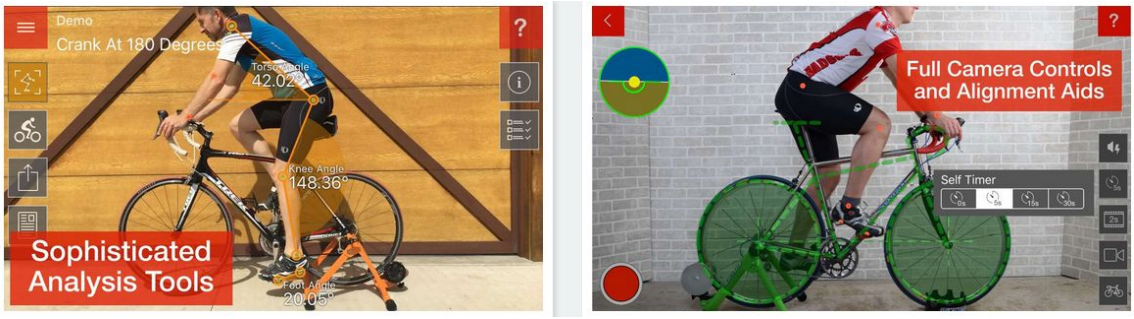

Figura 2.2: App Bike Fit Fast. Imágenes tomadas de la web oficial.

Entre sus aspectos interesantes presenta la capacidad de crear informes con todas las mediciones que se han obtenido durante la sesión y además permite compartir esta información mediante correo electrónico y aplicaciones en la nube como *Apple Icloud, Dropbox y Google Drive*.

Presenta la comodidad de que solamente necesita un dispositivo móvil de la marca *Apple* y un trípode para estar completamente operativo. La aplicación tiene un coste de 5,49 euros, siendo bastante asequible, sin embargo habría que añadirle el coste del dispositivo móvil que varían entre 400 y 1,000 euros (dependiendo del modelo y capacidad de almacenamiento elegidas).

2.2.2. BikeFit by STT Systems

Destacada como una de las aplicaciones más completas del mercado viene amparada por la marca deportiva BH, BikeFit es, en este momento, una de los puntos de referencias en relación con los estudios de biomecánica en bicicleta.

En el mercado desde finales del 2016 tiene un funcionamiento similar al ya presentado: un ciclista sobre un rodillo estático pedalea mientras se hace la grabación del video para posteriormente hacer la identificación y seguimiento de las marcas que, en este caso, se tratan de luces leds captadas con una cámara de alta velocidad. Tras la grabación de la sesión, el experto en biomecánica podrá ajustar la identificación de las marcas para una mayor precisión de las mediciones.

Ofrece una gran variedad de datos, no solo los ángulos que forman las

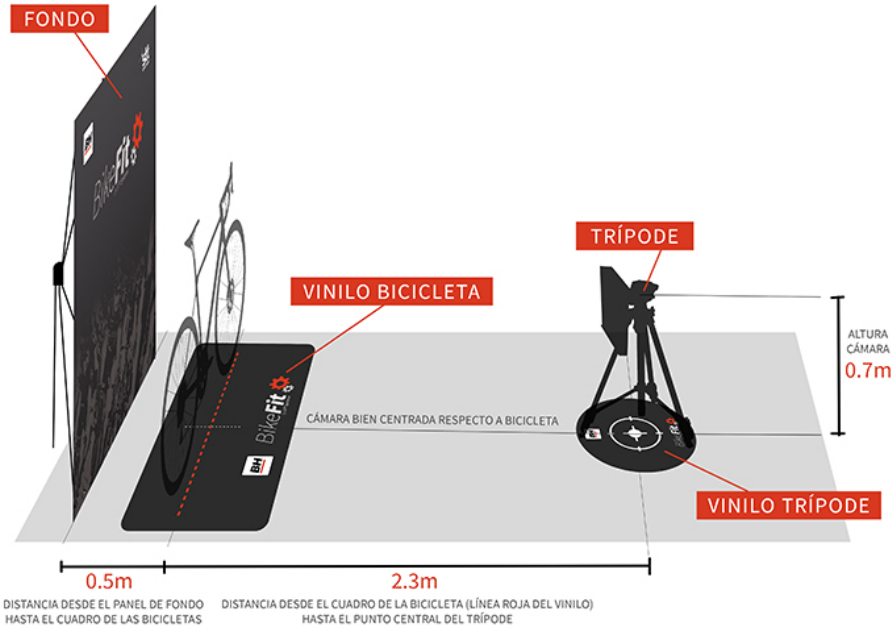

Figura 2.3: BikeFit by STT. Distribución del espacio.

articulaciones y va indicando en cada fotograma si los valores son óptimos o no. El programa posee una base de datos con los distintos modelos de cuadros de bicicletas de la marca BH clasificadas por tipos (carretera, montaña, triatlón, etc) y tamaño del cuadro, con esto consiguen unas mediciones aún más precisas. El programa ofrece consejos automáticos cuando se encuentra algún valor que está de los parámetros adecuados. Además una vez acabada la sesión permite emitir un informe con todos los datos adquiridos, para posterior consulta.

Uno de sus puntos negativos tal y como se puede apreciar en la figura 2.3, está el espacio necesario para el despliegue del equipo de biomecánica usado. Se recomiendan al menos cinco metros cuadrados de espacio para usar de forma eficiente el sistema.

Por otro lado nos encontramos ante un sistema que, para pequeñas empresas, puede resultar caro: aparte del ordenador personal y del trípode, requiere la adquisición de una cámara de alta velocidad cuyo precio suele ser de varios miles de euros.

2.3. Que desea aportar este proyecto

Tras ver las distintas posibles elecciones que existen en el mercado es interesante valorar que puede aportar el proyecto que se quiere desarrollar. Ante los métodos de longitud de piernas el primer paso es automatizar el proceso y el segundo, quizás más importante, dotarlo de una mayor precisión. Pese a que estos métodos tienen una tasa de efectividad aceptable no son ideales para un deportista que entrene de forma regular o intensiva.

Por otro lado aplicaciones como **Bike Fast Fit** carecen de una detección automática de las marcas, objetivo a cumplir en el desarrollo de esta aplicación.

Frente a herramientas como **BikeFit by STT Systems** se ofrece, lo primero, una herramienta asequible y lo segundo una mayor facilidad de uso.

Además ninguna de estas herramientas ofrece una representación del movimiento del deportista en un modelo 3D. La aplicación a desarrollar tratará de alcanzar esta meta y así tener una característica única y diferenciadora.

Capítulo 3

Recursos

En este apartado se describen los recursos utilizados para el desarrollo del proyecto, este abarca tanto la parte física (*hardware*) como la parte digital (*software*).

3.1. Recursos Hardware

3.1.1. Cámara Kinect

Como se ha descrito con anterioridad el recurso que se decidió usar para la obtención de imágenes fue la cámara/sensor de profundidad Kinect de Microsoft.

Una de las principales razones de la elección de este dispositivo de grabación frente a otros es que, probablemente, sea el más extendido entre todos los modelos RGB-D del mercado, contando con una comunidad activa a día de hoy, a su vez dispone de una abundante cantidad de tutoriales y guías de uso, por lo que facilita su aprendizaje y la posibilidad de obtener resultados satisfactorios en un plazo de tiempo bastante reducido.

Como se puede apreciar en las figura 3.1 y 3.2, este dispositivo posee una cámara RGB, un sensor de profundidad y un micrófono multi-array bidireccional que, en unión, permite la captura de imágenes y movimientos de

Figura 3.1: Cámara Kinect de Microsoft. Imagen tomada de[Wikipedia]

los cuerpos en tres dimensiones y también concede la posibilidad de realizar reconocimiento facial y de comando de voz.

Tal y cómo se ve en la figura 3.3 en condiciones óptimas (según el fabricante) la lente tiene un ángulo de visión (FOV) de 58º grados horizontales y 45º verticales. Además es posible aumentar el ángulo de FOV en 27º (tanto hacia arriba como hacia abajo) gracias a un pivote que incorpora la cámara.

Durante el transcurso del proyecto Microsoft terminó el desarrollo de la segunda versión de la cámara Kinect, diseñada para su nueva consola Xbox One. El salto tecnológico entre ambas versiones del hardware, la adquisición de un modelo de la 2.0 junto al estado intermedio de desarrollo del proyecto conllevó la valoración de adaptar el trabajo llevado a cabo hasta ese momento al nuevo modelo de cámara.

Como se puede ver en la Figura 3.4 la segunda iteración de la cámara es bastante más potente: las imágenes en RGB ahora están disponibles en alta definición, las imágenes en profundidad pasan de una resolución de 320x240 a 512x424, y lo que es más importante, además ahora tenemos la posibilidad de capturar imágenes en el espectro infrarrojo. En general la revisión del hardware mejora en todos los aspectos:

- Aumento del FoV horizontal. Antes teníamos 57º ahora 70.
- Aumento del FoV vertical, antes de 43 ahora hasta 60.
- Aumenta la detección de articulaciones de hasta 26, frente a 20.

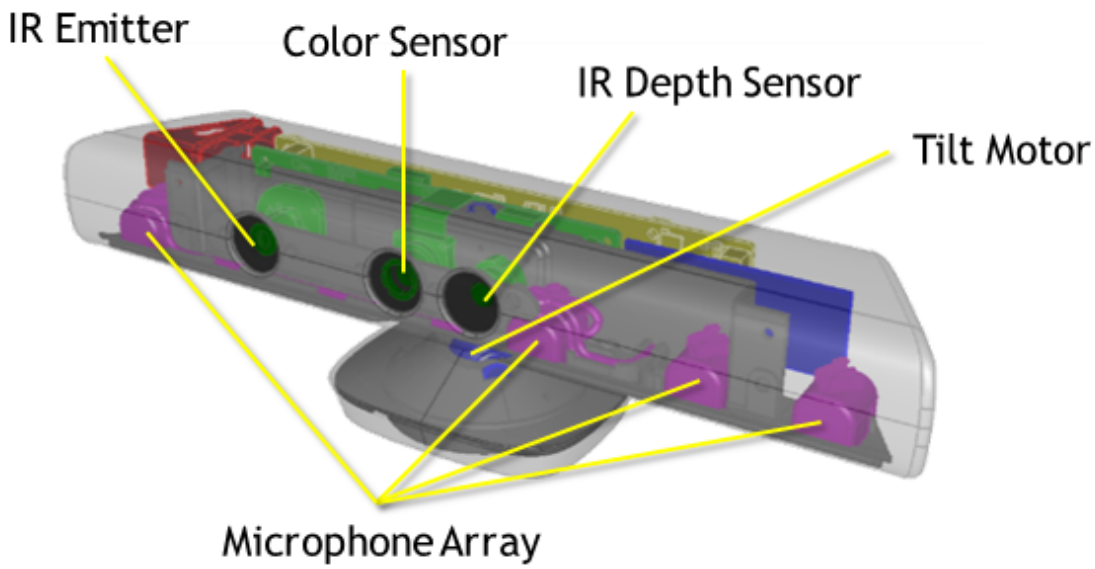

Figura 3.2: Diseño interno Cámara Kinect de Microsoft. Imagen tomada de[MSDN]

- Es capaz de reconocer hasta 6 personas a la vez (antes 2).
- Reconocimiento de gestos faciales y de las manos.

En contraposición tiene un aumento de requisitos:

- El primer modelo era compatible con USB 2.0, el nuevo requiere de forma obligatoria 3.0
- Solamente es compatible con Windows 8 en adelante, la versión original podía usarse con Windows 7 en adelante.

Su coste de venta al público a día de hoy no es comparable, el primer modelo se encuentra descatalogado. Sin embargo, si miramos su precio de venta original se puede apreciar que para su uso en ordenador es exactamente el mismo: 200 euros en total. En ambos casos trae el adaptador necesario para usarlo en ordenador.

Uno de los puntos más importantes a la hora de comparar ambas cámaras es la manera que tienen de calcular la profundidad. El modelo original emite

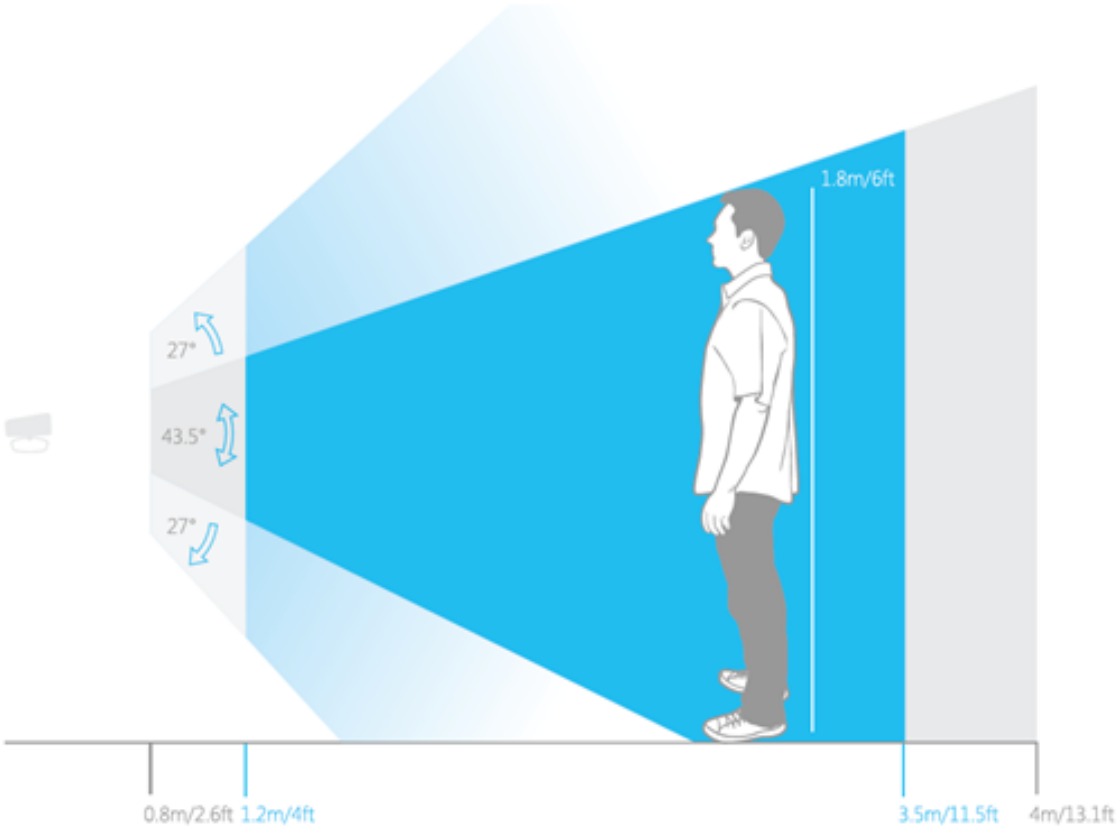

Figura 3.3: Diseño interno Cámara Kinect de Microsoft. Imagen tomada de[MSDN]

Feature	Kinect for Windows 1	Kinect for Windows 2
Color Camera	640 x 480 @30 fps	1920 x 1080 @30 fps
Depth Camera	320 x 240	512 x 424
Max Depth Distance	~4.5 M	~4.5 M
Min Depth Distance	40 cm in near mode	50 cm
Horizontal Field of View	57 degrees	70 degrees
Vertical Field of View	43 degrees	60 degrees
Tilt Motor	yes	no
Skeleton Joints Defined	20 joints	26 joints
Full Skeletons Tracked	2	6
USB Standard	2.0	3.0
Supported OS	Win 7, Win 8	Win 8
Price	$299	TBD

Figura 3.4: Comparativa de Kinect v1 frente a v2. Fuente [Zugara.com]

un haz láser infrarrojo que proyecta un patrón de puntos sobre los cuerpos cuya distancia quiere ser determinada, para luego ser captado por una cámara infrarroja y, mediante hardware, calcula la profundidad de cada punto. Este método tiene la inconveniencia de que no permite usar varias cámaras a la vez debido a que interfieren en las señales emitidas entre ellas. Sin embargo la versión 2.0 utiliza una nueva tecnología denominada *Time-of-flight*, que consiste en emitir haces de rayos infrarrojos y calculando el tiempo que tardan en volver tras impactar en la superficie cuya distancia se está calculando. Este método es bastante más estable, preciso y robusto frente a las interferencias.

Cuadro 3.1: Equipos usados

	Portátil	**Sobremesa**
S.O	Windows 7	Windows 10 Profesional
CPU	Intel i7 2600U	Intel i5 4600K
RAM	8 GB DDR3	16GB DDR3
HDD	360GB	2 TB
Pantalla	15,6”	27”

3.1.2. Estaciones de trabajo

Inicialmente el proyecto se empezó a desarrollar en una máquina personal (portátil), tras la valoración y aceptación del cambio de modelo de cámara se debió usar una segunda estación de apoyo que cumpliese los requisitos mínimos que Microsoft indicaba para el uso del dispositivo Kinect 2.0, en este caso, el ordenador inicial no disponía de puerto USB 3.0 compatible, dicho problema se solventó usando un ordenador personal de mesa que cumplía dicho requisito. Las características de cada equipo puede verse en el Cuadro 3.1

3.2. Software

Una vez determinado el hardware el siguiente paso lógico sería escoger el software que más se adecuase al desarrollo del trabajo, dentro de este ámbito lo primero sería determinar que bibliotecas de software serían necesarias para llevar a cabo la tarea.

Para el tratamiento de imágenes se escogió la biblioteca OpenCV (Open Computer Vision), en la que se ahondará con mayor detalle en la sección 3.2.1. Es importante recalcar que la elección de esta biblioteca condiciona, casi en su totalidad, el resto de las elecciones dentro del ámbito del software.

Como entorno de desarrollo se ha escogido Visual Studio 2010 de Microsoft, posteriormente y tras el cambio producido con el dispositivo Kinect 2.0, fue obligatorio migrar a una versión más moderna del famoso compilador, en concreto se ha escogido la versión Visual Studio 2015[2].

Para el diseño de una de las interfaces y para la animación de un modelo 3D se ha escogido el motor de videojuegos Unity en su versión 5.5, tal y como se verá en el apartado 5.6.1.

El desarrollo de la memoria se decidió realizarlo en el lenguaje LaTex[3] usando la herramienta online *Overleaf*.

3.2.1. Biblioteca OpenCV

OpenCV es una biblioteca de código libre centrada en la visión por computación que proporciona al usuario más de 500 algoritmos optimizados para el manejo de imágenes y vídeos. Su historia comienza con una versión en estado Alfa que data de 1999, y no es hasta el año 2006 que la primera versión estable llega a disposición de los usuarios. Esta biblioteca está escrita en C y C++, lenguajes de alto nivel con una gran base de usuarios.

En el año 2009 llega la segunda versión de esta biblioteca, entre los cambios más importantes se encuentran en el interfaz de C++, con el objetivo de que fuese más fácil de usar, más segura (con respecto a la memoria a la hora de programar) y en general una mejor optimización de las funciones ya existentes.

En 2015 aparece la versión 3.0 de OpenCV, que convive hasta día de hoy con la versión 2.4, en esta versión aparecen compatibilidades con el lenguaje interpretrado *Python* y se establece también una base en *Java* y *MATLAB/OCTAVE*. Aparte se preparan *wrappers* (añadir al glosario) para *CSharp*, *Perl*, *CH*, *Haskell* y *Ruby*. Ante tantas posibilidades a la hora de programar la aplicación se ha elegido *C++* como opción, debido a que era la base de la biblioteca en si, su facilidad de uso.

Esta biblioteca se distribuye bajo licencia BSD, permitiendo su uso con fines comerciales o de investigación dentro de los límites de la misma. Obviamente esto ha permitido su rápida expansión y uso en el ámbito de tratamiento y procesado de imágenes y videos.

Para llevar a cabo el proyecto se ha hecho un uso intensivo de las funciones que proporciona OpenCV para poder alcanzar las metas propuestas: eliminación de fondo y detección de marcas.

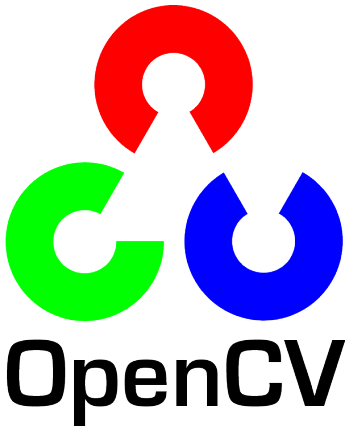

Figura 3.5: Logo de la biblioteca OpenCV

3.2.2. Visual Studio

Microsoft Visual Studio es un entorno de desarrollo integrado (sus siglas en inglés corresponden a IDE) para sistemas operativos Windows. Soporta múltiples lenguajes de programación entre los que se encuentran C++ y C#.

Desde el año 2005 Microsoft ofrece de forma gratuita la *Ediciones Express*, que consisten en versiones básicas separadas por lenguajes de programación o plataforma enfocadas a estudiantes y programadores que estén iniciándose.

Este entorno de desarrollo se ajusta a la perfección a las exigencias del proyecto a desarrollar: acepta todos los lenguajes de programación asociados a OpenCV, tiene guías de como integrar la biblioteca a un proyecto (enlace?) y Kinect está fuertemente integrado dentro del IDE.

Otro de los puntos fuertes es que es bastante usado por la comunidad, mirando estadísticas en el foro Stack Overflow, podemos observar que las dudas relacionadas con IDE, sin coste y que estén relacionadas son OpenCV son:

- Unas 15 preguntas para *CodeLite*
- Cerca de 3,283 preguntas para *Code::Blocks*
- Más de 13,317 preguntas relacionadas con *Visual Studio*.

Figura 3.6: Logo de Visual Studio 2013

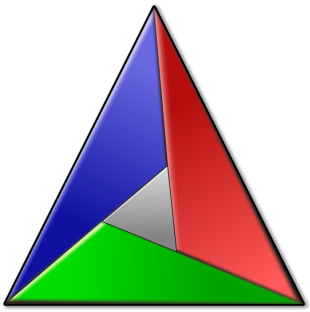

Figura 3.7: Logo de CMake

Para poder integrar de forma correcta OpenCV en el entorno de programación fue necesario generar el código para integrarlo en Visual Studio. Para ello fue necesario usar la harramienta CMake (punto 3.2.3).

3.2.3. CMake

Es una herramienta multiplataforma de generación o automatización de código. Su nombre proviene de la abreviatura para "Cross Platform Make". Al igual que OpenCV se trata de una una herramienta bajo licencia BSD.

Su uso es necesario para integrar el código fuente en Windows y por tanto, en proyectos con Visual Studio.

3.2.4. Unity

La herramienta Unity consiste en un motor de videojuegos multiplataforma (disponible en sistemas operativos Windows, Linux y OS X) que a su vez también dispone de opción de compilación multiplataforma (incluyendo consolas y dispositivos móviles). Para el desarrollo del proyecto se ha usado la versión 5.5 disponible desde marzo del 2015.

Unity posee un tipo de licencia denominada *Unity Personal* que posee todas las prestaciones del motor original con la única restricción de compilar con una pantalla inicial con el logo de Unity y la leyenda *Made with Unity*. En caso de que dicha aplicación tenga destino comercial existe un tope de 100.000 dólares, tras lo cual se deberá o adquirir una licencia Pro o tener una subscripción a la Plus. Teniendo en cuenta la naturaleza docente sin ánimo de lucro del proyecto, Unity se ajustaba perfectamente a las necesidades del mismo.

Para programar en Unity se usa como lenguaje de programación C# y tiene compatibilidadad con programas de diseño tales como Blender, 3ds Max, Maya, etc. Por otro lado permite elegir Visual Studio como el editor de código de los proyectos desarrollados en esta plataforma.

Unity además ofrece una tienda de recursos online denominada *Unity Asset Store*, en la que se puede acceder a contenido diverso para los diferentes proyectos, entre otros componentes se pueden encontrar: modelos 3D, materiales, texturas, sistemas de partículas, tutoriales, etc. Hay que destacar que los paquetes en dicho recurso pueden ser tanto de pago como gratuitos y están sujetos a la licencia de uso de Unity.

3.2.5. StarUML

Los diagramas elaborados en las etapas de análisis y diseño del proyecto han sido elaboradas en el lenguaje UML, para ello se ha hecho uso de la herramienta StarUML.

StarUML es una herramienta de software libre de edición UML, es compatible con la mayoría de los tipos especificados en el diagrama de UML 2.0 y fue desarrollada en Delphi.

Figura 3.8: Logo de Unity. Fuente [Wikipedia]

3.2.6. Irfanview

Irfanview es un programa de edición de imágenes gratuito para el uso privado no comercial, ha sido la herramienta básica para la edición de imágenes en este proyecto y para la memoria del mismo.

3.2.7. LaTex

Para el desarrollo de este documento se ha utilizado como herramienta LaTex, que es un sistema de composición de textos, orientado a la creación de documentos escritos que presenten una alta calidad tipográfica. El editor usado para dicha tarea es la herramienta en línea *Overleaf*, entre sus ventajas podemos encontrar que no necesita ser instalada y que permite ver en tiempo real el resultado del texto escrito.

Se trata de un software libre bajo licencia LPPL (Licencia Pública del Proyecto LaTeX).

Capítulo 4

Planificación

4.1. Metodología

La metodología desde el punto de vista de la ingeniería del software escogida para desarrollar el proyecto ha sido el Modelo Evolutivo en Espiral tal y como está definida en Boehm, 1988[4]. Este modelo tiene especial cuidado con el riesgo que aparece a la hora de desarrollar software. En un punto inicial se determinan las posibles rutas de desarrollo y se opta por aquella que tiene un riesgo más aceptable, y se continua repitiendo este patrón en un ciclo en espiral tal y como se ve en la figura 4.1.

La principal razón de aplicar esta metodología al proyecto se encuentra en que una gran parte importante del desarrollo se centra en la prueba de herramientas e implementación de técnicas. Debido a esto es necesario que las primeras tareas sean refinadas a medida que el proyecto evoluciona, escogiendo aquellas que impliquen un menor riesgo para el proyecto.

4.2. Temporización

Tal y como se explicó en la sección 1.2 el proyecto se divide en varias partes. Para su correcto desarrollo se ha tratado de llevar a cabo una distribución correcta del tiempo a emplear en cada tarea que compone el total del

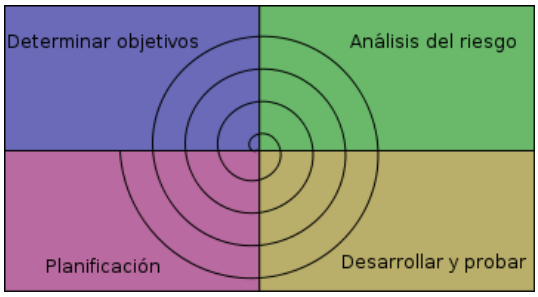

Figura 4.1: Representación del modelo en espiral. Fuente [Wikipedia]

proyecto.

Tal y como se puede comprobar en la figura 4.2, dentro del desarrollo del proyecto la mayor parte del mismo se ha dedicado a la que podríamos denominar *etapa de desarrollo*. En esta etapa estaría contenido tanto el procesado de las imágenes como el desarrollo y diseño de la interfaz.

Si se quiere ahondar más en la etapa de desarrollo podemos observar la figura 4.3 donde se puede apreciar que casi el 70 % está exclusivamente dedicado a la detección de las marcas así como diseño e implementación de la interfaz. Tareas como la segmentación y el análisis de la postura han tenido un menor impacto en la etapa de desarrollo.

4.3. Coste del proyecto

Dentro del desarrollo de cada proyecto hay que tener en cuenta el coste que este puede suponer tanto en materiales como software. Siguiendo la propuesta del proyecto se ha tratado usar la mayor cantidad de software libre posible (y en los casos que no fuese posible: licencias gratuitas con limitaciones), gracias a esta mentalidad se ha conseguido un coste 0 en en referencia al software.

Centrándonos en el hardware, ha sido posible que el material para la captura de movimientos fuese cedido por parte del departamento de Ciencias de la Computación e Inteligencia Artifcial de la Universidad de las Palmas de Gran Canaria. Pese a todo, el coste oficial del sensor Kinect v2 es de 150 euros siendo necesario un conector especial con coste de 50 euros para poder

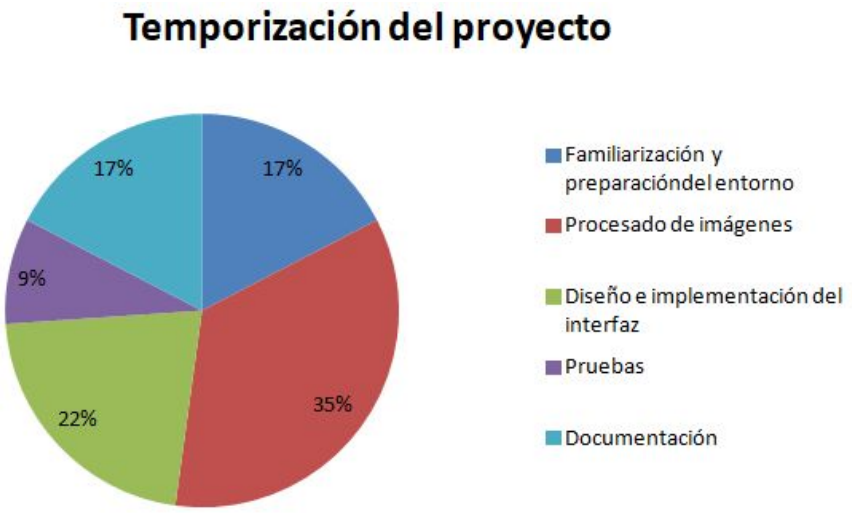

Figura 4.2: Representación de la temporización del proyecto.

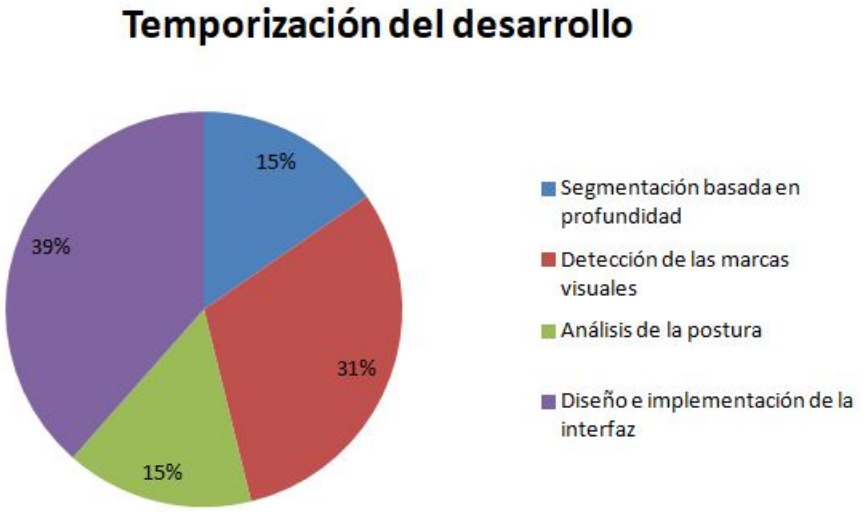

Figura 4.3: Representación de la temporización de la etapa de desarrollo.

Cuadro 4.1: Lista de actores

Actor	**Tipo**	**Definición**
Experto en biomecánica	Principal	Usuario experto que hace uso de la aplicación. Tiene acceso completo a todas las características de la misma.

usarlo en ordenadores. Aunque en fases tempranas del proyecto se usó el modelo original de Kinect a día de hoy está descatalogado y no se tienen precios oficiales del mismo.

Para que todo funcione correctamente es necesario un equipo personal con Windows 8.1 en adelante (ya sea de escritorio o portátil) que tenga al menos un puerto USB. 3.0, habiendo tanta variedad de modelos disponibles en el mercado se ha decidido usar un valor medio entre diferentes modelos que cumplan estos requisitos, aproximadamente 500 euros.

4.4. Modelo de casos de uso

Es importante tener claro los casos de uso de nuestro proyecto, siendo el primer paso identificar los actores que serán parte de ellos y enumerar uno a uno los casos asociados a cada actor.

Posteriormente se mostrarán diagramas de caso de uso individuales en los que se detallará de forma visual cada caso asociado a su actor. En el Anexo I se detallarán cada caso de uso en concreto.

4.4.1. Identificación de actores

Los actores son todos los seres que interactuan con el sistema, pero a su vez, es externo al mismo. En nuestro caso, tal y como se indica en la tabla 4.1, en conjunto de actores se limita únicamente al experto que hace uso de la aplicación desarrollada, podría tenerse la idea equivocada de que el deportista que es grabado puede ser un actor también, pero en ningún caso hará uso de la herramienta. Como se puede apreciar en el cuadro **??**.

A continuación se enumerarán las relaciones entre el actor (experto en

Cuadro 4.2: Relación actor-acciones

Actor	**Acción**	**Descripción**
Experto en biomecánica	Elegir ruta grabación	El usuario puede elegir la ruta en la que quiere guardar la sesión que va a grabar.
	Comenzar la grabación	Se empieza a grabar la sesión de imágenes a procesar
	Parar la grabación	Se para la grabación de imágenes a procesar
	Comprobar ruta de procesado	El experto comprueba que la ruta de imágenes para procesar es correcta
	Comenzar procesado	Una vez comprobada la ruta, empieza el procesado de imágenes
	Avanzar fotograma	El experto puede elegir avanzar el fotograma actual

biomecánica) y las acciones que tiene asociadas. Cuadro 4.2

4.4.2. Diagramas de casos de uso

Ya identificado el único actor de nuestro modelo podemos identificar los casos de uso (tabla 4.2 y así elaborar el diagrama correspondiente, en el que se presenta cada uno de los casos de uso a desarrollar. En la figura 4.4

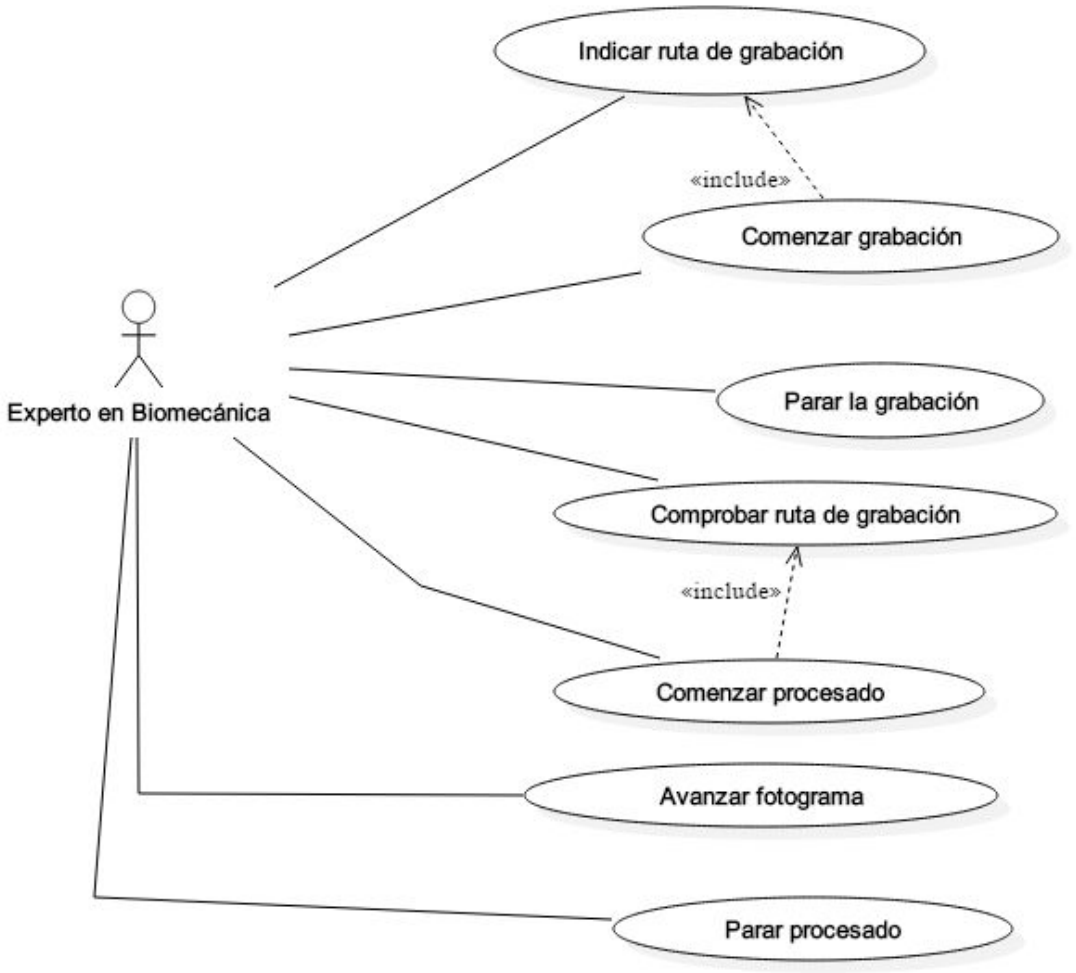

Figura 4.4: Diagrama de los casos de uso

Capítulo 5

Desarrollo e implementación

Esta sección del documento se centrará en lo que concierne al desarrollo e implementación del sistema. Para que el seguimiento del mismo sea más simple se ha decidido dividirlo en las diferentes tareas que conforman el desarrollo y que se comentarán de forma separada. Esta división se corresponde a la vista en la Figura 5.1.

Por otro lado se debería resaltar que el proceso de desarrollo e implementación ha sufrido, para casi todas sus tareas, dos puntos críticos en los que los requisitos de los mismos han podido cambiar. Por otro lado a la hora de implementar las interfaces hubo un tercer punto crítico que acabó afectando también al ajuste de marcas. Los hitos en concreto son:

1. Grabación y procesado imágenes Kinect v1
2. Grabación y procesado imágenes Kinect v2
3. Adaptar el proyecto como DLL[5]. *Exclusivo de las interfaces y ajuste de marcas.*

5.1. Recopilación de secuencias de test

La captura de imágenes es la base sobre la que se sustenta el resto de las tareas, es lógico que sin una disposición de las mismas no podemos hacer

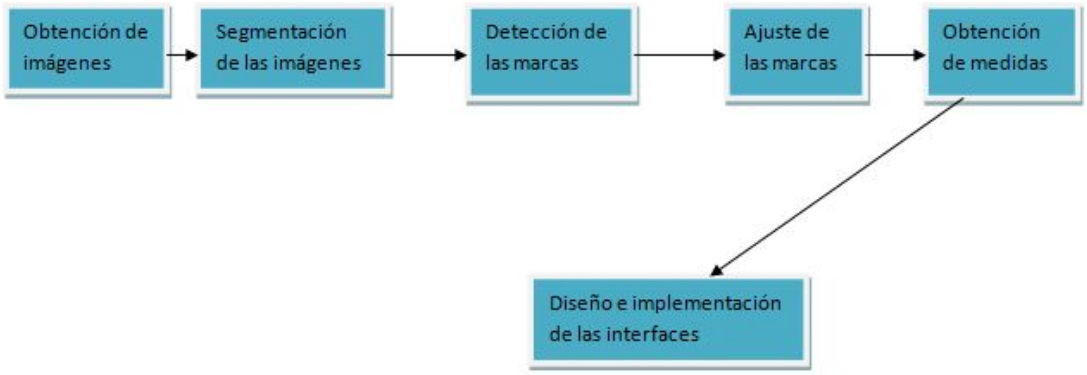

Figura 5.1: Represenstación de la división de tareas.

ningún tipo de procesado. Para ello se 'procedió instalar la cámara Kinect v1 e integrarla en el entorno de programación Visual Studio. Para esta primera iteración código se usó una combinación de la biblioteca OpenNI con OpenCV. Su esquema de funcionamiento era bastante sencillo:

```
if(Camara.isOpen())
Mostramos el fotograma actual
While(true)
        Grabamos en disco imagen RGB y profundidad
        if(tecla de escape)break
Fin del Programa
```

Con este sencillo procedimiento se obtenían las imágenes en RGB y profundad tal y como se muestran las figuras 5.2 y 5.3

Ante la llegada de la Kinect v2 hubo que cambiar el código de obtención de imágenes, OpenNI no ofrecía soporte para este modelo de captura. La solución se encontró instalando el SDK de *Kinect for Windows SDK 2.0*, siendo una pieza indispensable para el correcto funcionamiento debido a que, entre otras muchas cosas, el SDK instala los drivers obligatorios y necesarios del dispositivo.

Además el mismo SDK ofrece al usuario ejemplos de las distintas características de la cámara, usando estos ejemplos como base se hizo una simplificación del código de captura ofrecido por Microsoft para el proyecto. A

Figura 5.2: Imagen en RGB con la Kinect v1.

Figura 5.3: Imagen en profundidad con la Kinect v1.

Figura 5.4: Imagen en RGB con Kinect v2.

nivel de estructura se trató de mantener la misma filosofía que con la anterior rutina de captura

```
App() {
                GetDefaultKinectSensor(&kin_sensor_);
                if (FAILED(hr)) {
                        throw Exception;
                }

                if (kin_sensor_) {
                        Captura las 3 imagenes
                }
        if (tecla de escape) break;
Fin del Programa
```

El resultado de la captura se puede observar en las figuras 5.4, 5.5 y 5.6 para las imágenes de color, infrarrojo y profundidad respectivamente. Aunque ya se comentó de la diferencia teórica en cuánto a calidad entre ambas cámaras si se comparan las imágenes entre las capturas de cada una se confirma la ganancia que justifica el trabajo extra que implicó rehacer el código de captura.

Figura 5.5: Imagen en infrarrojos con Kinect v2.

Figura 5.6: Imagen en profundidad con la Kinect v2.

5.2. Segmentación de las imágenes

Durante el proceso de segmentación se trata de, usando los datos que aporta la imagen en profundidad, eliminar toda la información que no aporta ninguna información de interés para el procesado de las mismas. Para ello se decidió empezar a tratar solamente una pareja de imágenes RGB-D en lugar de todo el conjunto, una vez el procedimiento estuviese refinado sería replicado por el conjunto total de imágenes.

La idea inicial para la primera versión del proyecto se centró en obtener el rango de profundidad a la que se encontraban las marcas pegadas en el cuerpo del dentista. Para ello se desarrolló un procedimiento con el que el usuario debía pinchar en una ventana emergente a la altura de las marcas de colores. De la zona seleccionada se cogería un pequeña área de pixeles del que se extraería la profundidad, una vez repetido este procedimiento con todas las marcas, se obtendría la media de profundidad de cada una de las marcas.

Una vez obtenido este valor medio se utilizaría la función de OpenCV *threshold* para eliminar toda aquella información que estuviese fuera del rango de profundidad que nos interesase. Lo que esta función lleva a cabo es un umbralizado en función del valor que se le pase, así mismo permite varios modos de ejecución del umbralizado tal y como se ven en la figura 5.7, siendo el **binario** y **binario inverso** los que nos interesan.

Para no eliminar más información de la necesaria a la hora de realizar el umbralizado, en lugar de usar como valor de corte la media exacta se le ha dado un ligero margen de error y el resto de los Pixeles de la imagen se han puesto 0, esto significa que se han pintado de negro.

```
threshold(imagenFuente, Destino,
    MediaProfundidad-100, 0, THRESH_TOZERO);
threshold(imagenFuente, Destino,
    MediaProfundidad+100, 0, THRESH_TOZERO_INV);
```

Podemos ver un ejemplo del resultado en la figura 5.8 se puede observar que el resultado era bastante satisfactorio, sin embargo con el cambio de hardware el sistema de umbralizado no funciona exactamente igual, la razón de esta situación se verá en profundidad en la sección 5.3, la diferencia reside en que en lugar de seleccionar la zona dónde está la marca seleccionaremos **cualquier otro punto** del cuerpo que se desee umbralizar, debido a que

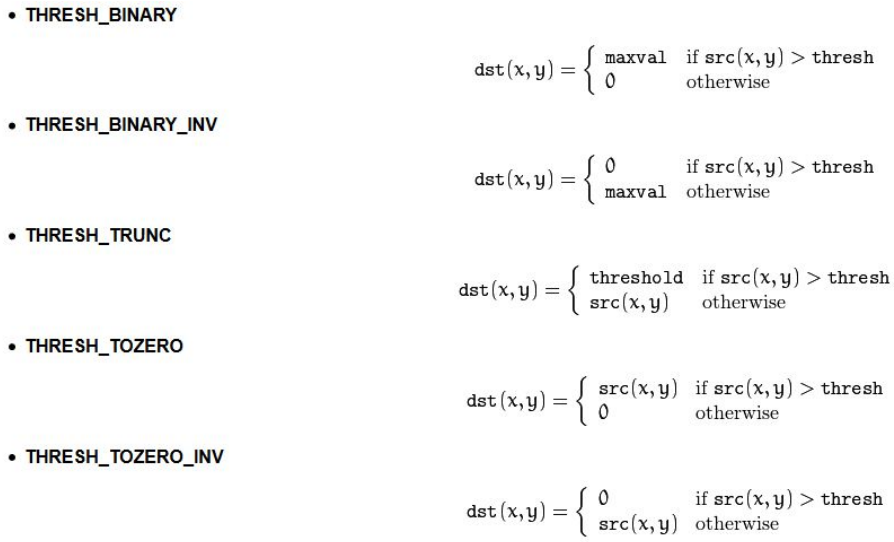

Figura 5.7: Tipos de umbralizado. Fuente API de OpenCV

ahora las marcas van a devolver un valor de distancia 0 tal y como se puede apreciar en la figura 5.9

5.3. Detección de las marcas visuales

El objetivo de este punto del procesado consiste en la detección de las marcas visuales, para ello se plantearon diversas posibilidades a la hora de seleccionar el tipo de marca que se emplearía. Inicialmente se valoraron tres posibilidades distintas: Códigos QR, una señal básica o marcas de colores. Cada una de las posibilidades representa algún tipo de ventaja. Se puede ver un ejemplo de cada tipo de marca en la figura 5.10

- **Los códigos QR.** A cada código QR se le podía asignar una posición concreta dentro del esqueleto humano, una vez detectados y transformado el código en la información que contiene evitaría la futura necesidad de encontrar a qué posición del cuerpo humano corresponde cada punto detectado.
- **Señal básica.** Una señal básica permitiría una fácil detección haciendo una búsqueda por texturas, al ser una imagen poco común la tasa de

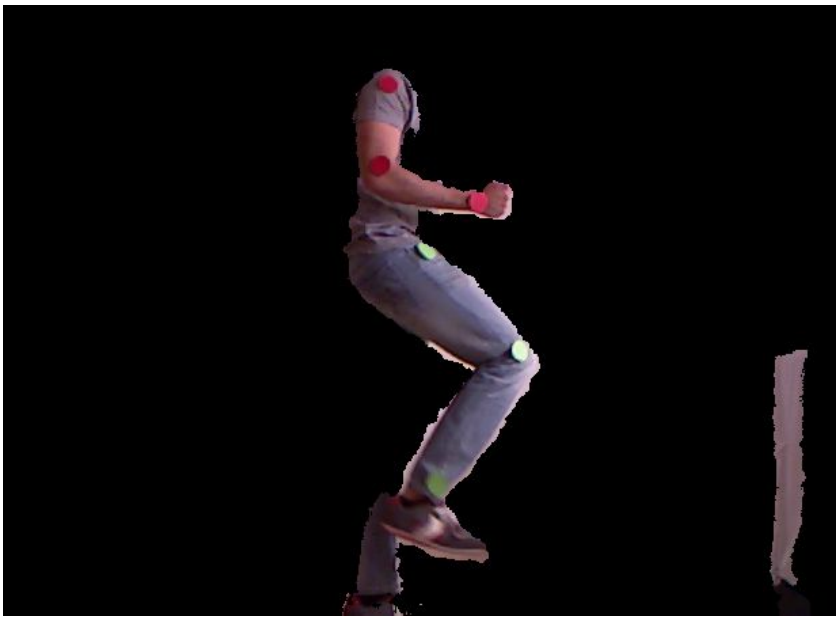

Figura 5.8: Ejemplo de aplicar el umbralizado.

aciertos sería, en teoría, elevada.

- **Marca de colores.** Sencilla de usar, detectar colores es uno de los puntos básicos en el procesado de imágenes.

La decisión al final de cuál elegir vino determinada por un factor externo: la resolución de la cámara. Aunque lo ideal hubiese sido elegir como elementos de diferenciación los códigos QR o las señales básicas se tuvo que optar por las marcas de colores, la calidad de la imagen obtenida no permitía hacer el seguimiento e identificación del código QR a la distancia que se encontraba la cámara del sujeto que estaba siendo grabado, y de forma similar ocurría con la señal a comparar con una textura simple.

Por tanto se empezó a trabajar sobre marcas de colores, para ello se modificó el procedimiento de obtención de profundidad, ahora a la vez que obtenía la profundidad que el usuario seleccionaba, se extraían los valores RGB de ese punto por separado y al igual que en el caso anterior, se calculaba la media de cada componente en el espectro de color.

Con estos valores se aplicaría la función *inRange* de OpenCV [6], que realizaría un umbralizado (tal y como se hizo para segmentar la profundidad),

Figura 5.9: Profundidad 2.0, se aprecia que la zona de las marcas da valor 0.

Figura 5.10: Ejemplo de las marcas valoradas.

pero en este caso en base a las medias de las componentes roja, verde y azul. Para ello, y al igual que se hizo con *threeshold* se daría un margen de error a la función.

```
inRange(ImagenFuent,
        Scalar(MediaR-15, MediaG-15, MediaB-15),
        Scalar(MediaR+15, MediaG+15, MediaB+15),
        ImagenDest);
```

En condiciones ideales esta solución hubiese sido perfecta, sin embargo los resultados no fueron satisfactorios, un elemento externo no se había tenido en cuenta y afectaba en gran medida a la calidad del resultado entre fotogramas: la luz ambiente. Esto implicaba que algunos fotogramas la detección fuese perfecta y en otros (que se distanciaban en décimas de segundo) los resultados fuesen inaceptables, con detecciones incorrectas.

La solución se encontró cambiando el espectro de colores de la imagen: se transformó la imagen original RGB al espectro HSV[7]. Este modelo creado en 1978 por Alvy Ray Smith consiste en una transformación no lineal del espacio de color RGB en las componentes de Matiz, Saturación y Valor o Brillo (Hue, Saturation y Value respectivamente). En este modelo el matiz se representa por una región circular, una región triangular separada, puede ser usada para representar la saturación y el valor del color. La representación de este modelo puede verse en la figura 5.11.

Con este nuevo modelo se volvió a modificar el procedimiento en donde se extraía la media de profundidad de las marcas y la media de las componentes RGB, para que ahora fuese la media de los componentes Matiz y Saturación la que se obtuviese ignorando los valores de brillo. A continuación se decidió volver a umbralizar usando la función *inRange* usando la media (más un ligero margen de error) tanto del Matiz como de la Saturación y dejando todo el espectro de Brillo disponible con la intención de paliar el efecto ejercido por la luz externa en las marcas de colores.

```
inRange(ImagenFuent,
        Scalar(MediaH-15, MediaS-15, 0),
        Scalar(MediaH+15, MediaS+15, 255),
        ImagenDest);
```

En este caso usando de base a la figura 5.12, se puede apreciar que los resultado de la figura 5.13 fueron satisfactorios. Para llegar a ese punto se

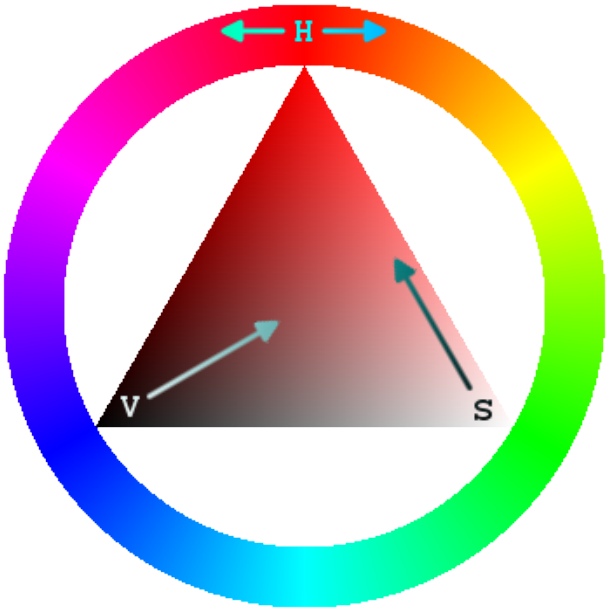

Figura 5.11: Triángulo representativo del modelo HSV. [Fuente Wikipedia]

usó la función *findContours* de OpenCV que permite la búsqueda de contornos en una imagen, una vez encontrado los contornos se realiza una tarea de eliminación de ruido mediante los procesos de dilatación y erosión[8] (transformaciones morfológica básicas).

```
findContours() // busca los contornos
erode (imagen)//
dilate(imagen)
```

La detección de marcas con la primera versión de la cámara kinect había finalizado en este punto, y a esta altura del proyecto fue cuando se obtuvo la versión 2.0 del receptor de imágenes. En la primera prueba de del nuevo modelo se apreció que este hardware simplificaría mucho la detección de imágenes: las marcas de colores serían sustituidas por trozos de tela reflectante. Con la detección de imágenes en infrarrojo los valores de dichas marcas en ese espectro de luz sería máximo (a nivel computacional 255) simplificando la búsqueda y aumentando la tasa de aciertos.

Debido a que el receptor de profundidad es el mismo que el de infrarrojos esto conllevaría un problema: aquellos valores que en el espectro infrarrojo fuesen máximos serían nulos en profundidad. La solución para recalcular la

Figura 5.12: Imagen base en espectro HSV.

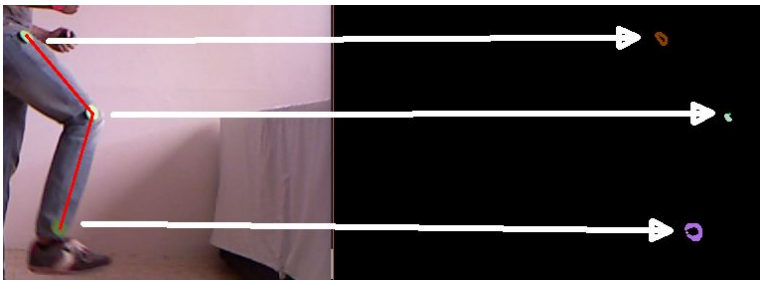

Figura 5.13: Ejemplo de detección de marcas tras el procesado HSV.

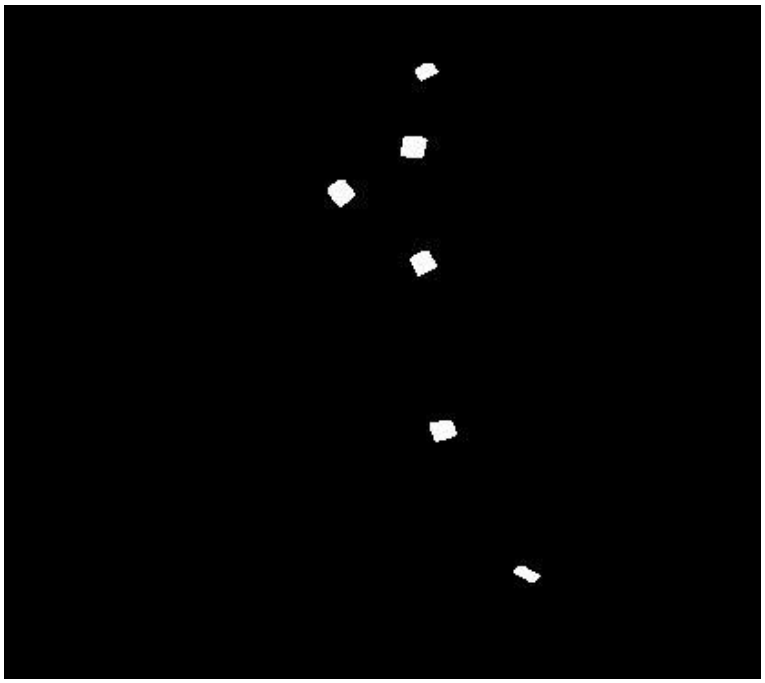

Figura 5.14: Resultado de la detección de marcas con Kinect v2.

profundidad de cada punto fue sencilla:

```
findContours() // busca los contornos
moments() //Obtener centro de contornos
dilate(imagen)
erode (imagen)//
Media del centro tras la transformacion
```

Utilizando la función *Moments* es posible obtener el momento central de cada contorno, si se transforma la imagen dilatándola y luego erosionándola se puede obtener una aproximación de la profundidad que se tenía el borde del contorno.

5.4. Ajuste de las marcas

Con las marcas ya obtenidas el siguiente paso era ajustarlas para obtener las medidas. Es importante resaltar que las imágenes de color y las de

profundidad no están alineadas en ambos modelos de cámara. Para la primera versión la biblioteca OpenNI había solventado el problema rellenando la matriz de la imagen de profundidad por el marco con valores 0 (negro) ajustándola con su pareja RGB. Sin embargo, llegados a este punto el proyecto estaba enfocado directamente a la versión 2.0 de la cámara.

La kinect 2.0 trae entre sus funciones una tabla con las equivalencias entre el desfase de imágenes, sin embargo el problema reside en que solamente se puede obtener dicho valor durante tiempo de grabación, y no en modo offline.

Este podría haber sido un problema que podría haber consumido más tiempo del deseado, sin embargo ya se había hecho la propuesta de animar un modelo en 3D con los datos obtenidos, así que para el ajuste de marcas usar la imagen infrarroja con los contornos se adecuaba a la perfección al problema.

Una vez obtenidos los contornos el siguiente punto era ordenarlos, para ello se decidió que era requisito indispensable que la primera imagen cumpliese unos requisitos: la posición de inicial del paciente grabado debía tener los puntos ordenados de la siguiente manera (en orden descendente):

- Hombro
- Codo
- Muñeca
- Cadera
- Rodilla
- Tobillo

La figura 5.14 muestra el filtrado de una imagen en infrarrojo en la posición inicial requerida. Una vez cumplido este requisito se pueden ordenar y unir los puntos de la imagen inicial , y adelantándonos a la sección 5.5, obtener los ángulos de la imagen inicial. Tal y como muestra la figura 5.15

El siguiente paso era determinar cuál era el mejor método de predicción para mantener los puntos ordenados entre fotogramas. Surgieron dos opciones:

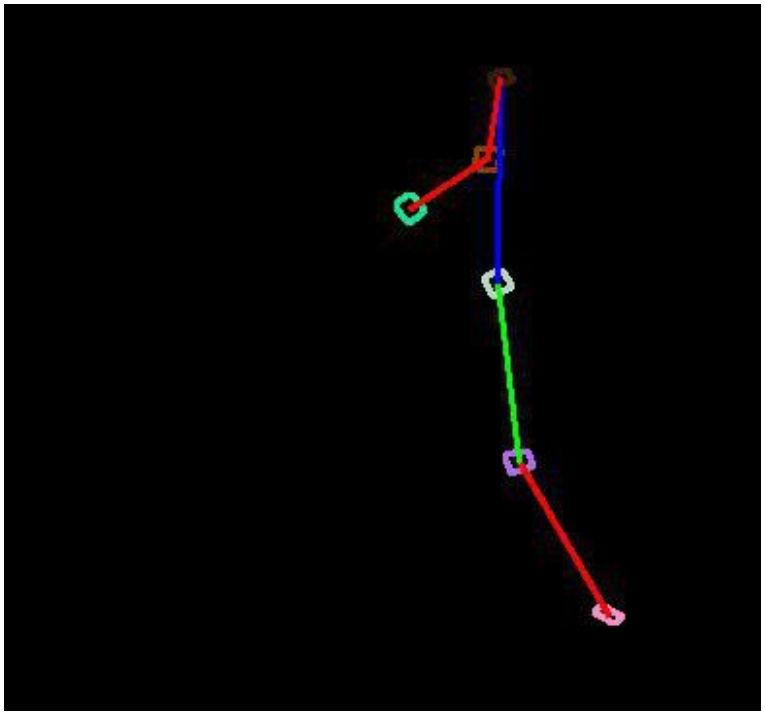

Figura 5.15: Conexión correcta de las articulaciones.

- Buscar la distancia mínima entre los puntos de un fotograma y otro. Para ello se hacía un cálculo entre los puntos de un fotograma y el siguiente, y se asignaban en función de la diferencia mínima de movimiento en el espacio de la imagen. Los resultados tuvieron una tasa de acierto bastante baja demostrando que el método no era deseable.
- Búsqueda de puntos por variación entre ángulos. En lugar de buscar la diferencia de los puntos en el espacio se filtró buscando la diferencia mínima total que formaban los ángulos del fotograma anterior y la combinatoria de todos los puntos del siguiente fotograma. Este método dio resultados satisfactorios.

Para ello se realizó un procedimiento que calculase la combinatoria posible de todos los puntos obtenidos y los almacenase en una matriz. Una vez calculadas todas las posibilidades se buscaría aquella combinación que al sumar la diferencia de cada componente diese un valor mínimo:

$$A = \begin{bmatrix} \alpha_1, & \dots & , \alpha_4 \end{bmatrix}^T \tag{5.1}$$

$$B = \begin{bmatrix} \beta_{1,1} & \dots & \beta_{1,4} \\ \vdots & \ddots & \\ \beta_{n,1} & & \beta_{n,4} \end{bmatrix} \tag{5.2}$$

$$B^i = \begin{bmatrix} \beta_{(i,1)} & \beta_{(i,4)} \end{bmatrix}^T \tag{5.3}$$

$$argmin_i = \left\| A - B^{-i} \right\|_1 \tag{5.4}$$

Tal y como se aprecia en la figura 5.16 los resultados fueron más que correctos.

5.5. Obtención de medidas

Dentro de este apartado se habla de la adquisición de las medidas necesarias para el diagnóstico del experto. En realidad, este punto del desarrollo

Figura 5.16: Comparación entre dos fotogramas consecutivos.

ha estado presente desde el comienzo del mismo, por ejemplo tal y como se explicó en el apartado 5.3 con la obtención de la profundidad (que al final se descartó su uso, pero se obtuvo igualmente).

Este punto hace hincapié en la obtención de los ángulos y ya que es el valor que se usará dentro del estudio biomecánico resulta de gran importancia que este cálculo sea lo más preciso posible, para ello se decidió que la mejor manera de obtener dicho valor era usar la arcotangente[9], en concreto la implementación *(atan2)*.

$$atan\theta = \frac{sen\theta}{cos\theta}$$

Sabiendo que tenemos tres puntos: C, e_1,e_2 los vectores A y B estarían compuestos de tal manera:

$$Ax = e_1x - Cx; Ay = e_1y - Cy$$

$$Bx = e_2x - Cx; By = e_1y - Cy$$

A partir de estos de vectores, se calcula el seno y el coseno

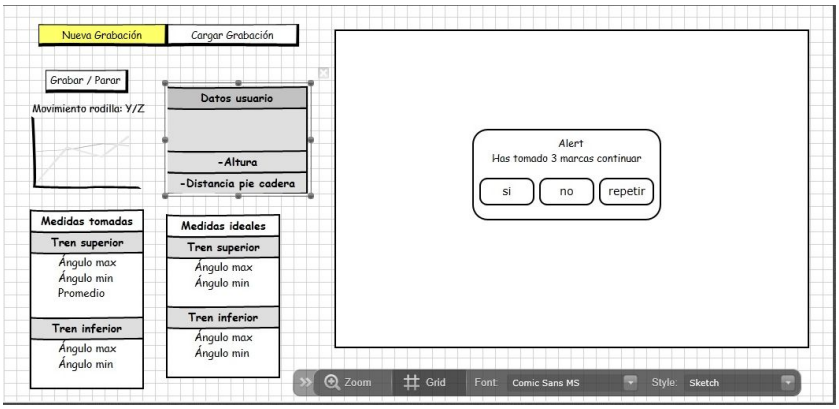

Figura 5.17: Borrador de la interfaz.

5.6. Diseño e implementación de las interfaces

El punto final del desarrollo de la aplicación consiste en el diseño de las interfaces, esta fase sufrió varias transformaciones durante el desarrollo de la aplicación. Tal y como se ve en en el Mock Up (figura 5.17) el diseño inicial consistía en una única interfaz unificada que permitiese la grabación, visualización y procesado de imágenes de forma simultánea.

Pese a que no era un requisito expuesto de forma inicial dentro del ámbito del proyecto se quiso abarcar como una posibilidad que daría un mejor acabado al proyecto en si mismo. Sin embargo las dificultades surgieron a la hora de cambiar el modelo de cámara para la captura de movimientos.

Como ya se ha explicado, la estación de trabajo portátil no cumplía el requisito básico del puerto USB 3.0 que exige el nuevo modelo de Kinect y como alternativa se decidió segmentar el programa en dos, cada uno con su respectiva interfaz: Una para captura (que se llevaría a cabo en un ordenador compatible con la cámara) y otra para el procesamiento de las imágenes.

Como se comentó en la sección de tecnologías usadas, para la animación del muñeco se decidió usar Unity, cuyo lenguaje de programación es esencialmente C#. Ante esto surgió el siguiente problema: hasta ahora todo el proyecto había sido desarrollado en otro lenguaje distinto al que usaba la herramienta elegida. Esta situación hizo que se plantearan dos opciones

básicas: buscar otra herramienta o intentar adaptar el código para que fuese compatible con Unity.

Tal y como se recoge en la páginas oficial de Microsoft [10] es posible portar código en lenguaje C++ hacia C# en forma de DLL. Para ello y siguiendo la guía que la propia Microsoft ofrece, el primer paso dentro de este proceso es configurar en el IDE el proyecto para que compilase como una DLL en lugar de que lo hiciese en forma de ejecutable.

La forma por la cuál se consigue invocar código de C++ a C# se hace mediante uso de la directiva específica de Microsoft **dllexport**. Dllexport permite indicar al enlazador de la librería en cuestión que ese elemento (función, clase, etc.) debe ser visible fuera de la DLL. Sin este atributo el método únicamente sería visible dentro de la librería.

5.6.1. Interfaz procesado

En este punto se detalla el desarrollo de la interfaz que corresponde al código de procesado de imágenes. A continuación se explicará brevemente como funciona el entorno de programación de Unity.

Escenas

Al crear un proyecto se genera de forma automática una escena, cada escena contiene los diferentes elementos que conforman la aplicación, a nivel de abstracción cada escena correspondería a un nivel único. Entre los diferentes elementos que pueden componer la escena resaltaremos tres: Assets, interfaz de usuario, los scripts.

Assets

Tal y como define la propia guía de Unity [11] un asset es una representación de cualquier elemento que puede ser utilizado en un proyecto de Unity. Un asset puede ser un archivo creado en un entorno externo a Unity (aunque no es un requisito necesario), y dentro de este grupo se engloban ejemplos

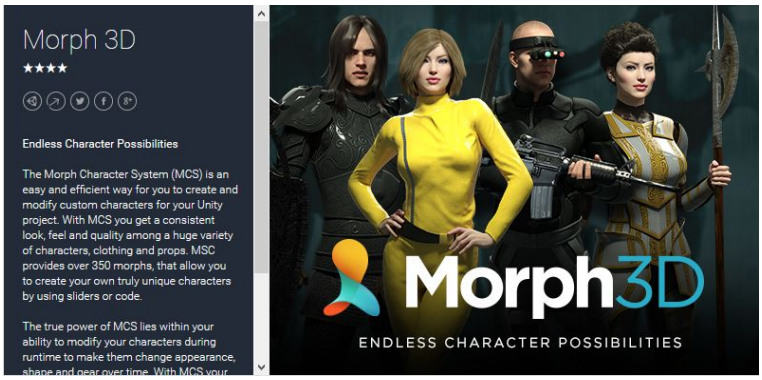

Figura 5.18: Asset Morph 3D, imagen oficial de la Asset Store.

tales como un modelos 3D, archivos de audio, imágenes, o cualquiera de los otros tipos de archivos que Unity soporta.

Como ya se mencionó en el apartado de herramientas, Unity ofrece una tienda online de Assets[12], con una gran variedad de assets que pueden ser tanto gratuitos como de pago. Esto abrió la posibilidad de encontrar modelo en 3D que se deseaba animar sin necesidad de tener conocimientos de modelado.

Los tres requisitos por lo cuales que se filtraron la búsqueda fueron:

1. Que fuese gratuito.

2. Que tuviese aspecto humano

3. Que tuviese como característica un cuerpo rigged.

Que un cuerpo tenga como característica *rigged* implica que a la hora de diseñar el modelo se ha decidido introducirle un esqueleto articulado que permite la animación de las articulaciones.

Tras una búsqueda en la Unity Asset Store la elección fue Morph 3D tal y como se puede apreciar en la figura 5.18

Interfaz de usuario y scripts

La herramienta de creación de interfaz de usuario (GUI por sus siglas en inglés) de Unity permite crear de forma sencilla e intuitiva interfaces agregando componentes de diversa índole: ya sean los nombrados Assets o elementos internos de Unity como puede ser un simple botón.

Estos elementos pueden reaccionar a scripts generando eventos (cambios de escena), modificando algún componente (realizando un movimiento en algún modelo) o responder ante la entrada de un usuario.

dllexport y Código unsafe

Una vez adquiridos los conocimientos básicos clave el primer paso a realizar fue un script simple que sencillamente llamase a la DLL. Para ello se configuró siguiendo estas dos pautas de forma estricta:

1. La DLL tiene que estar compilada en 64 bits.
2. Obligatoriamente tiene que estar en la carpeta principal del proyecto (no permite invocaciones externas).

El siguiente punto a tratar dentro de la llamada a la librería se centra en la devolución de varios valores en cada llamada, para ello se decidió usar una estructura que contuviese todos los datos buscados. Al tratarse de punteros obligó a crear un bloque de código *unsafe*, ya que es requisito indispensable en C# para este tipo de variables [13].

En este punto de la ejecución ya se tenían los datos de movimiento: puntos y grados. Ante esta situación se plantearon dos opciones:

1. Calcular la equivalencia entre puntos del espacio en la imagen y en el modelo 3D.
2. Usar los grados para asignar los movimientos en grados.

Por sencillez se optó por la segunda opción, para ello se usó la estructura de Unity *Quaternion*. Los cuaterniones son usados de forma interna por

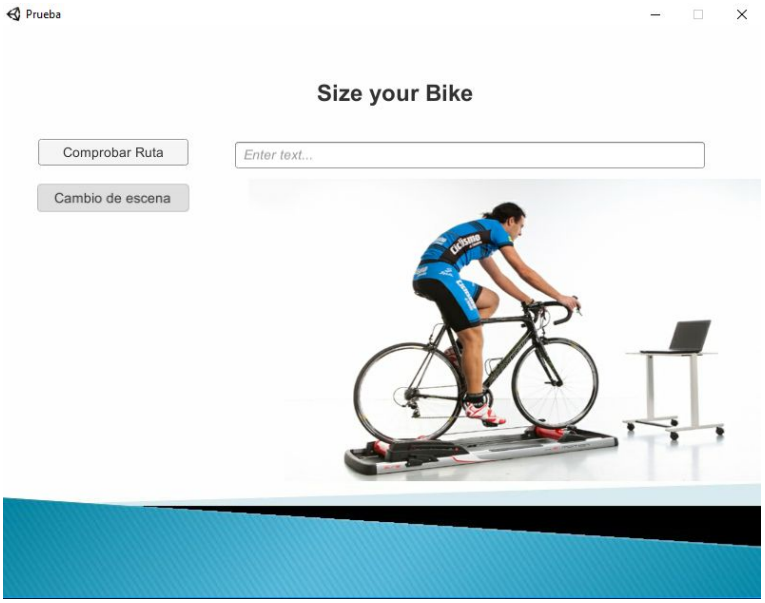

Figura 5.19: Primera escena: menú inicial.

Unity para representar todas las rotaciones. De todas las opciones posibles se escogió aplicar una rotación de Euler, que devuelve una rotación que gira z grados alrededor del eje z, x grados alrededor del eje x, y y grados alrededor del eje y (en ese orden). Aplicando dicha transformación a los puntos seleccionados (hombro, codo, cadera y rodilla) se consiguió transportar los movimientos de la grabación al modelo 3D.

Una vez alcanzada la meta de movimiento se pasó a diseñar la interfaz, que consistiría en dos escenas distintas: la primera una pantalla inicial donde indicar la ruta de las imágenes a tratar (Figura 5.19) y la segunda donde se ejecuta el procesado de las imágenes y se muestra el resultado (5.20).

Para obtener la ruta de archivos se optó por usar el buscador de archivos que proporciona Unity en el espacio de nombres *UnityEditor*, sin embargo, Unity no permite cargar dicho espacio en tiempo de ejecución por tanto hubo que buscar una solución. En este caso se valoraron tres opciones:

1. Crear un buscador de archivos desde cero. Se descartó por la excesiva complejidad.

2. Buscar en Unity Asset Store. Dentro de la tienda de la aplicación se

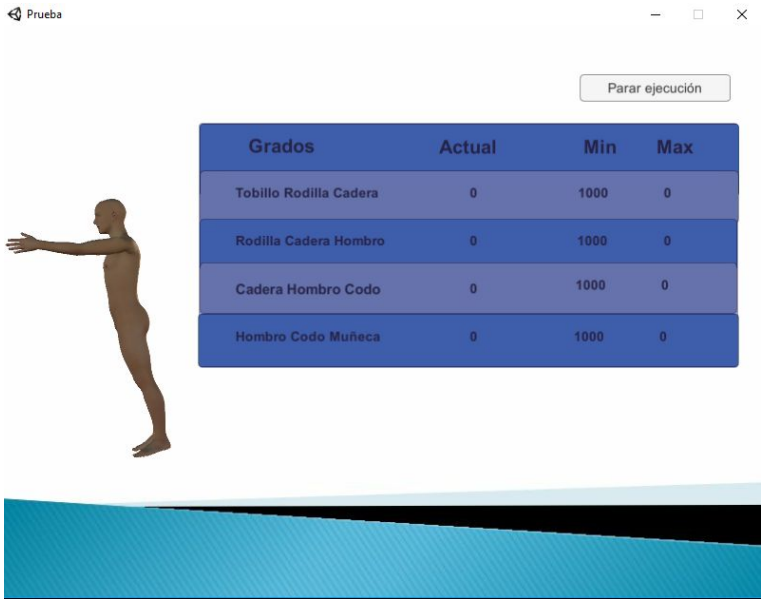

Figura 5.20: Segunda escena: procesado y animación.

encontraron varios exploradores de archivos, sin embargo todos eran de pago con precios que oscilaban entre los 10 y 40 euros. Al no tener fin comercial este proyecto se descartó esta opción.

3. Escribir la ruta en una pestaña de entrada. Se habilitó un campo de escritura en la que el usuario podía escribir una ruta dentro de su equipo. Para que el sistema fuese más robusto se hizo que comprobase que la ruta existía y que al menos contuviese las tres carpetas obligatorias con cada tipo de imagen. Mientras no se cumplan estas condiciones el botón para cambiar de escena permanecerá inactivo.

Para la escena de procesado se colocó el modelo en 3D y a su lado una tabla que mostraba los ángulos para los siguientes casos:

- Fotograma actual
- Valor mínimo entre todos los fotogramas
- Valor máximo entre todos los fotogramas

Para cada fotograma se irían comparando los valores con los obtenidos anteriormente, en caso de que alguno superase el máximo o el mínimo el valor

se actualizaría. Una vez procesados todos los fotogramas el programa vuelve a la escena inicial. Además en todo momento se va mostrando una miniatura de las imágenes en RGB y la segmentación de las marcas del fotograma que se está analizando, para completar la información visual ya dada.

Por último se decidió añadir un botón que permitiese salir de la escena de procesado y volviese a la de inicio en caso de que el experto lo requiriese.

El funcionamiento de este interfaz se ve con mayor detalle en el Apéndice C.

5.6.2. Interfaz de grabación

Para la interfaz de grabación se decidió utilizar un diseño sencillo y minimalista, consistiendo en 3 botones que darán toda la funcionalidad que necesita el usuario tal y como se muestra en la Figura 5.21. Cabe destacar que desde la versión 2012 de Visual Studio Microsoft ha dejado parcialmente abandonado el diseño de interfaces en el lenguaje C++ y centrándose y potenciándolo en C#. Aprovechando el conocimiento adquirido para la creación del proyecto en Unity se decidió que también se exportaría el código de grabación como una DLL a un proyecto de C#

A cada botón se le ha asignado una acción distinta

- **Ruta.** Abre un explorador de archivos y permite seleccionar la carpeta donde se almacenan las imágenes.
- **Grabar.** Invoca a la DLL para la grabación, si no se ha seleccionado ninguna ruta no permite su activación. Si el flujo de ejecución es correcto pasará por parámetro la dirección en la que se quieren almacenar las imágenes y el programa de captura crea tres carpetas (Infra, color, prof) en donde se almacenan las imágenes. Además cambia el mensaje de estado indicando que se está grabando.
- **Parar.** En caso de estar activo el proceso de grabación lo acaba y cambia el mensaje de estado indicando que ha finalizado el proceso.

El funcionamiento de la interfaz se verá con mayor detalle en el Apéndice B.

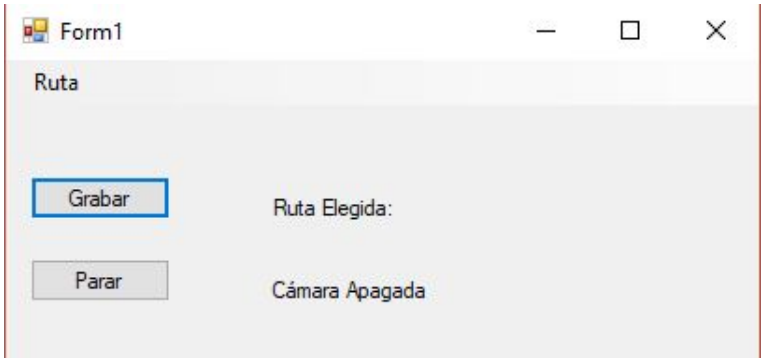

Figura 5.21: Aspecto del interfaz de grabación.

Capítulo 6

Pruebas

En esta fase se trata de someter a evaluación la aplicación, verificando su correcto funcionamiento y estabilidad, buscando posibles fallos de implementenación y comprobando su usabilidad a la vez de si cumple los requisitos definidos en la frase previa del análisis.

Debido a la naturaleza iterativa de este desarrollo, la fase de pruebas no se ha fijado para el final del mismo, durante el transcurso de cada una de las etapas de implementación se aseguraba la correcta funcionalidad del proyecto en su conjunto, hasta llegar al punto final dónde se comprobaba que el conjunto completo funcionaba en armonía.

Aún así, ciertos elementos del desarrollo tuvieron su espacio personal dedicado a las pruebas dónde se sometían a una serie de tests que comprobaban su fiabilidad y correcto funcionamiento.

6.1. Carga de imágenes

Durante gran parte del proceso se usó únicamente una pareja de imágenes en RGB y profundidad para ir aplicando las técnicas desarrolladas, siendo un punto de inflexión la necesidad de la carga de todo el conjunto de imágenes para comprobar que, de forma secuencial, el código desarrollado funcionaba correctamente. Aún pareciendo un asunto trivial esta función de ayuda tuvo

tres aproximaciones distintas, cada una mejorando con respecto a la anterior:

- **Lectura de rutas desde un fichero.** La primera carga de imágenes funcionaba leyendo la ruta en la que estaban contenidas las imágenes desde un fichero de texto. Cada vez que se encontraba un salto de línea se asumía que la ruta era esa y se cargaba la imagen mediante funciones de OpenCV. Como primera aproximación no era mala idea pero pronto se buscó una solución alternativa, escribir en ficheros de texto la ruta de cada imagen era engorroso y realizar un script que llevase a cabo esta función consumiría más tiempo del deseado, además este sistema era especialmente sensible a problemas con caracteres ocultos.
- **Uso de la librería dirent.h** Esta librería diseñada para sistemas UNIX permite la búsqueda y carga de archivos. Este encabezado no existe en IDE Visual Studio, fue necesario introducirlo a mano. Con esta librería se realizó un pequeño programa con el que se comprobaba que se obtenían las direcciones de los elementos almacenados en una carpeta de forma correcta.
- **Uso de funciones nativas de C#.** Con el cambio de C++ a C# se consiguió una mayor funcionalidad, ahora no se dependía de encabezados externos para la búsqueda y carga de las rutas, era el propio lenguaje el que proporcionaba dicha funcionalidad.

6.2. Aplicado de filtros y búsqueda de marcas

Una vez resuelto el problema de la carga de imágenes se desarrolló una pequeña prueba que aplicaba los valores de umbralizados obtenidos de una la imagen de muestra al subconjunto completo de imágenes.

```
Obtener_valor_defiltrado()
Mientas haya imagenes
        aplicar valores;
fin
```

Una vez comprobada que la segmentación se aplicaba de forma correcta se realizó sobre este mismo conjunto de prueba un testeo de búsqueda de marcas.

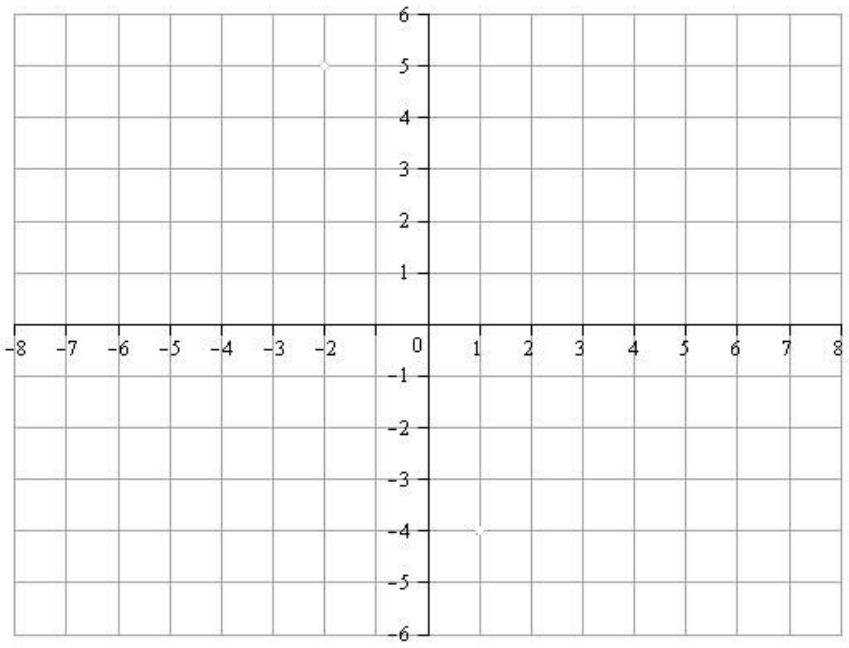

Figura 6.1: Eje de coordenadas usado para las pruebas.

6.3. Prueba de ángulos

Para verificar el cálculo correcto de los ángulos se desarrolló una función de testeo que cargaba la imagen de un eje de coordenadas (Figura 6.1), y sobre dicho eje de forma iterativa y aleatoria seleccionaba tres puntos en el espacio y obtenía el valor que conformaban dichos puntos.

En este punto se comprobó el correcto funcionamiento para casos especiales: ángulos de 0, 90, 180 grados. Los resultados fueron satisfactorios.

6.4. Obtención de combinatoria

Tal y como se explicaba en el punto 5.4 la forma de encontrar la correspondencia usada entre los puntos de los distintos fotogramas consistía en calcular la menor diferencia entre la sumatoria entre ángulos. Para ello era necesario asegurarse que el procedimiento de combinatoria funcionase de for-

ma correcta, para ello se realizó un pequeño test que ante una entrada con distintos valores realizaba el proceso combinatorio y lo volcaba en un fichero de texto.

6.5. Prueba final

Para la prueba final se realizó una prueba de caja blanca, que se centra en los detalles procedimentales del software, para ello se escogen distintos valores de entrada para examinar cada uno de los posibles flujos de ejecución del programa y cerciorarse de que se devuelven los valores de salida adecuados.

Por tanto, una vez verificada la correcta ejecución de todas las partes que conformaban el desarrollo del proyecto se decidió aplicar la totalidad del mismo a varios subconjuntos de imágenes. El esquema seguido sería el siguiente

1. Obtención y carga de imágenes.
2. Segmentación en profundidad.
3. Búsqueda e identificación de marcas.
4. Cálculo de ángulos y ordenación de marcas.
5. Aplicación de los datos obtenidos al modelo 3D.

Se verificó que la correcta comprobación de los distintos módulos de la aplicación implicó un resultado satisfactorio en el conjunto global del mismo.

Capítulo 7

Trabajo futuro y conclusiones

Como se ha visto el objetivo de este proyecto era montar un sistema de detección de marcas basándose en imágenes RGB-D para obtener las medidas de ajuste personalizadas en bicicletas. Tal y como se ha podido comprobar durante el desarrollo de este documento se han encontrando diversas dificultades que, dentro del rango de posibilidades, se han tratado de solventar.

Resaltar que se han cumplido los objetivos básicos del proyecto:

- Captura de imágenes
- Segmentación de las imágenes
- Detección de marcas visuales
- Ajuste del esqueleto
- Obtención de medidas

A su vez se han cumplido otros objetivos no propuestos inicialmente pero que, durante el desarrollo del proyecto, han surgido y se tomaron en consideración:

- Uso de nuevas tecnologías: Kinect 2.0
- Interfaz en Unity con animación de modelo 3D

7.1. Posibles mejoras

Una vez finalizado el proyecto se ha analizado el trabajo realizado y aunque en general el resultado ha sido satisfactorio (hay que recordar que nos encontramos ante un proyecto de fin de carrera con unos objetivos delimitados y no ante una aplicación comercial) no hay duda que hay margen de mejora, esto abre varias líneas de trabajo futuro dentro del mismo proyecto.

7.1.1. Unificación de las interfaces

Debido a los problemas de compatibilidad entre los equipos y la tecnología se ha tenido que dividir el programa en dos. El primero realiza la captura de imágenes y el segundo su procesado. Esta situación no es deseable y una de las principales líneas de trabajo debería centrarse en encontrar un equipo que cumpliese los requisitos de la cámara y a su vez no sufriese las incompatibilidades de Visual Studio y OpenCV. Esto mejoraría la experiencia de usuario, facilitando el uso de la aplicación y haciendo que sea más cómodo.

Además dentro de esta unificación se encontrarían englobados dos puntos que se habían discutido como trabajo futuro pero se descartaron con el paso del tiempo:

- **Ejecución en tiempo real.** Tener las dos aplicaciones en el mismo ordenador permitiría la ejecución en tiempo real de la aplicación. Esto implica que, a la vez que se obtienen las imágenes se vayan viendo los resultados en pantalla.
- **Integración imágenes en el interfaz de Unity.** Aunque se ha conseguido mostrar las imágenes en tiempo de ejecución sin molestar en la interfaz principal, lo ideal sería que estuviesen integradas en el mismo interfaz y no como una ventana emergente.

7.1.2. Mejora del diseño de la interfaz

Ambas interfaces, aunque son solventes e intuitivos, carecen de un diseño profesional deseado. Aunque no estaba como objetivo que el aspecto de las

aplicaciones fuese comercial, sería deseable que se asemejase al aspecto de una lo más posible (aunque no sea el objetivo poner a la venta la aplicación).

Para cumplir este objetivo lo ideal sería contar con la ayuda de un diseñador profesional que indicase las pautas mínimas y necesarias que hicieren que la experiencia de usuario fuese lo más agradable posible a la vista.

7.1.3. Obtención de más valores

Sería deseable poder obtener más valores para mejorar la precisión del diagnóstico biométrico, en este caso el problema reside más en la falta de conocimientos sobre qué datos se requieren más que la implementación de dichas funciones. Para ello lo ideal sería contar con la asistencia de un experto en la materia que asesorase sobre qué datos son necesarios, cuales serían deseables y, en caso de que la situación se diese, cuales de los disponibles no son necesarios y se puede prescindir de ellos.

7.1.4. Emisión de informes

Tal y como hacen ya otras aplicaciones sería ideal almacenar información y emitir informes para que el paciente y el experto los tengas disponibles en cualquier momento y no solamente en el de ejecución de la aplicación.

De forma idealizada se tendría una base de datos de cada cliente con cada sesión pudiendo observar las sesiones pasadas para tener referencias de los diferentes estudios y los valores ya obtenidos previamente.

7.2. Conclusiones

En este proyecto se ha tratado de diseñar y desarrollar una herramienta que mediante la captura de imágenes RGB-D permitiese una medición precisa de los ángulos que forman las articulaciones de un ciclista para un correcto posicionamiento del sillín. Llegados a este punto se puede decir que se ha cumplido el objetivo creando una aplicación funcional con un grado de

precisión correcto.

Aún así es importante destacar que la biomecánica es un punto emergente y como se ha visto en el apartado 7.1 este proyecto es susceptible a mejoras que amplíen su funcionalidad y calidad, tanto del servicio que presta como de la experiencia de usuario. Algunas de estas mejoras van más allá de lo propuesto inicialmente pero su inclusión haría que la calidad del producto final fuese bastante superior.

Tras el completo desarrollo y finalización del proyecto se puede afirmar que el tratamiento de imágenes no es una tarea trivial, muchos factores pueden afectar al proceso. En especial señalar que los lugares de captura, así como el hardware que se use para capturar, influyen bastante en la calidad de las imágenes, teniendo efectos perniciosos en forma de ruido que complican aún más el procesado de las mismas.

Para acabar, destacar que como proyecto académico se ha cumplido con las metas fijadas en un Proyecto de Fin de Carrera, haciendo uso de los conocimientos adquiridos durante los años de estudio de la misma. Destacar que uno de los aspectos más interesantes del proyecto en si mismo ha sido el uso de nuevas tecnologías por parte del estudiante: desde bibliotecas como OpenCV, a herramientas como Unity e incluso lenguajes que no habían sido usados hasta el momento, siendo el caso de C#. Esto añade un valor académico extra al proyecto en si mismo y ha forzado el tener que adaptarse ante las distintos problemas que surgían en cada fase del desarrollo. El realizar un proyecto de esta índole es un acercamiento al trabajo que se espera encontrar en el mercado, mostrando la importancia de cumplir los objetivos, tener constancia y saber adaptarse antes las diferentes limitaciones que van surgiendo durante el desarrollo del mismo.

Glosario

DLL Siglas en inglés de biblioteca de vínculos dinámicos, término con el que se refiere a los archivos con código ejecutable que se cargan bajo demanda de un programa por parte del sistema operativo. Esta función es exclusiva de sistemas operativos Windows . 41, 60, 62, 65

drivers Siglas en inglés para kit de desarrollo de software, hace referencia un conjunto de herramientas de desarrollo de software que le permite al programador o desarrollador de software crear aplicaciones informáticas . 42

FoV Siglas en inglés del término Campo de Visión (Field of Vision), hace referencia a la extensión de mundo observable en un momento dado. 24

IDE Integrated Development Environment, siglas en entorno de desarrollo integrado, consiste en una aplicación informáticaproporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software. 30, 60, 68

Mock Up En la manufactura y diseño, un Mock-Up es un modelo a escala o tamaño real de un diseño o un dispositivo, utilizado para la demostración, evaluación del diseño, promoción, y para otros fines. Un Mock-Up es un prototipo si proporciona al menos una parte de la funcionalidad de un sistema y permite pruebas del diseño. 59

offline Anglicismo del término fuera de línea, estado en el que un elemento se encuentra desconectado de un sistema o red. 55

pixeles Un píxel o pixel, (acrónimo del inglés picture element, 'elemento de imagen'), es la menor unidad homogénea en color que forma parte de una imagen digital. 47

RGB-D Siglas en inglés de Rojo, Verde, Azul y Profundidad. Referido a la obtención de un conjunto de datos de una imagen en color RGB y la obtención de un valor de profundidad para cada punto de dicha imagen. 13, 23, 47, 71, 73

script El script o guión es un programa usualmente simple, que por lo regular se almacena en un archivo de texto plano. Su uso habitual consiste en realizar diversas tareas como combinar componentes, interactuar con el sistema operativo o con el usuario. . 60, 62, 68

SDK Término para ontrolador de dispositivo, consiste en un programa informático que permite al sistema operativo interaccionar con un periférico, haciendo una abstracción del hardware y proporcionando una interfaz para utilizar el dispositivo . 42

Bibliografía

[1] J.R. Gómez-Puerto, Marzo Edir da Silva Grigoletto, Bernardo H. Viana Montaner, D. Vaamonde, and José Ramón Alvero Cruz. La importancia de los ajustes de la bicicleta en la prevención de las lesiones en el ciclismo: aplicaciones prácticas. *Revista andaluza de medicina del deporte*, 1(2):73–81, 2008.

[2] Jeff Martin. *Visual Studio 2015 Cookbook, 2nd Edition*. Packt Publishing, 2nd edition, 2016.

[3] Guía de uso básico de latex del sitio web sharelatex. https://es.sharelatex.com/learn/Main_Page, Julio 2017.

[4] M. H. Brown. Exploring algorithms using balsa-ii. *Computer*, 21(5):14–36, May 1988.

[5] Qué es un archivo dll. definición y uso en el sitio oficial de microsoft. https://support.microsoft.com/es-es/help/815065/what-is-a-dll, Julio 2017.

[6] Robert Laganiere. *OpenCV Computer Vision Application Programming Cookbook*. Packt Publishing, 2nd edition, 2014.

[7] George H. Joblove and Donald Greenberg. Color spaces for computer graphics. *SIGGRAPH Comput. Graph.*, 12(3):20–25, August 1978.

[8] Uso de las funciones de erosión y dilatación en opencv. http://docs.opencv.org/2.4/doc/tutorials/ imgproc/erosion_dilatation/erosion_dilatation.html, Julio 2017.

[9] Cálculo de ángulos. http://www.mathopenref.com/arctan.html, Julio 2017.

[10] Crear y utilizar una biblioteca de vínculos dinámicos (c++). https://msdn.microsoft.com/es-es/library/ms235636.aspx, Julio 2017.

[11] Manual de unity, assets. https://docs.unity3d.com/es/current/Manual/AssetWorkflow.html, Julio 2017.

[12] Unity asset store. https://www.assetstore.unity3d.com/en/, Julio 2017.

[13] Referencia c-sharp, codigo unsafe. https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/keywords/unsafe, Julio 2017.

Apéndice A

Casos de uso

En este apéndice se mostrarán los distintos casos de uso de forma detallada. Para ello se seguirá el siguiente esquema:

- **Nombre.** Nombre del caso de uso.
- **Identificador.** Número identificativo del caso de uso
- **Actor principal.** Se trata del actor que accederá a las distintas opciones de la aplicación.
- **Descripción.** Descripción resumida de las razones y resultados del caso de uso.
- **Flujo normal.** Indica el camino que satisface los intereses del personal involucrado, proveyendo una descripción detallada de las acciones del usuario y las respuestas del sistema.
- **Flujo alternativo.** Indica todo los escenarios o bifurcaciones, incluyendo éxito o fracaso. En combinación con el flujo normal debería cubrir todos los caminos posibles de ejecución.
- **Notas.** Se incluirá cualquier comentario adicional al caso de uso.

Cuadro A.1: **Identificador:** 001

Nombre	Indicar ruta de grabación
Actor principal	Experto en Biomecánica
Descripción	El usuario indica la ruta en la que desea almacenar las imágenes
Flujo normal	1.El usuario pulsará sobre el botón ruta 2.En el explorador de archivo elegirá la ruta deseada 3.Se mostrará una ventana emergente con la ruta 4.Aparecerá la ruta indicada en el programa
Flujo alternativo	
Notas	

Cuadro A.2: **Identificador:** 002

Nombre	Comenzar grabación
Actor principal	Experto en Biomecánica
Descripción	El usuario comenzará la sesión de grabación
Flujo normal	1.El usuario pulsará sobre el botón de empezara grabar 2.Se crearán 3 carpetas en la ruta indicada que contendrán las imagenes RGB, Profundidad e Infrarrojo. 3.Saldrán tres ventanas emergentes mostrando las capturas 4.Cambiará el mensaje del estado de la cámara
Flujo alternativo	1a. Si el usuario no ha elegido ruta saldrá un mensaje impidiendo la grabación 1b. Si la cámara no está funcional no comenzará la grabación
Notas	

Cuadro A.3: **Identificador:** 003

Nombre	Parar grabación
Actor principal	Experto en Biomecánica
Descripción	El usuario parará la sesión de grabación
Flujo normal	1.El experto pulsará el botón de parar o la tecla de acceso rápido “q”. 2. Las ventanas emergentes se cerrarán. 3.El estado de la grabación cambiará a parado.
Flujo alternativo	
Notas	

Cuadro A.4: **Identificador:** 004

Nombre	Comprobar ruta de grabación
Actor principal	Experto en Biomecánica
Descripción	El experto indicará la ruta donde se encuentran las imágenes a procesar
Flujo normal	1.El usuario escribirá la ruta de las imágenes 2.Pulsará en el botón de comprobar 3.Se activará el botón de iniciar el procesado
Flujo alternativo	2a. Si la ruta es incorrecta no permitirá continuar 2b.Si la ruta no contiene las 3 carpetas no permitirá continuar 2c.Si las carpetas no contienen el tipo de imágenes requerido no permitirá continuar
Notas	

Cuadro A.5: **Identificador:** 005

Nombre	Comenzar procesado
Actor principal	Experto en Biomecánica
Descripción	El experto iniciará el procesado de imágenes
Flujo normal	1. El experto pulsará sobre el botón de comenzar 2.La pantalla cambiará a la de procesado.
Flujo alternativo	
Notas	Mientras no se obtenga una ruta correcta no se podrá avanzar de esta pantalla

Cuadro A.6: **Identificador:** 006

Nombre	Avanzar fotograma
Actor principal	Experto en Biomecánica
Descripción	El usuario irá avanzando uno a uno cada fotograma para su procesado
Flujo normal	1.El usuario apretará la barra espaciadora para cambiar de fotograma 2.A cada fotograma saldrá una ventana emergente con las imágenes procesadas 3.Se animará el modelo 3D en cada fotograma 4.Para cada fotograma se darán los datos obtenidos 5.Al llegar al último fotograma se volverá a la pantalla de carga de imágenes.
Flujo alternativo	1a.Si alguno de los fotogramas no es válido se volverá a la pantalla inicial de carga de imágenes
Notas	

Cuadro A.7: **Identificador:** 007

Nombre	Parar procesado
Actor principal	Experto en Biomecánica
Descripción	El usuario irá avanzando uno a uno cada fotograma para su procesado
Flujo normal	1.El usuario apretará la el botón de parar ejecución 2.La ejecución se detendrá y la aplicación volverá a la pantalla inicial
Flujo alternativo	

Apéndice B

Uso de la herramienta de grabación

A continuación se detallarán las acciones que se le pueden llevar a cabo con la herramienta de grabación, teniendo detalle en todos los posibles casos.

Una vez se haya ejecutado la aplicación se obtendrá una vista como la de la figura B.1. En este punto habrá 3 botones para el usuario:

- Ruta
- Grabar
- Parar

Es requisito indispensable que el usuario elija una ruta para poder empezar a grabar, en caso de que que se trate de realizar la acción se lanzará una ventana emergente instando a que, por favor, se escoja dónde se desea almacenar las imágenes antes de empezar con el proceso. Tal y como se puede ver en la figura B.2

Por otro lado, si se trata de de parar una grabación mientras ésta no haya empezado todavía se lanzará otra ventana emergente indicando que antes de poder parar una grabación se debe empezar primero el proceso.

Por tanto la única opción que queda al usuario es pulsar sobre el botón de

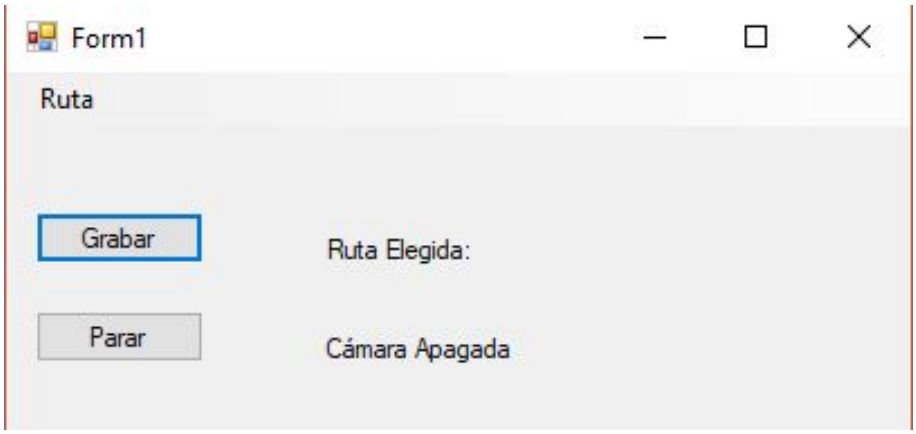

Figura B.1: Estado inicial de la interfaz de grabación.

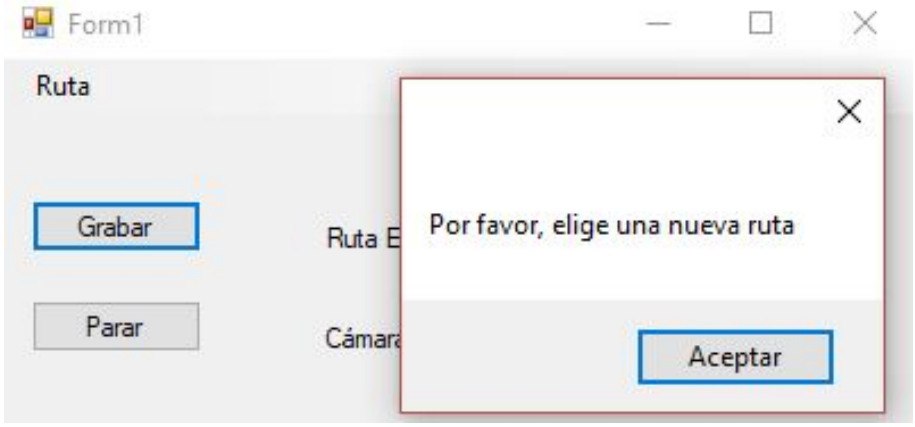

Figura B.2: Mensaje de error al intentar iniciar la grabación sin una ruta.

Ruta, al hacerlo se abrirá un explorador de archivos que permitirá al usuario navegar por las carpetas de su equipo personal, indicando dónde prefiere almacenar las imágenes. Este explorador además permite crear carpetas en la ruta seleccionada tal y cómo muestra la Figura B.3. Al hacerlo se mostrará un mensaje de que la elección ha sido un éxito y junto al botón de grabar se mostrará la ruta elegida por el usuario. Figura B.4

Ahora el usuario tiene la opción de empezar una sesión, al darle al botón de grabar saldrán tres ventanas emergentes mostrando la grabación en profundidad (imagen tono azulado), infrarrojo (escala de grises), y la de RGB. Aunque la imagen a color tiene una resolución de alta definición (1920x1080) se ha decidido reescalarla a la mitad de su tamaño a la hora de mostrarla por pantalla, permitiendo ver todas las imágenes a la vez, junto a la interfaz. Figura B.5:

Si durante la grabación pulsamos el botón de parar o la tecla especial "q"se parará la ejecución y volveremos al mismo estado que la figura B.1

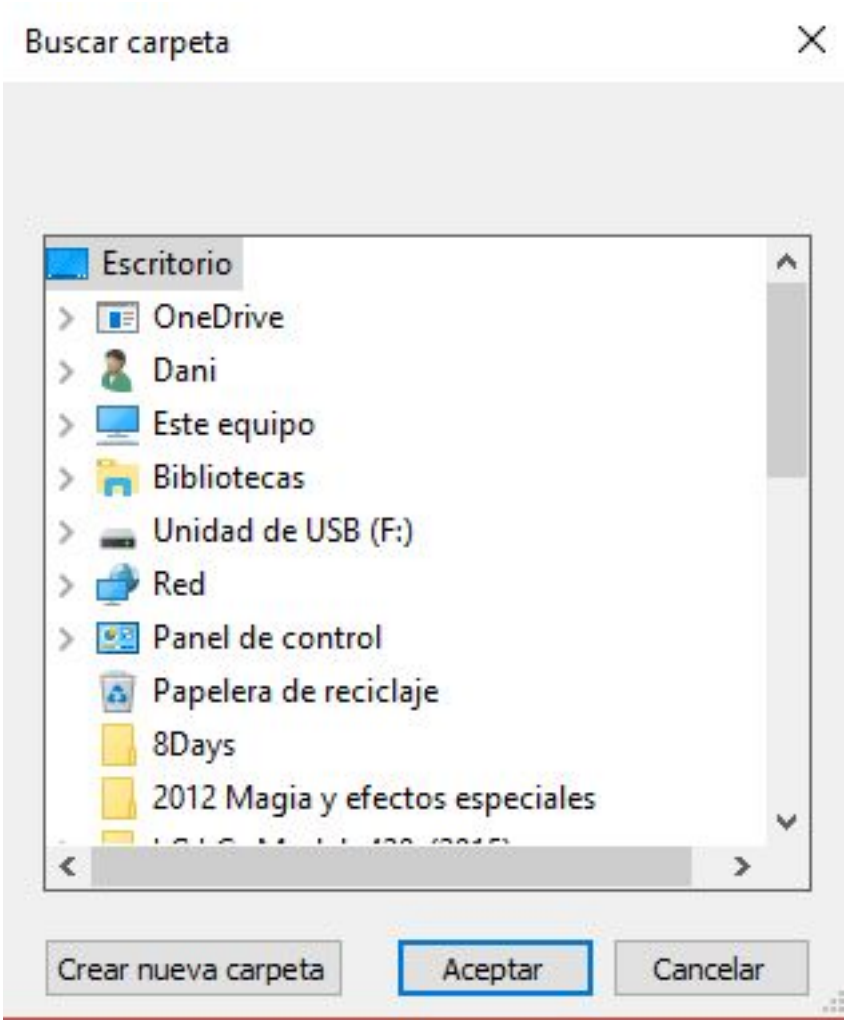

Figura B.3: Explorador de archivos.

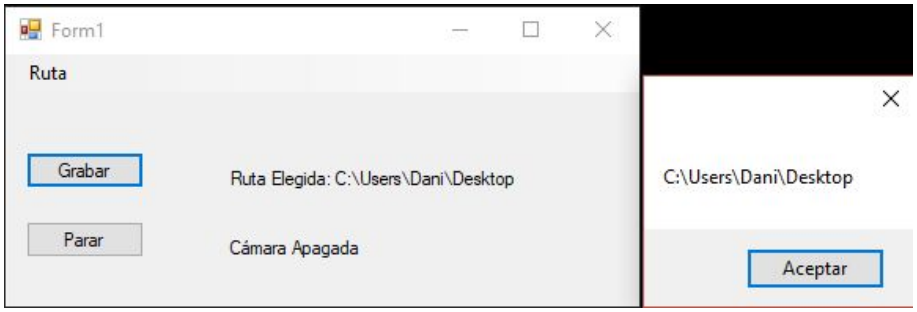

Figura B.4: Estado tras elegir la ruta.

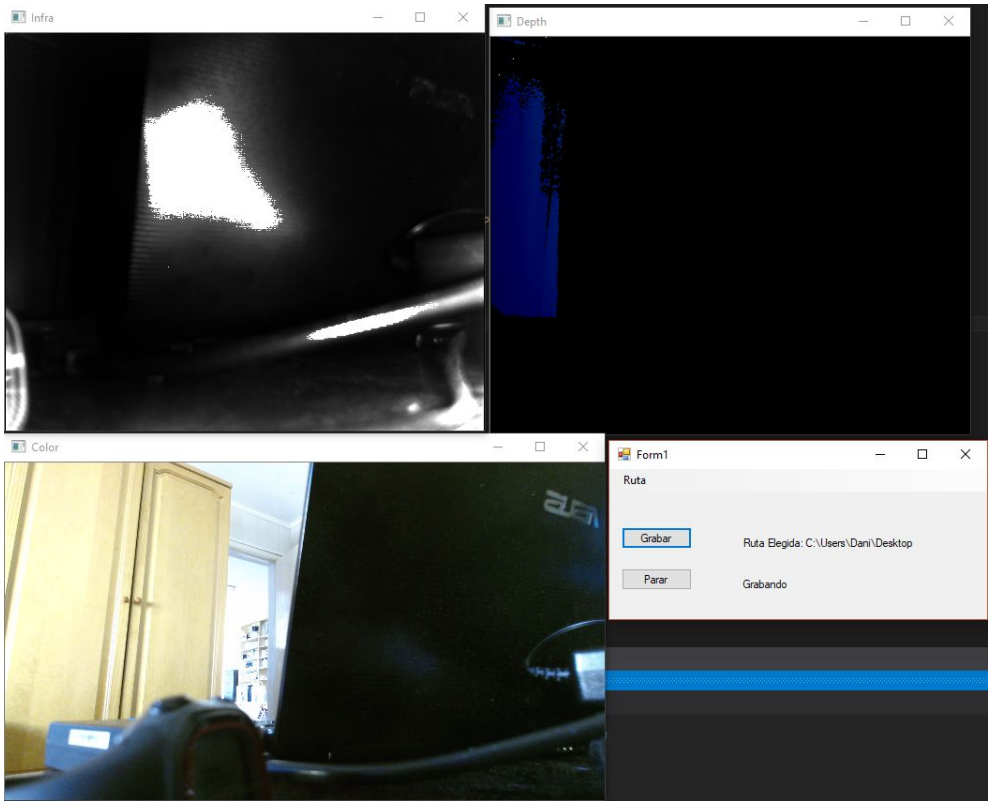

Figura B.5: Estado durante la grabación.

Apéndice C

Uso de la herramienta de detección y procesado

A continuación se procederá a detallar como hacer uso de la herramienta de detección y procesado de imágenes.

Para que la aplicación funcione es requisito necesario que la carpeta de **SizeYourBike** esté en la misma ubicación que el ejecutable. Figura C.1

Tal y como se puede apreciar en la figura C.2 la escena principal del programa consta de dos botones y un cuadro de texto. En el cuadro de texto se debe introducir la ruta donde se contengan las carpetas con las imágenes (obligatoriamente tienen por nombre: color, Infra, prof). Una vez introducida una ruta se puede pulsar el botón para comprobar dicha ruta. En caso de que

Figura C.1: Ejecutable y carpeta del proyecto.

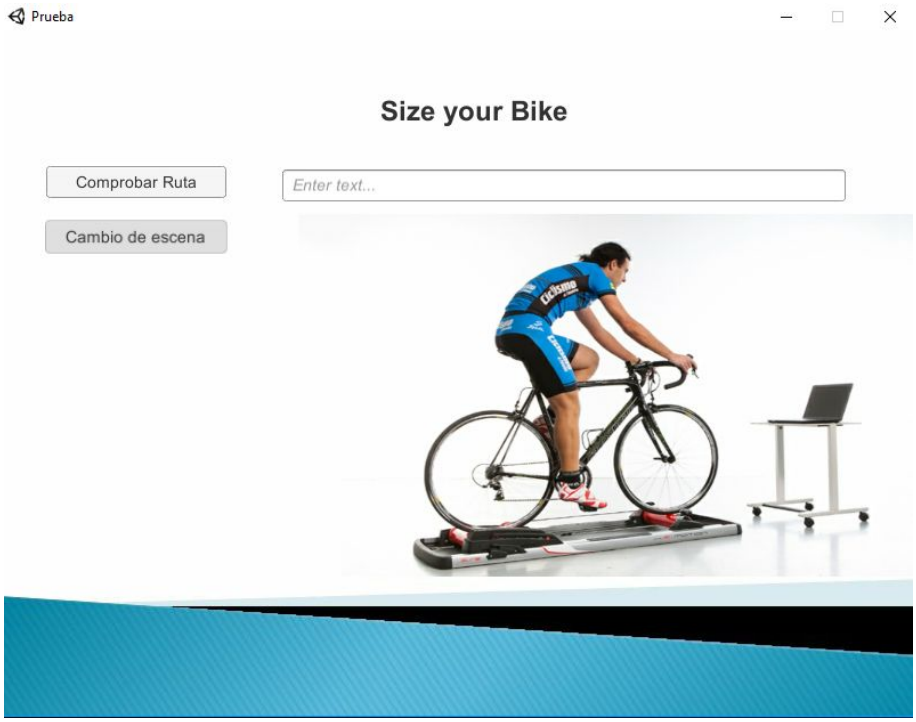

Figura C.2: Escena inicial de la aplicación.

no sea una ruta válida (figuras C.3 y C.4) el botón que permite el cambio de escena permanecerá inactivo, impidiendo que el programa pueda recibir por entrada datos inválidos.

Si por el contrario, y fijándonos en el ejemplo de las figuras C.5 y C.6, se introduce una ruta correcta, tras pulsar el botón de comprobar la ruta, el botón para cambiar la escena se volverá activo y permitirá la transición a la pantalla de muestra de datos.

Como se puede observar en la figura C.7, la pantalla de muestra de datos comienza con un modelo 3D en una posición determinada y con la tabla con unos valores por defecto, dichos valores están fuera de rango lógico para mostrar que aún no se ha empezado el procesado.

Para comenzar con el tratamiento será necesario pulsar la tecla “espacio”, esto hará que surja una ventana emergente en la que, tal y como se muestra en la figura C.8, deberemos seleccionar la zona en la que se encuentra el paciente evitando las marcas que tiene pegadas al cuerpo (esto indicará al programa

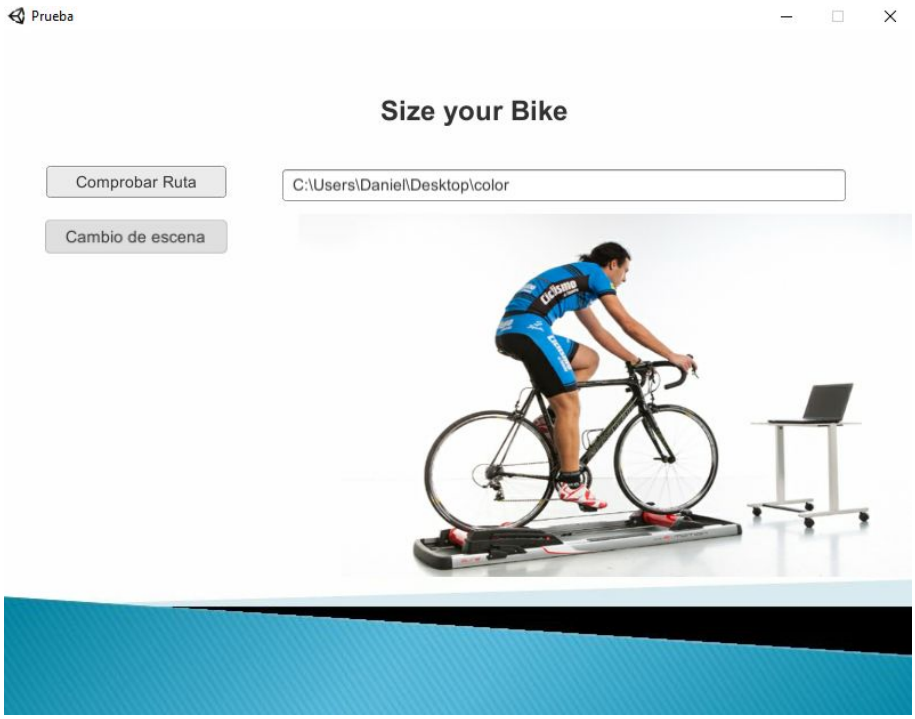

Figura C.3: La aplicación no cambia ante una ruta incorrecta.

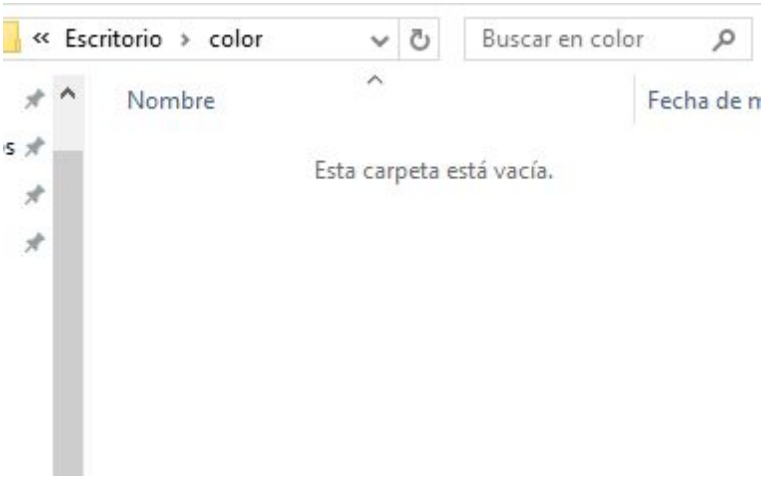

Figura C.4: Carpeta con ruta incorrecta.

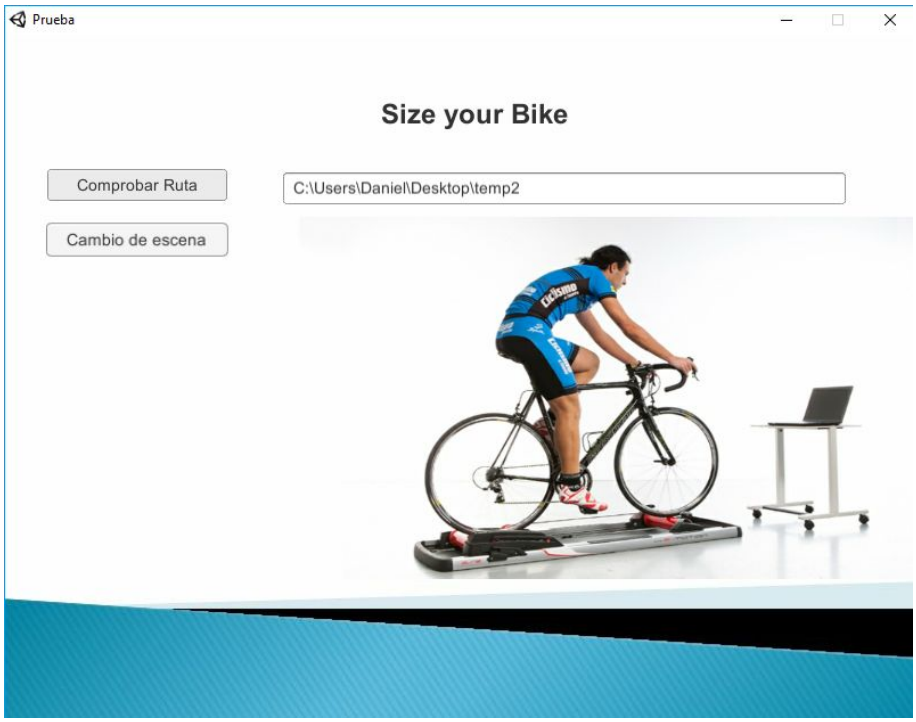

Figura C.5: Activación del botón de cambio de escena ante una ruta correcta.

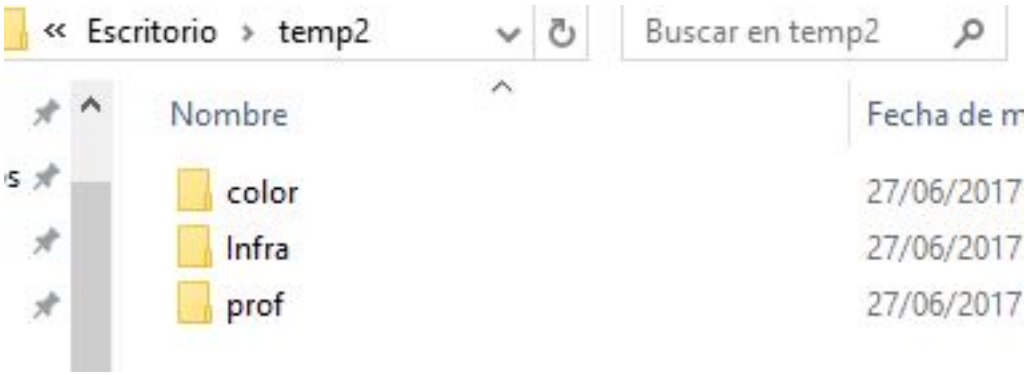

Figura C.6: Se muestra una ruta correcta.

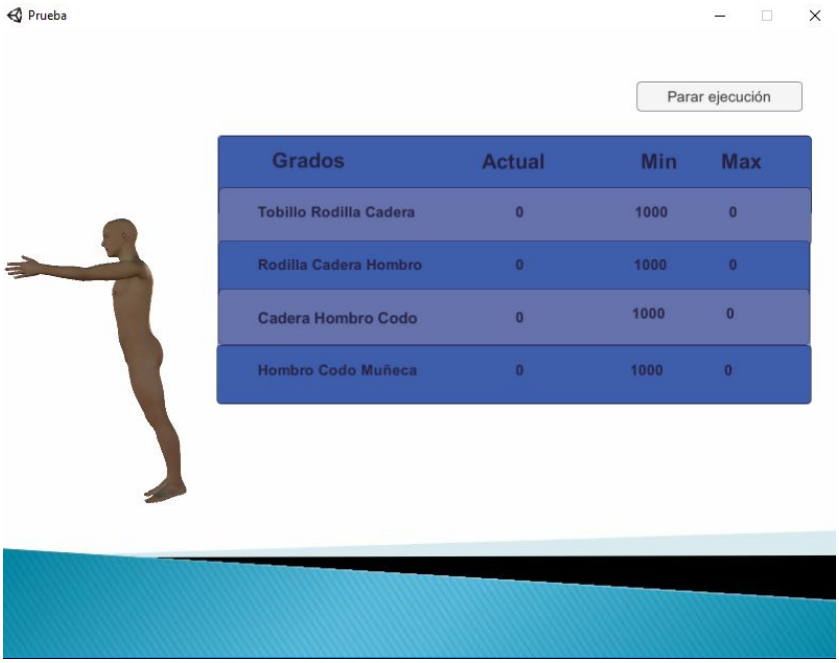

Figura C.7: Pantalla de muestra de datos.

la profundidad en la que se encuentra el sujeto). Una vez hechas las selecciones que se consideren necesarias se pulsará la tecla .ESC"para comenzar el procesado.

Tal y como se ve en la figura C.9 se han actualizado los datos mostrados en la tabla, mostrando los grados del fotograma actual y los valores mínimo y máximos que se han obtenido hasta este punto del procesado (se considera que si en algún momento uno de estos valores alcanzados se encuentra fuera del rango óptimo, el sillín se encuentra mal regulado). Además, al mismo tiempo, se muestra una miniatura del fotograma analizado tanto en RGB como con la detección de sus marcas para ir comparando con el modelo 3D, (figura C.10.

Una vez alacanzado el límite de fotogramas que se encuentra en la ruta proporcionada el programa volverá automáticamente a la pantalla inicial, en caso de que se desee parar le ejecución (por haber introducido una ruta con una sesión incorrecta o por haber alcanzado un fotograma en el que ya se han encontrado valores no deseados) se puede volver a la escena inicial haciendo uso del botón de "Parar ejecución".

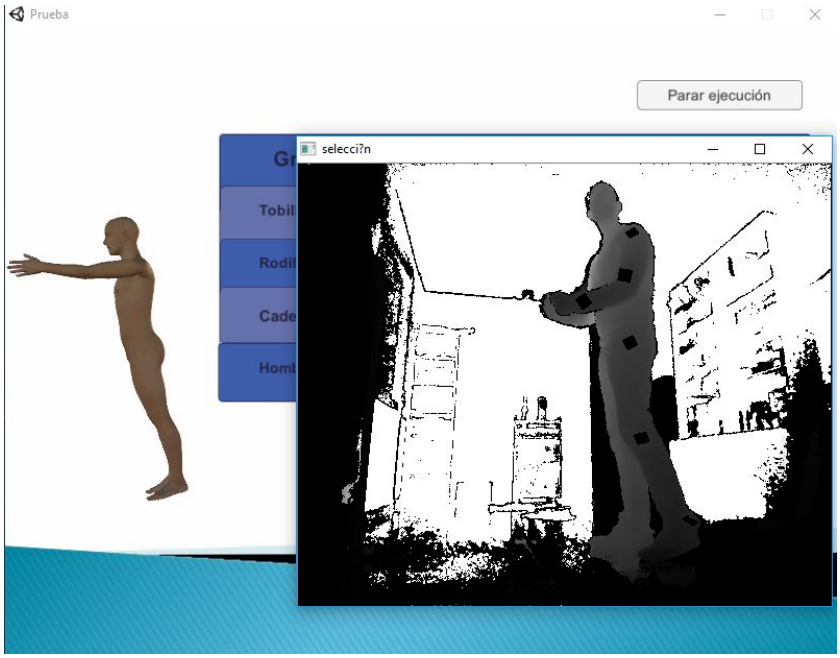

Figura C.8: Búsqueda de profundidad a procesar.

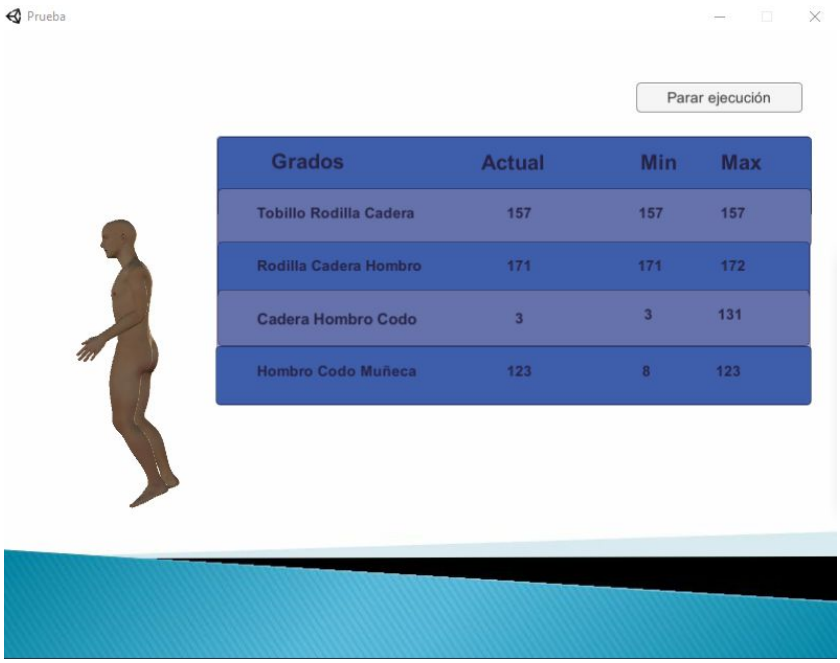

Figura C.9: Estado tras avanzar fotogramas.

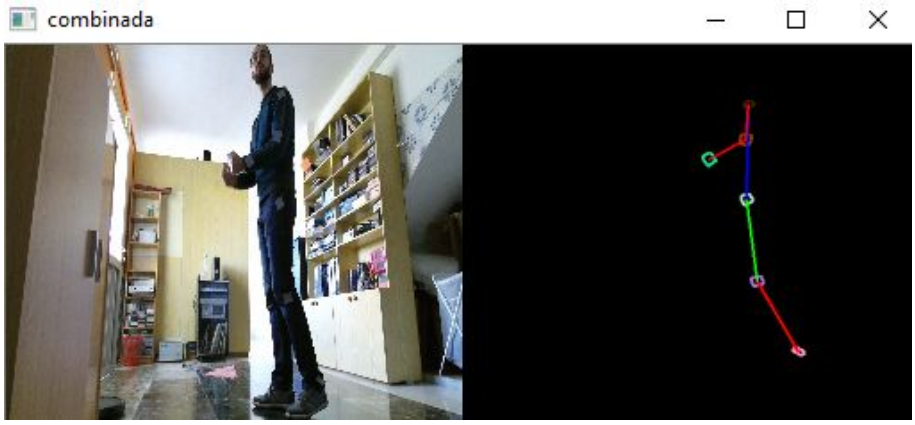

Figura C.10: Muestra de imágenes para comparar junto al modelo 3D.