UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Máster Oficial en Sistemas Electrónicos Avanzados.
Sistemas Inteligentes

Universidad de Alcalá

Tésis de Máster

Estimación robusta de la dirección de la cara a partir de un modelo 3D de la cabeza del usuario, obtenido mediante un scan-láser y de técnicas de visión estéreo

Sebastián Bronte Palacios
2009

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Máster Oficial de Sistemas Electrónicos Avanzados. Sistemas Inteligentes

Tésis de Máster

Estimación robusta de la dirección de la cara a partir de un modelo 3D de la cabeza del usuario, obtenido mediante un scan-láser y de técnicas de visión estéreo

Alumno: Sebastián Bronte Palacios

Director: Dr. D. Luis Miguel Bergasa Pascual

Tribunal:

Presidente: D. Rafael Barea Navarro.

Vocal 1º: D. Jose Manuel Rodriguez Ascariz.

Vocal 2º: D. Luis Miguel Bergasa Pascual.

CALIFICACIÓN:.....

FECHA: .....

A mis padres ... y a Aida

“ Lo que con mucho trabajo se adquiere, más se ama ”
Aristóteles

Agradecimientos

Como en el proyecto de fin de carrera, no puedo empezar de otra forma este apartado sin darle las gracias a mis padres y a mi abuela. Sin ellos esto jamás habría sido posible, puesto que no habría llegado hasta donde lo he hecho.

En segundo lugar agradecer a mi tutor Luis Miguel Bergasa el apoyo recibido, por sus consejos y sus lecciones, puesto que, además de tutor en esta Tesis de Máster, me ha ido guiando a lo largo de la última parte de la carrera, además de apoyarme a la hora de hacer la labor de investigación y ofreciéndome el trabajo en el que estoy ahora mismo.

También agradecer a Pedro Jiménez su ayuda y colaboración en la adaptación de su software, el apoyo en parte del mío propio y las ideas que me ha sugerido para la realización de las pruebas realizadas en esta Tesis de Máster.

Como no, agradecer a mis compañeros del laboratorio donde he desarrollado el proyecto, y amigos derivados del mismo: Balky, Juanpa, Miguel, Pablo, Ivan, Diego, José Luis, Amaia, Álvaro, Jesús, Nacho y todos los demás, por su colaboración, su ayuda, los momentos que hemos pasado juntos, y el hecho de haberles quitado parte de su tiempo para dejarse escanear y hacer las distintas pruebas que en esta Tesis de Máster se presentan, además de las ideas dadas para realizar el trabajo y las reuniones de grupo realizadas. Mención especial para Pablo, puesto que siempre ha tenido un rato para aportar alguna idea y ayudarme en su implementación práctica y para Ivan por ser buen compañero de trabajo.

También quiero dedicar unas palabras para mis amigos más cercanos: Fran y Cris, Esteban, Adrián, Murcia, Manzanares, Semper, Sara, Alberto, David ..., sin los cuales no habría tenido ratos de relax, e incluso de inspiración en este trabajo y por supuesto de diversión, pues todo no va a ser trabajo académico, sino que también el esparcimiento es necesario a la hora de seguir estudiando y trabajando. Mención especial para Esteban por los ratos de desahogo mutuo que hemos tenido a lo largo de todo el año y sobre todo en la última parte del mismo.

De manera especial quisiera agradecer la inmensa cantidad de cafés a los que me han invitado dos buenos amigos: Santi y Dani. Sin estos cafés de relax a media mañana y sin las comidas amenas y divertidas que hemos pasado tantos días, no habría sido lo mismo. Agradecimiento especial a Santi, sin el que, gracias a una mala casualidad, lo que viene en el siguiente párrafo, no habría sido posible.

Por último he de dedicar mis últimas palabras de agradecimiento, y no por ello las menos importantes, sino todo lo contrario, a alguien muy especial que cambió mi vida, poco después de conocernos. Aida María. Gracias por hacerme conocer la felicidad que antes jamás había imaginado. Gracias por ser mi musa particular en la que encuentro inspiración, gracias por haberte conocido, gracias por todo, gracias a ti.

Índice general

Índice de figuras

Índice de tablas

Lista de algoritmos

Parte I

Resumen

Resumen

En esta Tesis de Máster se proponen 2 mejoras para proporcionar mayor robustez a un sistema de seguimiento de caras en 3D basado en visión estéreo de mayor envergadura. Con este sistema se pretende localizar la cabeza de un usuario y seguir la dirección de la misma. El objetivo final es monitorizar la dirección de la cara de una persona con objeto de usar esta información en el diseño de un sistema de inatención de conductores.

Las mejoras propuestas son, por un lado, la disminución del tiempo de ejecución del algoritmo POSIT (Pose from Orthography and Scaling with ITerations), utilizado para la estimación de la pose del modelo 3D respecto de las imágenes proyectadas, y por otro lado, la mejora de la estimación, mediante la sustitución de los modelos genéricos por modelos 3D específicos para un usuario concreto.

Para la reducción del tiempo de ejecución del cálculo de pose se utilizan las predicciones anteriores del algoritmo RANSAC (RANdom SAmple Consensus) en lugar de partir, como se hace habitualmente, de una posición aleatoria.

En cuanto al las mejoras de los modelos 3D para cada usuario, se realizan varios pasos: la adquisición mediante un scan-láser de 3 vistas del usuario, el diezmado de las muestras de cada una de las vistas y la composición del modelo 3D a partir de los datos 2D diezmados obtenidos en cada una de las vistas.

Se muestran diferentes resultados experimentales que justifican las mejoras propuestas y se presentan las conclusiones de las mismas.

Palabras clave: visión computacional, modelos 3D, detección de esquinas, POSIT, RANSAC.

Abstract

In this Masther Thesis two improvements for a 3D face tracking system are presented. This main tracking system is based on an stereo vision system. This system is espected to be an accurate way to find the user's head and track its position and direction. The porpose of this system is to evaluate the head direction of a person, and, in consequence, the general state of a person attending to where the user is paying attention. This information will be used to develop a driver inattention detection system.

The improvements are, on one hand, the reduction of the execution time for the pose estimation algorithm, POSIT (Pose from Orthography and Scaling with ITerations), used to estimate the the 3D model pose with regard to the proyected points on the image. In the other hand, an improvement in the estimation will be proposed using specific 3D models for each user instead of using generic 3D models.

The execution time improvement for the pose algortihm is done by taking the previous correct estimation from RANSAC (RANdom SAmple Consensus) algorithm, instead of using a random initial estimation, as the basic algorithm does.

As for the 3D model improvement for every user, several steps are needed: the acquisition of samples through an scan-laser of three views for each user, the decimation of samples for every view and the composition of the 3D model from 2D decimate data, obtained for each view.

Experimental results that justify the improvements made and the conclusions will be exposed.

Keywords: computer vision, 3D models, feature points detection, POSIT, RANSAC.

Parte II

Memoria

Capítulo 1

Introducción

El presente proyecto se enmarca dentro de la línea de investigación de los Sistemas de Transporte Inteligente (STI) desarrollada en el grupo RobeSafe del Departamento de Electrónica de la Universidad de Alcalá.

El objetivo de este grupo es mejorar la seguridad, eficiencia y confort del transporte, mejorando la funcionalidad de los coches y las carreteras utilizando tecnologías de la información.

Para poder alcanzar estos objetivos, es necesario reducir el problema en varios módulos: de percepción o adquisición, procesamiento, control, interfaz con los seres humanos, etc. Existe una gran variedad de sensores utilizados en vehículos comerciales, cuya finalidad es obtener información del entorno para poder actuar en función de la misma. Estos sensores son los ultrasonidos, radar, láser, cámaras de vídeo, infrarrojas, etc. En el caso que nos ocupa, nos centraremos en las cámaras de vídeo, procesando la información con técnicas de visión por computador. Ésta se diferencia de las anteriores en que la riqueza de información proporcionada es superior y permite interpretar mejor la escena completa donde se está trabajando, puesto que con otros sensores nos vemos bastante más limitados.

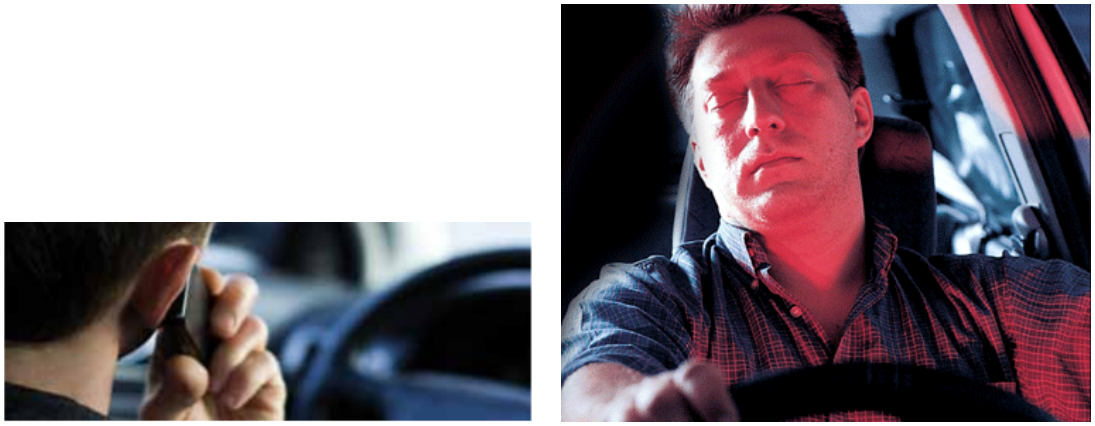
Para mejorar los distintos aspectos relacionados con la conducción, se integran múltiples sistemas. Estos sistemas están orientados fundamentalmente a incrementar la seguridad tanto de los que están dentro del coche como los que están fuera, y también, a evitar el accidente en situaciones peligrosas (seguridad activa) o paliar sus efectos si el accidente ya se ha producido (seguridad pasiva).

El propósito de este trabajo es la implementación de varias mejoras para un software de seguimiento de objetos en el espacio, concretamente de la cara del conductor. Una de las mejoras es la reducción del tiempo de ejecución en una parte del algoritmo, la que estima la pose del objeto en el espacio, y por otro lado, se implementa un sistema de adquisición de malla a través de un scan-láser para generar una máscara de la cara, concreta para cada usuario, más precisa que la que se pueda realizar de manera manual.

El diezmado inteligente de mallas 3D expuesto en esta Tesis de Máster se puede utilizar para otros propósitos, por ejemplo, en la industria de los vídeo-juegos y en general del tratamiento 3D de mallas y texturas simultáneamente. Con la técnica propuesta conseguimos centrarnos en los vértices más relevantes del objeto, dejando de lado aquellos que no tienen mucha información en su zona correspondiente de textura. Este método es utilizable si se tienen mallas 3D con fotografías utilizadas como texturas asociadas a cada punto.

1.1. Motivación

A la vista de las estadísticas ofrecidas por la DGT, uno de cada 3 accidentes mortales es debido a alguna distracción al volante o a la somnolencia. Las principales causas de las distracciones al volante son controlar otros dispositivos (GPS's, teléfonos móviles, la radio), los propios ocupantes del vehículo pueden provocar accidentes al distraer o entorpecer las maniobras al conductor. Existen más causas que pueden provocar distracciones, como encender un cigarrillo en el coche, insectos que entorpezcan la conducción o que sean potencialmente peligrosos que pueden ocasionar que tanto el conductor como los ocupantes centren su atención en espantarlo o hacer que salga del habitáculo. En cuanto a la somnolencia, el hecho de dormir mal el día anterior, el cansancio acumulado durante la actividad diaria, el llevar mucho tiempo sin descansar, u otros factores como enfermedades relacionadas con el descanso, e incluso ciertos fármacos, como los antihistamínicos, pueden provocar una reducción considerable en los reflejos que necesitamos a la hora de conducir un coche. En la figura 1.1 se ilustran gráficamente las 2 causas de accidentes más frecuentes antes comentadas.



(a) Distracción por atención al teléfono móvil (b) Somnolencia

Figura 1.1: Causas principales de accidentes durante la conducción

Las distintas firmas automovilísticas tienen en sus manos, cifras similares de accidentes por las causas antes mencionadas a nivel mundial. Al ver que un tercio de los accidentes es debido a distracciones, se plantea la opción de incluir sistemas que adviertan de estas circunstancias a los conductores para obrar en consecuencia. En la actualidad lo que se está haciendo es detectar estas circunstancias midiendo ciertos parámetros de la conducción con sensores embarcados en el vehículo, y en el momento que se detecta inatención se avisa al conductor mediante algún estímulo visual, sonoro o háptico como son señales visuales en el ordenador de a bordo, pitidos en el sistema de audio del coche, o hacer que el asiento vibre para despertar al conductor. Incluso hay algunas marcas que para llamar la atención del conductor pegan un frenazo corto, lo que aumentan la alerta del conductor, y no afecta demasiado a la trayectoria general del coche, sin peligro para los que vengan detrás por frenar de forma no esperada.

Otro de los motivos por los que se ha realizado esta Tesis de Máster es para comprobar hasta qué punto es útil la inclusión de modelos 3D precisos generados con un scan-láser en un sistema desarrollado por el grupo, con el que se realiza un seguimiento en 3 dimensiones de la cara del conductor, para evaluar su estado de atención en la carretera.

1.2. Objetivos

El objetivo de esta Tesis de Máster se encuentra en la aplicación de unas mejoras para un sistema de seguimiento de caras. El seguimiento de la dirección a la que apunta una cara es fundamental en cierto tipo de aplicaciones, en las que se requiere evaluar dónde mira un usuario. Estos sistemas sirven de base para otros más complejos, como aquellos en los que se evalúa la detección de distracciones o inatención del conductor, los cuales registran estas posiciones a lo largo del tiempo y tienen en cuenta la dirección aproximada a la que la cara apunta para evaluar el centro de atención, junto con la estimación de dirección de la mirada. En este último problema, no entraremos en este trabajo, puesto que excede de los propósitos del mismo.

La primera de las mejoras que se proponen se centra en el algoritmo de cálculo de pose, con objeto de que su ejecución se realice en menos iteraciones por cada frame, lo cual hace que el software se ejecute más rápido.

Otro de los objetivos de esta Tesis de Máster es obtener una base de datos faciales 3D con la ayuda de un scan-láser, para la ayuda al sistema de tracking 3D que se realiza en el software desarrollado por el grupo de trabajo. El problema planteado al utilizar este instrumento es que, al ser la malla de puntos devuelta por el escáner muy densa (dependiendo del modo de adquisición, de 4000 a 86000 puntos por toma), hay que reducir la información de forma que solo nos quedemos con la información más característica en un sentido visual, esto es, zonas con gran contraste donde podamos encontrar esquinas en la imagen. Con estas esquinas, se pasa a realizar un diezmado de la malla 3D para quedarnos con los puntos más significativos. Por último se realiza la unión de las 3 vistas calculando la matriz de transformación a partir de la selección de puntos correspondientes.

Capítulo 2

Estado del arte

2.1. ADAS (Advanced Driver Assistance Systems)

El término ADAS engloba a todos los sistemas que intervienen en la asistencia a la conducción. Este término tan general puede dividirse en 2 términos algo más concretos como la seguridad del coche y la seguridad de la carretera, que a su vez engloban diversos subsistemas. Algunos de estos subsistemas son los siguientes:

- In vehicle navigation system
- Adaptive cruise control (ACC)
- Lane departure warning systems
- Lane change assistance
- Collison avoidance systems
- Intelligent speed adaptation
- Night Vision
- Adaptive light control
- Pedestrian protection
- Automatic parking
- Traffic sign recognition
- Blind spot detection
- Driver inatention and drowsiness detection
- Car2car communication
- Hill descent control
- Environment and fuel saving

... y otros subsistemas que terminarán haciendo la labor del conductor más placentera, cómoda y segura a lo largo de los años con su penetración en el mercado y su progresiva generalización en forma de extras. Un listado más exhaustivo de estos sistemas así como información más detallada acerca de los mismos se encuentra en [1, 2].

Explicar todos y cada uno de estos sistemas es algo que se escapa de los objetivos de la Tesis de Máster. De todos ellos, analizaremos en mayor profundidad los sistemas de detección de distracciones y somnolencia, los cuales están siendo desarrolladas y se están empezando a implantar como extra en varias firmas automovilísticas, sobre todo en los automóviles de gama alta.

La implantación de estos sistemas en automóviles requiere de la utilización de varios sensores para la obtención de datos. La naturaleza de los mismos es variada, y la fusión de varias fuentes es conveniente para reducir errores en la toma de datos y por tanto la toma de decisiones, con objeto de asegurar la robustez del sistema.

2.2. Sistemas comerciales de detección de distracciones y somnolencia en conductores

Distintas firmas automovilísticas de renombre están apostando en la actualidad por incluir estas mejoras en sus vehículos de gama alta. Cada una de estas firmas tiene distintos puntos de vista acerca de cómo afrontar este problema. Las soluciones planteadas para resolverlo utilizan varios tipos de sensores, como puede ser cámaras de visión, sensores láser, etc. A continuación pasamos a describir las soluciones adoptadas por las distintas marcas.

2.2.1. City Safety

Este sistema ha sido desarrollado por la marca Volvo. Tiene como finalidad frenar el vehículo para evitar colisiones frontales en ciudad, provocadas fundamentalmente por despistes en el conductor. Este sistema se plantea como freno de emergencia en situaciones de riesgo, pero no se plantea nunca como un sustituto de los frenos. También se evita que el coche tome el control y se haga uso intensivo de este sistema para evitar interferencias con la forma de conducir normal, puesto que la frenada es corta y seca para quedarse a unos 40 cm del vehículo que está delante.

El uso de este sistema está limitado por ciertos factores que se explican en [3], y que algunos de los más importantes se detallan a continuación:

- Para el cálculo de distancias utiliza un láser de infrarrojos, que se basa en la medición de elementos que reflejan bien este tipo de luz, como por ejemplo las matrículas, objetos que estén más abajo no serán detectados adecuadamente.
    - Si el vehículo que está en frente está muy sucio la detección no se realizará correctamente y por tanto el sistema fallaría.
    - Hay países en los que las matrículas no son reflectantes, con lo que en principio esto no funcionaría.
- Si el vehículo se desplaza lateralmente, la detección de la luz reflejada puede ser errónea y la distancia mal calculada, con lo que el sistema puede no evitar el choque a tiempo.
- En condiciones de adherencia resbaladizas, la distancia de frenado es mayor, por tanto el sistema podría quedarse a menor distancia o llegar incluso a chocar, aunque el propio

sistema utilizará mayor fuerza de frenada junto con el control de estabilidad para mantener al vehículo correctamente.

- Si el vehículo tiene una carga que sobresale o la matrícula inclinada o más baja puede calcular mal la distancia restante.
- Si el vehículo precedente está dando marcha atrás, este sistema se desactivará, puesto que tiene como misión evitar colisiones de vehículos estacionados o caminando en la misma dirección. Si se acciona la marcha atrás en el propio vehículo el City Safety será desactivado.
- El límite inferior de velocidad se sitúa en 4 km/h, para evitar interferencias con el aparcamiento.
- Se da prioridad siempre a la decisión del conductor, en caso de que este también esté frenando, acelerando o girando el volante, aunque la colisión sea inevitable.
- Si los sensores son obstaculizados, el sistema se inhabilita automáticamente.

Este sistema está siendo implantado en los todo-terrenos de la marca y en los vehículos de muy alta gama. En la figura 2.1 se muestra un esquema básico de funcionamiento del sistema.

Figura 2.1: City Safety - Volvo

2.2.2. Driver Alert Control

Este sistema también ha sido diseñado por Volvo y consiste en una evaluación del estilo de conducción a través de la desviación que se produce en el carril por el que se va, según se indica en la página de la firma, localizada en [4].

En este sistema se tiene una cámara en la luna delantera, apuntando hacia la carretera, con la cual se evalúa si ha habido un cambio involuntario de carril sin motivo aparente, causa evidente de cansancio.

Los sistemas por los cuales se alerta al conductor son un pitido y también una señal visual a través del ordenador de a bordo.

No se evalúan aceleraciones ni otros parámetros que en otros sistemas sí se evalúan, no se aprende la forma de conducir de cada usuario, si no que solo se concentra en evaluar cómo se sale del carril por el que se circula basándose en sistemas LDW (Lane Departure Warning), en lugar de basarse en parámetros que evalúan el estado del conductor directamente. Los sistemas

LDW se basan en la detección robusta de líneas de carretera, y se utiliza como base para otros sistemas de ayuda a la conducción, como por ejemplo la implementación de pilotos automáticos del coche en proyectos como autopía, detección de salida involuntaria de carril (como se usa en este caso) y otras posibles aplicaciones.

2.2.3. Attention Assist

Este sistema está desarrollado por la casa Mercedes-Benz. En él no se utiliza la visión artificial como método para evaluar visualmente el estado del conductor, sino que se utilizan varios sensores distribuidos por el vehículo para estimar varios parámetros como la velocidad, aceleración, movimientos laterales etc. Este sistema aprende la manera de conducir del usuario en estado normal para, una vez aprendido este patrón de conducta, sea capaz de diferenciar la desviación del mismo y avisar tanto acústica como visualmente del hecho de que se ha detectado un comportamiento anómalo provocado por la somnolencia o las distracciones al volante. Más información sobre este sistema se puede encontrar en [5]. En la figura 2.2 se muestra un esquema en el que se muestran los distintos puntos en los que se centra el sistema y de los cuales recoge información, para el entrenamiento y para la evaluación de el nivel de atención del conductor.

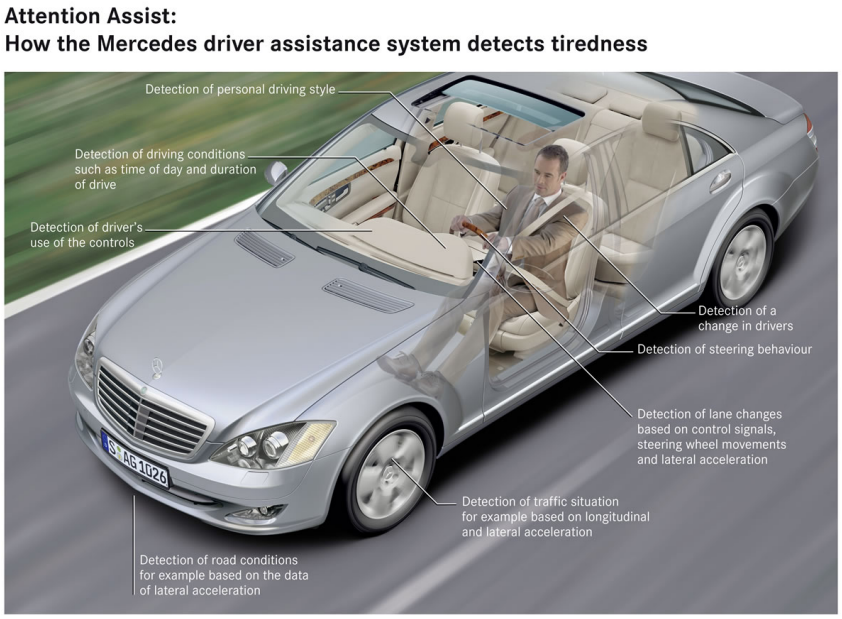


Figura 2.2: Attention Assist de Mercedes - Benz

2.2.4. BMW Group Research

Este grupo de desarrollo de la firma alemana, en colaboración con la universidad de Würzburg ha desarollado un sistema que informa al conductor de su propio estado en todo momento. Se implementa a partir de una cámara que recoge los movimientos del ojo cuando mira de frente y a partir de ahí estima si el usuario se encuentra en alguno de los estados siguientes: vigilia, reducción de la atención, cansancio o sopor (estado más peligroso, cercano al microsueño). En

este caso los indicadores utilizados son luces de color rojo en caso de detección de somnolencia y señales acústicas para el aviso en caso de sopor. Más información sobre el sistema se puede encontrar en [6] y en la página web del BMW Group Research. En la figura 2.3, se muestran 2 imágenes de ejemplo del prototipo desarrollado.

Figura 2.3: Sistema de detección de distracciones de la firma BMW

2.2.5. APCS (Advanced Pre-Colision System) with Driver Attention Monitor

Este sistema está englobado dentro de la gama de sistemas Lexus LS Hybrid Safety and Security [7]. Utiliza una cámara de infrarrojos, apuntando hacia el conductor, para monitorizar la dirección de la cara cuando el coche está en marcha. El método que se utiliza para avisar al conductor, es, en primer lugar pitidos, y si no, ya el coche actúa sobre los frenos, pegando frenazos cortos para hacer reaccionar al conductor si este está dormido por mucho tiempo porque se detecte una salida de carril y prepara los frenos para una frenada de emergencia y reducir los efectos de un choque en caso de que este se produzca.

Una captura de pantalla del frontal del sistema en una animación flash de presentación del sistema en la web de Lexus se muestra en la figura 2.4.

Figura 2.4: Lexus APCS con Driver Attention Monitor

2.3. Aproximaciones al problema de seguimiento de caras y estimación de pose

En esta sección se ilustrarán las distintas aproximaciones al problema realizadas por diversos autores, con los distintos métodos que se han venido utilizando desde hace aproximadamente 15 años hasta la actualidad. Se dará un repaso breve a todas ellas y se estudiaran con algo más de profundidad los casos en los que se tengan sistemas similares al que se trata en esta Tesis de Máster, que son, basados en modelos y marcas faciales. Más información de la que se expondrá a continuación se encuentra en [19].

El problema de la estimación de la pose de la cara y del seguimiento de la misma en una imagen, ha supuesto un gran reto a lo largo de muchos años en el ámbito de la tecnología y más concretamente de la visión artificial. Para ello se han realizado distintas aproximaciones, cada vez más complicadas a medida que la capacidad computacional de los equipos ha ido dando margen para refinar los algoritmos.

Algunas de las estrategias seguidas para abordar este problema se resumen en la tabla 2.2. Esta tabla ordena los métodos empezando por los más simples y terminando por los más complicados y mezclas de los mismos. A continuación se analizan las distintas aproximaciones en un poco más de detalle, analizando las ventajas e inconvenientes de cada una de ellas.

- En los métodos **basados en apariencia** se compara la imagen actual de la cabeza con una base de datos de imágenes etiquetadas para encontrar la que da el mínimo error. Para el cálculo del error se utilizan varias métricas o incluso, en versiones refinadas, se utilizan diversos tratamientos en las imágenes para aumentar el ratio de reconocimiento y por tanto el error en la estimación en la pose.
  La ventaja principal es que es que la base de datos puede ser ampliada fácilmente añadiendo nuevas imágenes para cubrir más posibles poses. La otra ventaja que tiene este método es que no se requiere muestras negativas en la base.
  Existen varios inconvenientes asociados con estos métodos. En primer lugar si la base de datos de gestos es grande, el tiempo para la realización de comparaciones es muy alto. Se asume que la región de la cara está bien detectada dentro de la imagen, lo cual no siempre es cierto e influye en la calidad de la estimación. Si no se utiliza interpolación, las poses que se pueden llegar a estimar son discretas. Además, al basarse en la apariencia, para cada usuario se tiene que tener una base de datos personalizada, con el inconveniente del cambio de iluminación, apariencia, etc. Se han realizado algunos intentos para mejorar estos últimos aspectos extrayendo imágenes de bordes o transformadas intermedias y aplicando diversas métricas.

- Los métodos **basados en arrays de detectores** tienen como particularidad que se tiene un detector específico entrenado para cada pose, y se lanzan en paralelo en cada una de las iteraciones. Se basan en la suposición de que los detectores no entran jamás en conflicto. La detección es discreta a menos que se utilice interpolación. No se requiere tener localizada la cara dentro de la escena puesto que se asume que los detectores ya hacen ese trabajo.
  La ventaja es que no hace falta tener una estimación inicial de la localización de la cara, la detección y la estimación de la pose se realizan simultáneamente y funcionan bien con

- imágenes en alta y baja resolución.
  En cambio, se necesita una gran base de datos para cada pose específica, además que se necesita la inclusión de muestras negativas, lo cual aumenta considerablemente el conjunto de entrenamiento. El tiempo de proceso incrementa linealmente al número de grados de

libertad y resolución de la variable estimada, por lo que es imposible la implementación en tiempo real de un sistema preciso basado en este tipo de algoritmos. El tiempo de ejecución puede verse en parte reducido poniendo estos detectores en cascada, como se hace en Viola & Jones [18].

- Los métodos de **regresión no lineal** se basan en la aplicación de un modelo de regresión a una imagen o cara base, para efectuar las rotaciones adecuadas. El problema fundamental aquí es encontrar la función adecuada que mejor simule las nuevas poses y apariencias. Además, estas herramientas de regresión aplicadas a las imágenes, son muy lentas, puesto que al ser las imágenes grandes, la dimensionalidad del problema de regresión es muy alta. Se puede utilizar PCA o métodos similares para reducir la dimensionalidad del problema, pero el problema fundamental sigue estando en encontrar la función adecuada. Las redes neuronales han sido las que más se han probado para la búsqueda automática de funciones. La ventaja principal es la sencillez en su implementación, entrenamiento debido a que lo utilizado son redes neuronales. La principal desventaja es encontrar el conjunto adecuado para el entrenamiento y la necesidad de dar previamente una estimación de la localización de la cara dentro de la imagen.

- Los métodos **empotrados múltiples** (Manifold embedding) utilizan transformadas de reducción de dimensionalidad tales como PCA y sus variantes no lineales KPCA con kernel gausiano. Con estos métodos se extraen las características más básicas de la imagen y se comparan con un conjunto de entrenamiento de poses en una dimensionalidad mucho más reducida. Combinando métodos pueden utilizarse además como entrada para clasificadores y aumentar la eficiencia y precisión de los métodos.
  El gran problema de estos métodos es que no son supervisados, con lo que pueden no funcionar bien si la base para generar el espacio PCA no ha sido la adecuada. Otro punto débil es que la gran variedad de situaciones dadas en condiciones reales hacen que el sistema pierda precisión si se pretende entrenar para todos los casos posibles o que deje de funcionar en los mismos si no se entrena para estos escenarios.

- En el caso de los **modelos flexibles** en este caso se tiene un modelo genérico de la cara. Este modelo se coloca sobre la imagen, y se va deformando iterativamente hasta que encuentra la mínima distancia entre puntos característicos de la cara.
  La primera aproximación realizada para la identificación de la pose es que, para cada posible pose de la cara, se crea un modelo y posteriormente se realiza una comparación. Al basarse en puntos característicos y no en toda la imagen, el sistema es menos dependiente del usuario, pero las poses son discretas y si se quiere precisión el número de comparaciones a realizar es muy alto.
  Una mejora a esta aproximación se basa en tener en cuenta la proyección que se realiza sobre la cámara en 2D. El espacio de entrenamiento se reduce con PCA, y además existen mejoras de asociar el punto con textura con lo que el posterior seguimiento, si se realiza es más robusto. Luego existen otras mejora adicionales sobre estos métodos, aunque tienen la limitación de que si alguno de los puntos de interés se oculta, se pierde la precisión, o el sistema puede fallar directamente.

- Los **modelos geométricos** se basan en relaciones geométricas que se pueden considerar en la cara. La primera de ellas es la simetría que existe entre ambos lados de la cara. Por otro lado existen relaciones geométricas entre puntos característicos de la cara, como son la nariz, las esquinas exteriores de los ojos y de la boca. Se asume que son coplanares los puntos de los ojos y los de la boca aproximadamente y realizando operaciones sobre la posición relativa en la imagen de estos puntos se puede averiguar la pose de la cara. Otra

aproximación es calcular el punto de fuga formado por líneas paralelas en la cara y calcular a partir de ahí la orientación 3D.

- En cuanto a los **métodos de seguimiento**, se basan en la utilización del movimiento relativo de la cabeza entre frames para estimar las variaciones en la pose. Una explicación más amplia de estos métodos se ofrecerá con más profundidad en la sección 2.3.1, puesto que son los métodos sobre los que se sustenta el sistema base sobre el que se va a hablar en esta Tesis de Máster.

- Por último, los **métodos híbridos** utilizan diversas variantes de los anteriores, sobre todo utilizando los de seguimiento con alguno de los otros anteriores más sencillos, para reducir la carga computacional a pesar de la complicación del algoritmo global.

Aproximación	Trabajos representativos
Métodos basados en plantillas de apariencia	
▶Comparación de imágenes	Error cuadrático medio [20], Correlación cruzada normalizada [21]
▶Comparación de imágenes filtradas	Gabor wavelets [22]
Array de detectores	
▶Machine learning	SVM [23], Cascadas Adaboost [24]
▶Redes neuronales	Router Networks [25]
Métodos basados en regresión no lineal	
▶Herramientas de regresión	SVR [26, 27], Feature-SVR [28]
▶Redes neuronales	MLP [29–31], Redes convolucionales [32], LLM [33], Feature-MLP [34]
Métodos empotrados múltiples (Manifold Embedding)	
▶Subespacios lineales	PCA [22], Autoespacios de Pose [35]
▶Subespacios kernelizados	KPCA [37], KLDA [37]
▶Subespacios no lineales	Isomap [38, 39], LE [40], LLE [41], Biased-Mainfold Embeding [42], SSE [43]
Modelos flexibles	
▶Descriptor de características	Elastic graph matching [44]
▶Modelos de apariencia activa	ASM [45], AAM [46, 47]
Modelos geométricos	
▶Caracterísitcas faciales	Métodos en el plano y en 3D [48], geometria proyectiva [49], punto de fuga [50]
Métodos de seguimiento	
▶Seguimiento de características	RANSAC [51, 52], Mínimos cuadrados ponderados [53], Restricciones
▶Seguimiento de modelo	psicológicas [54]
▶Transformaciónes afínes	Búsqueda de parámetros [55–57], Mínimos cuadrados [58], Plantillas dinámicas [59]
▶Filtros de particulas basados en	Constancia de brillo y profundidad [60]
apariencia	Difusión adaptativa [61], Modelo de estado dual y lineal [62]
Métodos híbridos	
▶Geométrico y de seguimiento	Configuración de características locales + seguimiento SSD [63, 64]
▶Plantillas de apariencia y seguimiento	AVAM con fotogramas clave [65], Template Matching with particle fitlering [66]
▶Regresión no lineal y seguimiento	SVR+filtro de partículas 3D [62]
▶Empotrados múltiples y seguimiento	Comparación PCA+HMM de densidad continua [67]
▶Otras combinaciones	Regresión no lineal+geométrica+filtro de partículas [68], KLDA+EGM [37]

Tabla 2.2: Tabla resumen de las distintas aproximaciones seguidas para resolver el problema de la pose

2.3.1. Métodos de seguimiento

Estos métodos trabajan con el movimiento relativo de la cabeza a lo largo de los frames para ir calculando la pose de la cara en cada instante. Estos sistemas tienen un alto grado de precisión pero una buena inicialización del sistema es necesaria, y la primera detección es fundamental, con la consiguiente restricción de la zona de la imagen donde se encuentra la cabeza. Algunos de los métodos utilizan mínimos cuadrados lineales o ponderados o resolución SVD (a través de autovectores y autovalores) para estimar la pose.

Lo más frecuente es utilizar la correlación de unos parches en un entorno cercano a partir de puntos característicos, y a partir de estos, se reconstruye la pose utilizando una perspectiva blanda [69], en lugar de una perspectiva normal, como utiliza POSIT (que se verá en las técnicas utilizadas), para reconstruir las matrices R y T, de cambio de referencia de la cámara al sistema de referencia global.

Otra opción descrita en [53] aproxima la rotación a una transformación afín, entre frames consecutivos, cuando esta es pequeña (se supone que el desplazamiento entre 2 frames de una secuencia es suficientemente pequeño como para hacer esta suposición) y con esto se pueden estimar 2 grados de libertad usando mínimos cuadrados ponderados .

Existen varios métodos basados en técnicas complejas para aproximar mejor la relación entre frames llegando al punto del uso de SIFT (Scale-Invariant Feature Transform) [70] para encontrar pares de puntos correspondientes, o también sistemas que hacen uso de un modelo predeterminado [54, 63], o un matching estéreo con un filtrado RANSAC [52]. Una combinación de estos 2 últimos es la que se tiene en el software con el que se trabaja en esta Tesis de Máster.

En el caso de los algoritmos que usan una malla 3D rígida, se realiza una correspondencia entre los puntos 3D de la malla y la textura, o en este caso, las secuencias de entrada.

También existen aproximaciones en las que se usan sistemas estéreo para obtener con mayor precisión la pose en 3D al tener bien calibradas las cámaras y saber la correspondencia entre las mismas [48, 60].

2.4. Sistemas de seguimiento de caras y evaluación de estado del conductor.

En esta sección se pasará a describir algunos sistemas comerciales utilizados para la evaluación automática de la pose y del estado de consciencia del conductor, y por último otros sistemas subvencionados por el gobierno para la implementación de un sistema capaz de monitorizar a un conductor para vehículos de transporte.

Para implementar este tipo de sistemas, es necesario la localización de la cara en la escena en todo momento, así como la dirección de la mirada, que es donde se presta atención. Para obtener esta información se hace necesaria la utilización de sensores adecuados. En los casos presentados en esta sección el sensor utilizado por excelencia son las cámaras de vídeo.

2.4.1. Sistemas de seguimiento comerciales: Seeing machines

2.4.1.1. Facelab

Este software, desarrollado por la compañía seeing machines, tiene como característica principal que es capaz de realizar detección y seguimiento de ojos y cara.

En la web del fabricante [8] se, dice que es lo suficientemente robusto para aguantar en diversidad de escenarios como pruebas en cámaras montadas en un coche, procesamiento en un portátil, etc. También se dice del programa que es capaz de trabajar con ojos distintos, lentillas, gafas y otras perturbaciones.

Un ejemplo de captura de pantalla del programa la podemos ver en la figura 2.5.

Como características principales de este programa, se dice que es capaz de proveer el suficiente soporte para realizar labores de seguimiento tanto en ojos como en cara, siendo capaz de detectar características significativas, tales como:

1. Cabeza
   - Posición y rotación
2. Ojos
   - Posición y rotación del ojo
   - Dirección de la mirada hacia la pantalla y desde el modelo
   - Distancia de vergencia del ojo
   - Eventos sacádicos
3. Párpados (por cada párpado)
   - Parpadeo (frecuencia, eventos y duración)
   - Apertura
   - Índice de fatiga PERCLOS
4. Características faciales
   - Labios
   - Otras características de los párpados
5. Datos temporales
   - Registros por número de frame
6. Rotaciones de la cabeza
   - Seguimiento y recuperación de $\pm 90^{o}$ en el eje
   - $\pm$ 45^{o} en el eje x
7. Rotaciones de la mirada
   - $\pm 45^{o}$ en eje y

- $\pm 22^{o}$ en eje x

8. Recuadro de la cara
   - Rango horizontal de seguimiento hasta 0.35 m
   - Rango vertical de seguimiento hasta 0.23 m
   - Rango de distancia hasta 0.6 m
9. Precisión del seguimiento
   - $\pm$ 1 mm en translación y $\pm$ 1 ^{o} en rotación de la cabeza
   - $0.5^{o} \pm 1^{o}$ en la dirección de la mirada

Figura 2.5: Ejemplo de funcionamiento de FaceLab

2.4.1.2. DSS (Driver State Sensor)

Este sistema, desarrollado por la misma empresa que el anterior, tiene como característica que es específico para la evaluación de estado del conductor, mientras que el otro software es genérico, no es específico para el mundo de la automoción. Este sistema mide la dirección hacia donde tiene puesta la atención el conductor evaluando tanto la dirección de la cara como la de la mirada. Este sistema ha sido descrito tanto en la página web de la empresa desarrolladora como en otros medios, los cuales se pueden ver en [9, 10]

En el pack se incluye tanto el software como hardware necesario, y dentro de este último, tanto el de proceso como el de adquisición, basado en cámaras. La parte hardware se muestra en la figura 2.6. Este dispositivo se instala en el vehículo allí donde se quiera utilizar, normalmente vehículos de transporte, como camiones.

Las principales características de este sistema son las siguientes:

- Puede evaluar la fatiga del conductor
- Detecta distracciones
- Es totalmente automático, sin contacto
- Trabaja en tiempo real, además de realimentarse continuamente.

Algunas de estas características, son heredadas del programa anterior, FaceLab.

Figura 2.6: DSS (Driver State Sensor)

2.4.1.3. FaceAPI

Por último, como base de los 2 anteriores productos está la plataforma de desarrollo ofrecida por la misma compañía [11], para que se puedan utilizar los datos detectados por la misma a más alto nivel. Proporciona la base para calcular todos los parámetros que se ofrecían internamente en FaceLab, poder extraerlos con facilidad y poder desarrollar a partir de ahí.

Un ejemplo de la aplicación de este sistema, dentro de DSS, es el que se muestra en la figura 2.7.

En el folleto del FaceAPI, podemos encontrar como características destacadas las siguientes:

- Seguimiento y estimación de la pose de la cabeza
- Detección de marcas faciales
- Seguimiento de expresiones faciales
- Extracción de la textura de la cara

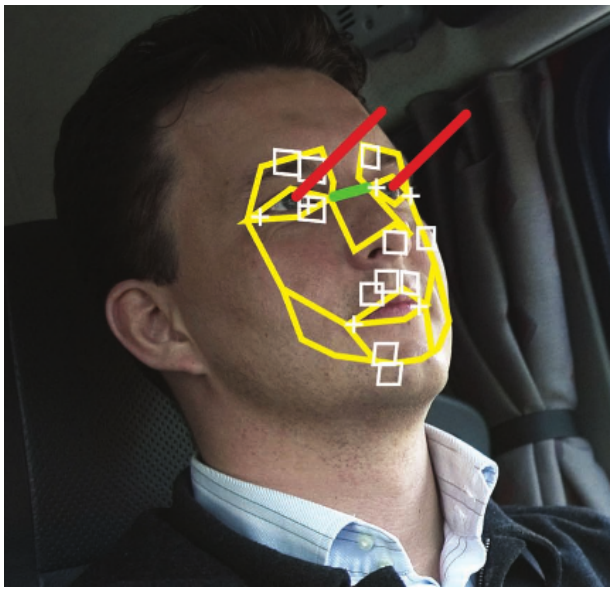


Figura 2.7: Ejemplo de aplicación de FaceAPI en DSS

2.4.2. Proyecto Cabintec

En este proyecto se pretenden realizar sistemas inteligentes que detecten el comportamiento del conductor, y estudiar tanto el comportamiento del vehículo como del conductor en instantes previos a un accidente, particularizando para el caso de vehículos de transporte. Se trata de un proyecto estratégico y singular financiado por el gobierno de España, que cuenta con la participación de universidades, empresas, centros de investigación y asociaciones del sector. Entre las universidades, se encuentra la Universidad de Alcalá. El proyecto es tan amplio, que se ha dividido en varios subproyectos:

- Hito : pretende garantizar una conducción segura a partir de tecnologías de la información.
- Alerta: prevención de falta de atención del conductor, prevención de somnolencia y detección de gestos inadecuados.
- Señal: señalización, que indica al conductor información acerca de las señales de su entorno.
- Estable: condiciones de estabilidad de vehículos pesados en diferentes condiciones para prevenir vuelcos.
- Registra: se centra en el diseño de un dispositivo de registro que facilite la reconstrucción de accidentes, similar a las cajas negras.
- Asiste: sistema que presentará la información del estado del vehículo, dispositivos y ayuda a la conducción
- Conecta: difusión y seguimiento del proyecto, tanto técnica como económicamente.

La Universidad de Alcalá participa en los subproyectos Alerta e Hito. El trabajo realizado en esta Tesis de Máster está ubicado en el subproyecto Hito. Más información del proyecto Cabintec global se puede encontrar en [12].

2.4.2.1. Hito

Esta sección se centra en el subproyecto Hito de Cabintec, el cual, como se ha dicho antes, pretende ayudar a una conducción segura con la ayuda de las tecnologías de la información.

El grupo Robesafe del Departamento de Electrónica de la Universidad de Alcalá es uno de los participantes en este proyecto. En nuestro caso, el objetivo del sistema implementado ha sido la obtención de la dirección de la cara para obtener una dirección aproximada de dónde se está fijando la atención. Más información sobre este sistema se encuentra ampliada en el capítulo 5 y en el artículo asociado [17]. Otras entidades del convenio analizan otros parámetros, como por ejemplo localizar la dirección del volante, posición de los brazos, etc.

Los 3 puntos fundamentales englobados dentro del proyecto Hito son los siguientes:

- Metodología de validación de los sistemas embarcados.
- Validación en el simulador
- Acordar los requisitos necesarios para todos los subproyectos.

Al principio, antes de embarcar estos sistemas en uno real, se realizan las pruebas en el simulador, para evitar de riesgos reales innecesarios. Una vez se han recogido datos suficientes, se pasa a la implementación posible en sistemas embarcados en vehículos reales.

En la figura 2.8 se muestra el simulador utilizado para este proyecto. En este simulador las distintas universidades toman datos, vídeos e imágenes para realizar sus propios sistemas de detección de inatención sobre una gran variedad de conductores.

Figura 2.8: Simulador Cabintec - Hito

Capítulo 3

Métodos de adquisición de modelos 3D: Scan-láser 3D

3.1. Introducción

Existen varios métodos para la generación de modelos en 3D. La primera aproximación que se puede realizar es mediante visión computacional. Mediante un sistema de adquisición estéreo, compuesto por 2 o más cámaras sincronizadas, se puede realizar, a partir de los parámetros intrínsecos y extrínsecos de la cámara, una reconstrucción aproximada del entorno 3D que rodea a la cámara. En esta reconstrucción, el error de estimación de la profundidad es proporcional a la distancia a la que se encuentra el objeto, y también es dependiente de la separación entre cámaras.

La precisión obtenida por este método no es demasiado alta, con lo que si queremos tener un sistema que nos sirva como referencia, para tener más precisión, tendremos que utilizar técnicas basadas en láser.

En el caso de las técnicas de adquisición basadas en láser, se obtiene mayor precisión, puesto que se utiliza el tiempo de vuelo de la señal y, a partir de ahí, se sabe la distancia a la que ha ido el rayo hasta un punto determinado en ida y vuelta. Realizando un barrido vertical con láser, se tiene no solo la reconstrucción en un plano sino la reconstrucción completa de la superficie 3D que sea capaz de reflejar hacia el escáner, es decir, el objeto.

Este último aspecto es importante, puesto que implica que para extraer la información 3D del objeto hay que realizar varios escaneos, y luego encajar, a partir de puntos característicos, las distintas vistas, puesto que la posición del objeto respecto de la cámara, en cuanto este se mueva un poco, va a cambiar, por tanto este tipo de ajustes van a ser necesarios.

3.2. Tipos de escáneres

3.2.1. De contacto

Como su propio nombre indica, existe contacto físico entre el objeto y el dispositivo que toma las medidas, con el consiguiente problema de que el objeto escaneado puede ser dañado en el proceso. Se utiliza en la fabricación de piezas y puede ser muy preciso. Al basarse en dispositivos mecánicos, tales como brazos robóticos, para realizar el escaneo, hace el proceso muy lento en

comparación con escáneres basados en láser.

3.2.2. Activos / de no contacto

Estos escáneres se basan en el láser como método de adquisición de datos. Se llaman activos porque no solo captan información sino que emiten el rayo láser para realizar la medida. La radiación utilizada no tiene por qué ser exclusivamente luz sino electromagnética con otras frecuencias o incluso con ondas sonoras. Como se explicará en la sección 3.3, el método de medida se hace calculando el tiempo que tarda la señal en ir y volver desde el objeto escaneado, teniendo en cuenta que la velocidad de la luz es constante y conocida.

3.3. Fundamentos (Activos)

El escáner láser funciona como un radar, pero en este caso se sustituyen las ondas electomagnéticas o sonoras (sonar) a determinadas frecuencias, por luz generada por láser, la cual es coherente y representa una buena fuente de luz, mejor que la generada por diodos u otros dispositivos con un haz muy abierto y una dispersión frecuencial más alta. Basándose en el tiempo de vuelo, esto es, en el tiempo que tarda una señal en propagarse hacia un objeto, rebotar y ser captada por el receptor, podemos calcular distancias.

El diodo láser genera un haz fino. Este haz se dispersa con una lente cilíndrica, lo cual hace que el rayo se disperse y genere una franja horizontal. Esta luz reflejada por el objeto se capta por un detector y se pasa a realizar el proceso de triangulación para saber exactamente las coordenadas. El triángulo lo forman el emisor, el receptor y el objeto, y las distancias entre emisor y receptor. Por último, para realizar el barrido vertical, se utiliza un espejo que rota para ir modificando el ángulo con el que el rayo se refleja e ir modificando la altura a la que el rayo impacta contra el objeto.

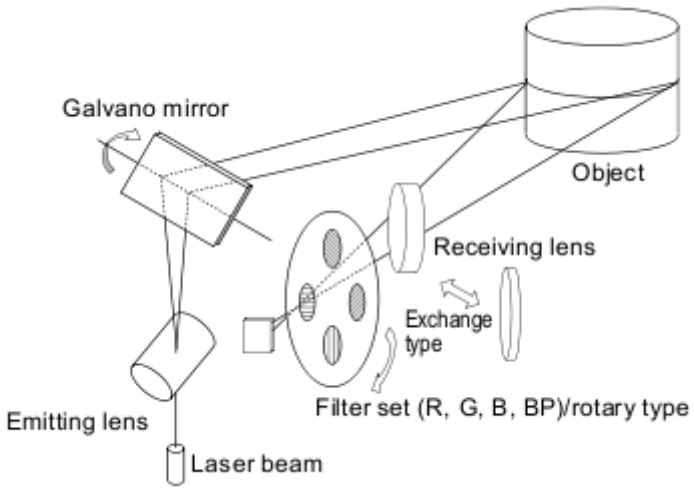


Figura 3.1: Principio de funcionamiento del escáner

Con esto no podemos asegurar que se escanee el objeto completamente, puesto que, solo se detectan las partes que es capaz de reflejar el objeto hacia el CCD detector. Por tanto, la

composición completa del objeto requerirá de la repetición de este procedimiento varias veces y hallar las matrices de rotación y translación que lleven las diferentes tomas a una toma común.

3.4. Konika Minolta Vivid 910i

Este es el scan-láser (escáner activo) que se va a utilizar para recoger los datos 3D. A continuación expondremos algunas de sus características, como los accesorios que incluye, las características técnicas, el software que utiliza y cómo hacer que funcione en el ordenador donde se realicen los escaneos.

Figura 3.2: Vivid 910i

3.4.1. Accesorios

Los accesorios que incluye este hardware son:

- El trípode para poder soportar el escáner y así poder hacer tomas correctas, sin que el escáner se mueva, y para poder regularlo con las distintas manivelas en los distintos ángulos sin apenas esfuerzo y de manera precisa.
- El soporte de calibración (opcional)
- Lentes con distintas distancias focales, para tomar distintos tipos de imágenes, según la necesidad.
- Regulador de balance de blanco, a la hora de realizar la foto.

Figura 3.3: Accesorios para la familia de scáneres vivid

3.4.2. Software

El software básico para el manejo del escáner consta del controlador para el adaptador de bus SCSI a USB y el más importante, el programa de adquisición: Poligon Editing Tool. Con este software no solo se adquieren los modelos y la imagen correspondiente al último instante del escaneo, sino que además es capaz de realizar operaciones con las mallas, como diezmarlas con métodos convencionales, triangularlas, editarlas, rellenar agujeros en las mismas, etc. Además se pueden realizar alineamientos entre mallas con distintos puntos de vista (sin extraer la información de la matriz de transformación), para poder luego fusionar las mallas teniendo una referencia común. Tienen la posibilidad de exportar la información 3D a formatos propietarios y a formatos más estándar.

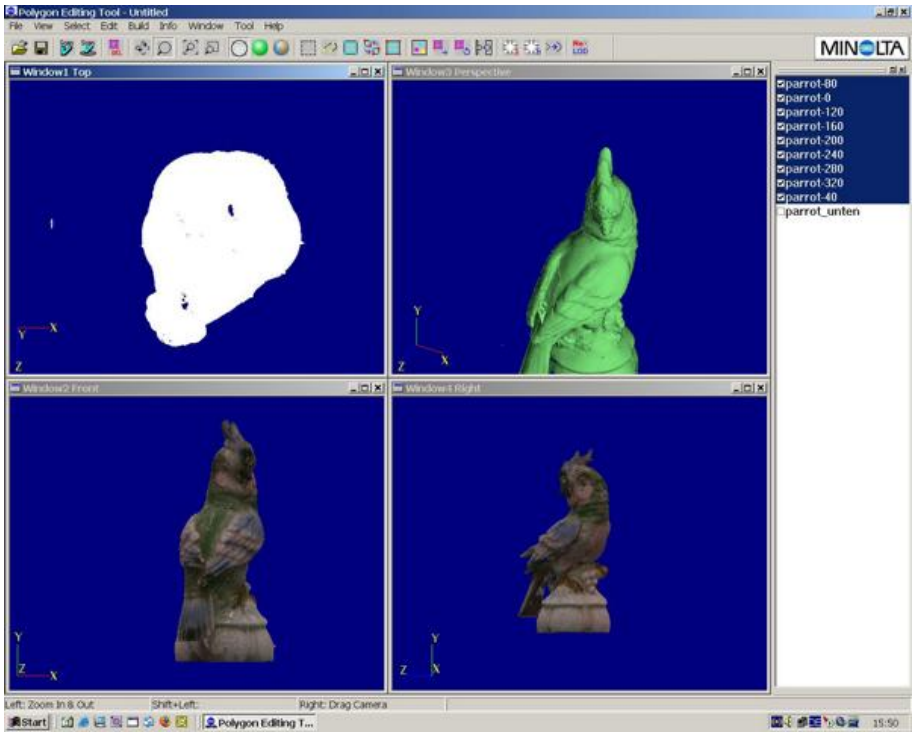


Figura 3.4: Poligon editing tool

3.4.3. Captura de una toma

Para realizar una toma una vez instalado el software adecuado, se han de seguir ciertos pasos:

1. Se ha de montar el escáner sobre el trípode, fijarlo adecuadamente, conectarlo al ordenador y a la corriente eléctrica.
2. Si la conexión ha sido correcta, en la pantalla del escáner aparecerá la palabra REMOTE.
3. Entonces arrancaremos el ordenador, puesto que si conectamos el scan-láser después de arrancar windows, la conexión remota no será correcta y se producirá un fallo de acceso al escáner cuando se intente realizar la captura.
4. La llave USB que contiene la licencia del programa ha de estar colocada antes de su ejecución, puesto que si no, el programa no arrancará y no se podrá realizar la toma.
5. Una vez arrancado windows y viendo que en la pantalla del propio escáner aparece la palabra REMOTE se procede a arrancar el programa de adquisición. Dentro de él hay que ir a la opción de importar desde escáner, la cual está contenida en Archivo->Import->Scan->One Scan / Step Scan, o también desde la barra de herramientas
6. En la pantalla de captura, si no da ningún error propio del escáner o del programa, con el botón AF podemos ver lo que se va a adquirir por la cámara para tener una referencia previa y colocar el objetivo antes de realizar la captura. También antes de realizar la captura se pueden modificar parámetros del escáner tales como resolución, tipo de filtrado del ruido de medida, rellenado automático de agujeros, etc.
7. Para realizar el escaneo, se pulsa al botón Scan, y si ha resultado satisfactorio, esto es, no hay demasiados agujeros en la toma, la resolución es correcta y la malla obtenida también, se procede a guardar la vista. Si no es la última, se puede continuar realizando escaneos, y se guardará con el mismo nombre base variando el número final entre cada toma.

Capítulo 4

Técnicas empleadas

En este capítulo se hace un repaso de las técnicas de visión computacional y de transformaciones espaciales para manipulación de datos 3D utilizadas en esta Tesis de Máster.

4.1. Estimación de modelo

4.1.1. RANSAC (RANdom SAmple Consensus)

Este método se utiliza cuando tenemos una serie de muestras de una observación, y se pretenden extraer las muestras que siguen un modelo. Las muestras que no siguen un modelo se denominan outliers, mientras que las que sí lo siguen son inliers.

En el caso de ajuste por mínimos cuadrados, se tienen en cuenta todas las muestras observadas a la hora de calcular el modelo. Esto supone que las muestras con mucho ruido o aquellas que no pertenezcan al modelo que establecemos, se incluyen, alterando los parámetros del mismo y haciendo que la estimación no sea correcta si algunos puntos se alejan demasiado del modelo.

La transformada de Hough, hace que cada una de las muestras vote en un espacio de parámetros del modelo. En este caso, se aislan bastante bien los puntos que no pertenecen al modelo, pero tenemos varios inconvenientes asociados a esta transformada. Uno de los más importantes es que es bastante pesado de ejecutar en un tiempo razonable cuando el número de puntos a evaluar es grande. Otro de los inconvenientes es la correcta selección del espacio de parámetros, puesto que, dependiendo de cuál se elija, tendremos problemas o no. Además, también tenemos el problema de la granularidad del que elijamos en el contador de Hough, lo cual hará que se estimen bien los parámetros o no.

RANSAC selecciona un conjunto aleatorio de entre los datos como núcleo del modelo y compara con el resto a ver si la mayoría de datos se ajusta el modelo, adjuntándolos a una lista de inliers. También comprueba el error asociado de esos datos que parecen ajustarse al modelo y repite este proceso varias veces hasta que no se mejore el error o se llegue a un número de iteraciones máximo.

En visión artificial, este algoritmo se suele utilizar para establecer las mejores correspondencias entre 2 imágenes de entrada en un sistema estéreo, para la posterior obtención de las matrices de transformación de una cámara a otra. Para ello selecciona de entre todos los puntos característicos los ”n” mejores para poder realizar la estimación de lo que corresponda, en nuestro caso la estimación de la matriz de correspondencia entre vistas.

Este método sigue los pasos indicados en el siguiente algoritmo:

Algoritmo 1 RANSAC

```
Require: datos modelo n k t d
  iteraciones := 0
  mejor_modelo := nil
  mejor_subconjunto := nil
  mejor_error := ∞
  while iteraciones < k do
    posibles_inliers := n - se toman datos aleatoriamente
    posible_modelo := param. del modelo ajustado a posibles_inliers
    subconjunto := posibles_inliers
    for puntos fuera de posibles_inliners do
      if puntos que se ajustan a posible_modelo con error < t then
        subconjunto ⇐ puntos
      end if
      if cantidad de puntos incluidos en subconjunto >d then
        mejor_modelo := posible_modelo
        error_actual := error(subconjunto)
      end if
      if error_actual < mejor_error then
        mejor_modelo := mejor_modelo
        mejor_subconjunto := subconjunto
        mejor_error := error_actual
      end if
    end for
    iteraciones++
  end while
  return mejor_modelo mejor_subconjunto mejor_error
```

En el caso de OpenCV, el número de puntos que se toman como núcleo en el modelo, y por tanto, los que devuelve, está fijado a 7. El comportamiento del algoritmo es prácticamente el mismo, con pequeñas diferencias debidas a la implementación específica para el problema de las correspondencias.

4.2. Modelo de proyección de cámara

Para la implementación de esta Tesis de Máster, como para el software base, se han de tener los conocimientos de cómo, sabiendo las coordenadas en el mundo de un objeto en 3 dimensiones, su matriz de rotación y translación respecto de la cámara y sabiendo su matriz de calibración, o en su caso la distancia focal; obtener las coordenadas en la cámara de la proyección de ese objeto.

La proyección que se realiza en la cámara es fundamental a la hora de abordar otros algoritmos como los de estimación de pose, que se verán en la sección 4.3.1. Una ilustración de cómo se realiza esta proyección se muestra en la figura 4.1.

A continuación se pasa a describir matemáticamente el proceso de transformación de un objeto en 3D a su proyección en una cámara.

El primer paso que se ha de realizar es el cambio de coordenadas desde el mundo hasta el sistema de coordenadas cuya referencia está en la cámara. Para ello se han de tener o estimar las matrices de rotación y translación del sistema de coordenadas global (Rw) al sistema de coordenadas de la cámara (Rc).

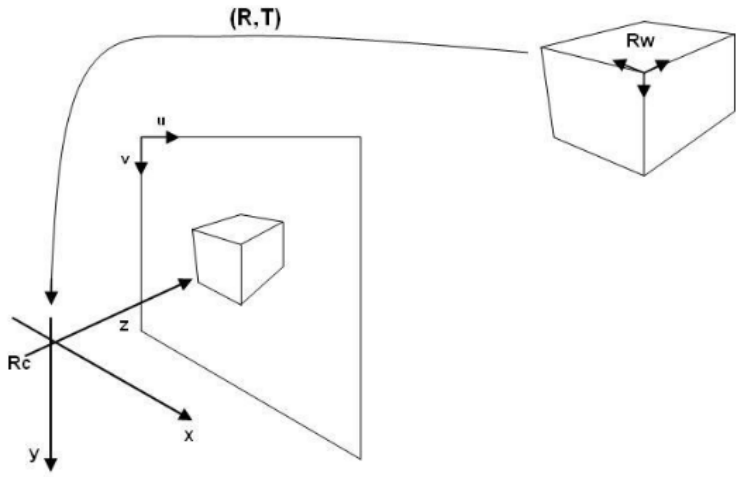


Figura 4.1: Proyección de un objeto 3D en la cámara, con cambios entre sistemas de coordenadas

$$P_c = \begin{pmatrix} R & T \\ 0 & 1 \end{pmatrix} P_w \tag{4.1}$$

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} = \begin{pmatrix} R & T \\ 0 & 1 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \tag{4.2}$$

A continuación se utiliza la distancia focal o la matriz de intrínsecos, que hace de constante de proporcionalidad para ajustar el tamaño del objeto proyectado en la imagen final. Existen varias formas de expresar esta transformación, utilizando la matriz de intrínsecos o la distancia focal. Si asumimos que la matriz de intrínsecos no tiene más términos que la distancia focal, y que esta no varía en x o y, sino que es constante, tenemos lo siguiente:

$$\begin{pmatrix} sx \\ sy \\ s \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} \tag{4.3}$$

Por último, sustituyendo s en el resto de ecuaciones, nos queda lo siguiente:

$$\begin{cases} x = fX_c/Z_c \\ y = fY_c/Z_c \end{cases} \tag{4.4}$$

Se ha de tener en cuenta que el centro de proyección de la cámara no se encuentra siempre en el punto $(0, 0)$, sino que éste está en un lugar de coordenadas arbitrarias (u_0, v_0). Además, para tener exactamente el píxel donde ha quedado representado un punto, se ha de dividir entre el tamaño del sensor CCD, esto es, se ha de tener en cuenta la anchura y altura del sensor (d_x, d_y)Por tanto, la ecuación 4.4 queda de la siguiente forma:

$$\begin{cases} u = x/d_x + u_0 \\ v = y/d_y + v_0 \end{cases} \tag{4.5}$$

4.3. Estimación de pose

4.3.1. POSIT (Pose from Orthography and Scaling with ITerations)

Este algoritmo, desarrollado por DeMenthon y otros en [13], tiene como objetivo la estimación de la pose de un objeto en 3D a partir de sus proyecciones 2D, siendo definida la pose como la orientación y la translación en los 3 ejes del espacio, respecto de un centro, que puede ser prefijado o no.

Este algoritmo parte de una suposición de que las matrices iniciales son aleatorias, para luego ir corrigiendo el error iterativamente.

Algoritmo 2 SoftPOSIT

Require:

puntos del objeto

$P_k = (X_k, Y_k, Z_k, 1)^T = \left(\tilde{P_k}, 1\right),\ 1 \leq k \leq M$

puntos de las imágenes

$p_j = (x_j, y_j),\ 1 \leq j \leq N$

$m \Leftarrow \gamma = 1/(maxM, N+1)$

$\beta \Leftarrow \beta_0,\ \beta_0 \approx 0{,}0004$ si no hay conocimiento a priori

$Q_1\ Q_2$ vectores de Pose iniciales, normalmente aleatorios

$\omega_k = 1,\ 1 \leq k \leq M$

while $\beta > \beta_{final} (\beta_{final} \approx 0{,}5)$ **do**

calcular las distancias $d^2_{jk} = (Q_1 \cdot P_k - \omega_k x_j)^2 + (Q_2 \cdot P_k - \omega_k y_j)^2,\ 1 \leq j \leq N,\ 1 \leq k \leq M$

calcular $m^0_{jk} = \gamma exp(-\beta(d^2_{jk} - \alpha)),\ 1 \leq j \leq N,\ 1 \leq k \leq M$

while $\|m^i - m^{i-1}\| < \delta$ **do**

Normalizar las filas importantes de m: $m^{i+1}_{jk} = \frac{m^i_{jk}}{\sum_{k=1}^{M+1} m^i_{jk}}, 1 \leq j \leq N,\ 1 \leq k \leq M+1$

Normalizar las columnas importantes de m: $m^{i+1}_{jk} = \frac{m^i_{jk}}{\sum_{j=1}^{N+1} m^{i+1}_{jk}}, 1 \leq j \leq N+1,\ 1 \leq k \leq M$

end while

calcular la matriz 4x4 $L = (\sum_{k=1}^{M} m'_k P_k P_k^T)$ con $m'_k = \sum_{j=1}^{N} m_{jk}$

calcular L^{-1}

calcular $Q_1 = \left(Q^1_1, Q^2_1, Q^3_1, Q^4_1\right)^T = L^{-1}\left(\sum_{j=1}^{N}\sum_{k=1}^{M} m_j k \omega_k x_j P_k\right)$

calcular $Q_2 = \left(Q^1_2, Q^2_2, Q^3_2, Q^4_2\right)^T = L^{-1}\left(\sum_{j=1}^{N}\sum_{k=1}^{M} m_j k \omega_k y_j P_k\right)$

calcular $s = (\| \left(Q^1_1, Q^2_1, Q^3_1\right) \| \| \left(Q^1_2, Q^2_2, Q^3_2, Q^4_2\right) \|)^{1/2}$

calcular $R_1 = \left(Q^1_1, Q^2_1, Q^3_1\right)^T /s,\ R_2 = \left(Q^1_2, Q^2_2, Q^3_2, Q^4_2\right)^T /s,\ R_3 = R_1 \times R_2$

calcular $T_x = Q^4_1/2,\ T_y = Q^4_2/2,\ T_z = f/s$

calcular $\omega_k = R_3 \cdot \tilde{P_k}/T_z + 1,\ 1 \leq k \leq M$

$\beta = \beta_{update}\beta\ (\beta_{update} \approx 1{,}05)$

end while

return $R = [R_1, R_2, R_3]^T\ \ T = (T_x, T_y, T_z)\ \ m = \{m_{jk}\}$

En la figura 4.2 vemos la interpretación geométrica del algoritmo. En ella tenemos un punto P, el cual se proyecta en distintos planos.

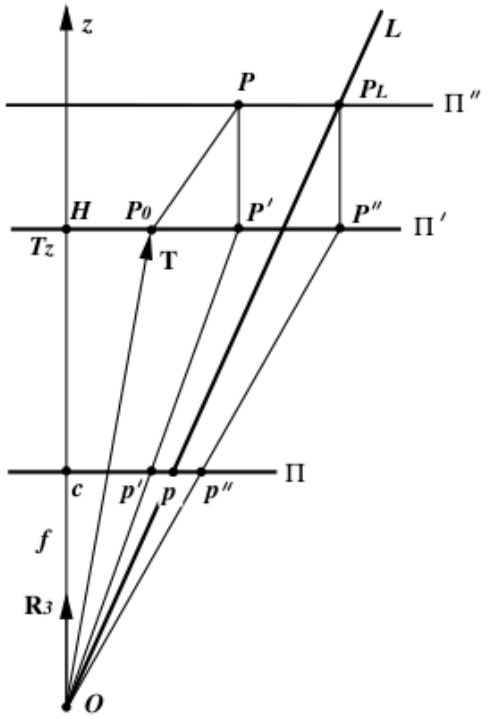


Figura 4.2: Interpretación geométrica de POSIT

En la figura 4.2 se pueden ver 3 planos, entre los cuales hay distintas formas de proyección: débil ($\Pi'' \rightarrow \Pi'$) y pin-hole($\Pi' \rightarrow \Pi$). La primera se realiza para tener todos los puntos en el mismo plano de referencia. La segunda proyección es necesaria para, en función de la distancia focal, saber la traslación inicial en el eje z, del plano donde se realiza la proyección débil Π'.

De este algoritmo tenemos implementaciones tanto en Matlab como en OpenCV. Para empezar a realizar los experimentos de modificación de esta función, se implementarán primeramente sobre Matlab, para luego incluir las modificaciones en OpenCV, que es donde se ejecutarán finalmente las modificaciones.

4.4. Detección de puntos característicos

La detección de puntos característicos, esquinas en nuestro caso, es fundamental a la hora de encontrar buenos puntos para realizar un seguimiento en la imagen. Para localizar las esquinas, se parte de un análisis de derivada en varias direcciones del espacio, puesto que una esquina es una variación en no solo una dirección del espacio, como lo serían los bordes, sino en varias direcciones simultáneamente. Por tanto, la mayoría de detectores de esquinas, van a utilizar por debajo un análisis de bordes, el cual no se tratará en esta memoria, puesto que se saldría de los objetivos de la misma. Asumiendo que se sabe cómo realizar un tratamiento de bordes, se procederá a explicar algunos de los métodos, entrando en más detalle en el que se ha utilizado en este trabajo.

Los métodos más comunes para encontrar esquinas, son Kitchen-Rosefeld y Harris. El método que se utiliza en esta Tesis de Máster para realizar la detección de esquinas es Harris.

4.4.1. Harris

Harris es un método de detección de esquinas, que se propuso como una mejora de otro método previo, el detector de Moravec. En el paper donde se describe este extractor de esquinas [14] se describen varios puntos flacos que tenía este detector y se pasa a mejorarlos.

1. En primer lugar se dice que este detector es anisótropo porque solo tiene respuestas en valores discretos de ángulos de rotación, para cada 45º. Para superar esta limitación, se propone una modificación a la hora de calcular la función que caracterizará el gradiente en todas las direcciones del espacio, de tal modo que se realice casi de manera continua.

   La función de energía que se calcula se puede simplificar como sigue:

$$E(x,y) = \sum_{u,v} \omega(u,v) \left[I(x+u, y+v) - I(u,v) \right]^2 = \sum_{u,v} \omega(u,v) \left[xX + yY + O\left(x^2 + y^2\right) \right]^2 \tag{4.6}$$

   donde los gradientes se pueden aproximar por:

$$\begin{cases} X = I \otimes (-1, 0, 1) = \frac{\partial I}{\partial x} \\ Y = I \otimes (-1, 0, 1)^T = \frac{\partial I}{\partial y} \end{cases} \tag{4.7}$$

   Donde esos vectores son en realidad las máscaras horizontales o verticales de Prewitt. Una vez calculado este gradiente, para desplazamientos de ángulo pequeños, tenemos lo siguiente:

$$E(x,y) = Ax^2 + 2Cxy + By^2 \tag{4.8}$$

   donde:

$$\begin{cases} A = X^2 \otimes \omega \\ B = Y^2 \otimes \omega \\ C = (XY) \otimes \omega \end{cases} \tag{4.9}$$

2. En segundo lugar se encuentra que la respuesta del operador es algo ruidosa debido a que la ventana de detección de bordes es rectangular y de un determinado número de píxeles. Para subsanar este error, se realiza un suavizado con un kernel gausiano circular.

$$\omega(u,v) = e^{-\frac{\left(u^2+v^2\right)}{2\sigma^2}} \tag{4.10}$$

3. A continuación se modifica la forma del cálculo del mínimo de E, en lugar de una minimización normal, haciendo una búsqueda de autovalores, teniendo en cuenta así, la variación según las direcciones, que es lo que ocurre realmente con las esquinas. Estos autovalores tienen relación con los términos cuadráticos de las series de Taylor de las funciones de autocorrelación, que es lo que se utilizaba anteriormente.

   Expresada la ecuación de E en forma matricial, tenemos lo siguiente:

$$E(x,y) = (x,y)\, M \begin{pmatrix} x \\ y \end{pmatrix} \tag{4.11}$$

donde M es lo siguiente:

$$M = \begin{pmatrix} A & C \\ C & B \end{pmatrix} \tag{4.12}$$

Una vez tenemos los autovalores de esta matriz indicada anteriormente, es mucho más fácil discutir sobre ellos, si lo que se tiene es una zona en la imagen en la que apenas hay cambios, una zona bordes rectos, una esquina, o incluso un pixel aislado.

En la figura 4.3 se muestra gráficamente la información que nos da en función de lo grandes que sean los autovalores que se obtengan de la matriz M. Se puede apreciar que cuando los dos autovalores son pequeños, significa que la variación de niveles de gris que sufre la imagen no es muy significativa, por tanto, estamos en la esquina inferior izquierda de la gráfica. Si alguno de los 2 es muy grande y el otro no lo es tanto, estamos en una zona en la que el autovalor y por tanto la pendiente es muy grande y el otro autovalor no lo es tanto, por lo que se deduce que es un borde recto, sin cambios en la dirección en esa zona. Por último, si los 2 autovalores son muy altos, es que hay un gran cambio en varias direcciones en la zona y por tanto existe una esquina. Si los valores de los autovalores son demasiado altos, se considera que es un punto aislado en la imagen, aunque el algoritmo lo detecte como una esquina.

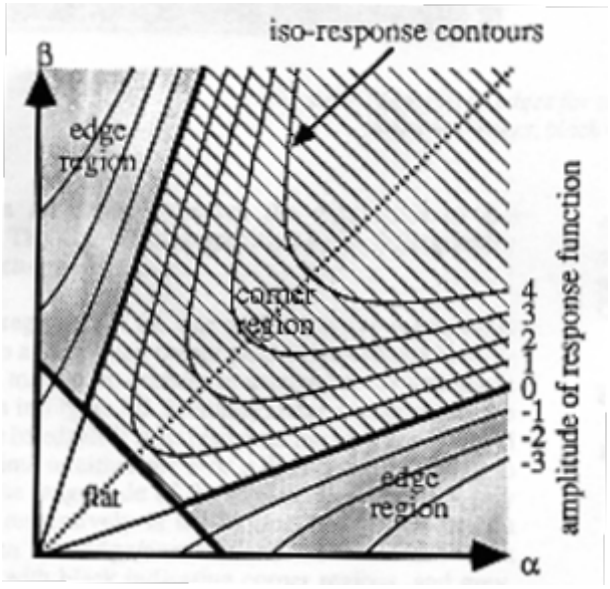


Figura 4.3: Correlación entre autovalores y su significado para el método de detección de esquinas de Harris

4. Por último se ajusta la sensibilidad del algoritmo a las esquinas mediante una variable k. Esto surge a partir de encontrar las soluciones de los autovalores de la matriz M.

$$\begin{cases} Tr\,(M) = \alpha + \beta = A + B \\ Det\,(M) = \alpha\beta = AB - C^2 \end{cases} \tag{4.13}$$

Para poder regular la calidad de las soluciones y por tanto la respuesta del filtro se utiliza la siguiente expresión:

$$R = Det\,(M) - k * Tr\,(M) \tag{4.14}$$

donde k es la variable que regula la respuesta del filtro, y lo abruptas que se desea que sean las esquinas detectadas o el número de las mismas.

4.5. Transformaciones espaciales

Las transformaciones espaciales, en el caso de la aplicación que se propone tienen un papel muy importante, puesto que en las tomas realizadas por el escáner 3D, la referencia del mismo es fija, siendo él mismo la referencia de coordenadas, pero el objeto en este caso es el que se mueve, por tanto, tanto la transacción como la rotación de cada vista respecto de la toma frontal es diferente para cada caso. Por tanto, se verá cómo son estas transformaciones espaciales y métodos para obtenerlas.

4.5.1. Translación

La translación, en general es llevar un punto en el espacio a otro sin alterar su orientación. La implementación de esta transformación es tan sencilla como la suma de matrices. En general:

$$A' = A + T \tag{4.15}$$

En particular, para 2 y 3 dimensiones:

$$\begin{cases} \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} t_x \\ t_y \end{pmatrix} + \begin{pmatrix} x \\ y \end{pmatrix} & 2D \\ \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} + \begin{pmatrix} x \\ y \\ z \end{pmatrix} & 3D \end{cases} \tag{4.16}$$

y en coordenadas homogéneas:

$$\begin{cases} \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} t_x \\ t_y \\ 1 \end{pmatrix} + \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} & 2D \\ \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} t_x \\ t_y \\ t_z \\ 1 \end{pmatrix} + \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} & 3D \end{cases} \tag{4.17}$$

4.5.2. Escalado

El escalado es una transformación que multiplica los elementos de entrada por un número para hacerlos mayores o menores de lo que son inicialmente. El escalado no tiene por qué ser igual en todas las direcciones del espacio y eso se conseguirá como veremos a continuación.

$$\begin{cases} \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s_1 & 0 \\ 0 & s_2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} & 2D \\ \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_3 \end{pmatrix} + \begin{pmatrix} x \\ y \\ z \end{pmatrix} & 3D \end{cases} \tag{4.18}$$

y en coordenadas homogéneas:

$$\begin{cases} \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} & 2D \\ \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} s_1 & 0 & 0 & 0 \\ 0 & s_2 & 0 & 0 \\ 0 & 0 & s_3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} & 3D \end{cases} \tag{4.19}$$

Para los propósitos de trabajo en 3D no es necesaria esta transformación puesto que se dan las coordenadas en mm's de los puntos 3D y la escala de scan-láser es la misma siempre. En 2D siempre puede interesar hacer un escalado de la imagen para poder realizar un zoom o un escalado.

En coordenadas homogéneas es posible juntar las operaciones anteriores para realizarlas a la vez, de la siguiente manera:

$$\begin{cases} \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s_1 & 0 & t_y \\ 0 & s_2 & t_x \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} & 2D \\ \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} s_1 & 0 & 0 & t_x \\ 0 & s_2 & 0 & t_y \\ 0 & 0 & s_3 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} & 3D \end{cases} \tag{4.20}$$

Si nos damos cuenta, la última columna de la matriz, está multiplicada por 1, que es la última coordenada de los vectores en coordenadas homogéneas, por tanto aplica la translación directamente utilizando una matriz cuadrada de transformación, fácil de construir y de aplicar para realizar sumas y multiplicaciones.

4.5.3. Rotación

La rotación consiste en una transformación por la cual no se modifica las proporciones ni los ángulos se modifican, pero sí la orientación del objeto en el espacio.

La rotación de un objeto en el espacio, puede realizarse según 3 ejes, como se puede ver en la figura 4.4, y según el eje sobre el que se rote, la matriz a aplicar varía.

Existen varios tipos de representación de las rotaciones: una de ellas es indicando solo los ángulos, y otra es la matriz de rotación.

Si nos fijamos en el siguiente eje de coordenadas, asociamos cada eje del sistema coordenado con un ángulo de rotación.

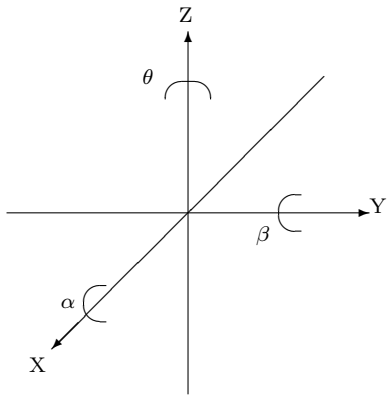


Figura 4.4: Función de ajuste de niveles de gris de las plantillas

En los distintos ejes:

$$R_\theta = \begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} R_\alpha = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} R_\beta = \begin{pmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.21)$$

Es importante el orden en que se aplican las rotaciones, puesto que así como el producto de matrices no es conmutativo, la orientación propiamente dicha en el espacio tampoco lo es.

Además, existen otras formas de representar las rotaciones, pero no se consideran en esta Tesis de Máster, al no ser necesarias en la misma. La forma más sencilla de realizarlas es el uso de cuaterniones, los cuales son una extensión de los números complejos, no solo en una dimensión, sino teniendo una base de 4 vectores, esto es, números hipercomplejos. El uso de cuaterniones supone la eliminación de las relaciones trigonométricas en las matrices de rotación. También se puede utilizar para representar las rotaciones de manera más simple el método de "Rodrigues", con el cual pasamos de una matriz de 3x3 a un vector de 3x1 que nos da una idea de los 3 ángulos que representan la rotación en 3D.

4.5.4. Métodos para la obtención de matrices de transformación a partir de varios puntos

Existen múltiples métodos para el cálculo de la matriz a partir de varios puntos. Varios de ellos se describen por encima a continuación, aunque lo que suele pasar con todos los métodos es que el error de salida es menor cuanto mayor es el número de puntos tomados en consideración.

Para resolver la matriz de transformación, habría que resolver una ecuación del tipo:

$$\begin{pmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \\ z_1 & \cdots & z_n \\ 1 & \cdots & 1 \end{pmatrix} = RT \begin{pmatrix} x'_1 & \cdots & x'_n \\ y'_1 & \cdots & y'_n \\ z'_1 & \cdots & z'_n \\ 1 & \cdots & 1 \end{pmatrix} \quad (4.22)$$

donde $n \geq 4$. El método para la resolución de la matriz RT se puede hacer mediante álgebra, tomándolo como un sistema de ecuaciones lineales. Siendo estrictos no sería exactamente un sistema de ecuaciones lineales, puesto que las rotaciones y translaciones introducen términos no lineales dentro de la matriz RT, pero dejando fijos éstos, se puede considerar el sistema lineal, por tanto se utilizan métodos sencillos y rápidos para poder resolver el sistema.

Si se considera este problema como no lineal, para resolver este problema de una manera más precisa, aunque utilizando más tiempo de cómputo, se puede utilizar un método de resolución de ecuaciones no lineales llamado Levenberg-Marquardt, que emplea mínimos cuadrados no lineales. Este método se utiliza normalmente si se necesita una precisión considerable en las matrices de transformación y también en los ángulos de rotación.

Como se ha dicho antes, existen varios métodos para resolver el sistema, pero el utilizado en este caso por el programa Meshlab es la obtención de autovalores y autovectores para calcular el vector de rotación más probable, y después, la translación se encuentra en la diferencia entre la matriz resultante de la rotación y la matriz original.

Teniendo estas relaciones entre puntos, por si alguna de las matrices calculadas no es correcta, se itera este proceso tomando de los 4 o más puntos obtenidos grupos de 4 en 4, se va resolviendo y se va iterando el proceso hasta que se encuentra con una matriz lo suficientemente buena o se ha superado un número límite de iteraciones.

Capítulo 5

Arquitectura del sistema de partida

5.1. Introducción

El sistema está basado en lo expuesto en el método desarrollado en el grupo por P. Jimenez et al, expuesto en el artículo [17], al cual se le ha aplicado una serie de mejoras en algunos de los puntos indicados en la publicación. A partir de este sistema base se amplían algunos puntos que son de interés para esta Tesis de Máster. El diagrama general de bloques del sistema se presenta en la figura 5.1.

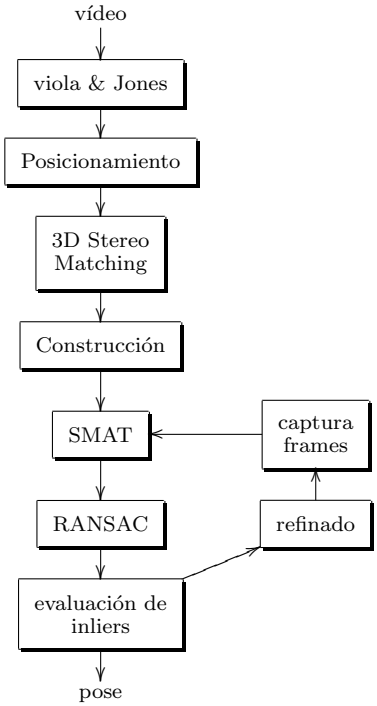


Figura 5.1: Esquema principal del sistema

En este capítulo se realizará una descripción general del sistema antes presentado así como una explicación más profunda de algunos puntos clave para esta Tesis de Máster

5.2. Descripción general

Partiendo del diagrama de la figura 5.1 se destaca lo siguiente.

Uno de los puntos clave, para que el sistema funcione correctamente es la inclusión de la información de la calibración, para obtener una estimación de las coordenadas 3D de los puntos a través de la aplicación de las ecuaciones que gobiernan los sistemas estéreo.

A continuación se explicarán los puntos clave del sistema de tracking, referentes al establecimiento del modelo facial, sin el cual, no se podría hacer tracking correctamente, puesto que para realizar un seguimiento en el espacio, necesitamos tener una idea de cómo es el objeto que queremos seguir, y por tanto, debemos generar el modelo previamente.

Posteriormente se aplica un algoritmo de detección de caras, lo cual sirve para establecer la región de búsqueda reducida dentro de la imagen y reducir tiempo de cómputo y además para restringir posibles posiciones no válidas devueltas por algoritmos de cálculo de pose.

Una vez hemos acotado la región de búsqueda dentro de la imagen, lo que se hace es la selección de los puntos correspondientes a la maya, y la asociación de datos 2D - 3D, para poder realizar el seguimiento en las imágenes. Este sistema se supone que ha de incluir y borrar puntos, puesto que al tener giros, algunos de los puntos característicos de la cara desaparecen (no son visibles) y otros, por el contrario aparecen, por lo que se supone que el algoritmo debe soportar este tipo de situaciones.

Por último, una vez encontrados estos puntos, se realiza un seguimiento de un entorno a ellos y se buscan los candidatos más cercanos posibles al anterior frame en una distancia lo más reducida posible.

Con la distribución de puntos localizada actualmente, se aplica un algoritmo de cálculo de pose, para la obtención de la orientación del espacio del objeto en el espacio, en este caso la cara, a seguir.

5.3. Calibración

El sistema de adquisición de imágenes es parecido al sistema que visual de los animales, esto es, un sistema estéreo, en el cual se tienen 2 ojos. Este hecho, permite una estimación de la profundidad a la que se encuentran los objetos, previo entrenamiento.

En el caso de la visión por ordenador, se utilizan 2 cámaras que actúan como ojos. Sabiendo la distancia entre cámaras, combinando las imágenes y obteniendo el mapa de disparidad entre cámaras, se puede tener una estimación de la profundidad a la que tenemos los objetos aplicando las ecuaciones de la geometría epipolar que son las que nos dan las correspondencia entre puntos.

Este método tiene un error que es proporcional al cuadrado de la distancia estimada, ponderado por una constante. El error en la estimación de profundidad también depende mucho de la distancia entre cámaras, lo cual puede hacer que se tengan buenas estimaciones a distancias cortas y muy malas a distancias largas, o viceversa.

A la hora de realizar la calibración también hay que tener en cuenta el efecto de la distorsión que introducen las lentes. Algunos de estos efectos de distorsión se pueden corregir al ser términos cuadráticos de la serie de Taylor que determina las aproximaciones realizadas entre los modelos pin-hole y el modelo de lente plana. Estos efectos de distorsión aparecen frecuentemente y son de dos tipos: de barril y de cojín. Existen otros efectos no fácilmente corregibles, no lineales debidos a otras deformaciones de la lente que se pueden producir a la hora de la fabricación.

La implementación en este software de un proceso de calibración automático está en fase experimental, por lo que, en principio, se utilizará la toolbox de Matlab para esta parte del proceso con las adaptaciones pertinentes para el software de tracking.

Para la realización de la calibración se utilizan secuencias en las que se graba un damero, a partir del cual se detectan las esquinas. Por esta razón se utilizan este tipo de patrones, puesto que encontrar esquinas en un damero es un proceso muy sencillo para el que se utilizan detectores de esquinas como Harris. Haciendo que el patrón sea rectangular, tenemos información además de orientación, lo cual es útil para seleccionar esquinas correspondientes entre varios frames. Si el tablero fuese cuadrado, no habría forma de diferenciar qué esquina corresponde a cual si el tablero se rota. Se utilizan una secuencia variada de frames, con el tablero en distintas posiciones, e incluso más cerca del centro o en los extremos de las cámaras, para poder corregir las deformaciones de barril o cojín que se dan en la mayoría de las lentes reales.

Con los coeficientes de distorsión y las matrices de rotación y translación entre cámaras, se puede realizar fácilmente la transformación de puntos de una cámara a otra, con lo que tenemos ya información 3D del entorno común que ven las 2 cámaras.

Aplicando el proceso de calibración a las imágenes de damero utilizadas, se guardan las siguientes matrices:

- La matriz R de rotación entre centros de cámaras
- La matriz T, de translación entre centros de cámaras
- Las matrices que contienen los coeficientes de los términos de segundo grado que modelan la distorsión de segundo orden, esto es, las correcciones aplicadas para paliar la distorsión de barril o de cojín. (Fr y Fl)
- La matriz Fundamental, que transforma directamente los puntos de una cámara en la otra en coordenadas pixélicas, no métricas, para lo cual está la matriz Esencial, pero ésta no es necesario utilizarla en principio y además es deducible de los datos anteriores.

Al ser el coche un espacio reducido en el que no se puede introducir un damero de tamaño adecuado para la calibración, se ha ideado un casco cuya parte de atrás contiene un damero. Con el giro de la cabeza se realizan las distintas poses necesarias para una correcta calibración, salvo la parte en la que se acerca a la cámara para corregir distorsiones de la lente, por tanto las calibraciones obtenidas son aceptables, pero si se encuentra en los extremos de la cámara esta distorsión no se verá correctamente corregida. Varios ejemplos de este damero se encuentran en la figura 5.2.

Por último, para el cálculo de la homografía entre una cámara y otra, en lugar de utilizar algoritmos como POSIT, que funcionan bien con puntos cuya distribución puede ser aleatoria, se utiliza la función de OpenCV cvFindHomography, que funciona mejor cuando se sabe que los puntos que se va a detectar son coplanares, como es este caso. Esto implica que se reduce el error al haber bastantes puntos seleccionados, y si encima, los que sean un tanto erroneos se eliminan con RANSAC, el error de las matrices de calibración disminuye todavía más. Esto supone que en la reconstrucción 3D, a la hora de establecer los puntos a través de la geometría epipolar, el sistema es más preciso, lo cual redundará en un error de seguimiento menor.

El sistema de calibración utilizado en estas secuencias, basado en el casco con el damero, se utiliza también para la obtención del ground truth con el que comparar los algoritmos que se están estudiando.

Figura 5.2: Ejemplos del casco con damero utilizado para la calibración

5.4. Establecimiento del modelo facial

El establecimiento del modelo facial se realiza manualmente introduciendo los puntos. Este proceso se lleva a cabo indicando en una vista la localización de un punto de interés, y en la otra vista marcar su correspondiente. Por tanto, ya tenemos caracterizado, a partir de las matrices de calibración de la cámara y las ecuaciones epipolares para establecimiento de correspondencias en secuencias estéreo, la posición 3D del punto.

Una vez repetido este proceso con varios puntos de interés, con el suficiente contraste en una vista, frontal normalmente, se comprueba la validez del modelo hasta que se realicen rotaciones laterales. En ese caso, se realizará una ampliación del modelo para tener la posibilidad del seguimiento de puntos, tanto lateralmente como frontalmente. Repetiremos este mismo proceso de ampliación del modelo con el otro lateral para tener el modelo completo.

Para el establecimiento del modelo completo se necesita la configuración de ciertos parámetros que dependen de la pose y que dan flexibilidad al software.

Uno de estos parámetros son umbrales que se aplican para dar más flexibilidad a la hora de moverse a unos puntos que a otros. Hay puntos característicos en la cara que tienen más posibilidad de moverse frente a otros, esto es, no son estáticos en el modelo 3D. La boca tiene bastante flexibilidad a la hora de moverse de posición respecto a la máscara original, por tanto, el margen que se asignará a éstos será mayor que el margen asignado a los puntos situados en las esquinas de los ojos.

Otro de los parámetros que se ha de configurar son los ángulos de ocultación. Estos ángulos se corresponden con los ángulos de rotación del modelo completo cuando ciertos puntos del mismo

desaparecen por el hecho de ser la cara un cuerpo sólido y opaco, con lo que quedan ocultos. Para asignarlos se seleccionan y se va rotando el modelo hasta que llega el ángulo desde el cual se empieza a ver un punto, y el que se termina de ver. Estas rotaciones están referidas a lo largo del eje de rotación que supone la cabeza y el cuello.

En la configuración del modelo completo, se guardan, además de los datos anteriores, los puntos correspondientes en cada una de las cámaras para un fotograma concreto (normalmente el inicial), las coordenadas de los puntos 3D y las matrices de calibración del sistema. Estos 2 últimos datos, no cambian a lo largo del proceso, salvo que se realice una ampliación, puesto que los puntos 3D no cambian aunque la apariencia de la cara sí lo haga, y las matrices de calibración no cambian a lo largo del procesado, puesto que la posición de las cámaras es fija gracias al montaje realizado en la cabina. Cuando el modelo 3D es ampliado, el número de puntos en los frames iniciales pueden no ser los mismos, por lo que habría que reajustar el modelo de nuevo.

En la aplicación software los atajos de teclado utilizados para añadir puntos son, en el frame de edición del modelo, pulsando 'a' una vez, se añade un punto. Este punto se edita teniendo la tecla ctrl pulsada mientras movemos el ratón. Después de situarlo en una vista, se traza la línea epipolar correspondiente en la otra vista y se sitúa en el punto correspondiente de esta recta. Una figura explicativa de este proceso se encuentra en la figura 5.3.

Figura 5.3: Proceso de adición de un punto

Para borrar un punto, lo seleccionamos con el ratón, o a través de los atajos de teclado y pulsamos la tecla 'd'. Un ejemplo de esto se encuentra en la figura 5.4, donde el punto a borrar en la primera imagen se encuentra marcado en rojo.

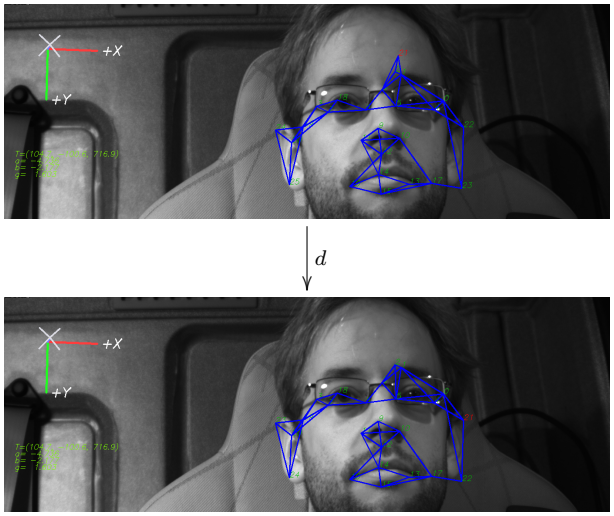
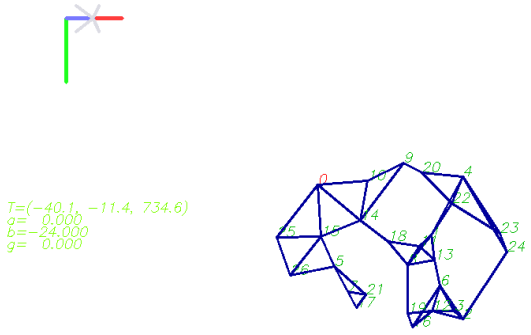


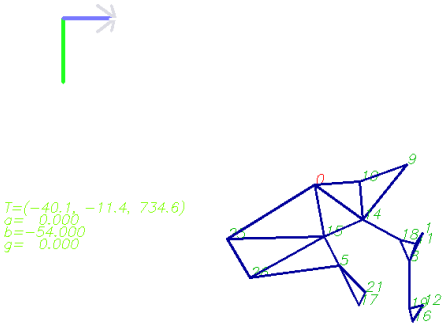
Figura 5.4: Proceso de borrado de un punto

Para modificar los márgenes de deformación de la malla en determinados puntos, se ha de entrar en modo umbral, y para ello se pulsa la tecla 't'. Una vez en este modo y seleccionando cualquier punto, podemos modificar su umbral pulsando las teclas 1, 2 o 3. Esto corresponde a que al modelo, para estos puntos se le permite una deformación baja, media o alta.

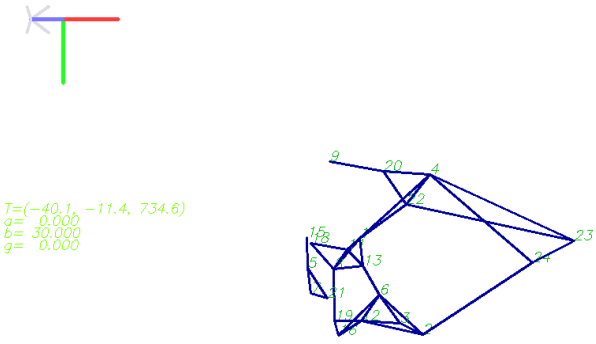
Por último, para modificar los ángulos de visibilidad de los puntos, a partir de los cuales quedan ocluidos o no suficientemente visibles, se utiliza lo siguiente. Lo primero de todo es activar el modo de ocultamiento o no de puntos en la imagen, lo cual ayuda bastante a la hora de establecer los ángolos para los que los mismos son visibles. En la ventana de edición del modelo, se gira éste con las teclas de dirección izquierda y derecha. En el punto que creemos que dichos puntos pueden quedar ocluidos, se pulsan las teclas + y - para los ángulos mínimos y máximos de visibilidad. Esto en principio se ha de realizar con cada punto, aunque a veces con los márgenes que vienen por defecto es suficiente. Un ejemplo de una malla manual a la que se le han configurado los ángulos de ocultamiento se muestra en la figura 5.5.



(a) fontal



(b) izquierda



(c) derecha

Figura 5.5: Ángulos de ocultamiento para una malla manual

5.5. Detección de la cara

Para centrar el modelo y evitar que se salga de ciertos límites, se utiliza Viola & Jones [18] detector de características, basado en Haar Wavelets y Adaboost, entrenado para la detección de caras.

El funcionamiento de este algoritmo se basa en varias características:

1. En primer lugar, se ha de tener una base de datos en la que haya tanto muestras positivas, esto es, que contengan el objeto que se quiera detectar, como muestras negativas, las cuales no contendrán el objeto a detectar.

2. A esta base de datos se le aplica detección de características invariantes, extraídas a partir de la transformada Wavelet.

3. Una vez se tienen caracterizadas las muestras de las bases de datos con sus coeficientes Wavelet, se realiza una comparación a través de un clasificador Adaboost, y se establecen los umbrales para cada una de las características.

4. Se encadenan los detectores Adaboost desde la característica principal detectada que más espacio ocupa en la imagen, hasta las que menos. Si se pasan todos los detectores se da una muestra por válida, si no, la muestra se rechaza en el momento que alguno de los clasificadores de resultado negativo.

5. Esto se guarda en un archivo que contiene los umbrales, el tipo de característica detectada y el orden de la característica obtenida, esto es, la importancia de la misma.

Este detector es invariante a escala, rotación (hasta cierto punto, con unos 30^{o} aproximadamente) y a translación.

A la hora de aplicarlo, cuando arranca el software se leen los archivos de características, se toma la imagen, y se aplican los puntos 2 a 4 para realizar la detección sobre la imagen. Para agilizar el proceso, aprovechando la propiedad de invarianza a escala, se reduce el tamaño de la imagen antes de procesarla.

Se utiliza este detector al ser eficiente y al tener una alta tasa de acierto, sobre todo, teniendo en cuenta las condiciones que tenemos, en las que hay casi siempre una cara presente y que ésta tampoco se mueve rápidamente, con lo que es fácilmente localizable.

Por tanto se pueden aplicar restricciones espaciales a las predicciones de puntos realizadas para la pose y por tanto, recalcular la estimación para que se ajuste lo más posible a la zona de la imagen donde se encuentra la cara.

5.6. Selección y tracking de los puntos

Con el modelo 3D generado, se buscan los puntos más cercanos en la imagen para encontrar esquinas que sean fácilmente rastreables. Una vez tenemos estos puntos asociados a los manualmente establecidos, se guarda este parche inicial y se busca un parche de unas dimensiones de 15, 20 y 30 píxeles alrededor del punto de interés. A lo largo de la secuencia, se va guardando otro parche adicional que se actualiza únicamente si la distancia del punto ha sobrepasado un umbral, y también el último parche que se tiene de un punto concreto y también se busca en las resoluciones anteriormente dichas.

Si con el primer parche es suficiente para realizar el seguimiento, el algoritmo para y da la posición obtenida como buena, por lo que no se realizan el resto de pasadas. Si no se ha encontrado un parche lo suficientemente bueno en las cercanías del punto anterior, se pasa al parche intermedio y se realiza la búsqueda con las 3 resoluciones, y si no, se realiza la búsqueda con el último parche de ese punto. Si no se ha logrado una buena localización del punto, ese punto se descarta en ese frame y se toma como oculto, lo cual se tiene en cuenta a la hora del cálculo de la pose.

El seguimiento se realiza haciendo un matching en las cercanías de los parches, para cada uno de los puntos. En condiciones normales, el movimiento realizado no es muy grande y el matching se realiza de forma rápida y sin problemas. Si se produce alguna rotación, y se empiezan a ver nuevos puntos por un lado y desaparición de puntos por el lado contrario. Esto provoca que se realice una actualización del número de parches y asociaciones a cada uno de los puntos. Esto es debido a la definición de los ángulos de visibilidad asociados a cada punto 3D. Estos ángulos de visibilidad provocarán que se destruyan parches y se creen otros según los puntos que se vean en cada momento.

En la figura 5.6 se muestra un frame en las 2 cámaras, en la que se muestran los parches que se utilizan en el seguimiento una vez inicializada la posición.

Figura 5.6: Parches asociados a cada punto visible del modelo para la realización del seguimiento

5.7. Estimación de la pose

Una vez se tienen los puntos en la imagen, se realiza la asociación entre los puntos 2D y los 3D con la ayuda de las matrices de calibración, que se aplicarán al algoritmo de cálculo de pose, el algoritmo RANSAC, descrito en la sección 4.1.1, para eliminar posibles puntos que no se ajustan al modelo establecido, y por último POSIT, descrito con profundidad en la sección 4.3.1, como algoritmo que calcula iterativamente la pose del modelo respecto de cada una de las cámaras que conforman el sistema estéreo.

La implementación de RANSAC utilizada para filtrar puntos que no se ajustan al modelo es sencilla. En la parte de cálculo de modelo, esto es, el núcleo de RANSAC se utiliza POSIT. Para ver cuales son los puntos que mejor se ajustan al modelo, se toman varios puntos 2D en grupos de 7. Se asume que esos puntos seleccionados son todos pertenecientes al modelo y se calcula la matriz de transformación (R, T) para ellos. Luego, con esa matriz calculada comprobamos, del resto de puntos que queden sin evaluar, cuántos de ellos se aproximan al modelo calculado y con qué margen de error se desvían del modelo calculado. Iterando RANSAC un número suficiente de veces con distintos puntos 2D entre sí (se asume que el modelo 3D es fijo a lo largo de todo el vídeo), se terminará encontrando una buena aproximación de las matrices que necesitamos.

En este caso, para hacer que RANSAC sea depurable, se necesita que las secuencias pseudoaleatorias sean siempre las mismas, por lo que se han adoptado 2 soluciones para esto: si el número de puntos no es muy grande, las secuencia se generan con Matlab y se guardan en un archivo, el cual se carga al principio. En cambio si el número de puntos es muy grande, como en este caso, 60 y 128, para los que se han realizado las pruebas, las secuencias se generan en el programa y se guardan en memoria RAM durante la ejecución.

Este proceso se ha de repetir para todo frame del vídeo, si se ha localizado una cara. Con esto se pueden obtener a lo largo del tiempo, las variaciones de pose y las coordenadas de la dirección de la cara en cada instante de tiempo.

Al ser éste un sistema estéreo, a la hora de estimar la pose global, se realiza una fusión de las 2 poses obtenidas, teniendo en cuenta el cambio de referencia existente entre una vista y otra.

Al utilizar POSIT la imagen centrada se toma como referencia para realizar el cálculo de la pose y sin embargo la referencia del modelo 3D es otra, por lo que se ha de realizar una pequeña transformación en el vector de traslación que ha de aplicarse si se quiere que la proyección 2D sea correcta.

Capítulo 6

Mejoras propuestas

En esta Tesis de Máster se proponen 2 mejoras fundamentales en el sistema de caracterización de la dirección a la que mira la cabeza presentado en el capítulo anterior.

Para aumentar la precisión del sistema y mejorar alguna de las características que posee, se ha planteado lo siguiente: por un lado reducir el tiempo de proceso en la estimación de POSIT, según se explica en la sección 6.1, y por otro lado el aumento de precisión y robustez debido a la inclusión de modelos 3D adquiridos mediante scan-láser 3D.

La primera mejora se ha implementado primero en Matlab y después en C++ para su utilización inmediata en el software global y posterior comprobación. La versión escrita en C++ se ha realizado de tal forma que sea fácilmente integrable con lo que había implementado de OpenCV y que su adaptación fuese rápida.

La segunda mejora incluida, es la que mayor trabajo ha necesitado, y se trata de componer un modelo 3D diezmado que conserve la apariencia de la persona concreta, con un número de puntos significativos, pero sin tener demasiados, puesto que muchos puntos pueden sobrecargar al sistema. Para ello, ante la imposibilidad de realizar un modelo completo con una sola captura del scan-láser 3D, se toman 3 vistas, que luego se componen como se indicará más adelante.

6.1. Modificación de POSIT propuesta

A partir del algoritmo POSIT básico definido en el apartado anterior, pasamos a realizar algunas modificaciones sobre el mismo, para reducir el tiempo de ejecución de manera significativa. En el caso que se estudia, se puede reducir el tiempo de ejecución del algoritmo aprovechando el hecho de que se han realizado otras estimaciones anteriormente. Si nos fijamos en el pseudocódigo del algoritmo base, mostrado en el listado 2, da pié a poder utilizar la predicción anterior de la pose, pero en el algoritmo implementado tanto para C/C++ como para Matlab no se implementa esta mejora, con lo que modificamos el código fuente.

La modificación realizada está en los parámetros iniciales, en los cuales se incluyen la matriz de rotación y el vector de translación calculados en el frame anterior. Esto mejora los resultados en lo que a tiempo de ejecución se refiere, si tenemos en cuenta que la variación de movimiento entre frames consecutivos va a ser pequeña, que es la situación habitual en la aplicación.

La inclusión de los resultados del frame anterior se ha incluido de la siguiente forma:

1. Se comprueba que exista una predicción anterior

2. Se calcula la estimación de puntos 3D en coordenadas homogéneas y las proyecciones de los mismos sobre la imagen.

3. Si no existe predicción anterior, lo calculado anteriormente es válido y se continúa como POSIT normal. Si existe predicción anterior, se calcula el término de escalado a partir de las coordenadas anteriores.

4. Por último, con el término de escalado anterior, se recalculan las proyecciones en la imagen a partir de la predicción anterior

Algoritmo 3 Modificaciones de inicialización de POSIT en Matlab

```
...
% checking if it is the stantard POSIT or modified POSIT
if nargin == 3
  ui = centeredImage(:,1);
  vi = centeredImage(:,2);
end

wi = ones(nbPoints, 1); % homogeneous image coordinates
homogeneousWorldPts = [worldPts, wi];

% previous prediction inclusion
if nargin~=3
    wi = (homogeneousWorldPts*prev_pred(3,:)')./prev_pred(3,4);
    ui = wi.*centeredImage(:,1);
    vi = wi.*centeredImage(:,2);
end

objectMat = pinv(homogeneousWorldPts) % pseudoinverse

converged = 0;
count = 0;

...
```

Primero se realizan las modificaciones sobre la función implementada en Matlab, para tener una referencia a la hora de modificar las funciones de OpenCV. Las pruebas realizadas en Matlab para el algoritmo POSIT y POSIT modificado se describen en el siguiente capítulo. Se observa que la modificación del algoritmo propuesto, depende en gran medida de varios factores, como son el ángulo en el que nos estemos moviendo que afecta a la variación de la posición de los puntos entre frames. Como primera prueba sencilla para ver si el algoritmo tiene la suficiente eficiencia, se ha probado utilizar un cubo. Este cubo se rotará en las 3 direcciones del espacio, además de utilizarse pasos distintos. Como medida de tiempo se ha utilizado el número de iteraciones que tarda el algoritmo en converger, suponiendo que el tiempo de cada iteración es el mismo.

En líneas generales, la modificación que se ha propuesto en la implementación de POSIT de OpenCV es la inclusión de la anterior predicción. En el caso que nos ocupa no solo se va a realizar una única estimación de pose, con lo que la matriz inicial de transformación aleatoria tiene sentido, sino que esta estimación de pose va a ser en cada frame, con lo cual, a partir de una matriz inicial aleatoria puede hacer que perdamos tiempo si entre 2 frames la variación de la pose no ha diferido mucho. Aprovechando que el procedimiento que utiliza POSIT para encontrar esta matriz de transformación es por aproximaciones sucesivas, se utiliza la predicción

anterior, la cual se supone cerca de la actual y por tanto el error inicial disminuye, con lo que se consigue en principio una reducción en el tiempo de proceso. El esquema de la introducción de la mejora en el algoritmo es el mostrado en la figura 6.1.

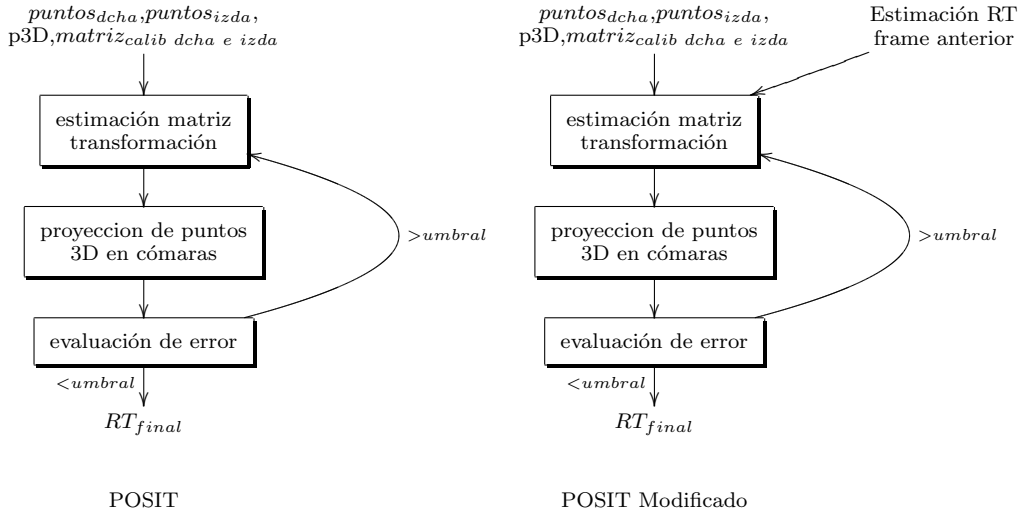


Figura 6.1: Comparación entre el sistema original y el modificado

Para el mejor aprovechamiento del algoritmo en el software anterior, se traduce el código en lenguaje Matlab a código en C y se incluye en el software de seguimiento de caras.

El código fuente de POSIT original, implementado en OpenCV no contempla la posibilidad de incluir la predicción ni de extraer el número de iteraciones empleadas ni el error asociado a la estimación, por lo que ha habido que desarrollarlo.

6.2. Generación del modelo diezmado desde datos 3D

En el sistema comentado en el capítulo anterior, vemos que para el tracking es necesario la introducción de un modelo 3D, que en el caso del programa hasta ese momento, se realizaba de manera manual, introduciendo una serie de puntos contrastados lo suficiente para que tengan posibilidad de seguimiento. El problema es que la malla hay que generarla a mano, con los errores que ello implica, puesto que la calibración entre cámaras puede inducir a una mala estimación de los puntos 3D, y por tanto errores en la estimación de las coordenadas 3D y de los movimientos posteriores del modelo.

Por tanto, si se quiere tener independencia del modelo con la calibración para tener un modelo más preciso y por tanto aumentar la estimación de posición del modelo es necesario obtener estos puntos de manera más exacta, aunque debemos de encontrar un compromiso entre exactitud y tiempo de cómputo, puesto que añadir muchos puntos supone un tiempo de proceso que se dispara, por lo que conviene diezmar de manera inteligente el modelo.

Para obtener el modelo diezmado completo 3D a partir de un scan-láser, conceptualmente se han de seguir una serie de pasos que se comentan a continuación.

Para empezar, se adquieren varias vistas con el scan-láser para tener un modelo completo de la cara. Se eligen con un ángulo suficiente para que haya una parte en común y otra y otra que no se pueda ver desde otra vista. Por ejemplo, si empezamos con la vista frontal de la cara de un sujeto, las orejas y la parte lateral no se observan bien porque el láser no llega a esas zonas a menos que la incidencia del mismo no sea perpendicular. Por tanto, es necesario realizar varias tomas para tener un modelo completo. Experimentalmente se han elegido 3: frontal, lateral derecho y lateral izquierda.

La resolución del scan-láser se puede controlar en cierta medida, puesto que se tienen modos de adquisición fino y rápido, los cuales tienen una resolución de aproximadamente unos 50000 y 5000 puntos por toma respectivamente. Teniendo en cuenta que hay que fusionar vistas y que hay partes en común entre vistas que al fusionar vistas se solapan y se eliminan parte, los modelos compuestos en bruto están aproximadamente en unos 90000 y 10000 puntos respectivamente. Esto implica que, si se introdujesen estas mallas tal y como están, el tiempo de proceso sería impensable e incluso partes del algoritmo podrían fallar ante tal cantidad de puntos a realizar tracking.

Por tanto, el número de puntos ha de ser reducido de manera inteligente, esto es, buscando las zonas dentro de la imagen con mayor contraste, es decir, que supongan puntos en la imagen que sean fácilmente rastreables, o sea, las esquinas. Como detector de esquinas se ha utilizado el detector de Harris, puesto que para la detección de características, es el más adecuado, en comparación con otros métodos implementados en OpenCV, además mediante el mismo se asegura la detección de esquinas que son más adecuadas para seguir. Además, el método empleado en el software para un ajuste final de la malla a los puntos para comenzar el seguimiento de la pose de la cara se basa en la búsqueda de este tipo de características en la imagen inicial.

Con los puntos 2D seleccionados, y sabiendo las correspondencias entre puntos 2D-3D, sabemos cuál es al punto 3D obtenido por el scan-láser. Para obtener esta información, tanto de correspondencia 2D-3D como la propia información 3D, leemos los archivos .vrml proporcionados por el software de adquisición del scan-láser, si se habilita la opción de integrar la imagen como textura.

La estructura de los archivos vrml facilita la lectura de datos para nuestros propósitos, puesto que está bien estructurada y es fácilmente entendible por las etiquetas que contiene. Un esquema de esta estructura se encuentra en la figura 6.2.

Como vemos en la figura 6.2, es fácilmente legible el archivo por un programa en lenguaje c. Por esta razón éste ha sido el formato de conversión de salida para esta parte del algoritmo. Además se aprovecha el hecho de que los puntos están ordenados de arriba a abajo y de izquierda a derecha según se mira la imagen. Por tanto, con una ordenación estándar de menor a mayor de los puntos obtenidos es suficiente para, de una sola pasada, terminar la búsqueda.

Uno de los problemas en la búsqueda de asociaciones 2D - 3D es que no para todos los puntos de la imagen existe muestra 3D. Estas no correspondencias son debidas fundamentalmente a 2 motivos:

1. Lo más frecuente es que fuera del objeto escaneado, el resto de objetos de la escena estén muy alejados, por tanto fuera del rango del scan-láser.

2. La superficie no refleje, con lo que no se tiene medida de distancia en el scan-láser, entonces, también queda el punto sin asignarse. Esto es frecuente en algunos materiales, como las

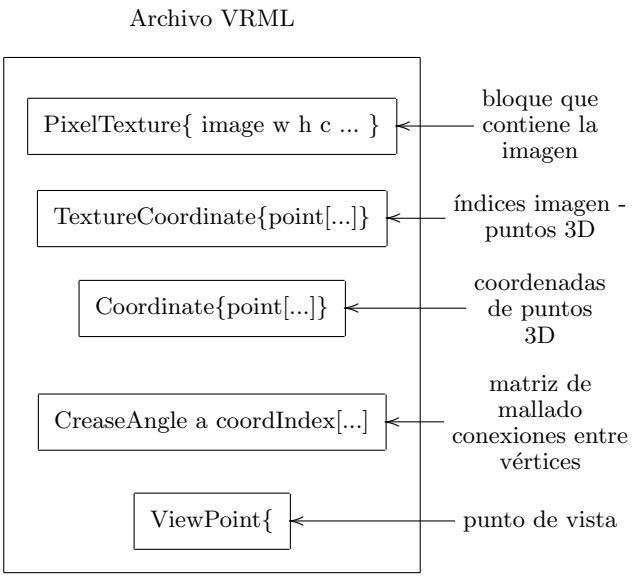


Figura 6.2: Estructura de los archivos VRML generados por el programa de adquisición

pinturas empleadas en las papeleras, el pelo humano, etc. Esto último es importante, puesto que cuando se adquieran datos, no se va a obtener un modelo completo 3D de toda la cabeza, sino de lo correspondiente a la piel.

Por tanto, a la hora de buscar puntos, si el punto concreto queda en una zona que no hay información, pero cerca hay una zona en la que sí lo hay, se toma un entorno del punto en el que puede que sí exista. Se toma un entorno de ± 5 píxeles alrededor del punto seleccionado con el detector de esquinas. Por lo cual, se evitan ciertos problemas de que el detector de esquinas las detecte en las zonas de pelo dentro de la cara, y aseguramos la correspondencia con un punto 3D adquirido de un punto cercano al seleccionado. Con esto relajamos las condiciones y obtenemos puntos entre regiones límite que son interesantes para la composición del modelo.

Una vez tenemos las distintas vistas del modelo, queda sacar las matrices de transformación entre vistas, para unificarlas y generar una única careta con la que trabajar en el software final. Esto se consigue utilizando un software externo, libre y modificado lo suficiente para sacar la información de las matrices que necesitamos.

Con esas matrices de transformación es sencillo generar la máscara compuesta de la cara, que incluye puntos de la cara frontal, nariz, ojos, orejas, etc. Precisamente por tener algo más de precisión en las orejas, en los giros que realiza la cabeza se toman estas vistas laterales y luego se encajan.

Por último, con la malla de puntos compuesta por las 3 vistas encajadas, se realizan unas transformaciones necesarias debido al cambio de referencia del escáner a la referencia del software de tracking, y se adaptan los datos al formato de archivo adecuado para lectura correcta por el software.

Una vez leído el archivo, lo único que queda por hacer es ajustar el modelo 3D de forma que encajen todos los puntos en las 2 vistas. Un ejemplo de esto último lo podemos ver en la figura 6.3. Por último, un esquema que representa el procedimiento general para obtener el modelo 3D diezmado se representa en la figura 6.4.

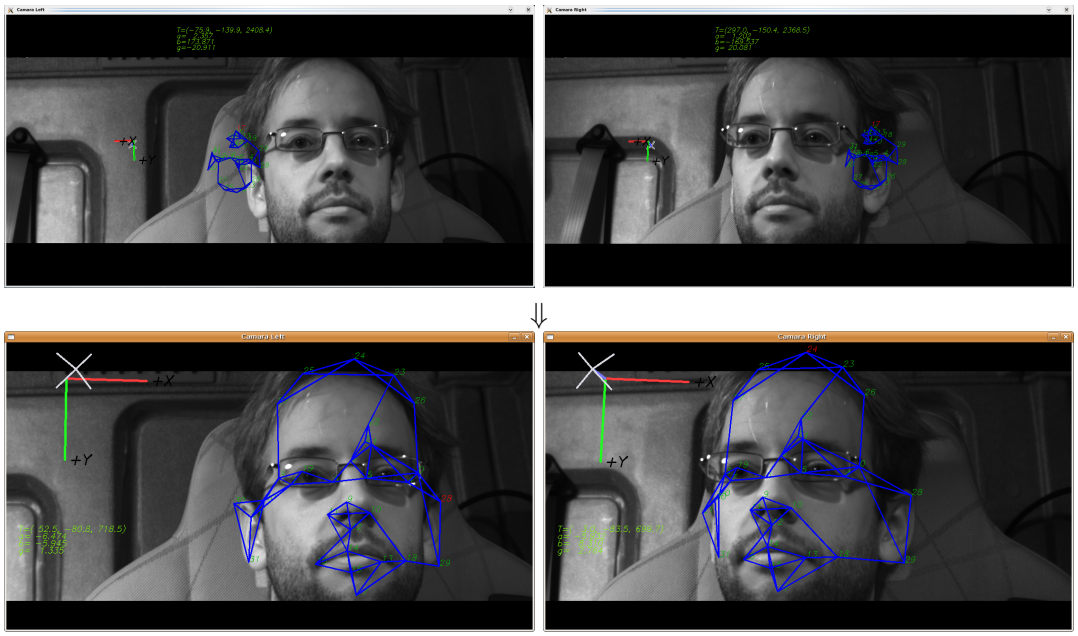


Figura 6.3: Ajuste final de imágenes en el software de tracking

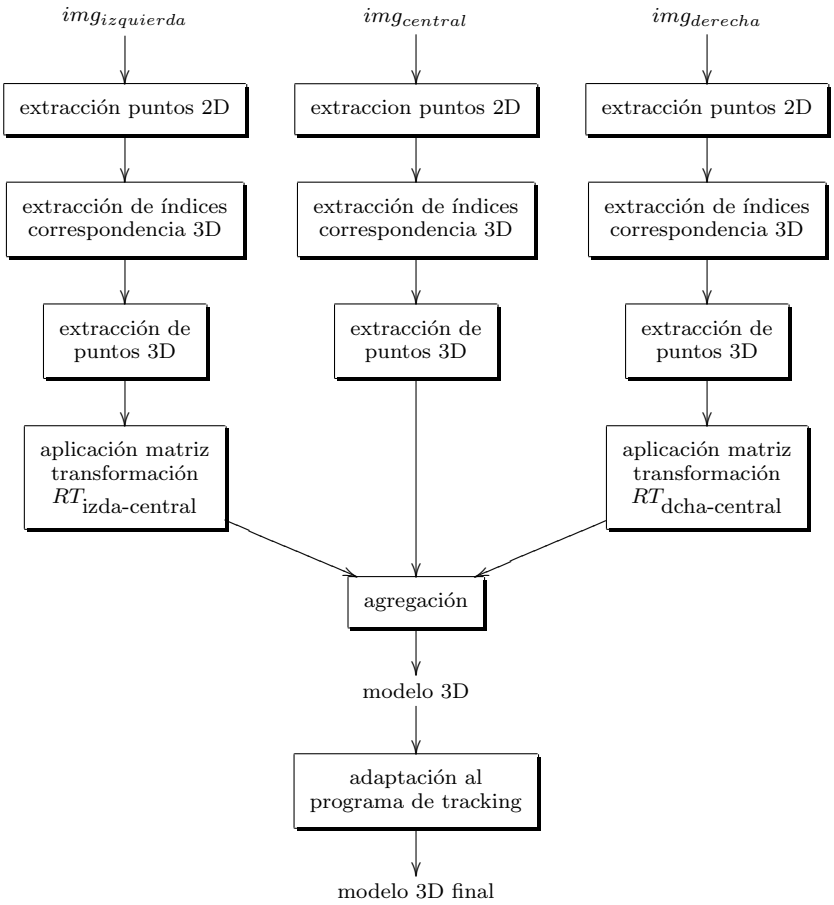


Figura 6.4: Esquema implementado para el método de diezmado automático

6.2.1. Procedimiento de obtención

A continuación, se detalla cómo se realiza la obtención del modelo diezmado

1. Se ejecuta el programa, y se va a menú->File->import->digitalizer->one scan / step scan, o en la propia barra de herramientas.

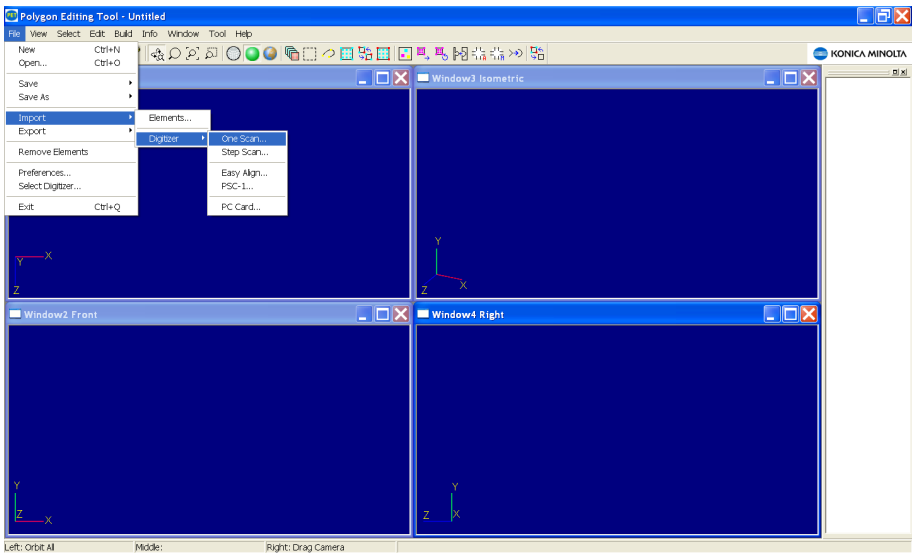


Figura 6.5: Captura de pantalla de la ventana principal para el acceso a la adquisición de datos

2. Se realiza la captura de las 3 vistas (frontal, derecha e izquierda) desde el scan-láser con el software de adquisición Poligon Editing Tool.

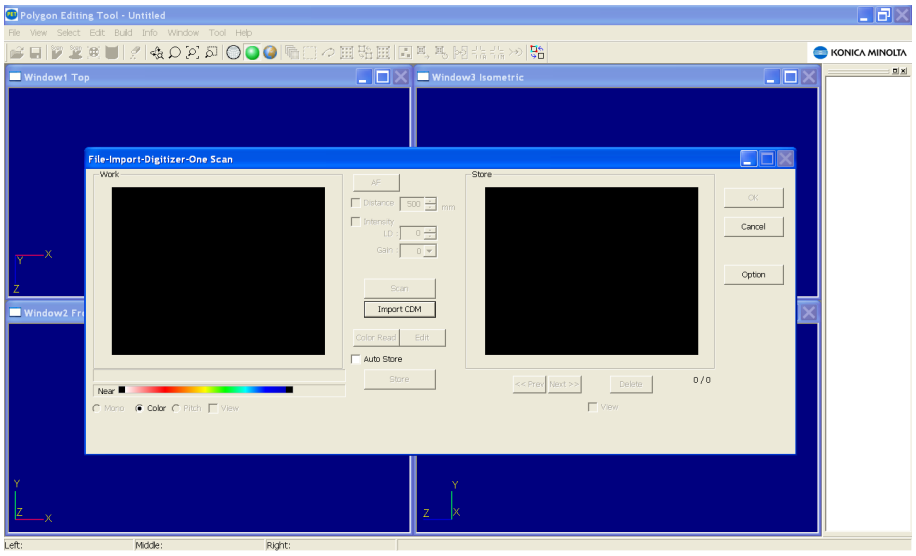


Figura 6.6: Pantalla de captura del software de adquisición

3. Se filtran los datos recortando datos 3D que no son interesantes, como por ejemplo la zona de la ropa, cuello o similar, respetando orejas y otros puntos de interés. También ha de eliminarse los puntos 3D del fondo, si es que existen.

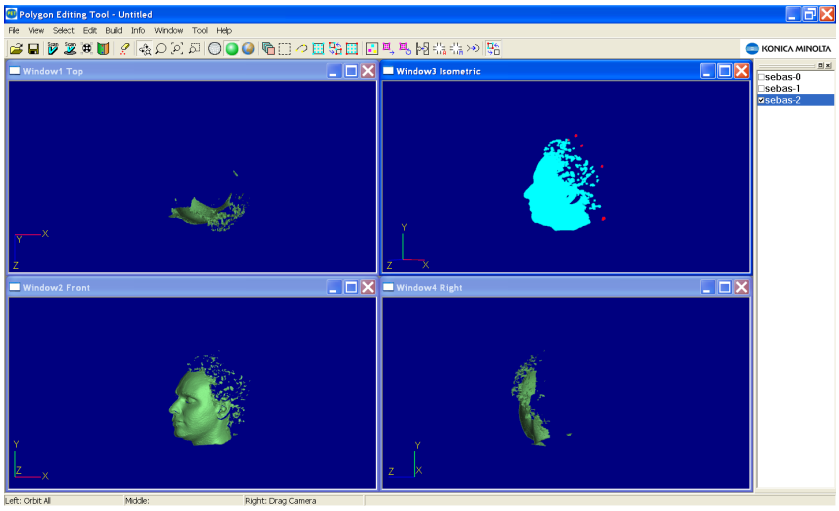


Figura 6.7: Eliminación de puntos de los escaneados originalmente, marcados en rojo para ser borrados posteriormente

4. Se convierten los archivos a formatos .wrl y .str. Es necesario transformar el archivo de entrada a estos 2 formatos por varias razones.

   a) En primer lugar, toda la parte inicial se desarrolló a partir de ficheros .wrl, en formato VRML 2.0, por eso la necesidad de convertir los datos en este tipo de archivo. Además estos tienen la particularidad a la hora de guardar de que se ha de forzar que la imagen esté incluida en el archivo, para que se genere también la tabla de índices que conectan la posición (u, v) en la imagen con el índice del punto correspondiente 3D.

   b) En segundo lugar se necesita la conversión al formato str al ser uno de los formatos compatibles entre el programa de captura y el software Meshlab.

5. Se extraen la imagen correspondiente a la captura, ya sea directamente a través del programa de captura o a través del archivo vrml con un script de Matlab desarrollado para este propósito.

6. Se especifica un archivo de configuración con los parámetros que va a tener la región de interés en la imagen, además del archivo de imagen y el nombre del archivo .wrl para cada una de las 3 tomas. De momento esta configuración es manual, aunque se va a implementar un programa de detección basado en Viola & Jones para implementar la detección automática de caras. Este archivo tendrá la siguiente estructura:

   a) IMAGEN: nombre y extensión de archivo de imagen de una toma.

   b) VRML: nombre y extensión del archivo .wrl que contiene los datos de la toma correspondiente a la imagen.

   c) ROIX: coordenada X inicial del área de interés donde se encuentre el objeto en la imagen.

d) ROIY: coordenada Y inicial del área de interés donde se encuentre el objeto en la imagen.

e) ROIWIDTH: anchura del área de análisis en la imagen.

f) ROIHEIGHT: altura del área de análisis en la imagen.

Un ejemplo de archivo completo de configuración es el siguiente:

```
IMAGEN: img_ivan_0.png
VRML: modelos3D_retocados/ivan-0_retocado.wrl
ROIX: 195
ROIY: 15
ROIWIDTH: 280
ROIHEIGHT: 375
IMAGEN: img_ivan_1.png
VRML: modelos3D_retocados/ivan-1_retocado.wrl
ROIX: 175
ROIY: 10
ROIWIDTH: 275
ROIHEIGHT: 370
IMAGEN: img_ivan_2.png
VRML: modelos3D_retocados/ivan-2_retocado.wrl
ROIX: 155
ROIY: 20
ROIWIDTH: 285
ROIHEIGHT: 385
```

7. Se aplicar el programa **vrmlextract**. Este programa extraerá los puntos de interés basándose en detección de esquinas por Harris. Devolverá como salida archivos binarios con la información de las coordenadas de los puntos de interés, los índices obtenidos correspondientes, la información 3D asociada y las matrices de translación iniciales asociadas al punto de vista y asociadas con la primera medida que realiza el escáner. La sintaxis a utilizar será la siguiente:

   ```
   ./vrmlextract archivo_configuracion.txt
   ```

   Un ejemplo con imágenes que ayudan a describir las distintas fases del proceso que realiza este programa, se muestra en la figura 6.8.

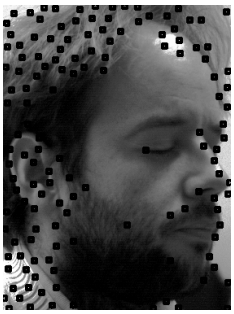
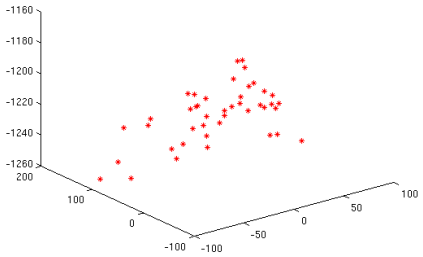
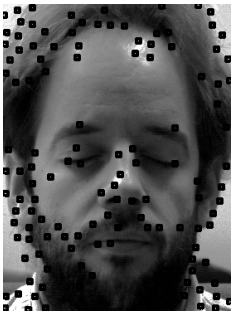
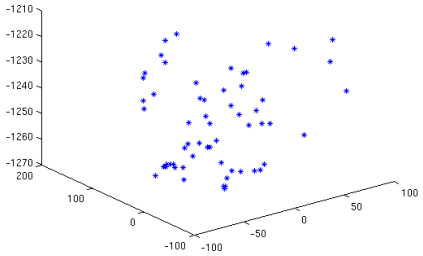
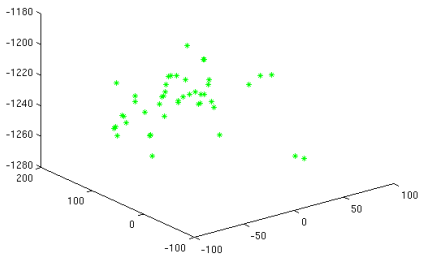
Imagen obtenida desde scan-láser	Esquinas detectadas en la cara	Puntos 3D asociados a esquinas extraídos
		
		
		

Figura 6.8: Proceso de extracción de puntos 3D desde 2D

8. Se obtenen las matrices de transformación entre vistas. Para ello no podemos utilizar el software que se provee con el scan-láser, puesto que no devuelve la información que necesitamos, que es el valor de las matrices utilizadas para la transformación a la hora de unir las mallas 3D de las distintas vistas. Por tanto, se recurre al programa libre **Meshlab**, aunque modificado para que saque por pantalla la información de las matrices de transformación necesarias para hacer que las mallas encajen.

   a) Dentro del programa Meshlab se leen todos los archivos de malla 3D. Una captura de pantalla en este punto del proceso se muestra en la figura 6.9.

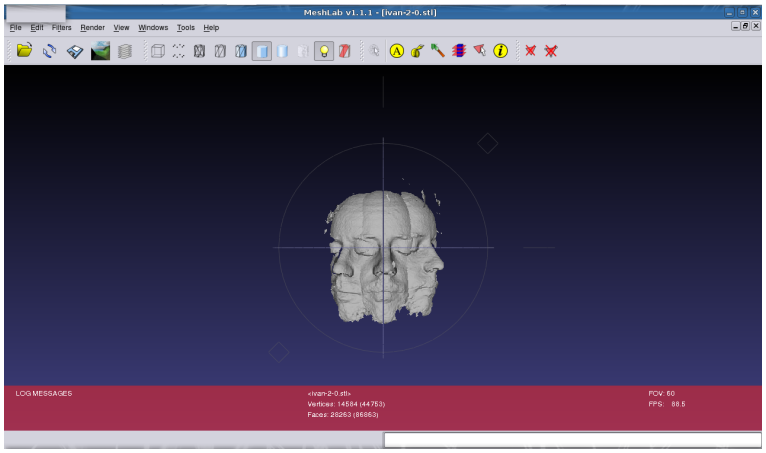


Figura 6.9: Pantalla principal de Meshlab, con modelos cargados

   b) Se fija una de las máscaras como punto de referencia de alineamiento, normalmente la frontal, y a partir de ella, se referencian las otras 2.

   c) Se aplica un "Point Based Glueing", en el cual se seleccionan 4 o más puntos. Un ejemplo de la selección de los primeros pares de puntos se muestra en la figura 6.10.

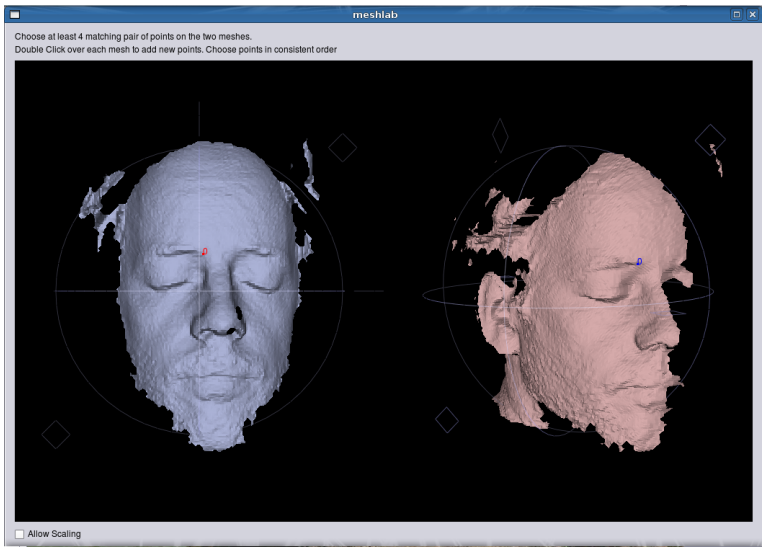


Figura 6.10: Pantalla de selección de puntos para alineación

d) Se calcula la matriz de translación y rotación con el botón process.

e) Se copian los datos de la matriz de transformación de coordenadas a Matlab para poder aplicar la transformación con los datos 3D obtenidos en los archivos binarios. Se recomienda llamar a la matriz TR_XY siendo x e y el número de la vista correspondiente, por ejemplo, TR_01 es la matriz de transformación de la vista 0 a la vista 1. Si la vista 1 es la vista frontal, se recomienda poner la referencia al principio, esto es, por ejemplo, TR_10 o TR_12 o lo que corresponda.

f) Se guarda esta matriz en un archivo .mat para poderla leer posteriormente.

9. Después se leen con el script de Matlab llamado **componer**, al cual se le pasa como argumentos el nombre del usuario que se quiere componer y el orden de las matrices que se va a aplicar. En caso de que la primera toma no sea la frontal y por tanto la referencia, el flag orden pasa a ser 1 en lugar de 0. Previamente se ha tenido que nombrar la matriz de rotación adecuadamente, como se ha recomendado anteriormente. Después de la aplicación de este script se obtienen varios archivos de salida, uno en formato binario y otro en formato Matlab con el modelo de la cara diezmado y unido las distintas facetas de la cara. Un ejemplo de la representación de la máscara obtenida después de componer se muestra en la figura 6.11.

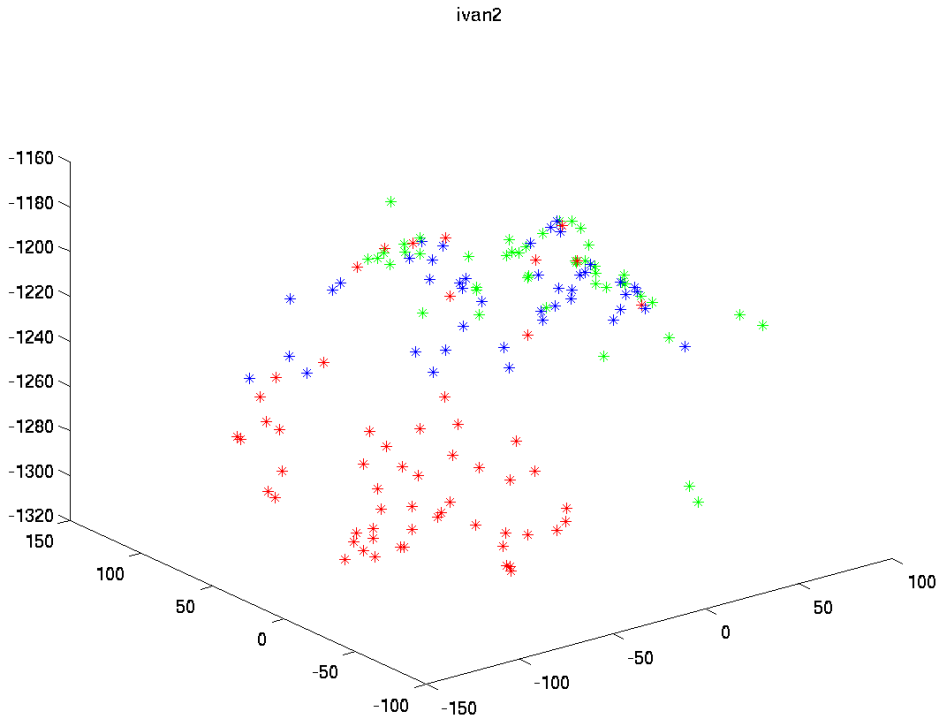


Figura 6.11: Ejemplo de modelo 3D agregado

10. Por último, para convertir el archivo al formato adecuado para ser leído por el programa de tracking 3D, se ha de aplicar el programa **modelfrom3D** al que se han de pasar como

parámetros, la matriz de calibración de la cámara para unos determinados vídeos, y la matriz de datos 3D, ya sea en formato binario o en formatos anteriores del programa (para asegurar la compatibilidad con versiones anteriores). La necesidad de la introducción de las matrices de calibración viene dada porque el modelo va a ser aplicado a un vídeo concreto, y ese vídeo es la consecuencia de la proyección de una escena real, por tanto, ahí entra la necesidad de la matriz de calibración. Para poder colocar la malla posteriormente, se aplica una traslación incial $T = \begin{pmatrix} 0 & -30 & 700 \end{pmatrix}$.

11. Los archivos generados de salida, output.dat, output.dat2d y output.dat3d, se copiarán con el mismo nombre que el vídeo de destino en la misma carpeta que el vídeo, conservando las extensiones.

12. Con los archivos de salida generados, pasamos a su lectura y ajuste manual para que encajen dentro del software de tracking. Un ejemplo de un ajuste manual realizado es el mostrado en la figura 6.12.

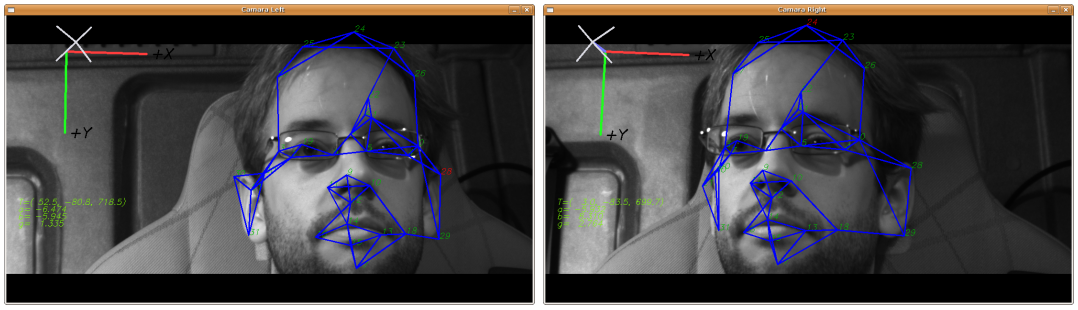


Figura 6.12: Imágenes ajustadas manualmente en el programa de tracking

6.2.2. Ángulos de ocultamiento automáticos

Al tener que configurar un gran número de ángulos de ocultamiento a mano al tener muchos puntos, se ha optado por implementar una estimación que ayudase a configurarlos automáticamente.

Se ha observado que el sistema de coordenadas calculado para el modelo no da una dirección exacta de la cabeza, sino que da una estimación bastante aproximada, por lo que, si queremos calcular los ángulos de ocultamiento automáticamente, se ha de tener en cuenta varias cosas que se pasan a enumerar en esta sección.

En primer lugar se calcula el centro de giro de la cabeza utilizando una función que supone la cabeza como un cilindro, proyecta los puntos en el eje Y y estima el mejor círculo que modela la cabeza, con un centro y un radio, teniendo en cuenta la cabeza completa, incluyendo las orejas. Esta función ya estaba implementada, pero con modelos con un número bajo de puntos, no era demasiado precisa ni útil, puesto que no se tarda tanto en configurar los ángulos manualmente.

En segundo lugar, se calcula el punto más alejado en ese plano (esto es, sin tener en cuenta la coordenada Y de los puntos 3D), y se calcula el punto más alejado del centro calculado con RANSAC. Se toma este punto como aproximación del pico de la nariz, que realmente es un buen indicador de la dirección de la cabeza.

Una vez seleccionado este punto como referencia, tomamos la recta entre el centro y este punto como la recta de la cual surgen todos los ángulos. Se calcula este ángulo como referencia y se resta al resto de ángulos que se obtengan. El resto de ángulos se calculan como el arcotangente de la coordenada que va desde el centro hasta el frente, entre la coordenada que va desde el centro hacia el lateral.

Además, los ángulos tienen distinto márgen según el grado de rotación de la cabeza, puesto que no es esférica. Esto implica que pasado determinados ángulos, hay puntos que tienen más márgen de visibilidad que otros. Los puntos frontales de la cara, en cuanto ésta supera ciertos ángulos, se ocultan rápidamente, mientras que los puntos laterales tienen un márgen de visibilidad superior, por lo que se tienen en cuenta también estos aspectos a la hora de asignar los ángulos. Una ilustración de lo que se quiere explicar con los puntos explicados anteriormente se encuentra en la figura 6.13.

Con este método conseguimos una configuración aproximada de los ángulos de ocultación, suponiendo un modelo de cara cilíndrico y después realizando correcciones se obtienen ángulos que son aceptables y parecidos a los que de una configuración manual se obtendrían.

El problema que tienen estos ángulos es que son simétricos, esto es, el márgen hacia un lado es el mismo que el que va hacia el otro lado. Esto no se cumple en puntos cercanos a la nariz, puesto que ésta hace sombra, pero esta parte es más complicada y no está implementada todavía.

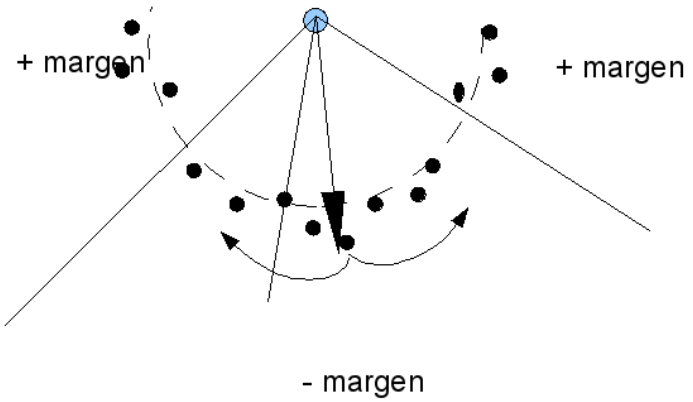


Figura 6.13: Esquema de establecimiento de ángulos de ocultamiento mediante aproximación cilíndrica

Capítulo 7

Resultados

En este capítulo pasamos a exponer los resultados obtenidos en las distintas partes del sistema explicadas anteriormente.

En primer lugar, se presentarán unos primeros resultados de las pruebas preliminares con un cubo aplicando POSIT en Matlab, comparando la versión modificada con la inicial, en función de distintos parámetros, para tener una primera idea de la mejora que ésto supone.

A continuación se traducirá POSIT a C/C++ y se integrará la función en el sofware final para comprobar si la optimización es todo lo buena que debería.

Por último se probará la introducción del modelo 3D adquirido mediante escáner al software final y se compararán resultados con los obtenidos con modelos creados manualmente.

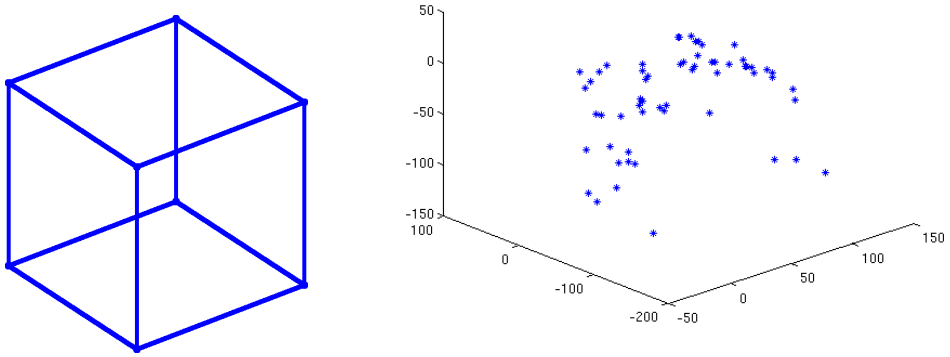
7.1. Resultados preliminares de POSIT vs mPOSIT con Matlab y con un cubo como modelo

7.1.1. Introducción

Con las mejoras implementadas en el algoritmo POSIT, comentadas en el apartado 6.1, se procede a probar con un cubo cuyas coordenadas son las siguientes:

$$cubo = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \tag{7.1}$$

cuya representación en 3D se representa en la figura 7.1. También se realizará un barrido para ver el rendimiento de POSIT si se varía el número de puntos, para lo cual se necesita un modelo más denso. Se utilizará un modelo diezmado aplicando el método propuesto en esta Tesis de Máster, el cual se muestra también en la figura 7.1.



(a) cubo (b) cara

Figura 7.1: Modeos utilizados para los experimentos de evaluación de POSIT

Inicialmente se realizará un experimento para evaluar cómo afecta la rotación en el número de iteraciones. Para ello se aplicarán rotaciones con diversos pasos en radianes. En el resto de experimentos este valor se mantendrá fijo y será de $\frac{\pi}{32}$. La rotación se aplicará al cubo, a través de una matriz de transformación homogénea según lo visto en la sección 4.5 dedicada a las transformaciones espaciales. La matriz de transformación se multiplica por los puntos en 3D y, teniendo en cuenta las ecuaciones de proyección descritas en la sección 4.2, se obtiene la proyección simulada en la cámara. Existen otras formas de representarlo, como por ejemplo la función plot3 de Matlab, que realiza una presentación en 3D de los datos introducidos, aunque esta función introduce una perspectiva cuyos ejes pueden no tener la misma norma para representar el objeto entero, lo cual puede distorsionar la representación del objeto, lo que obligará a introducir ciertas transformaciones que se analizarán más adelante.

Los parámetros utilizados en POSIT para la ejecución de los tests son los siguientes:

- Las coordenadas 2D de los puntos proyectados en la imagen.
- Los puntos 3D del modelo
- La distancia focal, para estos ejemplos se ha fijado en 100, salvo el caso en el que se ha realizado un barrido en este parámetro, o que por circunstancias del modelo u otros parámetros el algoritmo no converja.
- El error interno de POSIT, en su implementación de Matlab es fijo, y es igual a 1. El número de iteraciones no está limitado. Si no converge el algoritmo se queda constantemente intentando minimizar el error. En otras palabras, cae en un mínimo local dentro del cual no encuentra una solución óptima, y al no tener inercia para salir de ese mínimo local, queda buscando la solución en ese entorno indefinidamente.
- En la variante modificada de POSIT, para las iteraciones que no son la inicial, se introduce la estimación anterior de la matriz de rotación y translación.

Todos los experimentos medirán distintos errores medidos sobre los parámetros de POSIT: el parámetro de error interno al algoritmo y el error de reproyección a 2D.

Además se obtienen datos para comprobar el número de iteraciones con las que se ha ejecutado el algoritmo, así podemos comparar la optimización real a la que se llega. Se pondera la

optimización por el número de veces evaluadas por lo que se obtiene un ratio que nos da una idea de cuántas iteraciones estamos ahorrando por cada ejecución del algoritmo. Este ratio se calcula para cada ángulo (α, β, γ) de la siguiente forma:

$$r_\alpha = \frac{1}{\#pasos} \sum_{i=1}^{N} cont_\alpha(i) - cont\_mod_\alpha(i) \tag{7.2}$$

En este caso el número de pasos se corresponde con el número de frames sintéticos evaluados y por tanto con el número de veces a las que se ha llamado al algoritmo, con lo que este ratio se puede considerar como una media de la diferencia de iteraciones. En cuanto al error devuelto por la función de POSIT y el error de reproyección se evalúan la media y la desviación típica de cada uno de los experimentos completos (con 32 pasos para todos los casos que no se realizan barridos en el paso de rotación).

Para calcular el error de reproyección se utiliza la distancia euclídea a todos los puntos, expresado en la manera habitual en la ecuación 7.3 y en forma matricial en la ecuación 7.4.

$$error = \sqrt{\sum_{i=1}^{n} (u_i - u_i')^2 + (v_i - v_i')^2} \tag{7.3}$$

$$error = \sqrt{\sum_{i=1}^{n} traza\left((P_i - P_i')(P_i - P_i')^T\right)} \tag{7.4}$$

siendo P la matriz de los puntos proyectados sobre el plano imagen del modelo 3D utilizado y $P_i = (u_i, v_i)$ las coordenadas de cada punto. P' es la matriz de puntos obtenidos a partir de la proyección con las matrices devueltas por POSIT o mPOSIT.

En cuanto a las unidades utilizadas, en lo referente a dimensiones espaciales no son especificas, esto es, son unidades métricas genéricas. No tienen por qué estar expresados en mm, cm o m, sino que todo lo que se realiza en estos test es proporcional. Si se aumenta el tamaño del modelo del cubo, también ha de hacerse lo propio con otras medidas, como por ejemplo la distacia focal o los ejes utilizados como base del espacio. En cuanto a las unidades de ángulos, se toman en radianes, puesto que los parámetros de entrada de funciones como el seno o el coseno tienen esta magnitud.

7.1.2. Resultados con variación del paso de rotación

En este apartado se presentan los resultados de la comparativa de POSIT - mPOSIT variando el paso que se le da a la rotación de la matriz de transformación. Dicho de otro modo variamos la resolución con la que el ángulo que estamos evaluando va evolucionando. Este experimento es uno de los más interesantes, puesto que se comprueba la robustez del algoritmo cuando se tiene una predicción buena anterior, logrando reducir el número de iteraciones.

Se supone que cuanto más pequeño sea este paso, menor es la diferencia entre poses y por tanto, más cercana estará la matriz de rotación anterior de la actual, y por tanto mPOSIT se ejecutaría más rápido. Esta aproximación se puede considerar en el caso de que los movimientos del objeto sean lentos, puesto que si no, nos alejamos de una matriz válida y el número de iteraciones para corregirla aumentará, aunque la predicción ayudará a reducir el número de aproximaciones que se tenga que realizar.

A continuación se representan los resultados obtenidos para alguno de los pasos en los ángulos más representativos de los que se han realizado barridos y posteriormente se comentarán los resultados del barrido completo

7.1.2.1. paso de $\frac{\pi}{4}$

En este caso se presentan los resultados obtenidos con un paso de $\frac{\pi}{4}$, esto es, de 45 º en la rotación. Esta rotación en la realidad no es un caso que se vaya a presentar frecuentemente, pero aún así se analizarán los resultados obtenidos. Esta rotación es algo extrema puesto que en 33.3 ms es complicado rotar la cabeza un ángulo como ese, pero es uno de los peores casos posibles en la estimación de la pose entre frames, por lo que se analiza a continuación.

Para este caso, en la figura 7.2.a se muestra la gráfica que contiene el resultado de la reproyección, para los 2 algoritmos, junto con la proyección de puntos original, para tener una idea del error real que se comete con el algoritmo. También se muestra la gráfica del error devuelto por el algoritmo, en la figura 7.2.b. A continuación se muestra en la figura 7.2.c una gráfica representativa con el error de reproyección y por último, en la figura 7.2.d se presenta una gráfica con el número de iteraciones en función del frame sintético que esté procesando.

Realizando las gráficas para todos los pasos que se han estudiado, se ha visto que algunas son versiones submuestreadas de la final, por lo que, se mostrará solo 1, mientras que para el caso más representativo de todos, el paso de $\frac{\pi}{32}$, se muestran las gráficas para los 3 ángulos del espacio.

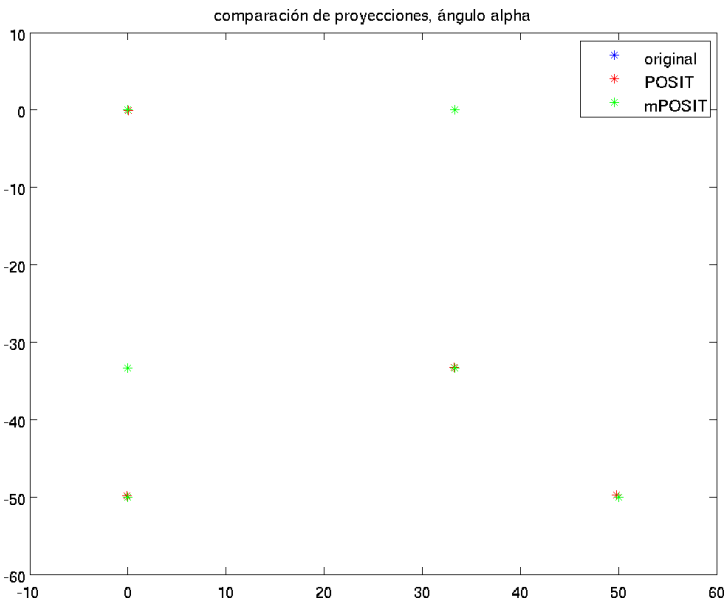
7.1.2.2. paso de $\frac{\pi}{32}$

Por último se analizan los resultados obtenidos para un paso de ángulo de $\frac{\pi}{32}$, lo cual equivale a 5.125^{o}. Estas rotaciones se dan cuando por ejemplo el coche está vibrando un poco, por las pequeñas irregularidades que pueda tener la carretera a lo largo de la conducción, o pequeños movimientos de la cabeza para prestar atención a objetos que se encuentran de frente.

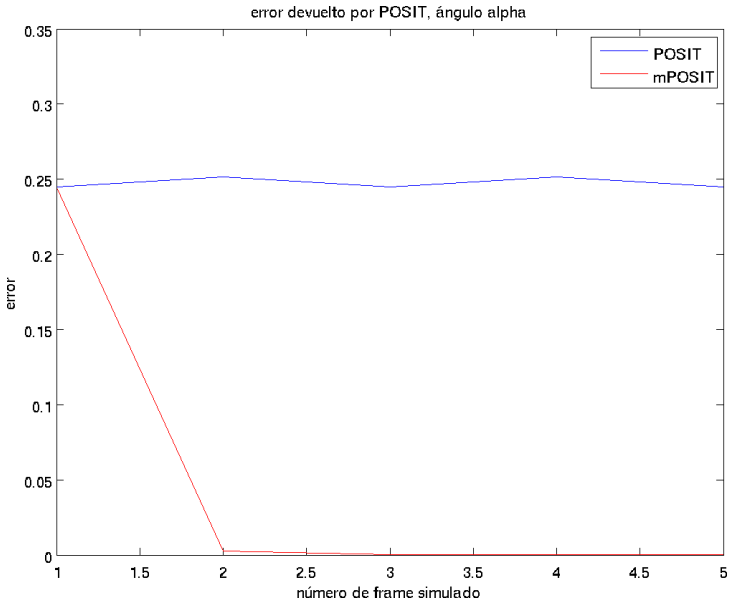
Como en el apartado anterior, se muestran en primer lugar las proyecciones simuladas y las obtenidas mediante POSIT y mPOSIT en la figura 7.3.a. A continuación se muestran las gráficas de error interno del algoritmo en la figura 7.3.b, el error de los puntos reproyectados en la figura 7.3.c y por último se mostrarán las gráficas de la evolución del número de iteraciones en la figura 7.3.d, para el ángulo alpha. Las correspondientes gráficas se representan igualmente para los ángulos beta y gamma en las figuras 7.4 y 7.5 respectivamente.

No se analizan casos de pasos de menor tamaño puesto que se ha visto que el algoritmo mejora su velocidad de ejecución para variaciones de ángulo entre frames pequeñas y por debajo de estas variaciones el algoritmo no tiene demasiado márgen de mejora y se empezaría a hablar ya de ruido por el hecho de que la imagen sea discreta (píxeles).

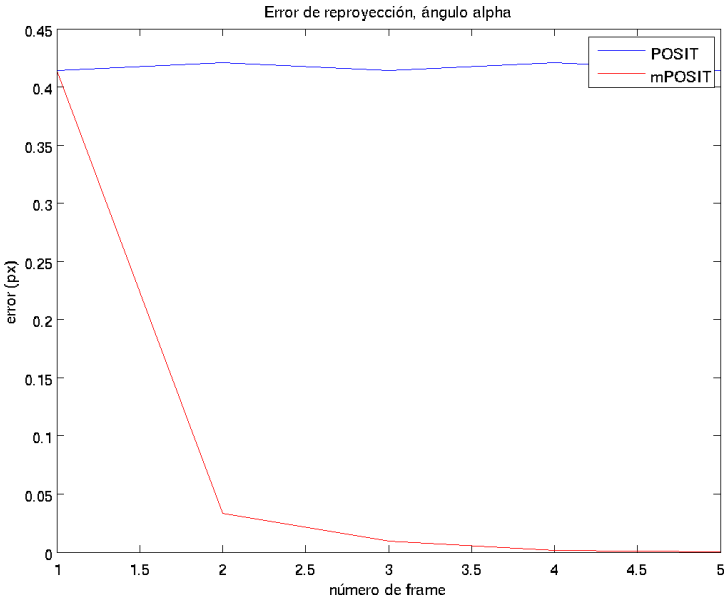
Como se puede apreciar en las figuras 7.3, 7.4 y 7.5, el error interno del algoritmo, que puede ser recogido a la salida, no es significativo dependiendo del ángulo, puesto que si vemos que para el ángulo alpha nos puede dar una buena aproximación de cuál va a ser el error cometido en el cálculo de la proyección en la cámara, en el resto de ángulos de giro esto no se cumple. Además, este error va a venir limitado por el márgen de error y por el número de iteraciones máximo que se le haya impuesto como condición de parada. El error obtenido al reproyectar los puntos



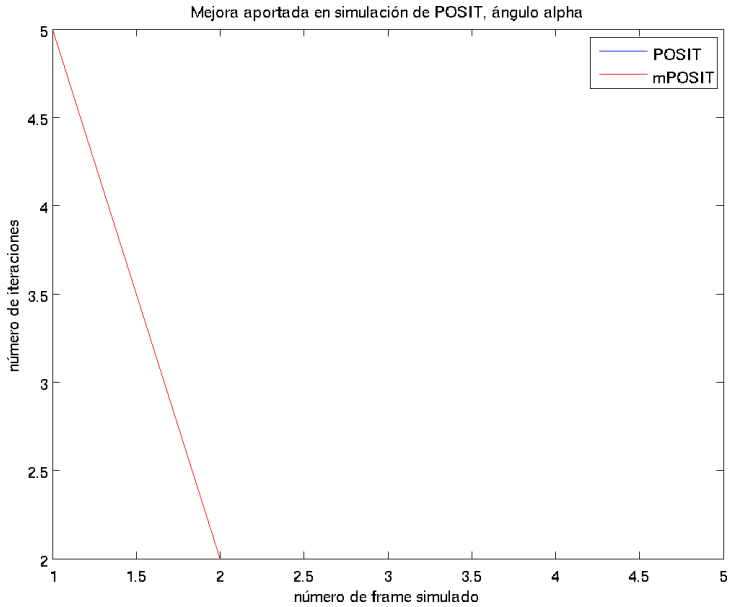
(a) Resultados de reproyección para la iteración 16



(b) error interno del algoritmo



(c) Evolución del error de reproyección



(d) Evolución del número de iteraciones

Figura 7.2: Resultados obtenidos para un paso de $\pi/4$ en el barrido en el ángulo α

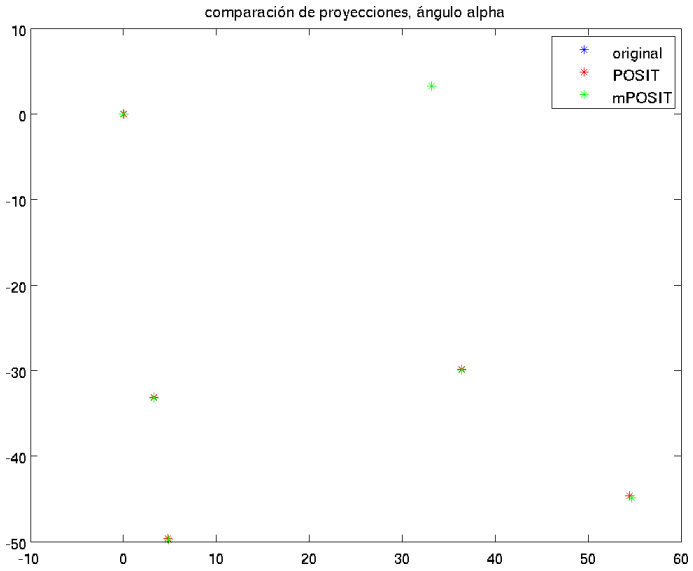
3D sobre el plano 2D con las matrices estimadas nos da una mejor idea de cual ha sido el error obtenido en la estimación.

Respecto a la reproyección del error, se puede observar que es fuertemente dependiente del ángulo. Así como en el ángulo alpha hay un fuerte decremento del mismo cuando estamos ejecutando el algoritmo mPOSIT respecto de su predecesor, no se puede decir lo mismo del ángulo beta y gamma. En las gráficas en las que se comparan las distintas proyecciones se puede apreciar este hecho, así como en las gráficas de error de reproyección.

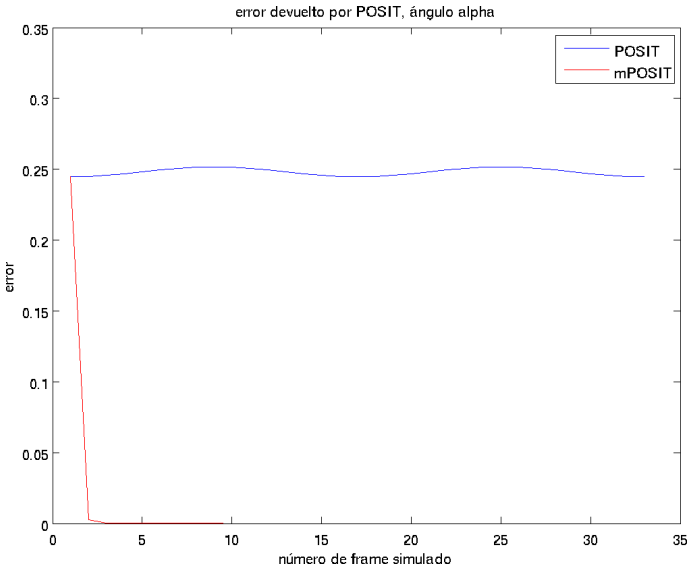
También se puede comprobar que las gráficas de los ángulos beta y gamma son iguales, puesto que a la hora del cómputo del error, las diferencias entre puntos proyectados, no son

iguales entre puntos correspondientes, pero las diferencias de unos puntos se contrarestan con las de otros, y el resultado numérico del error es el mismo, por tanto estas gráficas son iguales. Esto es debido a las simetrías existentes en el modelo utilizado, puesto que todos los lados son iguales todos los ángulos rectos, todas las caras iguales, etc.

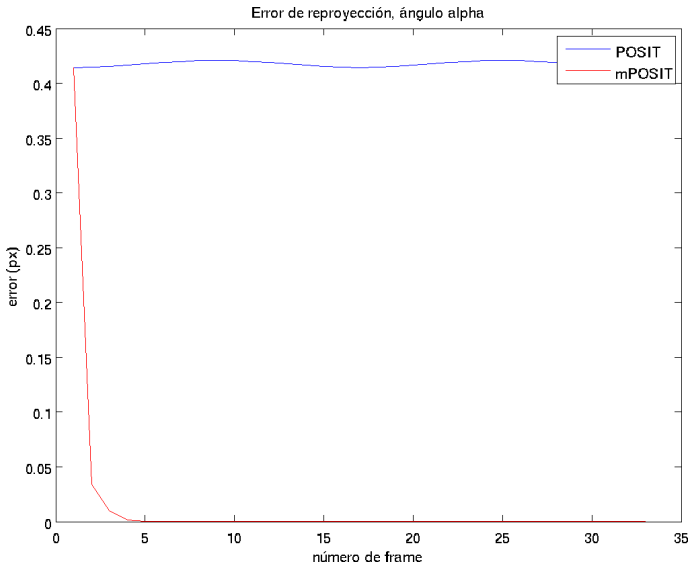
Respecto a las gráficas del número de iteraciones, aunque no se pueda apreciar bien en la subfigura 7.3.d debido a la reducción del tamaño, la gráfica de POSIT (azul) está continuamente en 5 iteraciones, mientras que en el resto de iteraciones se ha bajado a 2, puesto que la estimación previa es suficientemente buena, para que, con un par de correcciones en la matriz de rotación el algoritmo converja.



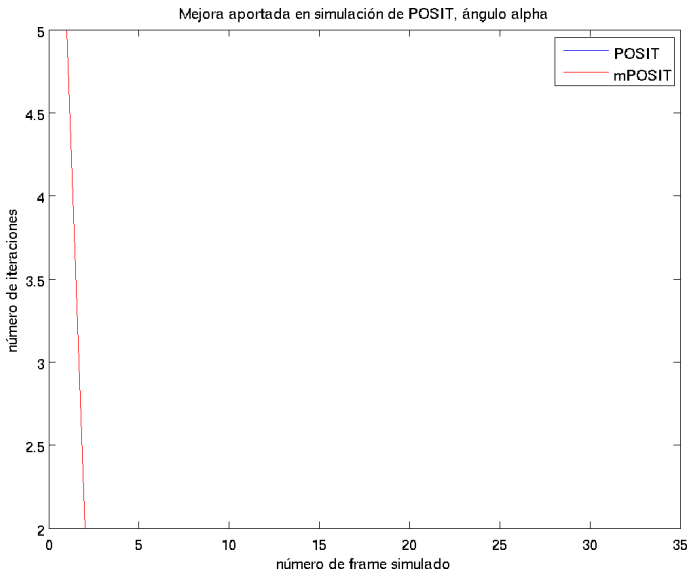
(a) Resultados de reproyección para la iteración 16



(b) error interno del algoritmo

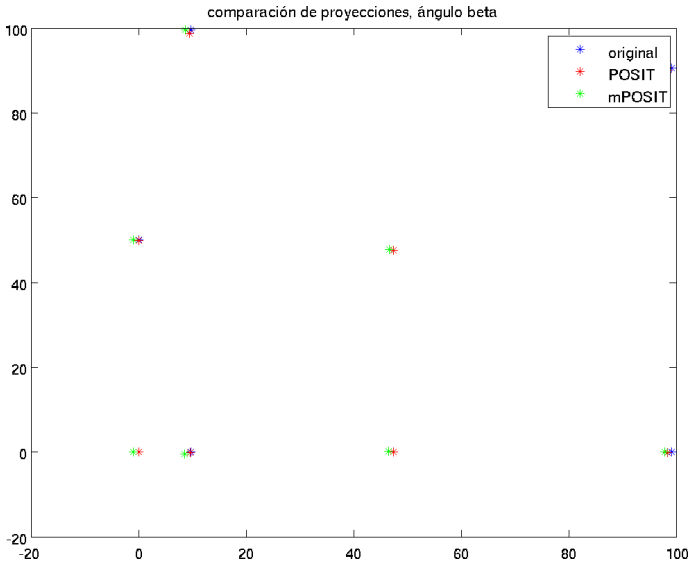


(c) Evolución del error de reproyección

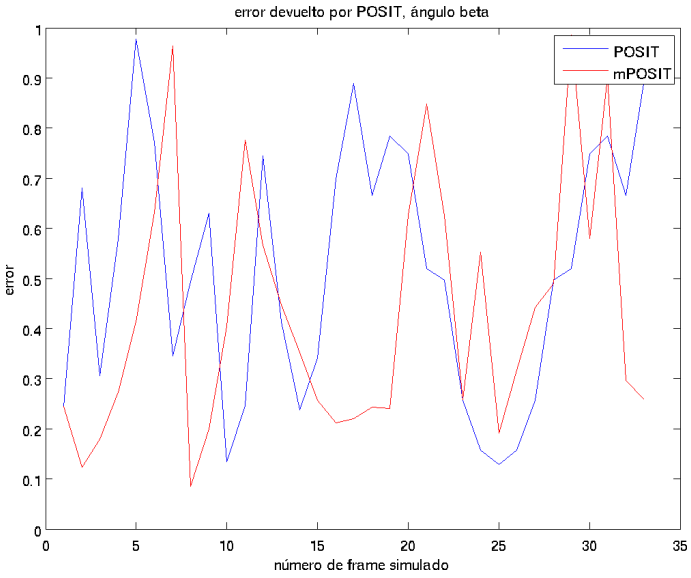


(d) Evolución del número de iteraciones

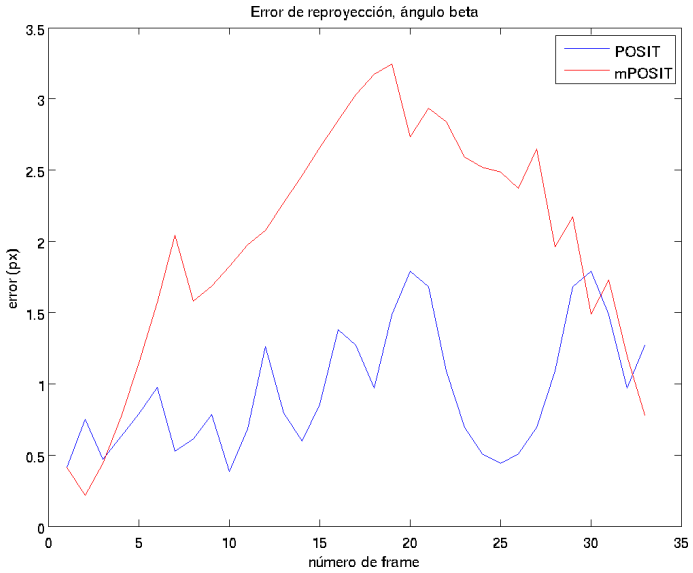
Figura 7.3: Resultados obtenidos para un paso de $\pi/32$ en el barrido en el ángulo α



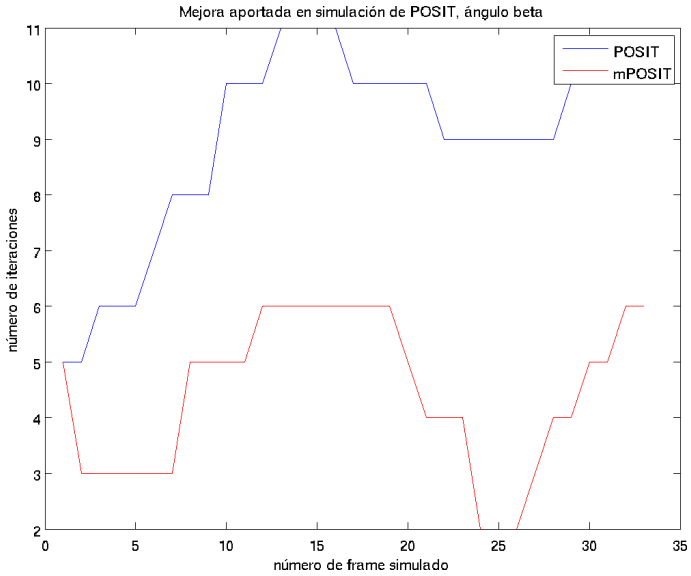
(a) Resultados de reproyección para la iteración 16



(b) error interno del algoritmo

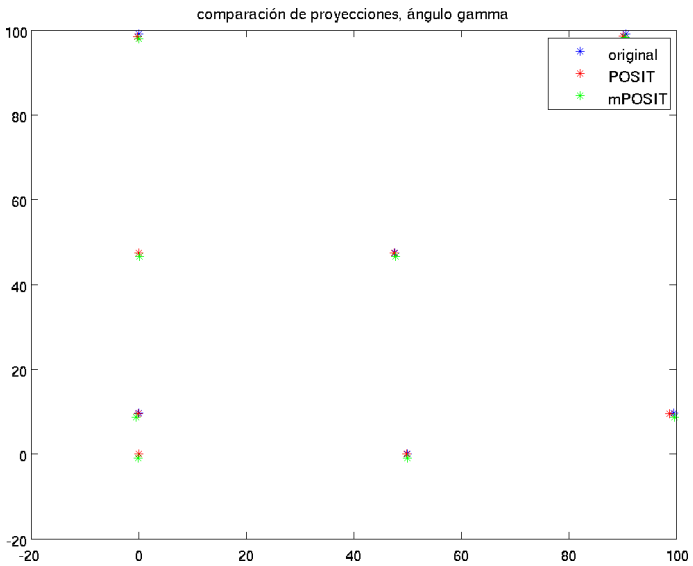


(c) Evolución del error de reproyección

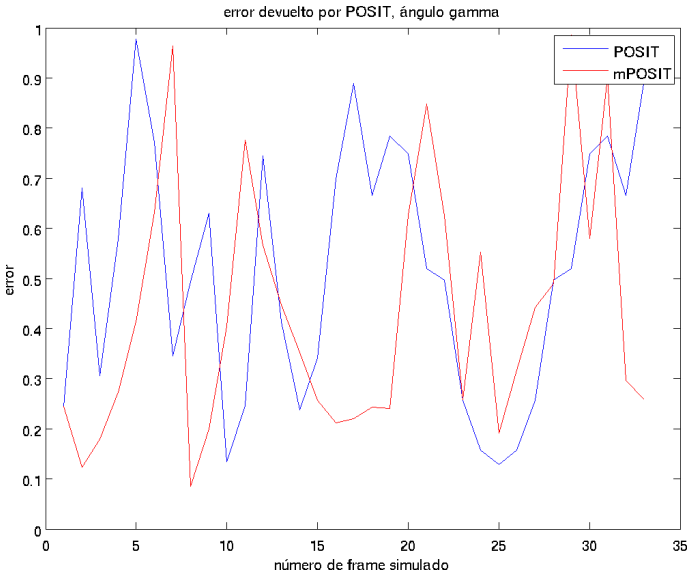


(d) Evolución del número de iteraciones

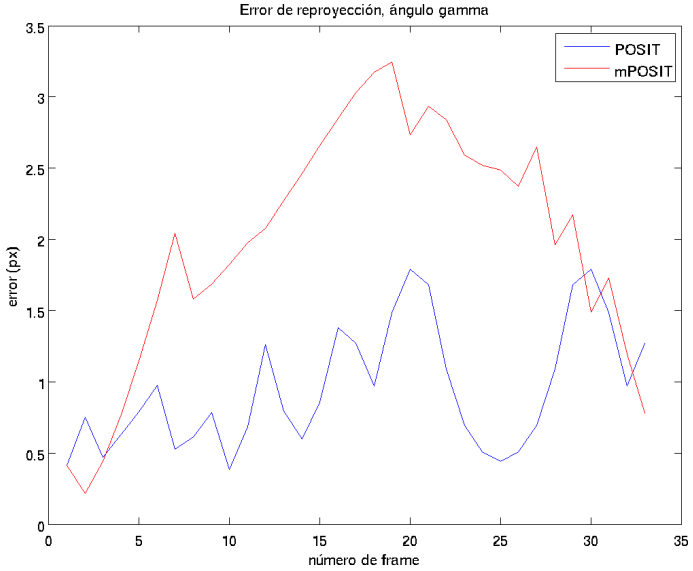
Figura 7.4: Resultados obtenidos para un paso de $\pi/32$ en el barrido en el ángulo β



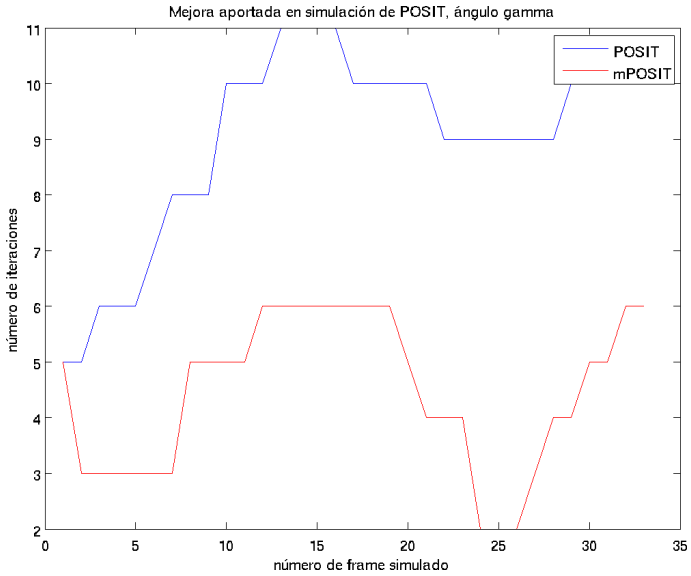
(a) Resultados de reproyección para la iteración 16



(b) error interno del algoritmo



(c) Evolución del error de reproyección



(d) Evolución del número de iteraciones

Figura 7.5: Resultados obtenidos para un paso de $\pi/32$ en el barrido en el ángulo γ

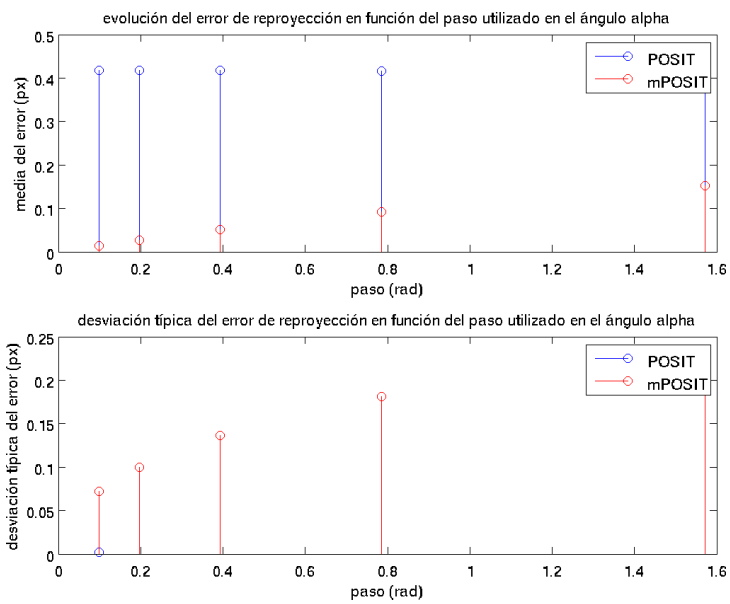
7.1.2.3. Comparativa del barrido completo

En este último apartado mostramos la comparativa de algunos parámetros de salida del algorimo para los distintos pasos que se han evaluado anteriormente.

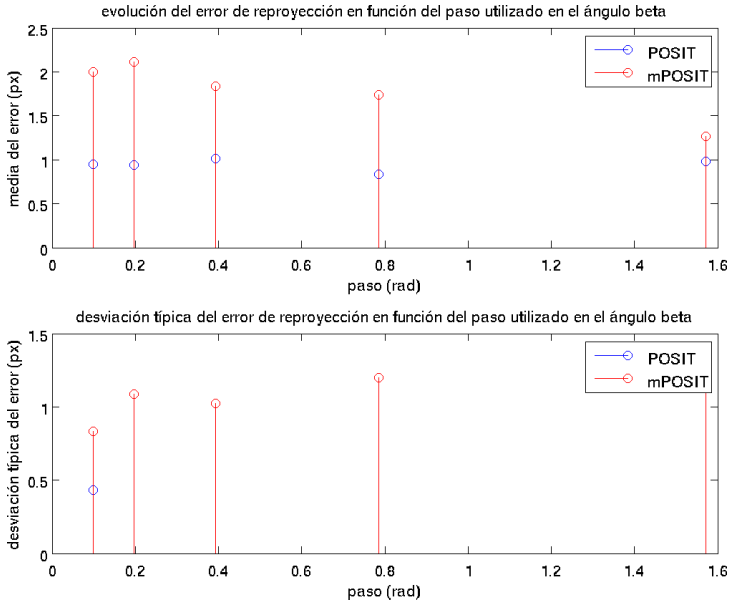
En primer lugar se muestra en la figura 7.6 las gráficas para los ángulos alpha, beta y gamma del error de reproyección, estudiando su media y desviación típica para cada paso evaluado. Por último se muestra en la figura 7.7 las gráficas para los distintos ángulos de la mejora en el número de iteraciones del algoritmo que ha supuesto la mejora. Las gráficas del error interno devuelto por el algoritmo no se muestran, puesto que, como se ha dicho antes, el error interno no supone

una medida de error fiable, por lo que no es interesante su más profundo estudio.

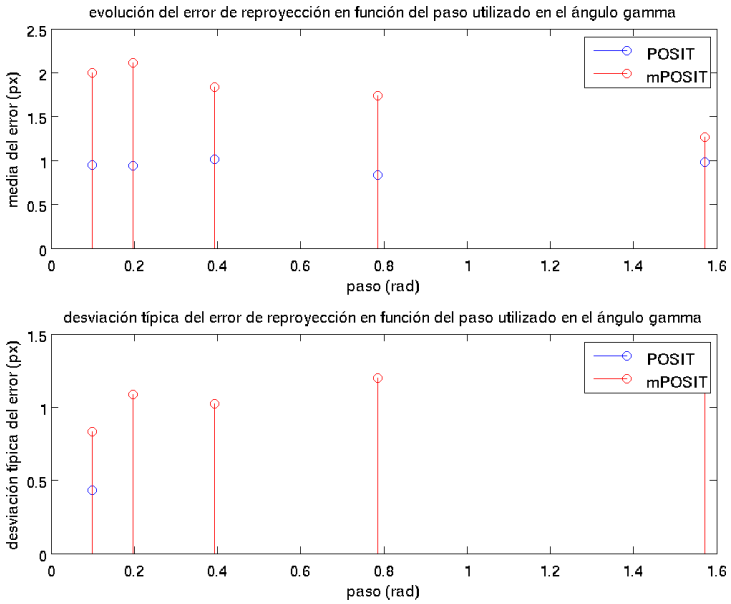
Como se ha observado en el apartado anterior, la tendencia vuelve a repetirse. En las gráficas asociadas al ángulo alpha, se ve cómo el error de reproyección se ve reducido, en su valor medio respecto del algoritmo original. Se aprecia asimismo la mejora en el número medio de iteraciones que se reducen con respecto al algoritmo original por frame para todos los casos, siendo de 3 iteraciones por frame, sobre un total de 5 que necesita POSIT.



(a) media y desviación típica para el ángulo α



(b) media y desviación típica para el ángulo β



(c) media y desviación típica para el ángulo γ

Figura 7.6: Comparativa del error de proyección según el paso utilizado

En cuanto al resto de ángulos se aprecia otra vez que las gráficas se repiten y además, tanto la media del error de reproyección como su desviación típica son mayores en el algoritmo modificado que en el normal. En cuanto a la mejora en el número de iteraciones que se reducen por frame no es tan buena como en el ángulo alpha. Los mejores resultados se obtienen con un paso menor y según vamos aumentando el paso en el ángulo, el ratio de mejora va decreciendo e incluso haciéndose negativo en algunos casos.

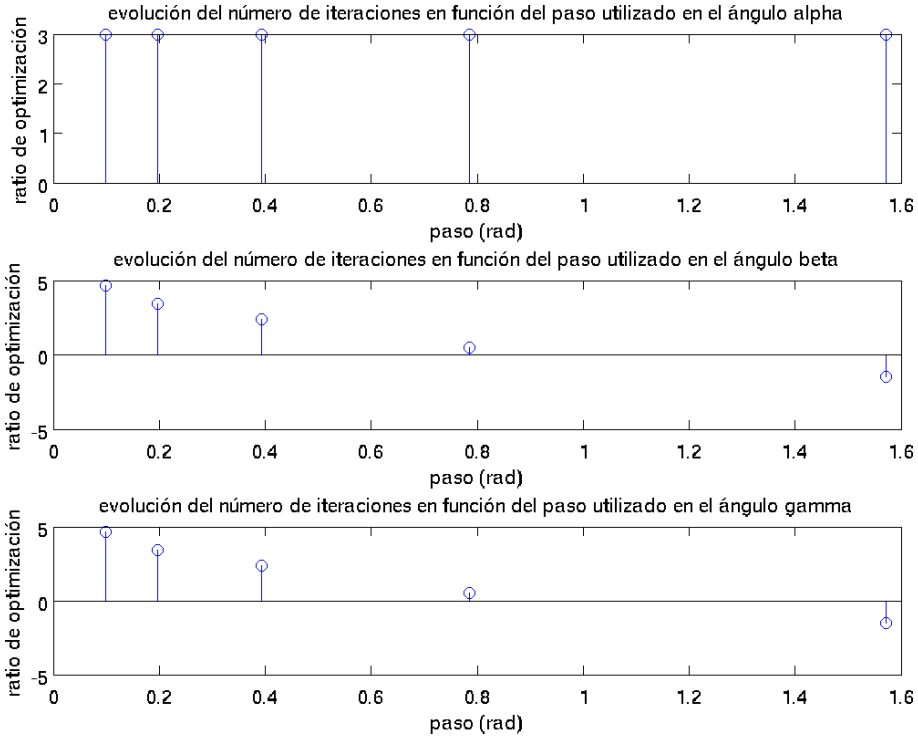


Figura 7.7: Comparación del ratio de mejora en el número de iteraciones según el paso

7.1.2.4. Conclusiones parciales de dependencia con el paso de variación del ángulo

Vemos que, si la variación del ángulo entre frames consecutivos no es muy alta, el algoritmo modificado funciona bien al no necesitar más iteraciones, si no, el ahorro del número de iteraciones no solo disminuye sino que además puede ser hasta negativo.

El error interno del algoritmo no nos da tanta información como el error de reproyección, al estar condicionado por el número de iteraciones y por el umbral que le hemos puesto, además de porque se calcula a partir de la iteración anterior, no desde la inicial, esto es, de la diferencia de la aproximación anterior con la actual, con lo que no puede tomarse como una referencia absoluta y no valdría como condición de parada para el algoritmo RANSAC.

El comportamiento del algoritmo es variable según el ángulo que se esté considerando. Así como con el ángulo alpha funciona bien, con beta y gamma el comportamiento no es tan óptimo.

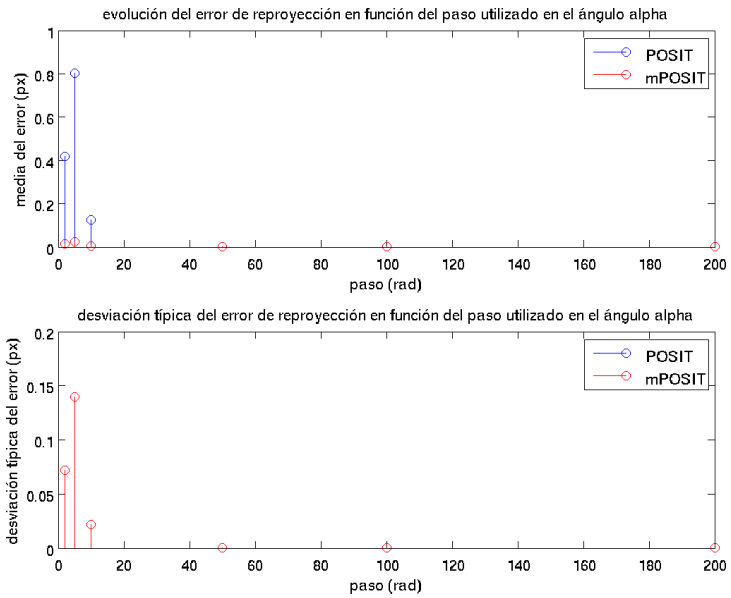
El error que existe al reproyectar los puntos 3D a 2D, es algo inferior en el algoritmo original comparándolo con el algoritmo modificado.

7.1.3. Resultados con variación de la translación en el eje Z

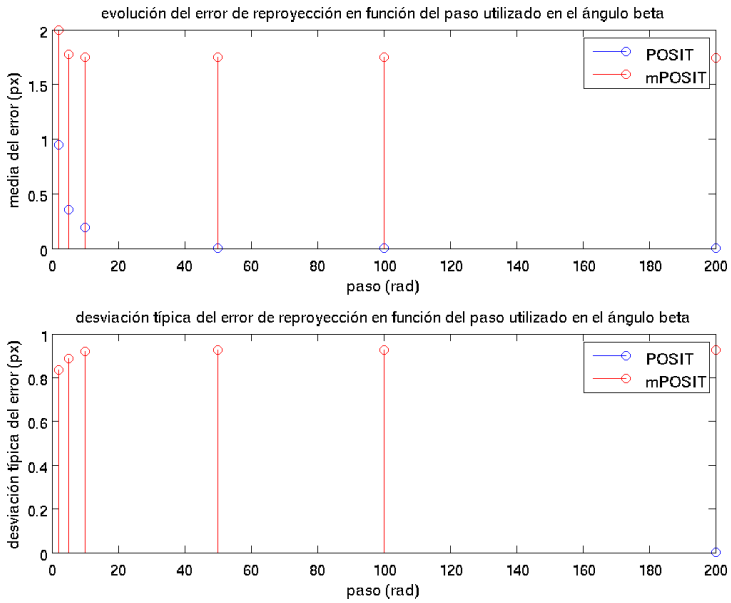
En este caso, se pretende evaluar cómo afecta al algoritmo POSIT los cambios en la translación en el eje Z, esto es, la profundidad a la que se encuentra el objeto visto desde la cámara. Los valores con los que se han realizado los barridos son los siguientes: 2, 5, 10, 50, 100 y 200.

7.1.3.1. Comparativa del barrido completo

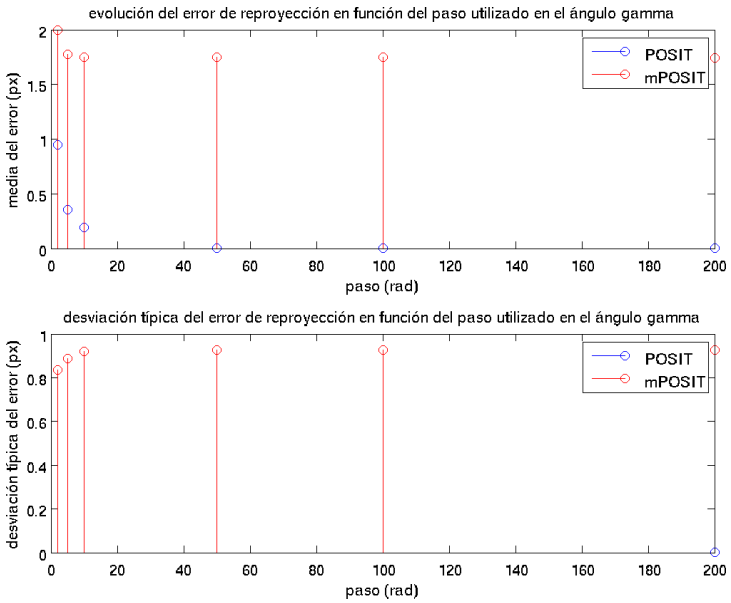
El resumen de resultados se muestra en las figuras 7.8 y 7.9.



(a) media y desviación típica para el ángulo α



(b) media y desviación típica para el ángulo β



(c) media y desviación típica para el ángulo γ

Figura 7.8: Comparativa del error de proyección según la translación en el eje Z

Como se puede observar en la figura 7.8, la tendencia sigue siendo igual en cuanto a los ángulos con los que el algoritmo funciona mejor o peor. Respecto al ángulo alpha, la media del error disminuye considerablemente aunque la desviación típica sea algo mayor. En cuanto al resto de ángulos, su comportamiento sigue siendo igual en los 2 debido al modelo geométrico utilizado, y además el error es superior al algoritmo original, tanto en media como en desviación típica.

En cuanto a la mejora de rendimiento del algoritmo, a la vista de las gráficas mostradas en la figura 7.9, se puede sacar como conclusión que la mejora en el número de iteraciones va reduciéndose a medida que se aleja el objeto. Además, se ve que la mejora, cuando los objetos son cercanos, está entre las 3 iteraciones por frame para el caso del ángulo alpha, y casi las 5 iteraciones para el resto de ángulos.

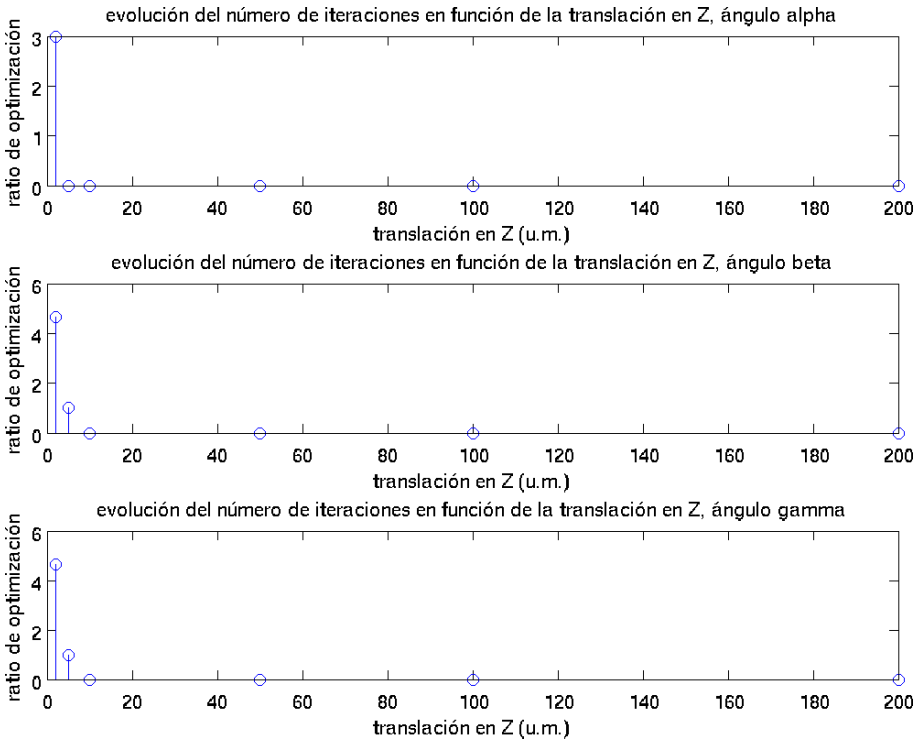


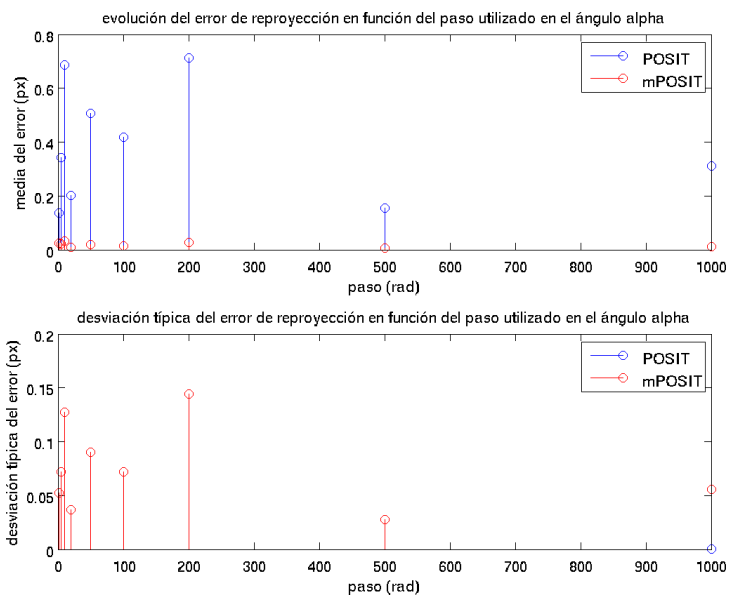
Figura 7.9: Comparación del ratio de mejora en el número de iteraciones según la translación aplicada

7.1.4. Resultados con variación de la distancia focal

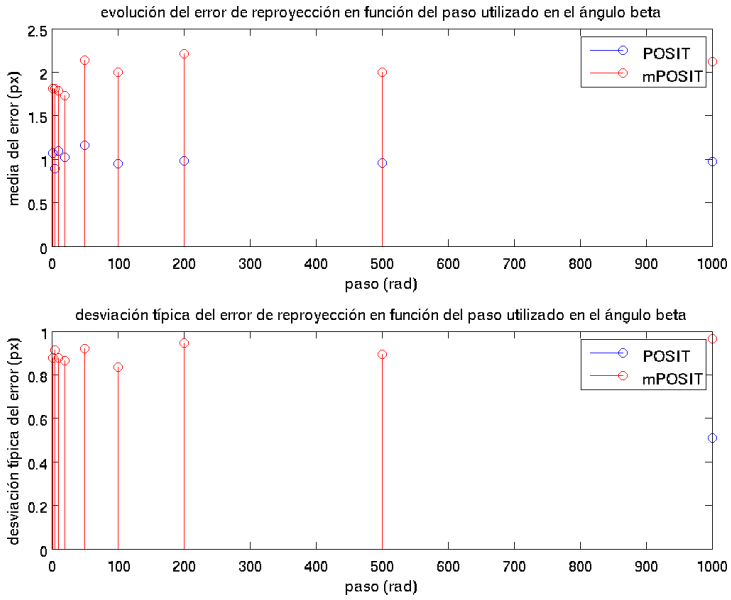
En este experimento se ha probado la influencia de la distancia focal en el algoritmo, puesto que es uno de los parámetros de entrada y es una de las variables con las que se realiza el escalado en la matriz de rotación y también la de traslación, además de utilizarse en la reconstrucción. Por tanto, se ha probado el algoritmo para distintas distancias focales: 2, 5, 10, 20, 50, 100, 200, 500 y 1000.

7.1.4.1. Comparativa del barrido completo

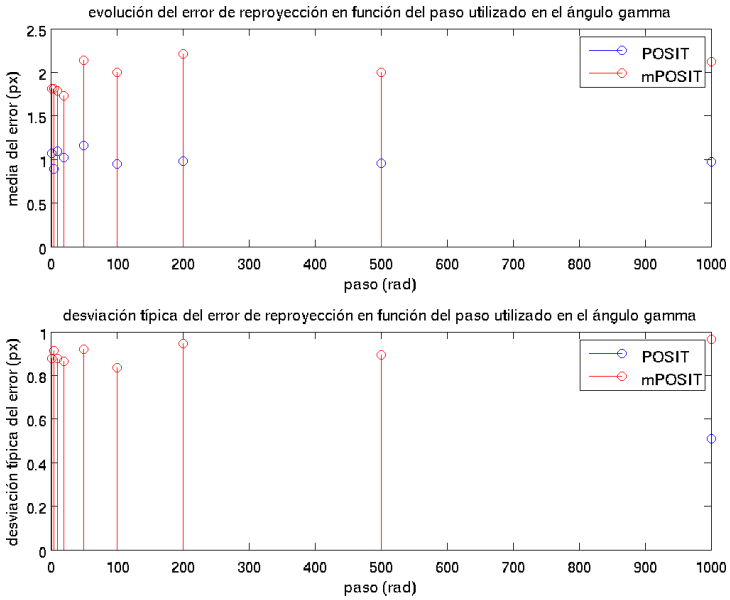
Al igual que pasaba con la translación en el eje z, las gráficas de error, tanto en media como en varianza siguen la misma tendencia, esto es, las gráficas de error para el ángulo alpha, mostradas en la figura 7.10, son algo mejores utilizando el algoritmo modificado que utilizando el algoritmo base, mientras que para el resto de ángulos se duplica aproximadamente el error cometido, por el algoritmo base.



(a) media y desviación típica para el ángulo α



(b) media y desviación típica para el ángulo β



(c) media y desviación típica para el ángulo γ

Figura 7.10: Comparativa del error de proyección según la distancia focal utilizada

En cuanto a la mejora en el número de iteraciones necesarias para ejecutar el algoritmo, según se muestra en la figura 7.11, vemos que para los 3 ángulos la mejora en el número de iteraciones está en torno a 2, y va creciendo de forma logaritmica a medida que crece la distancia focal, para llegar hasta unas 5 iteraciones de mejora por frame.

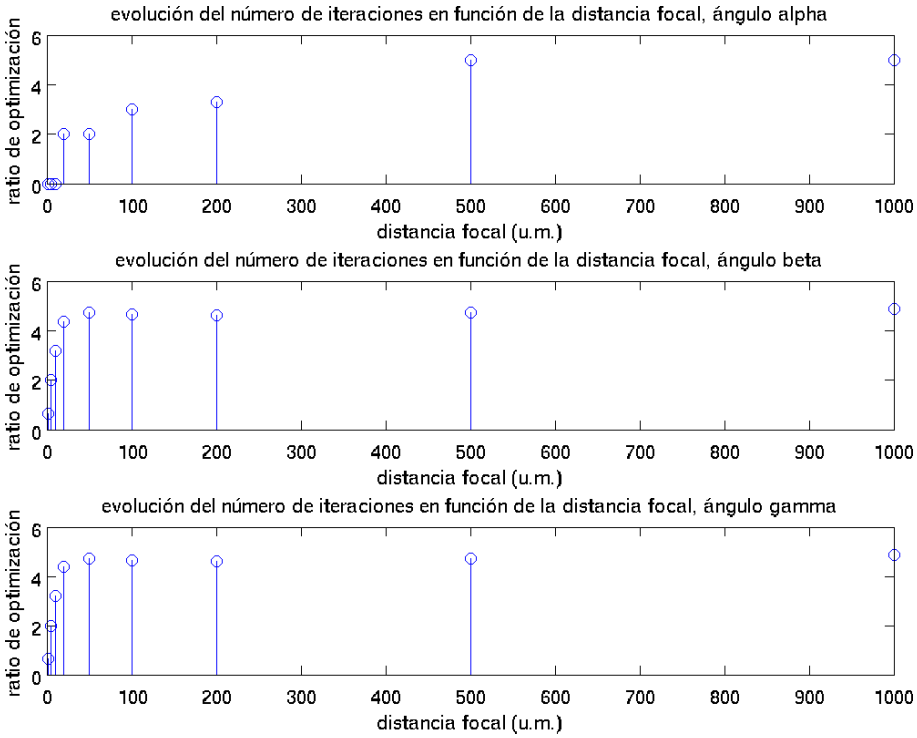


Figura 7.11: Comparación del ratio de mejora en el número de iteraciones según la distancia focal

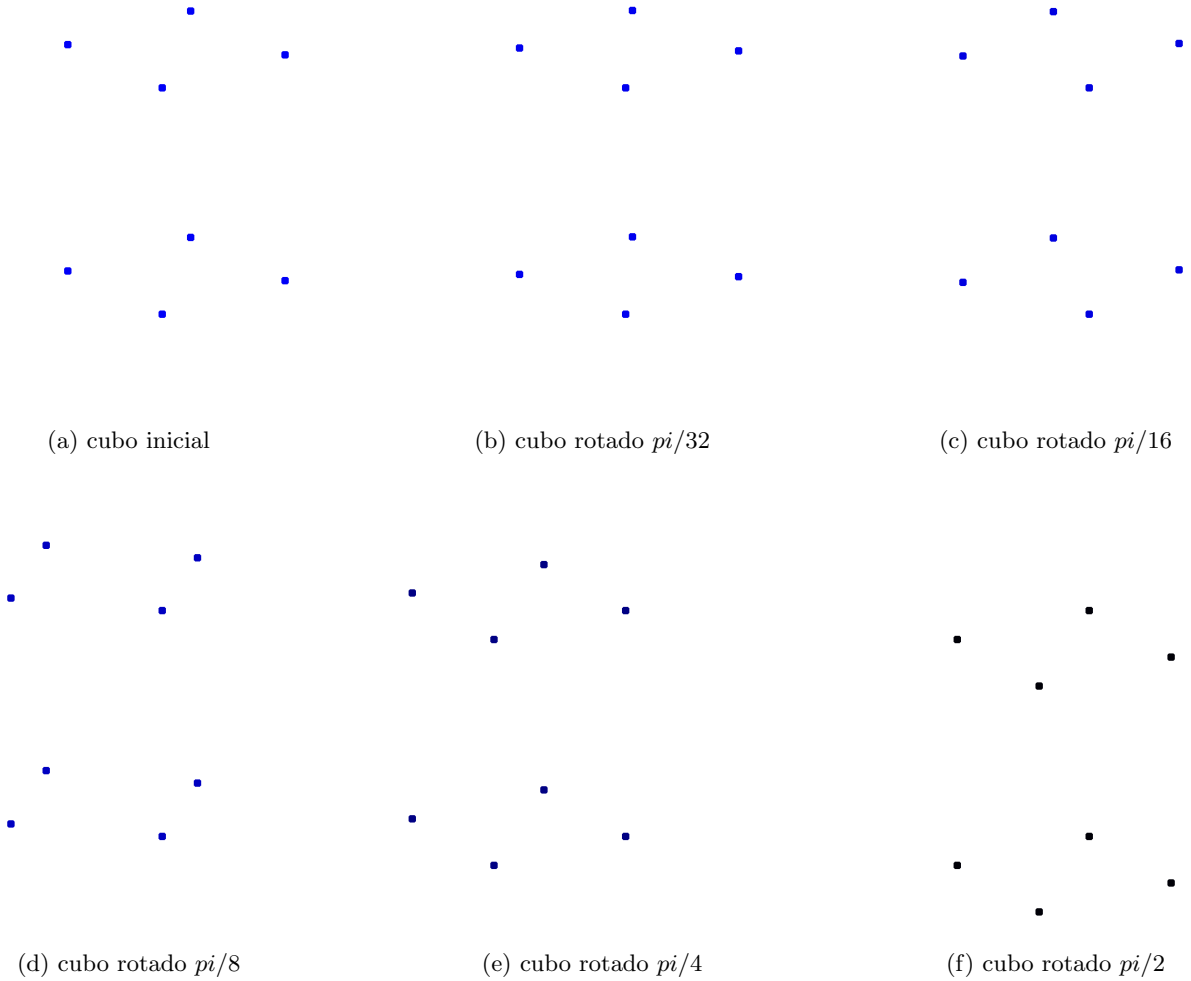
7.1.5. Resultados con variación del paso de rotación con deformación en la perspectiva utilizada

En este caso, existen varias modificaciones sobre el planteamiento base de las pruebas que se explican a continuación:

- Las coordenadas 3D del cubo se multiplican por 10
- La distancia focal se modifica hasta llegar a 1500 para que el algoritmo no se bloquee, puesto que el error es muy alto, la estimación no es buena y el algoritmo no converge.
- Para realizar la deformación de la perspectiva se utiliza la función plot de Matlab con los puntos del cubo. Una vez proyectada y guardada, la imagen se lee y se le aplica una erosión a fin de que los puntos se reduzcan únicamente a 1 pixel y puedan ser identificados fácilmente con la función máximo para que devuelva las coordenadas de estos puntos. A partir de aquí se ejecutaría el banco de pruebas normalmente.

Solo se mostrarán las pruebas realizadas con un paso de $\frac{\pi}{32}$, como el resto de pruebas, para tener la misma referencia que en experimentos anteriores de traslación y rotación.

Las rotaciones se han realizado en los distintos ángulos de rotación del espacio, coincidentes con los ejes del espacio de 3 dimensiones. En primer lugar, mostramos en las figuras 7.12 ejemplos visuales de qué supone un paso de $\pi/32$, $\pi/16$, $\pi/8$, $\pi/4$ y $\pi/2$ en el ángulo α. solidario con el eje vertical. En la figura 7.13 se presentan imágenes de ejemplo que se han rotado en los otros 2 ángulos del espacio restantes β y γ, para un paso de $\pi/16$.



(a) cubo inicial (b) cubo rotado $pi/32$ (c) cubo rotado $pi/16$

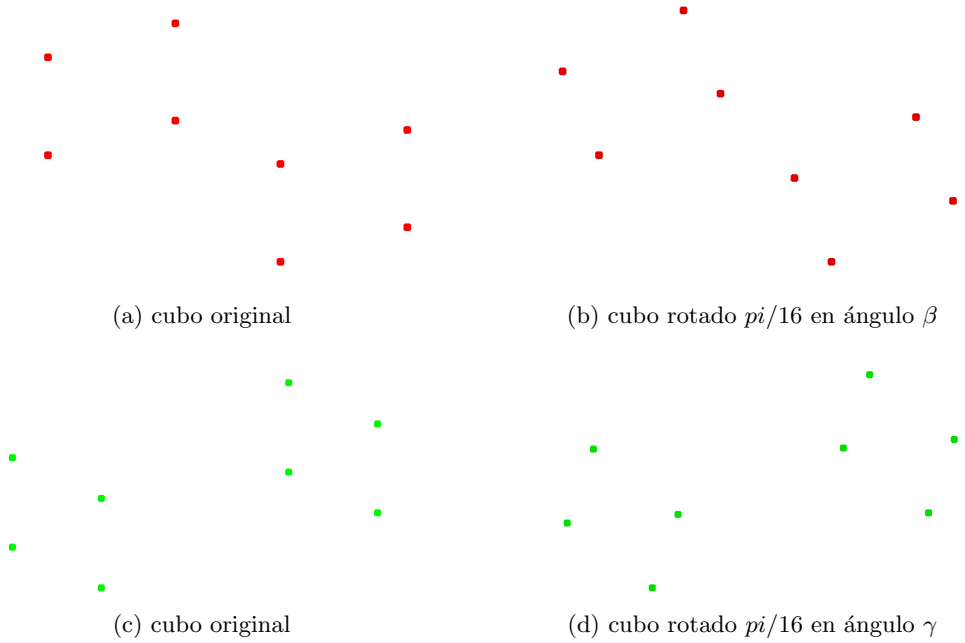
(d) cubo rotado $pi/8$ (e) cubo rotado $pi/4$ (f) cubo rotado $pi/2$

Figura 7.12: Distintos pasos de rotación aplicados en el ángulo α

Como se puede apreciar, se producen deformaciones en algunas aristas del cubo que no se introducen en las demás, con lo que ese factor afecta de manera fundamental a los resultados obtenidos como se verá posteriormente.

Ejemplos de cubos reproyectados junto con sus versiones originales se muestran en la figura 7.14. Como se puede apreciar, la predicción dada por POSIT y por mPOSIT son muy parecidas, y el cubo mantiene aproximadamente la pose, pero el error es bastante apreciable. Este comportamiento es explicable a la vista de la deformación incial que introduce la representación en Matlab no es la misma que la perspectiva de proyección utilizada por POSIT, por tanto, POSIT disminuye el error hasta llegar al mínimo local y por esto es capaz de mantener la pose, pero

no realiza el escalado en cada dimensión de manera independiente como se puede apreciar, sino que asume que el escalado en las 3 dimensiones es el mismo.



(a) cubo original

(b) cubo rotado $pi/16$ en ángulo β

(c) cubo original

(d) cubo rotado $pi/16$ en ángulo γ

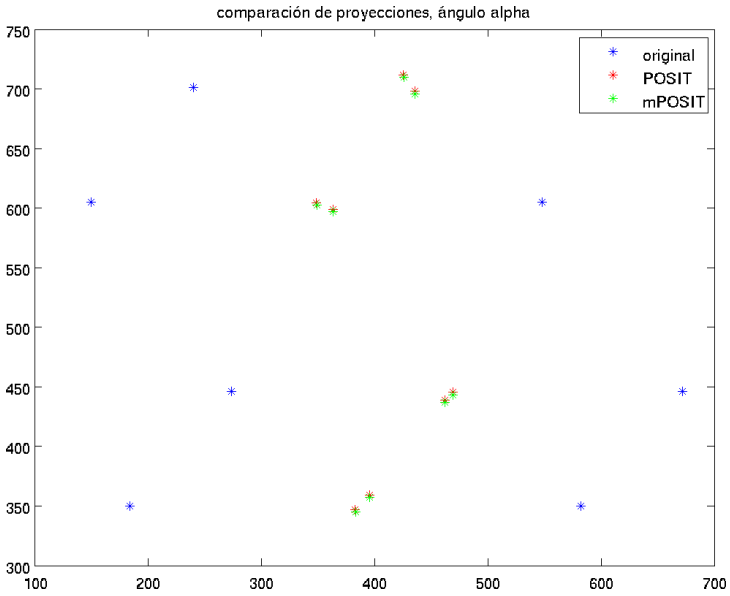
Figura 7.13: Paso de $\pi/16$ aplicado a los ángulos β y γ.

Una vez se han visto los ejemplos de reproyección, se muestran las gráficas de error devuelto por posit, que son aproximadamente iguales a las anteriores. El error representado en estas gráficas no tiene en cuenta el error que se obtiene si reproyectamos los puntos en la cámara, sino que es el error contado desde la anterior estimación, y esta no es la original, sino que va cambiando a lo largo de la evolución del algoritmo. Por tanto, vemos que se pueden obtener δ inferiores a 1, lo cual hace que el algoritmo converja, pero el error que tenemos reproyectando el punto es distinto y bastante más elevado. El error de reproyección se encuentra entorno a los 240 píxeles para una imagen de tamaño 1200x901 píxeles.

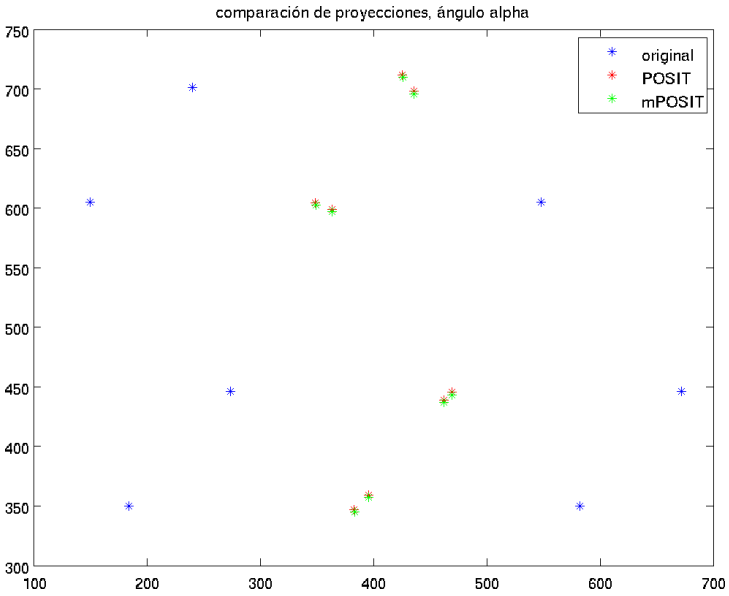
En cuanto a la reducción del número de iteraciones, vemos que el comportamiento general es aproximadamente el mismo, se aprecia que a pesar de las deformaciones y del error asociado, el número de iteraciones se sigue viendo reducido con el algoritmo modificado frente al normal.

7.1.6. Resultados con variación del número de puntos utilizando un modelo láser diezmado de cara

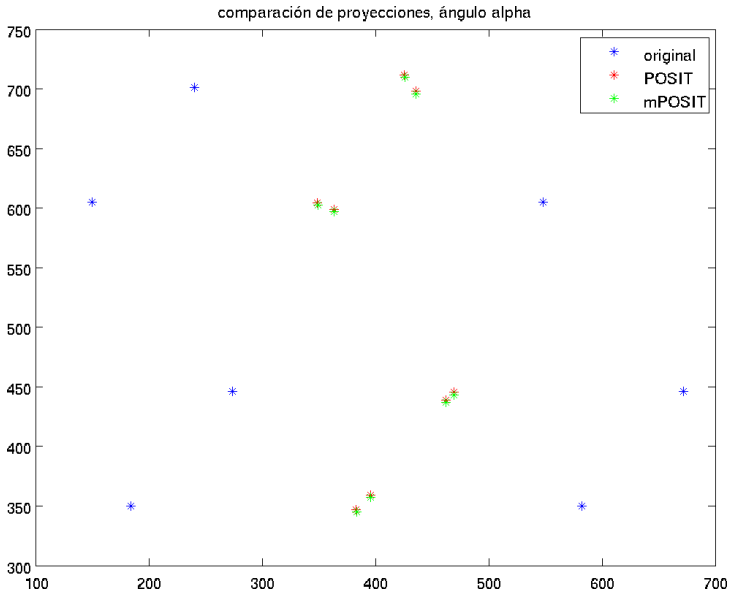
En este caso, el barrido se realiza con el número de puntos aplicados en el algoritmo. Para variar el número de puntos, se ha utilizado un modelo diezmado utilizando el algoritmo expuesto en esta Tesis de Máster, de la cara de un usuario, y se ha reducido el número de puntos con un diezmado normal, esto es se toman una de cada 'x' muestras.



(a) ángulo α



(b) ángulo β



(c) ángulo γ

Figura 7.14: Ejemplos de reproyección de puntos obtenidos cuando la perspectiva se ve modificada

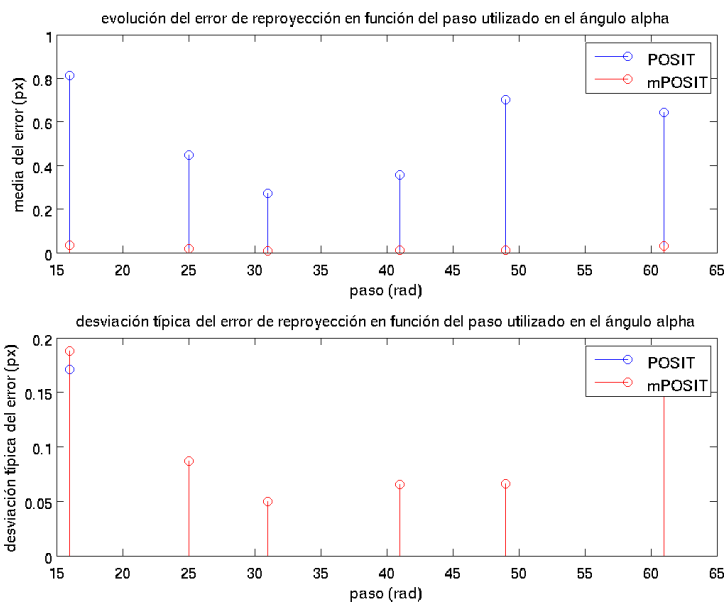
Para realizar un diezmado de la malla original (de 61 puntos), se han cogido unas relaciones de: 1, 1.25, 1.5, 2, 2.5 y 4. Esto es, se seleccionan para cada modelo, 1 de cada 1 muestras en el primer caso, en el segundo se cogerían 3 de cada 4, en el tercer caso, se tomarían 2 de cada 3, y así sucesivamente con una de cada 2, 2 de cada 5 y 1 de cada 4. Con lo cual quedan en realidad la siguiente cantidad de puntos en cada iteración: 61, 49, 41, 31, 25 y 16.

Los parámetros del test general también han tenido que ser modificados y sustituidos por $T = \begin{pmatrix} 10 & 10 & 400 \end{pmatrix}, f_l = 1500$ porque si no el algoritmo no terminaba de converger en un tiempo razonable o se estancaba. La traslación ha sido para no tenerlo muy cerca del centro de proyección, puesto que normalmente el objeto está a cierta distancia de la cámara. En ocasiones

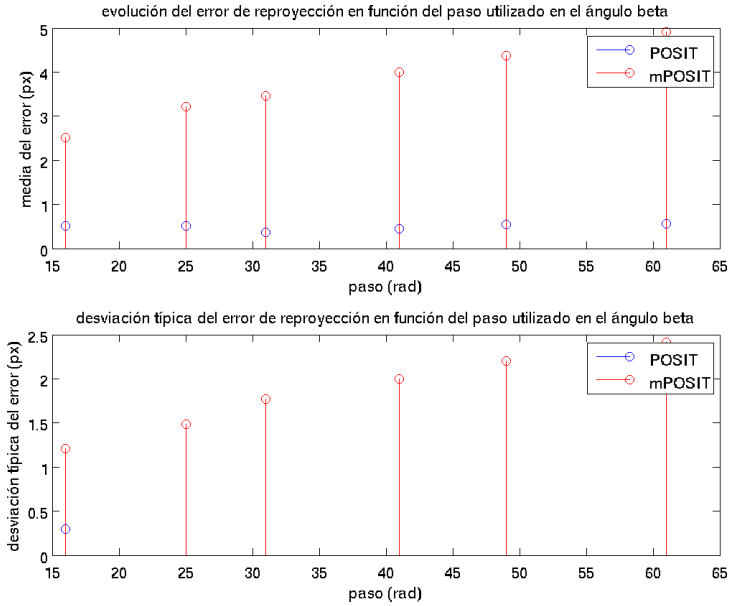
anteriores, la translación en el eje Z ha sido de 2 u.m. puesto que el modelo era bastante más pequeño

7.1.6.1. Comparativa del barrido completo

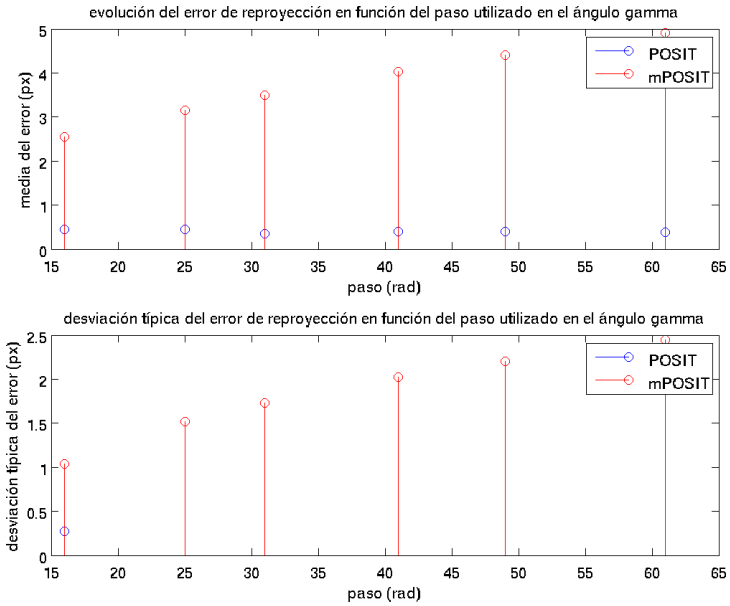
A continuación se muestran, como en los casos estudiados anteriormente, las gráficas de error de reproyección en la figura 7.15 y las gráficas de mejora del número de iteraciones por frame en la figura 7.17.



(a) media y desviación típica para el ángulo α



(b) media y desviación típica para el ángulo β



(c) media y desviación típica para el ángulo γ

Figura 7.15: Comparativa del error de proyección según el número de puntos utilizados

Vemos que las gráficas de error de reproyección del ángulo alpha siguen quedando mejores que el resto de algoritmos, y además el error no supera el píxel. En esta ocasión, las gráficas de error de reproyección para los ángulos beta y gamma son muy similares, pero si las miramos con detenimiento ya no son exactamente las mismas, sino que tienen pequeñas diferencias. Para ilustrar mejor estas diferencias en el perfil de la función de error de reproyección, se muestran las gráficas completas, para el caso de 25 puntos, el cual se muestra en la figura 7.16. Esta diferencia se aprecia sobre todo para un número de puntos bajo (16), en la cual, el ángulo beta tiene una desviación típica del error un poco superior al ángulo gamma.

En cuanto al valor del error vemos cómo la tendencia es al alza para los ángulos beta y gamma, mientras que para el ángulo alpha, la tendencia es a la baja en el valor de la media. Para el caso de los ángulos beta y gamma, tanto la media como la varianza del error superan los 5 píxeles cuando se utiliza el modelo completo con 61 puntos.

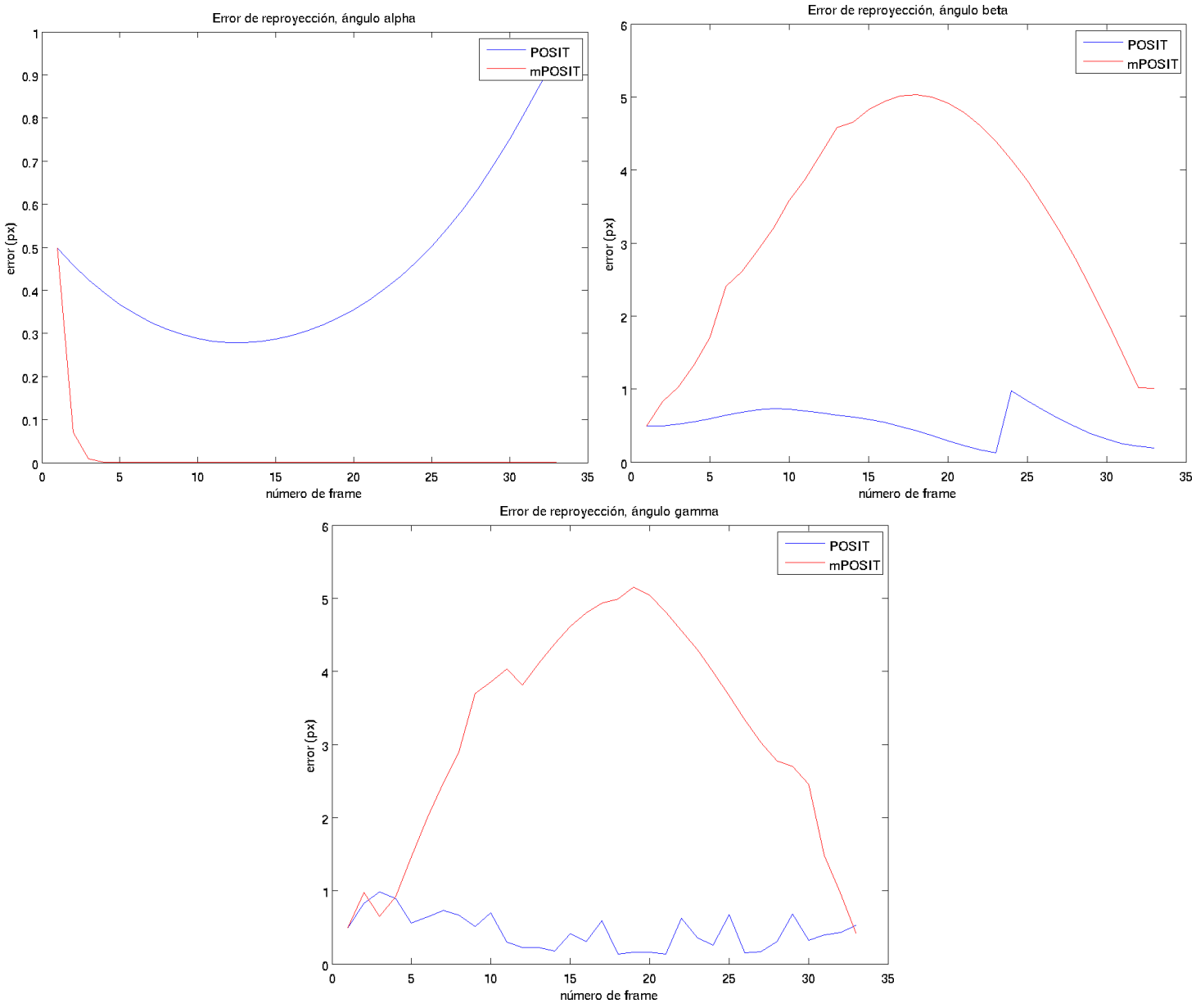


Figura 7.16: Comparativa del error de proyección según el número de puntos utilizados

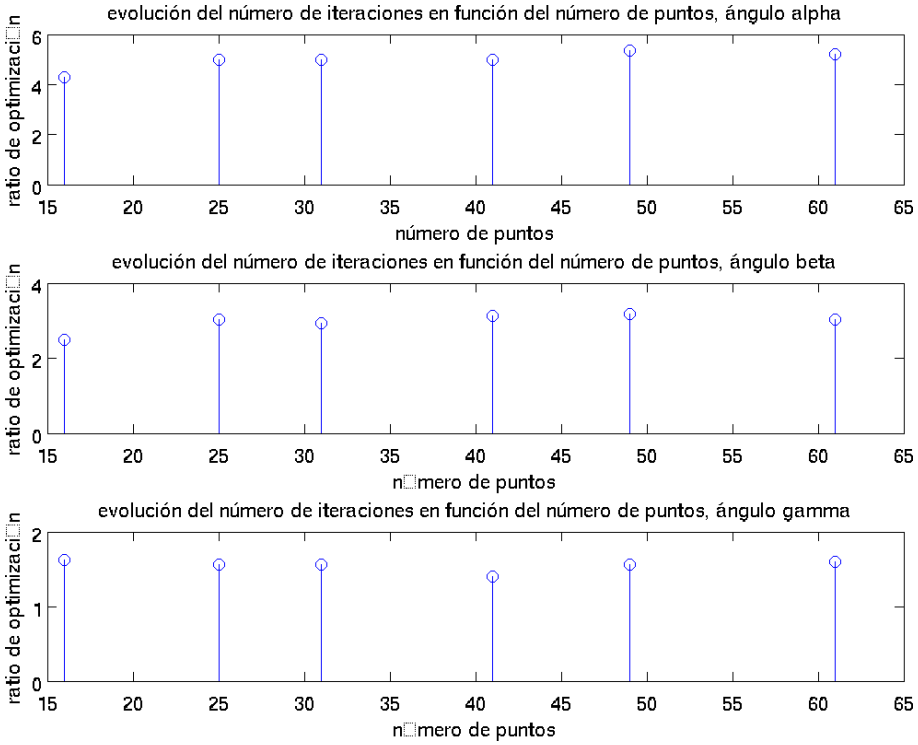


Figura 7.17: Comparación del ratio de mejora en el número de iteraciones según el número de puntos utilizados

En cuanto a la mejora del número de iteraciones es bastante superior en el ángulo alpha, después en el ángulo beta, y donde menor es la optimización es en el ángulo gamma. Estos resultados empiezan a ser más cercanos a la realidad, al ser la no tener la distribución de puntos 3D las propiedades de simetría que se encontraban en el caso del cubo.

7.1.7. Aproximación de implementación práctica de POSIT: cambio de sistema de coordenadas

En este apartado se muestran los resultados obtenidos en Matlab con una aproximación a la implementación previa a C++ de POSIT. Para evitar modificar el código fuente del algoritmo de OpenCV se ha propuesto utilizar un cambio de coordenadas en la malla 3D antes de pasar a POSIT los parámetros. La idea consiste en pasar como parámetro una malla a la que se aplica la rotación y traslación de la iteración anterior. Esta malla 3D se supone más cercana a la proyección anterior, por lo que, en principio se ahorrarían iteraciones en el algoritmo. Este cambio de referencia se realizaría de la manera que se representa en la figura 7.18.

En lugar de aplicar la malla $p3D$ sobre los puntos en la imagen $p2D$ y obtener unas matrices de cambio de coordenadas (R_1, T_1), se propone transformar la malla original $p3D$ utilizando la predicción anterior del algoritmo(R, T). Se aplica la matriz de transformación y se obtiene una malla temporal que sería válida en ese frame $p3D'$. Esta última malla es la que se pasaría como

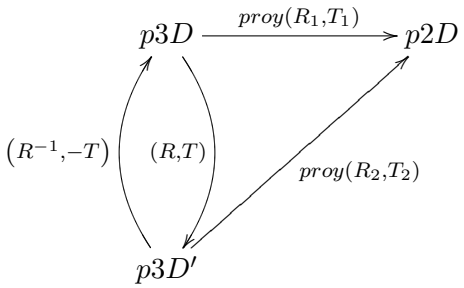


Figura 7.18: Esquema de aproximación previa a la implementación de POSIT

parámetro a POSIT y los puntos obtenidos desde la imagen $p2D$ y la distancia focal, en lugar de pasar las matrices (R, T). Para obtener R_1 y T_1 se deshace el cambio de referencia dado por (R, T).

La implementación del algoritmo en OpenCV es la clásica, y para hacer estas pruebas se ha utilizado la implementación clásica de Matlab, en lugar de la moderna, como se venía realizando hasta ahora en la mejora antes comentada.

Para ilustrar si la mejora puede aplicarse desde este punto de vista se pasa a mostrar únicamente la mejora en el número de iteraciones, en la figura 7.19, puesto que el error hemos visto que tampoco aumentaba en forma reseñable.

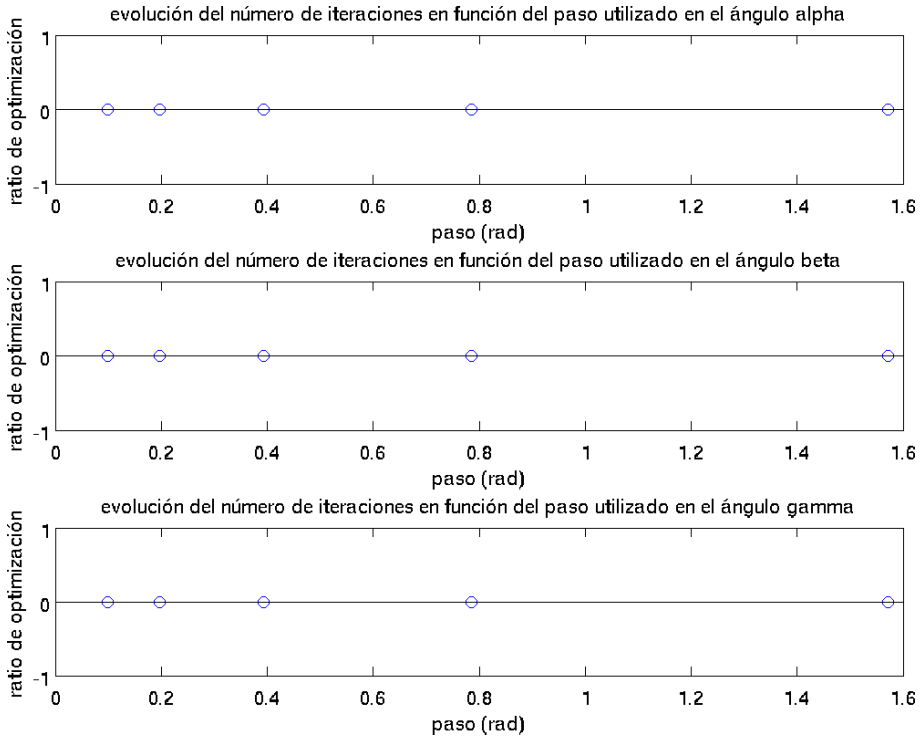


Figura 7.19: Comparación del ratio de mejora en el número de iteraciones según el número de puntos utilizados

Como se puede apreciar, la mejora es nula, al ser el número de iteraciones necesarias exactamente la misma para los 2 algoritmos, por lo que se descarta la aproximación.

7.1.8. Conclusiones parciales de simulación

Con los resultados anteriores, podemos sacar las siguientes conclusiones principales:

El error en la estimación que se obtiene con el algoritmo modificado es, dependiendo del ángulo, algo superior (para ángulos de rotación beta o gamma) o inferior (para ángulo alpha). De todos modos, el error de reproyección jamás supera los 3 píxeles, contando la media y la desviación típica.

También se ha comprobado experimentalmente que el error devuelto por el algoritmo no sirve en todos los casos como una estimación correcta del error de reproyección, aunque para algunos casos, este está correlado, puesto que la forma de las gráficas son parecidas y tienen valores similares.

En cuanto al número de iteraciones, se puede concluir que para la mayoría de los casos estudiados, se experimenta una mejora superior a un 50 % (3/5 frames ha sido lo habitual en los experimentos) en la reducción de las mismas. Aunque existan frames en los que el número de iteraciones del algoritmo es superior al del original, la mejora experimentada en el resto compensa el incremento experimentado en unos frames particulares.

Se ha podido comprobar que existe una fuerte dependencia de los resultados del algoritmo, tanto del modificado como del original respecto del ángulo con el que se trabaje. Si para el ángulo alpha, los resultados siempre han sido mejores, en el resto de ángulos de rotación han sido peores, aunque no se ha experimentado una reducción drástica de la efectividad del algoritmo.

En cuanto a la distancia respecto del sistema de referencia de la cámara, y la distancia focal de la misma, los resultados obtenidos han sido muy similares, tanto en error como en tiempo de ejecución.

Como se ha podido comprobar, no se ha podido independizar la función para evitar modificar el código fuente implementado en OpenCV, por lo tanto se ha de implementar la modificación en lenguaje C/C++. En el siguiente apartado se mostrarán los resultados de la implementación de las modificaciones en el código fuente.

Se puede apreciar cómo el error incrementa en ciertos ángulos con el número de puntos utilizado, aunque parezca lo contrario, tanto en media como en desviación típica. Esto es debido a que puede que existan varios puntos que no se ajusten bien al modelo que estima POSIT, y al realizar una estimación de error similar a la que se realiza con mínimos cuadrados calculando la matriz pseudo-inversa, no se eliminan puntos sino que se ajusta lo mejor que puede al conjunto total, por lo que el modelo falla más que si eliminásemos esos puntos que se salen del modelo con algoritmos tipo RANSAC.

7.2. Implementación práctica de la optimización de POSIT

Una vez implementadas estas modificaciones en lenguaje C/C++ sobre el código original de OpenCV, se pasa a probar su rendimiento sobre una secuencia de vídeo. Hay que tener en cuenta que esto se realiza junto con el algoritmo RANSAC, por tanto el número de veces que se ha de repetir este proceso es elevada, en cada frame incluso. Cada vez que se cambie el subconjunto de puntos 3D evaluados, se ha de generar de nuevo un objeto POSIT que inicializará la matriz

pseudo-inversa del conjunto de puntos 3D, necesaria para la inicialización del algoritmo.

Se presentarán tiempos de ejecución, tanto de la inicialización, como de ejecución de la versión de OpenCV y con la versión modificada.

7.2.1. Tiempos de cómputo

A continuación se muestran los tiempos de cómputo obtenidos para una prueba sencilla realizada, en la que se ejecuta POSIT y después mPOSIT con los mismos datos de entrada. Posteriormente se muestran los resultados obtenidos sobre una secuencia de prueba normal.

En la figura 7.20 se muestran los tiempos de ejecución en microsegundos, tanto de la inicialización como de la ejecución del algoritmo POSIT y mPOSIT. Como se puede comprobar, para el tiempo medio de inicialización del algoritmo se requieren 2,75 us, mientras que para la ejecución de POSIT se requieren 17,85 us y para mPOSIT 4 us, con lo que la mejora es considerable, aunque también hay que tener en cuenta que las entradas pasadas para los dos algoritmos es siempre la misma, por tanto, el tiempo de ejecución es pequeño en estos casos. Estos tiempos varían en función del procesador utilizado. En este caso se ha realizado con un Intel Centrino duo a 2.4 GHz 4 MB de caché por núcleo y bus frontal de 800 MHz.

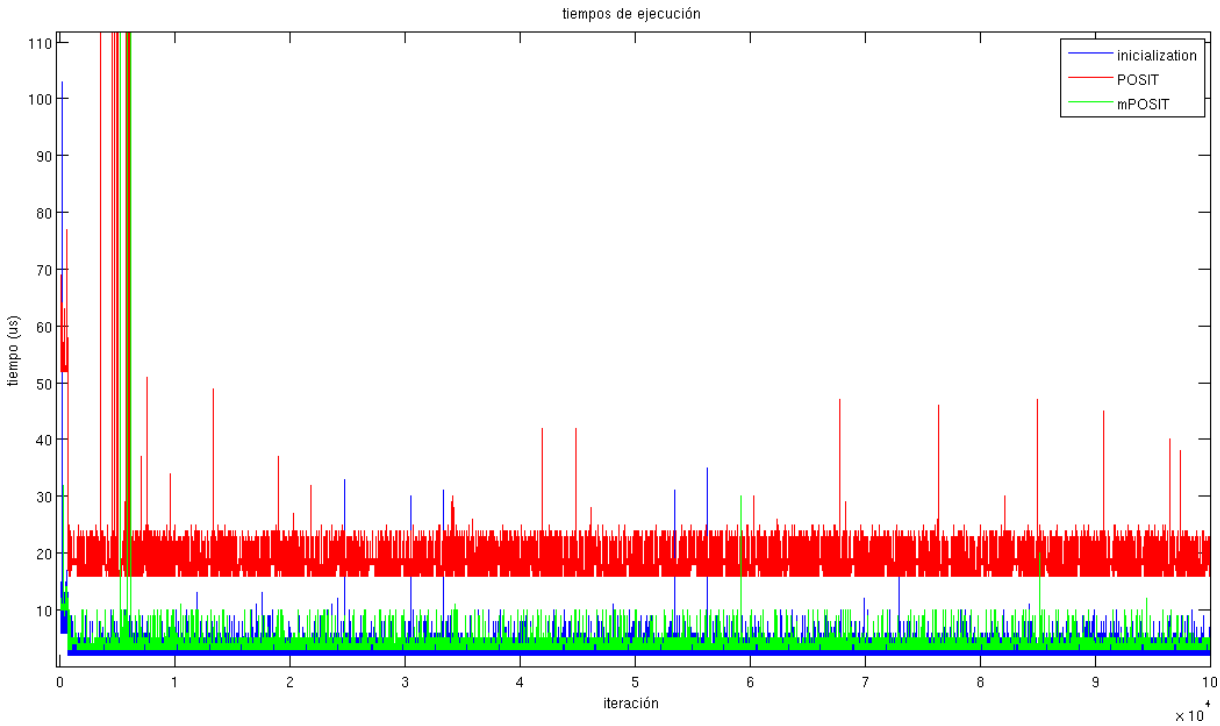


Figura 7.20: Tiempo de ejecución del algoritmo en las primeras pruebas de implementación

En cambio, en el software de seguimiento de caras utilizado, se ha implementado esta mejora con unos tiempos aproximadamente iguales a los dados por la implementación de OpenCV. Esto es debido al procedimiento utilizado para el cálculo de la matriz de transformación. En este software, se utiliza un RANSAC el cual se llama un número constante de iteraciones. En cada una de ellas se toma un subconjunto distinto de puntos, con lo cual el objeto POSIT se ha de estar

creando y destruyendo constantemente. Además, la predicción del frame anterior, es efectiva solo con las iteraciones en las que el conjunto utilizado está muy cerca de la predicción anterior. En el resto de frames, la predicción no es lo suficientemente cercana y el algoritmo puede llegar a tardar más en algunos casos, que para situaciones concretas, la modificación puede alterar demasiado la estimación inicial, y tardar más el algoritmo en converger. Las predicciones correctas suponen un porcentaje razonable de los casos, por lo que, si el conductor se está quieto y no hay demasiado error en la predicción, el tiempo de ejecución se ve reducido, pero en cuanto se introduzca ruido en la misma o haya variaciones significativas entre frames, el algoritmo tardará lo mismo o más que en la implementación de las librerías.

7.2.2. Conclusiones parciales de implementación

Para el caso sencillo vemos que la reducción del tiempo de cómputo está entorno al 77 %, puesto que tenemos un tiempo de ejecución de aproximadamente un cuarto del tiempo de cómputo del algoritmo original. También vemos que estos tiempos de ejecución son mas altos al principio que en régimen permanente, puesto que en estos casos ya actúan las optimizaciones del procesador y la caché.

En cuanto estas condiciones se dejan de cumplir, el algoritmo es solo eficiente si la variación entre frames es muy pequeña, y no en todas las iteraciones que se realizan en RANSAC, por tanto la mejora que supone es solo visible cuando se llega al régimen permanente, en el cual, los movimientos que se realizan por parte del conductor son mínimos.

7.3. Modelo 3D diezmado obtenido desde el scan-láser

En primer lugar se muestran ejemplos de modelos obtenidos desde el scan-láser por el método propuesto en esta tesis. Después se mostrarán los resultados de su inclusión en el software de seguimiento de la cabeza comparados con los modelos manuales.

7.3.1. Ejemplos de modelos obtenidos

A continuación se muestran resultados de modelos 3D automáticos que se han obtenido con el método propuesto. En primer lugar mostramos en la tabla 7.1 un resumen en la que se muestra el número de puntos en varias de las etapas del algoritmo completo para las distintas vistas, y estadísticas como el ratio de diezmado conseguido al final del proceso, teniendo en cuenta los puntos de todas las vistas aunque estos se repitan, no las mallas 3D fusionadas. Después, en las figuras 7.21, 7.22, 7.23, 7.24 y 7.25se muestran varias filas de imágenes, en las que se muestra de forma esquematizada el proceso particularizado para algunos usuarios.

Vemos en la tabla 7.1 que el porcentaje de diezmado que se consigue depende estrictamente del número de puntos obtenidos con Harris, de fácil seguimiento para poder utilizarlos luego en el software final. También influye algo en el número de puntos de salida si la malla se ha obtenido bien, puesto que si no se cogen puntos 3D asociados a puntos 2D en zonas clave no se obtendrán muchos de los puntos que marque Harris. El tipo de escaneo utilizado también tiene su influencia pero no tanta. El tipo de escaneo, repercute en la resolución del mismo, yendo desde un escaneo rápido en el cual se obtienen unos 4000 y 10000 puntos, mientras que en los escaneos de precisión se obtienen entre 40000 y 60000 puntos, llegando en algunos casos, si la superficie refleja bien la luz, a los 100000 puntos. La influencia de esto último no es tanta, puesto que el único efecto que tiene es que hay más puntos a elegir dentro de la misma zona. Si muchos puntos marcados

con Harris han caido en zonas de pelo, al no reflejar el láser, no existe correspondencia 3D y por tanto puede haber muchos puntos en Harris, pero muy pocos significativos fuera del pelo, lo cual supone una reducción de puntos significativa.

Nº usuario	Central: Nº puntos totales 3D/ Harris 2D/ salida 3D	Izquierda:Nº puntos totales 3D/ Harris 2D/ salida 3D	Derecha: Nº puntos totales 3D/ Harris 2D/ salida 3D	Nº puntos 3D agregado	Ratio de diezmado $\frac{p3d_1+p2d_2+p3d_3}{p3d_{agr}}$
1	54466 / 100 / 94	63027 / 103 / 90	79447 / 56 / 49	232	848,8791
2	8062 / 48 / 32	8592 / 35 / 22	8765 / 40 / 18	72	353,0416
3	9016 / 32 / 26	8548 / 29 / 16	8310 / 21 / 15	56	462,0357
4	55497 / 85 / 66	54339 / 32 / 24	58796 / 63 / 52	141	1195,9716
5	1905 / 114 / 7	2259 / 104 / 28	2664 / 110 / 24	59	115,7208
6	8415 / 44 / 34	12194 / 20 / 9	12478 / 53 / 44	86	384,7325
7	46664 / 57 / 47	68150 / 87 / 85	54788 / 82 / 74	205	827,3317
8	14308 / 139 / 59	15861 / 129 / 43	14584 / 119 / 45	146	306,5276
9	68577 / 64 / 57	62766 / 46 / 35	71207 / 39 / 32	123	1646,7479
10	14462 / 73 / 46	14223 / 51 / 18	12195 / 34 / 18	81	504,6913
11	113854 / 144 / 111	109552 / 144 / 118	93516 / 82 / 60	288	1100,4236

Tabla 7.1: Tabla resumen del número de puntos para distintos usuarios

Una vez mostrada la tabla, se exponen algunos casos concretos. La organización de las figuras es la siguiente: en primer lugar, se muestran las imágenes obtenidas desde la cámara fotográfica del escáner 3D. Después se muestran las esquinas obtenidas en las imágenes de las distintas vistas con el detector de Harris, con una constante para el detector de 0,004, un nivel de calidad de 0,001 y la distancia mínima entre esquinas de 15 píxeles. A continuación se muestran los puntos 3D asociados a cada vista correspondientes a cada esquina y por último el modelo 3D completo de la cara correspondiente, una vez obtenidas las matrices de transformación entre vistas.

En el caso de la figura 7.22, con el método estándar fallan algunos puntos en la cara. En consecuencia se trata la imagen para aumentar el contraste, ecualizandola en escala de grises, para obtener más puntos en los laterales. Esto último es necesario puesto que sin puntos en las orejas, el sistema se pierde en los giros que van desde el frontal hacia los laterales por falta de puntos visibles con los que calcular la pose.

Con la figura 7.25, se pretende demostrar que no solo este método es válido para obtener el modelo 3D de una cabeza, sino que puede valer en general para cualquier objeto, en este caso un robot. Además, como se ha visto en el ejemplo 11 de la tabla 7.1, al tener una superficie que refleja muy bien la luz, el número de puntos 3D obtenidos en el modelo aumenta considerablemente, aún después de hacer una limpieza del mismo (eliminando la mesa donde estaba apollado, los laterales, etc).

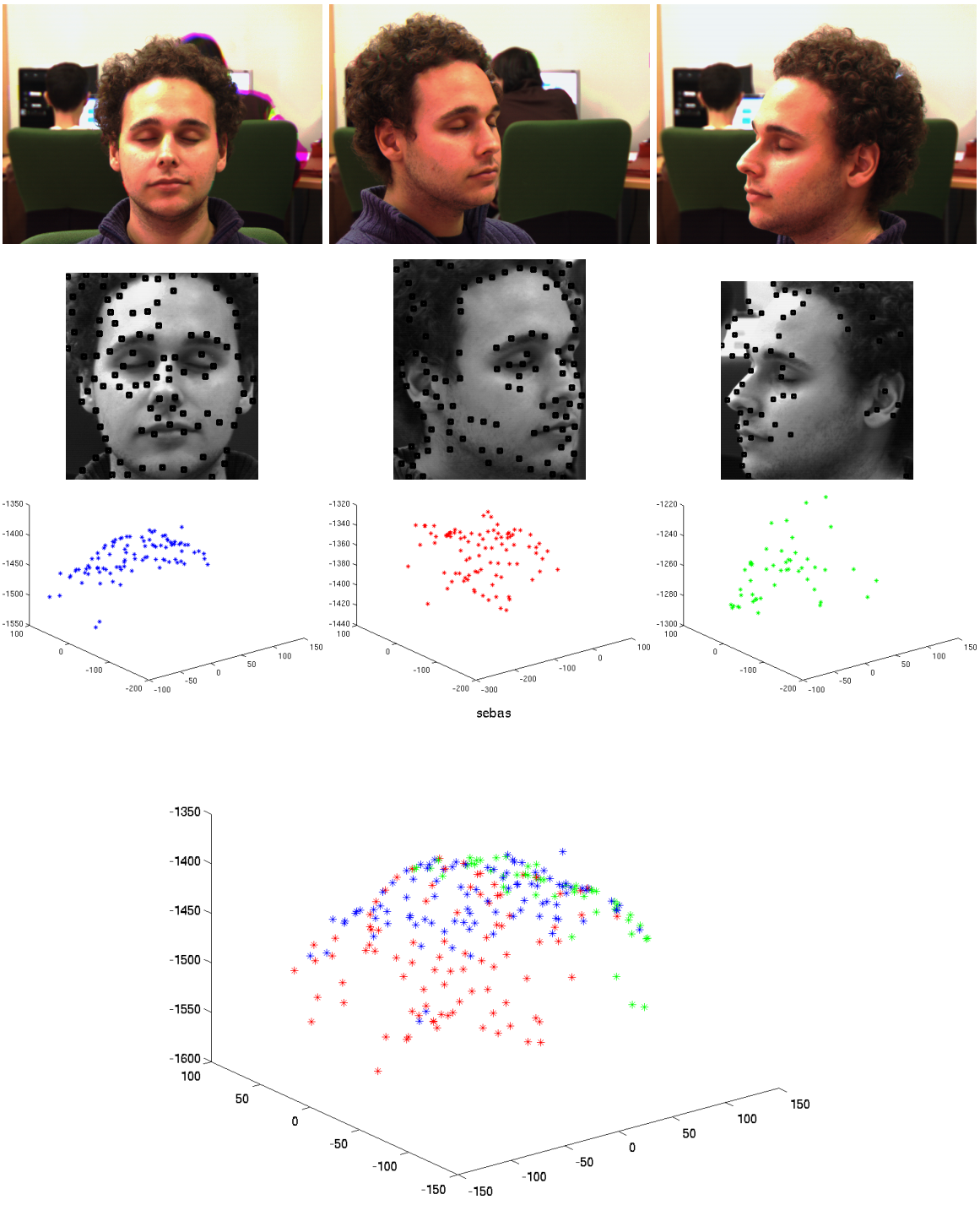


Figura 7.21: Ejemplo del proceso de extracción del usuario 1

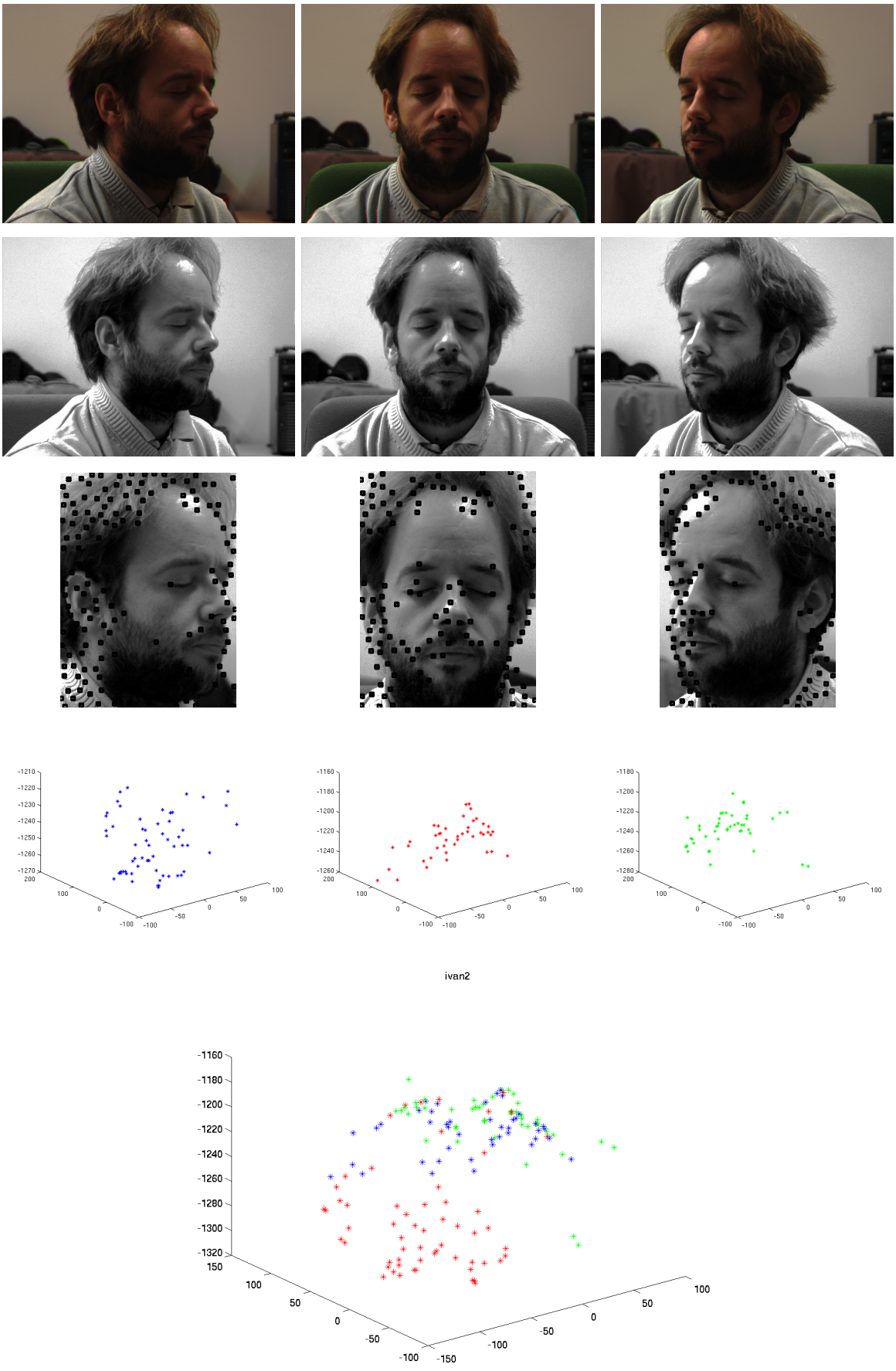


Figura 7.22: Ejemplo del proceso de extracción del usuario 8

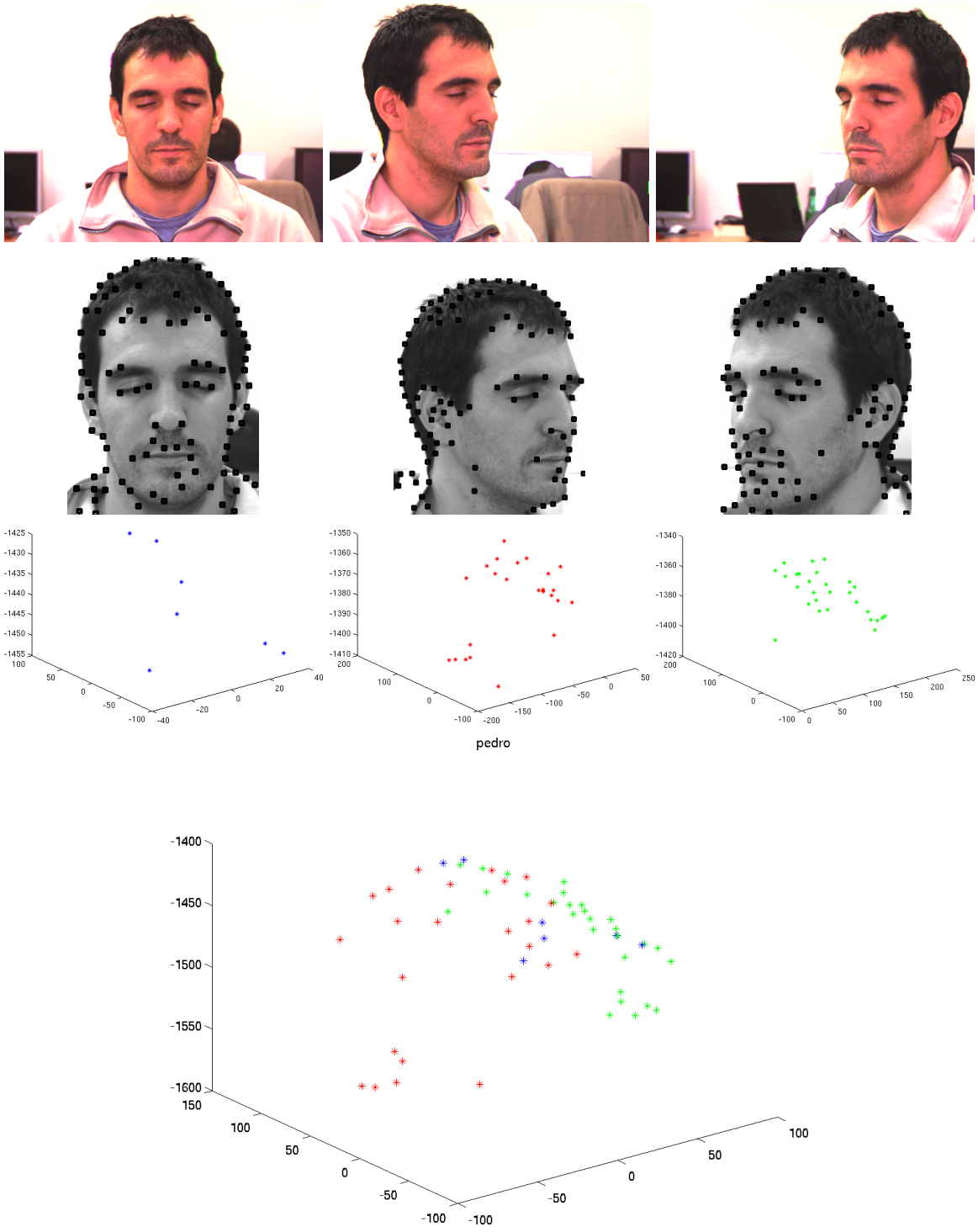


Figura 7.23: Ejemplo del proceso de extracción del usuario 5

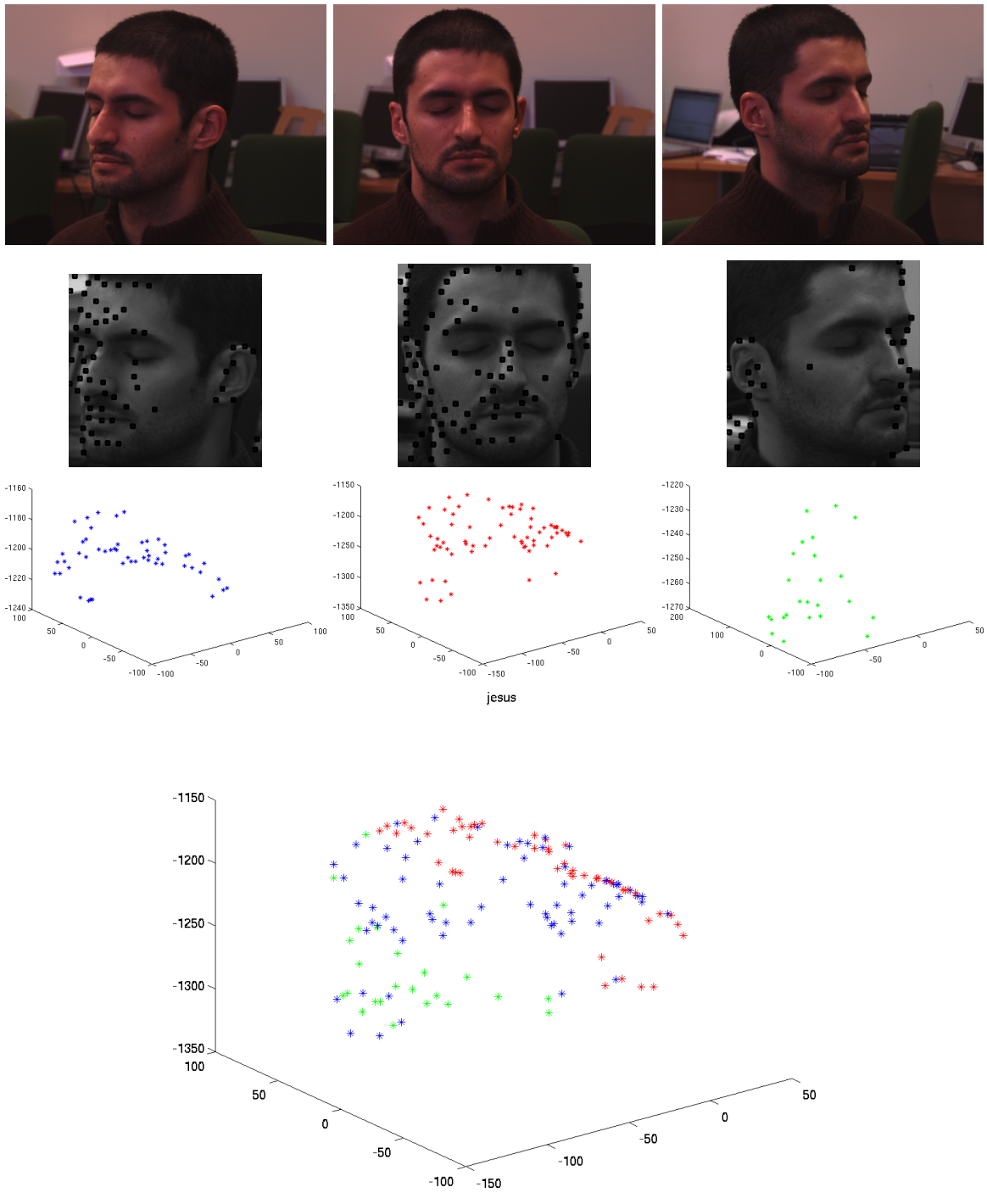


Figura 7.24: Ejemplo del proceso de extracción del usuario 4

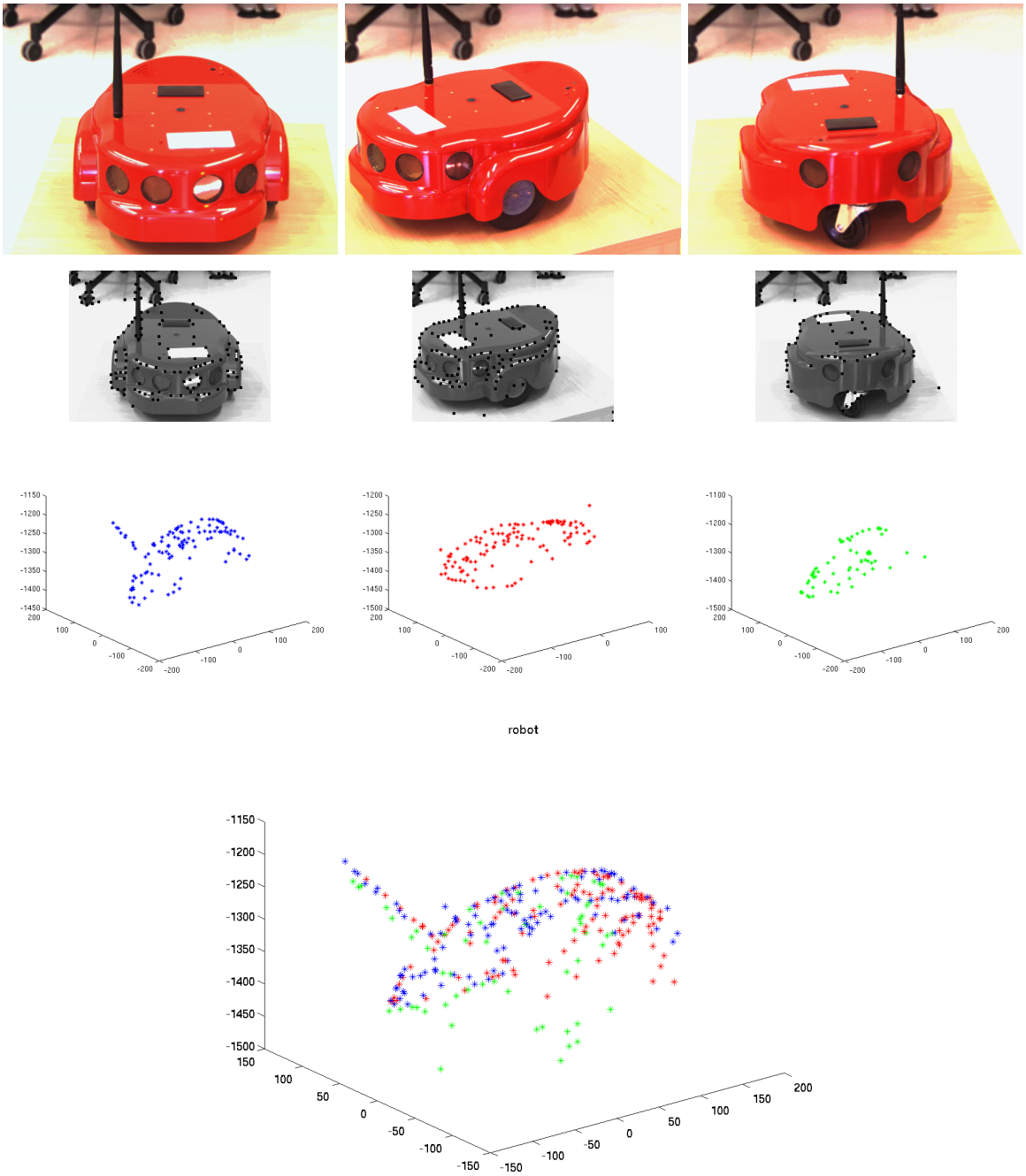


Figura 7.25: Ejemplo del proceso de extracción para otras aplicaciones

7.3.2. Resultados sobre secuencias de prueba

A continuación se exponen los resultados obtenidos utilizando versiones anteriores del programa con el modelo manual. El ground truth está extraído también con el programa anterior. Además se superpondrán los resultados obtenidos con la nueva versión que incluye las mejoras propuestas. Se han obtenido secuencias de ground truth utilizando un casco con un damero y cambiando la función de estimación de pose para hacerlo de manera más precisa que con POSIT, puesto que estamos en el caso de planos. Si el video posee una longitud superior al ground truth, en esta memoria solo se mostrará el tramo correspondiente del que se tiene información de referencia. Hay 2 usuarios en total sobre los que se han realizado las pruebas aunque existen más vídeos procesados con este método para más usuarios. El número de puntos ha sido de 59 y 128 puntos (máximo hasta ahora soportado por el programa). Las diferencias de resultados entre secuencias de prueba son debidas a una mala estimación inicial de parámetros de pose y traslación. También pueden existir diferencias al un valor diferente como punto de referencia para las rotaciones. El resto de la gráfica queda con una forma aproximadamente igual que la que estaba.

7.3.2.1. Secuencia nº 1

En este caso se evalúa una secuencia con una longitud total de 2088 frames de la cual se evalúan 543 frames de ground truth. El modelo utilizado para este usuario ha sido de 141 puntos, quitando los puntos que pertenecen al casco utilizado para la calibración, puesto que no corresponden con el modelo 3D de la cabeza que ha sido adquirido, con lo cual en total se han evaluado 128 puntos, que es en este caso lo máximo que permite el programa.

Un resumen de los errores obtenidos para esta secuencia se muestran en la tabla 7.2. En ella se puede observar que el error medio a lo largo de la secuencia evaluada es superior en la secuencia manual que en la de una densidad de puntos mayor. Sin embargo, si nos fijamos en las gráficas mostradas en la figura 7.26, sobre todo para los 2 últimos ángulos (yaw y roll) la estimación se ajusta más al ground truth, salvo en unos pocos tramos, que es donde la diferencia es mayor que con el método manual, y es donde penaliza el rendimiento del algoritmo. Se puede apreciar en este caso, que tanto el dato del error medio en el pitch como la gráfica confirman que funciona peor. Esto último es debido a que hay una mala recuperación en un punto concreto de la secuencia provoque el este error se vaya acumulando a lo largo de la misma, aunque se aprecia como la forma general se sigue aproximadamente, y ya en la parte final se aprecia cómo se vuelve a enganchar más al ground truth. Vemos también cómo al principio, se ajusta mejor en todos los casos al ground truth que el algoritmo manual. De todos modos, se ve cómo el error medio cometido por frame no llega a los 0.25 º en cualquier ángulo.

Método de obtención de la malla	Número de puntos en el modelo	número de frames evaluados	error medio pitch (rad)	error medio yaw (rad)	error medio roll (rad)	error medio total (rad)
manual	30	543	$7{,}4365 \cdot 10^{-4}$	0.0036	$3{,}5194 \cdot 10^{-4}$	0.0029
automático	128	773	0.0046	0.0044	$5{,}7266 \cdot 10^{-4}$	0.0042

Tabla 7.2: Tabla resumen de error para secuencia número 1

En la ejecución del algoritmo, se ha comprobado una mayor estabilidad, la recuperación es

más temprana en caso de pérdida, y esto es consecuencia de que existen más puntos donde apoyarse para poder calcular la pose. Estas pérdidas temporales de pose se dan sobre todo cuando se evalúan giros hacia los laterales.

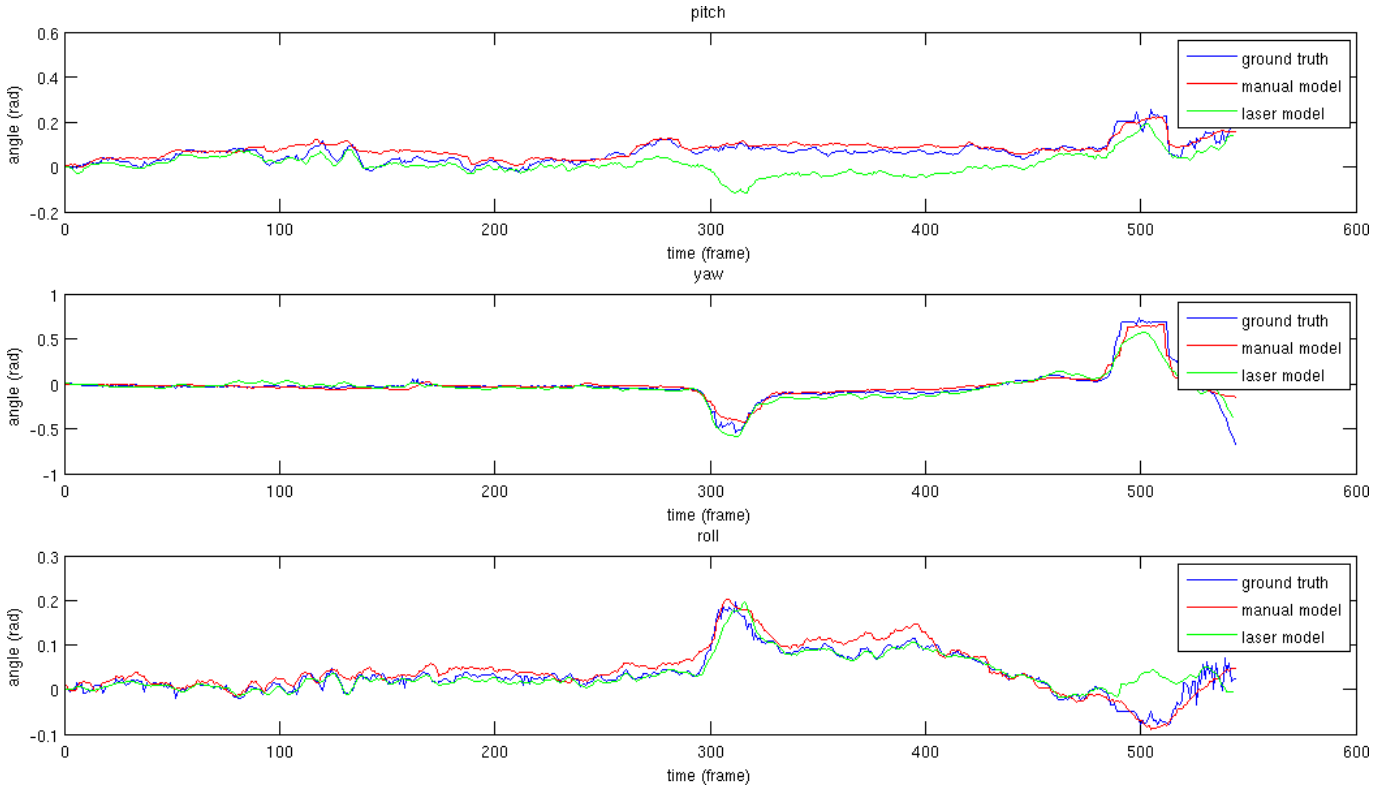


Figura 7.26: Ángulos obtenidos para la secuencia número 1

7.3.2.2. Secuencia nº 2

La secuencia evaluada en este caso tiene una longitud total de 2727 frames, de la cual se dispone de un ground truth de 746 frames. El usuario que se estudia en esta secuencia tiene un modelo de 50 puntos, puesto que el original tenía 59, pero se tuvo que eliminar los puntos 3D que no tenían información correcta al estar situados en el área del casco.

Los errores obtenidos para esta secuencia se muestran en la tabla 7.3 y las gráficas que muestran el resultado del algoritmo a lo largo de la secuencia en la figura 7.27. Se puede apreciar cómo el error en el ángulo pitch se ha visto reducido drásticamente respecto de la estimación manual. Este hecho se puede observar también en la gráfica correspondiente a este ángulo. El error en el ángulo roll es muy similar en los 2 casos aunque en la gráfica se aprecian 2 picos que no aparecen en el ground truth. Sin embargo el error de yaw es muy dispar y mejor en el caso manual aunque en para este caso llega a los 0.3 º y para el modelo laser a los 0.83 º. Esto es debido a que existe un error en la componente continua, no despreciable cuando se comete el último giro y además, en el caso de las rotaciones no se llega a girar del todo el modelo y no llega al ángulo que se produce realmente. Esto es debido a la falta de configuración de algunos de los parámetros del modelo, al ser tremendamente costoso ir ajustando los parámetros para cada punto, incluso en un modelo manual, en el que hay bastantes menos puntos.

Método de obtención de la malla	Número de puntos en el modelo	número de frames evaluados	error medio pitch (rad)	error medio yaw (rad)	error medio roll (rad)	error medio total (rad)
manual	23	746	0.0019	0.0050	$8{,}5795 \cdot 10^{-4}$	0.0032
automático	50	1351	$4{,}1298 \cdot 10^{-4}$	0.0141	0.0012	0.0046

Tabla 7.3: Tabla resumen de error para secuencia número 2

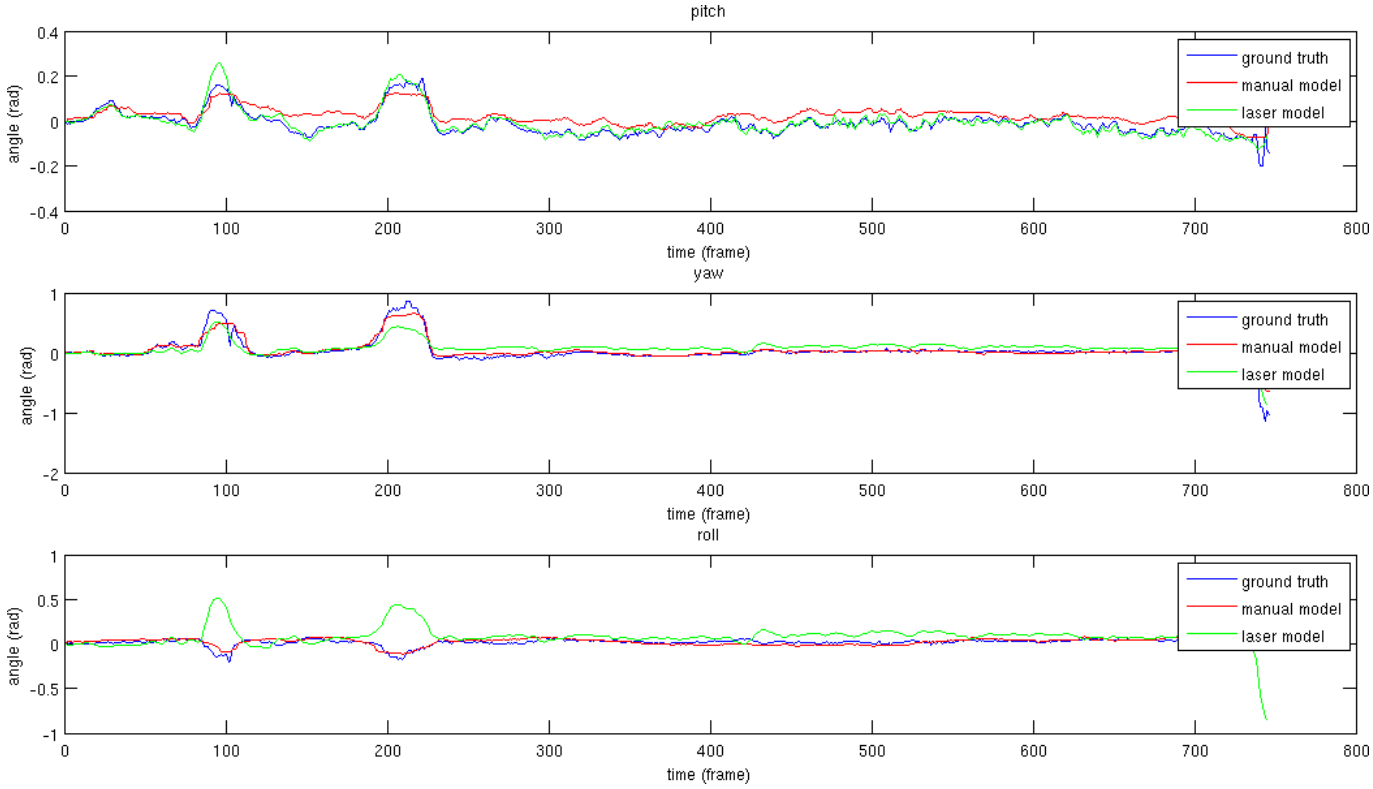


Figura 7.27: Ángulos obtenidos para la secuencia número 1

7.3.2.3. Secuencia nº 3

En esta secuencia se muestra un caso en el que se aprecia claramente la importancia de la colocación inicial correcta de la malla y la configuración de la misma.

Tanto en la tabla 7.4 como en la figura 7.28 se puede apreciar cómo la colocación inicial es fundamental. En todas las gráficas se ha eliminado el offset inicial que se tenía en el programa para poder comprarar correctamente con los resultados de ground truth y del programa original, puesto que el offset es en realidad un parámetro inicial que no tiene importancia si queremos analizar realmente la desviación de uno de los parámetros con el resto de evaluaciones.

En este caso, el vídeo completo es de una longitud de 3513 frames, de los cuales se evalúan 138 para el ground truth.

Método de obtención de la malla	Número de puntos en el modelo	número de frames evaluados	error medio pitch (rad)	error medio yaw (rad)	error medio roll	error medio total (rad)
manual	17	138	0.0028	0.0015	$6{,}6658 \cdot 10^{-4}$	0.0060
automático	50	715	0.0445	0.0194	0.0055	0.0224

Tabla 7.4: Tabla resumen de error para secuencia número 3

Se puede apreciar cómo la magnitud de la estimación obtenida con esta malla ampliada, es algo superior en alguno de los ángulos. La forma de las gráficas se preserva, pero algo menos dependiendo del ángulo. Para los 2 primeros ángulos (pitch y yaw) la estimación no es aceptable puesto que en el yaw supera el grado de error por frame y en el pitch los 2.5 º, mientras que para el roll lo es algo más. En nuestra aplicación el ángulo que da el giro respecto al frente de la cabeza hacia un lateral u otro (yaw), es el más importante. Sin embargo, aunque el error obtenido con este método haya sido algo superior, la ventaja fundamental es que añadir muchos más puntos da mayor robustez a la hora de ejecutar RANSAC, puesto que se tienen más puntos que pueden pertenecer al modelo, en los que apoyarse para estimar una pose y no perderse cuando el giro que se realice con la cabeza sea muy extremo.

También hay que tener en cuenta las magnitudes en las que se muestran las gráficas. Así mostradas podría parecer que el error es bastante grande, sin embargo, si nos fijamos en las magnitudes y las comparamos con el resto de gráficas, vemos como la resolución de las mismas es alta, y además se aproximan a la forma, por tanto la estimación no es tan mala como se podría llegar a pensar. Así también se muestra en las gráficas de error, puesto que se obtienen errores comparables con el resto de experimentos.

7.3.3. Tiempos de ejecución

También se ha realizado un pequeño análisis sobre los tiempos de ejecución del algoritmo con una gran cantidad de puntos. El número de iteraciones posibles que se deberían ejecutar para obtener la solución sigue un comportamiento como el que se muestra en la figura 7.29. Esta gráfica muestra el número de iteraciones necesarias en función del número de puntos que se tengan en total y también del subconjunto que se tome en cada iteración, esto es, se tiene una gráfica de las combinaciones del número total de puntos 3D tomados de N en N para ejecutar POSIT. El algoritmo RANSAC, para que funcione bien y encuentre una estimación aceptable de parámetros que se ajusten al modelo, se ha de ejecutar aproximadamente un 10 % de las combinaciones posibles. Vemos que con una cantidad de puntos como la que se venía utilizando, esto es, alrededor de 30, el número de iteraciones es razonable incluso para su ejecución en tiempo real. En cambio, si nos vamos a una mayor cantidad de puntos, como los utilizados en

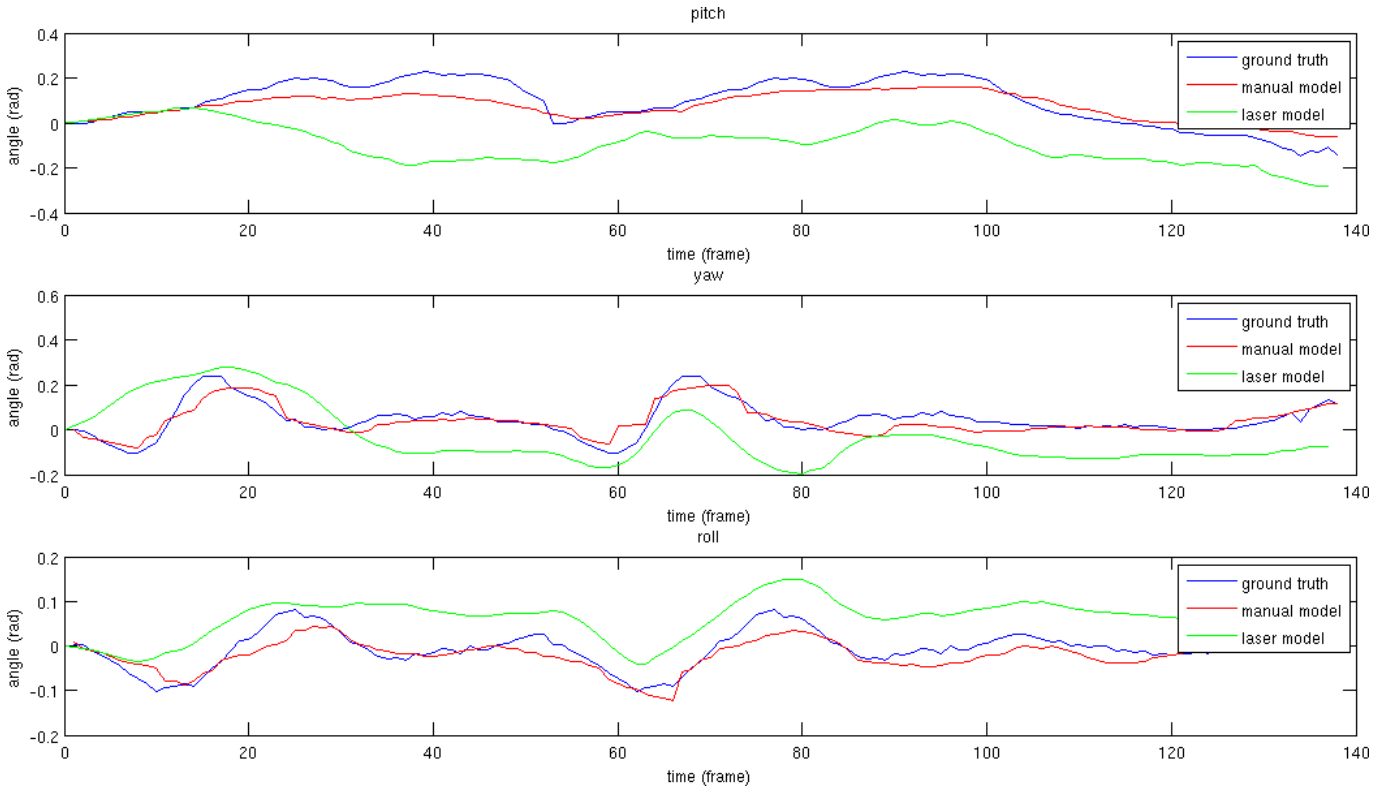


Figura 7.28: Ángulos obtenidos para la secuencia número 3

esta Tesis de Máster, entre 64 y 128, el número de iteraciones se dispara, como podemos ver, lo cual hace inviable su ejecución en tiempo real.

Con el modelo de 59 puntos 3D se obtuvieron unos tiempos de ejecución de 220 ms en media de todo el ciclo RANSAC + POSIT, en cambio, cuando el modelo utilizado es de 128 puntos, el tiempo de ejecución ha sido superior a los 330 ms.

7.3.4. Conclusiones parciales

La ampliación de tiempos no se puede introducir como mejora para un proceso de tiempo real, al tener tiempos de ejecución como los que se han expuesto anteriormente. Estos están superando varias veces los límites de tiempo impuestos, puesto que, para una tasa de adquisición de 30 fps, el límite de tiempo es de 33,3 ms y por tanto se hace totalmente inviable, solo en el tiempo de ejecución de estimación de la pose, porque además, el resto del programa realiza otros procesos, lo cual hace que su ejecución sea todavía más lenta y por tanto totalmente inviable.

El aumentar el número de puntos se planteó como medida para aumentar la robustez y la precisión del sistema, pero según se configure el modelo inicialmente y por el número de combinaciones que puede ejecutar RANSAC, no se obtiene un aumento de la precisión significativo, sino que, si no se ha configurado correctamente, el modelo puede inclusive perderla. Sin embargo, la robustez si que ha aumentado, puesto que sin realizar una configuración demasiado exhaustiva, se ha logrado realizar un seguimiento de la cabeza a lo largo de todas las secuencias sin cortes (a menos que estos sean debidos a fallos en la memoria o similares por ser una versión experimental

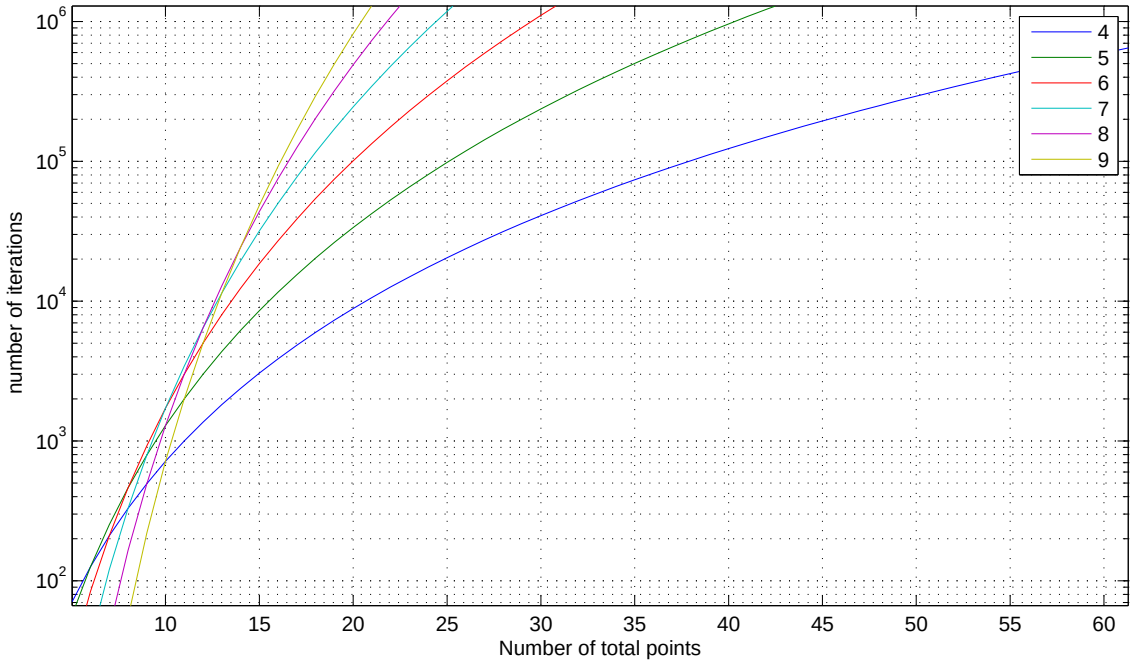


Figura 7.29: Número de iteraciones totales en función del número de puntos

del programa la utilizada para estas pruebas) ni tener que recolocar el modelo en algún punto concreto de las secuencias de test.

Capítulo 8

Conclusiones y Trabajos futuros

8.1. Conclusiones finales

Haciendo un análisis de combinatoria para el número de puntos a utilizar para ampliar la precisión del sistema, se ha llegado a la conclusión de que, para hacer que RANSAC sea determinista, se han de generar unas secuencias aleatorias de gran longitud, puesto que a medida que el número de puntos en el modelo crece, el crecimiento del número de secuencias a generar y el número de iteraciones necesarias para que al ejecutar RANSAC se cubra el 1 % de las posibilidades es factorial, cuyo crecimiento es bastante rápido, incluso más que el exponencial.

La implementación de POSIT optimizada, en la simulación ha resultado buena, pero en una implementación con condiciones reales, en las que no todos los puntos se ajustan al modelo exactamente no supera los resultados de POSIT normal. Además existen casos en los que aparecen picos en el tiempo de ejecución debido a inicializaciones incorrectas del algoritmo que hacen que tarde más en calcular la estimación y a entradas del sistema operativo en el procesador. Por último la dependencia del funcionamiento del algoritmo con el ángulo de rotación entre frames, lleva a que el algoritmo solo funciona cuando la variación es muy pequeña, cosa que no ocurre más que en un contado número de ocasiones, por lo que, en media el funcionamiento del algoritmo es igual. La implementación de OpenCV está optimizada al máximo mientras que esta implementación no está tan optimizada, por la inicialización y otros aspectos, por lo que el rendimiento es inferior al original.

Por último, la robustez del sistema aumenta a la par que la precisión se ve un tanto reducida en determinados casos, para los cuales la malla completa no esté bien ajustada inicialmente. El seguimiento se puede realizar a lo largo de mucho más tiempo sin que se pierda la pose y la traslación de la cara respecto de las cámaras, pero a cambio perdemos precisión, sobre todo en los parámetros más importantes, esto es, en los ángulos que conforman la pose.

8.2. Trabajos futuros

8.2.1. Configuración inicial de la malla una vez generada

Para el encajado automático de la malla en el frame inicial, se propone restringir el área de interés mediante el detector Viola & Jones. Así sabremos en una zona más exacta dónde colocar la malla inicialmente. Posterior a la aplicación de esta malla se comprobarían los puntos que proyectados diesen máximos y mínimos en las 2 dimensiones, para obtener una primera

estimación de la traslación que va a tener el modelo en la imagen, en lugar de utilizar una traslación inicial fija.

Para realizar un ajuste todavía mejor de los puntos, se tomarían, al igual que el algoritmo del programa completo, una serie de parches desde las imágenes obtenidas del escáner, para, por apariencia y correlación local, encajar algunos de los puntos, sobre todo de la parte frontal, obteniendo un posicionado inicial bastante próximo al que se encuentre en la imagen.

Como se ha dicho anteriormente, se pretende realizar una mejor estimación de ángulos de ocultación, tomando la malla 3D diezmada y trazando rayos desde cada punto para ver cuales son los puntos en 3D que pueden ocultarlos si se gira hacia los laterales, y tomar ese ángulo como ángulo de ocultamiento, con cierto márgen y en los 2 sentidos de rotación. Así se evitarían hacer aproximaciones como las antes realizadas, y por tanto una mala estimación de estos ángulos de ocultamiento, sobre todo con puntos cercanos a la nariz, los cuales tienen márgenes muy asimétricos. Este método de cálculo sería el más cercano a lo que sería la formación de imágenes real de una figura en 3D sobre un plano en 2D, que es exactamente lo que se intenta hacer con esta aproximación de tracking 3D basada en la proyección de la apariencia sobre 2 cámaras.

8.2.2. Estimación de la Pose

En cuanto a la estimación de la pose, se proponen varias mejoras que se pasan a enumerar a continuación.

En primer lugar se propone un estudio más a fondo de POSIT para ver cómo calcular la matriz pseudo-inversa de puntos 3D para reducir el tiempo de inicialización, o también estudiar cómo operar con una parte de los puntos 3D en lugar de con todos a la vez, para poder tener almacenados todos los puntos de un objeto 3D y ver de n en n cuales son los que mejor encajan a la hora de estimar la pose.

En lugar de ejecutar RANSAC siempre un número fijo de iteraciones, consumir un poco más de tiempo en encontrar un subconjunto bueno de puntos e ir evaluando el error haciendo que este descienda con un algoritmo de descenso por el grandiente, utilizando como métrica el error de reproyección en lugar del error devuelto por POSIT, pues como hemos visto en las simulaciones, este error no es el que realmente existe a la hora de reproyectar, sino que el error es el que existe respecto a la iteración anterior del algoritmo, la cual no conocemos.

Una alternativa para implementar el algoritmo completo de estimación de pose sería la utilización del modelo sin diezmar, proyectarlo entero sobre la imagen utilizada (en el caso de mallas con 4000 puntos o menos, y en lugar de utilizar un RANSAC y POSIT para obtener la pose, utilizar mínimos cuadrados u otro método de resolución rápido y correlaciones entre puntos para encontrar los puntos correspondientes entre frames. Al tener muchos más puntos, el sistema podría ser bastante más preciso y muchísmo más robusto de lo que lo es con estos puntos encontrados según el método de diezmado presentado.

8.2.3. Automatización completa del proceso de obtención de la malla

Para automatizar el archivo de configuración de la ROI para la extracción de puntos con Harris, se propone utilizar una primera fase con el detector de Viola & Jones para restringir automáticamente la zona donde se haya la cara, utilizando tanto las plantillas de cara lateral como las de cara frontal, para poder detectar la cara en cualquiera de las 3 vistas que se tienen, normalmente obtenidas desde el escáner. Así se evitaría parte de la configuración inicial requerida para la extracción de puntos característicos. Esta configuración inicial se vería reducida a la

introducción de los nombres de archivo y de las imágenes correspondientes, puesto que el resto de parámetros referentes a la ROI no serían necesarios.

Otra parte mejorable sería la de extracción de puntos 3D desde los archivos vrml (vrmlextract), al obtener las matrices, puesto que se ve cómo existen puntos que por andar cerca de pelo no obtienen correspondencia aún aplicando margen suficiente. Los modelos obtenidos han sido suficientemente densos para lo que se pretendía en esta Tesis de Máster, pero si se pretende aumentar el número de puntos de la malla para obtener modelos de entre 200 y 1000 puntos se debería refinar este algoritmo y tener también en cuenta la información de la matriz que da las conexiones entre puntos, esto es, la que da forma a la malla 3D. Corrigiendo esto también se reduciría el número de errores encontrados en algunos puntos, que han tendio que ser eliminados en etapas intermedias del algoritmo.

En cuanto a la parte más crítica del proceso, que se debe de realizar de momento con el apoyo de otros programas, es la búsqueda de puntos correspondientes entre las distintas vistas y el cálculo automático de la matriz de transformación entre vistas. Para realizarlo, se intentó buscar esta correspondencia con distintos procedimientos que se pasan a enumerar a continuación:

1. En primer lugar se probó utilizando transformaciones de perspectiva habituales, como perspectiva afín, proyectiva, etc., pero éstas no funcionaron puesto que la distorsión introducida era importante y por tanto se descartó el método automáticamente.

2. La segunda aproximación que se realizó fue utilizar algoritmos tipo SIFT o SURF (este último implementado en OpenCV) para la obtención automática de puntos correspondientes, pero no se obtuvieron buenos resultados aplicando posteriormente RANSAC a los resultados obtenidos de correspondencias, por lo que el método quedó descartado

3. Por último, se optó por la solución adoptada al final, esto es, utilizar un programa externo que fuese capaz de leer los datos completos para poder encontrar la matriz de transformación y que fuese capaz de devolverla de alguna forma, incluso modificando el código fuente, para poder utilizar esta matriz para unir las distintas vistas.

Por tanto, la línea de investigación que habría que seguir a partír de ahora sería profundizar en el algoritmo SURF o en algunas de sus variantes, puesto que en sus parámetros no se profundizó demasiado al no encontrar, después de varias pruebas resultados buenos. Por otro lado se podría aplicar este algoritmo entre frames próximos para generar un modelo incremental, esto es, que se amplie automáticamente según se van encontrando nuevos puntos a lo largo de la secuencia.

También sería interesante probar con correlaciones en ventanas pequeñas en determinados puntos y comprobar qué zonas de la imagen son similares y ver dónde están colocadas. Para restringir el área sería interesante también usar el detector de Viola & Jones, lo cual eliminaría falsos positivos y además, reduciría el tiempo de cómputo.

Si, aun dedicando más esfuerzos en esta parte no se consiguen resultados correctos se podría intentar algun método que encaje 3D de la malla basado en correlación de trozos de malla que sean comunes. También se puede intentar implementar esta alineación basándonos en transformadas, puesto que, concretamente la de Fourier tiene propiedades de invarianza a rotación y traslación. La transformada Wavelet sería útil si se buscase en multiresolución cuál es la parte en común entre las 2 vistas y estudiar el resto de componentes de la transformada para encontrar la relación entre vistas.

Parte III

Apéndice

Apéndice A

Manual de puesta a punto e instalación

A.1. Instalación de Linux

Para el desarrollo e implementación del sistema se ha utilizado la distribución de Linux Ubuntu / Kubuntu. Está basada en Debian y se ha elegido por la facilidad de instalación (basado en el instalador de Debian) y de configuración posterior. La versión desde la que se empezó con este proyecto a funcionar fue la 8.04 y posteriores.

Si la versión de Linux que se va a instalar es distinta de las anteriores, las instrucciones de instalación que se indican a continuación pueden variar ligeramente, puesto que cada instalación tiene su propio instalador. Algunas distribuciones permiten la selección de paquetes que se van a instalar antes de comenzar la copia de archivos. Otras distribuciones permiten elegir el entorno gráfico con el que queremos trabajar.

Para poder instalarlo necesitamos disponer del CD live, en el cual está incluida la instalación. Debemos arrancarlo al inicio del ordenador, antes de que arranque el sistema operativo del disco duro, por lo que si no está activada la opción de arranque, habrá que activarla en la BIOS.

Una vez ha arrancado, tenemos dos opciones, instalarlo en modo gráfico reducido (tipo MS-DOS), modo texto o arrancar el sistema operativo en modo live, probar el sistema y si nos convence instalarlo con el icono install que aparecerá en el escritorio. Los dos modos de instalación son muy parecidos, pero están lo suficientemente guiados para ser sencilla la instalación y no entrar en demasiada profundidad en este proceso, en el cual se piden cosas comunes a los sistemas operativos, como por ejemplo, seleccionar la zona horaria, el idioma y todos los formatos asociados (fecha y hora, teclado, etc), el particionado de disco, paquetes especiales a instalar (si está disponible con el instalador), etc.

A.2. Configuración de Linux

Sobre la instalación básica de este sistema operativo empezamos a instalar los paquetes necesarios para la lectura de los distintos formatos de imágenes y vídeos, codecs, librerías, compiladores, drivers y archivos de cabecera necesarios.

A.2.1. Compiladores

Para empezar, instalamos los compiladores pues es lo más básico con lo que vamos a trabajar. De todos los posibles instalamos los que más se utilizan, que son gcc y g++, en su versión 4.2, la cual viene por defecto.

La instalación de esta versión es necesaria si no es la que viene por defecto. Por tanto es recomendable hacer lo siguiente:

- apt-get/aptitude install gcc-4.2 y g++-4.2 (y todas sus dependencias)
- el gestor gráfico de paquetes synaptic (opción elegida).
- paquetes precompilados, otros administradores de paquetes con repositorios, etc.

Esta versión del compilador es necesaria para la correcta compilación del paquete ffmpeg que necesitaremos posteriormente. Si teníamos instalada otra versión, para hacer que funcione la que necesitamos, tenemos que introducir los siguientes comandos:

1. eliminamos los links actuales

```
sudo rm /usr/bin/gcc
sudo rm /usr/bin/g++
```

2. redirigimos a la versión que necesitamos

```
sudo ln  -s /usr/bin/gcc-4.x /usr/bin/gcc
sudo ln  -s /usr/bin/g++-4.x /usr/bin/g++
```

Una vez hemos hecho esto, ya podemos pasar a instalar las librerías necesarias.

A.2.2. Firewire (IEEE 1394)

A partir de las versiones 2.14.8 del kernel de Linux, el control de estos dispositivos viene integrado en el núcleo, pero para poder utilizarlo correctamente no solo nos basta con esto, sino que tenemos que introducir algunas librerías especificas para poder tratar sus datos correctamente.

Estas librerías son: raw1394, video1394, ohci1394 e ieee1394.

Las que no estén instaladas y activas, las instalamos con synaptic y las activamos con modprobe y la correspondiente librería si es que todavía no está funcionando después de instalar y reiniciar la máquina. Para este dispositivo tenemos la particularidad de que OpenCV puede no conectar directamente con el nombre de dispositivo adecuado, con lo que tendremos que hacer ciertos arreglos para que funcione correctamente.

Para comprobar que funciona todo correctamente, instalamos el programa Coriander y comprobamos que las capturas sean correctas.

A.2.3. Librerías

Necesitamos varias librerías para hacer funcionar correctamente todo lo que necesitamos en OpenCV y ffmpeg (los paquetes con el nombre de la librería correspondiente y los dev, que contienen el código fuente necesario para alguna que otra compilación o como mínimo las cabeceras)

- gtk2+
- glade
- libpng
- zlib
- libjpeg
- libtiff
- libjasper

Además de las 4 mencionadas antes para Firewire (raw, video, ohci e ieee)

A.2.4. Ffmpeg

Estas librerías proporcionan los codecs para tratar video comprimido con otros programas como pueden ser reproductores de video, plataformas de procesamiento de vídeo, etc.

Nosotros lo utilizaremos para que OpenCV posea sus librerías de procesamiento de vídeo y además para generar vídeos de prueba o de resultados a partir de otros vídeos o de imágenes ya procesadas.

Descargamos este programa en una versión compatible con OpenCV 1.0.0 o OpenCV 1.1.0.

La instalación de este programa se realiza como se indica en los txt de instalación:

```
./configure  --enable-shared
make
sudo make install
```

Después de la instalación, tenemos que añadir un par de carpetas para que la compilación de OpenCV sea correcta.

En el archivo /etc/ld.so.conf hay que añadir lo siguiente

```
/usr/local/lib
```

y en el archivo .profile (que está en la carpeta del correspondiente usuario), añadir:

```
$export LD_LIBRARY_PATH=/usr/local/lib
$export LD_LIBRARY_PATH=/usr/local/lib/opencv
```

Con estos 2 tendremos las carpetas donde están las librerías disponibles para que luego, en la compilación de OpenCV sean fácilmente accesibles y no nos den errores de compilación.

Si en la instalación ha resultado que las librerías no funcionasen, se recomienda repetir el proceso, realizando una desinstalación, mediante el siguiente comando:

```
sudo make uninstall
```

A.2.5. OpenCV

Estas son unas librerías básicas, de código abierto, que proporcionan una buena plataforma para el procesamiento de imagen y vídeo y posterior aplicación de visión artificial.

Estas librerías las descargamos desde sourceforge en su versión 1.0.

En caso de que no tengamos la cámara en la ubicación adecuada (/dev/video1394/0) tenemos 2 opciones para redirigirla: editar el archivo cvcap_dc1394.cpp y allí donde pone:

```
static char * videodev[4] = {
"/dev/video1394/0",
"/dev/video1394/1",
"/dev/video1394/2",
"/dev/video1394/3",
};
```

cambiamos los nombres de ese array por los nombres de dispositivo de vídeo que nos haya creado el sistema de forma predeterminada y recompilar OpenCV con esta modificación.

La segunda opción, que es por la que hemos optado, es crear un script con los siguientes comandos:

```
sudo mkdir /dev/video1394
sudo ln  -s /dev/video1394-0 /dev/video1394/0
```

Una vez hecho esto enlazamos el script que contiene estas 2 líneas al inicio del sistema, creando un enlace simbólico en la carpeta en la que se guardan los scripts de arranque del sistema en el modo adecuado. En nuestro caso utilizamos el 5

```
sudo ln  -s nombrescript /etc/rc5.d/S99video1394
```

Ahora estamos en condiciones de instalar OpenCV:

```
./configure
```

Con esta orden, comprobamos qué es lo que tiene y no instalado el sistema, para ver si tenemos todo en orden. Si entre los resultados sale que ffmpeg no está, hay que revisar la configuración de alguno de los pasos que hemos realizado, para hacer que funcione, pues es una parte básica. Si el problema encontrado es un problema de incompatibilidad de rutas, se pueden hacer unos links de la ruta predeterminada a la que nos pide, aunque lo mejor es editar es modificar el archivo configure, introduciendo las rutas correctas a los archivos de las cabeceras de ffmpeg, esto es a libswscale, libavformat, livavcodec, libavutils, etc.

Una vez se ha obtenido una detección de estas librerías correcta, tecleamos:

```
make
```

Y si la compilación es correcta:

```
sudo make install
```

Una vez hemos comprobado que la instalación se ha realizado correctamente y hemos añadido las rutas indicadas anteriormente a los archivos /etc/ld.so.conf y en .profile de la carpeta del usuario, ejecutamos el siguiente comando para que se carguen todos los módulos que hemos instalado:

```
sudo ldconfig
```

Con esto cargaremos todas las librerías que acabamos de instalar y que estén en los directorios que hayamos dejado indicados en los archivos indicados anteriormente. Con la opción -v podemos ver por pantalla que módulos se están cargando y que módulos o no.

Con esto ya hemos terminado la configuración y podemos empezar a trabajar.

A.3. Preparación de las aplicaciones

Una vez se haya instalado todo lo que se ha visto anteriormente, en los distintos directorios, lo que quedará por hacer será ejecutar en la consola los comandos make clean, para borrar todos los archivos antiguos, y make para generar los ejecutables.

A.3.1. Meshlab

Se utiliza este software como medio para extraer la matriz de transformación entre 2 matrices, una vez se han encontrado (manualmente) los puntos 3D correspondientes entre 2 mallas parecidas con puntos en común. Este software funciona tanto para windows como para linux, pero el problema que para windows, no se tienen los fuentes, mientras que para windows, sí, con lo que es posible así modificar el código fuente para poder sacar por consola o pantalla datos que nos son útiles, que en nuestro caso son las matrices de transformación y las coordenadas de los puntos 3D seleccionados para calcular las transformaciones.

Este software se puede bajar, como OpenCV de sourceforge, y es posible ejecutarlo en varias plataformas, siendo casi totalmente compatible entre las mismas al utilizar QT como motor gráfico. Para instalarlo en linux, es necesario bajar las fuentes y compilarlas. La compilación, como en la mayoría de programas de linux se hace con los siguientes comandos:

```
./configure
make
sudo make install
```

En ocasiones puede que haya partes del paquete que no estén del todo actualizadas, por lo que es conveniente bajar también alguna fuente de versiones anteriores, por si la compilación falla por estos detalles. Las versiones descargadas en nuestro caso para una correcta compilación han sido la 1.2 y la 1.1. Una vez se tenga compilado el programa y funcionando la primera vez, se comprueba que al menos la funcionalidad de la alineación funcione.

Meshlab no trabaja con archivos .vrml, como se venía tratando hasta ahora, lo que fuerza a que cuando se guarden los modelos desde el software de adquisición, no solo como vrml sino también como .stl, para que pueda leerlos Meshlab y trabajar con las mallas.

Una vez se ha compilado y comprobado que el programa funciona correctamente, se pasa a editar el código fuente para que extraiga a la salida la matriz de transformación. Para ello vamos

al archivo alignDialog.cpp y e introducir el código necesario para mostrar la matriz de rotación. En este caso se ha introducido lo siguiente:

```
MeshTreeWidgetItem::MeshTreeWidgetItem(MeshTree* meshTree, AlignPair::Result *A,
                                       MeshTreeWidgetItem *parent){

  ...

  QString buf=QString("Arc: %1 -> %2 Area: %3 Err: %4 Sample# %5 (%6)")
                      .arg((*A).FixName)
                      .arg((*A).MovName)
                      .arg((*A).area, 6,'f',3)
                      .arg((*A).err,  6,'f',3)
                      .arg((*A).ap.SampleNum,6)
                      .arg((*A).as.LastSampleUsed() );
  setText(0,buf);

/*Modification to show RT data*/
  std::cout << "TR: " << std::endl << std::endl;
  std::cout << A->Tr.ElementAt(0,0) << " " << A->Tr.ElementAt(0,1) << " " <<
               A->Tr.ElementAt(0,2) << " " << A->Tr.ElementAt(0,3) <<  std::endl;
  std::cout << A->Tr.ElementAt(1,0) << " " << A->Tr.ElementAt(1,1) << " " <<
               A->Tr.ElementAt(1,2) << " " << A->Tr.ElementAt(1,3) <<  std::endl;
  std::cout << A->Tr.ElementAt(2,0) << " " << A->Tr.ElementAt(2,1) << " " <<
               A->Tr.ElementAt(2,2) << " " << A->Tr.ElementAt(2,3) <<  std::endl;
  std::cout << A->Tr.ElementAt(3,0) << " " << A->Tr.ElementAt(3,1) << " " <<
               A->Tr.ElementAt(3,2) << " " << A->Tr.ElementAt(3,3) <<  std::endl <<
               std::endl;
  /*End of Modification to show RT data*/

  ...
```

Con esta matriz como salida en la consola, cuando realicemos la selección de puntos y procesemos la matriz de transformación, podremos copiarla en Matlab y, con una trasposición, realizar las transformaciones. El hecho de necesitar transponer la matriz es porque en las tarjetas gráficas se utilizan estas matrices directamente para realizar los cálculos, en lugar de utilizar las traspuestas.

Para la obtención de los puntos se ha de hacer los siguientes pasos:

1. Se cargan las 3 vistas de las que se quieren obtener los datos.
2. Se fija la vista frontal con el botón "Glue Mesh Here".
3. Se selecciona la otra vista y se pulsa al botón "Point Based Glueing"
4. Se seleccionan al menos 4 puntos por vista haciendo doble click en el punto de la vista correspondiente para seleccionar el punto. La selección de puntos ha de realizarse de manera que se seleccionan en una vista, y luego se busca y selecciona su correspondiente en la contraria. No se seleccionarán varios puntos en una vista y después, otros tantos en la otra vista.
5. Cuando se han introducido todos los puntos se pulsa intro y por último se pulsa al botón de Process.
6. Al dar a este botón se obtiene en la consola la matriz de transformación, la cual se copia en Matlab y se guarda en un archivo con la estructura que se sugiere en las instrucciones para la obtención de la malla completa.

En cuanto al resto de funcionalidad del programa, se pueden realizar transformaciones parciales de la maya, diezmado normal, diezmado preservando un poco la forma, mezcla de mallas, etc,

lo cual, para objetos que se vayan a incluir en escenas de simulación 3D se puede utilizar como un pequeño editor se tratase.

Bibliografía

[1] Rolf Adomat, Georg-Otto Geduld, Michael Schamberger, and Peter Reith. Advanced driver assistance systems for increased comfort an safety - current developments and an outlook to the future on the road. *Advanced Microsystems for Automotive Applications*, 2:431–446, 2003.

[2] IEE.Conf.Publ.No.483. Adas. international conference on advanced driver assistance systems. Number 483, 2001.

[3] Volvo Cars. City safety terms and conditions. `http://www.volvocars.com/es/footer/privacypolicy/Pages/CitySafetyTerms.aspx`, 2009.

[4] Volvo Cars. Volvo alert control. `http://www.volvocars.com/us/footer/about/NewsAndEvents/default/Pages/default.aspx?item=33`, 2007.

[5] Mercedes-Benz. Mercedes-benz to introduce attention assist into series production in spring 2009. `http://www.emercedesbenz.com/Aug08/11_001331_Mercedes_Benz_To_Introduce_Attention_Assist_Into_Series_Production_In_Spring_2009.html`, 2009.

[6] Onda Regional De Murcia. Bmw lucha contra los conductores dormidos. prueba un sistema que alerta antes de que aparezca el sueño. `http://www.orm.es/programas/revista_motor/motor150.htm`, Diciembre 2002.

[7] Lexus. Ls hybrid safety and security - advanced pre-collision system (apcs) with driver attention monitor. `http://www.lexus.com/models/LSh/features/safety/advanced_precollision_system_apcs_with_driver_attention_monitor.html`, 2009.

[8] Seeing Machines. Facelab. `http://www.seeingmachines.com/product/facelab`, 2009.

[9] Seeing Machines. Driver state sensor - fleet. `http://www.seeingmachines.com/product/dss`, 2009.

[10] Nick Cerneaz. Driver alertness. *Vision zero international*, pages 62–65, 2009.

[11] Seeing Machines. Faceapi. `http://www.seeingmachines.com/product/faceapi`, 2009.

[12] Cabintec. Cabintec, proyecto de desarrollo de cabina inteligente para el transporte por carretera para mejorar la seguridad vial mediante tecnologías inteligentes, comportamiento del conductor. españa. `http://www.cabintec.net`, 2009.

[13] Philip David, Daniel DeMenthon, Ramani Duraiswami, and Hanan Samet. Simultaneous pose and correspondence determination using line features. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages II–424–II–431 vol.2, 2003.

[14] Chris Harris and Mike Stephens. A combined corner and edge detector. *Alvey Vision Conference*, pages 365–376, 1988.

[15] Forsyth and Ponce. *Computer Vision, A modern Approach.* Prentice Hall, 2003, 2003.

[16] J.C., Abderrahim, and M. Diaz. Modified softposit algorithm for 3d visual tracking. In *Proc. IEEE International Symposium on Intelligent Signal Processing WISP 2007*, pages 1–6, 2007.

[17] Pedro Jimenez, Jesús Nuevo, and Luis M. Bergasa. Face pose estimation and tracking using automatic 3d model construction. *Computer Vision and Pattern Recognition Workshops, 2008. CVPR Workshops 2008. IEEE Computer Society Conference on*, pages 1–7, 2008.

[18] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. *CVPR*, 2001.

[19] Erik Murphy-Chutorian and M. M. Trivedi. Head pose estimation in computer vision: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009.

[20] Niyogi and W. Freeman. Example−based head tracking. *Proc. Int'l. Conf. Automatic Face and Gesture Recognition*, pages 374–378, 1996.

[21] D. Beymer. Face recognition under varying pose. *Proc IEEE Conf. Computer Vision and Pattern Recognition*, pages 756–761, 1994.

[22] J. Sherrah, S. Gong, and E. J. Ong. Face distributions in similarity space under varying head pose. *Image and Vision Computing*, 19(12):807–819, 2001.

[23] J. Huang, X. Shao, and H. Wechsler. Face pose discrimination using support vector machines (svm). *Proc. Int'l. Conf. Pattern Recognition*, pages 154–156, 1998.

[24] M. Jones and P. Viola. Fast multi−view face detection. *Mitsubishi Electric Research Laboratories, Tech. Rep.*, (96), 2003.

[25] H. Rowley, S. Baluja, and T. Kanade. Rotation invariant neural network−based face detection. *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 38–44, 1998.

[26] Y. Li, S. Gong, J. Sherrah, and H. Liddell. Support vector machine based multi−view face detection and recognition. *Image and Vision Computing*, 22(5):2004, 2004.

[27] E Murphy-Chutorian, A Doshi, and MM Trivedi. Head pose estimation for driver assistance systems: A robust algorithm and experimental evaluation. *Proc. IEEE Conf. Intelligent Transportation Systems*, pages 709–714, 2007.

[28] H. Moon and M. Miller. Estimating facial pose rom a sparse representation. *Proc. Int'l. Conf. Image Processing*, pages 75–78, 2004.

[29] E. Seemann, K. Nickel, and R. Stiefelhagen. Head pose estimation using stereo vision for human−robot interaction. *Proc. IEEE Int'l. Conf. Automatic Face and Gesture Recognition*, pages 626–631, 2004.

[30] R. Stiefelhagen, J. Yang, and A. Waibel. Modeling focus of attention for meeting indexing based on multiple cues. *IEEE Trans. Neural Networks*, 13:928–938, 2002.

[31] M. Voit, K. Nickel, and R. Stiefelhagen. Head pose estimation in single and multi−view environments results on the clear'07 benchmarks. *Proc. Int'l. Workshop Classification of Events Activities and Relationships*, 2007.

[32] M. Osadchy, Y. Le Cun, and M.L. Miller. Synergistic face detection and pose estimation with energy−based models. *J. Machine Learning Research*, 8:1197–1215, 2007.

[33] R. Rae and H. Ritter. Recognition of human head orientation based on artificial neural networks. *IEEE Trans. Neural Networks*, 9(2):257–265, 1998.

[34] N. Gourier, D. Hall, and J. Crowley. Estimating face orientation from robust detection of salient facial structures. *Proc. Pointing 2004 Workshop: Visual Observation of Deictic Gestures*, pages 17–25, 2004.

[35] S. Srinivasan and K. Boyer. Head pose estimation using view based eigenspaces. *Proc. Int'l. Conf. Pattern Recognition*, pages 302–305, 2002.

[36] Y. Fu and T. Huang. Graph embedded analysis for head pose estimation. *Proc. IEEE Int'l. Conf. Automatic Face and Gesture Recognition*, pages 3–8, 2006.

[37] J. Wu and M. Trivedi. A two−stage head pose estimation framework and evaluation. *Pattern Recognition*, 41(3):1138–1158, 2008.

[38] N. Hu, W. Huang, and S. Ranganath. Head pose estimation by non−linear embedding and mapping. *Proc. IEEE Int'l. Conf. Image Processing*, 2:342–345, 2005.

[39] B. Raytchev, I. Yoda, and K. Sakaue. Head pose estimation by nonlinear manifold learning. *Proc. Int'l. Conf. Pattern Recognition*, pages 462–466, 2004.

[40] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15:1373–1396, 2003.

[41] S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[42] V. Balasubramanian, J. Ye, and S. Panchanathan. Biased manifold embedding: A framework for person−independent head pose estimation. *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2007.

[43] S. Yan, Z. Zhang, Y. Fu, Y. Hu, J. Tu, and T. Huang. Learning a person−independent representation for precise 3d pose estimation. *Proc. Int'l. Workshop Classification of Events Activities and Relationships*, 2007.

[44] N. Krüger, M. Pötzsch, and C. von der Malsburg. Determination of face position and pose with a learned representation based on labeled graphs. *Image and Vision Computing*, 15(8):665–673, 1997.

[45] A. Lanitis, C. Taylor, and T. Cootes. Automatic interpretation of human faces and hand gestures using flexible models. *Proc. IEEE Int'l. Conf. Automatic Face and Gesture Recognition*, pages 98–103, 1995.

[46] T. Cootes, K. Walker, and C. Taylor. View−based active appearance models. *in Proc. Int'l. Conf. Automatic Face and Gesture Recognition*, pages 227–232, 2000.

[47] J. Xiao, S. Baker, I. Matthews, and T. Kanade. Real−time combined 2d+3d active appearance models. *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2:535–542, 2004.

[48] A. Gee and R. Cipolla. Determining the gaze of faces in images. *Image and Vision Computing*, 12(10):639–647, 1994.

[49] T. Horprasert, Y. Yacoob, and L. Davis. Computing 3d head orientation from a monocular image sequence. *Proc. Int'l. Conf. Automatic Face and Gesture Recognition*, pages 242–247, 1996.

[50] J.G. Wang and E. Sung. Em enhancement of 3d head pose estimated by point at infinity. *Image and Vision Computing*, 25(12):1864–1874, 2007.

[51] A. Gee and R. Cipolla. Fast visual tracking by temporal consensus. *Image and Vision Computing*, 14(2):105–114, 1996.

[52] G. Zhao, L. Chen, J. Song, and G. Chen. Large head movement tracking using sift−based registration. *Proc. Int'l. Conf. Multimedia*, pages 807–810, 2007.

[53] P. Yao, G. Evans, and A. Calway. Using affine correspondence to estimate 3d facial pose. *Proc. Int'l. Conf. Image Processing*, pages 919–922, 2001.

[54] R. Yang and Z. Zhang. Model−based head pose tracking with stereovision. *Proc. Int'l. Conf. Automatic Face and Gesture Recognition*, pages 242–247, 2002.

[55] R. Pappu and P. Beardsley. A qualitative approach to classifying gaze direction. *Proc. IEEE Int'l. Conf. Automatic Face and Gesture Recognition*, pages 160–165, 1998.

[56] A. Schödl, A. Haro, and I. Essa. Head tracking using a textured polygonal model. *Proc. Workshop Perceptual User Interfaces*, 1998.

[57] Y. Wu and K. Toyama. Wide−range, person and illumination insensitive head orientation estimation. *Proc. IEEE Int'l. Conf. Automatic Face and Gesture Recognition*, pages 183–188, 2000.

[58] M. L. Cascia, S. Sclaroff, and V. Athitsos. Fast, reliable head tracking under varying illumination: An approach based on registration of texture−mapped 3d models. *IEEE Trans. Pattern Anal. Machine Intell.*, 22(4):322–336, 2000.

[59] J. Xiao, T. Moriyama, T. Kanade, and J. Cohn. Robust full−motion recovery of head by dynamic templates and reregistration techniques. *Proc. Int'l. J. Imaging Systems and Technology*, 13(1):85–94, 2003.

[60] L. P. Morency, A. Rahimi, N. Checka, and T. Darrell. Fast stereo−based head tracking for interactive environments. *Proc. Int'l. Conf. Automatic Face and Gesture Recognition*, pages 375–380, 2002.

[61] K. Oka, Y. Sato, Y. Nakanishi, and H. Koike. Head pose estimation system based on particle filtering with adaptive diffusion control. *Proc. IAPR Conf. Machine Vision Applications*, pages 586–589, 2005.

[62] E. Murphy-Chutorian and M. Trivedi. Hybrid head orientation and position estimation (hyhope): A system and evaluation for driver support. *Proc. IEEE Intelligent Vehicles Symposium*, 2008.

[63] T Horprasert, Y Yacoob, and LS Davis. An anthropometric shape model for estimating head orientation. *Proc. Int'l. Workshop Visual Form*, pages 247–256, 1997.

[64] T. Jebara and A. Pentland. Parametrized structure from motion for 3d adaptive feedback tracking of faces. *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 144–150, 1997.

[65] L.-P. Morency, P. Sundberg, and T. Darrell. Pose estimation using 3d view−based eigenspaces. *Proc. IEEE Int'l. Workshop Analysis and Modeling of Faces and Gestures*, pages 45–52, 2003.

[66] S. Ba and J. M. Odobez. A probabilistic framework for joint head tracking and pose estimation. *Proc. Int'l. Conf. Pattern Recognition*, pages 264–267, 2004.

[67] KS Huang and MM Trivedi. Robust real−time detection, tracking, and pose estimation of faces in video streams. *Proc. Int'l. Conf. Pattern Recognition*, pages 965–968, 2004.

[68] J. Sherrah and S. Gong. Fusion of perceptual cues for robust tracking of head pose and position. *Pattern Recognition*, 34(8):1565–1572, 2001.

[69] K.Toyama. look, ma - no hands! hands-free cursor control with real-time 3d face tracking. *Proc. Perceptual User Interfaces*, pages 49–54, 1998.

[70] D.Lowe. Distinctive image features from scale-invariant keyponints. *Int'l J. Computer Vision*, 60(2):91–110, 2004.