ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
Ingeniería de la Salud

Algoritmos de detección y localización de puntos de referencia faciales en tiempo real para guiado en cirugía

Detection and localization algorithms of facial landmarks in real time for guided surgery

Realizado por
Martín Axelrad Tinoco
Tutorizado por
Enrique Nava Baro
Mario Padilla Delange
Departamento
Ingeniería de Comunicaciones

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Junio, 2018

Fecha defensa:
El Secretario del Tribunal

Resumen: La tasa de éxito de las cirugías endoscópicas se ha visto incrementada en gran medida tras la implementación de sistemas de guiado por ordenador. Estos permiten localizar con precisión la punta del endoscopio en la anatomía del paciente. evitando dañar tejido contiguo y la búsqueda de estructuras no visibles por el endoscopio. Es habitual encontrar, en el mercado de guiado en cirugía endoscópica, dispositivos centrados en sus características técnicas, dejando de lado la usabilidad, accesibilidad y comodidad del facultativo. Suelen ocupar grandes cantidades de espacio, tener precios elevados y requerir un campo de visión muy amplio, entorpeciendo la labor del personal.

En este trabajo se desarrolla un sistema de guiado para cirugía endoscópica nasosinusal capaz de localizar la orientación del endoscopio en el interor de las fosas nasales del paciente. Usando técnicas de visión por computador, el sistema detecta caras y puntos faciales característicos, generando un modelo completamente personalizado para cada paciente, sobre el cual se estiman las orientaciones del endoscopio a tiempo real. Al ubicar el dispositivo de grabación en el mismo endoscopio, se minimiza el espacio ocupado, las limitaciones para el personal, el aprendizaje previo y el contacto directo con el paciente.

Palabras claves: cirugía guiada por ordenador, visión por computador, endoscopia nasal, estimación de la pose, reconstrucción facial tridimensional

Abstract: The success of endoscopic surgery depends heavily on the implementation of computer-guided systems. Thanks to these, the physician can visualize the orientation of the endoscope in the patient's anatomy. This makes it possible to perform the operations with greater precision, avoiding damage to contiguous tissue and the searching of structures not visible through the endoscope. It is usual to find in the market of computer assisted surgery devices focused on their technical characteristics, leaving aside the usability, accessibility and comfort of the physician. These systems usually occupy a large amount of space in the operating room and require a very wide field of vision, obstructing the work of the physician.

In this work a guidance system is developed for endoscopic nasal surgery capable of locating the orientation of the rigid endoscope inside the patient's nostrils. Using computer vision techniques, the system detects the face and characteristic facial features using a camera located in the endoscope. A completely customized model is generated on which the endoscope orientations are estimated in real time. This work focuses on minimizing the usual procedure, avoiding the need for training, eliminating any limitation of movement of the physician and direct contact with the patient.

Keywords: computer assisted surgery, computer vision, endoscopic nasal surgery, pose estimation, three-dimensional facial reconstruction

Índice general

Capítulo 1

Introducción

1.1. Introducción

Las cirugías guiadas por computador han vivido grandes avances con los años desde que los primeros experimentos en este área se realizaron en el hospital universitario de Aachen en 1986. Actualmente es innegable la repercusión de estas tecnologías en muchos procedimientos quirúrgicos, con una estimación de volumen de mercado de casi 5 mil millones de dólares para 2021 [1].

El principal procedimiento que se ha visto afectado por la introducción de estas tecnologías es la cirugía neurológica con un 26 % [27] de mercado en las cirugías guiadas por computador. En segundo lugar se encuentran las cirugías endoscópicas, en la que se incluye la laparoscopía.

Las cirugías endoscópicas endonasales son aquellas encargadas de realizar operaciones en el interior de las fosas nasales sin necesidad de crear incisiones destinadas únicamente a la introducción del instrumental, al hacer uso de los orificios nasales. Las últimas tecnologías para el guiado en cirugías endoscópicas endonasales actuales presentan problemas comunes como su elevado coste, su gran tamaño, las limitaciones al facultativo y el contacto directo con el paciente.

Por lo tanto el mercado de la cirugía guiada por imagen tiene, tal como se mostrara en el estado del arte, un hueco para un dispositivo que libere al cirujano de ocupar un gran espacio en el quirófano, evitando las molestias al mínimo al dejar de limitar los movimientos del facultativo en la operación. En resumen, un dispositivo que modifique al mínimo el instrumental y el procedimiento quirúrgico actual.

La propuesta de este trabajo es un programa capaz de aplicar parte de las tecnologías actuales a un dispositivo pequeño, localizado encima del endoscopio, en el extremo opuesto a la cámara introducida en las fosas nasales. Este dispositivo consiste en una pequeña cámara apuntada hacia la cara del paciente, recolectando la variación de esta con el paso de la operación y un dispositivo que procese su información y nos ofrezca la localización a tiempo real del endoscopio respecto al paciente.

1.2. Antecedentes

En este proyecto, un análisis del estado del arte es importante para fundamentar la necesidad de un nuevo producto en un campo tan antiguo y explorado, que llega hasta nuestros días con alternativas muy innovadoras.

Esta sección está dividida en tres partes: historia de la endoscopia en cirugía, inicios de la cirugía endoscópica asistida por computador y el estado actual de esta tecnología y la realidad aumentada.

1.2.1. Historia de la cirugía endoscópica (1900 - 1975)

El primer acceso al seno maxilofacial se remonta a 1867 por Mollineti Un siglo después la cirugía endoscópica nasosinusal sufriría una verdadera revolución causada por Harold H. Hopkings al introducir un endoscopio compuesto por lentes situadas en una varilla, o en inglés *'rod-optic telescope'* en las fosas nasales de sus pacientes [13, p. 352].

El primer caso de visualización de las cavidades nasales se le otorga a Hirschmann en 1901 mediante el uso de un citoscopio, desarrollado por Nitze en 1879 y modificado

para poder realizar una endoscopia nasal y de los senos contiguos. En 1903 Hirschmann publicó sus avances en lo que fue el principio de las endoscopias endosinusales. En sus publicaciones introducía el instrumento a través de una incisión creada previamente tras la extracción de un diente canino (o colmillo) o entre los dientes cuando se sospechaba de una enfermedad en el seno maxilar. Aunque estos avances fueron los primeros pasos para la cirugía endoscópica, Hirschmann no contemplaba este tipo de operaciones como tratamiento, solo para diagnóstico[18, pp. 4-9].

En 1902 Reichert construye un instrumento de características similares, con el cual informó de como había tratado a tres pacientes con fístulas del hueso alveolar. En su caso, realizo pequeñas incisiones usando su instrumento para realizar tareas como cauterizar, drenar e irrigar algunas zonas. Su instrumento tenía un diámetro 2 mm mayor al de Hirschmann, el cual tenía 5 mm de diámetro.

Tras varios avances, fue en 1925 cuando en Nueva York, Matlz construye un instrumento mejorando la óptica, con el que accedía a través de la fosa canina o el meato inferior. A él se le atribuye el primer uso de la palabra "sinoscopia" como nombre para este tipo de exploración. También Portmann senior, en Francia y Botey en España mencionaron la posibilidad de la inspección del seno esfenoidal. Poco a poco esta técnica va ganando adeptos, aunque en el mismo año Zerniko señaló las dificultades de estos instrumentos para acabar asistiendo en un procedimiento serio, como la dificultad para esterilizarlo, la distorsión observada en las imágenes, y los reflejos de luz.

Tras el instrumento de Hopkins se produjeron muchas modificaciones en distintos países, además de la publicación de artículos que mostraban la eficacia de estos instrumentos para el diagnóstico de los senos. El desarrollo de la fibra óptica sustituyó a los sistemas de iluminación, que solían consistir en una bombilla en el interior del instrumento, consiguiendo una fuente de luz externa al cuerpo humano. Con esto en 1964 Timm consiguió las primeras fotografías a color del seno maxilofacial.

Con estas nuevas mejoras, la sinoscopia se propagó por Escandinava, gracias a Illum

y Japensens en 1972, por Holanda, gracias a Butter en 1973 y por Suiza gracias a Terrier en 1975, diagnosticando, documentando los procedimientos y asentando la endoscopia endosinusal tanto para diagnóstico como para tratamiento.

1.2.2. Historia de la cirugía endoscópica asistida (1975 - 2000)

Los primeros experimentos realizados usando el guiado por computador para cirugía en otorrinolaringología se realizaron en la universidad tecnológica y en el hospital universitario de Aachen en 1986. El experimento consistía en el uso de un robot pasivo, compuesto por seis articulaciones rotacionales, que contenían potenciómetros capaces de medir la rotación realizada en cada uno. Pasándole esta información a un ordenador, el sistema era capaz de obtener la posición de la punta del robot. Como inconvenientes, el sistema contaba con poca movilidad, un manejo complicado y no tenía la precisión suficiente[12].

Los investigadores de Aachen siguieron investigando en una unidad óptica compuesta por seis articulaciones contrabalanceadas conectada a un microprocesador dedicado. Al paciente se le realizaba una tomografía computarizada, con cortes de 2 mm de profundidad, con cuatro marcadores situados en la cabeza como puntos de referencia. Una vez obtenidas las imágenes médicas, el robot se localizaba las coordenadas de los marcadores tocándolos con su punta. Con los puntos de referencia, y las coordenadas tridimensionales de estos, el sistema era capaz de recuperar las localización de la punta del robot en las imágenes médicas.

En 1991 el grupo Aachen emitió un informe realizado en 200 procedimientos quirúrgicos evaluando los beneficios y los riesgos de la cirugía asistida por computador o CAS. Como riesgos, la recopilación de un modelo tridimensional mediante tomografía computarizada requería una radiación adicional sobre el paciente, y el sistema necesitaba más recursos humanos para su funcionamiento. Como beneficios, se aumentaba drásticamente la cantidad de información quirúrgica, lo que derivaba en una cirugía mas eficiente y una reducción de los riesgos para el paciente.

Krybus et al, también de Aachen, desarrollaron un sistema de localización por infrarrojos en 1991. Este sistema consistía en tres cámaras infrarrojas situadas encima de la mesa de operaciones que, mediante triangulación, ubicaban al instrumento mediante ledes emisores de infrarrojos situados en la parte distal y proximal del instrumento, así como en la cara del paciente.

El Viewing Wand es un sistema basado en brazos robóticos, con sondas esterilizables desmontables. Todo esto conectado a un ordenador situado en el quirófano, contando con un monitor de gran resolución. Este sistema contaba con el mismo mecanismo del primer experimento realizado por Aachen, con sondas de distintos tamaños, aunque finalmente se optó por la más pequeña al ser mas cómoda y fácil de usar. Además, se desarrollo un nuevo tipo de sonda curva, que permitía el acceso a las zonas que la pequeña no podía alcanzar, como los huecos de la zona superior frontal del paciente.

El Surgicom es una variante de la Viewing Wand, que contaba con seis transductores eléctricos que medían los cambios rotacionales en las articulaciones usando un tipo de brazo llamado electrogoniómetro, con una precisión de 0.05 grados. La precisión rotacional requería ser pequeña al obtener la posición y la orientación del brazo con seis rotaciones sucesivas.

El OptroTrak Infrared System usaba un software modificado del Surgicom, y su única diferencia es el brazo robótico con unas sondas emisoras de luz infrarroja. Esta emisión era detectada por un receptor infrarrojo, que además detectaba una banda craneal situada en el paciente que servía de referencia. La sonda consistía en un puntero con forma de lápiz construido en aluminio que contaba con un aro de 24 diodos infrarrojos, permitiendo la detección de los tres puntos necesarios para su localización. La banda craneal, compuesta de neopreno, contaba con 6 diodos emisores montados en forma rectangular.

El FlashPoint 5000 3D Optical Localizer contaba con el mismo funcionamiento del OtroTrak Infrared System, pero en este caso las emisiones eran detectadas por tres sensores de carga acoplada. Este dispositivo estaba situado a 1.5 metros y en un angulo de

45º respecto al paciente. Cada uno de los tres sensores tenía capacidad para observar un angulo de 30º de la zona quirúrgica. Las emisiones que llegaban al sensor generando una carga eléctrica, que una vez medida permitía localizar la posición del instrumento.

Por último, el InstaTrak System es un dispositivo que consiste en unos auriculares con una banda que contiene un transmisor electromagnético situado justo delante de la frente del paciente. Este sistema no se ve afectado por el ruido externo causado por instrumentos de metal, prótesis dentales u otros factores ambientales. Los auriculares, que son desechables, contienen seis bolas metálicas que sirven como referencia para la fuente electromagnética. El sistema era capaz de reconocer el instrumental mediante otras bolas metálicas situadas en este. Este sistema evitaba la colocación de marcas fiduciales en el rostro del paciente al tener los puntos de referencia incluidos en los auriculares.

1.2.3. Estado actual de la cirugía asistida por computador y la realidad aumentada (2000 - 2017)

Muchos de los dispositivos actuales, son modificaciones y mejoras de los anteriores. En este apartado se reúnen algunas alternativas que se acercan a la propuesta de este trabajo, o alguno de los sistemas más avanzados.

En 2003 Sebastian Vogt et Al desarrollaron un sistema de realidad aumentada destinada a cirugías medicas. El dispositivo contaba con dos cámaras con visión estereoscópica y una cámara infrarroja encargada de detectar unos marcadores que podían ir tanto en una estructura solida o a la instrumentación necesaria en la cirugía. Todo esto se montaba en la cabeza del facultativo, consiguiendo el mismo punto de vista que este, que se colocaba unas gafas de realidad virtual. En estas gafas de realidad virtual, el usuario observaba las imágenes obtenidas por las dos cámaras con visión estereoscópica y, mediante la localización de puntos de referencia por la cámara infrarroja, un modelo tridimensional sobrepuesto a las imágenes reales, que creaban la sensación de transparencia del paciente, pudiendo ver su interior [42].

También en 2003, un sistema parecido al anterior fue desarrollado por Wolfgang Birkfellner et Al [14]. Un sistema de dos cámaras localizaba distintos puntos de referencia, esta vez situados sobre la cabeza del paciente. El sistema incluía un algoritmo de detección de colisiones, por el cual la instrumentación usada en la cirugía podía ser localizada y se mostraban avisos de proximidad a estructuras de la cabeza o los puntos fiduciales.

En 2005, Kozo Konishi et al desarrollaron un sistema de laparoscopia con realidad virtual [29]. Este sistema estimaba el estado actual de los órganos internos al ser estos elementos no rígidos. La solución fue localizar la punta del endoscopio mediante sensores electromagnéticos a la vez que se realizaba una técnica de visualización del interior del cuerpo humano mediante ultrasonidos, a modo de ecografía que detectara los movimientos y variaciones de estos. Con esto conseguían sobreponer imágenes de la posición de los instrumentos y los órganos del paciente, proyectando imágenes de un modelo tridimensional de los órganos encima de la piel a modo de pantalla. Este dispositivo permite visualizar el interior del organismo a plena vista del facultativo.

En 2007, Jan Fischer et Al desarrollaron un sistema híbrido que permitía localizar puntos fiduciales del paciente además de localizar la propia cámara que los detectaba [20]. El sistema está compuesto por un detector de infrarrojos, parecido al FlashPoint, que detecta la pose de una cámara que lleva unos ejes de referencia reflectores de infrarrojos. Con esto, la cámara localizaba los puntos de referencia al tocar con el instrumental la zona de la imagen que se quería localizar. Con esto y mediante template matching el sistema era capaz de reconocer los puntos por parte de la cámara, y la pose de la cámara respecto al sistema infrarrojo.

En 2007, un dispositivo parecido al anterior fue desarrollado por Chen Jing et Al [26]. Este contaba con marcadores fiduciales reflectantes de infrarrojos, que eran localizados por una cámara infrarroja, con un eje de coordenadas asignado infrarrojo, que a su vez era detectado por tres cámaras situadas en el techo del quirófano. Con esto se obtenía un sistema parecido al anterior, pero sin la necesidad del template matching, teniendo como referencia los puntos fiduciales de la cara.

En 2010, un sistema ideado por Hongen Liao et al se encargaba de la localización de puntos fiduciales del paciente, a la vez que se le realizaba una MRI [34]. Este dispositivo consiste en una compleja estructura de espejos situados entre la cara del paciente y el cirujano, que refleja al paciente con imágenes de realidad virtual superpuestas. Además, un sistema de cámaras infrarrojas se encargaba de localizar tanto la cabeza del paciente, como la del cirujano y la instrumentación usada.

Un caso a destacar se desarrollo en 2016 por Simon Leonard et Al para la cirugía endoscopica endosinusal [32]. El sistema se encargaba de usar una técnica de visión por computador llamada *Structure from Motion*, que reconstruye escenas en tres dimensiones mediante varias imágenes consecutivas que sean representativas de la estructura tridimensional del objeto que se quiere reconstruir. Mediante las imágenes obtenidas por el endoscopio, se hacia una reconstrucción de las vías aéreas, creando una nube de puntos en la que posteriormente se localizaba la imagen actual del endoscopio.

1.3. Anatomía y fisiología de las fosas nasales

Estructuralmente el esqueleto de la nariz es tanto óseo como cartilaginoso. Además presenta mucosa, un dispositivo musculofacial y unas cubiertas cutáneas. Los elementos óseos que conforman la nariz se encuentran en la parte superior de la nariz y en la separación de las fosas nasales y están compuestos por las apófisis frontales del maxilar, la escotadura nasal del maxilar y los huesos nasales.

La composición cartilaginosa de la nariz, que le aporta soporte estructural y flexibilidad, está ubicada en su mayoría en la base de la nariz y se compone de cartílago hialino. Los cartílagos que la componen son: el cartílago del tabique, los cartílagos nasales laterales, los cartílagos alares mayores y otros cartílagos nasales accesorios[23, pp. 422 - 424].

La nariz da paso a las fosas nasales, uniéndose y empezando a desarrollar las paredes de esta. Su composición interna es de mucosa y piel. En la nariz se diferencian dos zonas

separadas por un pliegue de transición llamado limbo nasal, que separa el vestíbulo nasal, recubierto de piel, de la parte superior de la nariz, recubierto de mucosa.

Las fosas nasales cubiertas de mucosa continúan la nariz, y en su extremo más opuesto se abre a la nasofaringe por medio de las coanas, unas aperturas con forma de embudo. Las fosas nasales están separadas por el tabique nasal. En las paredes más distales se encuentran tres protuberancias de hueso esponjoso llamadas cornetes. Entre estos cornetes se encuentran los meatos, todos ellos localizados en la zona turbinal de las fosas nasales[23, p. 994]. Estas estructuras anatómicas se observan en la figura 1.1.

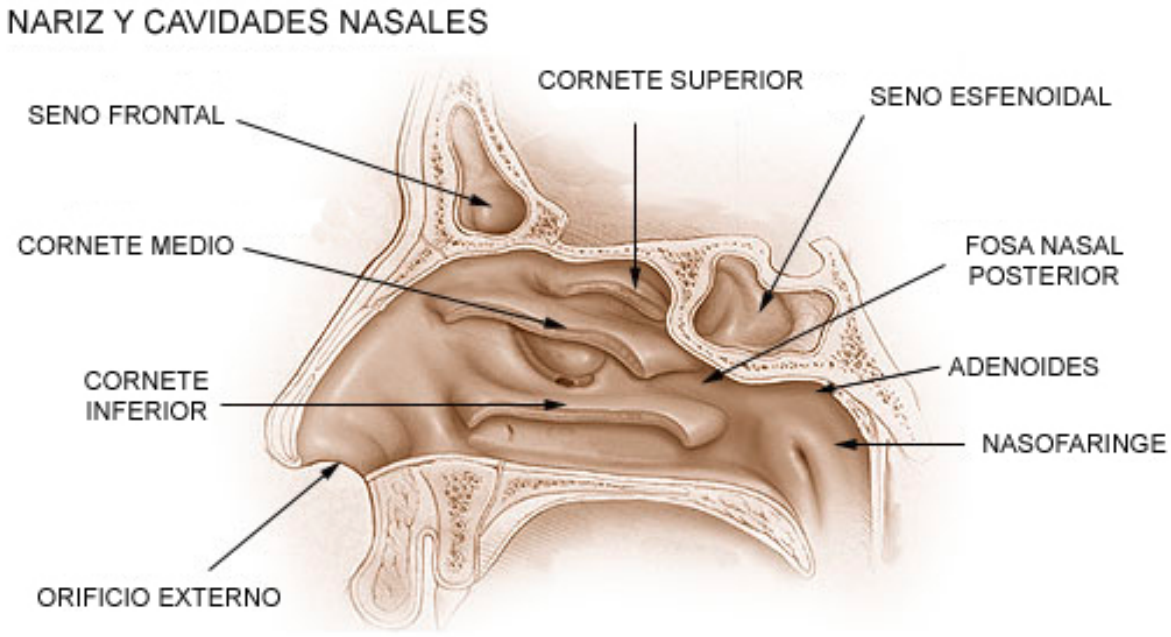

Figura 1.1: Anatomía de las fosas nasales.

Según el tipo de mucosa se localizan dos zonas en las fosas nasales: la zona respiratoria y la zona olfatoria. La región respiratoria es la encargada de calentar, humidificar, dirigir el aire a la región olfatoria y filtrar partículas indeseadas en suspensión. La región respiratoria es la que más interés tiene en este trabajo ya que es la zona más extensa y recorrida durante las operaciones endoscópicas nasales, ya que requieren acceder a otras

estructuras, como los senos, a través de los orificios de la nariz[23, p. 427].

Los senos paranasales son estructuras huecas rodeadas de mucosa localizada en los huesos colindantes a las fosas nasales, estando recubiertos de la misma mucosa que estas. Estas estructuras se crean por invaginación de las fosas nasales, desarrolladas a partir del nacimiento. Crecen desde el nacimiento hasta alcanzar su tamaño máximo en la adolescencia, obteniendo una estructura facial determinada. Dependiendo de cada individuo varían considerablemente en tamaño y por tanto en volumen por lo que la obtención de imágenes medicas personalizadas del paciente es importante en caso de requerir una mayor precisión en la operación[23, pp. 998-999]. Los senos paranasales son el maxilar, frontal, etmoidal y esfenoidal, situados como se observa en la figura 1.2[28].

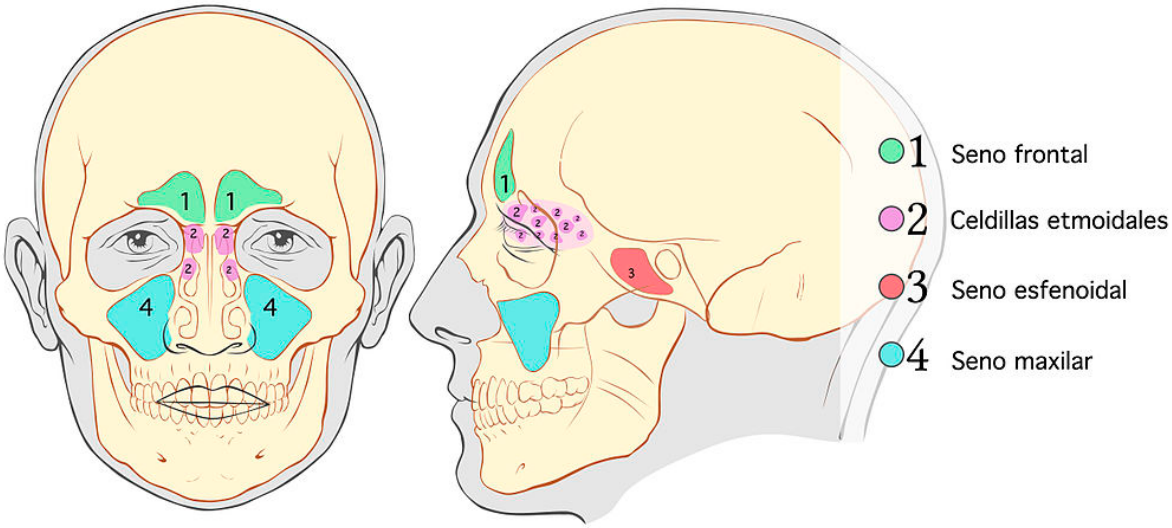

Figura 1.2: Ubicación de los senos paranasales.

1.4. Endoscopio

El endoscopio es un instrumento utilizado para visualizar el interior del cuerpo humano en zonas sin visión directa. Su funcionamiento general es la transmisión de las imágenes capturadas del interior del cuerpo hacia el exterior, iluminado por una fuente de luz en el

extremo introducido [31, pp. 1-8].

Actualmente el endoscopio se presenta en distintos tipos e implementaciones. El endoscopio rígido, que es el usado en este proyecto, es aquel compuesto por un tubo rígido normalmente metálico que se introduce en el cuerpo. Se observa un endoscopio rígido en la figura 1.3.

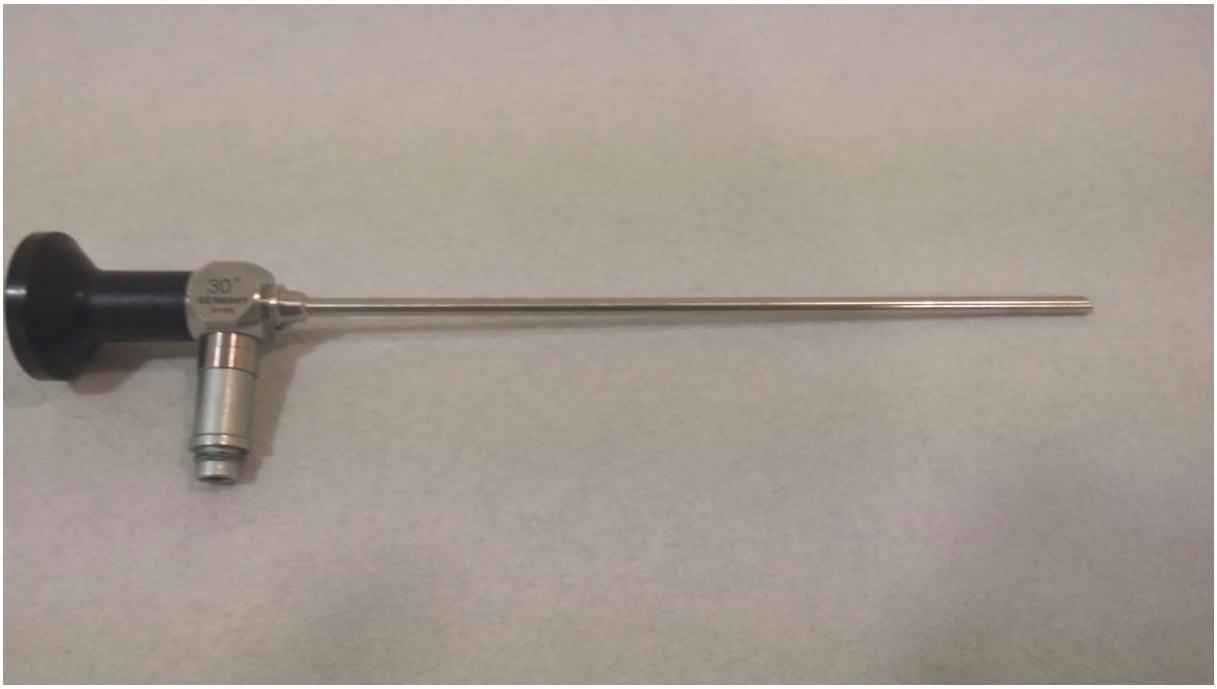

Figura 1.3: Endoscopio rígido.

En el extremo más distal al facultativo se encuentran los dispositivos de iluminación y los de captación de imágenes. La zona es iluminada mediante fibra óptica que introduce la luz desde el exterior, aunque también hay implementaciones con ledes que están ganando terreno por el gran avance de estas tecnologías, además de su mejor rendimiento y menor coste.

Con la miniaturización de las cámaras producidas por la gran solicitud por parte del mercado de los teléfonos móviles, los endoscopios se han beneficiado al poder introducir-

las directamente en el interior de los dispositivos, incluso situándolos en el extremo de este, evitando el uso de lentes y mejorando la captación de imágenes. Actualmente se han creado endoscopios con capacidad de captar imágenes en tres dimensiones del interior humano gracias a la mejora de las tecnologías de estas cámaras.

En el extremo opuesto del endoscopio se encuentra el conector de iluminación, que transmite la luz de su fuente al instrumento. Después de esta se encuentra el acople con la cámara, o el dispositivo de control si las cámaras son internas. También puede encontrarse lentes para la visualización directa de las imágenes por parte del facultativo.

Actualmente el endoscopio se presenta en distintos tipos e implementaciones. Se diferencian dos tipos principales de endoscopios: rígidos y flexibles. En este caso es interesante estudiar el rígido ya que no se podrá localizar la punta de uno flexible sin tener un modelo de este, únicamente basándose en características externas del paciente. Actualmente cualquier endoscopio puede ser transformado en flexible al localizar las cámaras en la punta, no teniendo necesidad de transmitir la luz por lentes.

El endoscopio rígido, que es el usado en este proyecto, es aquel compuesto por un tubo rígido normalmente metálico que se introduce en el cuerpo. Los endoscopios rígidos aportan firmeza al movimiento de las operaciones, permitiendo el acceso de otro instrumental. Normalmente incorporan un prisma en ángulo en su extremo introducido para variar el angulo de visualización de este, pudiendo variar entre 30º y 70º. Para el propósito de este trabajo, se usa un endoscopio rígido con una longitud de varilla de 30 cm, que podrá ser introducido un máximo de 16 cm en el interior del paciente.

1.4.1. Cirugía endoscópica endonasal

La cirugía endonasal ha evolucionado mucho con los años. Antiguamente se requería crear incisiones, por ejemplo, para acceder a través de la fosa canina[18, p. 18], o incluso aberturas mayores en el rostro para acceder a las estructuras deseadas. Estos procedimientos alteraban mucho las estructuras circundantes al área de interés de la operación, por lo

que se aumentaban los daños, las molestias del paciente y el tiempo de recuperación. En casos como la operación cercana al seno esfenoidal estas operaciones podías ser peligrosas por la cercanía al nervio óptico y la arteria carótida[18, p. 10].

La cirugía endoscópica endonasal evita las incisiones necesarias para tener una visión interna directa del paciente. El facultativo introduce por los orificios nasales un endoscopio de manera que se pueda visualizar el interior del paciente, accediendo directamente a las zonas de interés quirúrgico. Con la cirugía endoscópica endonasal el tiempo de preparación para la cirugía se reduce, los riesgos atribuidos al acceso de la zona de interés se reducen al mínimo, las molestias por parte del paciente se reducen y el tiempo de recuperación mejora significativamente. En la figura 1.4 [35] se observa un ejemplo de este procedimiento.

Figura 1.4: Cirugía endoscópica nasosinusal guiada

El principal caso de cirugía endoscópica endonasal en pacientes es la rinosinusitis cronológica, más conocida como sinusitis cronológica, que consiste en algún tipo de in-

Categoría	Ejemplo de patología
Enfermedades inflamatorias	Sinusitis crónica
Desordenes neurorinológicos	Neuralgia
Enfermedades neoplasicas	Tumores en la base del cráneo
Neuropatía óptica	Compresión del nervio ótpico
Enfermedades congénitas	Estenosis de las coanas

Tabla 1.1: Ejemplos de patologías indicadas para su tratamiento mediante cirugía endoscópica.

flamación de las vías aéreas o los senos circundantes. Este tipo de infecciones se tratan cuando los tratamientos antibióticos u otros medicamentos no surgen efecto y se mantiene crónico, entendiendo por esto la prevalencia de la inflamación por mas de tres meses. En la tabla 1.1 se indica los casos para lo que se indica la cirugía endoscópica nasal[2].

La endoscopia endonasal puede realizarse mediante anestesia local o general. En algunos casos se puede implantar un dispositivo médico con forma de balones que permiten expandir el canal de los senos paranasales en un proceso llamado sinuplastia.

El proceso de recuperación depende de cada operación y cada paciente. Normalmente con la inclusión del endoscopio en las cirugías nasales no es necesario la introducción de un tapón nasal. Dependiendo de la complejidad de la operación la incomodidad, el sangrado y la congestión desaparecen en pocos días.

1.4.2. Puntos de referencia faciales

Los puntos de referencia faciales han sido ampliamente estudiados y usados en tecnologías de visión por computador. Abarcan desde campos como la identificación de personas, el alineamiento facial, detectar distintos tipos de características dependiendo de la morfología facial, detección de enfermedades [10], en antropología forense para relacionar un cráneo con imágenes de la persona y mucho más [15].

Los puntos de referencia faciales son aquellos puntos que sean fácilmente reconocibles en la mayoría de las condiciones. Se especifica que tienen que ser detectables independientemente de la expresión facial, la orientación de la iluminación, que representen zonas representativas del modelo tridimensional facial, como salientes o valles diferenciables en distintas posiciones de la cabeza.

Alguno de los mejores puntos faciales son los ojos, los agujeros de la nariz y las comisuras de los labios al ser zonas con muchos contrastes. Hay una gran cantidad de modelos distintos con distinta cantidad en número de puntos de referencia faciales. En el caso de este proyecto, se usa un modelo constituido por 68 puntos faciales: 10 para las cejas, 12 para los ojos, 9 para la nariz, 20 para la boca y 17 par el contorno de la cara como se puede mostrar en figura 1.5 [37].

1.5. Objetivos

Los objetivos de este trabajo marcarán la ruta o plan de trabajo que se considera a continuación de esta sección. El objetivo principal es el desarrollo de un conjunto de herramientas software destinadas a ofrecer un soporte al facultativo en la cirugía endonasal.

Se considera que este dispositivo ofrece ventajas sobre el mercado actual como un bajo coste, una configuración sencilla, evitar el contacto con el paciente y reducir el espacio necesario, manteniendo una velocidad de ejecución a tiempo real.

Para lograr esto se disminuye la precisión que ofrece la competencia, lo que ,dentro de unos límites, es prescindible ya que el objetivo es localizar la cámara de visualización por lo que no requiere una precisión milimétrica.

Los objetivos de este trabajo son los siguientes:

- Desarrollo de un programa para el guiado en cirugía endoscópica nasosinusal
- Estimar la orientación de la cámara respecto a la cara del paciente

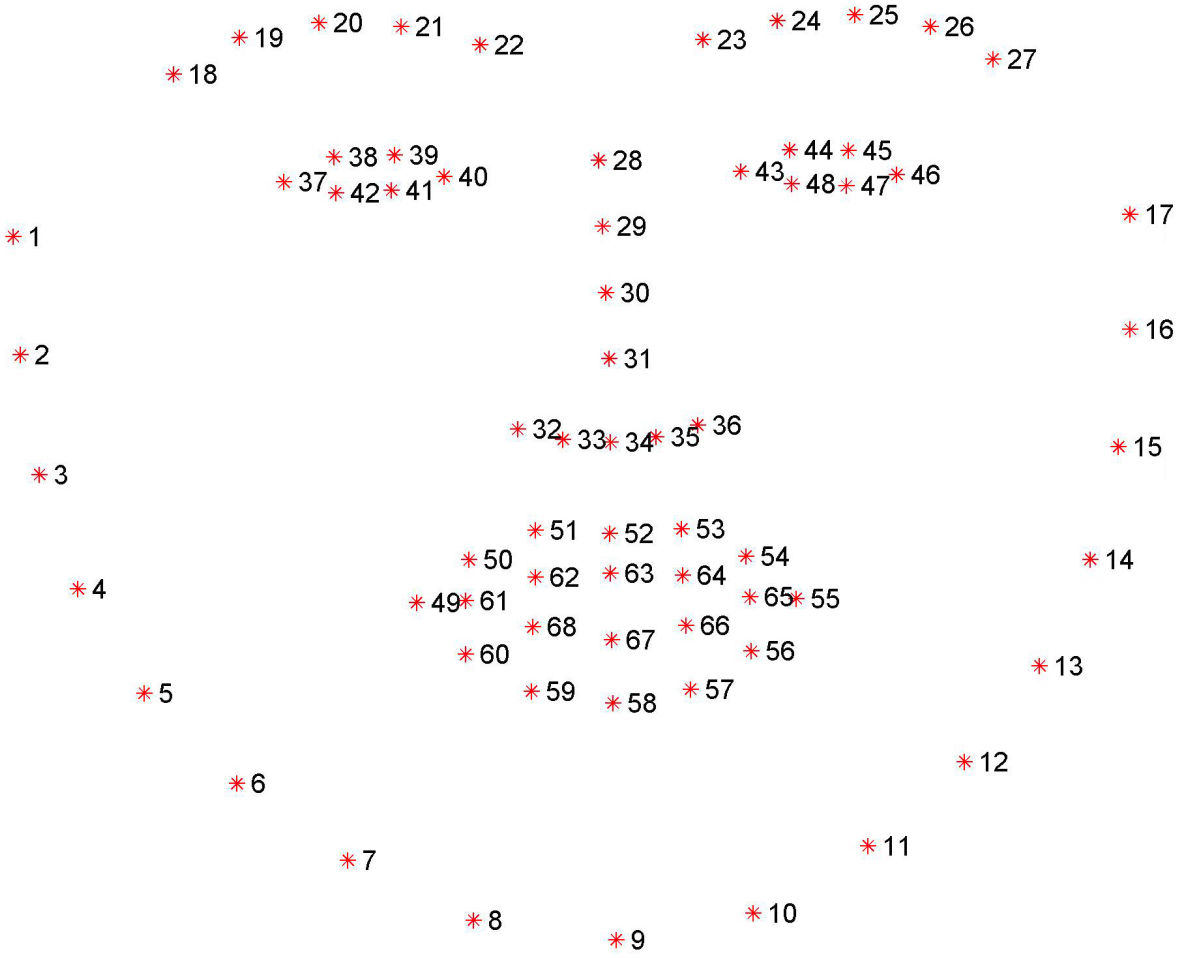

Figura 1.5: Modelo compuesto por 68 puntos de referencia faciales.

- Relacionar la orientación de la cámara con la ubicación de la punta del endoscopio
- Ejecución del sistema a tiempo real. El cotutor especifica que una imagen por segundo permite su usabilidad, pero el objetivo es llegar a las treinta imágenes por segundo.
- Permitir unos rangos de movimientos suficientes para el transcurso normal de la cirugía.
- Conseguir unos errores en orientación de la cámara y la punta del endoscopio dentro de lo aceptable para el proceso
    - Mantener el error en las rotaciones por debajo de los 5º por eje

    - Mantener el error en las traslaciones por debajo del centímetro por eje

- Crear una interfaz sencilla que permita la fácil configuración, ejecución y visualización de resultados

- Robustez y recuperación inmediata en caso de perdida de visión de la cara del paciente

1.6. Plan de trabajo

Para el desarrollo de este proyecto se seguirá un plan de trabajo dividido en las siguientes fases:

- Interactuar con la cámara para capturar y posteriormente filtrar y manipular imágenes durante el proceso quirúrgico. Este proceso se adaptará a las condiciones ofrecidas en un quirófano, tales como una fuerte iluminación directa y la aparición de objetos no deseados.

- Procesar la información detectada por la cámara localizando la cara del paciente y sus puntos de referencia faciales.

- Uso de algoritmos de reconstrucción tridimensional para obtener un modelo de la cara del paciente mediante sus puntos de referencia. Este proceso se realizará previamente a la cirugía.

- Programar un sistema capaz de relacionar la orientación del instrumento y la cara del paciente en el espacio tridimensional.

- Programar un proceso script capaz de usar todos los elementos previamente programados para obtener una detección de la orientación de la cámara lo más próxima posible a tiempo real.

- Diseño un sistema de representación visual del estado actual del instrumento respecto a la cara del paciente.

- Pruebas y depuración: por último, se hará un análisis exhaustivo del funcionamiento del sistema creado, sus bondades y carencias, y las posibles vías de mejora.

1.7. Estructura del documento

Este documento se estructura en 6 capítulos:

- Introducción. Capítulo actual. Abarca los conocimientos necesarios como base para la comprensión del proyecto. Realiza un estudio sobre el estado actual sobre el entorno en el que se sitúa el trabajo y establece los objetivos de este.
- Análisis. En este capítulo se exponen las diferente alternativas disponibles para resolver cada uno de los problemas que plantea el trabajo.
- Diseño. Se analiza el hardware disponible y se estructura el diseño software de la solución.
- Algoritmos. Se detallan los algoritmos usados a nivel teórico.
- Implementación. Se desarrolla el proceso seguido para la creación del sistema.
- Procedimiento. En este capítulo se expone el procedimiento quirúrgico y su interacción con el programa de este trabajo.
- Experimentos. Se realizan pruebas sobre el programa desarrollado para obtener datos necesarios de este y sus especificaciones.
- Conclusiones y trabajo futuro. Se analiza el proceso de desarrollo y los resultados del trabajo. Se destacan sus virtudes y carencias, concluyendo con las posibles lineas futuras a seguir.

Capítulo 2

Análisis

2.1. Introducción

Conseguir un sistema capaz de detectar la pose de un instrumento respecto al paciente, únicamente haciendo uso de una cámara es una tarea compleja. Por esto se presentan varias opciones que se deberán analizar para llegar a una propuesta concreta y bien definida.

Aunque se tratan temas que se pueden considerar modernos como la detección facial o la reconstrucción tridimensional, existen muchas técnicas desarrolladas para actuar en distintas situaciones. Este tipo de tareas son muy complejas y costosas computacionalmente, por lo que hasta hace poco no se precisaba de las tecnologías para ejecutar sus algoritmos.

Muchos estudios han centrado sus esfuerzos en mejorar la eficiencia de estos procesos y, junto con los nuevos avances en el sector tecnológico, han ubicado estas técnicas en una situación de gran utilidad e interés.

En este capítulo se analizan los principales procesos del sistema, analizando las posibilidades presentes y seleccionando las más indicadas para este trabajo.

2.2. Detección facial

Se entiende por detección facial, en visión por computador, por aquella técnica que mediante el análisis de imágenes permite detectar y localizar en esta un rostro humano. Este es uno de los primeros problemas que trataba de resolver la visión por computador, empezando su estudio hace unos 45 años [45].

Durante los últimos 20 años la detección facial automátizada ha despertado un gran interés, siendo usada para una gran cantidad de aplicaciones como la base para el reconocimiento facial, sistemas de vigilancia, analisis del comportamiento, reconocimiento de edad, etc. El desarrollo de internet y gracias al esfuerzo de la comunidad de desarrolladores, han ido surgiendo nuevos algoritmos más robustos y programas de evaluación.

En esta sección se analizan las distintas técnicas de detección, clasificadas en cuatro categorías según Yang et al. [44].

2.2.1. Métodos basados en el conocimiento

Son aquellos que aplican una serie de reglas para la detección de objetos [30]. Si el área analizada cumple cada una de estas normas se considerará que se ha encontrado un objeto del tipo deseado.

A este tipo de reglas se les suele denominar de sentido común, ya que descomponen el elemento a buscar en subpartes. Si estas partes son localizadas en una posicion concreta se considera que se ha encontrado el objeto. Aplicando a la detección de caras, una cara puede dividirse en un conjunto ordenado de elementos con unas formas específicas como son la nariz, los ojos, la boca, orejas, etc... A su vez, cada elemento puede ser dividido en otros.

El problema de esta técnica es la dificultad para seleccionar un rango dentro del cual una condición se cumpla, por lo que si las reglas son muy restrictivas se detectarán menos elementos de los esperados, y si son muy generales se obtendrán resultados que no sean

los deseados al detectar.

2.2.2. Características invariantes

Estos métodos se basan en la suposiciones de patrones generales que cumplen la mayoría de los objetos que se pretenden detectar. Por ejemplo, la consideración del color y la textura de un objeto son casos típicos de características invariantes.

El problema que presentan es la gran intolerancia a variaciones de iluminación o al ruido que modifica su textura. La necesidad de usar colores disminuye el rendimiento al ser más eficiente el uso de imágenes en escala de grises.

2.2.3. Template matching

Esta técnica consiste en encontrar coincidencias de una imagen representativa de un objeto (template) dentro de otra imagen [38]. Esta plantilla, que representa al objeto, se busca por toda la imagen, y a cada pixel se le asigna un valor según la similitud de la plantilla y su área colindante. Por lo tanto, el pixel que muestre una mayor coincidencia localiza el área en la cual se encuentra con más probabilidad el objeto.

Esta técnica presenta problemas como la invarianza a la escala, rotación o tamaño. Pequeñas variaciones tanto en la imagen como en la plantilla ofrecen resultados muy diferentes. Se pueden usar varias plantillas pero pueden diferir en tamaño. Generalmente representar un objeto complejo por plantillas es una tarea difícil de realizar. Además, el coste computacional de buscar cada plantilla sobre cada región es muy grande.

2.2.4. Métodos basados en la apariencia

Los métodos basados en la apariencia se basan en la definición de la forma de un objeto y sus componentes para detectarlo [24]. Son parecidos a los métodos basados en el conocimiento pero en este caso las reglas se aplican a la silueta de estos, determinando los limitados puntos de vista desde los cuales se puede observar el objeto. Obteniendo la

silueta de un objeto en concreto desde sus puntos de vista más comunes, se realiza una búsqueda de coincidencias dentro de otra imagen.

2.3. Reconstrucción facial tridimensional

La reconstrucción facial en el ámbito de la visión por computador consiste en la composición de un modelo tridimensional de una cara a partir de imágenes bidimensionales. Este modelo es necesario para orientar un sistema de coordenadas tridimensional sobre la cara del paciente. La reconstrucción será personalizada para cada paciente, evitando generalizaciones y alcanzando la máxima fidelidad en el modelo.

El objetivo de esta reconstrucción es la recuperación de la profundidad del objeto de imágenes planas que solo pueden ubicar el objeto en un espacio bidimensional. Para esto se usan varias imágenes, varias cámaras o distintos tipos de imágenes. En esta sección se presentan las principales técnicas para la reconstrucción facial.

2.3.1. Estructura por sombras

La estructura por sombras o en inglés "*shape-from-shading*" se basa en la sombra que generan las distintas estructuras de la cara [11]. Modelando la iluminación que afecta al rostro en cada imagen obtienen la profundidad de la cara según la sombra o brillos generados en la cara. Están basados en técnicas estadísticas de manera que tras un entrenamiento previo, por ejemplo con redes neuronales, o con un modelo genérico del rostro humano, permiten obtener una reconstrucción facial de una imagen. Son muy sensibles al posicionamiento e intensidad de la iluminación.

2.3.2. Mapas de profundidad

Los mapas de profundidad o en inglés "*depth maps*" usan dos o más cámaras posicionadas de manera que un mapa de sus diferencias permita una visión tridimensional [16]. Este método busca una funcionalidad similar a los ojos humanos. Con estas diferencias entre imágenes se pueden estimar las distintas profundidades de la cara en unos pocos

segundos.

2.3.3. Estructura por movimiento

La estructura por movimiento o en inglés "*structure from motion*" permite obtener un modelo tridimensional a partir de imágenes bidimensionales tomadas desde distintos ángulos y posiciones [39]. Por cada imagen se crea una nube de puntos que se relaciona con sus imágenes posteriores consecutivamente, creando un mapa de variación entre puntos. A partir de la tercera imagen, esta técnica prueba distintas orientaciones de las imágenes hasta que la estructura de la nube de puntos de todas las imágenes previas se vuelve consistente.

Esta técnica permite obtener la profundidad a partir de una sola cámara. La técnica no restringe la orientación con la que tomar las imágenes, ni siquiera el orden. Aunque no restrinja, esta técnica es muy sensible a la orientación ya que, dependiendo desde donde se tome, dará unos resultados potencialmente distintos a otros.

La nube de puntos presenta problemática, ya que las distintas técnicas para detectar estos puntos pueden devolver puntos muy distintos incluso en imágenes consecutivas o tomar parte del fondo. Para esto se localizan los puntos de referencia faciales más significativos de la cara, pasándole la localización de estos por cada imagen detectada. De esta forma nos aseguraremos que solo se tomen puntos significativos para el proceso y estables respecto a la orientación.

Capítulo 3

Diseño de la solución

3.1. Introducción

En este capítulo se exponen las decisiones tomadas para desarrollar el sistema propuesto. Para esto en una primera sección se analizarán las herramientas usadas para la creación del dispositivo, tanto hardware como software. En la segunda sección se presenta la estructuración del software desarrollado y sus relaciones.

3.2. Análisis de las herramientas

En esta sección se presentan las principales herramientas que se encuentran actualmente para el desarrollo del trabajo. Tras el análisis de sus prestaciones se procederá a la selección de aquellas que se adapten mas a las necesidades del proyecto. Dependiendo del tipo de herramienta a usar, esta sección se divide en herramientas hardware y software.

3.2.1. Herramientas hardware

El conjunto de herramientas hardware esta compuesto por todos aquellos elementos físicos sobre el que se sustenta el software. Normalmente se relaciona con la parte tangible de una computadora o un sistema informático, comprendiendo sus elementos eléctricos, electrónicos, electromecánicos y mecánicos. Para este trabajo, las herramientas hardware a analizar son la cámara con la que se obtendran las imágenes del paciente, y el sistema

	Pi Module v2	**SJCAM M10+**	**Logitech C160**
Resolución sensor (MP)	8	12	0.3
Resolución max vídeo (px)	1920 x 1080	2560 x 1440	640 x 480
FOV (º)	49 - 62	170 (regulable)	50
FPS max.	90	120	15
Batería	No	900 mAh	No
Conexión	Cable cinta	Wifi, cable	Cable, USB 2.0
Dimensiones (cm)	2.5 x 2.4	5 x 4 x 4	9 x 6 x 4
Precio actual (€)	20	65	10

Tabla 3.1: Especificaciones técnicas de las cámaras.

para el procesado de la información.

Cámara

Cuando se habla de cámara, generalmente se refiere a las cámaras tipo *pinhole* que mapean el mundo tridimensional a partir de los rayos de luz que infieren en su sensor, conocido como el centro de la cámara. Este proyecto esta enfocado en crear un dispositivo de bajo coste, por lo tanto las cámaras analizadas cumpliran con el requisito de ser baratas y accesibles a todo tipo de publico. Las cámaras que se analizarán son las siguientes:

- Raspberry pi Camera Module v2. La segunda versión del módulo de la famosa placa computadora Raspberry Pi.

- SJCAM M10+. Cámara de acción de la compañia SJCAM, de diseño cúbico y con grabación hasta resolución 2K.

- Webcam Logitech C160.

Las especificaciones técnicas de estos dispositivos se pueden observar en la tabla 3.1.

Una vez observadas sus características se ha optado por la cámara SJCAM M10+ para el desarrollo del trabajo por las siguientes conclusiones:

- Pude ser usada tanto inalambricamente y con función cámara web al conectarla al ordenador.
- La capacidad de regular el ángulo de visión permite seleccionar aquel en el que se visualice la cara del paciente, independientemente del tamaño del endoscopio.
- Aun siendo más cara que las otras opciones, sigue teniendo un precio muy competitivo para sus especificaciones.
- Su tamaño es relativamente pequeño para tener autonomía.
- Su resolución y su velocidad de imágenes por segundo es superior a las alternativas.

Sistema de procesamiento

El software debe ejecutarse en alguna plataforma que permita el análisis de los datos introducidos y obtener unos resultados que sean acordes a los esperados. Este sistema debe cumplir los requisitos de potencia computacional y compatibilidad con las librerías software requeridas. Para este trabajo se consideraran los siguientes sistemas:

- Lenovo ideapad Z510. Portatil de gama media-alta de la marca Lenovo de mediados de 2014.
- Raspberry Pi 3 Model B. Computador de placa reducida famosa por su tamaño compacto y su reducido precio.

Las especificaciones técnicas de estos dispositivos se pueden observar en la tabla 3.2.

Viendo sus características se observa que aunque la Raspberry Pi tiene una mejor relación calidad/precio, esta lejos de tener la potencia de procesado del portatil, además de ofrecer este último un sistema completo sin tener que añadir ningún periférico adicional. Por lo tanto el sistema usado para desarrollar este trabajo será el portátil Lenovo, mientras que en trabajos futuros se propondrá optimizar el sistema para la placa Raspberry.

	Lenovo ideapad Z510	**Raspberry Pi 3 mod B**
Vel. Procesador (GHz)	2.2	1.2
RAM (GB)	8	1
Memoria (TB)	1	No integrada (SD)
Pantalla	15.6”	No integrada (DSI)
Perifericos entrada	Teclado y touchpad	No integrada (USB)
Cámara y micrófono	Si	No integrados (CSI, USB)
Puertos entrada	USB 3.0, Lector tarjetas	USB 3.0, 40 GPIO pins
Conectividad	Wifi, LAN, Bluetooth	Wifi, LAN, Bluetooth
Dimensiones (cm)	38 x 26 x 2.5	8.7 x 5.8 x 0.5
Precio actual (€)	750	30

Tabla 3.2: Características técnicas de los sistemas de procesamiento.

3.2.2. Herramientas software

En este capítulo se muestran las principales herramientas software usadas en este proyecto. Algunas librerías requieren de otras en las que se basan, estas últimas no se mencionaran, aportando las referencias para que el lector pueda ampliar la información aquí recogida.

Ubuntu 16.04

Ubuntu es una distribución o distro del sistema operativo GNU/Linux. Se distribuye como software libre y abierto siendo una de las distros mas usadas de con un 49 % de cuota en el mercado de sistemas operativos. Aunque ofrece software para servidores y para sistemas en la nube, es reconocido por su sistema para el usuario medio ofreciendo una gran facilidad de uso y diseño agradable[9].

Python 3.6

Python es un lenguaje de programación interpretado, centrado en una sintaxis simple y una gran velocidad de desarrollo. Aunque normalmente es usado como un lenguaje de programación orientado a objetos, soporta múltiples paradigmas. Posee una licencia de

código abierto compatible con GNU. Actualmente se encuentran dos versiones incompatibles entre si, la 2.x y la 3.x. Python es multiplataforma, pudiendo ejecutarse en Linux, Mac OS o Windows[8].

OpenCV

OpenCV es una famosa librería de visión por computador desarrollada en sus inicios por intel. Su licencia BSD que le permite el libre uso para comercialización y desarrollo la han convertido en una de las librerías mas usadas desde su liberación en 1999. Está estructurada por módulos, de manera que se pueden usar las características básicas o ir añadiendo funcionalidades según se requiera. Esta desarrollada en C/C++, pero permite el desarrollo en algunos lenguajes como Java o Python. OpenCV es multiplataforma[7].

Dlib

Dlib es una librería compuesta por un conjunto de herramientas destinadas la creación de software de machine learning y otros sectores. Es una librería de código abierto y permite su uso a través de una licencia de tipo *Boost Software License*. Sus diversos componentes cubren áreas como el álgebra, hilos, interfaces gráficas, estructura de datos, *machine learning*, procesado de imágenes, minado de datos, tratamiento de XML, optimización numérica, redes bayesianas y muchos más. Entre sus principales características destaca la gran cantidad de documentación que presentan sus módulos y la gran portabilidad de su código[4].

NumPy

NumPy es una librería de Python que incorpora la capacidad de trabajo con arrays y matrices de datos muy optimizado. NumPy es de código abierto y aporta funciones matemáticas de alto nivel[6].

Matplotlib

Matplotlib es una librería de Python que permite la representación de datos en arrays de NumPy en gráficas 2D o 3D con gran facilidad. En palabras de su creador John Hunter

"*matplotlib tries to make easy things easy and hard things possible*". Es de código abierto y multiplataforma[5].

3.3. Arquitectura software

En esta sección se muestra la estructura que sigue la implementación, diferenciada en módulos funcionales. Cada uno de estos módulos están compuestos por una o más clases con funciones relacionadas.

3.3.1. Estructuración

El programa se divide en nueve módulos. De estos, se desarrollaran aquellos que sean considerados principales, ya que el resto son módulos de soporte que permiten realizar funciones secundarias o sirven como interfaz de las librerías usadas. Los módulos suplementarios son los siguientes:

- Classifier. Este módulo funciona como una interfaz entre los métodos de clasificación de OpenCV y los demás módulos.
- Utils. Módulo compuesto por métodos usados ampliamente por todos los módulos, con propósito general.
- Plot. Módulo que hace uso de la librería Matplotlib para representar los gráficos y resultados.
- Video. Este módulo contiene clases para la manipulación de vídeos. Contiene a la clase Camera que recupera el flujo de imágenes capturado por la cámara.

Módulo de imagen

Este módulo contiene las clases encargadas de recuperar, almacenar y manipular imágenes. Es dependiente de las librerías propias de Python y de OpenCV, además del módulo Utils. Sus clases se encargan de la carga, modificación, filtrado y almacenado de las imágenes.

Las dos clases que componen el módulo son las siguientes:

- Frame. Clase encargada de contener la información de la imagen y ofrece métodos para la recuperación y manipulación de características propias de esta. Permite la carga y guardado de múltiples imágenes, su representación, el filtrado y la extracción de características.
- Drawer. Esta clase se encarga de pintar distintos tipos de formas en una imagen. Permite representar puntos y rectángulos de una forma rápida y fácil al resto de módulos.

Módulo de detección

Este módulo contiene clases encargadas de detectar los elementos característicos de la cara. Ofrece el máximo nivel de abstracción respecto a la detección de AOI en el rostro del paciente. Es decir, adapta los parámetros de búsqueda para cada tipo de elemento a detectar, eliminando la necesidad de determinarlos a tiempo real minimizando la supervisión en el proceso de detección.

Este módulo es únicamente dependiente del módulo Classifier, del que obtendrá la funcionalidad de clasificación, proporcionando una serie de parámetros predefinidos para cada ocasión.

Las clases que componen este módulo son las siguientes:

- Detector. Es la encargada de detectar un objeto en particular en una imagen de la clase Frame dado un conjunto de datos entrenados para ese objeto. Por defecto detecta caras, y al poder devolver varios resultados, devolverá el mejor resultado localizado.
- EyesDetector, NoseDetector y MouthDetector. Extienden a la clase Detector. Estas clases comparten funcionalidad implementando detectores específicos para las distintas partes de la cara. Hacen uso de secciones de la cara para restringir la zona de búsqueda de cada parte.

- FrontalFaceDetector. Se encarga de coordinar tanto al detector de caras como a los detectores de sus secciones. Devuelve que elementos han sido detectados y donde. Ofrece una evaluación sobre la cara detectadas, siendo proporcional a la cantidad de subelementos correctamente localizados.

Módulo de puntos faciales

La finalidad de este módulo es la localización de los puntos de referencia faciales. Dada una imagen, representada como un objeto de la clase Frame, se obtienen los 68 puntos de referencia faciales. Aunque opcional, el paso de la localización de la cara será el caso normal de uso, permitiendo una localización más eficiente y efectiva.

Este módulo está conformado únicamente por una clase dependiente de la librería Dlib para la implementación de los algoritmos de machine learning que detecten los puntos de referencia. La clase es la siguiente:

- Landmarker. Esta es la clase principal del módulo y su objetivo es el de devolver una lista de puntos ordenados, correspondientes a los puntos de referencia faciales de una imagen dada. La clase admite tanto la posibilidad de una sola cara o de muchas en una misma imagen.

Módulo de reconstrucción

Este módulo se encarga de realizar el proceso de reconstrucción de la cara del paciente. El módulo genera esta reconstrucción mediante los puntos de referencia faciales detectados anteriormente.

Las clases que constituyen el módulo son las siguientes:

- Reconstructor. Clase principal del módulo. Toma como entrada los puntos detectados y se encarga de formatearlos correctamente para la entrada del algoritmo de reconstrucción. Devuelve el modelo tridimensional reconstruido.

- ReconstructionHelper. Esta clase agiliza el procedimiento de la reconstrucción al ofrecer distintas maneras de obtener el modelo tridimensional.

Módulo de pose

El módulo de pose se encarga del posicionamiento de la cámara respecto a la cara del paciente. Para esto requiere un modelo tridimensional formado por un conjunto de puntos tridimensionales, y los puntos faciales detectados en el instante que se quiera localizar la cámara. Este módulo permite la aplicación de transformaciones al conjunto de puntos, su alineación y la estimación de la orientación de la cámara en un instante dado.

Este módulo está compuesto por las siguientes clases:

- Positioner. Esta clase permite la obtención de la pose de la cámara a partir del modelo tridimensional de puntos generados del paciente y las imágenes de una cámara calibrada. Devuelve la orientación de la cámara.
- Pose. Clase principal del módulo. Esta clase se encarga de orquestar la funcionalidad de detección de la pose. Filtra y modifica los datos de la estimación de la pose para mejorar el resultado.
- Aligner. Esta clase se encarga de localizar los puntos tridimensionales de una forma adecuada para el procedimiento quirúrgico.
- Transformer. Esta clase funciona como soporte ofreciendo funciones que abarcan tareas como transformaciones, productos vectoriales y geometría vectorial.

Capítulo 4

Algoritmos

4.1. Introducción

En este capítulo se exponen todos los algoritmos usados en este trabajo. Tras el análisis de las diferentes técnicas disponibles, se explica teóricamente cada una de las alternativas seleccionadas. El nivel de profundidad teórico que se alcance varía según la importancia de la técnica en el trabajo, dejando al lector las referencias necesarias en caso de querer profundizar en un tema concreto.

4.2. Detección de caras

El algoritmo de detección de caras usado en este trabajo está basado en el algoritmo de detección de objetos desarrollado por Paul VIola and Michael Jones en 2001 [40]. Este algoritmo, llamado sistema de detección de objetos VIola-Jones, puede usarse para detectar una gran cantidad de objetos, aunque es famoso por la detección de caras.

El funcionamiento del algoritmo requiere de un entrenamiento. Para este entrenamiento es necesario un conjunto de imágenes representativas del tipo de objeto que se quiere detectar en todas sus variantes, poses e iluminaciones posibles. Además, para el entrenamiento será necesario un conjunto de imágenes negativas, que no representan de ninguna manera el objeto que se quiere detectar.

Esta técnica hace uso de unos descriptores llamados características de Haar que permiten encontrar patrones en imágenes basándose en los contrastes de esta. Para obtenerlos se usará un algoritmo de *machine learning* llamado Adaboost que unido al uso de cascadas de características permitirá encontrar un objeto en la imagen.

4.2.1. Características de Haar

Una vez obtenidas las imágenes, hay que extraer sus características. En este caso el algoritmo utiliza las llamadas características de Haar que son la que muestra la figura 4.1

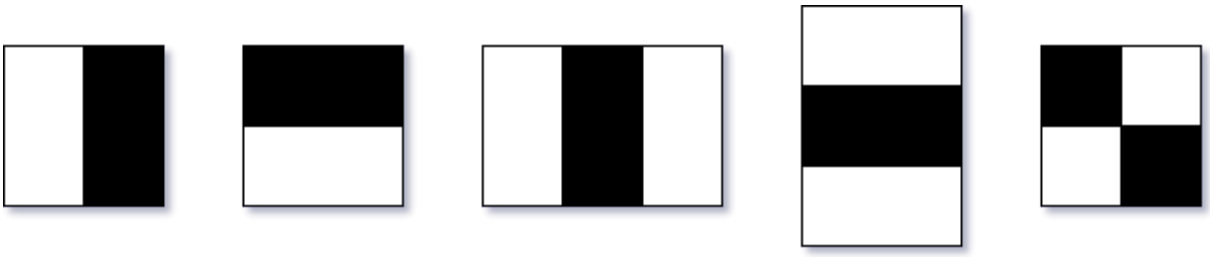

Figura 4.1: Conjunto básico de características de Haar

Estos son llamados los filtros de Haar básicos. Son una serie de rectángulos divididos de dos a cuatro secciones, en los que dos de ellos representan direcciones horizontales, otros dos direcciones verticales y uno direcciones diagonales. Funcionan como filtros siendo la parte negra aquella que nos ofrecerá una contribución positiva, y las de blanco una contribución negativa. El resultado de aplicar estos filtros sobre la imagen nos ofrece el gradiente de contraste en la imagen según la dirección propuesta por cada filtro.

Para formular la aplicación de estos filtros en la imagen, tomaremos como B aquellos conjuntos de pares de coordenadas situadas en la zona negra del rectángulo, y W aquellos encerrados por una zona blanca. Una vez obtenido el valor, el número resultante habrá que normalizar según el tamaño de la ventana que tenga la característica actual. Siendo I la matriz de intensidad de la imagen, el funcionamiento de este filtro se observa en la ecuación 4.1

$$V = \sum_{x,y \in B} I(x,y) \; - \sum_{x,y \in W} I(x,y) \tag{4.1}$$

En vez de hacer una pirámide de imágenes a distintas escalas, el escalado en este caso se realiza sobre las características de Haar, produciendo elementos de distintos tamaños tanto horizontal como verticalmente. El número de operaciones necesarias para aplicar todos estos tipos de filtros a la imagen es muy elevado. Por ejemplo, a una imagen de 24 x 24 píxeles únicamente, se le realizarán 162336 características distintas, cada una de ellas aplicadas sobre toda las posibilidades en la imagen. Además, se suele aplicar un conjunto de 14 características de Haar, aumentando en gran número la dimensionalidad del problema.

4.2.2. Imagen integral

Para resolver el problema de la cantidad de operaciones que hay que realizar, se introduce el concepto de imagen integral. La imagen integral de una imagen sobre un píxel, consiste en la suma de la intensidades de todos los píxeles superiores y a la izquierda de este, como se aprecia en la figura 4.2 (izquierda).

Este concepto de imagen integral es de gran utilidad en la aplicación de las características de Haar, al estar estos basados en el nivel de intensidad de rectángulos determinados en la imagen. La obtención de la imagen integral es un proceso muy eficiente que es calculable en un recorrido de la matriz de intensidad. La ecuación 4.2 representa la obtención de la imagen integral *II*, donde s, de ecuación 4.3, es la acumulación actual de intensidades en la fila de ese píxel, e I es la matriz de intensidad.

$$II(x,\, y) = II(x,\, y-1) + s(x,\, y) \tag{4.2}$$

$$s(x,\, y) = s(x-1,\, y) + I(x,\, y) \tag{4.3}$$

Este concepto de imagen integral se aplicará a las características de Haar, con una fórmula específica para cada rectángulo de esta característica (blanco o negro), dependiendo de la zona. Tomemos e1, e2, e3 y e3 como las esquinas superior izquierda, superior

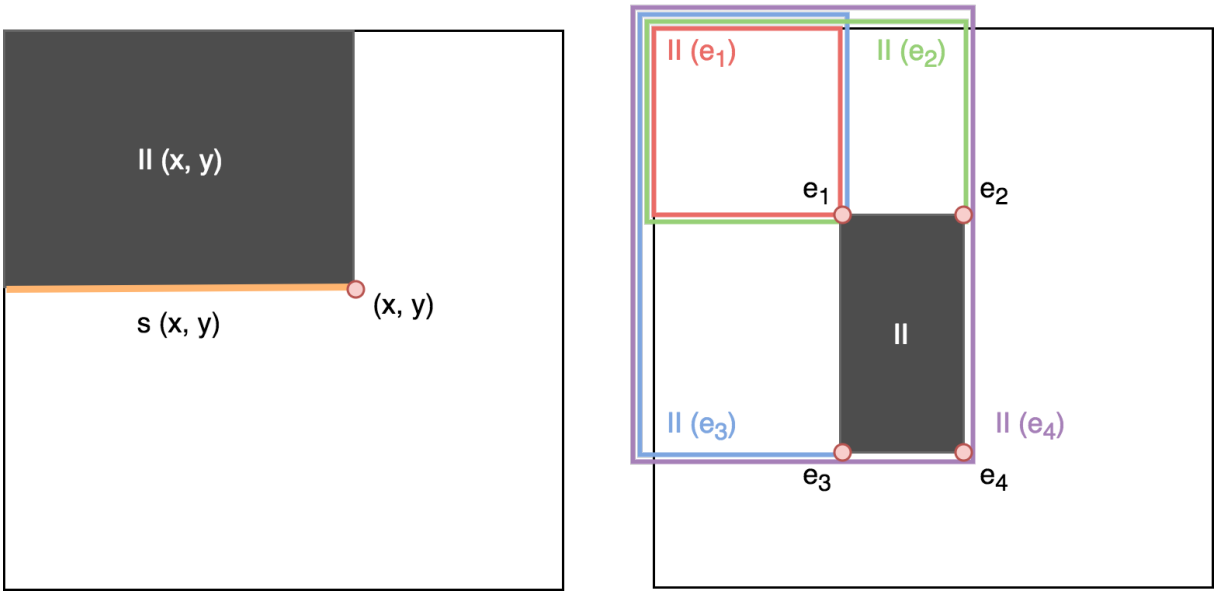

Figura 4.2: Imagen integral. Imagen integral aplicada a una sección de un característica de Haar.

derecha, inferior izquierda e inferior derecha respectivamente, tal como se muestra en la figura 4.2 (derecha). La fórmula para obtener la imagen integral del rectángulo será la siguiente:

$$II(e_1,\, e_2,\, e_3,\, e_4) = II(e_4) - II(e_3) - II(e_2) - II(e_1) \tag{4.4}$$

4.2.3. Clasificadores

Para resolver el problema de la cantidad de características distintas de Haar, se propone el uso de Adaboost [41]. El funcionamiento principal de Adaboost es la búsqueda de muchos clasificadores que funcionen ligeramente mejor que uno aleatorio, llamados clasificadores débiles. Una vez encontrados estos clasificadores, se unifican en un clasificador fuerte, ponderando a cada uno de los débiles según su desempeño. Esta comprobado cómo este método converge en clasificadores robustos.

Clasificador débil

Para obtener los clasificadores simples se obtiene un número de muestras ejemplo, que contengan elementos que se deban reconocer como positivos y otros como negativos. Consideremos la aplicación de una característica de Haar x a una escala y en una posición fija en cada una de las imágenes de ejemplo. La aplicación de la característica se modela como una función f(x) que identifica el valor del contraste resultado de ese filtro de Haar.

Según el resultado de la función f(x) aplicada a cada una de las muestras, se introduce el umbral Θ que las separa en positivas y negativas. Este umbral es el encargado de clasificar las imágenes, por lo tanto el clasificador débil debe asegurar que su resultado es mejor que una clasificación aleatoria. El proceso simplificado es el que aparece en la figura 4.3.

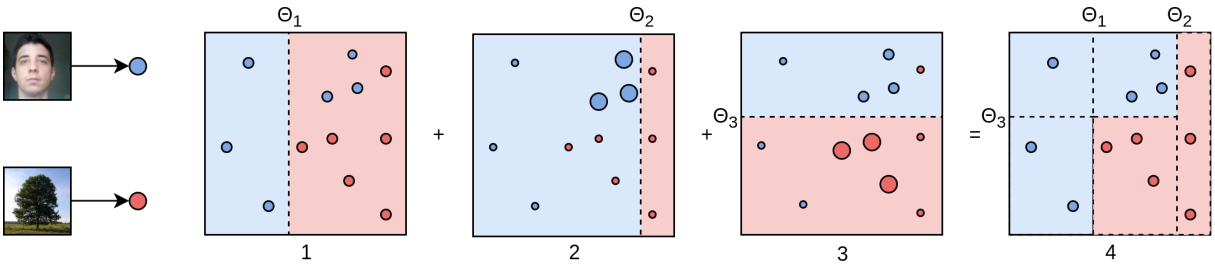

Figura 4.3: Proceso simplificado para la obtención de un clasificador débil.

Siendo las muestras azules las positivas (caras) y las rojas negativas (no caras), el umbral Θ se sitúa de manera aleatoria en su estado inicial (1). El resultado de este primer paso es la clasificación correcta e incorrecta de manera aleatoria de las características. Las muestras mal clasificadas aumentarán su ponderación, de manera que el umbral intentará minimizar el error en la clasificación ajustándose en mayor medida a estas (2). El proceso se repite (3) hasta obtener una serie de clasificadores que juntos compondrán el clasificador débil (4). Las fronteras de decisión de cada clasificador separan el conjunto en varias secciones, y cada sección tendrá un valor positivo o negativo según la mayor cantidad de muestras que se hayan conseguido en cada umbral.

Al crear un clasificador a partir de un umbral, este dotará de valores positivos o negativos según el tipo de muestra. Por lo tanto, puede darse el caso de que nuestro clasificador sea un buen detector de no caras de las muestras ejemplo. Esto no representa ningún problema si se quisiera clasificar elementos dentro de las muestras ejemplo. Pero en la práctica hay muchos objetos no cara que no se han tenido en cuenta, pudiendo obtener un resultado no controlado. Por lo tanto se introduce una variable binaria α que controlará que los umbrales asignen correctamente los elementos de cara como positivos y los elementos no cara como negativos, como se observa en la ecuación 4.5

$$h(x) = \begin{cases} \alpha & f(x) < \theta \\ \alpha & f(x) \geq \theta \end{cases} con\, \alpha \in \{-1,\, +1\} \tag{4.5}$$

Para obtener el parámetro del umbral Θ, se van probando distintos valores de este, obteniendo elementos correctamente clasificados y otros incorrectamente. Inicialmente todos los elementos tienen una ponderación idéntica, pero al ir probando distintos valores del umbral, aquellos elementos que sean incorrectamente clasificados verán aumentada esta ponderación, y aquellos que sean bien clasificados, disminuida. Con esto se consigue aumentar la necesidad de una buena clasificación en aquellos elementos que hayan sido clasificados incorrectamente en mayor medida.

En cada iteración en la búsqueda de un clasificador débil, se ordenan los elementos en orden descendente según el valor que les han asignado la función f(x) y se localiza el umbral que minimice el error, es decir, que la suma de las ponderaciones que están mal clasificadas sea mínimo. Una vez obtenido un clasificador débil para cada característica de Haar se selecciona aquella clasificación con un error más pequeño, eliminando el resto de características para ese caso.

Clasificador fuerte

La actualización de los pesos para la siguiente iteración w(t+1) dependen del peso de la iteración anterior w(t) y de un factor de actualización u(e_t) como se aprecia en

la ecuación 4.6. Este factor dependerá del error del clasificador, que al ser al menos un poco mejor que uno aleatorio, se deduce que $e_t < 0.5$. Por lo tanto, siendo y_i el resultado esperado, el factor de actualización será mayor que 1 si las muestras han estado mal clasificadas, aumentando su ponderación, y un número menor que 1 a aquellas bien clasificadas, disminuyendo su ponderación como se aprecia en la ecuación 4.7.

$$w_i(t+1) = \frac{1}{2}\, u(e_t)\, w_i(t) \tag{4.6}$$

$$u(e_t) = \begin{cases} \frac{1}{e_t} & h(x_i) \neq y_i \\ \frac{1}{1-e_t} & h(x_i) = y_i \end{cases} \tag{4.7}$$

El parámetro de iteraciones es fijado previamente T. El clasificador fuerte es resultado de la suma del clasificador débil obtenido en cada iteración multiplicado por un factor de ponderación λ_i de ecuación 4.9, que dependerá del error de este clasificador. Este factor tenderá a infinito cuando el error tienda a 0, y tenderá a 0 cuando el error tienda a 0.5, que sería equivalente a un clasificador aleatorio. Según el signo de esta suma, el resultado será clasificado como un ejemplo positivo o negativo, como se aprecia en la ecuación 4.8

$$H(x) = sign\left(\sum_{i=1}^{T} \lambda_i\, h_i(x)\right) \tag{4.8}$$

$$\lambda_i = \frac{1}{2}\, \log\left(\frac{1 - e_t}{e_t}\right) \tag{4.9}$$

4.2.4. Cascada de clasificadores

La cascada de clasificadores concatena una serie de clasificadores que, en un orden determinado, identifican una cara en cada subventana de la imagen. Normalmente los clasificadores con un alto coste computacional solo se ejecutan al final de la cascada si aquellos con bajo coste identifican una cara. Por lo tanto únicamente habrá una cara si el último clasificador da un resultado positivo, en caso de que alguno diera un resultado negativo, esa zona será descartada como se puede observar en la figura 4.4

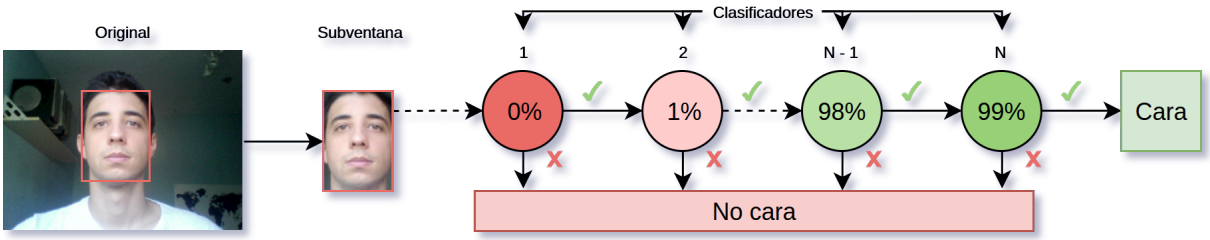

Figura 4.4: Cascada de clasificadores aplicada sobre una subventana

4.3. Puntos de referencia faciales

El algoritmo de detección de puntos de referencia faciales está basado en el descriptor de características HOG (Histogram Oriented Gradient), un clasificador lineal y una ventana deslizante que recorre la imagen. En este capítulo se centrará especial atención en el descriptor, añadiendo algunos detalles básicos del clasificador lineal y la ventana deslizante.

4.3.1. HOG

El descriptor de características HOG esta basado en la variación de contrastes que rodean a un píxel en particular (x, y) en una imagen I [17]. Este descriptor puede usarse tanto en imágenes en escalas de grises o RGB, pero únicamente se va a desarrollar en imágenes de escala de gris en este proyecto. El cálculo de la variación de contraste se realiza tanto en vertical como en horizontal según la ecuación 4.10. Se obtiene como resultado el resaltado de los bordes de la imagen.

$$\begin{aligned} g_x &= I(x+1, y) - I(x-1, y) \\ g_y &= I(x, y+1) - I(x, y-1) \end{aligned} \tag{4.10}$$

Una vez obtenido los gradientes en ambas direcciones para cada píxel, g_x para el gradiente vertical y g_y para el horizontal, se obtiene el ángulo Θ en la ecuación 4.11, y magnitud λ en la ecuación 4.12.

$$\theta = \arctan\left(\frac{g_y}{g_x}\right) \tag{4.11}$$

$$\lambda = \sqrt{g_x^2 + g_y^2} \tag{4.12}$$

Obtenidos los valores de ángulo y magnitud para cada píxel se crean dos matrices con el mismo tamaño de la imagen original con estos dos valores. Esta representación a nivel de píxel tiene que ser adaptada para el uso en un clasificador, ya que este requiere una entrada con dimensión fija que represente globalmente al elemento. Además, esta representación de píxel es muy sensible a variaciones en la forma como en la localización del objeto en la imagen. Se usará un vector de características para representar la imagen que capture la forma global del objeto de interés.

Para resolver el problema de la adaptación a un vector de características, HOG divide la imagen en celdas de un tamaño fijo, calculando en cada una de ellas el histograma, reagrupando finalmente estos para crear el vector de características. Con estos histogramas se obtiene la información de las orientaciones dominantes en la imagen y la distribución espacial de estos. Este proceso se resume en la figura 4.5.

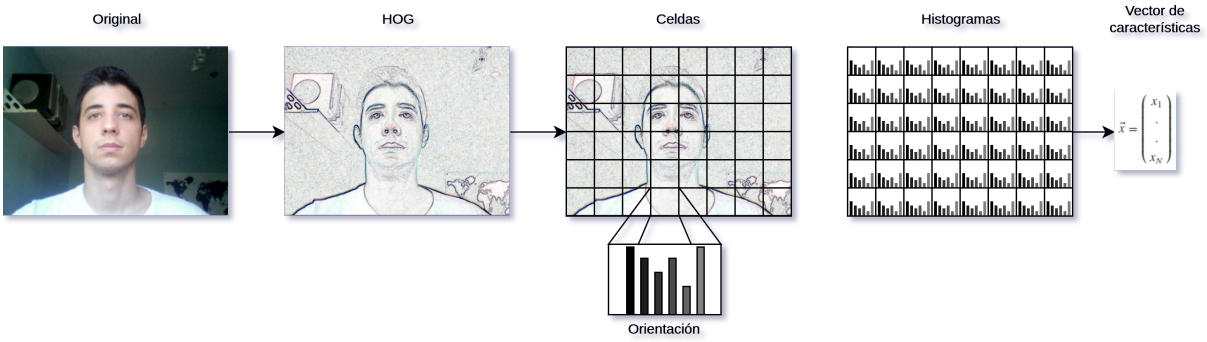

Figura 4.5: Vector de características a partir de los histogramas de cada celda

El primer paso es la creación de las celdas. Normalmente se usan celdas cuadradas, con un tamaño de lado de 6 a 8 píxeles. En el histograma, cada uno de los distintos valores vienen dados por un rango del ángulo propio de cada píxel, pudiendo estos ir de 0º a

360º contando con el signo, o de 0º a 180º si se elimina el signo, siendo esta última la más común. Cada valor del histograma comprende un rango de ángulos. Por ejemplo, sin contar el signo, los 180º se dividen entre 9 rangos que tendrá el histograma, por lo rango le corresponden 20º, al primero de 0º a 20º, al segundo de 20º a 40º, hasta el noveno 160º a 180º. Al eliminar el signo, gradientes con misma dirección y sentido inversos se consideran equivalentes, situándose en el mismo rango del histograma. Este proceso se puede observar en la figura 4.6

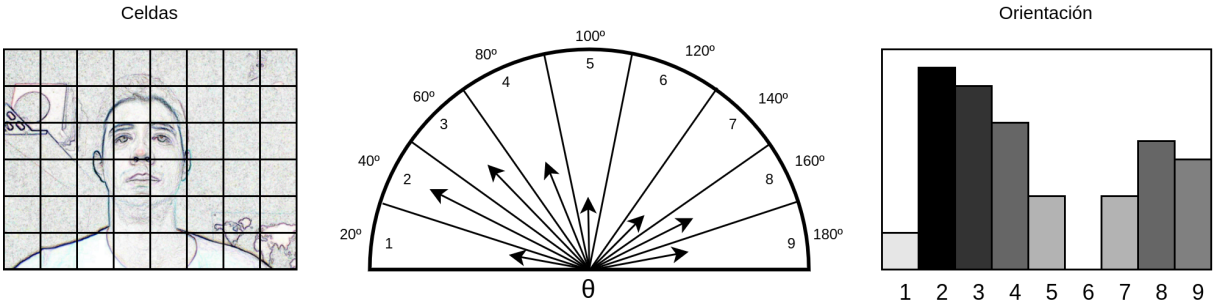

Figura 4.6: Creación del histograma a partir de los ángulos Θ de las direcciones de contraste en cada píxel.

La representación matemática de este último proceso, considera C la celda actual, k como cada uno de los rangos en los que se divide el histograma, $\delta\Theta$ el número de ángulos que le corresponde a cada valor del histograma, g el gradiente de cada píxel y h a la magnitud asociada a cada valor, de la ecuación 4.13. Además, un factor w_k controla que el gradiente actual corresponda al valor del histograma que se está calculando, como se puede observar en la ecuación 4.14

$$h(k) = \sum_{(x,\,y)\,\in\,C} w_k(x,\,y)\,g(x,\,y) \tag{4.13}$$

$$w_k(x,\,y) = \begin{cases} 1 & (k-1)\,\delta\theta \leq \theta(x,\,y) \leq k\,\delta\theta \\ b & en\,caso\,contrario \end{cases} \tag{4.14}$$

Un problema que se da al clasificar de esta manera los gradientes, es que dos de ellos con ángulos muy similares pueden acabar en distintos rangos del histograma si están

separados justamente por la frontera de estos. En este caso se asignan ambos gradientes a los dos rangos aplicando un factor a cada uno que depende de la distancia de cada gradiente al centro del rango del histograma. El factor garantiza que si la distancia del rango al centro es mayor que Θ, el factor sea cero. El factor w_k se actualiza según la ecuación 4.15.

$$w_k = \text{máx}\left(0,\, 1 - \frac{\theta(x,\, y) - \theta_k}{\delta\theta}\right) \tag{4.15}$$

Un problema similar se produce en las celdas, ya que dos puntos muy cercanos entre si pueden quedar separados en distintas celdas, siendo el resultado sensible a pequeñas variaciones de la forma del objeto. Se resuelve usando la misma estrategia, cada píxel se asigna a las cuatro celdas más cercanas, asignándole una ponderación para cada una según su distancia con su centro.

En formulación matemática w_x^{ij} (x, y), según la ecuación 4.16, y w_y^{ij}(x, y), según la ecuación 4.17, corresponden al factor en la dirección x e y, d_x^{ij} y d_y^{ij} representan la distancia en ambas direcciones y tanto x como y representan la distancia entre centros de celdas en las dos direcciones. El histograma resultado se actualiza según la ecuación 4.18

$$w_{ij}^x(x,\, y) = \text{máx}\left(0,\, 1 - \frac{d_{ij}^x}{\delta x}\right) \tag{4.16}$$

$$w_{ij}^y(x,\, y) = \text{máx}\left(0,\, 1 - \frac{d_{ij}^y}{\delta y}\right) \tag{4.17}$$

$$h_{ij}(x,\, y) = \sum_{(x,\, y)} w_{ij}^x(x,\, y)\, w_{ij}^y(x,\, y)\, w_k(x,\, y)\, g(x,\, y) \tag{4.18}$$

Finalmente, para presentar invarianza a distintos tipos de iluminación que afecten al objeto, el descriptor HOG normaliza los histogramas localmente. Para esto las celdas se organizan en bloques cuadrados, siendo el histograma de cada bloque la concatenación de cada histograma correspondiente a las celdas. Cada bloque se diferencia del siguiente por una celda, por lo tanto cada celda aportará su histograma a tantos bloques como celdas entren en cada bloque, pero en cada uno ofrecerá un histograma distinto al ser normalizado independientemente por cada bloque, esto ofrece una mejor invarianza a la iluminación. El histograma de cada bloque se representa mediante un vector v, y para

normalizar usa un valor ϵ muy próximo a cero que asegura la no división entre cero en ecuación 4.19 de normalización.

$$v' = \frac{v}{\sqrt{\| v \|^2 + \epsilon}} \quad siendo \quad v = (h_1, h_2, ..., h_n) \tag{4.19}$$

Este resultado será el vector de características que se le pasará al clasificador lineal, encargado de localizar si un elemento dado corresponde o no al objeto que se intenta localizar. Para esto, se utilizan distintas técnicas de entrenamiento que crear un hiperplano capaz de separar los elementos de entrenamiento positivos y negativos de la forma mas efectiva.

El método de ventana deslizante, consiste en aplicar HOG a la imagen sobre un rectángulo de tamaño determinado, de manera que no se aplique a la vez en toda la imagen, este rectángulo irá desplazándose por la imagen buscando los objetos de interés a diferentes escalas.

4.4. Structure from motion

La estructura por movimiento, como se ha comentado previamente, es una técnica de reconstrucción tridimensional de superficies mediante la comparación de dos imágenes representativas del objeto a reconstruir. Para esto se realiza una detección de puntos a cada imagen. Normalmente se crea una nube de puntos de características invariantes, pero en este caso los puntos que se van a detectar están específicamente definidos como los puntos de referencia faciales.

Los movimientos de cada punto en dos imágenes permite identificar la orientación de la cámara en los instantes de tiempo en los que se toma cada fotografía. Una vez obtenidas las orientaciones, se le asigna a cada punto una posición tridimensional mediante un proceso de triangulación. Finalmente, un ajuste llamado "*bundle adjustment*" permite minimizar el error acumulado de la reconstrucción [21].

Mediante esa técnica se consigue un posicionamiento de los puntos de referencia facial del paciente en el espacio tridimensional, pero este posicionamiento no es absoluto, sino relativo entre cada uno de los puntos, por lo tanto el factor de escalado se pierde en la reconstrucción. Es decir, el tamaño de la cara reconstruida seguramente diferirá con las medidas en el mundo real, pero únicamente por un factor de escala que se puede obtener relacionando una de estas medidas con una del mundo real previamente conocida. Para conocer este factor de escala se ubicará el endoscopio en la punta de la nariz del paciente, al ser esta uno de los puntos de referencia facial. Con esto, obtendremos la pose de la cámara y haciendo una regla de tres con la longitud del endoscopio, se obtendrá el factor de escala necesario para representar la cara del paciente correctamente.

4.4.1. Correspondencia de puntos

Normalmente en situaciones en las que se presente una nube de puntos, hay que realizar una correspondencia entre descriptores, es decir, dado un punto en una imagen cual es el punto de la imagen siguiente más similar. En este caso, al tener un número finito de puntos y estar ordenados, el algoritmo solo debe relacionar las coordenadas de un punto con las del punto en la misma posición de la siguiente imagen.

4.4.2. Reconstrucción inicial

Obtener el movimiento consiste en conocer la rotación y traslación que han ocurrido de un instante al consecutivo de la cámara. En esta situación inicial, se toma únicamente el primer par de imágenes obtenidas. Con la correspondencia entre las coordenadas de los puntos de cada imagen se obtendrán la matriz fundamental y la esencial, que representan los movimientos relativos entre cámaras [36].

Con las dos primeras imágenes tomadas se nos presenta el escenario de la figura 4.7. Se pueden considerar las dos fotografías como tomadas de distintas cámaras con parámetros intrínsecos idénticos. La geometría epipolar nos permite obtener la representación del punto tridimensional X en la segunda imagen, únicamente conociendo la posición en la primera. Esta relación viene dada por la matriz fundamental.

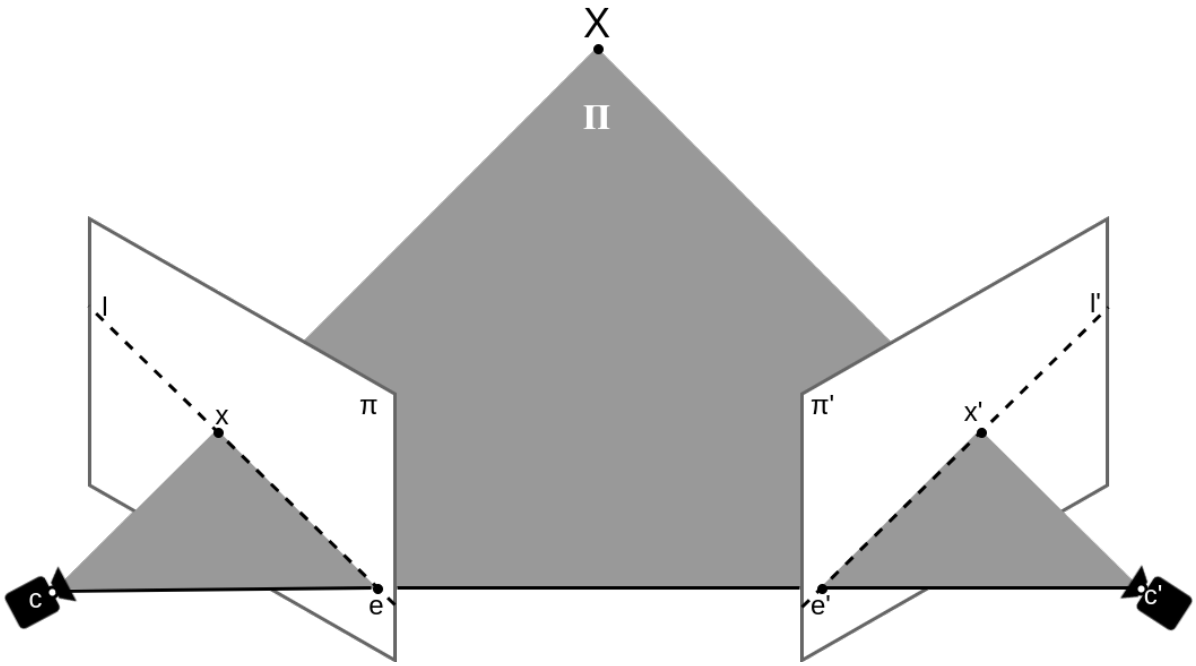

Figura 4.7: Relación entre dos cámaras, que visualizan un mismo punto, mediante geometría epipolar.

Es necesario conocer unos conceptos previamente:

- Centro de las cámaras (c). Orientación de la óptica de la cámara.
- Distancia focal. Distancia entre centro óptico de una lente y el foco.
- Planos de las imágenes (π). Cada uno de los planos que detecta cada cámara, separados de esta por la distancia focal, que mapea los objetos del mundo tridimensional en una imagen bidimensional.
- Epipolo (e). Intersección de la línea que separa los centros de dos cámaras con el plano de imagen de cada una de ellas.
- Plano epipolar (Π). Plano que contiene a los dos centros de cámara y al punto del espacio tridimensional.
- Línea epipolar (l). Línea resultado de la intersección del plano de imagen con el plano epipolar.

En la primera imagen, la línea que une el centro de la cámara con el punto a mapear representa las infinitas posiciones que se pueden deducir de la detección de este punto tridimensional. La visualización de esa recta en la segunda imagen corresponde con la línea epipolar. Por lo tanto, hay una homografía H_π entre la representación del punto en la primera imagen con el de la segunda. Esta homografía se formula como $x' = H_\pi x$. Además, la línea epipolar l' siempre pasa por el epipolo de esa imagen e', y como contiene con seguridad al punto mapeado x' en la segunda imagen $l' = e'x'$. Estas consideraciones se relacionan en la ecuación 4.20

$$l' = e' \cdot x' = e' \cdot H \cdot x = F \cdot x \tag{4.20}$$

Donde F es la matriz fundamental que mapea un punto del plano bidimensional de la primera imagen, como el conjunto de rectas de la segunda que pasan por su epipolo. Una propiedad muy importante de esta matriz fundamental es la siguiente: para cualquier par de puntos x $\leftrightarrow$ x' se satisface la restricción epipolar de la ecuación 4.21

$$(x')^t \, F \, x = 0 \tag{4.21}$$

Esto se cumple ya que al estar el punto x' contenido en la línea epipolar l' y según su relación con la matriz fundamental de la ecuación 4.22

$$(x')^t \, l' = 0 \rightarrow (x')^t \, F \, x = 0 \tag{4.22}$$

Si se cumple esta restricción a un conjunto de N puntos, se puede reformular a la ecuación 4.23.

$$\begin{bmatrix} x'_1 \cdot x_1 & x'_1 \cdot y_1 & x'_1 & y'_1 \cdot x_1 & y'_1 \cdot y'_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_n \cdot x_n & x'_n \cdot y_n & x'_n & y'_n \cdot x_n & y'_n \cdot y'_n & y'_n & x_n & y_n & 1 \end{bmatrix} \cdot \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = 0 \rightarrow \mathbf{A} \cdot \mathbf{f} = \mathbf{0} \quad (4.23)$$

Por lo tanto, hay que estimar la matriz fundamental de manera que el error sea lo menor posible respecto a todos los puntos. Para encontrar una única solución, el sistema debe ser de rango 8. Hay casos en el que el ruido puede provocar que el sistema tenga un rango 9, por lo tanto una solución aproximada es posible mediante el mínimo error cuadrático. Se puede realizar una estimación de la matriz fundamental mediante métodos iterativos como RANSAC.

Considerando el centro de la primera cámara como el origen de coordenadas del espacio tridimensional, la matriz esencial se usa para obtener la orientación del segundo centro de cámara. Una vez aplicada la matriz esencia, las coordenadas de la imágenes pasan de estar representadas en píxeles a estarlo en coordenadas normalizadas. Se considera K como la matriz de parámetros intrínsecos de la cámara, al ser ambas imágenes resultado de la misma cámara. Por lo tanto, la matriz esencial E se obtiene mediante la ecuación 4.24

$$E = K^t \, F \, K \quad (4.24)$$

4.4.3. Triangulación

Para reconstruir cada uno de los puntos, se usa la triangulación. Conociendo un par de puntos xx' que cumplan la restricción ($x'^t F x = 0$ y ambos centros de las cámaras obtenidos con la matriz esencial, se puede reconstruir el punto tridimensional X considerando π que

incluye a este y los dos centros de las cámaras. Las rectas que unen los centros de las cámaras con las representaciones del punto X en cada uno de sus planos de imagen se encuentran en este mismo plano π. Por lo tanto la intersección de estas rectas ocurrirá en un único punto X.

4.4.4. Añadir nuevas vistas

Una vez obtenida la reconstrucción inicial deben añadirse el resto de puntos detectados por cada imagen al modelo. No se puede usar el mismo método de la reconstrucción inicial ya que se considera que la primera cámara está localizada en el origen de coordenadas, y se obtiene la orientación de la segunda a una escala determinada, que no tiene porque coincidir con futuras reconstrucciones independientes.

Para resolver este problema se hace uso de la técnica PNP (Perspective-n-Point) que obtiene la orientación de una cámara, haciendo uso de los puntos que detecte y el modelo tridimensional inicial. Esta técnica se explicará en profundidad en el apartado de Estimación de la pose. Una vez obtenida la orientación de la cámara en cada instante para las siguientes imágenes, se vuelve a realizar una triangulación para obtener los puntos, añadiendo estos al modelo y refinandolo.

4.4.5. Bundle adjustment

Una vez procesadas todas las detecciones de las imágenes, es necesario un proceso de reajuste del error acumulado. Este error es origen tanto de la estimación de la matriz fundamental dada por los métodos iterativos, como por el error producido al añadir nuevas vistas a la reconstrucción inicial [25].

El error de reproyección se define como la diferencia en una imagen i del punto j detectado como x_{ij} a su posición original en la imagen obtenida mediante el punto en el espacio tridimensional X_j y la matriz de proyección de la imagen P_i. El error de reproyección ϵ se obtiene en la ecuación 4.25

$$\epsilon(P, X, x) = \sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij} \cdot \parallel x_{ij} - P_i \cdot X_j \parallel \tag{4.25}$$

Donde n es el número de imágenes, m es el número de puntos (68 en este caso) y w_{ij} se define en la ecuación 4.26

$$w_{ij} = \begin{cases} 1 & si\, el\, punto\, aparece\, en\, la\, imagen \\ 0 & en\, caso\, contrario \end{cases} \tag{4.26}$$

El *Bundle adjustment* permite refinar la reconstrucción minimizando la suma de distancias euclidianas entre los puntos proyectados y los reales. En la práctica, este ajuste es realizado por algoritmos de optimización no lineales. Para mejorar la velocidad de convergencia se usa una estructura jerárquica en subsecuencia que posteriormente se unen en una reconstrucción completa.

4.5. Estimación de la pose

La estimación de la pose consiste en averiguar la orientación de una cámara mediante su visualización planar de un objeto ubicado con precisión en el espacio tridimensional. Se necesita un modelo del objeto, es decir, se debe tener previamente conocimiento de la posición tridimensional de puntos etiquetados. La forma en la que aparezcan los puntos etiquetados en la imagen permite la obtención de la orientación de esta. El problema se resume a la resolución del sistema de la ecuación 4.27.

$$w \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{31} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.27}$$

Donde w es el factor de escala, [x y 1]^{t} las coordenadas homogéneas el punto localizado en la imagen, K representa los parámetros intrínsecos de la cámara, la siguiente es la matriz de parámetros extrínsecos de esta y [x y z 1]^{t} las coordenadas homogéneas del punto en el espacio.

Dependiendo del número de puntos usados el sistema tiene distintos grados de libertad. Aunque se añadan nuevos factores problemáticos, normalmente a una mayor cantidad de puntos se realiza una mejor localización. Teniendo más de tres puntos aparecen problemas con la coplanariedad y con el tratamiento de múltiples soluciones con ruido.

El caso más estudiado es el de P3P ya que es el número mínimo en el que el sistema está completamente restringido en el espacio tridimensional. Si se usan menos de tres puntos:

- P0P: El objeto mantiene los 6 grados de libertad en el espacio tridimensional. No se usa ningún punto por lo que no esta restringido de ninguna manera.
- P1P. El sistema tiene localizado un único punto, pero puede rotar en cualquier eje. En la traslación el sistema solamente permite el movimiento relativo a acercarse y alejarse de la cámara, ya que esta detectará el punto en la misma posición al mapearla en un plano. Por lo tanto el objeto tendría 4 grados de libertad.
- P2P. La rotación ya solo esta permitida sobre una línea que une a los dos puntos, pero al igual que en caso anterior se permite una traslación combinada con una rotación que mantenga la posición fija de los dos puntos en la imagen captada por la cámara.

4.5.1. P3P

Este caso es el más estudiado al ser el mínimo que restringe el objeto completamente en el espacio tridimensional. Aunque con tres puntos (A, B y C) no se permite ningún grado de libertad continuo, el sistema ofrece ocho soluciones. Normalmente cuatro se descartan al encontrarse detrás de la cámara que detecta el objeto. Por lo tanto suele añadirse un cuarto punto (D) que elimine esta ambigüedad [3]. Aparecen cuatro correspondencias entre puntos ($A \leftrightarrow u, B \leftrightarrow v, C \leftrightarrow w, D \leftrightarrow z$) como se aprecia en la figura 4.8.

Para resolver este problema se usa el teorema del coseno. Dado un triangulo arbitrario con vertices P, A y B, con un ángulo αformado entre los vectores $\overline{PA}$ y $\overline{PB}$ relacionados

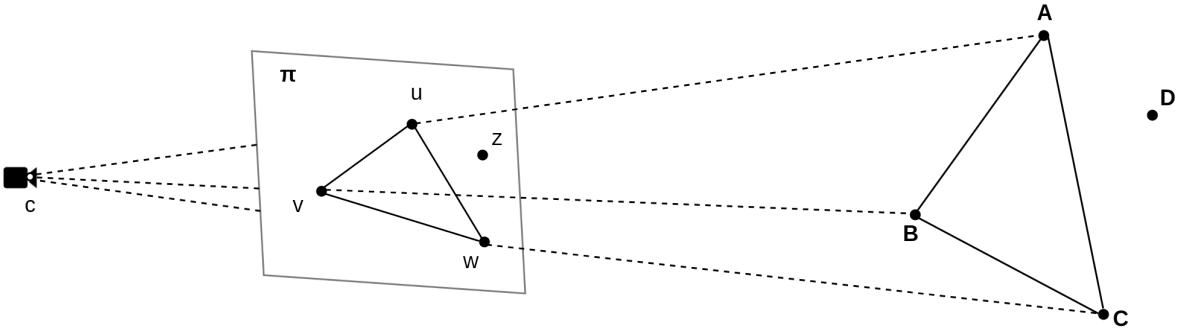

Figura 4.8: Relaciones entre puntos en P3P.

en la ecuación 4.28.

$$(\overline{PA})^2 + (\overline{PB})^2 - 2\overline{PA}\,\overline{PB} - \cos(\alpha) = (\overline{AB})^2 \tag{4.28}$$

Aplicando en el espacio tridimensional se llega al sistema de la ecuación 4.29. Resolviendo este sistema en combinación con RANSAC y aplicando el punto D se obtiene la orientación de la cámara.

$$\begin{cases} (\overline{PB})^2 + (\overline{PC})^2 - 2 \cdot \overline{PB} \cdot \overline{PC} \cdot \cos(\alpha_{v,w}) - (\overline{BC})^2 = 0 \\ (\overline{PA})^2 + (\overline{PC})^2 - 2 \cdot \overline{PA} \cdot \overline{PC} \cdot \cos(\alpha_{u,w}) - (\overline{AC})^2 = 0 \\ (\overline{PA})^2 + (\overline{PB})^2 - 2 \cdot \overline{PA} \cdot \overline{PB} \cdot \cos(\alpha_{u,v}) - (\overline{AB})^2 = 0 \end{cases} \tag{4.29}$$

4.5.2. EPnP

Para resolver la técnica PNP se diferencian los métodos iterativos y no iterativos. En este caso el EPnP (*Efficient Perspetctive-n-Points*) es un método no iterativo para la resolución de sistemas con una $n > 3$ sin pérdida de precisión que ofrece una mejora en eficiencia respecto a otras alternativas [33].

Se hace uso de cuatro puntos de control seleccionados arbitrariamente mediante los cuales se representarán los puntos del objeto. El punto de control j en el espacio tridimensional es c_w^j mientras que su representación en la imagen es c_c^j. De la misma manera se representan los puntos en el espacio tridimensional p_w^j, con ecuación 4.30, y en la imagen

p_c^j, con ecuación 4.31. Siendo α una ponderación aplicada a las coordenadas de control, con ecuación 4.32.

$$p_i^w = \sum_{j=1}^{4} \alpha_{ij} \cdot c_j^w \tag{4.30}$$

$$p_i^c = \sum_{j=1}^{4} \alpha_{ij} \cdot c_j^c \tag{4.31}$$

$$\sum_{j=1}^{4} \alpha_{ij} = 1 \tag{4.32}$$

Por lo tanto la ecuación 4.33 representa el sistema actualizado a resolver para EPnP.

$$w_i \cdot \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} f_u & 0 & u_c \\ 0 & f_v & v_c \\ 0 & 0 & 1 \end{bmatrix} \sum_{j=1}^{4} \alpha_{ij} \begin{bmatrix} x_j^c \\ y_j^c \\ z_j^c \end{bmatrix} \tag{4.33}$$

Donde las incógnitas son las 12 coordenadas de los puntos de control y los parámetros proyectivos w_i. Se observa de la última fila que $w_i = \sum_{j=1}^{4} \alpha_{ij} = z_j^c$ por lo tanto se obtiene el sistema de la ecuación 4.34.

$$\begin{aligned} \sum_{j=1}^{4} \alpha_{ij}\, f_u\, x_j^c + \alpha_{ij}\, (u_c - u_i)\, z_j^c = 0 \\ \sum_{j=1}^{4} \alpha_{ij}\, f_v\, y_j^c + \alpha_{ij}\, (v_c - v_i)\, z_j^c = 0 \end{aligned} \tag{4.34}$$

Este sistema ya no depende del factor w_i y concatenando para todos los puntos obtenemos un sistema linear con forma:

$$M \cdot x = 0 \tag{4.35}$$

Donde x es un vector que contiene a los cuatro puntos de control con sus 12 coordenadas y M es una matriz 2n x 12 obtenida de los parámetros de las dos ecuaciones anteriores. Con esto el conjunto de resultados será el Kernel o conjunto vacío de la matriz M. Con esto se puede obtener los parámetros intrínsecos R y t por métodos lineales de manera que se minimice el error.

Capítulo 5

Implementación

5.1. Introducción

En este capítulo se desarrollan las decisiones tomadas durante el desarrollo de los módulos presentados en la arquitectura software. Por cada mayor funcionalidad se muestra con pseudocódigo su funcionalidad, los problemas encontrados y la soluciones tomadas.

5.2. Detección de caras

Este proceso se implementa principalmente en el módulo de detección, haciendo este uso de los de imagen y clasificación. Gracias a este último se obtienen todas las áreas en las que se ha reconocido una cara mediante los algoritmos de OpenCV.

Un área de interes o AOI (*Area Of Interest*) de una imagen se define como una región de esta que contiene información útil. En el contexto de este trabajo, la principal región es la cara del paciente.

Se usa un rectángulo para representar las distintas áreas de interés. Este rectángulo puede ser representado de distintas formas, variando según librerías, pero en este casos se mantiene la representación de OpenCV.

La decisión de usar los rectángulos de OpenCV reside en su sencillez. El rectángulo de OpenCV, en Python, está formado por una lista de cuatro valores: los dos primeros corresponden a las coordenadas del punto superior izquierdo del rectángulo, los últimos dos a la anchura y la altura del rectángulo. Un ejemplo se observa en la figura 5.1.

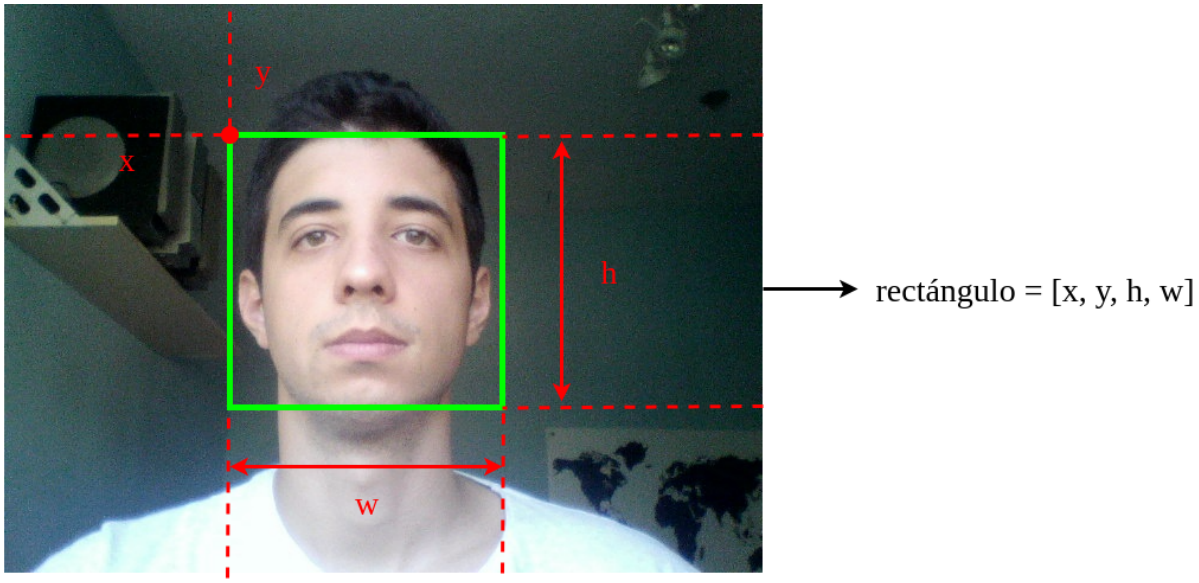

Figura 5.1: Representación de un rectángulo de OpenCV

Al comprobar la detección de caras de este módulo a tiempo real, se observa como varias imágenes se detectan incorrectamente. Normalmente son causadas por el ruido de la cámara y la iluminación, que confunden al algoritmo pudiendo detectar caras en el fondo o en la nariz.

Por lo tanto se hace necesario implementar una metodología por la cual el detector discierna entre lo que es o no una cara. Teniendo en cuenta que el endoscopio siempre pasa por las narinas, se define la mejor cara como aquella en la que cuyo centro sea mas cercano al centro de la imagen. Esto elimina la mayoría de detecciones causadas por ruido.

Lo que no elimina este último paso es la detección de caras en la nariz y otras regiones confundidas dentro de la cara, que están más cercanas del centro. Para esto se pasa a definir una cara como un conjunto de regiones ordenadas.

Región	Cara	Ojo	Nariz	Boca
Porcentaje	1x	18 %	32 %	32 %

Tabla 5.1: Porcentaje por cada componente de la cara.

El módulo de detección está conformado por clases basadas en la clase Detector que detectan secciones de la cara como ojos, nariz, boca y contorno. Por lo tanto una cara es considerada la mejor a mayor sea su composición.

Para esto se realiza la detección usando la clase FrontalFaceDetector, que unifica la detección de la cara con la detección de sus subsecciones. Según el número de componentes detectados se le asigna un porcentaje de confianza, en la que cada elemento de la cara aporta los porcentajes que se observan en la tabla 5.1.

El significado del porcentaje de la cara reside en que si no es detectada por el algoritmo, el porcentaje es inmediatamente 0 %. Cabe destacar que el 18 % de la tabla es para cada ojo, por lo que ambos conllevan un 36 % de detección, alcanzando un 100 % al sumar el resto de elementos.

Por último, se observa como al localizar componentes en la región de la cara, el algoritmo no coordina la posición de los elementos. Esto es, la nariz debe ir entre los ojos, y la boca debajo de esta. Por ejemplo, el algoritmo suele detectar varias bocas dentro de la región de la cara.

Para esto se separa en zonas la cara detectada en las cuales deben localizarse los componentes para tener coherencia entre sí. Son lo suficientemente grandes como para contener los elementos en caso de una rotación de la cabeza. Las regiones y los diferentes porcentajes posibles de una cara se muestran en la figura 5.2.

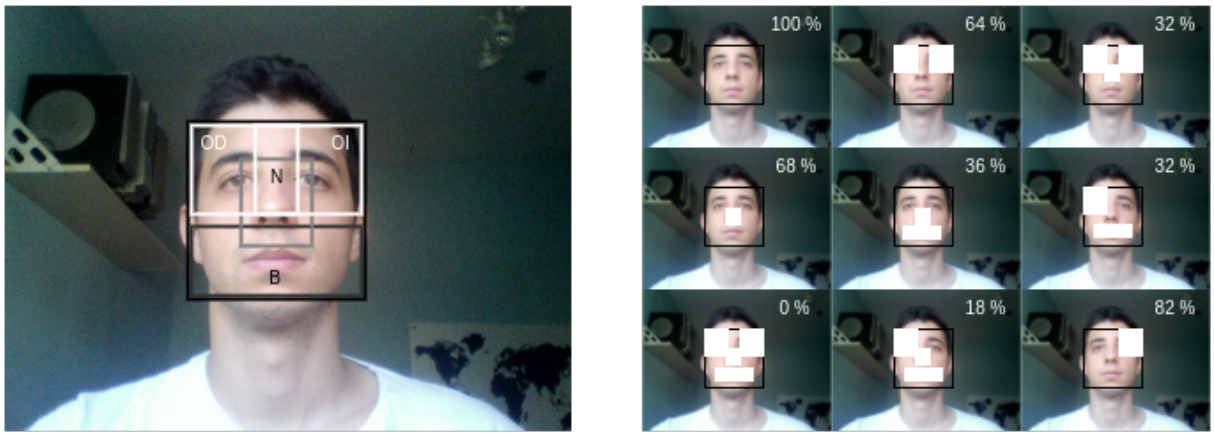

Figura 5.2: Regiones de búsqueda de la cara y porcentajes según detección la de componentes.

5.3. Puntos de referencia faciales

Esta funcionalidad se implementa en el módulo de localización. La librería DLib usa una representación de los AOI distinta a los de OpenCV, por lo que es necesaria una conversión realizada en el módulo util.

La implementación permite el paso o no de una región que localice la cara. De esta forma buscará en esta región en el caso de pasarse o se buscará en toda la imagen. La primera opción devuelve resultados más precisos de manera más eficiente. La segunda opción permite la localización de los puntos faciales en situaciones complicadas para el detector de caras, como que la cámara esté demasiado cerca de la cara.

5.4. Reconstrucción

Esta funcionalidad se implementa en el módulo homónimo. El método de reconstrucción de OpenCV no es compatible con Python, por lo que se genera código en C++ para cubrir esta funcionalidad, que será ejecutado desde el módulo de Python.

Para comunicar ambos códigos, se guardan los puntos de entrada bidimensionales en una carpeta específica por cada reconstrucción. Estos puntos deben ir formateados co-

rrectamente para satisfacer la estructura que requiere el método.

La primera línea debe contener tres valores con los parámetros intrínsecos de la cámara: la distancia focal y las dos coordenadas del centro de la imagen. Cada línea siguiente representa una imagen, en la que sus primeros 68 puntos corresponden con la coordenada X del punto, y los últimos con la coordenada Y.

Una vez reconstruidos los puntos, el código C++ guarda el fichero de salida de forma similar, con los parámetros instrinsecos actualizados y un punto tridimensional por cada línea.

El proceso de toma de imágenes para la reconstrucción y la reconstrucción en si se observa en el algoritmo 1.

5.5. Estimación de la pose

Esta funcionalidad se implementa en el módulo de posicionamiento. Conforma el proceso principal de la cirugía, requiriendo poca configuración tras la reconstrucción anterior.

La reconstrucción de la cara nos ofrece un conjunto de puntos tridimensionales, pero su localización en el espacio es a efectos prácticos es arbitraria. Es necesario ubicar los puntos de maneras que puedan ser útiles para la estimación de la pose.

Se opta por colocar los puntos de la cara en el eje de coordenadas, pasando a ser la estructura de referencia sobre la que el endoscopio se posiciona. Se toma el punto medio entre las narinas como el origen de coordenadas, que será el punto aproximado de introducción del endoscopio.

Una vez trasladados los puntos es necesaria su rotación de una manera útil para el proceso, por lo que se introduce el concepto de plano cara. Este plano se crea para que el facultativo tenga una referencia fácil de ubicar en cualquier momento de la cirugía

Algoritmo 1 Modelado facial tridimensional de la cara del paciente

Salida: Puntos tridimensionales P^3 como modelo del paciente

FIN $\leftarrow$ **falso**
mientras not FIN **hacer**
  Obtener conjunto de imágenes I
  por cada i $\in$ I **hacer**
    $c \leftarrow$ mejor cara detectada en i
    $P^2 \leftarrow$ puntos faciales localizados de la cara c
    $L \leftarrow \{\varnothing\}$
    si la localización es correcta $\forall p \in P^2$ **entonces**
      $L \leftarrow L + P^2$
    fin si
  fin por
  Guardar L en fichero de entrada F_i
  structureFromMotion(F_i)
  $P^3 \leftarrow$ puntos tridimensionales en fichero de salida F_o
  si P^3 es un modelo correcto **entonces**
    FIN $\leftarrow$ **cierto**
  fin si
fin mientras
devolver P^3

mediante el posicionamiento del endoscopio normal a este plano.

Para conformar el plano cara hacen falta tres puntos de la cara. Se toman los puntos de la base de la nariz y el punto superior a esta para formar el plano, y a continuación se desplaza de manera que contenga al punto medio de las narinas. Este procedimiento se observa en la figura 5.3.

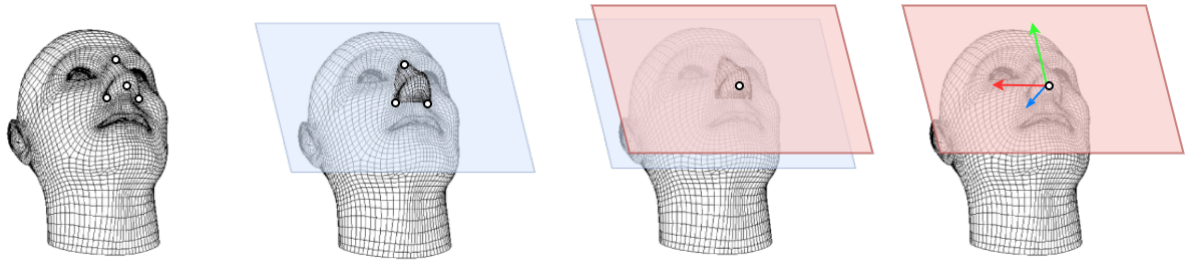

Figura 5.3: Posicionamiento del plano cara.

La orientación de los puntos se implementa en la clase Aligner, cuyas consideraciones para la colocación de los ejes son las siguientes:

1. El eje 'z' representa la profundidad del endoscopio. Por lo tanto, será normal al plano cara en dirección de la introducción del instrumental, con valores positivos a mayor sea esta.

2. El eje 'x' se coloca contenido en el plano cara, apuntando hacia el ojo derecho desde la punta de la nariz. De esta manera, al acercar el endoscopio a la frente se realizará una intuitiva rotación positiva.

3. El eje 'y' se coloca perpendicular a los dos anteriores.

La obtención de la pose para una imagen dada se implementa en la clase Positioner. Se usa el método de estimación de la pose de OpenCV 'solvePnP' para obtener los resultados a partir de un objeto de la clase Frame. Esta clase es muy sencilla pero de gran importancia ya que en ella se ha estudiado uno de los grandes retos encontrados durante el desarrollo.

Durante el desarrollo se detecta como el resultado del algoritmo es extremadamente sensible y muy propenso a ofrecer resultados incorrectos. La mayoría de estos resultados se filtran y procesan posteriormente, pero se observa que dependiendo de los elementos que se tengan en cuenta de la cara, la estimación es mejor o peor.

Dependiendo de que elementos se tengan en cuenta, se obtiene un conjunto de 255 caras diferentes, al estar compuestas por 8 elementos principales: el contorno de la cara, ambos ojos, ambas cejas, la nariz, la boca exterior y la boca interior. Estas dos últimas limitan la parte exterior e interior de los labios.

El método a seguir es realizar un experimento (que se desarrollará en su correspondiente capítulo) de manera que se obtenga un error en orientación por cada imagen tomada, conociendo su orientación real y la calculada.

La clase 'Pose' es la encargada de ejecutar todo el proceso de estimación de la pose: preprocesado de la imagen, obtención de la orientación mediante la clase 'Positioner' y, por último, filtrado y corrección de datos obtenidos. El funcionamiento resumido de la estimación de la pose se puede observar en el algoritmo 2.

Algoritmo 2 Estimación de la pose de la cámara respecto al paciente

Entrada: Modelo tridimensional M del paciente

Salida: Rotación R y traslación T en cada iteración

```
1: mientras cierto hacer
2:     Obtener imagen i
3:     Filtrar y redimensionar i
4:     si i contiene la cara del paciente entonces
5:         L ← Puntos faciales de referencia localizados en i
6:         T, R ← solvePnPRansac(M, L)
7:         Pintar y representar T y R
8:     fin si
9: fin mientras
```

Esta clase recibe un conjunto de puntos tridimensionales y unos parámetros intrínsecos de la cámara, sobre los que devolverá las estimaciones. En su ejecución se estima la pose actual de la imagen, mediante su representación por un objeto de tipo Frame. Opcionalmente, se le puede pasar una imagen de referencia y la distancia real a la que se ha tomado.

La imagen de referencia debe ser tomada normal al plano cara y de la forma más precisa posible, ya que sirve para eliminar el error inicial de la estimación. Una vez calculada la orientación de esta imagen, se restan los componentes de rotación obtenidos a las siguientes imágenes. Según la traslación obtenida, se generará un factor que relacione las unidades arbitrarias del modelo tridimensional y la distancia real.

Al pasarle una imagen, la clase llama a su instancia de la clase 'Positioner' para obtener los valores devueltos por el método 'solvePnP'. Los datos devueltos representan la orientación de la cámara.

Se observa como el algoritmo no es capaz de detectar la diferencia entre estar observando la cara de frente o por detrás, ya que esta está compuesta por un conjunto de puntos sobre su superficie. Por lo tanto, si se detecta que la localización de la cámara está sobre la zona positiva del eje Z (interior de la cara) se realiza una rotación sobre el eje Y, de manera que la localización del nuevo eje de coordenadas aparezca reflejada sobre el plano XY.

Se toma un vector que una el origen de coordenadas con la posición de la cámara, y se calcula el ángulo α sobre el plano XY según la ecuación 5.1. Sobre la orientación actual de la cámara, se realiza la rotación de ángulo total β como muestra la ecuación 5.2. Este proceso, y sus variables, se observa en la figura 5.4.

$$\alpha = \tan^{-1}\left(\frac{z}{x}\right) \tag{5.1}$$

$$\beta = 2 \cdot \alpha \tag{5.2}$$

En abundantes resultados de la orientación, los ángulos de las rotaciones superan una

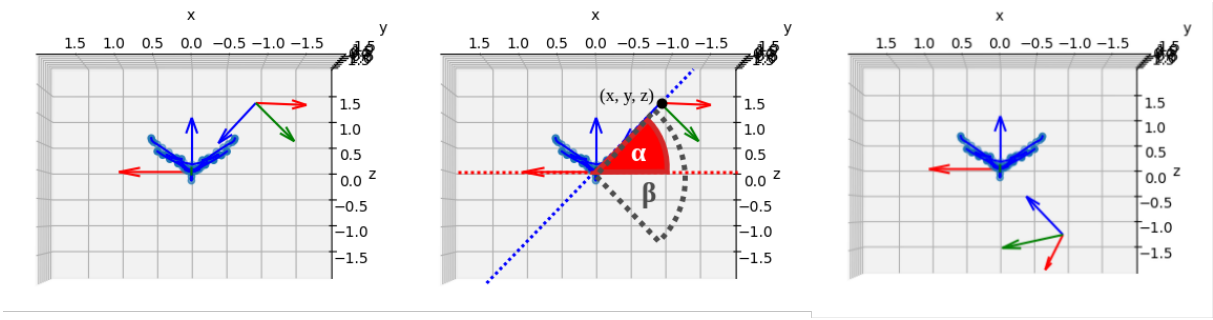

Figura 5.4: Corrección de sistema ubicado en el interior de la cara.

revolución, es decir, ángulos mayores a 2π radianes. Estas orientaciones, aunque correctas, representan valores que pueden ser simplificados, deshaciendo las revoluciones sobrantes.

Una vez obtenida las rotaciones correctas, se nota como representar los ángulos de 0 radian a 2π radian produce una discontinuidad entre los ángulos cercanos a 0, ángulos muy habituales para este proceso. Por lo tanto se traslada esta discontinuidad a los ángulos cercanos a π radians, pasando los valores a este ángulo a valores negativos. Se observa el concepto en la figura 5.5.

Durante el proceso de detección, alineamiento y reconstrucción se van acumulando errores en las rotaciones que se muestran posteriormente en la orientación de la cámara. Por esto se realiza la toma de una imagen de referencia por parte del facultativo, ayudado por el operador, lo más normal posible al plano cara. Al conocer su posición ideal, se toman sus componentes de rotación y se restan a las siguientes orientaciones.

Gracias a la orientación de la imagen de referencia, y conociendo la longitud del endoscopio, se consiguen relacionar las unidades arbitrarias del modelo con las reales. Este factor debe aplicarse tanto al modelo tridimensional, como a los puntos localizados en cada imagen y al resultado de la orientación de la cámara.

Un último problema que aparece en la estimación de la pose, es que el algoritmo de OpenCV no es capaz de distinguir entre una rotación y traslación y una rotación pura.

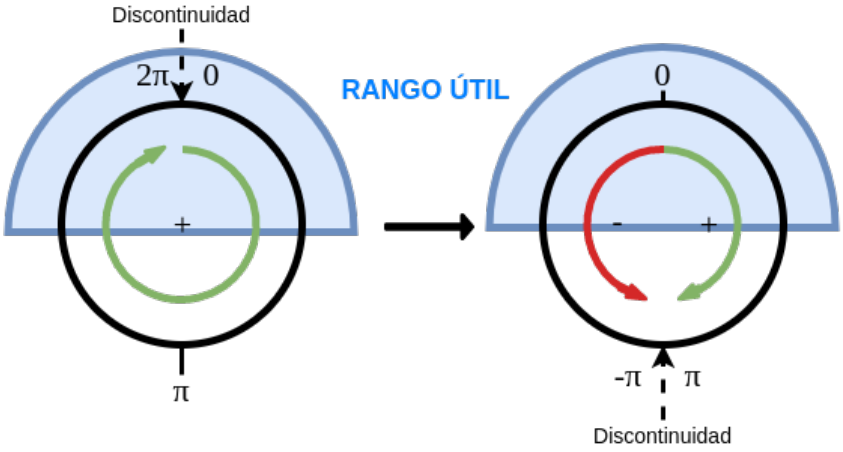

Figura 5.5: Modificaciones en las discontinuidades de ángulos.

Por lo tanto se pueden obtener resultados como los de la figura 5.6 (izquierda). Para esto, sabiendo que el endoscopio siempre va a pivotar por las narinas, se orienta el eje Z de manera que apunte hacia estas. De esta forma, se les otorga una nueva localización a la cámara obteniendo el resultado de la figura 5.6 (derecha).

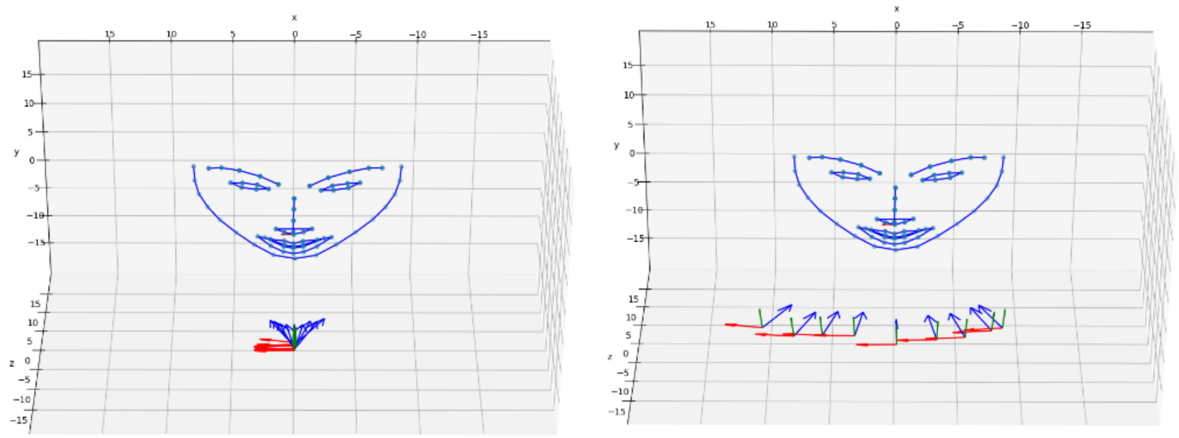

Figura 5.6: Resultado incorrecto de orientación (izquierda) y corrección (derecha)

Esta última técnica impone ciertas condiciones que favorecen a la estimación de la pose. Al apuntar el eje Z, la rotación detectada sobre este eje no aporta información útil para el facultativo. Una rotación sobre este significa una rotación sobre el cilindro metálico del endoscopio, que no modifica su localización de la punta, ni el plano de visión de la cámara del endoscopio. Lo único que modifica es la rotación de la imagen que capta el endoscopio, no teniendo ningún efecto real sobre los objetivos de este trabajo.

Otra consecuencia es la obtención directa de la orientación de la cámara y por tanto, la profundidad a la que se ha introducido el endoscopio. Se suma una copia del vector del eje Z (que apunta a las narinas) con un módulo igual a la longitud del endoscopio a la localización de la cámara, obteniendo la orientación de la punta. La distancia euclidiana entre el origen de coordenadas y la punta de la cámara indica la profundidad actual de esta.

Capítulo 6

Procedimiento

6.1. Introducción

En este capítulo se desarrolla la metodología que se seguirá en el transcurso de la cirugía. Se detallan los pasos a seguir tanto por parte del facultativo médico como del operador que controla el software que se desarrolla en este trabajo.

Acompañando a cada etapa se muestra la interfaz que se ha implementado para conseguir una configuración rápida y fácil sin necesidad de escribir código. Se destaca la sencillez de la interfaz al no ser el objetivo principal de este trabajo, ofreciendo una representación básica de las capacidades del sistema.

El proceso, y por lo tanto este capítulo, se divide en dos partes bien diferenciadas: el preoperatorio y el procedimiento quirúrgico. Aunque en el postoperatorio puedan realizarse distintas tareas, no se incluye en este capítulo ya que no comprende un paso relevante para el trabajo.

6.2. Preoperatorio

Es el conjunto de procesos relativos a la cirugía pero previos a esta. Por lo tanto abarca procesos psicológicos del paciente, pruebas preliminares y el seguimiento de dietas

o ayuno. En esta sección se desarrolla el preoperatorio desde el momento en el que el paciente y los facultativos entran en el quirófano hasta el instante previo del inicio de la cirugía.

6.2.1. Calibrado de la cámara

Para iniciar el procedimiento la cámara usada debe estar calibrada, es decir, es necesario conocer los parámetros intrínsecos de esta. Este proceso puede realizarse fuera del entorno quirúrgico, siendo preferible que sea de esta manera. Para la realización de este trabajo se ha usado la calibración por patrones de tablero de ajedrez.

La primera pantalla de la interfaz está relacionada con el dispositivo a usar para captar las imágenes. Se solicita el canal al que está conectada la cámara, por si hay más de una, y el nombre del dispositivo calibrado. Este nombre se asociará con los datos de la matriz de la cámara (mtx) y los vectores de rotación (dist). Tras introducir los datos debe observarse una pantalla como la figura 6.1.

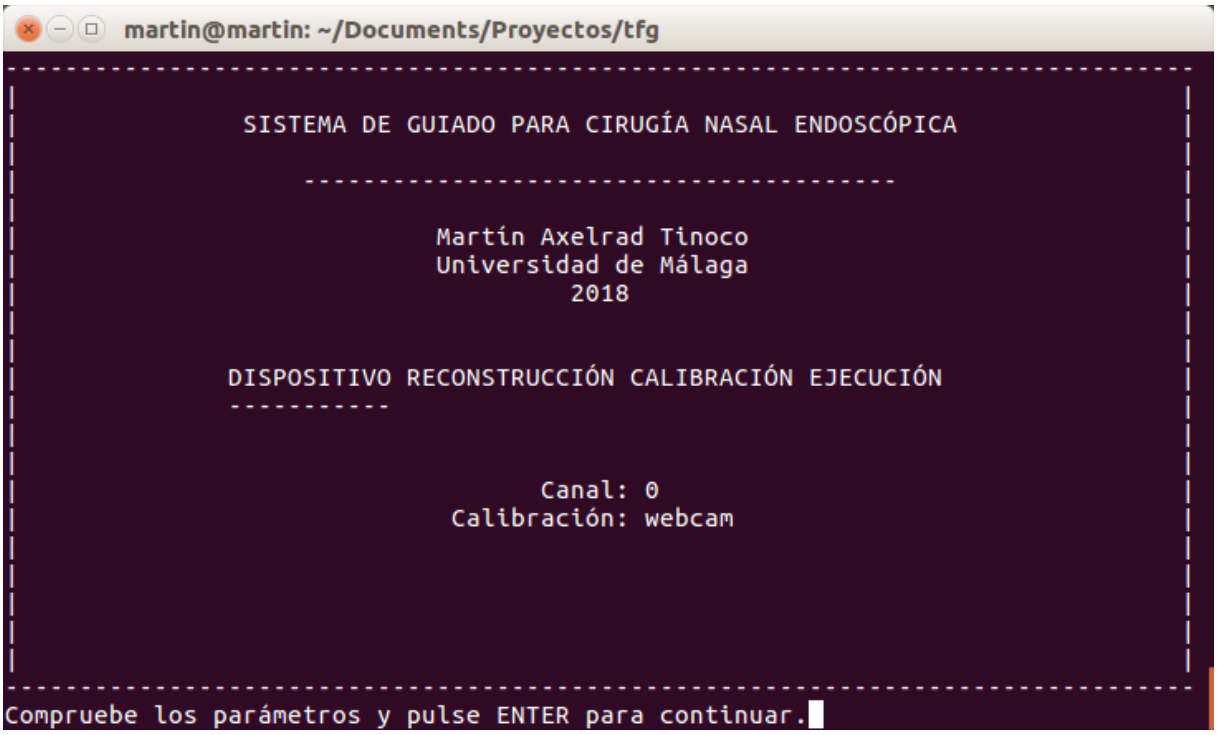

Figura 6.1: Interfaz para la configuración del dispositivo de grabación.

6.2.2. Modelado facial tridimensional

Este proceso se inicia cuando el paciente esta bajo los efectos de la anestesia, colocado en la mesa de operación y con la iluminación adaptada para el procedimiento quirúrgico. Una vez acoplada la cámara al endoscopio, el facultativo lo coloca en la punta de la nariz del paciente, y realiza un movimiento recto de lado a lado para capturar las imágenes como se aprecia en la figura 6.2.

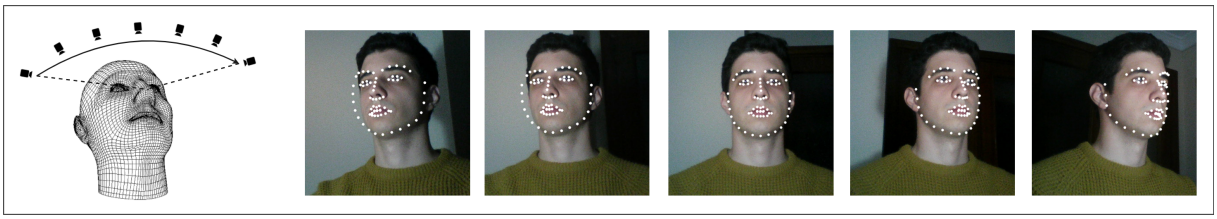

Figura 6.2: Toma de imágenes para la reconstrucción

La interfaz de usuario permite la carga de una reconstrucción ya realizada o crear una nueva. Para ambos casos solicita un nombre que identifique la reconstrucción. En el caso de crear una nueva reconstrucción se muestra por pantalla la visión de la cámara, y las instrucciones para obtener las imágenes que generen el modelo tridimensional.

Una vez finalizado el proceso debe observarse una ventana en la interfaz similar a la de la figura figura 6.3. Se observa un ejemplo de las ventanas de toma de imágenes y la reconstrucción obtenida en la figura 6.4

6.2.3. Imagen de referencia

Una vez obtenida la reconstrucción, la interfaz solicita la toma de una imagen de referencia de la cara del paciente, mostrando la visión de la cámara y las instrucciones para tomar dicha imagen. Con esta imagen se elimina cualquier error inicial que tenga la estimación de la pose.

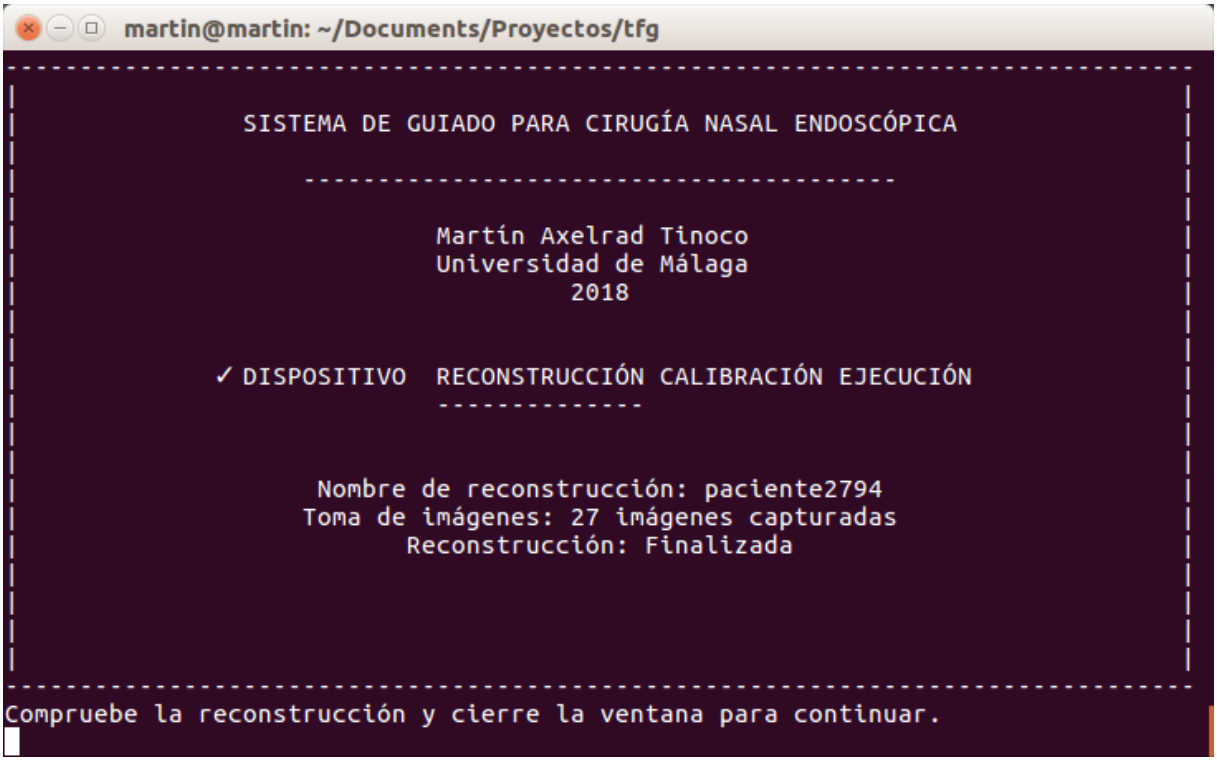

Figura 6.3: Interfaz de la toma de imágenes para la reconstrucción.

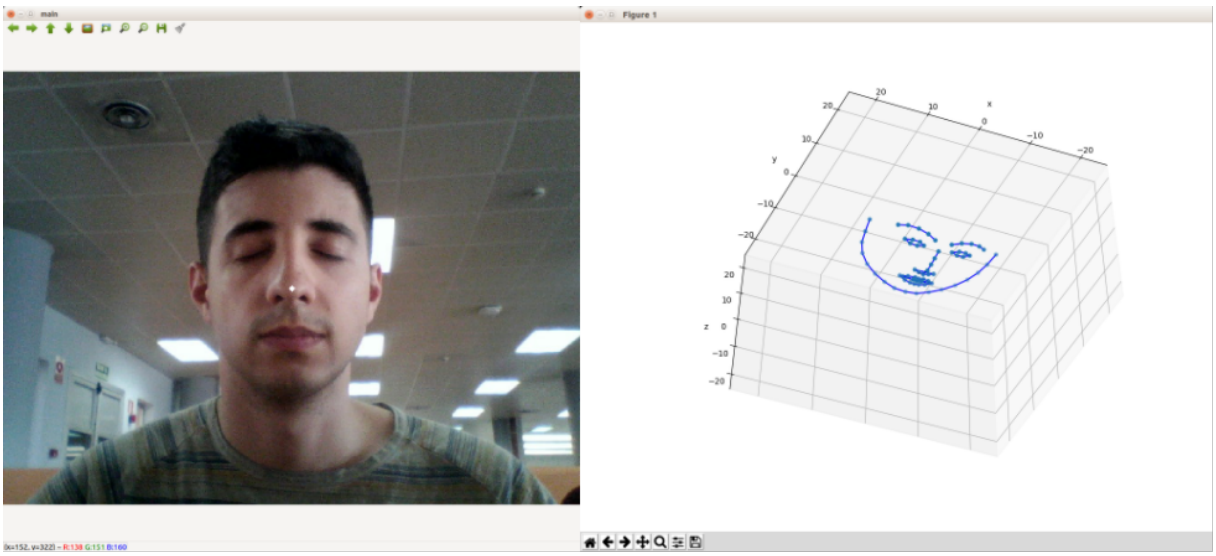

Figura 6.4: Ventanas de toma de imágenes y reconstrucción obtenida.

También se obtiene el factor de escala que permita calcular la relación entre las unidades arbitrarias del modelo y la realidad al solicitar la distancia a la que se ha tomado

la foto, que debe coincidir con la distancia de la cámara a la punta del endoscopio.

Al terminar el proceso la interfaz muestra un aspecto como el que se aprecia en la figura 6.5.

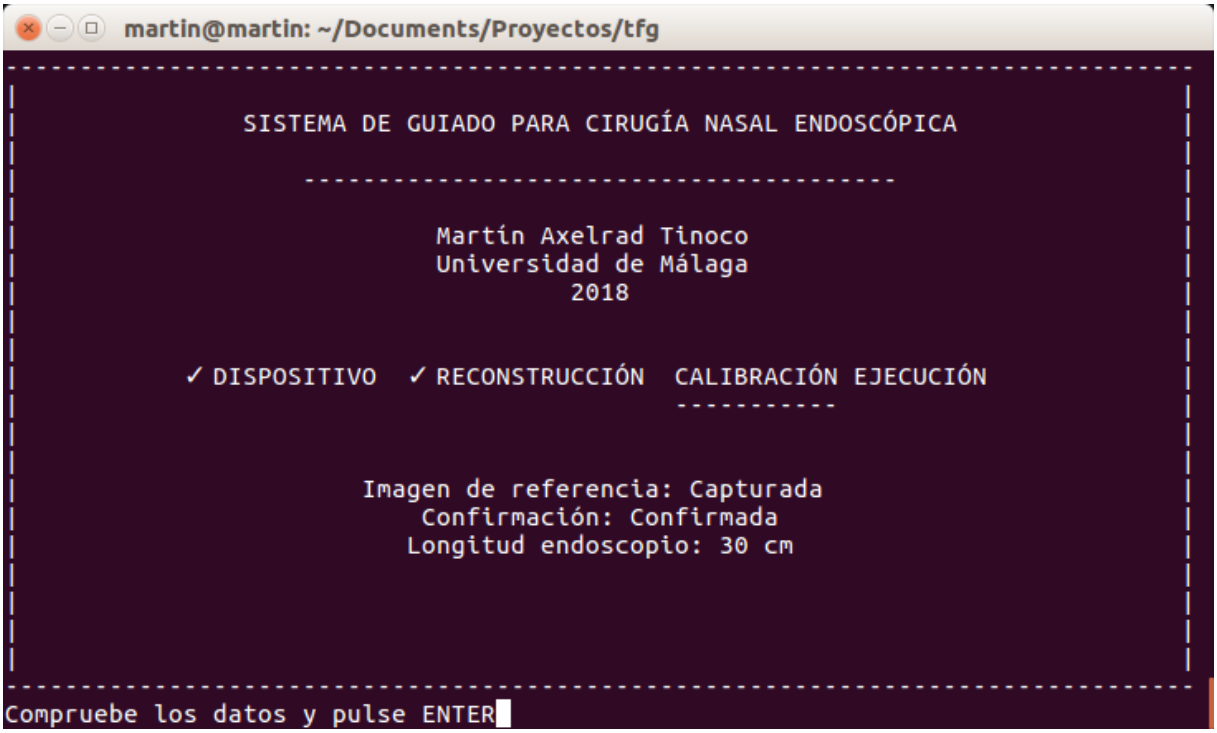

Figura 6.5: Interfaz tras la toma de la imagen de calibración y longitud del endoscopio.

6.3. Cirugía

La cirugía, en el sentido de este proyecto, consiste en todo el proceso comprendido entre el primer acercamiento del instrumental al paciente hasta el momento de la retirada de la anestesia. Durante todo este proceso el sistema se encargará de obtener la orientación de la cámara a tiempo real.

Una vez completados todos los pasos de configuración, se muestra en pantalla dos ventanas adicionales a la del terminal. La primera muestra la visión actual de la cámara con la cara y los puntos faciales detectados. La segunda, más grande que la anterior y a

la derecha, muestra la orientación de la cámara a tiempo real sobre el modelo del paciente.

La orientación de la cámara se representa con un eje de coordenadas. Su eje Z tiene la longitud del endoscopio, por lo que el final de la flecha representa la ubicación de la punta de este. En la ventana del terminal se muestran los datos a tiempo real de la localización de la punta del endoscopio y la rotación del instrumento.

Un ejemplo de lo que muestra la interfaz se observa en la figura 6.6.

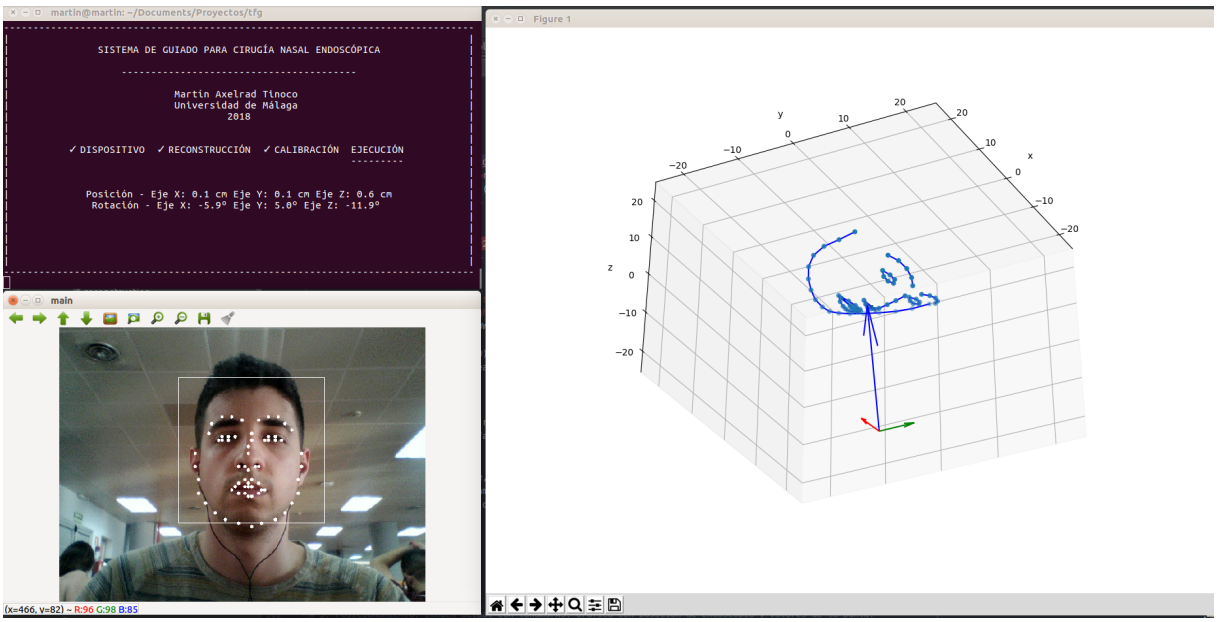

Figura 6.6: Interfaz de la ejecución del programa durante el procedimiento quirúrgico.

Capítulo 7

Experimentos

7.1. Introducción

En esta sección se realizaran los experimentos necesarios para obtener datos sobre los que basar las conclusiones y saber si se han alcanzado los objetivos de este trabajo.

El dilema que presentan los experimentos es la necesidad de un marco de referencia que permita obtener datos completamente controlados del entorno de la cirugía. Es decir, se necesitan unos datos reales para comparar con los obtenidos por el programa, pero estos son muy dificiles de conseguir al no poder controlar todas las variables necesarias en el mundo real.

Por esto se ha hecho uso de un entorno virtual en que se puedan controlar todas las variables necesarias. Para esto se ha colocado el modelo tridimensional de una cabeza en el programa Unity3D. Tras localizar la cabeza como dispone el alineado, se localiza la cámara en las posiciones deseadas para obtener los resultados a los distintos experimentos.

7.2. Detector de caras

La detección de caras es un apartado fundamental del proceso. Este es el que decide si se esta observando o no una cara durante la cirugía, por lo que debe ser rápido pero

		Clasificación		
		Cara	No cara	Total
Realidad	Cara	831 (71 %)	339 (29 %)	1070
	No cara	6 (2 %)	343 (98 %)	349

Tabla 7.1: Matriz de confusión del detector de caras

efectivo. En este apartado se estudian distintas técnicas para agilizar o mejorar la detección de caras.

7.2.1. Evaluación del detector

Para realizar esta evaluación, se han usado dos sujetos de la base de datos *Yale Face Database B*[43][22], en concreto los sujetos B15 y B18. Como imágenes negativas (que no contengan caras) se han tomado imágenes de forma aleatoria de la base de datos *Caltech Image Dataset*[19]. Tras realizar la detección de caras sobre todas las imágenes se obtiene la matriz de confusión de la tabla 7.1

Estos resultados se obtienen sobre imágenes con diferencias de iluminación, llegando a situaciones casi de oscuridad. Realizando pruebas sobre un entorno bien iluminado la detección de caras es casi completa, pero manteniendo los errores en falsos positivos.

7.2.2. Límites de movimiento

La limitación en movimiento de rotación viene dada por la capacidad del detector de caras en reconocer caras desde distintos ángulos. Para esto se ha rotado la cámara en cada uno de los ejes, apuntando a la punta de la nariz, manteniendo una distancia constante de 30 cm y tomando una foto cada cinco grados.

Todas las imágenes se han tomado con un FOV de 50 y en una resolución de 1920 x 1080 píxeles. Estas imágenes se pasan al detector de caras, observando cual es la primera en la que se encuentre una cara. Por último, para afinar el proceso se han tomados fotos

	Rotación min.	Rotación max.	Rango
Eje x	-45º	62º	107º
Eje y[1]	-45º	44º	89º
Eje z	-17º	19º	36º

Tabla 7.2: Rango de rotaciones sobre cada eje que limita la detección de caras

cada un grado en los límites detectados, para poder afinar más el resultado. Con esto se obtienen los resultados de la tabla 7.2.

7.2.3. Tamaño óptimo de la cara

Dependiendo del sensor, cada cámara nos ofrece distintas resoluciones de imagen. Esta resolución afecta en gran medida a la eficiencia de los algoritmos, ya que aumentan el área en número de píxeles en el que se tiene que ejecutar. Este detalle es particularmente costoso para el algoritmo de detección de cara, que necesita recorrer toda la imagen con muchas características de Haar a distintas escalas.

Se considera el tamaño óptimo de la cara a aquel en el que se obtenga una cantidad de detecciones suficientes como para considerarse efectivo. Para esto se escala el conjunto de imágenes de caras tomados en la evaluación sin modificaciones y se les aplica la detección de caras, obteniendo el porcentaje de aciertos y el tiempo requerido en el PC Lenovo Z510 como se aprecia en la figura 7.1.

Observando los resultados se toma el tamaño óptimo de la cara como un rectángulo de 40 píxeles por lado, llegando a procesar 133 imágenes por segundo. Tras obtener el tamaño de la cara con una cámara a la distancia máxima (30 cm), se podrá escalar la imagen de manera que el tamaño de la cara sea de 40 píxeles por lado o mayor.

[1]Datos obtenidos con una rotación fija en el eje x de 90º

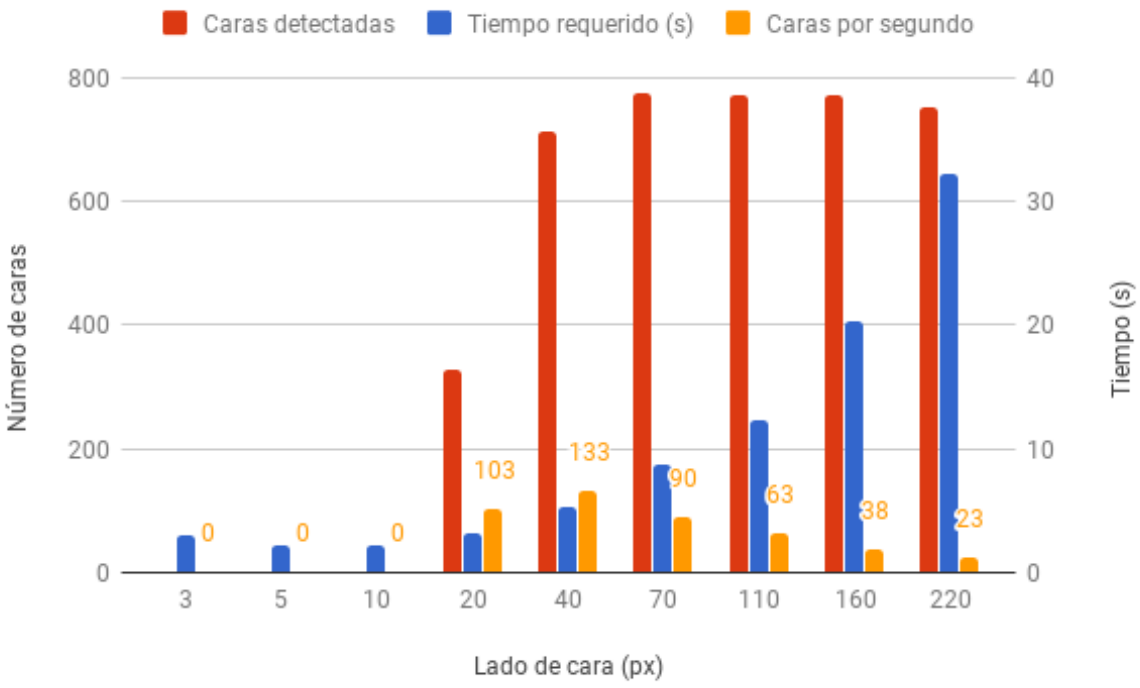

Figura 7.1: Capacidad de detección de caras según el tamaño de estas.

7.3. Estimación de la pose

La estimación de la pose es el objetivo principal de este trabajo. La estimación se ejecuta durante el procedimiento quirúrgico y requiere un procesado de la información a tiempo real. En esta sección se estudia la fidelidad de la estimación y la capacidad de procesamiento de información del programa durante el procedimiento quirúrgico. En caso de desaparecer la visión directa con la cara, el programa debe seguir buscando caras y adaptarse lo antes posible en caso de detectarse.

7.3.1. Composición de cara óptima

Como se comenta en el capítulo de implementación, la composición de la cara afecta en gran medida el proceso de estimación de la pose. Los elementos de la cara se dividen en ocho zonas, cada una formada por un conjunto de puntos que podrán aparecer o no en la solución final. En un principio, la intuición indica que a más elementos, mejor estimación,

pero esto no se cumple en la realidad.

Para representar cada una de las caras, se le asigna un número entre 1 y 255. Este, pasado a binario, representa por cada posición que elemento se incluye. En caso de contener un 1 en la posición del elemento se incluye en el conjunto, y con un 0 se excluye. Esta relación se observa en la figura 7.2.

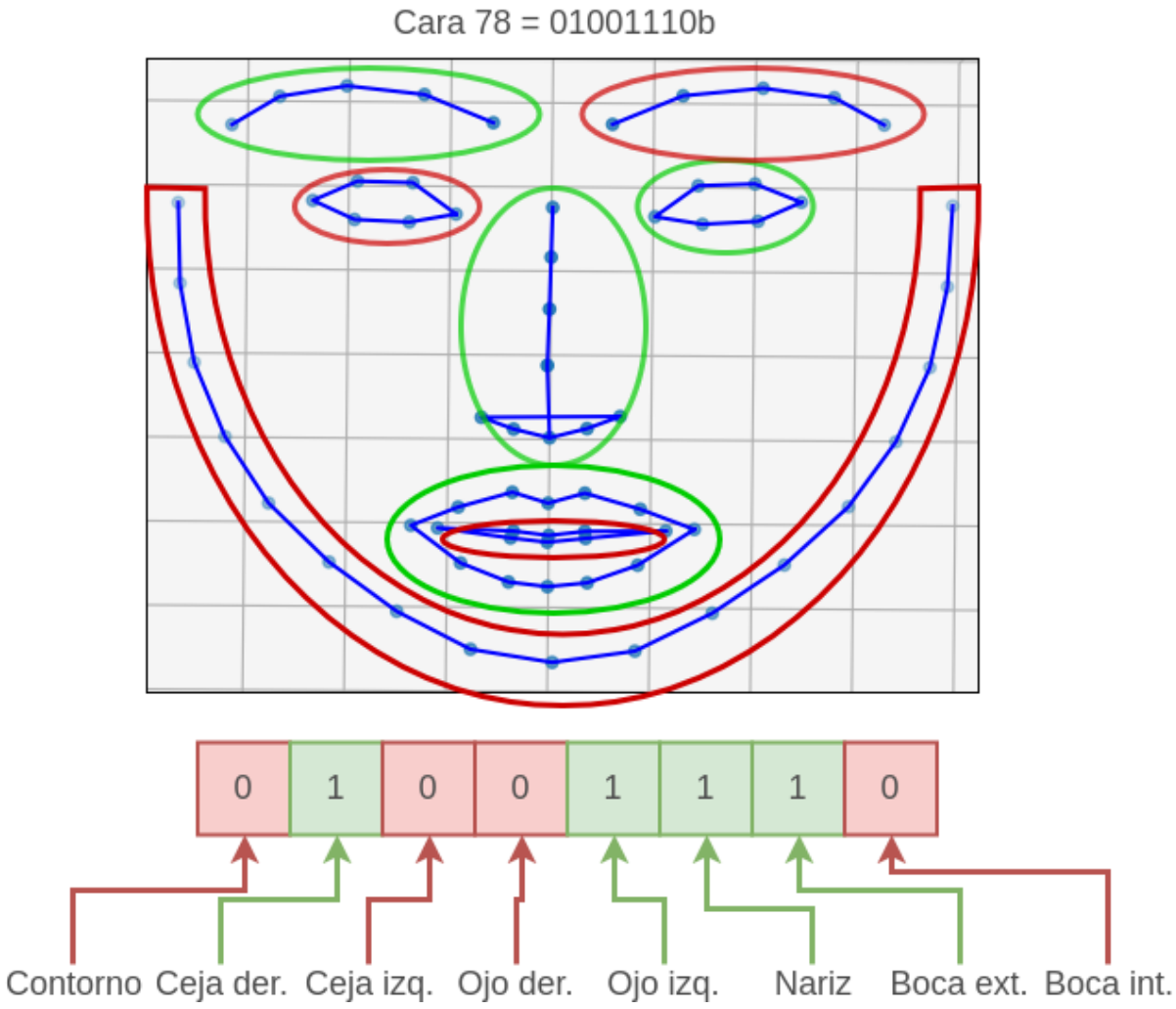

Figura 7.2: Relación entre el número de la cara y su composición.

Para encontrar la mejor cara, se le asigna una valoración a cada una dependiendo de su error en las estimaciones. Se estiman las imágenes del entorno virtual tanto las rotadas sobre el eje X como el eje Y. Comparando los resultados con los valores reales se obtienen

los errores de rotación y traslación por cada imagen.

La mejor cara será aquella que obtenga el error mínimo, dando prioridad a los ángulos más cercanos a 0, y menos a aquellos más alejados. Para esto, se crea una función rampa con pendiente positiva al acercarse al 0 y negativa al alejarse como se observa en la gráfica de la figura 7.3.

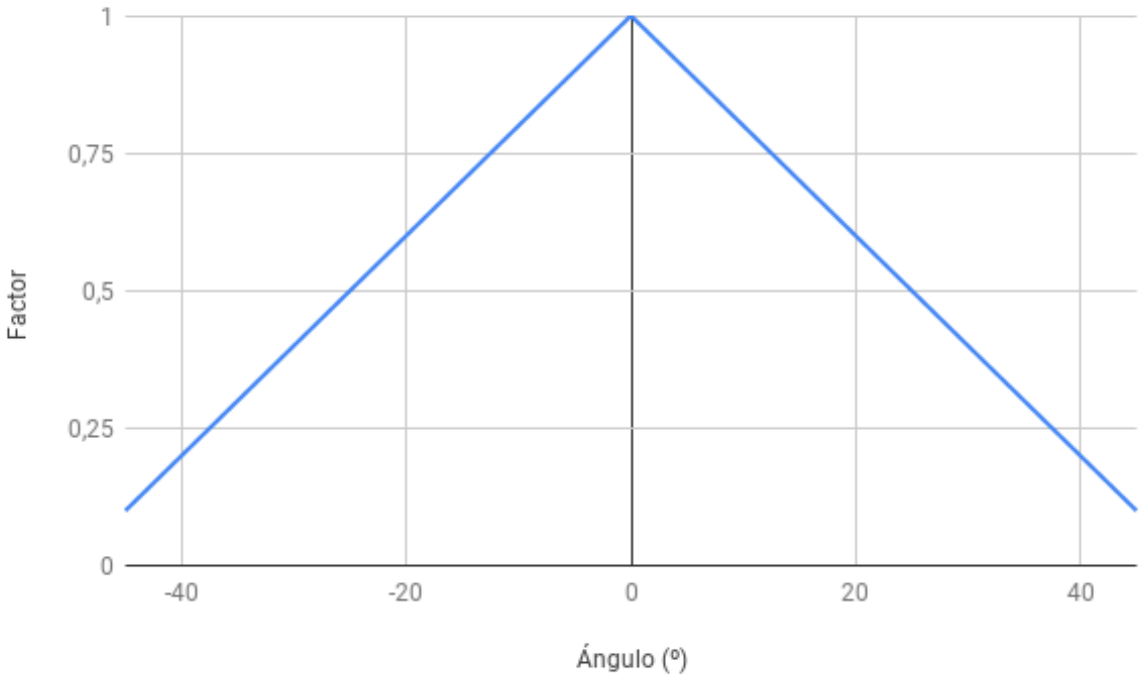

Figura 7.3: Factor aplicado a cada error según su ángulo

De esta manera, el valor del error disminuirá linealmente a más alejado esté del centro. Se observa que la función corta al eje horizontal a los 50ª, comprendiendo este el mayor rango en los límites de movimiento. Se limitan los rangos de ángulos a los requeridos por la cirugía, es decir, se tomarán las imagenes rotadas sobre X desde los -45º a los 15º, y las rotadas sobre Y de -30º a 30º.

Una vez calculados todos los errores y obteniendo su media por cada cara, se ponderan

los errores de rotación y traslación, aportando cada uno el 50 % del resultado final. Los resultados se observan en la gráfica de la figura 7.4.

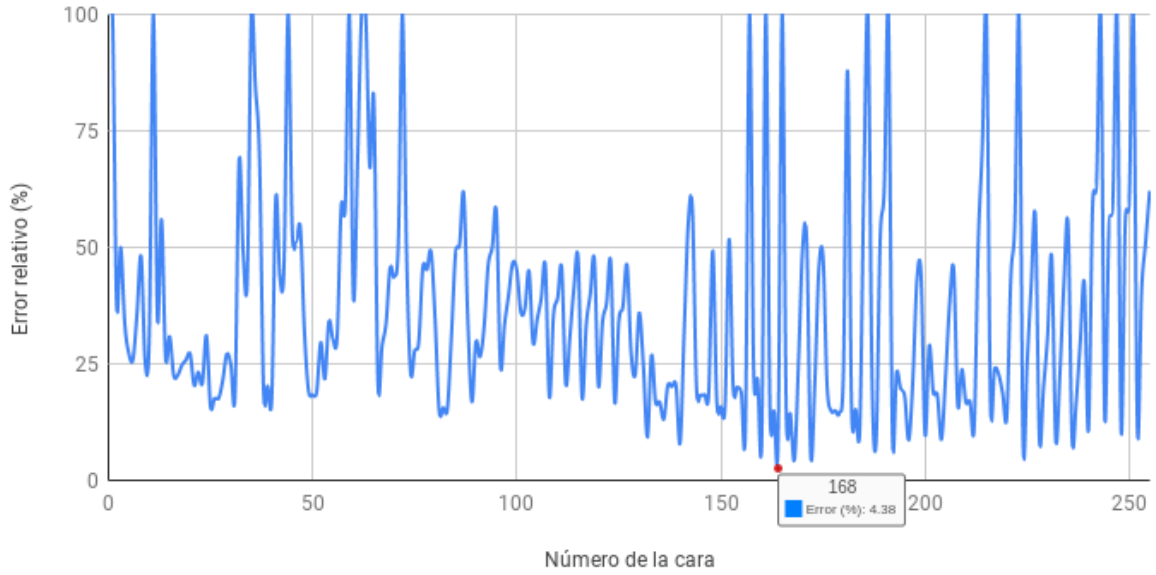

Figura 7.4: Evaluación por cada composición de cara.

Los resultados se exponen en porcentaje de error relativo, ya que se toma el máximo de los errores como un 100 %, y el resto de de las caras representan el porcentaje de mejora respecto a este. El resultado muestra como la cara 168 es la que ofrece un menor error. Su composición consta del contorno de la cara, la ceja izquierda y el ojo derecho.

7.3.2. Tamaño de la cara

Como se concluye en la sección de la detección de caras, el tamaño de cara óptimo es de 40 píxeles por lado. Caras menores a este valor no son detectadas con tanta precisión, y mayores no ofrecen una mejor detección pero si incrementan el tiempo de procesado. El problema aparece en la estimación de la pose, cuando este tamaño no es óptimo en cuanto a la precisión a la estimación.

Este error en la estimación aparece al calcular las posiciones de los puntos faciales sobre una cara muy pequeña, cuya detección no es del todo precisa, aumentando este error al escalarse la imagen a su tamaño original para la posterior estimación. Un ejemplo de estos errores se observan en la figura 7.5, donde los puntos azules representan los puntos originales y los rojos los obtenidos tras escalar la imagen. Por lo tanto hace falta encontrar el tamaño de cara, mayor a 40 píxeles por lado, que optimice la estimación de la pose.

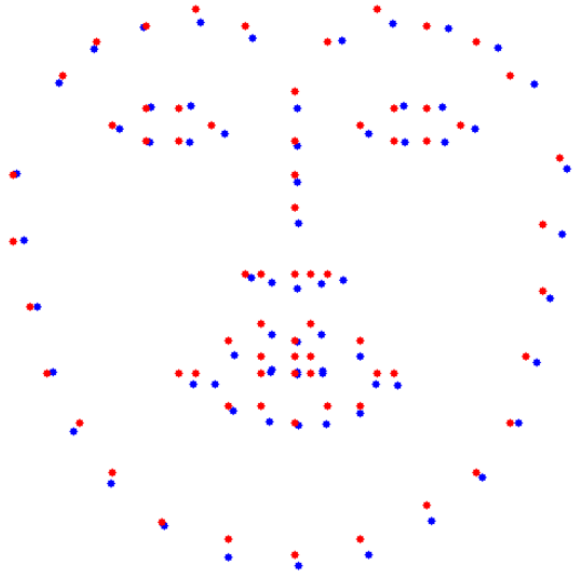

Figura 7.5: Errores producidos al localizar los puntos faciales en una imagen escalada.

Para calcular el tamaño óptimo, se obtiene un error por cada tamaño, desde los 40 píxeles hasta el tamaño real de la cara que se ha limitado a 545 píxeles. Por cada tamaño se obtiene el error medio en rotación por cada eje. Los resultados se observan en la gráfica de la figura 7.6.

Como se observa, los resultados aparecen como errores relativos, cuya razón ya se ha explicado en el apartado anterior. El error mínimo lo ofrece el tamaño de cara de 105 píxeles por lado, con error del 13 % respecto al máximo.

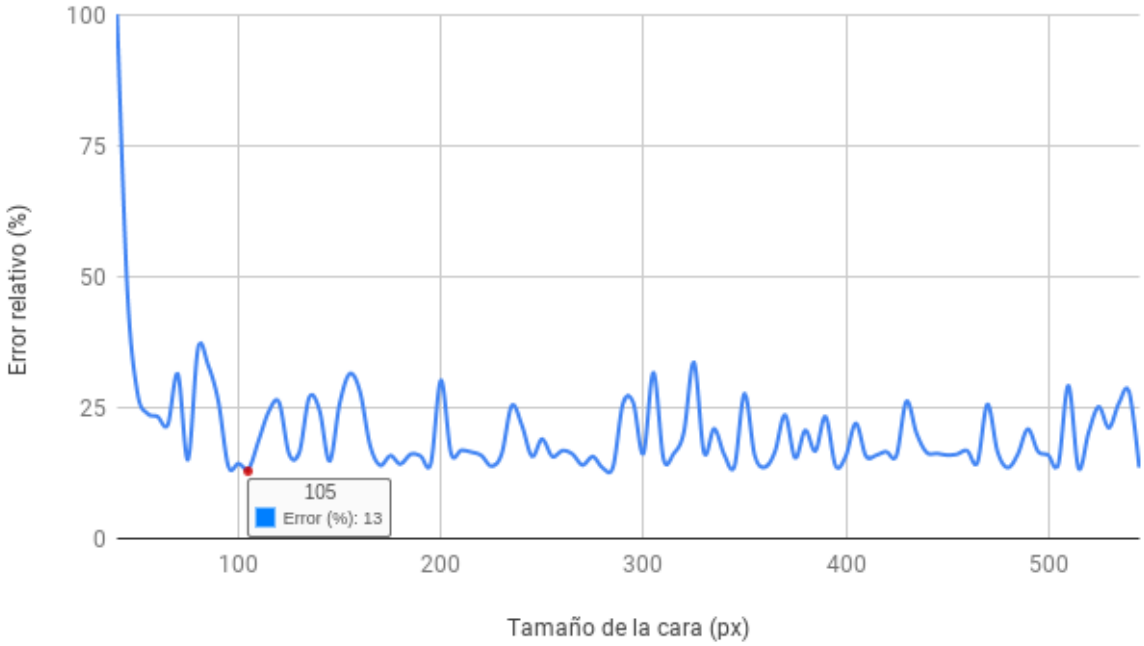

Figura 7.6: Evaluación por cada tamaño de cara.

En realidad, se han obtenido otros dos valores con el mismo valor del 13 %, correspondientes con lados de cara de 280 píxeles y 515 píxeles. Se selecciona el 105 al ser el menor de ellos, ofreciendo una velocidad de ejecución mayor.

7.3.3. Velocidad de la estimación

Para calcular la potencia del algoritmo, se realizará una toma de imágenes sobre un entorno real. En este caso no se hace uso del entorno virtual ya que, aunque se grabara un vídeo, los resultados no serían un reflejo fiel a la realidad, además de no requerir un control total para este paso.

Se hará uso del concepto fotogramas por segundo (FPS) para calcular la velocidad media a la que se ejecuta el programa. Para obtener unos resultados más representativos, se calculan los FPS de cada imagen, es decir, la velocidad instantánea a la que funciona el algoritmo en esa imagen. De esta forma si la cámara tarda más de lo general en procesar

un fotograma, no afectará al experimento en su totalidad.

Para comprobar la eficacia del resultado del experimento anterior, se toman los 10 tamaños de cara que mejor resultado ofrecen y se calcula la media de los FPS al ejecutar el programa sobre 100 imágenes. Los resultados se observan en la figura 7.7.

Figura 7.7: Velocidad de procesamiento según el tamaño de la cara

Se observa como el tamaño de cara de 105 píxeles obtiene una velocidad de procesamiento de 52,306 fotogramas por segundo, y como disminuye de forma exponencial esta velocidad al aumentar el tamaño de las caras.

Al establecer un tamaño de cara fijo se consigue que, independientemente de la resolución de la cámara, el sistema de una respuesta a tiempo real. Por lo tanto se toman 100 imágenes a diferentes escalas, calculando en cada una de ellas su FPS. Con esto, se comprueba el efecto del aumento del número de píxeles con la disminución de la velocidad del sistema como se observa en la tabla 7.3.

Altura (px)	**Anchura (px)**	**Píxeles**	**Píxeles (%)**	**FPS**	**FPS (%)**
480	640	307200	100	57,328	100
960	1280	1228800	400	50,129	87
1440	1920	2764800	900	45,171	79
1920	2560	4915200	1600	41,951	73
2400	3200	7680000	2500	37,946	66

Tabla 7.3: Efecto del tamaño de imagen sobre la velocidad de procesado.

En la tabla se toma como referencia la velocidad de la resolución 480x640 píxeles. Se observa como la velocidad de procesado disminuye muy lentamente comparado con el aumento de píxeles. En la resolución máxima de 2400x3200 píxeles, con 25 veces más píxeles que la referencia, el sistema sigue ofreciendo información a tiempo real, no llegando ni si quiera a disminuir a la mitad de la velocidad.

Los datos de velocidad de procesamiento se han medido eliminando cualquier tipo de salida que no sean los datos obtenidos de la estimación de la pose. En el caso de la interfaz, los procesos como mostrar las imágenes a tiempo real o la representación tridimensional del endoscopio respecto a la cara requieren de un procesado que afecta en gran medida al proceso. En la tabla 7.4 se observa el cambio de velocidad dependiendo de que muestre la interfaz, correspondiendo cada número entre paréntesis a los siguientes datos:

1. Calculo de datos de la estimación de la pose. Rotaciones y traslaciones.
2. Visión actual de la cámara
3. Detección de cara y de puntos faciales
4. Representación tridimensional
5. Mostrar los datos por pantalla

Se observa como la representación tridimensional de la cara y el endoscopio perjudica la velocidad de ejecución considerablemente. Comparando con la velocidad de solo el cálculo, se produce una disminución del 91,4 %.

Elementos	1	1, 2	1, 2, 3	1, 4	1, 5
FPS	52,3	35,65	29,03	4,5	42,08

Tabla 7.4: Velocidad de ejecución según información representada

7.3.4. Precisión de la estimación

Una vez obtenidos todos los valores necesarios para una correcta estimación de la pose, se capturan los resultados para el posterior análisis de su precisión. Para esto es fundamental el uso de las fotos obtenidas de la cara tridimensional en Unity3D.

Se toman las orientaciones de la cámara en cada imagen como el resultado de los experimentos, y su combinación con la longitud del endoscopio para definir la posición de la punta.

Como se ha expuesto anteriormente, los datos de rotación sobre el eje Z no se consideran al no ser útil su información. Por lo tanto, los valores a tener en cuenta para conocer la precisión de la estimación son la rotación sobre el eje X, la rotación sobre el eje Y, y la distancia euclidiana entre la posición de la cámara y el origen de coordenadas.

Para calcular tanto el error de rotación como de traslación, se usa el error absoluto como indicador de la calidad de la estimación. El valor absoluto $\triangle x$ es el resultado de la relación entre el valor medido x_0 y el valor real x, como aparece en la ecuación 7.1.

$$\triangle x = x_0 - x \tag{7.1}$$

Como se observa, el error absoluto puede ser tanto positivo como negativo. Una vez tenemos el error absoluto, se calcula la desviación media a partir de este. La desviación media otorga un rango de precisión para cada medición tomada, y se representa por D_m, definida en la ecuación 7.2. En esta ecuación, la N representa el número de mediciones totales, una por ángulo.

$$D_m = \frac{\sum_{i=0}^{N} \mid \triangle x_i \mid}{N} \tag{7.2}$$

Se calcula una desviación media por cada eje de rotación y la distancia calculada. Tomando una medición m y una desviación absoluta d_m, se representa su valor r como aparece en la ecuación 7.3. Es decir, el valor r se encuentra contenido en el rango de la inecuación 7.4.

$$r = m \pm d_m \tag{7.3}$$

$$m - d_m \leq r \leq m + d_m \tag{7.4}$$

Tomando imágenes rotadas sobre el eje X, se obtienen los resultados de la estimación de las rotaciones de la tabla 7.5. De estas mismas imágenes se obtienen los resultados de las distancias a la que se encuentra la cámara de la tabla 7.6. Los mismos casos aplicados a las imágenes rotadas sobre el eje Y se aprecian respectivamente en la tabla 7.7 y tabla 7.8.

Con estos datos, se obtienen los resultados respecto a la orientación de la cámara. El objetivo real de este trabajo es la orientación de la punta del endoscopio, que se deriva directamente de la de la cámara. En esta caso, comparten las mismas rotaciones, con sus respectivos errores, pero las traslaciones varían al ubicarse a una distancia fija (la longitud del endoscopio) a través del eje Z.

En las tablas de rotaciones, se observan filas coloreadas de rojo y naranja. En la tabla de rotaciones sobre el eje X, se observa como para ángulos menores a -30º la estimación llega a su límite de rango al detectar estos con valores cercanos a -30ª. Por lo tanto, estos valores no se han tenido en cuenta para el cálculo de errores, pero si se tomarán en cuenta en los rangos de movimiento del instrumento.

En el caso de la fila roja, se ha realizado una detección incorrecta, por lo tanto estos datos también se descartan. En esta situación, se considera que la estimación debería ser parecida a la de su ángulo positivo al ser la cara simétrica. Por esto para el cálculo de

errores se ha tomado en cuenta el valor del ángulo positivo por partida doble.

A partir de estas cuatro tablas calcula la desviación media final para cada eje de rotación y la distancia haciendo la media entre sus valores. Por lo tanto se obtiene una desviación media sobre el eje X de **2,38º**, sobre el eje Y de **0,78º** y en distancia de **0,78 cm**.

Los errores de rotación de la cámara, unido al error en la distancia, pueden provocar una acumulación de errores al establecer la pose de la punta del endoscopio. Los errores en rotación provocan un mayor desplazamiento de la punta con un aumento de la distancia entre esta y la cámara. Por lo tanto es necesario calcular los errores de la ubicación de la punta.

Se calculan las traslaciones de la punta teniendo en cuenta el eje Z, no incluyendo los datos de las columnas coloreadas y teniendo en cuenta que el error absoluto en este caso coincide con la distancia esperada al estar apuntando al origen. Los datos obtenidos de las imágenes rotadas sobre el eje X se aprecian en la tabla 7.9, y los rotados sobre el eje Y en la tabla 7.10.

Se observa como la desviación media de traslación sobre el eje X es de **0,141 cm**, el del eje Y de **0,124 cm**, y el del eje Z de **0,711 cm**.

7.3.5. Velocidad de recuperación

La velocidad de recuperación del sistema tras la pérdida de visión de la cara es inmediata. Al no haber ningún tipo de seguimiento facial, el programa comprueba en cada iteración si se ha localizado una cara, volviendo al flujo normal en caso positivo.

Ángulos esperados (º)		**Ángulos obtenidos (º)**		**Error absoluto (º)**		**|Error absoluto| (º)**	
Eje x	Eje y	Eje x	Eje y	Eje x	Eje y	Eje x	Eje y
-45,00	0,00	-29,15	-1,25	15,85	-1,25	15,85	1,25
-40,00	0,00	-29,26	-0,39	10,74	-0,39	10,74	0,39
-35,00	0,00	-29,47	-0,43	5,53	-0,43	5,53	0,43
-30,00	0,00	-27,64	0,92	2,36	0,92	2,36	0,92
-25,00	0,00	-21,61	0,01	3,39	0,01	3,39	0,01
-20,00	0,00	-15,04	0,61	4,96	0,61	4,96	0,61
-15,00	0,00	-12,17	-0,98	2,83	-0,98	2,83	0,98
-10,00	0,00	-7,14	-0,42	2,86	-0,42	2,86	0,42
-5,00	0,00	-4,77	0,01	0,23	0,01	0,23	0,01
0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
5,00	0,00	4,45	-0,65	-0,55	-0,65	0,55	0,65
10,00	0,00	8,44	1,37	-1,56	1,37	1,56	1,37
15,00	0,00	11,44	-0,56	-3,56	-0,56	3,56	0,56
				Desviación media (º)		2,23	0,55

Tabla 7.5: Errores de rotación en las orientaciones obtenidas al rotar sobre el eje X

Ángulo (º)	**Distancia esperada (cm)**	**Distancia obtenida (cm)**	**Error absoluto (cm)**	**|Error absoluto| (cm)**
-45,00	30	29,81	-0,19	0,19
-40,00	30	29,11	-0,89	0,89
-35,00	30	29,05	-0,95	0,95
-30,00	30	28,45	-1,55	1,55
-25,00	30	28,39	-1,61	1,61
-20,00	30	29,07	-0,93	0,93
-15,00	30	29,14	-0,86	0,86
-10,00	30	29,00	-1,00	1,00
-5,00	30	29,51	-0,49	0,49
0,00	30	30,00	0,00	0,00
5,00	30	30,05	0,05	0,05
10,00	30	30,38	0,38	0,38
15,00	30	30,81	0,81	0,81
			Desviación media (cm)	0,75

Tabla 7.6: Errores en distancia de las orientaciones obtenidas al rotar sobre el eje X

Ángulos esperados (º)		**Ángulos obtenidos (º)**		**Error absoluto (º)**		**|Error absoluto| (º)**	
Eje x	Eje y	Eje x	Eje y	Eje x	Eje y	Eje x	Eje y
0,00	30,00	5,36	29,69	5,36	-0,31	5,36	0,31
0,00	25,00	4,04	25,23	4,04	0,23	4,04	0,23
0,00	20,00	5,37	21,26	5,37	1,26	5,37	1,26
0,00	15,00	2,22	17,39	2,22	2,39	2,22	2,39
0,00	10,00	1,76	11,10	1,76	1,10	1,76	1,10
0,00	5,00	-0,97	5,59	-0,97	0,59	0,97	0,59
0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
0,00	-5,00	-1,56	-5,64	-1,56	-0,64	1,56	0,64
0,00	-10,00	0,48	-11,00	0,48	-1,00	0,48	1,00
0,00	-15,00	-0,21	-16,65	-0,21	-1,65	0,21	1,65
0,00	-20,00	1,89	-21,81	1,89	-1,81	1,89	1,81
0,00	-25,00	3,60	-26,71	3,60	-1,71	3,60	1,71
0,00	-30,00	-30,81	5,37	-30,81	35,37	30,81	35,37
				Desviación media (º)		2,52	1,00

Tabla 7.7: Errores de rotación en las orientaciones obtenidas al rotar sobre el eje Y

Ángulo (º)	**Distancia esperada (cm)**	**Distancia obtenida (cm)**	**Error absoluto (cm)**	**|Error absoluto| (cm)**
30,00	30	28,82	-1,18	1,18
25,00	30	29,02	-0,98	0,98
20,00	30	28,95	-1,05	1,05
15,00	30	28,90	-1,10	1,10
10,00	30	29,49	-0,51	0,51
5,00	30	29,96	-0,04	0,04
0,00	30	30,00	0,00	0,00
-5,00	30	29,89	-0,11	0,11
-10,00	30	29,60	-0,40	0,40
-15,00	30	29,18	-0,82	0,82
-20,00	30	28,86	-1,14	1,14
-25,00	30	28,36	-1,64	1,64
-30,00	30	31,52	1,52	1,52
			Desviación media (cm)	0,81

Tabla 7.8: Errores en distancia de las orientaciones obtenidas al rotar sobre el eje Y

Ángulo (º)	**Posición esperada (cm)**			**Posición obtenida (cm)**			**| Error absoluto | (cm)**		
	Eje x	Eje y	Eje z	Eje x	Eje y	Eje z	Eje x	Eje y	Eje z
-45,00	0,000	0,000	0,000	-0,005	0,090	0,162	0,005	0,090	0,162
-40,00	0,000	0,000	0,000	-0,008	0,435	0,776	0,008	0,435	0,776
-35,00	0,000	0,000	0,000	-0,005	0,467	0,826	0,005	0,467	0,826
-30,00	0,000	0,000	0,000	0,024	0,719	1,373	0,024	0,719	1,373
-25,00	0,000	0,000	0,000	-0,003	0,594	1,500	0,003	0,594	1,500
-20,00	0,000	0,000	0,000	0,010	0,242	0,901	0,010	0,242	0,901
-15,00	0,000	0,000	0,000	-0,016	0,181	0,842	0,016	0,181	0,842
-10,00	0,000	0,000	0,000	-0,008	0,124	0,992	0,008	0,124	0,992
-5,00	0,000	0,000	0,000	0,000	0,041	0,489	0,000	0,041	0,489
0,00	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
5,00	0,000	0,000	0,000	0,000	0,004	-0,054	0,000	0,004	0,054
10,00	0,000	0,000	0,000	-0,009	0,056	-0,375	0,009	0,056	0,375
15,00	0,000	0,000	0,000	0,008	0,161	-0,796	0,008	0,161	0,796
				Desviación media (cm)			0,008	0,212	0,732

Tabla 7.9: Errores en las traslaciones de la punta obtenidas al rotar sobre el eje X

Ángulo (º)	**Posición esperada (cm)**			**Posición obtenida (cm)**			**| Error absoluto | (cm)**		
	Eje x	Eje y	Eje z	Eje x	Eje y	Eje z	Eje x	Eje y	Eje z
30,00	0,000	0,000	0,000	0,585	-0,094	1,020	0,585	0,094	1,020
25,00	0,000	0,000	0,000	0,419	-0,061	0,885	0,419	0,061	0,885
20,00	0,000	0,000	0,000	0,382	-0,087	0,973	0,382	0,087	0,973
15,00	0,000	0,000	0,000	0,327	-0,040	1,045	0,327	0,040	1,045
10,00	0,000	0,000	0,000	0,098	-0,015	0,499	0,098	0,015	0,499
5,00	0,000	0,000	0,000	0,004	0,001	0,042	0,004	0,001	0,042
0,00	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
-5,00	0,000	0,000	0,000	-0,011	0,003	0,112	0,011	0,003	0,112
-10,00	0,000	0,000	0,000	-0,076	-0,003	0,392	0,076	0,003	0,392
-15,00	0,000	0,000	0,000	-0,235	0,004	0,788	0,235	0,004	0,788
-20,00	0,000	0,000	0,000	-0,423	-0,033	1,055	0,423	0,033	1,055
-25,00	0,000	0,000	0,000	-0,738	-0,093	1,463	0,738	0,093	1,463
-30,00	0,000	0,000	0,000	0,519	-0,107	-1,423	0,519	0,107	1,423
				Desviación media (cm)			0,275	0,036	0,689

Tabla 7.10: Errores en las traslaciones de la punta obtenidas al rotar sobre el eje Y

Capítulo 8

Conclusiones y Trabajo futuro

Tras finalizar el desarrollo del programa y el estudio de los resultados, en este capítulo se analiza el proceso de creación y analisis del trabajo. Se exponen las conclusiones y los posibles trabajos futuros que presenta este trabajo.

8.1. Conclusiones

El objetivo principal de este trabajo es la creación de un sistema capaz de ubicar la punta del endoscopio en el interior de las cavidades nasales del paciente. Gracias a esto, el facultativo tiene una referencia espacial, además de la visual que ofrece la imagen del endoscopio.

Comparado con sus alternativas, este prototipo libera mucho espacio en el quirófano, del que otras máquinas ocupan una cantidad considerable. Además del espacio, estas máquinas requieren visión directa, y el facultativo suele tapar esta visión, haciendo el trabajo de este más estresante y dificultoso. Otra gran ventaja es la eliminación de cualquier tipo de contacto directo con el paciente, eliminando marcas fiduciarias o elementos que requieran un aumento del tiempo del preoperatorio.

Al estar situada la cámara en el endoscopio, el facultativo no pierde ninguna de sus cualidades y no requiere ningún tipo de aprendizaje al realizar una modificación mínima sobre la instrumentación para la que está formado.

Como desventaja, pierde precisión sobre las alternativas disponibles, pero en este caso no es un factor decisivo ya que el endoscopio es un instrumento de apoyo, de visualización, por lo que la precisión de su orientación no es vital, y una buena aproximación, dentro de unos límites, es más que suficiente. El uso de sus alternativas para este mismo propósito no está justificado, al requerir un precio y espacio elevados y no necesitar tal precisión.

Gracias al uso de librerías de visión por computador como OpenCV o Dlib, el primer contacto con estas técnicas es a primera vista sencillo. Su facilidad de uso, unido con la rápida implementación que permite Python, permite alcanzar una gran cantidad de objetivos en un corto tiempo de plazo. Pero, tras la detección de puntos faciales, se observa como los algoritmos de reconstrucción y estimación de la pose son demasiado sensibles a pequeñas variaciones, requieriendo un formato especial para su uso y poca documentación en algunos casos. Por lo tanto se requiere iterar sobre cada uno de los pasos anteriores, mostrando una complejidad y problemática mayor en cada uno.

Llega un momento del proceso en el que se observa como los resultados obtenidos por la estimación de la pose siguen siendo en su mayoría incorrectos, y un análisis exhaustivo de los pasos anteriores muestran como todos los pasos se han realizado correctamente. Esta situación lleva a comprender la poca estabilidad que ofrece el algoritmo de estimación, que devuelve datos muy distintos para valores cercanos. En este momento se centran los esfuerzos en recuperar toda la información correcta posible y filtrar aquella cuyos valores son incorrectos.

Alguno de los parámetros que afectan a la estabilidad del proceso son la detección de caras incorrectas, la técnica de toma de imágenes para la reconstrucción, la incapacidad del algoritmo de estimación para detectar si se encuentra detrás de la cara, la composición de la cara, la incapacidad de discernir entre traslación con rotación y una rotación pura, entre otras. Una vez identificados, se realizan experimentos y análisis que permiten una configuración estable.

Los resultados de este procedimiento son fructíferos, la estabilidad del sistema es total, y la precisión, rango de movimiento y velocidad son más que suficientes para realizar un proceso quirúrgico cómodo, únicamente obteniendo beneficios en el uso de este sistema.

Con todo esto, se ofrece un sistema que permite la obtención de datos fiables y a tiempo real de la estimación de la pose de un endoscopio rígido usado en cirugía endoscópica. Ademas, la sencilla interfaz permite hacer uso de esta tecnología sin requerir ningún tipo de conocimiento en programación.

Entrando en detalle en los resultados obtenidos de los experimentos, el sistema es capaz de ejecutar la estimación de la pose a más de 50 imágenes por segundo. Se considera tiempo real aquella ejecución a más de 30 imágenes por segundo, por lo tanto este resultado es sobresaliente. Al mantener un tamaño de cara fijo, es capaz de transmitir información a tiempo real aun aumentando considerablemente la resolución de las imágenes. Se observa como la precisión en las rotaciones es mejor sobre el eje Y que sobre el X, deduciendo que esta diferencia proviene de la generación de la reconstrucción tridimensional de la cara del paciente a partir de imágenes rotadas sobre el eje Y. De todas formas, con una 0,78º para el eje Y y 2,38º para el eje X, las medidas de las rotaciones se consideran correctas.

En cuanto a las distancias, el objetivo se ha logrado al mantener el error por debajo del centímetro, alcanzando una baja cifra de 0,78 cm. Al estimar la pose de la punta del endoscopio, los errores en las traslaciones sobre los ejes X e Y son los deseados, con 0,141 y 0,124 centímetros respectivamente de media. En cuanto al eje Z, con 0,711 cm de error, es el mayor causante del error en la distancia, y, aun siendo un valor aceptable, tiene margen de mejora. Este valor es mayor al tener unas medidas menos precisas de la profundidad del modelo reconstruido de la cara.

Por último, los rangos de movimiento son correctos, ya que cumplen con la mayoría de las situaciones, siendo de -30ª a 30º sobre el eje Y, y de -30º a 15ª sobre el eje X. El límite superior del eje X es mayor, pero se ha limitado al no ser habitual en este tipo de cirugía, consiguiendo unos resultados más ajustados a casos reales. La única limitación

que presenta el eje Z es en el caso de la detección de caras, variando de -20º a 20º aproximadamente, siendo este un rango más que asumible y habitual para el facultativo.

Para finalizar, como crecimiento personal, el desarrollo de este trabajo me ha aportado una gran cantidad de conocimientos muy útiles en mi formación como ingeniero de la salud. He obtenido conocimientos tanto del mundo médico, como el de visión por computador, aprendizaje automático e informática. Además, este proceso de investigación, desarrollo e innovación, con su necesidad de formación continua, los problemas encontrados y la búsqueda de resultados cada vez mejores, han aportado aptitudes como la constancia, esfuerzo y mejora en la toma de decisiones.

8.2. Trabajo futuro

Este trabajo está dirigido a crear un sistema que sirva como base de futuras mejoras. Por lo tanto, los usos futuros son una parte fundamental de este trabajo, y muchas de sus decisiones han sido tomadas con estos en mente. A continuación se describen las principales mejoras y direcciones futuras a tomar a partir de este trabajo:

- Implementación en un sistema empotrado, portátil y con gran capacidad de procesamiento o de transmisión de información. Su objetivo es la creación de un dispositivo con hardware específico, que permita contener tanto la cámara, el dispositivo de procesamiento, batería y acople al endoscopio en un mismo elemento.

- Mejora en la medida de profundidad del modelo facial reconstruido usando nuevas técnicas de reconstrucción o cámaras de visión tridimensional.

- Relacionar la información obtenida con imágenes médicas del paciente sobre las que representar la orientación del endoscopio. Capturando imágenes biomédicas tridimensionales del paciente, se puede relacionar el modelo tridimensional reconstruido mediante los puntos de referencia faciales. Una vez relacionados, es posible mostrar la orientación del endoscopio en un entorno tridimensional que muestre la anatomía

personalizada del paciente. Gracias a estas técnicas se pueden conseguir rutas de acceso a determinadas localizaciones y ubicación inmediata de elementos anatómicos o patológicos.

- Interfaz avanzada que permita aprovechar la velocidad de procesado del programa. Actualmente la sencilla interfaz disminuye la velocidad de procesamiento y es requerido el crear una interfaz destinada al uso profesional del programa.

- Mejora de la precisión en cuanto a la distancia de la orientación de la cámara. El margen de mejora para la distancia reconocida sobre el eje Z de la cámara es amplio, siendo ideales en el caso de ser menores a medio centímetro.

- Estudio de los movimientos para la toma de imágenes destinadas a la reconstrucción tridimensional. El objetivo es equiparar los errores obtenidos para las rotaciones sobre el eje X y sobre el eje Y.

- Uso de la realidad aumentada representar tanto la anatomía del paciente, la ubicación del endoscopio y su visión actual en un mismo campo de visión para el facultativo. De esta manera, el cirujano no debe apartar la vista sobre el paciente, ya que la realidad aumentada se encarga de mostrarle toda la información necesaria.

Apéndice A

Codigos de programacion

A.1. Interfaz gráfica

```
from tfg import *
import warnings
warnings.filterwarnings("ignore")
# Medidas de la interfaz
width = 80
heigth = 21
# Estados de la configuracion
ready = []
actual = "DISPOSITIVO"
remaining = ["RECONSTRUCCION",
    "CALIBRACION", "EJECUCION"]
# Funciones para la construccion
# de la interfaz
def repeat(elem, num):
    res = ""
    for i in range(0, num):
        res += elem
    return res
def spaces(num):
    return repeat(" ", num)
def dash(num):
    return repeat("-", num)
def line(content=None):
    if content is None: content =
        spaces(width - 2)
    print("|" + content + "|")
def lines(num):
    for i in range(0, num):
        line()
def middle(content):
    clen = len(content)
    free = width - 2 - clen
    ini = int(free / 2)
    end = width - 2 - ini - clen
    line(spaces(ini) + content +
        spaces(end))
def left(content):
    clen = len(content)
    free = width - 2 - clen
    line(content + spaces(free))
def separator():
    print(dash(width))
def semi_separator():
    middle(dash(int(width / 2)))
```

```
def portada():
    separator()
    line()
    middle("SISTEMA DE GUIADO PARA
        CIRUGIA NASAL ENDOSCOPICA")
    line()
    semi_separator()
    line()
    middle("Martin Axelrad Tinoco")
    middle("Universidad de Malaga")
    middle("2018")
    line()
def progress():
    global ready, actual, remaining
    res = ""
    for iready in ready:
        res += "[OK]" + iready + " "
    ini = len(res)
    res += actual
    num = len(res) - ini
    res += " "
    for iremaining in remaining:
        res += iremaining + " "
    end = len(res) - num - ini
    middle(res)
    middle(spaces(ini) + dash(num)
        + spaces(end))
def screen(content=[]):
    new_lines(30)
    portada()
    line()
    progress()
    lines(2)
    for icontent in content:
        middle(icontent)
    nlines = len(content)
    lines(heigth - 13 - nlines - 1)
    separator()
def invalid_value():
    print("-------------------------------------
    print("| ERROR: El valor
        introducido no es correcto.
        Por favor, vuelva a
        intentarlo |")
    print("-------------------------------------
def new_lines(num):
    for i in range(0, num):
        print("\n")
# Presentacion
screen()
# Conexion con dispositivo
content = []
valid = False
while not valid:
    channel = input("Seleccione el
        canal del dispositivo de
        grabacion (defecto = 0): ")
    try:
        channel = int(channel) if
            channel is not None and
            channel is not "" else 0
        camera =
            ImprovedCamera(channel)
    except:
        screen(content + ["Canal:
            [ERROR] No disponible"])
        continue
    valid = True
```

```
content.append("Canal: " +
    str(channel))
screen(content)
# Dispositivo de calibracion
valid = False
while not valid:
    device = input("Indique el
        nombre del dispositivo
        calibrado: ")
    try:
        mtx, dist =
            loadParams(device)
    except:
        screen(content +
            ["Calibracion: [ERROR]
            Parametros no
            econtrados"])
        continue
    valid = True
content.append("Calibracion: " +
    device)
screen(content)
input("Compruebe los parametros y
    pulse ENTER para continuar.")
ready.append(actual)
actual = remaining.pop(0)
# Configuracion y toma de imagenes
# para la reconstruccion
valid = False
rec_helper = ReconstructionHelper()
while not valid:
    content = []
    screen()
```

```
    try:
        rec_name =
            input("Introduzca el
            nombre de la
            reconstruccion o deje
            vacio para realizar una
            nueva: ")
        if rec_name is "":
            screen()
            rec_name =
                input("Introduzca un
                nombre para la
                reconstruccion: ")
            content.append("Nombre
                de reconstruccion: "
                + rec_name)
            screen(content + ["Toma
                de imagenes: En
                espera..."])
            print("Pulse ENTER para
                comenzar la toma de
                imagenes...")
            frames = []
            key = None
            started = False
            finished = False
            window =
                cv2.namedWindow('main',
                cv2.WINDOW_NORMAL)
            cv2.moveWindow('main',
                800, 0)
            cv2.resizeWindow('main',
                1120, 1080)
```

```
while not finished:
    frame = camera.read()
    if started:
        frames.append(frame)
        frame_dot =
            Drawer.point(frame,
            rectCenter(rectFrame(frame)),
            color=Drawer.RED)
    else:
        frame_dot =
            Drawer.point(frame,
            rectCenter(rectFrame(frame)))
    frame_dot.show(winname='main')
    key = cv2.waitKey(10)
    if key is 13:
        if not started:
            screen(content
                + ["Toma
                de
                imagenes:
                En
                proceso..."])
            print("Pulse
                ENTER
                para
                finalizar
                la toma
                de
                imagenes...")
            started = True
        else:
            cv2.destroyWindow('main')
            finished =
                True
content.append("Toma de
    imagenes: " +
    str(len(frames)) + "
    imagenes capturadas")
screen(content)
screen(content +
    ["Reconstruccion:
    Procesando..."])
reconstruction =
    rec_helper.reconstruct_from_frames
    rec_name, frames)
points3d =
    reconstruction[0]
points3d =
    Aligner(points3d).align()
plotter =
    plotter_face(points3d)
content.append("Reconstruccion:
    Finalizada")
screen(content)
print("Compruebe la
    reconstruccion y
    cierre la ventana
    para continuar.")
while not
    plotter.is_closed():
    plotter.draw()
plotter.close()
screen(content +
    ["Comprobacion:
    Esperando
    respuesta..."])
repetir =
    input("Reconstruccion
    correcta? (s/n): ")
```

```
            if repetir is 'n':
                continue
        reconstruction =
            rec_helper.load_results(rec_name)
        valid = True
    except:
        screen(["[ERROR]
            Reconstruccion fallida.
            Compruebe los datos de
            entrada."])
        input("Pulse ENTER para
            reintentar.")
points3d = reconstruction[0]
points3d =
    Aligner(points3d).align()
# Toma de la imagen de referencia
# la calibracion
ready.append(actual)
actual = remaining.pop(0)
valid = False
while not valid:
    content = []
    screen(content + ["Imagen de
        referencia: Capturando..."])
    print("Situe la camara en la
        posicion inicial deseada y
        pulse ENTER.")
    cv2.namedWindow('main',
        cv2.WINDOW_NORMAL)
    cv2.moveWindow('main', 800, 0)
    cv2.resizeWindow('main', 1120,
        1080)
    key = None
    ref_frame = None
    while ref_frame is None:
        frame = camera.read()
        frame_dot =
            Drawer.point(frame,
            rectCenter(rectFrame(frame)))
        frame_dot.show(winname='main')
        key = cv2.waitKey(1)
        if key is 13:
            ref_frame = frame
    content.append("Imagen de
        referencia: Capturada")
    screen(content +
        ["Confirmacion:
        Esperando..."])
    print("Compruebe la imagen y
        pulse ENTER.")
    ref_frame.show(wait=False,
        winname='main')
    cv2.destroyWindow('main')
    res = input("Imagen correcta?
        (s/n): ")
    if res is 'n': continue
    valid = True
content.append("Confirmacion:
    Confirmada")
screen(content)
# Se solicita la longitud del
    endoscopio
valid = False
while not valid:
    try:
        real_dist =
            int(input("Distancia
            del endoscopio en
            centimetros: "))
```

```
        valid = True
    except:
        screen(content + ["[ERROR]
            El valor introducido no
            es correcto"])
        input("Pulse ENTER para
            reintentar.")
content.append("Longitud
    endoscopio: " + str(real_dist)
    + " cm")
screen(content)
input("Compruebe los datos y pulse
    ENTER")
content = ["HA FINALIZADO LA
    CONFIGURACION"]
screen(content)
# Comienza el proceso de estimacion
# Se crea el poser con los datos
    introducidos
# y un tamanio de cara de 105
    pixeles
poser = Pose(points3d, mtx, dist,
    ref_frame=ref_frame,
    real_dist=real_dist,
    face_side=105)
ready.append(actual)
actual = remaining.pop(0)
# Funciones que representan la
    estimacion
def repr_position(origin):
    global real_dist
    vector =
        Transformer.norm(origin)
    vector = [real_dist * elem for
        elem in vector]
    point =
        [str("{0:.1f}".format(origin[i]
        - vector[i])) + " cm" for i
        in range(0, len(origin))]
    return "Posicion - Eje X: " +
        point[0] + " Eje Y: " +
        point[1] + " Eje Z: " +
        point[2]
def repr_rotation(rvec):
    res =
        [str("{0:.1f}".format(r2g(elem)))
        + "" for elem in rvec]
    return "Rotacion - Eje X: " +
        res[0] + " Eje Y: " +
        res[1] + " Eje Z: " + res[2]
# Carga de los elementos graficos
plotter = plotter_face(points3d,
    scale=15)
plotter.geometry(width=1120,
    height=900, posx=800, posy=0)
cv2.namedWindow("main",
    cv2.WINDOW_NORMAL)
cv2.moveWindow('main', 0, 479)
cv2.resizeWindow('main', 725, 460)
scale = [int(real_dist/5),
    int(real_dist/5), real_dist]
elem = None
fps = 0
fpst = 0
# Estimacion de la pose
while True:
    t0 = time()
    frame = camera.read()
    if elem is not None:
        plotter.remove(elem)
```

```
    retval, origin, axis, rvec,
        face_rect, landmarks =
        poser.run(frame)
    if retval:
        elem = plotter.axis(axis,
            origin, scale=scale)
        frame = Drawer.rect(frame,
            face_rect)
        frame =
            Drawer.points(frame,
            landmarks)
        content =
            [repr_position(origin),
            repr_rotation(rvec),
            "FPS: " + str(fps)]
        screen(content)
        fpst += fps
    frame.show(winname="main")
    plotter.draw()
    cv2.waitKey(1)
    fps = 1 / (time() - t0)
```

A.2. Clasificador

```
from collections import defaultdict
from enum import Enum
from tfg.image import *

class ClassifierType(Enum):
    """Tipos de clasificadores"""
    HAAR_FRONTAL_FACE_ALT = 1
    HAAR_EYE =              2
    HAAR_MCS_MOUTH =        3
    HAAR_MCS_NOSE =         4
    LBP_FRONTAL_FACE =      5
    LBP_PROFILE_FACE =      6

class ClassifierTypeDict(object):
    """Enlaza el tipo de
       clasificador con su fichero
       entrenado."""
    def __init__(self):
        self._dictionary = {
          ClassifierType.HAAR_FRONTAL_FACE_ALT:
'classifiers/haarcascade_frontalface_alt.xml',
          ClassifierType.HAAR_EYE:
'classifiers/haarcascade_eye.xml',
          ClassifierType.HAAR_MCS_MOUTH:
'classifiers/haarcascade_mcs_mouth.xml',
          ClassifierType.HAAR_MCS_NOSE:
'classifiers/haarcascade_mcs_nose.xml',
          ClassifierType.LBP_FRONTAL_FACE:
'classifiers/lbpcascade_frontalface.xml',
          ClassifierType.LBP_PROFILE_FACE:
'classifiers/lbpcascade_profileface.xml'
       }
       self._dictionary =
          defaultdict(lambda:
          'classifiers/haarcascade_frontalface_a
          self._dictionary)
    def getPath(self, type):
       res =
          self._dictionary.get(type)
       return res
    def getClassifier(self, type):
       res =
          cv2.CascadeClassifier(self.getPath(typ
       return res

class Classifier(object):
    """Detecta objetos especificos
       en una imagen."""
    def __init__(self, type =
       ClassifierType.HAAR_FRONTAL_FACE_ALT,
              scaleFactor = 1.1,
                 minNeighbors =
                 5,
              minSize = (0,0),
                 maxSize = None):
       self._classifier =
          ClassifierTypeDict().getClassifier(typ
       self._scaleFactor =
          scaleFactor
       self._minNeighbors =
          minNeighbors
       self._minSize = minSize
       self._maxSize = maxSize
       self._frame = None
```

```
    def getParams(self):
        """Devuelve arametros
           actuales para la
           clasificacion."""
        return (self._scaleFactor,
           self._minNeighbors,
           self._flags,
           self._minSize,
           self._maxSize)
    def setParams(self,
        scaleFactor, minNeighbors,
        flags, minSize, maxSize):
        """Configura los parametros
           de clasificacion."""
        self._scaleFactor =
           scaleFactor
        self._minNeighbors =
           minNeighbors
        self._flags = flags
        self._minSize = minSize
        self._maxSize = maxSize
    def setScaleFactor(self,
        scaleFactor):
        """Establece el factor de
           escala"""
        self._scaleFactor =
           scaleFactor
    def setMinNeighbors(self,
        minNeighbors):
        """Establece el vecindario
           minimo."""
        self._minNeighbors =
           minNeighbors
    def setMinSize(self, minSize):
        """Establece el tamanio
           minimo."""
        self._minSize = minSize
    def setMaxSize(self, maxSize):
        """Establece el tamanio
           maximo"""
        self._maxSize = maxSize
    def setFrame(self, frame):
        """Fija la imagen en la
           cual clasificar"""
        self._frame = frame
    def isEmpty(self):
        """Comprueba si el
           clasificador ha sido
           cargado."""
        return
           self._classifier.empty()
    def detect(self, frame = None):
        """Devuelve los rectangulos
           encontrados en la
           imagen."""
        if frame is None:
           frame = self._frame
        res =
           self._classifier.detectMultiScale(
           frame.mat,
           self._scaleFactor,
           self._minNeighbors,
           0,
           self._minSize,
           self._maxSize
        )
        return res
```

A.3. Detector

```
from tfg.classifier import *

class Detector(object):
    """Superclase encargada de la
       deteccion de elementos
       faciales."""
    def __init__(self, type =
       ClassifierType.HAAR_FRONTAL_FACE_ALT):
       self._classifier =
          Classifier(type)
       self._result = []
    def num(self):
       """Devuelve el numero de
          caras detectadas."""
       return len(self._objects)
    def get(self):
       """Devuelve los ultimos
          objetos detectados."""
       return self._result
    def detect(self, frame, rect =
       None):
       """Detecta un elemento en
          la imagen.
       Busca dentro de un
          rectangulo si se
          incluye."""
       if rect is not None:
          frame =
             frame.rectangle(rect)
       res =
          self._classifier.detect(frame)
       if rect is not None:
          x, y, _, _ = rect
          temp = []
          for sub in res:
             temp.append(addToRect(sub,
                dx = x, dy = y))
          res = temp
       else: res =
          arrays2rects(res)
       return res
    def best(self, rects, rect):
       """Devuelve el mejor objeto
          localizadp"""
       center = rectCenter(rect)
       best = None
       dist = None
       if len(rects) > 0:
          for rect in rects:
             ncenter =
                rectCenter(rect)
             ndist =
                distPoints(ncenter,
                center)
             if dist is None or
                ndist < dist:
                dist = ndist
                best = rect
       res = best
       return res
```

```
    def valid(self):
        """Comprueba si la
            deteccion es
            correcta."""
        return self._result is not
            None
    def setClassifier(self, type =
        ClassifierType.HAAR_FRONTAL_FACE_ALT):
        """Establece el
            clasificador."""
        self._classifier =
            Classifier(type)
    def getClassifier(self):
        """Devuelve el clasificador
            actual."""
        return self._classifier
    def getSection(self, rect):
        """Devuelve la region de
            busqueda como un
            rectangulo."""
        res = rectSection(rect)
        return res
    def draw(self, frame):
        """Dibuja los resultados
            como rectangulos."""
        res = Drawer.rect(frame,
            self.get(), Drawer.BLUE)
        return res
    def drawSection(self, frame,
        face_rect):
        """Dibuja la zona de
            busqueda como un
            rectangulo."""
        res = Drawer.rect(frame,
            self.getSection(face_rect),
            Drawer.GREEN)
        return res

class EyesDetector(Detector):
    """Detecta ambos ojos en una
        imagen."""
    def __init__(self):
        super().__init__(ClassifierType.HAAR_EYE)
        self._result = [None, None]
    def detect(self, frame,
        face_rect):
        """Detecta ambos ojos en
            una imagen"""
        eyel_section, eyer_section
            =
            self.getSection(face_rect)
        eyel_rects =
            super().detect(frame,
            eyel_section)
        eyer_rects =
            super().detect(frame,
            eyer_section)
        self.clear()
        if len(eyel_rects) > 0:
            self._result[0] =
            self.best(eyel_rects,
            eyel_section)
        if len(eyer_rects) > 0:
            self._result[1] =
            self.best(eyer_rects,
            eyer_section)
        res = self.get()
        return res
```

```
    def clear(self):
        """Elimina los
           resultados."""
        self._result = [None, None]
    def getLeftEye(self):
        """Devuelve el ojo
           izquierdo."""
        return self._result[0]
    def getRightEye(self):
        """Devuelve el ojo
           derecho."""
        return self._result[1]
    def leftEyeDetected(self):
        """Comprueba si el ojo
           izquierdo ha sido
           detectado."""
        res = self.getLeftEye() is
           not None
        return res
    def rightEyeDetected(self):
        """Comprueba si el ojo
           derecho ha sido
           detectado."""
        res = self.getRightEye() is
           not None
        return res
    def bothDetected(self):
        """Checks if the both eyes
           are detected."""
        res =
           self.leftEyeDetected()
           and
           self.rightEyeDetected()
        return res
    def draw(self, frame):
        """Dibuja los ojos en una
           imagen."""
        res = Drawer.rects(frame,
           self.get(), Drawer.BLUE)
        return res
    def getSection(self, face_rect,
        n = None):
        """Devuelve la zona de
           busqueda."""
        eyel_rect =
           rectSection(face_rect,
           ew = 0.5, eh = 0.6)
        eyer_rect =
           rectSection(face_rect,
           sw = 0.5, eh = 0.6)
        res = (eyel_rect, eyer_rect)
        if n is not None:
           res = res[n]
        return res
    def drawSection(self, frame,
        face_rect):
        """Dibuja la zona de
           busqueda"""
        res = Drawer.rects(frame,
           self.getSection(face_rect),
           Drawer.GREEN)
        return res

class NoseDetector(Detector):
    """Deteccion de narices en una
       imagen."""
    def __init__(self):
        super().__init__(ClassifierType.HAAR_MCS_
```

```
    def detect(self, frame,
        face_rect):
        """Detecta la nariz en una
            cara."""
        nose_rect =
            self.getSection(face_rect)
        nose_rects =
            super().detect(frame,
            nose_rect)
        if len(nose_rects) > 0:
            self._result =
            self.best(nose_rects,
            self.getSection(face_rect))
        res = self.get()
        return res
    def getSection(self, face_rect):
        """Devuelve la zona de
            busqueda."""
        res =
            rectSection(face_rect,
            sw = 0.3, ew = 0.7, sh
            = 0.3, eh = 0.8)
        return res

class MouthDetector(Detector):
    """Deteccion de bocas en una
        imagen."""
    def __init__(self):
        super().__init__(ClassifierType.HAAR_MCS_MOUTH)
    def detect(self, frame,
        face_rect):
        """Deteccion en bocas en
            una cara."""
        mouth_rect =
            self.getSection(face_rect)
        mouth_rects =
            super().detect(frame,
            mouth_rect)
        if len(mouth_rects) > 0:
            self._result =
            self.best(mouth_rects,
            self.getSection(face_rect))
        res = self.get()
        return res
    def getSection(self, face_rect):
        """Devuelve la zona de
            busqueda."""
        res =
            rectSection(face_rect,
            sh = 0.6)
        return res


class
    FrontalFaceDetector(Detector):
    """Detecta una cara y sus
        regiones."""
    def __init__(self, eyer =
        False, eyel = False, nose =
        False, mouth = False):
        super().__init__()
        self._eyer = eyer
        self._eyel = eyel
        self._nose = nose
        self._mouth = mouth
```

```
        self._result = {"face" :
            None, "eyer" : None,
            "eyel" : None, "nose" :
            None, "mouth" : None}
    def toDetectAll(self):
        self._eyer = True
        self._eyel = True
        self._nose = True
        self._mouth = True
    def detect(self, frame):
        """Detecta la cara y los
            elementos definidos."""
        self.clear()
        super_res =
            super().detect(frame)
        if len(super_res) > 0:
            face_rect =
                self.best(super_res,
                rectFrame(frame))
            self._result["face"] =
                face_rect
            if self._eyer or
                self._eyel:
                eyes = EyesDetector()
                eyes.detect(frame,
                    face_rect)
                if self._eyer:
                    self._result["eyer"]
                        =
                        eyes.getRightEye()
                if self._eyel:
                    self._result["eyel"]
                        =
                        eyes.getLeftEye()
            if self._nose:
                nose = NoseDetector()
                nose.detect(frame,
                    face_rect)
                self._result["nose"]
                    = nose.get()
            if self._mouth:
                mouth =
                    MouthDetector()
                mouth.detect(frame,
                    face_rect)
                self._result["mouth"]
                    = mouth.get()
    def clear(self):
        """Elimina los
            resultados."""
        self._result = {"face" :
            None, "eyer" : None,
            "eyel" : None, "nose" :
            None, "mouth" : None}
    def get(self, name = "face"):
        """Devuelve el elemento
            especificado."""
        res = self._result[name]
        return res
    def getAll(self):
        """Devuelve todos los
            elementos."""
        res = super().get()
        return res
```

```
    def getDetected(self):
        """Devuelve los elementos
           detectados."""
        res = {}
        for key in
            self._result.keys():
            item = self._result[key]
            if item is not None:
                res[key] = item
        return res
    def detected(self, name="face"):
        """Comprueba si el elemento
           ha sido detectado."""
        return self.valid() and
            name in self._result
            and self._result[name]
            is not None
    def elementsDetected(self):
        """Comprueba que elementos
           han sido detectados."""
        res = {"face" : False,
            "eyer" : False, "eyel"
            : False, "nose" :
            False, "mouth" : False}
        if
            self.elementDetected("face"):
            res["face"] = True
        if
            self.elementDetected("eyel"):
            res["eyel"] = True
        if
            self.elementDetected("eyer"):
            res["eyer"] = True
        if
            self.elementDetected("nose"):
            res["nose"] = True
        if
            self.elementDetected("mouth"):
            res["mouth"] = True
        return res
    def isFace(self):
        """Comprueba si se ha
           detectado la cara"""
        return
            self.elementDetected("face")
    def isFullFace(self):
        """Comprueba si la cara
           contiene todos sus
           elementos."""
        f, er, el, m, n =
            self.elementsDetected()
        return f and er and el and
            m and n
    def getEvaluation(self):
        """Evalua la cara segun la
           deteccion."""
        res = -1
        if self._face is not None:
            res = 0
            if self._eyer is not
                None:
                res += 0.18
            if self._eyel is not
                None:
                res += 0.18
            if self._nose is not
                None:
                res += 0.32
```

```
            if self._mouth is not
                None:
                res += 32
        return res
    def drawElement(self, frame,
        name):
        """Dibuja el elemento
            especificado."""
        res = Drawer.rect(frame,
            self.getElement(name))
        return res
    def drawElements(self, frame,
        names):
        """Dibuja los elementos
            especificados."""
        res = Drawer.rects(frame,
            names)
    def draw(self, frame, color =
        Drawer.GREEN):
        """Dibua todos los
            elementos."""
        res = Drawer.rects(frame,
            self.getDetected().values(),
            color = color)
        return res
```

A.4. Imagen

```
from glob import glob
import cv2
from tfg.utils import *


class Frame(object):
    """Representa una imagen."""
    def __init__(self, mat = None):
        self._mat = mat
    @property
    def mat(self):
        """Devuelve una copia del
            array."""
        return self._mat.copy()
    @property
    def shape(self):
        """Devuelve las dimensiones
            del array."""
        return self._mat.shape
    @property
    def h(self):
        """Devuelve la altura de la
            imagen."""
        return self.shape[0]
    @property
    def w(self):
        """Devuelve la anchura de
            la imagen."""
        return self.shape[1]
    @property
    def size(self):
        """Devuelve una tupla con
            la anchura y altura de
            la imagen."""
        return (self.w, self.h)
    def load(self, filename, flags
        = 1):
        """Carga una imagen de un
            fichero."""
        assert self.isEmpty(), "The
            Frame is already
            loaded."
        self._mat =
            cv2.imread(filename,
            flags)
        res = self
        return res
    @staticmethod
    def loads(path, regex="*.jpg",
        flags = 1):
        """Carga un conjunto de
            imagenes de una
            carpeta"""
        filenames = glob(path + "/"
            + regex)
        filenames.sort()
        res = []
        for filename in filenames:
            res.append(Frame().load(filename,
                flags))
        return res
    def save(self, filename):
        """Guarda la imagen en un
            fichero."""
        cv2.imwrite(filename,
            self._mat)
```

```
    def show(self, wait = 1,
        winname = "Frame"):
        """Muestra la imagen por
            pantalla"""
        cv2.imshow(winname,
            self._mat)
        return cv2.waitKey(wait)
    def resize(self, factor = 1):
        """Escala la imagen por un
            factor."""
        res =
            Frame(cv2.resize(self._mat,
            (0,0), fx = factor, fy
            = factor))
        return res
    def rectangle(self, rect):
        """Devuelve el rectangulo
            de la imagen"""
        x, y, w, h = rect
        res =
            Frame(self._mat[y:y+h,
            x:x+w])
        return res
    def section(self, sh = 0, eh =
        1, sw = 0, ew = 1):
        """Devuelve una seccion de
            la imagen"""
        assert eh > sh and sh <= 1
            and eh <= 1, "Invalid
            heigth"
        assert ew > sw and sw <= 1
            and ew <= 1, "Invalid
            width"
        sh = f2i(self.h * sh)
        eh = f2i(self.h * eh)
        sw = f2i(self.w * sw)
        ew = f2i(self.w * ew)
        res =
            Frame(self._mat[sh:eh,
            sw:ew])
        return res
    def gray(self):
        """Devuelve la imagen en
            escala de gris"""
        res =
            Frame(cv2.cvtColor(self._mat,
            cv2.COLOR_BGR2GRAY))
        return res
    def copy(self):
        """Copia la imagen."""
        return
            Frame(self._mat.copy())
    def draw(self, type, obj, color
        = (255, 255, 255)):
        """Pinta datos sobre la
            imagen."""
        res = Drawer.draw(self,
            type, obj, color)
        return res
    def isGray(self):
        """Comprueba si la imagen
            esta en escala de
            grises."""
        return len(self.shape) is 2
    def isEmpty(self):
        """Comprueba si el array ha
            sido cargado."""
        return self._mat is None
```

```
class Drawer(object):
    """Dibuja formas geometricas en
        imagenes."""
    WHITE = (255, 255, 255)
    GREEN = (0, 255, 0)
    BLUE = (255, 0, 0)
    RED = (0, 0, 255)
    BLACK = (0, 0, 0)
    @staticmethod
    def draw(frame, type, obj,
        color = WHITE):
        """Dibuja los elementos
            especificados."""
        res = frame.copy()
        if type == "r":
            res =
                Drawer.rect(frame,
                obj, color)
        elif type == "rr":
            res =
                Drawer.rects(frame,
                obj, color)
        elif type == "p":
            res =
                Drawer.point(frame,
                obj, color)
        elif type == "pp":
            res =
                Drawer.points(frame,
                obj, color)
        return res
    @staticmethod
    def rect(frame, rect, color =
        WHITE):
        """Dibuja rectangulos en la
            imagen"""
        x, y, w, h = rect
        pt1 = (x, y)
        pt2 = (x + w, y + h)
        res =
            Frame(cv2.rectangle(frame.mat,
            pt1, pt2, color))
        return res
    @staticmethod
    def rects(frame, rects, color =
        WHITE):
        """Dibuja un conjunto de
            rectangulos."""
        res = frame.copy()
        for rect in rects:
            res = Drawer.rect(res,
                rect, color)
        return res
    @staticmethod
    def point(frame, point, color =
        WHITE):
        """Dibuja un punto en la
            imagen"""
        res =
            Frame(cv2.circle(frame.mat,
            point, 3, color, -1))
        return res
```

```
    @staticmethod
    def points(frame, points, color
        = WHITE):
        """Dibuja un conjunto de
            puntos en la cara."""
        res = frame.copy()
        for point in points:
            res = Drawer.point(res,
                tuple(point), color)
        return res
    @staticmethod
    def line(frame, pt1, pt2, color
        = WHITE):
        """Dibuja una linea en la
            imagen"""
        res =
            Frame(cv2.line(frame.mat,
            pt1, pt2, color))
        return res
```

A.5. Cámara

```
from threading import Thread, Lock
from tfg.image import *

class Video(object):
    """Representa un video."""
    def __init__(self):
        self._capture = None
        self._fps = None
    def load(self, path):
        """Carga un video desde un
            fichero"""
        self._capture =
            cv2.VideoCapture(path)
        return self
    def read(self):
        """Captura la siguiente
            imagen de un video."""
        success, mat =
            self._capture.read()
        res = Frame(mat)
        return success, res
    def getFps(self):
        """Devuelve los fotogramas
            por segundo."""
        return
            self._capture.get(cv2.CAP_PROP_FPS)
    def getSize(self):
        """Devuelve el tamanio de
            las imagenes."""
        size =
            (int(self._capture.get(
        cv2.CAP_PROP_FRAME_WIDTH)),
                int(self._capture.get(
        cv2.CAP_PROP_FRAME_HEIGHT)))
        return size
    def isLoaded(self):
        """Comprueba si el video se
            ha cargado."""
        return self._capture is not
            None
    def release(self):
        """Libera la carga del
            video."""
        self._capture.release()
class Camera(Video):
    """Representa imagenes de una
        camara"""
    def __init__(self, channel = 0):
        super().__init__()
        self.load(channel)
    def load(self, channel = 0):
        """Comienza la captura."""
        self._capture =
            cv2.VideoCapture(channel)
class ImprovedCamera(Video):
    """Detecta imagenes usando
        distintos hilos."""
    def __init__(self, channel = 0):
        super(ImprovedCamera,
            self).__init__()
        self.camera =
            Camera(channel)
        self.success, self.frame =
            self.camera.read()
        self.lock = Lock()
        self.thread =
            Thread(target=self.update,
            args=())
        self.thread.start()
```

```
    def update(self):
        """Carga imagenes de la
           camara segun esten
           disponibles."""
        while True:
            success, frame =
                self.camera.read()
            self.lock.acquire()
            self.success,
                self.frame =
                success, frame
            self.lock.release()
    def read(self):
        """Devuelve la ultima
           imagen capturada"""
        self.lock.acquire()
        frame = self.frame.copy()
        self.lock.release()
        return frame
    def __exit__(self, exc_type,
        exc_value, traceback):
        """Libera los hilos en
           ejecucion"""
        self.thread.join()
        self.camera.release()
```

A.6. Reconstrucción

A.6.1. Python

```
from tfg.detector import
    FrontalFaceDetector
from tfg.image import *
from tfg.landmark import Landmarker
from tfg.utils import *


class ReconstructionHelper(object):
    """Facilita la configuracion de
       la reconstruccion"""
    def
       reconstruct_from_points(self,
       device, name, points):
       params =
           reconstrParams(device)
       return Reconstructor(name,
           points,
           params).reconstruct()
    def
       reconstruct_from_frames(self,
       device, name, frames):
       points =
           self._frames2points(frames)
       return
           self.reconstruct_from_points(device,
           name, points)
    def reconstruct_from_path(self,
       device, name, path, regex):
       frames =
           self._frames_in_path(path,
           regex)
       return
           self.reconstruct_from_frames(device,
           name, frames)
    def
       reconstruct_from_images(self,
       device, name, regex):
       path =
           'reconstruction/images/'
           + device
       return
           self.reconstruct_from_path(device,
           name, path, regex)
    def load_results(self, name):
       return
           Reconstructor.results(name)
    def _frames2points(self,
       frames):
       detector =
           FrontalFaceDetector()
       landmarker = Landmarker()
       points = []
       for frame in frames:
           detector.detect(frame)
           if detector.detected():
               rect = detector.get()
               landmarks =
                   landmarker.locate(frame,
                   rect)
               points.append(landmarks)
       return points
    def _frames_in_path(self, path,
       regex):
       return Frame.loads(path,
           regex)
```

```
class Reconstructor(object):
    """Reconstruye un modelo
       tridimensional segun los
       puntos bidimensionales
       pasados."""
    def __init__(self, name,
        points, params):
        self._name = name
        self._points = points
        self._params = params
    def reconstruct(self):
        """Ejecuta el proceso"""
        self.formatPoints()
        self.savePoints()
        self.execute()
        return
            Reconstructor.results(self._name)
    def formatPoints(self):
        """Formatea los puntos
           adecuadamente."""
        points = []
        for point_set in
            self._points:
            x_pts, y_pts = [], []
            for point in point_set:
                x, y = point[0],
                    point[1]
                x_pts.append(x)
                y_pts.append(y)
            x_pts.extend(y_pts)
            points.append(x_pts)
        self._points = points
```

```
    def savePoints(self):
        """Almacena los puntos
           bidimensionales"""
        path =
            "reconstruction/reconstructions/"
            + self._name
        removeCreateDir(path)
        with open(path +
            "/inpoints.txt", "w+")
            as outfile:
            outfile.write(self.writableLine(self.
                + "\n")
            outfile.write(self.writableLines(self
            outfile.close()
    def writableLine(self, list):
        """Construye una linea apta
           para guardar."""
        res = ""
        for elem in list:
            res += str(elem) + " "
        res = res[:-1]
        return res
    def writableLines(self, list):
        """Construye un conjunto de
           lineas aptas para
           guardar."""
        res = ""
        for elem in list:
            res +=
                self.writableLine(elem)
                + "\n"
        res = res[:-1]
        return res
```

```
    def execute(self):
        """Ejecuta el codigo C++ de
           reconstruccion"""
        os.chdir("reconstruction/c++")
        os.system("g++
            -I/usr/local/include/opencv
            -I/usr/include/eigen3
            -O3 -g3 -Wall -c
            -fmessage-length=0 -MMD
            -MP"
                    " -MF'main.d'
                       -MT'main.o'
                       -o 'main.o'
                       'main.cpp'")
        os.system("g++
            -L/usr/local/lib -o
            'reconstruction' main.o
            -lopencv_sfm
            -lopencv_stereo
            -lopencv_viz "
                  "-lopencv_surface_matching
                      -lopencv_tracking
                      -lopencv_plot
                      -lopencv_shape
                      -lopencv_video
                      "
                  "-lopencv_calib3d
                      -lopencv_features2d
                      -lopencv_highgui
                      -lopencv_imgcodecs
                      -lopencv_objdetect
                      "
                  "-lopencv_imgproc
                      -lopencv_core")
        os.system("./reconstruction"
            + " " + self._name)
        os.chdir("./../..")
    def results(name:str):
        """Carga los resultados de
            la reconstruccion."""
        path =
            "reconstruction/reconstructions/"
            + name +
            "/outpoints.txt"
        with open(path) as infile:
            data = [x.strip() for x
                in
                infile.readlines()]
        params = [float(x) for x in
            data.pop(0).split()]
        points3d = []
        for line in data:
            point3d =
                array2float(line.split())
            points3d.append(point3d)
        return points3d, params
```

A.6.2. C++

```
#include <iostream>
#include <eigen3/Eigen/Core>
#include <opencv2/sfm.hpp>
#include <opencv2/viz.hpp>
#include <opencv2/calib3d.hpp>
#include <opencv2/core.hpp>
#include <opencv2/core/eigen.hpp>
#include
    <opencv2/sfm/projection.hpp>
#include
    <opencv2/sfm/triangulation.hpp>
#include
    <opencv2/sfm/fundamental.hpp>
#include <opencv2/sfm/numeric.hpp>
#include
    <opencv2/sfm/conditioning.hpp>
#include
    <opencv2/highgui/highgui.hpp>
#include <string>
#include <unistd.h>
#define GetCurrentDir getcwd
#include <iostream>
#include <fstream>
using namespace std;
using namespace cv;
using namespace cv::sfm;

void extractData(const string
    filename, vector<Mat>
    &points2d, Matx33d &K) {
    int npoints = 68;
    ifstream file(filename.c_str());
    if (file.is_open()) {
        double f, cx, cy;
        string line;
        getline(file, line);
        istringstream
            linestream(line);
        linestream >> f >> cx >> cy;
        K = Matx33d(f, 0, cx, 0, f,
            cy, 0, 0, 1);
        while (getline(file, line))
            {
            istringstream
                linestream(line);
            Mat_<double> frame(2,
                npoints);
            for (int i = 0; i < 2;
                i++) {
                for (int j = 0; j <
                    npoints; j++) {
                    double val;
                    linestream >>
                        val;
                    frame[i][j] =
                        val;
                }
            }
            points2d.push_back(frame);
        }
    } else {
        cout << "No se ha podido
            abrir el fichero" <<
            endl;
    }
}
```

```
void saveData(string filename,
    vector<Mat> points3d, Matx33d
    K) {
    ofstream file(filename.c_str());
    double f = K(0, 0);
    double cx = K(0, 2);
    double cy = K(1, 2);
    file << f << " " << cx << " "
        << cy << endl;
    int n3dpoints = points3d.size()
        - 1;
    for (int i = 0; i < n3dpoints;
        i++) {
        Mat point3d = points3d[i];
        file <<
            point3d.at<double>(0)
            << " " <<
            point3d.at<double>(1)
            << " " <<
            point3d.at<double>(2)
            << endl;
    }
    Mat point3d =
        points3d[n3dpoints];
    file << point3d.at<double>(0)
        << " " <<
        point3d.at<double>(1) << "
        " << point3d.at<double>(2);
}
```

```
int main(int argc, char *argv[]) {
    vector<Mat> points2d;
    Matx33d K;
    String path =
        "../reconstructions/" +
        string(argv[1]);
    string infilename = path +
        "/inpoints.txt";
    string outfilename = path +
        "/outpoints.txt";
    extractData(infilename,
        points2d, K);
    bool is_projective = true;
    vector<Mat> Rs_est, ts_est,
        points3d;
    reconstruct(points2d, Rs_est,
        ts_est, K, points3d,
        is_projective);
    saveData(outfilename, points3d,
        K);
}
```

A.7. Puntos faciales

```
import dlib
from tfg.utils import *


class Landmarker(object):
    """Detecta los puntos de
       referencia faciales."""
    def __init__(self,
       predictor_path =
       "landmarks/shape_predictor_68_face_landmarks.dat"):
       self._predictor =
          dlib.dlib.shape_predictor(predictor_path)
    def locate(self, frame, rect =
       None):
       """Detecta los puntos para
          una cara."""
       if rect is None:
          rect = rectFrame(frame)
       shape =
          self._predictor(frame.mat,
          rect2dlib(rect))
       res = shape2points(shape)
       return res
    def multi(self, frame, rects):
       """Detecta los puntos para
          varias caras."""
       res = []
       for rect in rects:
          det =
             self.locate(frame,
             rect)
          res.append(det)
       return res
```

A.8. Pose

```
import cv2
from sympy import Plane
import math
from tfg.detector import
    FrontalFaceDetector
from tfg.interface import *
from tfg.landmark import Landmarker
from tfg.plot import Plotter3D
from tfg.utils import *

class Pose(object):
   """Orientacion de una camara
      mediante su vision de la
      cara"""
    def __init__(self, points3d,
        mtx, dist, ref_frame=None,
        real_dist=None, face_side =
        None):
        self._ffdetector =
            FrontalFaceDetector()
        self._landmarker =
            Landmarker()
        self._positioner =
            Positioner(points3d,
            mtx, dist)
        self._refrvec = None
        self._factor = None
        self._ratio = None
        self.load_reference(ref_frame,
            real_dist, face_side)
    def load_reference(self,
        ref_frame, real_dist=None,
        face_side = None):
      """Carga de los valores de
         referencia"""
      if ref_frame is not None:
          self._ffdetector.detect(ref_frame)
          if face_side is not
              None:
              face_rect =
                  self._ffdetector.get()
              border = face_rect[2]
              self._ratio =
                  face_side /
                  border
          retval, origin, axis,
              rvec, face_rect,
              landmarks =
              self.run(ref_frame)
          if retval:
              self._refrvec = rvec
              if real_dist is not
                  None:
                  self._factor =
                      real_dist /
                      Transformer.dist3points(ori
    def run(self, frame):
      """Ejecucion del proceso"""
      frame =
          self.preprocess(frame)
      retval, rotation,
          translation, face_rect,
          landmarks =
          self.get_position(frame)
      if retval:
          axis =
              Pose.orient(rotation)
```

```
            translation =
                Pose.aim(translation,
                axis)
            rotation, translation,
                axis =
                self.corrector(rotation,
                translation, axis)
            translation =
                Pose.aim(translation,
                axis)
            translation =
                self.to_real(translation)
            return True,
                translation, axis,
                rotation, face_rect,
                landmarks
        else:
            return False, None,
                None, None, None,
                None
    @staticmethod
    def aim(origin, axis):
      """Apunta el sistema sobre
         las narinas"""
        d =
            Transformer.dist3points(origin)
        z_axis =
            Transformer.norm(axis[2])
        origin = [-d * zi for zi in
            z_axis]
        return origin
    @staticmethod
    def orient(rvec):
      """Obtiene la matriz de
         rotacion a partir del
         vector"""
        return
            Transformer.rotate_rvec(rvec,
            Plotter3D.CARTESIAN)
    def corrector(self, rvec, tvec,
        axis=None):
      """Correccion de la
         orientacion"""
        if axis is None: axis =
            Pose.orient(rvec)
        if tvec[2] > 0:
            x, _, z = tvec
            angle = 2 *
                math.atan(z/abs(x))
            if x < 0:
                angle *= -1
            rvec_mirror = [0,
                angle, 0]
            tvec[2] *= -1
            axis =
                Transformer.rotate_rvec(rvec_mirro
                axis)
            rvec =
                Transformer.mtx_rvec(axis)
        rvec = [rveci % (2 * pi)
            for rveci in rvec]
        rvec = [angle - 2 * pi if
            angle > pi else angle
            for angle in rvec]
        if self._refrvec is not
            None:
        rvec = [rvec[i] -
            self._refrvec[i] for i
            in range(0, 3)]
```

```
            rvec = [rveci % (2 *
                pi) for rveci in
                rvec]
            rvec = [angle - 2 * pi
                if angle > pi else
                angle for angle in
                rvec]
            axis = Pose.orient(rvec)
        return rvec, tvec, axis
    def get_position(self, frame):
      """Posicionamiento de la
         imagen"""
        frame_opt =
            self.optimize(frame)
        self._ffdetector.detect(frame_opt)
        face_rect =
            self._ffdetector.get()
        if face_rect is not None:
            landmarks =
                self._landmarker.locate(frame_opt,
                face_rect)
            if self._ratio is not
                None:
                landmarks =
                    [[int(comp /
                    self._ratio) for
                    comp in
                    landmark] for
                    landmark in
                    landmarks]
                face_rect =
                    [int(comp /
                    self._ratio) for
                    comp in
                    face_rect]
                retval, rvec, tvec =
                    self._positioner.position(landmark
                return retval, rvec,
                    tvec, face_rect,
                    landmarks
        else: return False, None,
            None, None, None
    def to_real(self, tvec):
      """Conversion a unidades
         reales"""
        if self._factor is not None:
            tvec = [self._factor *
                tveci for tveci in
                tvec]
        return tvec
    def preprocess(self, frame):
      """Preprocesado de la
         imagen"""
        if not frame.isGray():
            frame = frame.gray()
        return frame
    def optimize(self, frame):
      """Escalado de la imagen"""
        if self._ratio is not None:
            frame =
                frame.resize(self._ratio)
        return frame


class Positioner(object):
   """Orientacion de puntos
      respecto a un modelo
      tridimensional"""
    face_number = 196
    def __init__(self, points3d,
        mtx, dist):
        self._points3d = points3d
```

```
        self._mtx = mtx
        self._dist = dist
        self._last_rvec = None
        self._last_tvec = None
    @staticmethod
    def face_from_section(pos, p2d,
        p3d=None):
      """Componentes de una cara
         segun su numero"""
        p2d = p2d.copy()
        sections2 = [p2d[0:17],
            p2d[17:22], p2d[22:27],
            p2d[27:36], p2d[36:42],
            p2d[42:48], p2d[48:60],
            p2d[60:]]
        p2d.clear()
        if p3d is not None:
            p3d = p3d.copy()
            sections3 = [p3d[0:17],
                p3d[17:22],
                p3d[22:27],
                p3d[27:36],
                p3d[36:42],
                p3d[42:48],
                p3d[48:60], p3d[60:]]
            p3d.clear()
        nb = bin(pos)[2:].zfill(8)
        val = [int(nbi) == 1 for
            nbi in nb]
        for i in range(0, 8):
            if val[i]:
                p2d += sections2[i]
                if p3d is not None:
                    p3d +=
                        sections3[i]
        if p3d is not None:
            return p2d, p3d
        else:
            return p2d
    def position(self, points2d):
      """Orientacion de puntos
         bidimensionales"""
        points3d = self._points3d
        points2d = points2d
        p2d, p3d =
            Positioner.face_from_section(Positione
            points2d, points3d)
        retval, rvec, tvec =
            cv2.solvePnP(self.points3numpy(p3d),
            lists2numpy(p2d),
            self._mtx, self._dist)
        rvec, tvec =
            simplify(rvec.tolist()),
            simplify(tvec.tolist())
        return retval, rvec, tvec
    def points3numpy(self,
        points3d):
      """Conversion de puntos a
         array numpy"""
        array =
            np.zeros((len(points3d),
            3))
        i = 0
        for point in points3d:
            x, y, z = point
            subarray = np.array([x,
                y, z])
            array[i] = subarray
            i += 1
        return array
```

```
class Aligner(object):
   """Alineacion de puntos de la
      cara"""
    MOUTH_R = 49 - 1
    MOUTH_L = 55 - 1
    EYE_R = 37 - 1
    EYE_L = 46 - 1
    NOSE_B = 34 - 1
    NOSE_U = 28 - 1
    NOSE_R = 32 - 1
    NOSE_L = 36 - 1
    def __init__(self, points):
        self._points = points.copy()
        self.load()
    def load(self):
      """Carga de puntos"""
        self._mouthR =
            self._points[self.MOUTH_R]
        self._mouthL =
            self._points[self.MOUTH_L]
        self._eyeR =
            self._points[self.EYE_R]
        self._eyeL =
            self._points[self.EYE_L]
        self._noseB =
            self._points[self.NOSE_B]
        self._noseU =
            self._points[self.NOSE_U]
        self._noseR =
            self._points[self.NOSE_R]
        self._noseL =
            self._points[self.NOSE_L]
        self._noseM =
            Transformer.midpoint(self._noseR,
            self._noseL)
    def align(self):
      """Proceso de alineado"""
        origin = self._noseB
        self._points =
            Transformer.translate(origin,
            self._points)
        x, y, z = self.axis()
        mtx =
            Transformer.mtx_new_axis(x,
            y, z)
        self._points =
            Transformer.rotate(mtx,
            self._points)
        return self._points
    def axis(self):
        y = [x - y for x, y in
            zip(self._noseU,
            self._noseM)]
        z = [- zi for zi in
            Transformer.normal_vector(self._noseR,
            self._noseL,
            self._noseU)]
        x = Transformer.cross(z, y)
        return Transformer.norms(x,
            y, z)


class Transformer(object):
   """Realiza transformaciones
      geometricas"""
    @staticmethod
    def cross(vec1, vec2):
        vec1np, vec2np =
            lists2numpy((vec1,
            vec2))
```

```
        vecnp = np.cross(vec1np,
            vec2np)
        return numpy2list(vecnp)
    @staticmethod
    def rotate(mtx, points):
        points = [np.dot(mtx,
            list2numpy(point)) for
            point in points]
        return numpys2list(points)
    @staticmethod
    def rotate_rvec(rvec, points):
        return Transformer.rotate(
         Transformer.rvec_mtx(rvec),
             points)
    @staticmethod
    def translate(origin, points):
        t = [-i for i in origin]
        points = [p3d.translate(*t)
            for p3d in
            [Point3D(*point) for
            point in points]]
        return s3ts(*points)
    @staticmethod
    def rvec_mtx(rvec):
        mtx = cv2.Rodrigues(
         np.asarray(rvec))[0].tolist()
        return mtx
    @staticmethod
    def mtx_rvec(mtx):
        res = np.zeros((3, 3))
        mtx = np.asarray(mtx)
        rvec = cv2.Rodrigues(mtx)[0]
        return list(map(float,
            rvec))
    @staticmethod
    def mtx_new_axis(x, y, z):
        mtx = np.zeros((3, 3))
        mtx[0, :], mtx[1, :],
            mtx[2, :] =
            lists2numpy((x, y, z))
        return mtx
    @staticmethod
    def norm(vector):
        x, y, z = vector
        m = sqrt(pow(x, 2) + pow(y,
            2) + pow(z, 2))
        return [x / m, y / m, z / m]
    @staticmethod
    def norms(*vectors):
        res = []
        for vector in list(vectors):
            res.append(Transformer.norm(vector))
        return res
    @staticmethod
    def midpoint(pt1, pt2):
        pts1 = t3s(pt1)
        pts2 = t3s(pt2)
        return
            s3t(pts1.midpoint(pts2))
    @staticmethod
    def normal_vector(pt1, pt2,
        pt3):
        plane = Plane(t3s(pt1),
            t3s(pt2), t3s(pt3))
        normalv =
            s3t(plane.normal_vector)
        return
            Transformer.norm(normalv)
```

```
    @staticmethod
    def inverse(vector):
        return [-i for i in vector]
    @staticmethod
    def dist3points(pt1, pt2=(0, 0,
        0)):
        x1, y1, z1 = pt1
        x2, y2, z2 = pt2
        return sqrt(pow(x1 - x2, 2)
            + pow(y1 - y2, 2) +
            pow(z1 - z2, 2))
```

A.9. Gráficos

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import
    Axes3D
from tfg.utils import *

class Plotter3D(object):
    """Representa objetos
       tridimensionales."""
    CARTESIAN = [[1, 0, 0], [0, 1,
       0], [0, 0, 1]]
    ORIGIN = [0, 0, 0]
    def __init__(self, xticks =
       None, yticks = None, zticks
       = None):
       self._figure = plt.figure()
       self._figure.canvas.mpl_connect('close_event',
          self.close_event)
       self._ax =
          self._figure.add_subplot(111,
          projection='3d')
       self._mng =
          plt.get_current_fig_manager()
       self._xticks = xticks
       self._yticks = yticks
       self._zticks = zticks
       self._closed = False
       self.load()
    def load(self):
       """Carga de la interfaz."""
       self._ax.set_xlabel("x")
       self._ax.set_ylabel("y")
       self._ax.set_zlabel("z")
       if self._xticks is not
          None:
          self._ax.autoscale(False)
       if self._xticks is not
          None:
          self._ax.set_xlim(self._xticks)
       if self._yticks is not
          None:
          self._ax.set_ylim(self._yticks)
       if self._zticks is not
          None:
          self._ax.set_zlim(self._zticks)
    def points(self, points3d,
       color = None):
       """Representacion de puntos
       tridimensionales."""
       X, Y, Z =
          self.t2ns_sep(points3d)
       return
          self._ax.scatter(X,Y,Z,
          c=color)
    def point(self, point3d, color
       = None):
       """Representacion de un
          punto tridimensional."""
       X, Y, Z =
          list2numpy(point3d)
       return
          self._ax.scatter(X,Y,Z,
          c = color)
    def vectorFromPoints(self,
       pini, ptend, color = None):
```

```
        """Generacion de un vector
            grafico a partir de
            puntos."""
        component = [ptend[i] -
            pini[i] for i in
            range(0, 3)]
        return
            self.vectorFromComponent(pini,
            component, color)
    def vectorFromComponent(self,
        pini, component, color =
        None):
        """Generacion de un vector
            grafico a partir de
            componentes"""
        X, Y, Z = pini
        U, V, W = component
        return self._ax.quiver(X,
            Y, Z, U, V, W, color =
            color)
    def vector(self, ptini, ptend =
        None, component = None,
        color = None):
        """Generacion de un vector
            grafico"""
        if ptend is not None:
            return
                self.vectorFromPoints(ptini,
                ptend, color)
        elif component is not None:
            return
                self.vectorFromComponent(ptini,
                component, color)
```

```
    def linkFace(self, points):
        """Union de puntos
            faciales."""
        face = points[0:17]
        eyebrownr = points[17:22]
        eyebrownl = points[22:27]
        eyer = points[36:42] +
            [points[36]]
        eyel = points[42:48] +
            [points[42]]
        nose = points[27:31] +
            [points[33]]
        noseext = points[31:36] +
            [points[31]]
        mouthe = points[48:60] +
            [points[48]]
        mouthi = points[60:68] +
            [points[60]]
        self.linkMultiPoints(face,
            eyebrownl, eyebrownr,
            eyer, eyel, nose,
            noseext, mouthe, mouthi)
    def axis(self, axis =
        CARTESIAN, origin = ORIGIN,
        scale=None):
        """Representacion de eje
            tridimensional."""
        x, y, z = axis
        if scale is not None:
            if isinstance(scale,
                list):
```

```
            x, y, z = [[scale[i]
                * comp for comp
                in axis[i]] for
                i in range(0,
                len(axis))]
        else:
            x, y, z = [[scale *
                comp for comp in
                axisi] for axisi
                in axis]
    xv = self.vector(origin,
        component=x, color='r')
    yv = self.vector(origin,
        component=y, color='g')
    zv = self.vector(origin,
        component=z, color='b')
    return [xv, yv, zv]
def endoscope(self, axis,
    origin, length):
    """Representacion de
        endoscopio."""
    z = axis[2]
    return self.vector(origin,
        component=[-length *
        z[0], -length * z[1],
        -length * z[2]])
def clear(self):
    """Reinicio de la figura."""
    self._ax.cla()
    self.load()
def remove(self, elem):
    """Eliminacion de un
        elemento grafico."""
    if isinstance(elem, list):
        for el in elem:
            try:
                el.remove()
            except:
                continue
    else:
        elem.remove()
def deletes(self, *elems):
    """Eliminacion de varios
        elementos graficos"""
    res = [self.delete(elem)
        for elem in elems]
    return res
def face(self, points):
    """Representacion de una
        cara."""
    self.points(points)
    self.linkFace(points)
def linkMultiPoints(self,
    *multipoints):
    """Union de varios
        puntos."""
    for points in
        list(multipoints):
        self.linkPoints(points)
def linkPoints(self, points3d,
    color = 'b'):
    """Union de dos puntos."""
    X, Y, Z =
        self.t2ns_sep(points3d)
    self._ax.plot(X, Y, Z,
        color = color)
def plot(self, xticks = None,
    yticks = None, ion = True):
```

```
        """Pinta los elementos."""
        if xticks is not None:
            plt.xticks(xticks)
        if yticks is not None:
            plt.yticks(yticks)
        if ion:
            plt.ion()
        plt.show()
    def draw(self):
        """Actualiza la interfaz."""
        plt.pause(0.0001)
        plt.draw()
    def t2ns_sep(self, points3d):
        """Adaptacion a matplotlib
            de puntos"""
        X, Y, Z = [list2numpy(elem)
            for elem in
            zip(*points3d)]
        return X, Y, Z
    def close_event(self, event):
        """Captura el evento de
            cerrado de la figura."""
        self._closed = True
    def is_closed(self):
        """Comprueba si la figura
            esta cerrada."""
        return self._closed
    def close(self):
        """Cierra la figura."""
        plt.close()
    def geometry(self, width=700,
        height=600, posx=0, posy=0):
        """Mueve la ventana en la
            pantalla."""
        self._mng.window.wm_geometry(str(width)
            + "x" + str(height) +
            "+" + str(posx) + "+" +
            str(posy))


def plotter_face(points3d,
    scale=None, no_axis = False):
    """Configuracion de una figura
        con una cara
        tridimensional"""
    if scale is not None:
        points3d = [[scale * comp
            for comp in point] for
            point in points3d]
    limit = max([max(point) for
        point in points3d])
    range = [-2 * limit, 2 * limit]
    plotter = Plotter3D(range,
        range, range)
    plotter.face(points3d)
    if not no_axis: plotter.axis()
    plotter.plot()
    return plotter
```

A.10. Utilidades

```
import json
import os
from math import sqrt, pow, pi
import dlib
import numpy as np
from sympy import Point3D


def simplify(l):
    if isinstance(l, list):
        if len(l) > 1:
            return [simplify(i) for
                i in l]
        elif len(l) == 1:
            return l[0]
        else:
            return []
    else:
        return l
def f2i(f):
    """Redondea un numero decimal
        al entero mas cercano."""
    return int(round(f))
def r2g(radian):
    """Radianes a grados"""
    return radian * 180 / pi
def g2r(angle):
    """Grados a radianes"""
    return angle * pi / 180
def addRects(rect1, rect2):
    """Suma dos rectangulos."""
    x1, y1, w1, h1 = rect1
    x2, y2, w2, h2 = rect2
    res = (x1 + x2, y1 + y2, w1 +
        w2, h1 + h2)
    return res
def addToRect(rect, dx = 0, dy =
    0, dw = 0, dh = 0):
    """Modifica un rectangulo"""
    x, y, w, h = rect
    res = (x + dx, y + dy, w + dw,
        h + dh)
    return res
def rectSection(rect, sw = 0, ew =
    1, sh = 0, eh = 1):
    """Seccion de un rectangulo."""
    x, y, w, h = rect
    x = x + w * sw
    y = y + h * sh
    w = w * (ew - sw)
    h = h * (eh - sh)
    res = (f2i(x), f2i(y), f2i(w),
        f2i(h))
    return res
def rectOrigin(rect):
    """Origen de un rectangulo"""
    x, y, _, _ = rect
    res = (x, y)
    return res
def rectSize(rect):
    """Tamanio de un rectangulo."""
    _, _, w, h = rect
    res = (w, h)
    return res
def rectCenter(rect):
    """Centro de un rectangulo."""
    x, y, w, h = rect
    x = f2i(x + w/2)
    y = f2i(y + h/2)
    res = (x, y)
    return res
```

```
def rectFrame(frame):
    """Rectangulo con dimensiones
       de una imagen"""
    x, y = (0, 0)
    w, h = frame.size
    res = (x, y, w - 1, h - 1)
    return res
def distPoints(pt1, pt2):
    """Distancia euclidea entre dos
       puntos."""
    x1, y1 = pt1
    x2, y2 = pt2
    res = f2i(sqrt(pow(x1 - x2, 2)
       + pow(y1 - y2, 2)))
    return res
def array2rect(array):
    """Array a rectangulo"""
    x, y, w, h = array
    res = (x, y, w, h)
    return res
def arrays2rects(arrays):
    """Arrays a rectangulos"""
    res = []
    for array in arrays:
        res.append(array2rect(array))
    return res
def array2float(array):
    """Array con decimales"""
    return [float(x) for x in array]
def loadParams(name):
    """Carga de parametro de
       calibracion."""
    path =
       "calibration/calibrations/"
       + name
    mtx = np.load(path + "/mtx.npy")
    dist = np.load(path +
       "/dist.npy")
    return mtx, dist
def reconstrParams(name):
    """Carga de datos
       reconstruidos."""
    mtx, dist = loadParams(name)
    return (mtx[0][0], mtx[0][2],
       mtx[1][2])
def removeDir(path):
    """Elimina un directorio."""
    if os.path.isdir(path):
        for i in os.listdir(path):
            os.remove(os.path.join(path,
               i))
        os.rmdir(path)
def createDir(path):
    """Crea un directorio."""
    os.mkdir(path)
def removeCreateDir(path):
    """Vacia un directorio."""
    removeDir(path)
    createDir(path)
def load_json(file):
    """Carga datos de un fichero
       JSON"""
    res = None
    try:
        with open(file) as infile:
            res = json.load(infile)
    except:
        pass
    return res
```

```
def save_json(file, object):
    """Guarda datos en un fichero
        JSON"""
    with open(file, 'w') as outfile:
        json.dump(object, outfile)
def rect2dlib(rect):
    """Rectangulos de tipo DLib"""
    x, y, w, h = rect
    l = np.uint32(x).item()
    t = np.uint32(y).item()
    r = np.uint32(x + w).item()
    b = np.uint32(y + h).item()
    return dlib.rectangle(l, t, r,
        b)
def dlib2rect(dlib_rectangle):
    """Rectangulos DLib a
        rectangulos"""
    x = dlib_rectangle.left()
    y = dlib_rectangle.top()
    w = dlib_rectangle.right() - x
    h = dlib_rectangle.bottom() - y
    return (x, y, w, h)
def shape2points(shape):
    """Dlib puntos a puntos"""
    npoints = 68
    points = []
    for i in range(0, npoints):
        points.append((shape.part(i).x,
            shape.part(i).y))
    return points
def list2numpy(l):
    """Lista a numpy array."""
    size = len(l)
    array = np.zeros((size))
    for i in range(0, size):
        array[i] = l[i]
    return array
def numpy2list(n):
    """Array numpy a lista"""
    return list(n)
def lists2numpy(ls):
    """Listas a numpy arrays"""
    array = np.zeros((len(ls),
        len(ls[0])))
    for i, l in enumerate(ls):
        temp = list2numpy(l)
        array[i] = temp
    return array
def numpys2list(ns):
    """Arrays numpy a listas."""
    res = [numpy2list(n) for n in
        ns]
    return res
def t3s(point3d):
    """Lista a puntos
        tridimensionales."""
    return Point3D(*point3d)
def s3t(point3d):
    """Puntos tridimensionales a
        lista"""
    x, y, z = point3d
    return [x, y, z]
def t3ss(*t3s):
    """Listas a puntos
        tridimensionales"""
    res = [t3s(t3) for t3 in
        list(t3s)]
    return res
def s3ts(*s3s):
    """Puntos tridimensionales a
        listas"""
    res = [s3t(s3) for s3 in
        list(s3s)]
    return res
```

Bibliografía

[1] Global image guided surgery devices market us 4.8 billion by 2021. https://www.ihealthcareanalyst.com/image-guided-surgery-devices-market.

[2] Diseases of the sinuses: A comprehensive textbook of diagnosis and treatment. *The Laryngoscope 107*, 4 (1997), 547–547.

[3] The p3p (perspective-three-point) principle. http://iplimage.com/blog/p3p-perspective-point-overview/, May 2008.

[4] Dlib. http://dlib.net/, Jun 2018.

[5] Matplotlib. https://matplotlib.org/, Jun 2018.

[6] Numpy. http://www.numpy.org/, Jun 2018.

[7] Opencv library. https://opencv.org/, Jun 2018.

[8] Python. https://www.python.org/, Jun 2018.

[9] Ubuntu. https://www.ubuntu.com/, Jun 2018.

[10] Ali, A. A., Richmond, S., Popat, H., Toma, A. M., Playle, R., Pickles, T., Zhurov, A. I., Marshall, D., Rosin, P. L., Henderson, J., and et al. A three-dimensional analysis of the effect of atopy on face shape. *The European Journal of Orthodontics 36*, 5 (2013), 506–511.

[11] Atick, J. J., Griffin, P. A., and Redlich, A. N. Statistical approach to shape from shading: Reconstruction of three-dimensional face surfaces from single two-dimensional images. *Neural Computation 8*, 6 (1996), 1321–1340.

[12] B., A. J. Computer-aided endoscopic sinus surgery. *The Laryngoscope 108* (1998), 949–961.

[13] BATRA, P. S. *Practical medical and surgical management of chronic rhinosinusitis.* SPRINGER INTERNATIONAL PU, 2016.

[14] Birkfellner, W., Figl, M., Matula, C., Hummel, J., Hanel, R., Imhof, H., Wanschitz, F., Wagner, A., Watzinger, F., Bergmann, H., and et al. Computer-enhanced stereoscopic vision in a head-mounted operating binocular. *Physics in Medicine and Biology 48*, 3 (2003).

[15] Cordon, O., Damas, S., Del Coso, R., and ibáñez, O. Soft computing developments of the applications of fuzzy logic and evolutionary algorithms research unit at the european centre for soft computing.

[16] Dainese, G., Marcon, M., Sarti, A., and Tubaro, S. Accurate depth-map estimation for 3d face modeling. 1–4.

[17] Dalal, N., and Triggs, B. Histograms of oriented gradients for human detection. 886–893 vol. 1.

[18] Draf, W. Technique of paranasal endoscopy. *Endoscopy of the Paranasal Sinuses* (1983).

[19] Fei-Fei, L., Fergus, R., and Perona, P. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *2004 Conference on Computer Vision and Pattern Recognition Workshop*.

[20] Fischer, J., Eichler, M., Bartz, D., and Strasser, W. A hybrid tracking method for surgical augmented reality. *Computers & Graphics 31*, 1 (2007), 39–52.

[21] García, C. A. *Estudio de técnica de generación de modelos 3D a partir de secuencias de imágenes.* PhD thesis, 2017.

[22] Georghiades, A., Belhumeur, P., and Kriegman, D. From few to many: generative models for recognition under variable pose and illumination. *Proceedings*

Fourth IEEE International Conference on Automatic Face and Gesture Recognition (Cat. No. PR00580).

[23] Gray, H. *Anatomy of the Human Body*. Lea & Febiger, 1918.

[24] Hsu, G. S., Loc, T. T., and Chung, S. L. A comparison study on appearance-based object recognition. 3500–3503.

[25] Hung, Y. S., and Tang, W. K. Projective reconstruction from multiple views with minimization of 2d reprojection error. *International Journal of Computer Vision 66*, 3 (2006), 305–317.

[26] Jing, C., Yongtian, W., Yue, L., and Dongdong, W. Navigating system for endoscopic sinus surgery based on augmented reality. *2007 IEEE/ICME International Conference on Complex Medical Engineering* (2007).

[27] Kersten-Oertel, M., Jannin, P., and Collins, D. L. The state of the art of visualization in mixed reality image guided surgery. *Computerized Medical Imaging and Graphics 37*, 2 (Mar 2013), 98–112.

[28] Komorniczak, M. Senos paranasales. `https://da.m.wikipedia.org/wiki/Fil:Senos_paranasales.jpg`, Oct 2011.

[29] Konishi, K., Hashizume, M., Nakamoto, M., Kakeji, Y., Yoshino, I., Taketomi, A., Sato, Y., Tamura, S., and Maehara, Y. Augmented reality navigation system for endoscopic surgery based on three-dimensional ultrasound and computed tomography: Application to 20 clinical cases. *International Congress Series 1281* (2005), 537–542.

[30] Kouzani, A., He, F., and Sammut, K. Commonsense knowledge-based face detection. *Proceedings of IEEE International Conference on Intelligent Engineering Systems* (1997).

[31] Leiner, D. C. *Digital Endoscope Design*. Society of Photo-Optical Instrumentation Engineers (SPIE), 2015.

[32] Leonard, S., Reiter, A., Sinha, A., Ishii, M., Taylor, R. H., and Hager, G. D. Image-based navigation for functional endoscopic sinus surgery using structure from motion. *Medical Imaging 2016: Image Processing* (2016).

[33] Lepetit, V., Moreno-Noguer, F., and Fua, P. Epnp: An accurate o(n) solution to the pnp problem. *International Journal of Computer Vision 81*, 2 (2008), 155–166.

[34] Liao, H., Inomata, T., Sakuma, I., and Dohi, T. 3-d augmented reality for mri-guided surgery using integral videography autostereoscopic image overlay. *IEEE Transactions on Biomedical Engineering 57*, 6 (2010), 1476–1486.

[35] Mutter, J. C. Mani zadeh md endoscopic sinus surgery, Oct 2010.

[36] Quintana, C. J. C. *Determinación de distancias entre objetos de una imagen.* PhD thesis, 2011.

[37] Sagonas, C., Antonakos, E., Tzimiropoulos, G., Zafeiriou, S., and Pantic, M. 300 faces in-the-wild challenge: database and results. *Image and Vision Computing 47* (2016), 3–18.

[38] Swaroop, P., and Sharma, N. An overview of various template matching methodologies in image processing. *International Journal of Computer Applications 153*, 10 (2016), 8–14.

[39] Torresani, L., Hertzmann, A., and Bregler, C. Nonrigid structure-from-motion: Estimating shape and motion with hierarchical priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence 30*, 5 (May 2008), 878–892.

[40] Viola, P., and Jones, M. Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001* (2001).

[41] Viola, P., and Jones, M. Fast and robust classification using asymmetric adaboost and a detector cascade. 1311–1318.

[42] Vogt, S., Khamene, A., and Sauer, F. Reality augmentation for medical procedures: System architecture, single camera marker tracking, and system evaluation. *International Journal of Computer Vision 70*, 2 (2006), 179–190.

[43] Yale. The extended yale face database b.

[44] Yang, M.-H., Kriegman, D., and Ahuja, N. Detecting faces in images: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*, 1 (2002), 34–58.

[45] Zafeiriou, S., Zhang, C., and Zhang, Z. A survey on face detection in the wild: Past, present and future. *Computer Vision and Image Understanding 138* (2015), 1–24.