UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

INGENIERÍA
EN INFORMÁTICA

PROYECTO DE FIN DE CARRERA

ARGES: Sistema de Posicionamiento en Interiores Basado en Técnicas de Realidad Aumentada

Ricardo Muñoz Castellanos

Febrero, 2012

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Departamento de Informática

PROYECTO FIN DE CARRERA

ARGES: Sistema de Posicionamiento en Interiores Basado en Técnicas de Realidad Aumentada

Autor: Ricardo Muñoz Castellanos
Director: Carlos González Morcillo

Febrero, 2012

E-mail: ricardomc.universidad@gmail.com
© 2012 Ricardo Muñoz Castellanos.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front- Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Se permite la copia, distribución y/o modificación de este documento bajo los términos de la licencia de documentación libre GNU, versión 1.3 o cualquier versión posterior publicada por la *Free Software Foundation*, sin secciones invariantes. Puede consultar esta licencia en http://www.gnu.org.

Muchos de los nombres usados por las compañías para diferenciar sus productos y servicios son reclamados como marcas registradas. Allí donde estos nombres aparezcan en este documento, y cuando el autor haya sido informado de esas marcas registradas, los nombres estarán escritos en mayúsculas o como nombres propios.

Este documento fue compuesto con LaTeX. Imágenes generadas con OpenOffice.

TRIBUNAL:

Presidente:

Vocal 1:

Vocal 2:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL 1	VOCAL 2	SECRETARIO
Fdo.:	Fdo.:	Fdo.:	Fdo.:

“Tenemos dos ojos, pero solo una visión del mundo”

Susan Barry

Resumen

El campo de la visión por computador y de la Realidad Aumentada (RA) ha experimentado un gran auge en los últimos años, gracias, en gran parte, a los nuevos paradigmas de interacción, que surgen del preciso posicionamiento con 6 grados de libertad que se obtiene mediante el *tracking* óptico.

Actualmente existen multitud de servicios proporcionados sobre la base del geoposicionamiento, ya sea sobre dispositivos receptores GPS o en teléfonos móviles. Es habitual la utilización de esta tecnología como ayuda a la conducción o para localizar lugares de interés turístico en ciudades. Sin embargo, la propia naturaleza de GPS no permite su utilización en el interior de edificios.

En el proyecto Arges se plantea el desarrollo de un sistema de posicionamiento y orientación en interiores empleando diversos métodos basados en Visión por Computador (VC) y *tracking* 3D. El enfoque modular del proyecto permite la utilización de diferentes algoritmos y técnicas complementarias (como el uso de marcas visuales, descriptores de imagen o la construcción de mapas del entorno mediante visualización estereoscópica) que pueden ser utilizadas para calcular la posición del usuario en el interior de un edificio con gran precisión.

Abstract

Computer Vision and Augmented Reality (AR) fields have experimented great advances in the latest years. This improvement is due to the development of new interaction paradigms which are based on the accurate positioning with 6 degrees of freedom obtained through optical tracking.

Nowadays, a great amount of services are provided over the base of geopositioning. This technology is commonly used to help driving or to locate touristic interest areas. However, due to the GPS characteristics, the positioning is not possible indoors.

The main objective in the Arges project can be defined as the development of a system for positioning and orientation in indoor environments using different techniques based on computer vision (CV) and 3D tracking. The component-based architecture of the project allows the usage of different algorithms and techniques (such as visual beacons, image descriptors or environment mapping trough visual stereo) which can be utilized to obtain the 3D position inside a building with great accuracy.

Agradecimientos

Quiero dar las gracias a mi familia, en especial a mis padres y mi hermano, por su contínuo apoyo, cariño y por animarme siempre a seguir adelante.

Gracias a Isabel, por ser mi lugar en el mundo y mi alegría de cada día.

Gracias a todos los amigos que me han acompañado en los duros días de estudio a lo largo de la carrera. A mis compañeros de clase, en especial Pablo, con quien más noches de insomnio he compartido. A mis amigos de la residencia José Castillejo, con los que estuve viviendo durante 5 largos años y que son los que me han visto "crecer" como ingeniero informático.

Y, por supuesto, muchas gracias a los miembros de *Oreto*, por toda la ayuda y amistad que me han prestado y por cederme un hueco para trabajar dentro de su laboratorio. En especial, gracias a Carlos González Morcillo, por todo el tiempo que ha dedicado en mí y mi proyecto, por su ayuda, su paciencia y su amistad.

Sin vosotros, nada de esto habría sido posible.

Ricardo Muñoz Castellanos

Índice general

Resumen **XI**

Abstract **XIII**

Índice general **XIX**

Índice de figuras **XXI**

Índice de tablas **XXV**

Lista de algoritmos **XXVII**

Listado de acrónimos **XXIX**

1. Introducción **1**
1.1. Visión por computador y Realidad Aumentada 2
1.2. Impacto socio-económico 3
1.3. Justificación y problemática 4
1.4. Estructura del documento 5

2. Antecedentes **7**
2.1. Informática Gráfica 8
2.1.1. Base matemática 8
2.2. Odometría 15
2.2.1. SLAM 16
2.3. Visión por Computador 22
2.3.1. Fundamentos ópticos 22
2.3.2. *Tracking* y métodos de *tracking* 24
2.4. Visión Estereoscópica 26
2.4.1. Geometría epipolar 27
2.4.2. Fundamentos matemáticos 28
2.5. Metadatos 33
2.5.1. Estándar MPEG-7 34

3. Objetivos **39**

4. Metodo de trabajo **43**
4.1. Metodología de trabajo . . . 43
4.2. Herramientas . . . 43
4.2.1. ARToolkit . . . 44
4.2.2. OpenCV . . . 45
4.2.3. MPEG-7 FexLib . . . 50
4.2.4. Otras herramientas software . . . 51
4.2.5. Hardware . . . 53

5. Arquitectura **55**
5.1. Módulo de calibración de la cámara . . . 57
5.2. Módulo de Entrada/Salida . . . 59
5.2.1. Submódulo de entrada . . . 59
5.2.2. Submódulo de salida . . . 60
5.3. Módulo de Cálculo del Posicionamiento . . . 60
5.3.1. Submódulo de histórico de posiciones . . . 62
5.3.2. Factoría de métodos de *tracking* . . . 62
5.4. Módulo de *tracking* con marcas . . . 63
5.4.1. Inicialización del sistema de *tracking* . . . 64
5.4.2. Detección de una marca . . . 65
5.4.3. Obtención del posicionamiento . . . 66
5.5. Módulo de *tracking* con descriptores visuales . . . 67
5.5.1. Implementación de los descriptores visuales . . . 70
5.5.2. Almacenamiento en la base de datos . . . 70
5.5.3. Estructura de datos para agilizar la búsqueda . . . 72
5.5.4. Obtención de la posición y la orientación . . . 74
5.6. Módulo de *tracking* estéreo . . . 74
5.6.1. Extracción de parámetros intrínsecos . . . 75
5.6.2. Extracción de *landmarks* y *optical flow* . . . 76
5.6.3. Cálculo de la matriz fundamental y la matriz esencial . . . 77
5.6.4. Extraer matriz de rotación y vector de traslación . . . 78
5.6.5. Triangulación y proyección de los *landmarks* . . . 80

6. Evolución y costes **83**
6.1. Evolución del proyecto . . . 83
6.1.1. Concepto del software . . . 83
6.1.2. Análisis de requisitos . . . 84
6.1.3. Iteración 1 . . . 85
6.1.4. Iteración 2 . . . 85
6.1.5. Iteración 3 . . . 87
6.1.6. Iteración 4 . . . 88
6.1.7. Iteración 5 . . . 88
6.2. Recursos y costes . . . 89

7. Resultados **91**
7.1. Pruebas de *tracking* con marcas . . . 91
7.1.1. Pruebas en entorno virtual . . . 92
7.1.2. Pruebas en entorno real . . . 96
7.2. Pruebas de *tracking* estéreo . . . 98
7.2.1. Pruebas en entorno virtual . . . 98
7.2.2. Pruebas en entorno real . . . 101
7.3. Pruebas posicionamiento con descriptores visuales . . . 102
7.3.1. Pruebas en entorno virtual . . . 103
7.3.2. Pruebas en entorno real . . . 126
7.4. *Profiling* . . . 128
7.4.1. *Profiling* del módulo de *tracking* con marcas . . . 128
7.4.2. *Profiling* del módulo de *tracking* estéreo . . . 129
7.4.3. *Profiling* del módulo de *tracking* con descriptores . . . 129
7.4.4. *Profiling* con integración de los módulos . . . 130

8. Conclusiones y propuestas **133**
8.1. Objetivos alcanzados . . . 133
8.2. Propuestas de trabajo futuro . . . 134
8.3. Conclusión personal . . . 137

Anexos **139**

A. Manual de usuario para la calibración de la cámara **139**
A.1. Preparativos . . . 139
A.2. Ejecución de la calibración . . . 140
A.3. Ejemplo de ejecución . . . 142

B. Diagramas de clases **145**

C. Imágenes de las pruebas en entorno real **149**

D. Código fuente **153**

E. GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007 **155**

Bibliografía **163**

Índice de figuras

2.1. Mapa conceptual de los contenidos del capítulo 8
2.2. Transformaciones concatenadas sobre un objeto. A) Primero se ha realizado una rotación en el eje Z, seguido de un escalado en el eje X. B) En primer lugar se ha realizado el escalado en el eje X, seguido de la rotación en el eje Z 15
2.3. Vista general del proceso SLAM. 17
2.4. Ejemplo de reconstrucción de un monumento utilizando el software Photo Tourism. Imagen original: Noah Snavely 18
2.5. Ejemplo de *optical flow* en una imagen. Las flechas indican el desplazamiento que han tenido los *landmarks* con respecto a la imagen anterior. Imagen original: David Stavens. 21
2.6. Parámetros intrínsecos de una cámara. 22
2.7. Transformación geométrica dada por los parámetros extrínsecos entre la posición inicial y final de una cámara. Imagen: *Learning OpenCV* [19] 24
2.8. Clasificación de las diferentes técnicas de *tracking* visual 25
2.9. Ejemplos de patrones para *tracking* basado en marcas 26
2.10. Elementos básicos de la geometría epipolar. 27
2.11. Transformación de un punto P del mundo real a su proyección p en el plano de cámara utilizando una homografía. Imagen: *Learning OpenCV* [19]. . . . 32
2.12. Ejemplo de imágenes estereoscópicas rectificadas. Las líneas horizontales representan las líneas epipolares. Imagen: ejemplos de la biblioteca OpenCV-2.2. 34
2.13. Clasificación de los diferentes descriptores visuales del estándar MPEG-7. . . 36
2.14. Ejemplo de uso de LIRE Mosaics. 38

4.1. Diagrama que representa el flujo de trabajo iterativo e incremental del proyecto. 44
4.2. Ejemplo de aplicación realizada con la biblioteca ARToolkit 45

5.1. Diagramas de casos de uso del proyecto. 55
5.2. Organización de los módulos de Arges 56
5.3. Plano de la planta baja de la Escuela Superior de Informática 60
5.4. Patrones *Facade* y *Singleton* dentro del sistema de cálculo del posicionamiento. 63
5.5. Pasos para la detección los bordes de una marca en ARToolkit 65
5.6. Sistema de coordenadas de la marca con respecto de la cámara en ARToolkit. Imagen: Carlos González Morcillo. 67
5.7. Transformación de los vectores de coeficientes en cadenas de texto 71
5.8. Tabla *Image* del esquema de base de datos 71

5.9. Ejemplo de construcción de un árbol de Voronoi 73
5.10. Puntos característicos extraidos con el algoritmo Shi-Tomasi 77
5.11. Proyección de un punto en el espacio a partir de dos vistas esteroscópicas del mismo. Imagen: *Learning OpenCV* [19]. 80
5.12. Triangulación de un punto 3D con cámaras mal alineadas 82

7.1. Patrones de marca utilizados en las pruebas 92
7.2. Vistas de las cámaras para las pruebas de *tracking* estéreo. 98
7.3. Error en posición y orientación con 4 posiciones empleando únicamente *Color Layout* 105
7.4. Error en posición y orientación con 16 posiciones empleando únicamente *Color Layout* 105
7.5. Error en posición y orientación con 64 posiciones empleando únicamente *Color Layout* 108
7.6. Error en posición y orientación con 4 posiciones empleando únicamente *Edge Histogram* 109
7.7. Error en posición y orientación con 16 posiciones empleando únicamente *Edge Histogram* 110
7.8. Error en posición y orientación con 64 posiciones empleando únicamente *Edge Histogram* 112
7.9. Error en posición y orientación con 4 posiciones empleando únicamente *Scalable Color* 113
7.10. Error en posición y orientación con 16 posiciones empleando únicamente *Scalable Color* 114
7.11. Error en posición y orientación con 64 posiciones empleando únicamente *Scalable Color* 116
7.12. Error en posición y orientación con 64 posiciones; *Color Layout*(0.4), *Edge Histogram*(0.1), *Scalable Color*(0.5) 118
7.13. Error en posición y orientación con 64 posiciones; *Color Layout*(0.3), *Edge Histogram*(0.2), *Scalable Color*(0.5) 120
7.14. Error en posición y orientación con 64 posiciones; *Color Layout*(0.3), *Edge Histogram*(0.1), *Scalable Color*(0.6) 121
7.15. Comparación de imágenes para prueba 1 del *tracking* con descriptores en escenario real 127
7.16. Resultados *profiling* del *tracking* con marcas 128
7.17. Resultados *profiling* del *tracking* estéreo 129
7.18. Resultados *profiling* del *tracking* con descriptores 130
7.19. Resultados *profiling* con integración de los tres módulos de *tracking* 131

A.1. Ejemplo de archivo XML para la aplicación 141
A.2. Capturas realizadas para la calibración 142
A.3. Detección de esquinas en el patrón de calibración 143

B.1. Diagrama de clases para el sistema ARGES 146
B.2. Diagrama de clases para el módulo de *tracking* con descriptores 147

C.1. Imágenes utilizadas en las pruebas de *tracking* con marcas en entorno real. . . 149
C.2. Imágenes utilizadas en las pruebas de *tracking* estéreo en entorno real. 150
C.3. Imágenes utilizadas en las pruebas de *tracking* con descriptores en entorno real. 151

Índice de tablas

6.1. Desglose del coste económico de Arges. 89

7.1. Resultados para el *tracking* con marcas en entorno virtual. Patrón 1 visto de frente. 93
7.2. Resultados para el *tracking* con marcas en escenario virtual. Patrón 2 visto de frente. 94
7.3. Resultados para el *tracking* con marcas en escenario virtual. Patrón 3 visto de frente. 95
7.4. Resultados para el *tracking* con marcas en escenario virtual. Patrón 1 visto desde diferentes ángulos. 96
7.5. Resultados para el *tracking* con marcas en escenario virtual. Patrón 3 visto desde diferentes ángulos. 97
7.6. Resultados para el *tracking* con marcas en escenario real. 97
7.7. Resultados obtenidos en la triangulación 3D usando visión estereoscópica en escenario virtual 101
7.8. Resultados para el *tracking* estéreo en escenario real. 102
7.9. Resultados con 4 posiciones empleando únicamente *Color Layout* 104
7.10. Resultados con 16 posiciones empleando únicamente *Color Layout* 106
7.11. Resultados con 64 posiciones empleando únicamente *Color Layout* 107
7.12. Resultados con 4 posiciones empleando únicamente *Edge Histogram* 109
7.13. Resultados con 16 posiciones empleando únicamente *Edge Histogram* 110
7.14. Resultados con 64 posiciones empleando únicamente *Edge Histogram* 111
7.15. Resultados con 4 posiciones empleando únicamente *Scalable Color* 113
7.16. Resultados con 16 posiciones empleando únicamente *Scalable Color* 114
7.17. Resultados con 64 posiciones empleando únicamente *Scalable Color* 115
7.18. Resultados con 64 posiciones; *Color Layout*(0.4), *Edge Histogram*(0.1), *Scalable Color*(0.5) 118
7.19. Resultados con 64 posiciones; *Color Layout*(0.3), *Edge Histogram*(0.2), *Scalable Color*(0.5) 119
7.20. Resultados con 64 posiciones; *Color Layout*(0.3), *Edge Histogram*(0.1), *Scalable Color*(0.6) 121
7.21. Comparativa entre las imágenes reales y las imágenes estimadas por el sistema de *tracking* con descriptores para la prueba 2 125
7.22. Resultados para el *tracking* con descriptores en escenario real sin histórico. . 126
7.23. Resultados para el *tracking* con descriptores en escenario real con histórico. . 127

Lista de algoritmos

1. Asociación de datos en SLAM 19
2. Algoritmo de calibración de Arges 58
3. Algoritmo de cálculo de la posición del usuario. 61
4. Algoritmo de indexación de imágenes en base de datos. 69
5. Algoritmo general de *tracking* con descriptores visuales. 69
6. Algoritmo de *tracking* estéreo. 75

Listado de acrónimos

2D	Dos Dimensiones
3D	Tres Dimensiones
CAD	*Computer-aided design*
DDL	*Description Definition Language*
FKE	Filtro de Kalman Extendido
GIS	*Geographycal Information System*
GPS	*Global Positioning System*
HSV	*Hue, Saturation, Value*
LIRE	*Lucene Image REtrieval*
MDS	*Multimedia Description Squemes*
RA	Realidad Aumentada
SIFT	*Scale-Invariant Feature Transform*
SLAM	*Simultaneous Location and Mapping*
SQL	*Structured Query Language*
SURF	*Speeded Up Robust Features*
SVD	*Singular Value Decomposition*
VC	Visión por Computador

VE Visión Estereoscópica

XML *Extensible Markup Language*

YAML *YAML Ain't Markup Language*

Capítulo 1

Introducción

Desde su implantación para uso civil en 1995, el *Global Positioning System* (GPS) o Sistema de Posicionamiento Global, ha ido masificando su uso hasta el punto de integrarse dentro de los llamados *smartphones*. En la actualidad, los sistemas GPS vienen integrados incluso en los sistemas de control de vehículos, lo que da cuenta de la importancia que se le da al conocer el lugar de emplazamiento del usuario.

A pesar de su amplia utilización, una de las más importantes limitaciones de este sistema es su incapacidad funcional dentro de casas y edificios. La recepción de las señales satélite es nula en un lugar cerrado. Por otra parte, la precisión obtenida en receptores para aplicaciones civiles de GPS (entre 1 y 5 metros de error), no sería suficiente para algunas aplicaciones de localización en interiores.

Partiendo de esta problématica, se propone el presente proyecto de fin de carrera, titulado ARGES (*Augmented Reality techniques for a Geometric Emplacement System*), el cual constituye un sistema de posicionamiento en interiores cuya base tecnológica es la Visión por Computador y las técnicas de *tracking* empleadas en sistemas de RA.

Como curiosidad, mencionar que Arges, el acrónimo del proyecto, también es el nombre que recibía uno de los cíclopes, hijos de Urano y Gea, según la mitología griega. El símil entre el ojo del cíclope y la cámara de vídeo única utilizada en el sistema inspiraron este nombre.

1.1. Visión por computador y Realidad Aumentada

La **Visión por Computador (VC)** o **Visión Artificial**, constituye uno de los campos tecnológicos más estudiados y desarrollados en los últimos años. El objetivo de la VC es el estudio y desarrollo de interpretar la salida de determinados sensores de visión, en la mayoría de las ocasiones cámaras de vídeo.

Gracias a la implantación de cámaras en dispositivos cotidianos como los teléfonos móviles, cada vez se abren posibilidades mayores para la VC. El incremento exponencial del ámbito de aplicación de técnicas de visión artificial está claramente relacionado con el incremento en la capacidad de cómputo de los dispositivos portátiles.

En la actualidad, las técnicas de VC se utilizan en multitud de entornos industriales. El reconocimiento facial, seguimiento de objetos en tareas de vídeo-vigilancia, búsqueda de imágenes a través de características visuales o el mapeo de una escena para generar un modelo 3D son algunos ejemplos de exitosa implantación comercial.

El campo de la RA aprovecha los avances realizados por la VC, realizando lo que se conoce como *Registro de la realidad*. Según Azuma [1], este registro de la realidad comprende la captación de las características del entorno y su posterior transformación con el fin de conocer:

- La posición y orientación del usuario.
- La posición de un objeto en la escena.
- Las características de un objeto identificado en la escena.

La estimación de la posición y orientación de la cámara es lo que se conoce como *tracking*.

Arges implementa una serie de métodos de *tracking* para tener una estimación lo más precisa posible del posicionamiento empleando diversos elementos de la escena que son percibidos por la cámara. Posteriormente se dará información más detallada de los diferentes métodos de *tracking* existentes.

1.2. Impacto socio-económico

Desde sus comienzos, la VC ha estado orientada a su utilización en el campo de la robótica, de tal forma que se pudiera dotar a los robots de una mayor independencia motriz conociendo el escenario en el que interaccionan. Sin embargo, en los últimos años, se está dando mucha importancia a las investigaciones relacionadas con la implantación de dispositivos[1] de VC en humanos con carencias en la visión [2].

En el mundo industrial, cada vez son más habituales las inversiones en VC para el ahorro de costes de producción. Para cumplir este objetivo, la VC proporciona una mejora en los controles de calidad, por ejemplo en las industrias de conservas, controlando fallos en el cierre de las latas o en cualquier proceso de producción, detectando defectos en las piezas producidas. Evitando la producción de piezas defectuosas, el periodo de tiempo de amortización de la inversión en VC es muy corto, ya que se reducen costes tanto en salarios como en costes de devolución de piezas.

Aunque el desarrollo de la VC en robots se ha dedicado casi por completo a la industria, el mundo del ocio cada vez encuentra nuevas formas de innovación en este campo. En el mundo de los videojuegos se ha ampliado el antiguo paradigma del jugador delante de la televisión con el mando de la videoconsola en la mano. Gracias a la VC se prescinde del mando y se emplea el propio cuerpo como ejecutor de las acciones del videojuego. Un ejemplo llamativo es *Microsoft Kinect*, un dispositivo de cámara con sensor de profundidad desarrollado para la videoconsola *Xbox360*, el cual vendió alrededor de 10 millones de unidades entre noviembre de 2010 y enero de 2011.

El presente proyecto se enmarca dentro de la Cátedra Indra-UCLM, en el proyecto *"ElCano: Multimodal Indoor Navigation Infrastructure for Disabled People"*[2]. El objetivo principal es el desarrollo de diversas técnicas de posicionamiento visual de forma que se obtenga una sinergia de dichos métodos en un sistema GIS (Geographical Information System) común.

[1] Se suelen utilizar microchips de silicio capaces de ejercer tareas fotorreceptoras. Otros sistemas más avanzados utilizan matrices de electrodos de platino en el córtex visual en ambos hemisferios del cerebro [3].

[2] Para más información sobre el proyecto ElCano: http://catedraindra.uclm.es/elcano

Existen unos 38 millones de discapacitados en Europa, por lo que, en este sentido, la presión que ejerce la Unión Europea para fomentar la independencia de las personas discapacitadas hace preveer regulaciones y directrices que fomenten la implantación de este tipo de sistemas en instalaciones como las mencionadas anteriormente [5].

La ausencia de una tecnología o estándar dominantes en cuanto a sistemas de posicionamiento en interiores hace que este proyecto pueda resultar interesante de cara a todo tipo de aplicaciones que requieran del servicio que ofrece (localización de personal, aplicaciones de seguridad, etc.). Estas aplicaciones pueden verse implementadas en todo tipo de edificios de administraciones públicas, donde las personas discapacitadas son susceptibles de requerir de sus servicios (hospitales, juzgados, comisarías, ...), empresas de gestión de infraestructuras de transporte (aeropuestos, estaciones de autobuses o trenes, ...), así como, en general, empresas que pongan sus servicios a disposición del público por medio de grandes infraestructuras (estadios, museos, hipermercados, ...).

1.3. Justificación y problemática

Desarrollar un sistema de posicionamiento dentro de un edificio requiere que los resultados obtenidos sean de gran precisión. En el caso general de proporcionar servicios de información basados en la posición del usuario, un error de un metro puede ser admisible. En el caso de proporcionar interfaces avanzadas basadas en el paradigma de RA, errores superiores a 2cm causan que el registro no se realice adecuadamente.

Existen diversas técnicas de *tracking* y métodos de apoyo al posicionamiento visual. Inicialmente el proyecto ElCano se planteó como un sistema multimodal que integra diversas técnicas de posicionamiento en interiores (tales como RFID, Wifi o métodos basados en VC).

El *tracking* mediante marcas visuales es un método ampliamente utilizado por su versatilidad y limitadas necesidades computacionales. Las marcas ofrecen un método de posicionamiento de 6 grados de libertad muy preciso en distancias cortas. Una marca de 12 cm de lado puede ser percibida correctamente, con cámaras de 1 MegaPíxel, a distancias inferiores a 2 metros. Estas técnicas no funcionan correctamente cuando el espacio de trabajo aumenta (como en el caso de posicionamiento en el interior de edificios) o varían las condiciones de

explotación (iluminación, oclusiones parciales, entornos no controlados,...).

De este modo, el proyecto Arges se plantea con el propósito de integrar diversos métodos de *tracking* visual para su incorporación al proyecto ElCano. Además del método de *tracking* basado en marcas, se han incluido dos tipos de métodos de posicionamiento sin marcas; empleando geometría epipolar y mediante descriptores de imagen. El proyecto se encargará de realizar una primera integración de las medidas obtenidas con cada método.

A continuación se describen las secciones de primer nivel que forman el presente documento.

1.4. Estructura del documento

Este documento ha sido redactado respetando la normativa de proyectos de fin de carrera de la Escuela Superior de Informática (UCLM) [6]. El contenido está dividido en los siguientes capítulos:

- **Capítulo 2: Antecedentes**

  En este capítulo se mostrarán los sistemas existentes en materia de posicionamiento y se explicarán los aspectos que han tenido que ser estudiados para poder llevar a cabo este proyecto de fin de carrera.

- **Capítulo 3: Objetivos**

  En este capítulo se explican los objetivos concretos que se persiguen con la realización de este proyecto.

- **Capítulo 4: Metodología de trabajo**

  En este capítulo se describe la metodología que se ha seguido para el desarrollo del proyecto, así como las herramientas software y dispositivos hardware empleados.

- **Capítulo 5: Arquitectura**

  En este capítulo se explica detalladamente el diseño e implementación de cada uno de los módulos del sistema, así como la integración de los mismos.

- **Capítulo 6: Evolución y costes**

  En este capítulo se exponen las etapas por las que ha pasado el desarrollo el proyecto. Además se muestra información del coste económico del proyecto.

- **Capítulo 7: Resultados**

  En este capítulo se muestran las diferentes pruebas aplicadas a cada uno de los módulos que componen el sistema y los resultados obtenidos.

- **Capítulo 8: Conclusiones y propuestas**

  En este capítulo se muestra un análisis del trabajo realizado y se describen posibles líneas de trabajo futuro sobre el propio proyecto.

Capítulo 2

Antecedentes

En este capítulo se realizará un resumen de las principales áreas de conocimiento y conceptos estudiados y en qué medida han intervenido en el desarrollo del proyecto. Debido a la falta de sistemas presentes en el mercado con las características de Arges, el estudio del estado del arte se realizará atendiendo a aquellos aspectos funcionales que tengan presencia en el sistema.

La figura 2.1 muestra un esquema con los contenidos que se desarrollarán a lo largo de este capítulo. La sección de **Informática Gráfica** dará una panorámica general de la base matemática sobre la que se sustenta Arges; en la sección de **Odometría** se explicarán las características de los sistemas odométricos y su aplicación para posicionamiento visual; en la sección de **Visión por Computador** se verán algunos fundamentos ópticos que intervienen en los diferentes sistemas que utilizan esta tecnología, así como los diferentes tipos de *tracking* existentes; en la sección de **Visión Estereoscópica** se verán los fundamentos matemáticos y geométricos que hacen posible la implementación de sistemas esteroscópicos artificiales; por último, en el apartado de **Metadatos**, se verá una panorámica general del estándar de metadatos multimedia MPEG-7 y se mostrarán diferentes implementaciones de descriptores visuales.

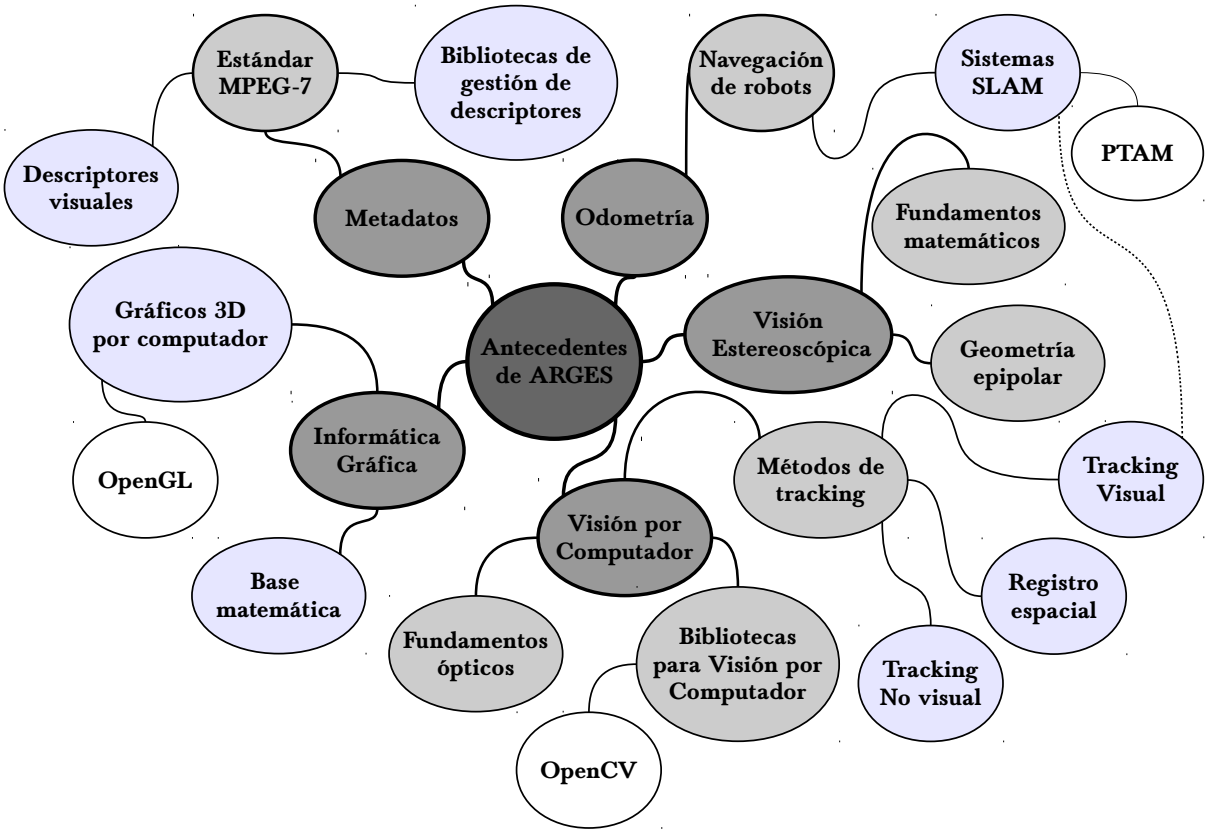


Figura 2.1: Mapa conceptual de los contenidos del capítulo

2.1. Informática Gráfica

2.1.1. Base matemática

Para poder comprender cómo aplicar la informática gráfica en el mundo de la VC es imprescindible conocer una serie de conceptos básicos relacionados con la geometría espacial. Puesto que se va a trabajar con la posición y la rotación de los elementos del mundo, será necesario revisar algunos conceptos relacionados con el espacio euclídeo de tres dimensiones, $\Re^3$.

2.1.1.1. Vectores y puntos

Un **vector** es una magnitud que tiene una longitud (también llamada módulo) y una dirección, y que se representa gráficamente como un segmento con una punta de flecha en uno de sus extremos. Generalmente se notan asignándoles una letra con un segmento o flecha en la parte superior ($\vec{v}$). Cuando un vector tiene módulo 1, se le denomina vector unitario e implica que es un vector normalizado (denotado como $\hat{v}$).

Los vectores en 3D son susceptibles de ser expresados como una tupla de 3 números reales, lo que facilita su representación y uso en el ordenador:

$$\vec{v} = (v_1, v_2, v_3) \tag{2.1}$$

El módulo del vector $\vec{v}$, que se nota como $|\vec{v}|$ es:

$$|\vec{v}| = \sqrt{v_1^2 + v_2^2 + v_3^2} \tag{2.2}$$

A continuación se mostrarán las operaciones básicas susceptibles de realizarse con vectores:

- **Suma de vectores.** Sean $\vec{u} = (u_1, u_2, u_3)$ y $\vec{v} = (v_1, v_2, v_3)$ vectores en $\Re^3$, la suma de ambos es:

$$\vec{u} + \vec{v} = (u_1 + v_1, u_2 + v_2, u_3 + v_3) \tag{2.3}$$

- **Producto de un vector por un escalar.** Sea un número real k y un vector $\vec{v} = (v_1, v_2, v_3)$, el producto escalar de ambos consiste en el producto de k por cada componente del vector:

$$k\vec{v} = (kv_1, kv_2, kv_3) \tag{2.4}$$

- **Producto escalar entre vectores.** Sean $\vec{u} = (u_1, u_2, u_3)$ y $\vec{v} = (v_1, v_2, v_3)$ vectores en $\Re^3$, el producto escalar de ambos es:

$$\vec{u} \cdot \vec{v} = (u_1 v_1 + u_2 v_2 + u_3 v_3) \tag{2.5}$$

Otra forma de calcularlo sería con la siguiente ecuación:

$$\vec{u} \cdot \vec{v} = |\vec{u}|\,|\vec{v}| \cos(\alpha) \tag{2.6}$$

Siendo α el ángulo formado entre los dos vectores.

Se dice que $\vec{u}$ y $\vec{v}$ son *perpendiculares* si el producto vectorial de ambos es cero.

- **Producto vectorial entre vectores.** Sean $\vec{u} = (u_1, u_2, u_3)$ y $\vec{v} = (v_1, v_2, v_3)$ vectores en $\Re^3$, su producto vectorial, denotado como $\vec{u} \times \vec{v}$ es:

$$\vec{u} \times \vec{v} = (u_2v_3 - u_3v_2, u_1v_3 - u_3v_1, u_1v_2 - u_2v_1) \tag{2.7}$$

  El resultado del producto vectorial de dos vectores es un tercer vector perpendicular a ambos.

Un **punto** es una magnitud que tiene una localización pero carece de módulo o longitud.

Sean dos puntos p y q, la distancia entre ambos (denotada como $\overline{pq}$) es igual al módulo del vector que los une. Dicho vector es calculado restando las componentes del segundo vector y del primero:

$$\overline{pq} = \left|\overrightarrow{q - p}\right| \tag{2.8}$$

Generalmente, al trabajar con gráficos por computador, la representación de vectores y puntos se realiza según un *sistema homogéneo* de representación. Para ello, se añade una cuarta coordenada w a la tupla que representa el punto ($w = 1$) o el vector ($w = 0$) en cuestión. De esta forma, un punto p se representaría como $(p_x, p_y, p_z, 1)$ y un vector $\vec{v}$ como $(v_x, v_y, v_z, 0)$.

Usando esta representación estamos simplificando el tratamiento de estos elementos geométricos, haciendo que las transformaciones sobre los mismos se realicen únicamente con productos.

2.1.1.2. Matrices

Una matriz es una tabla de elementos distribuidos en *m* filas y *n* columnas de la siguiente forma:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

A continuación se presentan las operaciones más comunes que podemos realizar con matrices:

- **Suma de matrices.** Sean A y B dos matrices de igual dimensión:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{pmatrix}$$

  Se define la suma de A y B como:

$$A + B = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \cdots & a_{1n} + b_{1n} \\ a_{21} + b_{21} & a_{22} + b_{22} & \cdots & a_{2n} + b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} + b_{m1} & a_{m2} + b_{m2} & \cdots & a_{mn} + b_{mn} \end{pmatrix} \tag{2.9}$$

- **Producto de una matriz por un escalar.** Sea la matriz A, definida anteriormente y un número real k, el producto de ambos (denotado como $k \cdot A$ es:

$$k \cdot A = \begin{pmatrix} ka_{11} & ka_{12} & \cdots & ka_{1n} \\ ka_{21} & ka_{22} & \cdots & ka_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ ka_{m1} & ka_{m2} & \cdots & ka_{mn} \end{pmatrix} \tag{2.10}$$

- **Producto de dos matrices.** Sean las matrices A y B, tales que el número de columnas de A es igual al número de filas de la matriz B:

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} B = \begin{pmatrix} b_{11} & \cdots & b_{1p} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{np} \end{pmatrix}$$

  El producto matricial de ambas (denotado como $A \cdot B$, $A \times B$ o, simplemente AB) es:

$$AB = \begin{pmatrix} a_{11}b_{11} + \cdots + a_{1n}b_{n1} & \cdots & a_{11}b_{1p} + \cdots + a_{1n}b_{np} \\ \vdots & \vdots & \vdots \\ a_{m1}b_{11} + \cdots + a_{mn}b_{n1} & \cdots & a_{m1}b_{1p} + \cdots + a_{mn}b_{np} \end{pmatrix} \tag{2.11}$$

- **Inversa de una matriz.** En gráficos por computador es muy usual la utilización de la inversa de una matriz. Dada una matriz A, su inversa (denotada como A^{-1}) es:

$$A^{-1} = \frac{1}{|A|} Adj(A^t) \tag{2.12}$$

  Siendo $|A|$ el determinante de la matriz, A^t la traspuesta de la matriz y $Adj()$ su matriz de adjuntos.

  La principal característica de la matriz inversa es que el producto de una matriz por su inversa es igual a la matriz identidad I.

- **Transformada SVD de una matriz.** La transformada SVD o *Singular Value Decomposition* se utiliza como herramienta para la resolución de sistemas de ecuaciones. Dada una matriz A, dicha matriz puede ser descompuesta en un producto de tres matrices con la siguiente forma:

$$A = UDV^T \tag{2.13}$$

  donde U y V son matrices ortogonales[1] y D es una matriz diagonal.

[1] Las matrices ortogonales son aquellas en las que la inversa de la matriz coincide con su traspuesta.

Más adelante se verá la utilidad de la transformada SVD. Para más información acerca de cómo realizar la descomposición ver [24].

2.1.1.3. Transformaciones en el espacio utilizando matrices

Cuando se trabaja con gráficos por computador, la orientación y la posición de los elementos representados están indicadas utilizando matrices. Estas matrices suelen ser de 4x4 al estar trabajando con un sistema homogéneo. Realizando productos de matrices, podemos aplicar transformaciones sobre objetos en el espacio. Las transformaciones básicas que podemos realizar son la traslación, la rotación y el escalado.

- **Traslación**

  Se define la traslación como el cambio de un objeto de una posición a otra. Esta traslación está representada por un vector de traslación $\vec{t} = (t_x, t_y, t_z)$.

  Este vector está contenido dentro de la matriz de traslación que tiene la siguiente forma:

$$T = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.14}$$

- **Rotación**

  Se definen las matrices de rotación para poder rotar un objeto en uno de los ejes de coordenadas. Al haber tres ejes de coordenadas en $\Re^3$, tenemos tres matrices de rotación:

$$R_x(\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.15}$$

$$R_y(\phi) = \begin{pmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 1 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.16}$$

$$R_z(\phi) = \begin{pmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.17}$$

Siendo ϕ el ángulo de rotación sobre el que queremos girar el objeto.

- **Escalado**

  El escalado es la transformación que se aplica a los objetos para aumentar o disminuir su tamaño. La matriz que se aplica para ello es la siguiente:

$$S = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.18}$$

- **Concatenación de transformaciones**

  La concatenación de transformaciones se realiza aplicando el producto de las matrices que representan dichas transformaciones para obtener lo que se conoce como *matriz compuesta* (C). Cuando aplicamos una serie de transformaciones sobre un elemento, es importante tener en cuenta el orden de aplicación de las matrices de transformación, ya que el producto de matrices no es conmutativo.

La concatenación de transformaciones se realiza multiplicando en primer lugar aquellas que se desean aplicar al final. De otra forma pueden obtenerse resultados indeseados.

En la figura 2.2 puede verse la diferencia de resultados cuando aplicamos las transformaciones en un orden u otro.

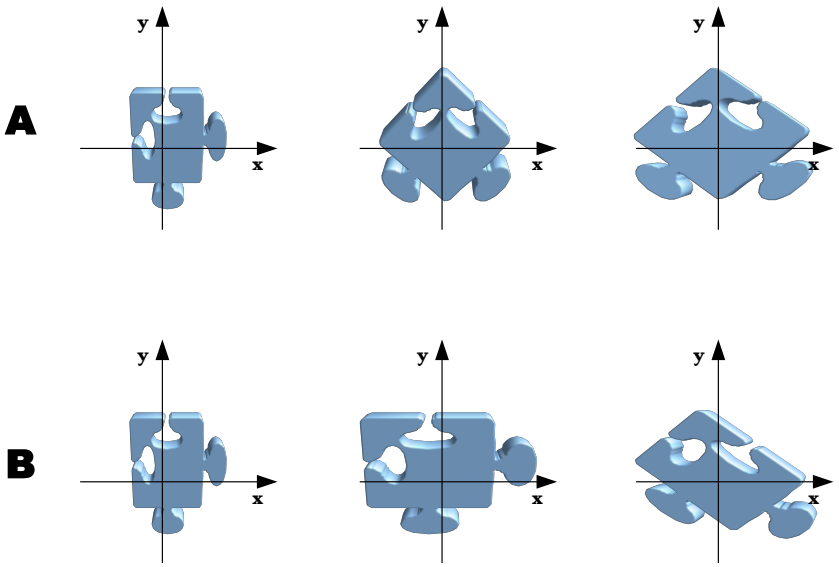


Figura 2.2: Transformaciones concatenadas sobre un objeto. A) Primero se ha realizado una rotación en el eje Z, seguido de un escalado en el eje X. B) En primer lugar se ha realizado el escalado en el eje X, seguido de la rotación en el eje Z

2.2. Odometría

Los sistemas de posicionamiento en interiores tienen su origen en el desarrollo de sistemas de odometría para robots. La odometría centra su atención en cómo llevar a cabo la estimación de la posición de vehículos con ruedas dentro de un escenario determinado. Aunque los sistemas odométricos funcionan con cualquier tipo de sensor que les proporcione información útil para su funcionamiento, para el desarrollo de este proyecto se ha hecho especial hincapié en el estudio de sistemas de *odometría visual*.

La **odometría visual** se define como *"el proceso mediante el cual se determina la posición y la orientación de una cámara o de un sistema de cámaras mediante el análisis de una secuencia de imágenes adquiridas, sin ningún conocimiento previo del entorno. Como el movimiento del sistema de cámaras es solidario con el movimiento del vehículo, a esta técnica también se le conoce como ego-motion (movimiento propio)"* [7].

El proceso odométrico entraña la dificultad de cómo utilizar los datos recibidos por los sensores del robot (en este caso la cámara) para estimar la construcción del mapa del entorno. Esto se conoce como *Simultaneous Location and Mapping (SLAM)*.

2.2.1. SLAM

SLAM fue desarrollado originalmente por Hugh Durrant-Whyte y John J. Leonard [8], basándose en los trabajos anteriores de Smith, Self y Cheeseman [9]. SLAM se incluye dentro del campo de la robótica conocido como *Robótica probabilística*. Consiste en la utilización de técnicas empleadas en odometría visual para la construcción de un mapa del entorno de tal forma que mediante este mapa y a través de algoritmos estadísticos, el sistema pueda inferir la posición de la cámara.

Todo sistema SLAM se compone generalmente de una serie de partes: extracción de *landmarks*, asociación de datos, estimación del estado, actualización del estado y actualización de *landmarks* (ver figura 2.3).

En las siguientes secciones se explicará en qué consiste cada una de las partes que componen SLAM.

2.2.1.1. *Landmarks*

Los ***landmarks***[2] son aquellos puntos de referencia que tomará el sistema para llevar a cabo el mapeo del sistema. Estos puntos son obtenidos mediante los sensores específicos para dicho proceso, y en el caso de la odometría visual serían las esquinas de los objetos o los

[2]Una traducción aproximada de la palabra *landmark* en castellano podría ser "punto característico" o "punto clave". Ambas traducciones serán utilizadas indistintamente a lo largo de este documento para referirse al mismo concepto.

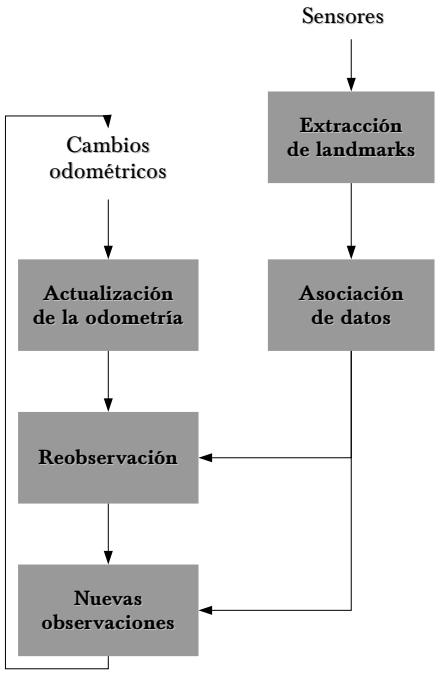


Figura 2.3: Vista general del proceso SLAM.

puntos de colisión entre ellos, los cuales definen su geometría y permiten al sistema conocer la forma de los elementos que está viendo.

Existen multitud de algoritmos para la obtención de *landmarks*, como por ejemplo SIFT (*Scale-Invariant Feature Transform*) [10]. SIFT es utilizado para extraer las características distintivas de las imágenes en escala de grises, es decir, puede utilizarse para reconocer el mismo punto característico entre diferentes vistas de un mismo objeto. Un sistema significativo que ha utilizado este tipo de método es el Photo Tourism [11]. Este sistema es capaz de reconocer las características 3D representadas en varias fotos digitales 2D de un objeto para formar la estructura 3D de dicho objeto (ver figura 2.4). Una evolución de SIFT es SURF (*Speeded Up Robust Features)* [12]), el cual permite detectar un mayor número de puntos que SIFT, procesarlos más rápidamente y de forma más robusta.

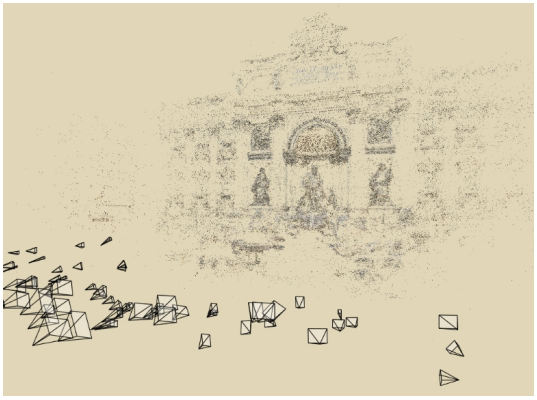


Figura 2.4: Ejemplo de reconstrucción de un monumento utilizando el software Photo Tourism. Imagen original: Noah Snavely

Más orientados a la búsqueda de esquinas en la geometría de los objetos de una imagen se encuentran el detector Harris o una evolución del mismo: Shi-Tomasi [13].

El algoritmo de **Shi-Tomasi** ha sido el utilizado en Arges para la búsqueda de los puntos característicos de la escena al ser más robusto frente a rotaciones que el detector Harris.

Una vez se tienen extraidos los *landmarks* de la escena, es necesario realizar una asociación entre ellos para establecer cuáles son nuevos y cuáles han sido reobservados. Esto es lo que se conoce como *Matching*.

2.2.1.2. Asociación de los datos *(Matching)*

Los *landmarks* son obtenidos en cada pasada del algoritmo SLAM. De esta forma, cada vez se obtiene, por una parte puntos que no habían sido vistos previamente, y por otra parte puntos que han sido vistos ya y estén registrados en el mapa del entorno, pero cuya posición no va a ser exactamente la misma que la que se había estimado previamente. Esto último es debido a que un cambio en la posición o la rotación de la cámara hace que la estimación de la posición 3D de un punto característico tenga una pequeña variación. Sería redundante volver a guardar un punto diferente en una posición muy cercana a otro cuando se está observando el mismo punto. Por esta razón los sistemas SLAM toman una parte de su tiempo en realizar una asociación de datos.

Esta asociación de datos, también conocida como *matching*, trata de estimar cuándo hay que añadir un nuevo *landmark* al conjunto de los almacenados o cuándo se establece que se ha visualizado un *landmark* ya existente y no es necesario almacenarlo. El algoritmo 1 explica cómo se realiza esta tarea[3]:

Algoritmo 1 Asociación de datos en SLAM

1: Se extraen los *landmarks* visibles de la escena y se calcula su posición 3D para el mapa.
2: Se realiza una asociación de los *landmarks*. Cada uno se asocia al más cercano a su posición de los que ya se encuentran almacenados en la base de datos y que han sido re-observados N veces.
3: Cada par de *landmarks* asociados se pasan por un *filtro de validación*:
4: **si** el par pasa el filtro de validación **entonces**
5: El *landmark* observado se considera el mismo que el ya almacenado en la base de datos y se incrementa el número de veces que éste ha sido re-observado.
6: **si no**
el *landmark* observado se considera uno diferente de los que ya se tienen en la base de datos, por tanto se almacena como uno nuevo y su contador de observaciones se pone a 1.
7: **fin si**

Para calcular la separación entre los puntos característicos, normalmente se utiliza como medida la distancia euclídea, por ser sencilla y rápida de calcular. Otros métodos incluyen el cálculo de la llamada distancia de *Mahalanobis*, la cuál es más exacta pero de un cálculo más complejo. Los filtros de validación que se utilizan para realizar estos algoritmos se basan en implementaciones de métodos estadísticos como es el *Filtro de Kalman Extendido* o FKE [14].

2.2.1.3. *Optical flow*

El *matching* de puntos característicos es utilizado normalmente cuando el contexto se considera como "estático", de tal forma que los pasos que se siguen son:

[3]Este algoritmo suele recibir el nombre de *Aproximación al vecino más cercano*

1. Extracción de *landmarks* en la imagen *n*.

2. Extracción de *landmarks* en la imagen *n+1*.

3. Asociación entre los *landmarks* extraidos de ambas imágenes.

El principal problema de realizar estos pasos es que el tiempo de computación requerido para la extracción de puntos en una imagen es demasiado alto como para aplicarlo en cada uno de los frames si se está trabajando con un sistema de tiempo real. Por esta razón, se utilizan técnicas de ***Optical flow***. Los algoritmos de *optical flow* se encargan de buscar la situación de puntos ya conocidos en un frame posterior al que se utilizó para extraer los puntos, puesto que se supone que contendrá un desplazamiento muy pequeño con respecto al anterior y la geometría que ven ambos es muy similar.

A través de los algoritmos de *optical flow*, se trabaja sobre un contexto "dinámico", en el que un conjunto de puntos característicos van evolucionando a través de los frames. En este contexto, los pasos a seguir serían:

1. Extracción de *landmarks* en la imagen *n*.

2. Búsqueda de la posición de los *landmarks* de la imagen *n* en la imagen *n+1*.

Se puede ir realizando el seguimiento de un conjunto de puntos a través de diversos frames hasta considerar que la mayoría de los puntos con los que se contaban al principio ya no son visibles en la actual posición de la cámara (ver figura 2.5). Llegados a ese punto, se volvería a realizar una nueva búsqueda de puntos. De esta forma, se puede constituir un sistema de VC con estructura en movimiento en tiempo real [18].

Existen diferentes algoritmos que intentan resolver el problema del *optical flow*. Algunos de los más utilizados son:

- **Algoritmo de *Camus*:** es un algoritmo simple basado en búsqueda por "vecindad" a lo largo de los *n* frames anteriores. Por cada posición de píxel calcula un valor que indica la medida en la que el píxel se ha movido dentro de una localicación espacio-temporal [16].

Figura 2.5: Ejemplo de *optical flow* en una imagen. Las flechas indican el desplazamiento que han tenido los *landmarks* con respecto a la imagen anterior. Imagen original: David Stavens.

- **Algoritmo de *Horn & Schunk*:** utiliza una condición global de suavidad, para lo cual se basa en los gradientes espaciales y temporal; con esto pretende resolver el "problema de la apertura" [17].

- **Algoritmo de *Lucas & Kanade*:** permite la extracción de las dos componentes de la velocidad asumiendo que el flujo es constante en una cierta vecindad local del píxel que se está considerando. Partiendo de esta premisa, resuelve las ecuaciones del *optical flow* para todos los píxels de esta vecindad. Existe la variante del algoritmo de *Lucas & Kanade* piramidal, en donde se realiza el algoritmo para diferentes resoluciones de una misma imagen tomando de todo ello los resultados más fiables [20].

Todos los aspectos relacionados con la extracción de *landmarks* de una imagen y las técnicas de *optical flow* son posibles gracias a los avances realizados en materia de VC. En la siguiente sección se mostrarán los aspectos más importantes de esta materia.

2.3. Visión por Computador

Los sistemas basados en odometría visual, tienen como base de funcionamiento la realización de un buen registro de la realidad o *tracking*. Para ello, la herramienta de la que se dispone es el análisis de la escena que se está visualizando. Por esta razón, la VC es una pieza clave en el desarrollo de estos sistemas, ya que es la que hace posible el poder obtener datos referentes al escenario que se está viendo y poder inferir cualquier dato necesario para ejecutar el *tracking*.

En las siguiente secciónes se verán algunos de los fundamentos ópticos en los que se basa la VC; por otra parte, se explicarán diferentes métodos de implementación del *tracking*.

2.3.1. Fundamentos ópticos

Un paso necesario para poder trabajar con cualquier sistema de VC, es realizar la calibración de la(s) cámara(s) que vayamos a utilizar. Calibrar una cámara consiste en obtener la medida de diferentes parámetros que definen a la cámara en cuestión, así como las medidas de distorsión de la lente. Estos parámetros de dividen en intrínsecos y extrínsecos.

Los **parámetros intrínsecos** de una cámara son aquellos que describen el funcionamiento de la cámara y que son constantes. Entre estos parámetros se encuentran la distancia focal, el centro óptico y el punto principal. En la figura 2.6 se pueden visualizar estos tres parámetros.

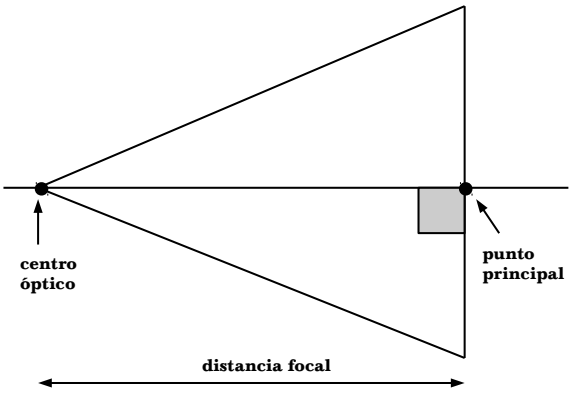


Figura 2.6: Parámetros intrínsecos de una cámara.

- El **centro óptico** o centro de proyección, denotado como $\boldsymbol{O}$, es el punto de la cámara desde el cual parten todos los rayos que son proyectados en el plano de imagen de la cámara.

- El **punto principal**, denotado como $\boldsymbol{c}$, es la proyección ortogonal del centro óptico sobre el plano de cámara.

- La **distancia focal**, denotada como $\boldsymbol{f}$, es la distancia existente entre el centro óptico y el punto principal.

Los **parámetros extrínsecos** son aquellos que definen la posición y la orientación de la cámara con respecto al mundo real.

Una vez realizada la calibración de la cámara, se obtienen una serie de matrices y vectores con valores que contienen los valores de los parámetros descritos anteriormente. En primer lugar, se obtiene la llamada **Matriz de cámara**, denotada como **M**, que contiene los valores de distancia focal y punto principal de la cámara. Esta matriz tiene la siguiente forma:

$$M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Por otra parte, se obtiene un vector con los factores de distorsión de la cámara. Normalmente este vector contiene 5 valores, correspondientes a 3 factores de distorsión radial, denotados como k_i y 2 factores de distorsión tangencial, denotados como p_j. El número de factores de distorsión radial puede verse incrementado hasta 5, dependiendo del algoritmo de calibración que se realice.

Tanto la matriz de cámara como el vector de distorsión contienen únicamente parámetros intrínsecos, lo que significa que serán constantes. No ocurre lo mismo con los parámetros extrínsecos, que varían en función de la situación de la cámara en el escenario, y que producen una matriz de rotación y un vector de traslación. Estos dos elementos indican la transformación geométrica que hay que realizar sobre la cámara para alcanzar la posición y la rotación

finales desde su posición inicial. En la figura 2.7 se puede ver gráficamente la implicación de los parámetros extrínsecos.

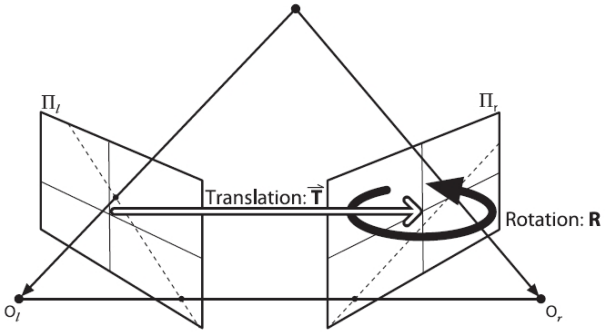


Figura 2.7: Transformación geométrica dada por los parámetros extrínsecos entre la posición inicial y final de una cámara. Imagen: *Learning OpenCV* [19]

2.3.2. *Tracking* y métodos de *tracking*

Aunque la mayoría de los sistemas actuales empleados en odometría se basan en el empleo de cámaras de vídeo como entrada de datos al sistema, existen otras alternativas. De esta forma pueden dividirse los métodos de *tracking* en dos grandes grupos: ***tracking no visual*** y ***tracking visual***.

Para el *tracking* no visual existen diferentes posibilidades. Una posibilidad sería el empleo de sensores o receptores de red *Wifi* o *Bluetooth*. Sin embargo, estos sistemas se encuentran supeditados a la presencia de emisores de la tecnología en cuestión en el propio edificio de tal forma que se garantice su cobertura en todo el área que requiera su servicio. Aunque se contemplan como alternativas al *tracking*, este tipo de fuentes de información adolecen de una gran falta de precisión, lo que los convierte en elementos poco adecuados para esta tarea.

Otras alternativas más precisas aunque costosas consisten en utilizar dispositivos hardware de medición. Los dispositivos usados para ello son giróscopos (miden la orientación de forma absoluta) y los acelerómetros (miden la aceleración del movimiento en las tres dimensiones).

Por otra parte están los métodos de *tracking* visual, en los cuales se utilizan videocámaras como sensores de entrada. Existen dos tipos de técnicas con las que afrontar la im-

plementación de un sistema de *tracking* visual: *Bottom-up* y *Top-down* [4]. El esquema de la figura 2.8 ilustra los diferentes tipos de técnicas asociados a cada uno de estos dos enfoques.

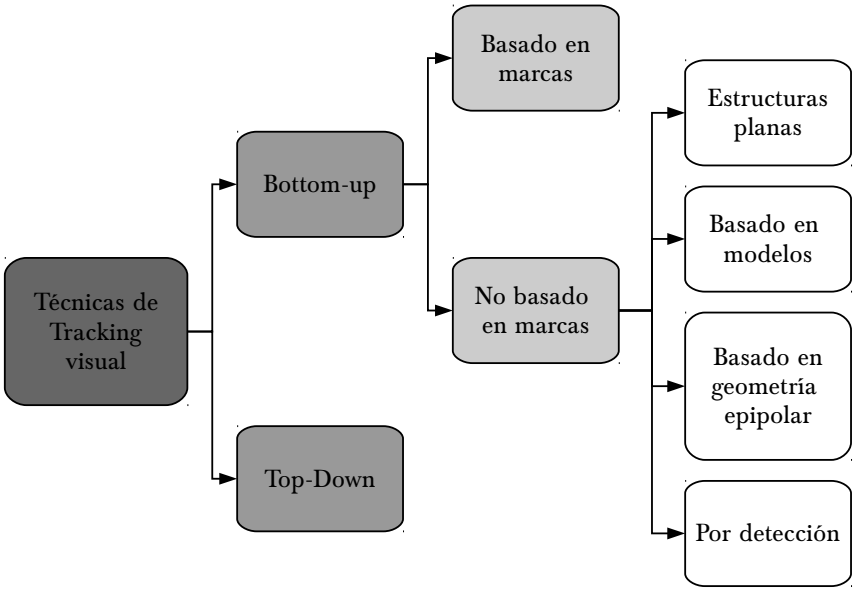


Figura 2.8: Clasificación de las diferentes técnicas de *tracking* visual

Con el enfoque *Bottom-up*, los 6 grados de libertad (3 dimensiones de posición y 3 dimensiones de orientación) se obtienen a través de características 2D de bajo nivel y su relación geométrica 3D.

El ***tracking* basado en marcas** funciona identificando ciertos patrones o marcas, los cuales normalmente son en blanco y negro para facilitar su localización y procesamiento y tienen una forma predeterminada (ver figura 2.9). Con este tipo de *tracking* se obtiene una matriz de transformación entre la marca y la cámara con la que se puede calcular el posicionamiento global de forma muy precisa.

Empleando técnicas de ***tracking* sin marcas** se buscan formas de identificar las características de los objetos presentes en la escena como referencias para el posicionamiento.

- ***Tracking* con análisis de estructuras planas:** se basa en la detección y análisis de superficies planas presentes en la escena. Estas tipo de *tracking* tiene el inconveniente del alto coste computacional que requiere.

Figura 2.9: Ejemplos de patrones para *tracking* basado en marcas

- ***Tracking* basado en modelos:** emplea modelos geométricos de tipo CAD (*Computer-Aided Design*) de partes del entorno sobre el que se va a ejecutar el *tracking*.
- ***Tracking* basado en geometría epipolar:** basa su funcionamiento en el empleo de restricciones geométricas existentes cuando se toman imágenes de la misma escena desde dos puntos de vista diferentes pero cercanos, lo que se conoce como Visión Estéreo.
- ***Tracking* por detección:** utiliza otro tipo de restricciones geométricas de forma similar al basado en geometría epipolar.

Por otro lado, el enfoque *Top-down* obtiene estos 6 grados de libertad a través de técnicas de filtrado (por ejemplo filtros bayesianos) aplicados al modelo de movimiento de la escena.

2.4. Visión Estereoscópica

La **Visión Estereoscópica (VE)** consiste en el empleo de dos vistas del mismo espacio desde diferentes puntos de visionado, de tal forma que, fusionando ambas vistas y con un correcto procesamiento, se aprecia sensación de profundidad en la escena observada.

La visión en 3D de los seres humanos es debida al procesamiento de las imágenes que llegan de los dos ojos al mismo tiempo. Al fundir ambos puntos de vista, el cerebro es capaz de proporcionar sensación de profundidad.

Para poder imitar la función del cerebro en cuanto a la visión humana, es necesario acudir a ciertas herramientas geométricas y matemáticas que se detallarán en las siguientes secciones. El objetivo último que se persigue es poder obtener datos del mundo en coordenadas tridimensionales partiendo de coordenadas bidimensionales conocidas, de tal forma que se pueda efectuar el *tracking* sin marcas.

2.4.1. Geometría epipolar

La geometría más básica en la que se basa cualquier sistema de VE es la **geometría epipolar**. Se parte de la base de que en un sistema geométrico de este tipo se tienen una cámara izquierda y una cámara derecha.

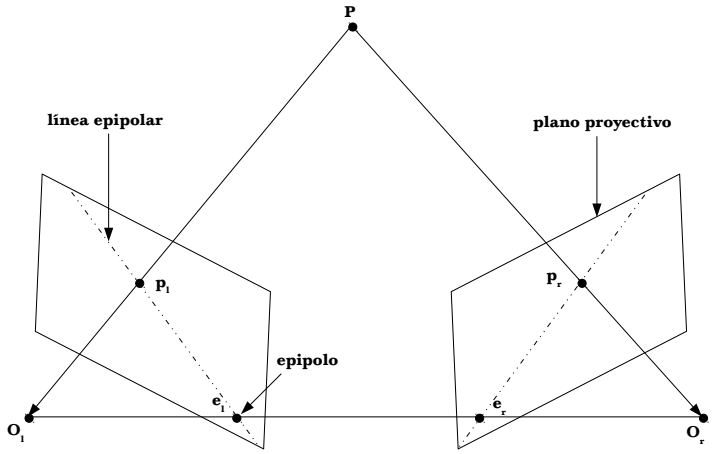


Figura 2.10: Elementos básicos de la geometría epipolar.

En la figura 2.10 se pueden observar los elementos que componen la geometría epipolar de un sistema esteroscópico. A cada cámara le corresponde un centro de proyección, denotados como O_l y O_r, y un plano proyectivo, Π_l y Π_r. El punto P, perteneciente al mundo físico tiene una proyección dentro de cada uno de los planos proyectivos. A estas proyecciones se las denomina como p_l y p_r.

Unos puntos geométricos importantes aquí son los llamados **epipolos**. Un epipolo e_l (respectivamente e_r) en un plano de imagen Π_l (respectivamente Π_r) se define como la imagen

del centro de proyección de la otra cámara O_r (respectivamente O_l). Teniendo presentes estos elementos, existe un plano formado entre el punto P y los dos epipolos e_l y e_r denominado **plano epipolar**. Las líneas $p_l e_l$ y $p_r e_r$ (líneas que atraviesan los centros de proyección y los correspondientes epipolos), se llaman **líneas epipolares**.

Para poder entender la utilidad de estos elementos se parte de que el punto P es visto en la cámara de la derecha. Debido a que la cámara solo ve p_r (la proyección de P en Π_r), el punto P podría estar localizado en cualquier lugar de la línea definida por p_r y O_r. Obviamente, esta línea contiene P, pero también contiene un gran número de puntos. Lo interesante de esto es que esta línea se encuentra proyectada en el plano de la imagen izquierda Π_l; de hecho, esa proyección es lo que antes ha sido definido como línea epipolar.

Con este escenario, pueden extraerse una serie de propiedades para la VE:

- Todo punto 3D visto por ambas cámaras está contenido en el plano epipolar que intersecta cada imagen en una línea epipolar.
- Dado un punto 2D en una imagen, su punto asociado en la otra imagen se encuentra a lo largo de la correspondiente línea epipolar. Esto es lo que se conoce como **restricción epipolar**.
- La restricción epipolar implica que la búsqueda bidimensional de puntos asociados entre dos vistas se convierte en una búsqueda unidimensional a través de las líneas epipolares una vez conocida la geometría epipolar del sistema estereoscópico.

2.4.2. Fundamentos matemáticos

Una vez conocidos los elementos que describen un sistema estereoscópico, se mostrará la utilización de dos matrices a través de las cuales pueden realizarse transformaciones sobre el sistema estereoscópico. Estas matrices son la *matriz esencial* y la *matriz fundamental*.

2.4.2.1. La matriz esencial

Se define la matriz esencial (E) de un sistema estereoscópico como aquella que contiene la información de traslación y rotación que relaciona las dos cámaras en el espacio físico.

Esto significa que, conocidas las coordenadas de un punto p_l en la imagen izquierda, puede conocerse su punto correspondiente en la imagen derecha p_r multiplicando dicho punto por la matriz esencial del sistema, de tal forma que se cumple la siguiente ecuación:

$$p_r^T E p_l = 0 \tag{2.19}$$

Es importante indicar que la matriz esencial no tiene en cuenta los parámetros intrínsecos de la cámara, por ello es por lo que trabaja en coordenadas físicas.

2.4.2.2. La matriz fundamental

La matriz esencial contiene toda la información acerca de la geometría de una cámara con respecto de la otra, pero no posee información de las cámaras en sí. En la práctica resulta más útil trabajar en unidades de píxel, ya que estas son las que se conocen en primera instancia al trabajar con imágenes.

Para conocer la relación entre un píxel en una imagen y su píxel correspondiente en la otra imagen a través de la linea epipolar debe introducirse información de los parámetros intrínsecos de las cámaras. Para hacer esto, cada punto p se sustituye por q, aplicando la matriz de cámara al anterior punto p, de tal forma que $q = Mp$, o, de forma equivalente, $p = M^{-1}q$. De esta forma, la ecuación 2.19 se convierte en:

$$q_r^T (M_r^{-1})^T E M_l^{-1} q_l = 0$$

Partiendo de esto, se define la matriz fundamental F como:

$$F = (M_r^{-1})^T E M_l^{-1} \tag{2.20}$$

Sustituyendo:

$$q_r^T F q_l = 0 \tag{2.21}$$

En esencia, la matriz F es igual que la matriz E, excepto que F opera en coordenadas píxel de la imagen, mientras que E trabaja con coordenadas físicas.

Existen un par de propiedades matemáticas interesantes acerca de la matriz fundamental:

- La matriz F es una matriz de 3x3 que tiene solamente siete grados de libertad, lo que significa que, de los nueve componentes de la matriz, dos pueden ser calculados a partir de los otros siete.
- El rango de la matriz F es 2.

Además de estas propiedades, existen otras propiedades geométricas:

- Dada una geometría bifocal, la matriz F es función exclusiva de dicha geometría, es decir, es independiente de todos los puntos P de la escena y de las proyecciones p_l y p_r de los puntos de la escena.
- La matriz F define las líneas epipolares a través de las ecuaciones:

$$l_r = Fm_l \tag{2.22}$$

$$l_l = F^T m_r \tag{2.23}$$

- La matriz F define los epipolos a través de las ecuaciones:

$$Fe_l = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.24}$$

$$F^T e_r = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.25}$$

2.4.2.3. Homografías

Otro elemento útil a la hora de calcular la geometría 3D del escenario visto a través de las cámaras son las *homografías planares* o, simplemente, *homografías* [21].

Una homografía es una matriz (generalmente de 3x3) cuya misión es la de mapear los puntos de una superficie planar bidimensional a otra. Según esta definición, una matriz que transforme los puntos de un plano en puntos del plano de cámara, será considerada una homografía. Esta transformación se realiza en términos de coordenadas físicas.

Es posible expresar este mapeo en términos de multiplicación de matrices utilizando coordenadas homogéneas para expresar el punto que se está visualizando (P) y el punto proyectado en el plano de imagen (p). Éstos se definen como:

$$P = \begin{bmatrix} X & Y & Z & 1 \end{bmatrix}^T$$

$$p = \begin{bmatrix} x & y & 1 \end{bmatrix}^T$$

La homografía relaciona ambos puntos de la siguiente manera:

$$p = sHP \tag{2.26}$$

Se ha introducido el parámetro s, el cual es un factor de escala arbitrario (entendiéndose que la homografía está definida bajo ese factor, debido al desconocimiento de las medidas del mundo real). En la figura 2.11 puede verse la implicación de utilizar la homografía en este escenario.

Podría llegar a pensarse que las funciones de la matriz esencial y de la homografía son las mismas, puesto que ambas realizan transformaciones en la geometría del sistema de VE sin tener en cuenta los parámetros intrínsecos de la cámara. La diferencia reside en que la homografía trabaja sobre planos que son visibles desde la cámara y relacionaría un punto de ese plano con el de su propio plano. La matriz esencial no tiene en cuenta esta restricción

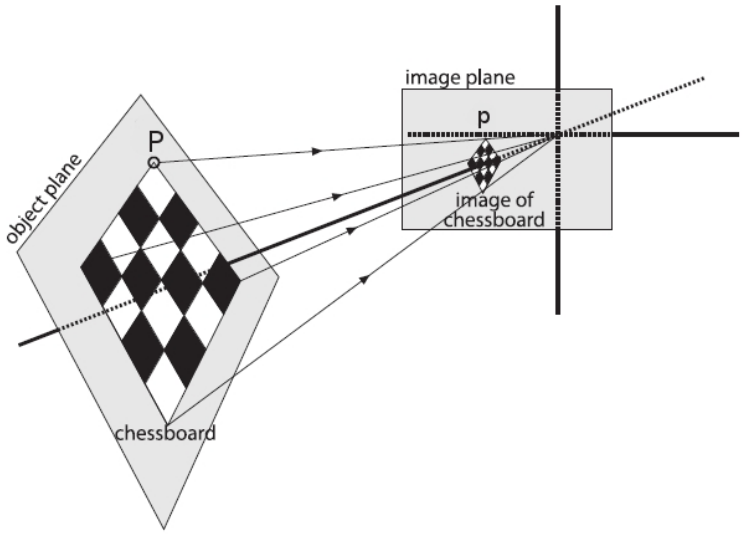


Figura 2.11: Transformación de un punto P del mundo real a su proyección p en el plano de cámara utilizando una homografía. Imagen: *Learning OpenCV* [19].

y solo es capaz de relacionar un punto en una imagen que se encuentre alineado con otra imagen.

2.4.2.4. Rectificación

Para simplificar el proceso de reconstrucción 3D, es muy común aplicar una transformación a las vistas de las cámaras para facilitar la búsqueda de correspondencias que se puedan encontrar en líneas de scan horizontales. Esta acción recibe el nombre de ***rectificación***.

Hablando en términos de la geometría epipolar, la rectificación busca que las líneas epipolares sean horizontales para ambos planos de cámara. Esto implica que el epipolo va a ser llevado al infinito, es decir, $e = (1, 0, 0)^T$.

Existen diferentes algoritmos con los que computar la rectificación de las imágenes. Dos de ellos son el *algoritmo de Hartley* y el *algoritmo de Bouguet*.

- **Algoritmo de Hartley**

  El algoritmo de Hartley [22] tiene la característica de que es capaz de realizar una rectificación de las imágenes sin la utilización de los parámetros extrínsecos del sistema. Para ello, tiene como base de cálculo la matriz fundamental.

Su funcionamiento consiste en encontrar homografías que mapeen los epipolos hacia el infinito mientras se busca minimizar las disparidades computadas entre las imágenes estéreo. Esta tarea se simplifica cuando se tiene un conjunto de correspondencias (*matchings*) de puntos entre cada par de imágenes. A través de estas correspondencias es sencillo calcular la matriz fundamental y utilizarla junto con los parámetros intrínsecos de la cámara para realizar la rectificación.

La ventaja del algoritmo de Hartley es que puede ser utilizado para un sistema de *egomotion* o de cámara en movimiento. Sin embargo, los resultados son algo más inexactos que con el algoritmo de Bouguet, en gran parte porque se desconoce la escala del mundo real con respecto al sistema estereoscópico.

- **Algoritmo de Bouguet**

  Dadas la matriz de rotación y el vector de traslación entre dos imágenes estéreo, el algoritmo de Bouguet [23] simplemente intenta minimizar la cantidad de cambios de reproyección producidos por cada par de imágenes mientras maximiza el area de visionado común.

  Este algoritmo es adecuado en situaciones en las que puedan emplearse patrones de calibración tales como en un brazo robotizado o en una instalación con cámaras de seguridad.

2.5. Metadatos

Se conocen como metadatos a aquellos datos que proporcionan información sobre otros datos. En general, los metadatos contienen información que no suele ser importante para el usuario, pero si lo es para los sistemas que manejan los objetos sobre los que informan los metadatos. Un ejemplo de metadato podría ser la fecha de creación de un archivo.

Los metadatos son utilizados en diversos campos de la informática como son la web semántica o la minería de datos. En función del tipo de dato que describan, existen un conjunto muy diverso de estándares que definen la forma de estos metadatos [25]. Algunos de estos estándares son: ID3, Dublin Core, OWL, MPEG21, etc.

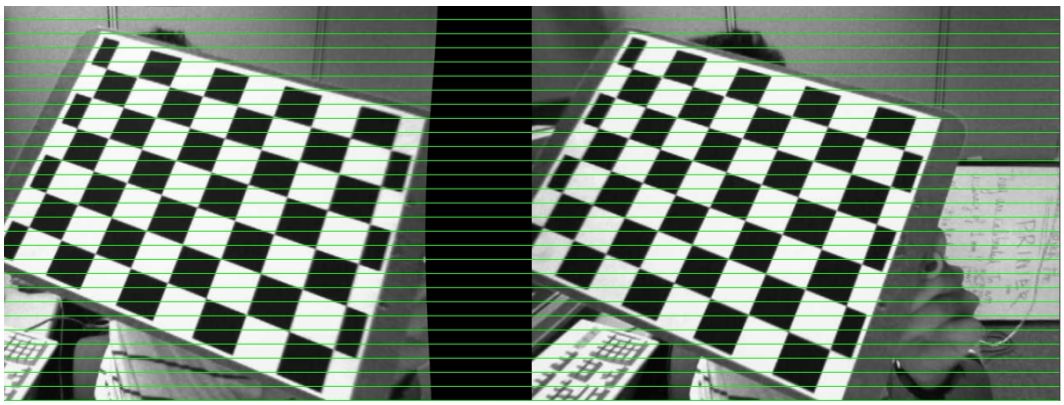


Figura 2.12: Ejemplo de imágenes estereoscópicas rectificadas. Las líneas horizontales representan las líneas epipolares. Imagen: ejemplos de la biblioteca OpenCV-2.2.

El estándar que se ha estudiado para este proyecto es MPEG-7, cuyas características se explican en la siguiente sección.

2.5.1. Estándar MPEG-7

MPEG-7, también conocido como **Interfaz de Descripción de Contenidos Multimedia (*Multimedia Content Description Interface*)**[4] es un estándar ISO/IEC desarrollado por el grupo MPEG. Su objetivo principal es el de establecer una representación estandarizada de la información audiovisual a través de sus metadatos.

El estándar se compone de un conjunto de partes bien diferenciadas:

1. **Sistemas:** Incluye el formato binario para codificar las descripciones MPEG-7 y la arquitectura terminal.

2. **Description Definition Language (DDL):** DDL es un lenguaje basado en XML que permite la creación de nuevos *Esquemas de Descripción* y, posiblemente, *Descriptores*. También permite la extensión y modificación de los *Esquemas de Descripción* existentes.

[4]Toda la documentación acerca del estándar MPEG-7 puede ser revisada en la página web: *http://mpeg.chiariglione.org/standards/mpeg-7/mpeg-7.htm*

3. **Visual:** Las *Herramientas de Descripción Visual* consisten en estructuras básicas y descriptores que cubren los siguientes aspectos visuales básicos: color, textura, forma, movimiento, localización y reconocimiento de caras.

4. **Audio:** Se proporcionan estructuras para la descripción del contenido del audio. Estas estructuras contienen un conjunto de descriptores de bajo nivel (aspectos espectrales, paramétricos y temporales de una señal de audio) y otras herramientas de descripción más específicas que incluyen reconocimiento de sonidos, herramientas de descripción de timbres de instrumentos musicales, ...

5. **Multimedia Description Schemes (MDS):** Los *Esquemas de Descripción Multimedia* se encargan de especificar la relación entre los descriptores y los *Esquemas de Descripción* con el elemento multimedia.

6. **Reference Software:** Contiene un software de simulación para *Descriptores* (Ds), *Esquemas de Descripción* (DSs), *Esquemas de Codificación* (CSs) y DDL.

7. **Conformance:** Incluye líneas guía y procedimientos para la realización de pruebas.

8. **Extracción y uso de descripciones:** Proporciona información sobre extracción y uso de herramientas de descripción.

9. **Perfiles:** Contiene una colección de perfiles estandarizados y niveles para el MPEG-7, especificados en ISO/IEC 15938.

10. **Definición de Esquemas:** Contiene una colección de esquemas recopilados de diferentes estándares, correciones y enmiendas.

2.5.1.1. Descriptores Visuales

En este proyecto se ha utilizado una parte muy específica del estándar MPEG-7: los **descriptores visuales**.

MPEG-7 proporciona un conjunto muy amplio de descriptores visuales, que dan información acerca de aspectos muy diferentes tanto de imágenes como de vídeos. En la figura 2.13 puede verse una clasificación de estos descriptores según el aspecto visual que describen.

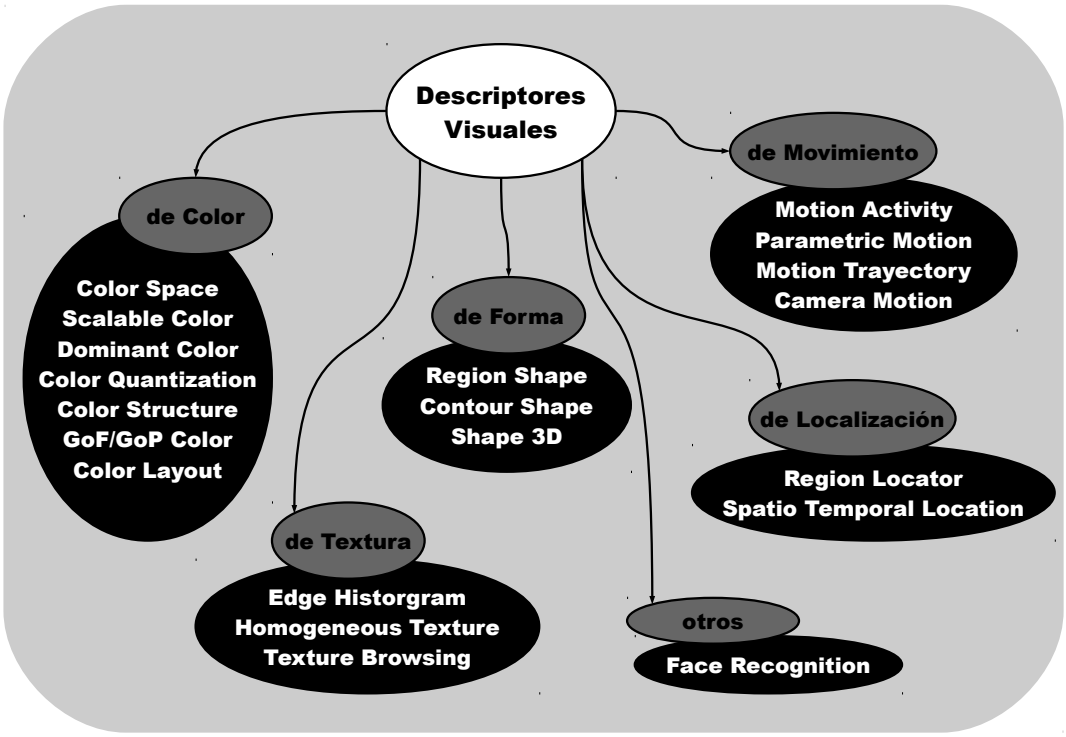


Figura 2.13: Clasificación de los diferentes descriptores visuales del estándar MPEG-7.

De este conjunto de descriptores visuales, a continuación se explicarán brevemente aquellos que han sido utilizados dentro de Arges.

- **Color Layout**

  Este descriptor permite representar la distribución espacial del color dentro de las imágenes de una manera muy compacta, con lo cual, representa una herramienta de gran utilidad a la hora de buscar imágenes a partir de modelos determinados, y lo hace con gran eficiencia y velocidad. Su fácil cálculo permite también usarlo en la comparación de secuencias de imágenes, en las cuales se precisa un análisis de similitud entre cada una de sus componentes.

  Las grandes ventajas de este descriptor son:

  - No depende ni del formato, ni de la resolución, ni del margen dinámico de las imágenes o vídeos en que se use. Por este motivo, puede usarse para comparar

imágenes o vídeos con distintas resoluciones o para comparar imágenes enteras con partes de imágenes.

- El software/hardware que requiere es mínimo (usa solamente 8 bytes por imagen cuando trabaja por defecto). Esto lo convierte en un descriptor adecuado para ser utilizado en dispositivos móviles en los que los recursos se ven limitados por la capacidad del hardware.
- Las características que analiza de una imagen las representa en dominio frecuencial de manera que los usuarios pueden manipular con facilidad su contenido ciñéndose a la sensibilidad del sistema visual humano.
- Permite trabajar con distintas precisiones de descripción.

- **Edge Histogram**

  Es un descriptor que facilita información sobre el tipo de contornos o bordes que aparecen en la imagen.

  Trabaja dividiendo la imagen en 16 (4x4) sub-imágenes y es capaz de analizar en ellas el tipo de bordes existentes con el uso de distintos filtros que le permiten diferenciar si son bordes horizontales, verticales, oblicuos o aleatorios. Su utilización principal es la comparación imagen-imagen, especialmente en imágenes con poca uniformidad de contornos.

- **Scalable Color**

  Este descriptor consiste en un histograma de color en el espacio HSV[5], codificado con una medida de Haar.

  Su representación se puede escalar para adecuarla al máximo tamaño de datos con el que se quiere trabajar.

  Este descriptor resulta útil al ser utilizado en comparaciones entre imágenes o en búsquedas basadas en características de color. La fiabilidad de la búsqueda aumenta proporcionalmente al número de colores distintos que se tengan en cuenta en el histograma.

[5]Del inglés, *Hue, Saturation, Value* (Matiz, Saturación, Valor).

2.5.1.2. Un ejemplo de aplicación con Descriptores Visuales: LIRE Mosaics

Existen multitud de bibliotecas para el desarrollo de aplicaciones empleando el estándar MPEG-7. Un ejemplo de biblioteca de este tipo es **LIRE** (*Lucene Image REtrieval*)[6]. LIRE es una biblioteca Java basada en el código de los programas Caliph & Emir (*Common and Lightweight PHoto Annotation & Experimental Metadata-based Image Retrieval*). Esta biblioteca proporciona funciones especialmente diseñadas para la extracción de descriptores visuales y comparación entre los mismos para la creación de programas de comparación entre imágenes.

Los descriptores empleados en esta biblioteca son 4: Dominant Color, Scalable Color, Color Layout y Edge Histogram.

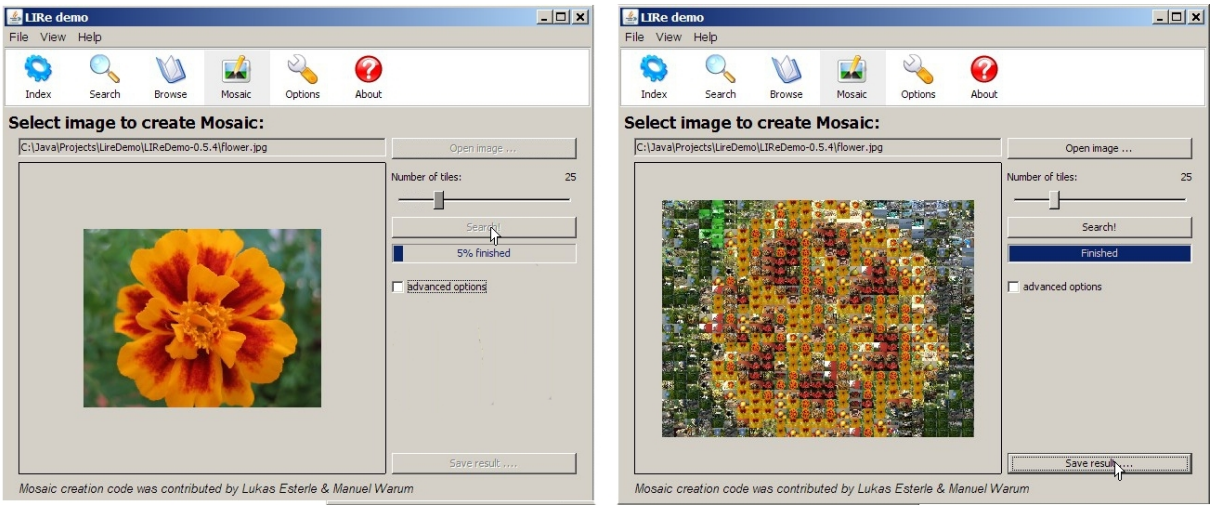


Figura 2.14: Ejemplo de uso de LIRE Mosaics.

La figura 2.14 muestra un ejemplo de aplicación realizada con la biblioteca LIRE, llamada LIRE Mosaics. Esta aplicación funciona creando un índice de imágenes de las cuales se conocen sus descriptores visuales. Una vez creado el índice, se proporciona una imagen individual y, basándose en los descriptores de color, el programa es capaz de realizar un mosaico que imite la forma y el color de la imagen individual con las fotos que tiene indexadas.

[6]Toda la documentación y la biblioteca pueden ser descargados desde la página web del proyecto Lire: *http://www.semanticmetadata.net/lire/*

Capítulo 3

Objetivos

El objetivo principal de Arges es el desarrollo de un sistema para el posicionamiento en interiores empleando técnicas de Visión por Computador y Realidad Aumentada. Este sistema será utilizado para guiar al usuario en el interior de un edificio, en el marco del proyecto ElCano.

A partir de este objetivo principal se definen una serie de subobjetivos funcionales específicos que detallan el alcance del presente Proyecto Fin de Carrera:

- **Soporte para métodos de *tracking* basados en marcas.**

  El sistema permitirá el uso de técnicas de registro de la realidad basadas en marcas. Este tipo de técnicas son ampliamente utilizadas por su potencia, limitadas necesidades de cómputo y versatilidad. Además tienen la capacidad de ser capaces de proporcionar información de posicionamiento absoluto con gran precisión, sin necesidad de tener en cuenta información histórica.

- **Cálculo del posicionamiento mediante análisis de la geometría epipolar.**

  Arges incorporará un módulo de registro visual sin marcas, con el objetivo de paliar las limitaciones del *tracking* con marcas. El paradigma será el de la simulación de Visión Estereoscópica mediante análisis de la geometría epipolar de la escena. Debido a que solo se permite el uso de una cámara, este *tracking* se desarrollará siguiendo algoritmos especialmente diseñados para trabajar en *ego-motion*.

- **Soporte para el posicionamiento mediante descriptores de imagen.**

  Como apoyo para el posicionamiento, se incorporará soporte de *tracking* basado en la extracción y comparación de descriptores de imagen. Se tendrá una base de datos de posiciones de imágenes conocidas de la escena y de los descriptores que las definen.

  A través de algoritmos de comparación de descriptores, el módulo será capaz de dar información del sitio más cercano desde el que se puede ver la escena actual según las imágenes de la base de datos. De esta forma se podrá dar información de posicionamiento absoluto, al igual que con el registro basado en marcas con cualquier configuración de la cámara (posición y rotación) pero con una granularidad y precisión sensiblemente inferior.

- **Desarrollo modular. Bajo acoplamiento entre módulos.**

  El sistema debe permitir utilizar un único módulo u obtener una salida compuesta por la fusión de la información procedente de diversos métodos de *tracking*. Para ello, se proporcionará una interfaz amigable que permita la visualización de los módulos que se seleccionen.

- **Sistema de pruebas determinista.** Para medir la fiabilidad del sistema será necesario desarrollar entornos de prueba virtuales para medir el error cometido en cada método. Gracias a los entornos de prueba virtuales, se podrán conocer las medidas reales y las estimadas con mayor precisión. Después de realizar pruebas en entorno virtual, se desarrollarán pruebas en un escenario real.

- **Bajos requisitos computacionales.**

  El sistema debe funcionar en tiempo real a una tasa interactiva de 20fps. Se pretende que el sistema pueda ser utilizado sin problemas en dispositivos con limitada capacidad de cómputo.

- **Multiplataforma y uso de estándares libres.**

  El desarrollo de Arges se realizará siguiendo estándares y tecnologías libres multiplataforma, con el objetivo de que pueda ser utilizado en el mayor número de platafor-

mas posibles. Gracias a la naturaleza multiplataforma de las bibliotecas utilizadas, Arges podría ser compilado para su ejecución en plataformas GNU/Linux, Windows y MAC.

Capítulo 4

Metodo de trabajo

En este capítulo se mostrarán los pasos seguidos para el desarrollo completo del proyecto Arges, así como las herramientas que se han ido utilizando para hacerlo posible.

4.1. Metodología de trabajo

Al tratarse de un proyecto con una amplia labor de investigación, se decidió dividir su elaboración en módulos. Cada uno de estos módulos coincide con un sistema de *tracking*. Una vez implementados y probados por separado estos módulos, se procedió a efectuar la integración de los mismos dentro del sistema completo, desarrollado también de forma modular. En la Figura 4.1 se puede observar el flujo de trabajo aproximado que se ha seguido durante el desarrollo de este proyecto de fin de carrera, que responde a una **metodología de desarrollo iterativa e incremental**.

La actividad crítica dentro de este flujo de trabajo se encontró en el desarrollo del módulo de *tracking* estéreo. Aquí fueron necesarias muchas iteraciones hasta dar con un prototipo que se ajustara a lo que realmente se necesitaba.

4.2. Herramientas

En esta sección se detallan el conjunto de herramientas, tanto software como hardware, que han sido utilizadas en el desarrollo de este proyecto. En primer lugar se dará una panorámi-

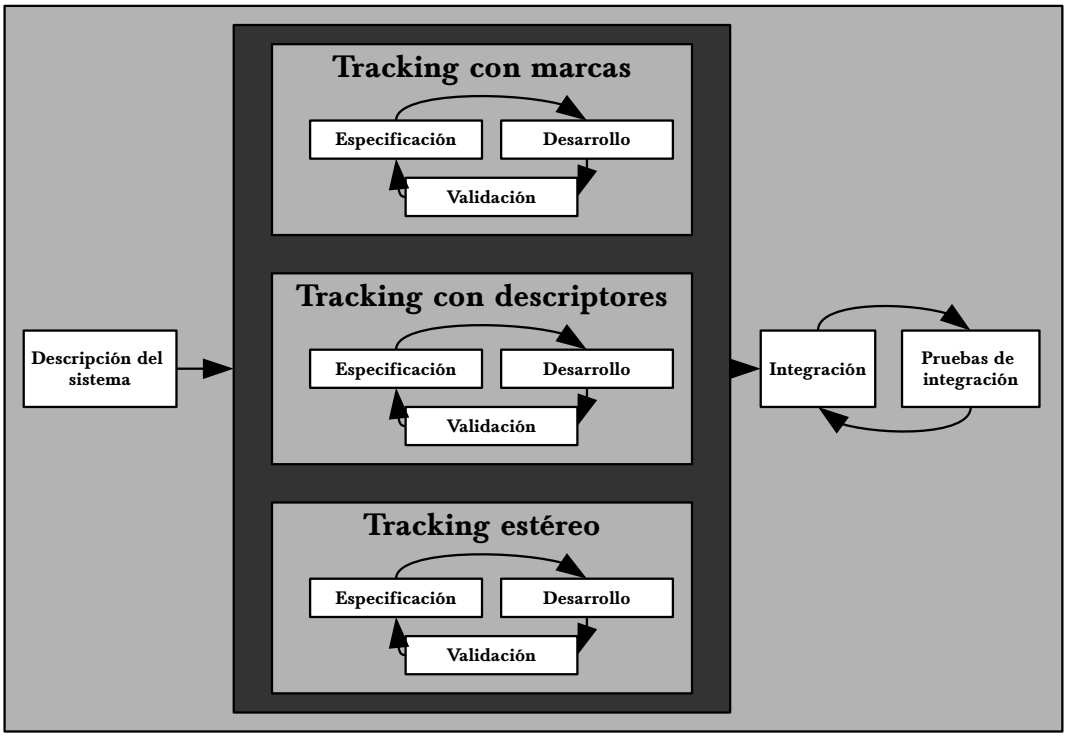


Figura 4.1: Diagrama que representa el flujo de trabajo iterativo e incremental del proyecto.

ca general del funcionamiento de las bibliotecas ***ARToolkit*** y ***OpenCV***, que constituyen la base de este proyecto.

4.2.1. ARToolkit

ARToolkit[1] es una biblioteca escrita en lenguaje C y empleada en el desarrollo de aplicaciones de RA. Desarrollada en el *Human Interface Technology Laboratory* de la Universidad de Washington, se distribuye bajo licencia GPL.

Su principal utilidad reside en la simplicidad para la creación de aplicaciones de realidad aumentada utilizando *tracking* con marcas y gráficos 3D generados con ***OpenGL*** (ver Figura 4.2).

[1]Toda la documentación y software de ARTookit puede ser descargado desde su página web oficial: http://www.hitl.washington.edu/artoolkit/

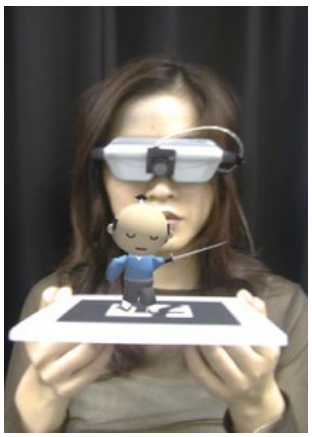


Figura 4.2: Ejemplo de aplicación realizada con la biblioteca ARToolkit

Algunas de las características incluidas en ARToolkit son:

- *Tracking* de posición/orientación para cámara individual.
- Empleo de marcas simples en blanco y negro para el *tracking*.
- Capacidad para la utilización de cualquier patrón como marca.
- Calibración sencilla de la cámara.
- Permite el desarrollo de aplicaciones de RA en tiempo real.
- Multiplataforma.

ARToolkit ha sido utilizado en Arges para la implementación del *tracking* con marcas.

4.2.2. OpenCV

OpenCV[2] (*Open Source Computer Vision*, es una biblioteca que implementa un conjunto de funciones cuyo uso está orientado a la Visión por Computador en tiempo real [15]. Su código es de libre distribución para uso tanto académico como comercial.

[2]Toda la documentación y software de OpenCV puede ser descargado desde http://opencv.willowgarage.com/wiki/

Actualmente pueden encontrarse implementaciones de OpenCV en C, C++, Python o Java, además de funcionar para Windows, Linux, Android o Mac. Para este proyecto se ha optado por utilizar la version 2.2 en C++[3] para Linux.

Algunas de las operaciones y funciones capaz de realizar la biblioteca OpenCV son:

- Operaciones sobre vectores y matrices.
- Funciones de dibujado de figuras simples (círculos, líneas, rectángulos,...).
- Persistencia XML/YAML.
- Clustering.
- Funciones sobre imágenes, tales como: transformaciones en perspectiva, detección de puntos característicos y descriptores, cálculo y operaciones sobre histogramas, ...
- Análisis de movimiento y *tracking* de objetos.
- Creación de interfaces graficas.
- Calibración de cámaras.
- Operaciones con cámaras estéreo.

A continuación, se realizará una descripción de los tipos de datos más representativos y de las funciones más generales para el funcionamiento de un programa con OpenCV.

4.2.2.1. Módulos

Existen un total de 9 módulos en los que se dividen las funciones de OpenCV:

- **Módulo *core*:** Es el que alberga el conjunto de estructuras de datos básicas para trabajar con la biblioteca. Contiene las operaciones sobre vectores y matrices y las funciones de dibujado de figuras simples.

[3]En este documento se explicarán los aspectos sintácticos de OpenCV referentes también a la versión en C++.

- **Módulo *imgproc*:** Contiene el conjunto de operaciones para el procesamiento y manipulación de imágenes.
- **Módulo *features2d*:** Aquí se encuentran las funciones para la extracción de descriptores y puntos característicos de imágenes.
- **Módulo *flann*:** Contiene funciones de *clustering* y de búsqueda en espacios multidimensionales para la implementación de sistemas de aprendizaje.
- **Módulo *objdetect*:** Dentro de este módulo se encuentran diferentes funciones para la detección de objetos.
- **Módulo *video*:** Contiene funciones de análisis de movimiento y de utilización del Filtro de Kalman.
- **Módulo *highgui*:** Contiene funciones para la implementación de interfaces gráficas sencillas. Estas funciones permiten:
  - Crear y manipular ventanas que puedan mostrar imágenes y "recordar" su contenido.
  - Añadir barras de desplazamiento a las ventanas y manejar eventos de ratón.
  - Leer y escribir imágenes en disco.
  - Leer vídeo de una cámara o un archivo y guardar vídeo en un archivo.
- **Módulo *calib3d*:** Alberga las funciones para el calibrado de cámaras, estimación de posición y tratamiento de imágenes en estéreo.
- **Módulo *ml*:** Conjunto de clases y funciones para clasificación estadística, regresión y *clustering* de datos.

4.2.2.2. Tipos de datos básicos en OpenCV

El tipo de datos más básico y sobre el cual se efectuan la mayoría de operaciones en OpenCV es la matriz, denominada *cv::Mat*.

cv::Mat es tratado como un objeto con filas y columnas sobre cuyas celdas pueden almacenarse tipos primitivos de datos. Generalmente se utilizan para representar las imágenes de forma matricial.

En OpenCV existe un convenio para tratar el formato de los tipos primitivos de datos. Los tipos primitivos que soporta son *unsigned char, signed char, unsigned short, signed short, int, float* o *double*. La notación empleada para referirse a estos tipos de datos también contempla la precisión (o profundidad de los bits como se denomina en la documentación de OpenCV) y el número de canales[4]. De esta forma, los tipos de datos se notan de la siguiente forma: CV_*«precisión en bits»*{U‖S‖F}C*«número de canales»*.

Por ejemplo, si se tiene una matriz de CV_32FC3, se almacenan números de 32 bits de tipo *float* y con 3 canales.

Una vez creadas las matrices, OpenCV implementa un conjunto de operaciones básicas que pueden realizarse sobre ellas, tales como:

- **Escalado:** M * «número»
- **Multiplicación de matrices:** M * N
- **Transposición:** M.t()
- **Inversión:** M.inv()
- . . .

Otro tipos de datos propios de OpenCV que resultan muy útiles son ***cv::Point*** y ***cv::Point3***. Estas clases representan puntos en 2 dimensiones y 3 dimensiones respectivamente. Los valores para las coordenadas que representan a un punto, al igual que ocurría con las matrices, pueden ser almacenados en diferentes tipos de datos. De esta forma, *cv::Point2i* corresponde a la clase que representa un punto 2D con coordenadas de tipo *int*; *cv::Point3f* corresponde a la clase que representa un punto 3D con coordenadas de tipo *float*.

Por último, mencionar la clase ***cv::Size***, que se utiliza para establecer las dimensiones de un rectángulo.

[4]Se entiende que una imagen en blanco y negro será almacenada en un *cv::Mat* de un solo canal, mientras que una imagen en RGB será almacenada en un *cv::Mat* de tres canales.

4.2.2.3. Funciones más comunes de OpenCV

En este apartado se comentarán algunas de las funciones más utilizadas en los programas realizados con OpenCV.

- *void* ***cv::NamedWindow****(const string& winname, int flags).* Esta función crea una ventana donde poder visualizar imágenes o vídeo.
  - *winname*: Nombre asignado a la ventana que será usado como identificador de la misma.
  - *flags*: Opciones para la visualización de la ventana. Actualmente solo se permite usar *CV_WINDOW_AUTOSIZE*, que ajusta la ventana al tamaño de la imagen que está mostrando.
- *cv::Mat* ***cv::imread****(const string& filename, int flags)*. Carga una imagen desde un archivo e introduce los valores de sus píxeles en un objeto *cv::Mat*.
  - *filename*: Nombre del archivo de imagen que se va a cargar.
  - *flags*: Entero que indica la distribución de colores de la imagen:
    - `>0`: La imagen cargada será forzada a tener 3 canales de color.
    - `=0`: La imagen se cargará en escala de grises.
    - `<0`: La imagen se cargará según su distribución de canales de color original.
- *void* ***cv::imshow****(const string & winname, const cv::Mat& image)*. Muestra una imagen en la ventana especificada.
  - *winname*: Nombre de la ventana.
  - *image*: Imagen que va a ser mostrada.
- *void* ***cv::Mat::release()***. Se encarga de la eliminación de la memoria del objeto *cv::Mat* que llama a la función.
- *void* ***cv::waitKey****(int delay)*. Mantiene el programa en espera hasta la pulsación de una tecla.

- *delay*: Lapso de tiempo de espera en milisegundos. Si se introduce un valor cero, significa que esperará todo el tiempo que sea necesario a recibir un evento de teclado.

Una vez conocidas algunas funciones de OpenCV, A continuación se muestra un pequeño ejemplo de código con OpenCV en el que se mostraría una imagen cargada desde un archivo en una ventana:

```
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
int main(int argc, char *argv[]){
    cv::Mat img = cv::imread("/home/riky/img.jpg");
    cv::namedWindow("Imagen");
    cv::imshow("Imagen", img);
    cv::waitKey();
    img.release();
    return 1;
}
```

4.2.3. MPEG-7 FexLib

MPEG-7 Feature Extraction Library[5] constituye una extensión de *OpenCV* que implementa funciones de extracción y tratamiento de descriptores visuales definidos en el estándar MPEG-7.

La versión utilizada para este proyecto ha sido implementada sobre la versión 2.2 en C++ de *OpenCV*. Los descriptores visuales gestionados por la biblioteca son los siguientes:

- *Color Structure Descriptor* (CSD)
- *Scalable Color Descriptor* (SCD)
- *GoF/GoP Color Descriptor* (GoF/GoP)
- *Color Layout Descriptor* (CLD)

[5] http://www.cs.bilkent.edu.tr/ bilmdg/bilvideo-7/Software.html

- *Dominant Color Descriptor* (DCD)
- *Homogeneous Texture Descriptor* (HTD)
- *Edge Histogram Descriptor* (EHD)
- *Face Recognition Descriptor* (FRD)

Las funciones de esta biblioteca permiten extraer los datos de los descriptores de imagen y el valor de comparación existente entre ellos para implementar el *tracking* con descriptores de Arges.

4.2.4. Otras herramientas software

4.2.4.1. Aplicaciones de desarrollo

- **Eclipse CDT (*Versión 7.0.2*)**: entorno de desarrollo con soporte para C/C++.
- **GNU Emacs (*Versión 23.1.1*)**: editor de textos con el que se han realizado pequeñas modificaciones de código y con el que se ha editado y compilado todo el código LaTeXpara la documentación.
- **GCC (*Versión 4.4.5*)**: de este conjunto de compiladores se ha utilizado el compilador *g++*.
- **Make**: utilizado para el sistema de *Makefiles*, que automatizan la compilación del código del proyecto.

4.2.4.2. Bases de datos

- **libmysql++ (*Versión 3.0.9-1*)**: biblioteca de desarrollo de MySQL para C/C++.
- **MySQL Server (*Versión 5.1.49-1*)**: servidor de bases de datos de MySQL.
- **MySQL Client (*Versión 5.1.49-1*)**: cliente de bases de datos de MySQL.

- **MySQL Administrator (*Versión 1.2.12*)**: consola visual de administración que permite administrar fácilmente el entorno MySQL y obtener información más exhaustiva de cómo están operando las bases de datos.
- **MySQL Query Browser (*Versión 1.2.12*)**: herramienta gráfica para la consulta de la base de datos con un editor SQL con resaltado de sintaxis. Las tablas resultantes de la consulta se pueden editar antes de que los cambios sean realizados en la base de datos.

4.2.4.3. Documentación y gráficos

- **OpenGL** [26]: constituye una API multilenguaje y multiplataforma destinada a la creación de gráficos en 2D y 3D. Ha servido para la creación de los gráficos mostrados en la interfaz.
- **Dia (*Versión 0.97.1*)**: sirve para generar todo tipo de diagramas. Ha sido utilizado para la generación de los diagramas UML que definen el sistema.
- **LibreOffice Draw (*Versión 3.3.2*)**: herramienta utilizada en la generación de gráficos y algunos diagramas de la documentación.
- **LibreOffice Calc (*Versión 3.3.2*)**: herramienta utilizada para la realización de cálculos en las tablas de las pruebas y en la generación de gráficos estadísticos a partir de estas tablas.
- **Gimp (*Versión 2.6.10*)**: herramienta de manipulación de gráficos utilizada para la generación de imágenes en la documentación.
- **Blender (*Versión 2.60.0*)**: suite de modelado 3D utilizada en la generación de escenarios 3D para la simulación del sistema en un entorno controlado.
- **LaTeX**: programa de maquetación con el que se ha generado esta documentación.

4.2.4.4. Lenguajes

Arges ha sido desarrollado empleando empleando el lenguaje de programación **C/C++** [27] [28], ya que constituye el estándar en aplicaciones de VC e informática gráfica.

4.2.4.5. Sistema operativo

El sistema operativo utilizado para desarrollar todo el proyecto ha sido **Ubuntu 10.10**, también conocido como Ubuntu *Maverick Meerkat*. Este sistema operativo posee núcleo de *GNU/Linux*, y está basado en *Debian*.

4.2.5. Hardware

El equipo utilizado en el desarrollo y pruebas del proyecto ha sido un ordenador portátil *Acer Aspire 6930*; procesador *Intel Core 2 Duo T5800* 2.00 GHz, 2GB de RAM y tarjeta gráfica *NVIDIA GeForce 9300M GS* con 512MB de RAM.

La cámara empleada en el desarrollo del proyecto ha sido la interna del ordenador, mientras que para las pruebas en el escenario real se ha utilizado una webcam *Logitech Sphere AF*, compatible con *GNU/Linux*.

Además, se ha utilizado el servidor `devoreto.esi.uclm.es` de la Escuela Superior de Informática (UCLM) para el control de versiones en ciertas partes del proyecto.

Capítulo 5

Arquitectura

En esta sección se abordará la explicación de la estructura de cada una de las partes que componen Arges. La descripción del sistema se realizará siguiendo una aproximación *Top-down*, comenzando por cómo se han integrado los módulos que componen el sistema para después explicar el funcionamiento de cada módulo, desde los de más alto nivel hasta los módulos de *tracking*.

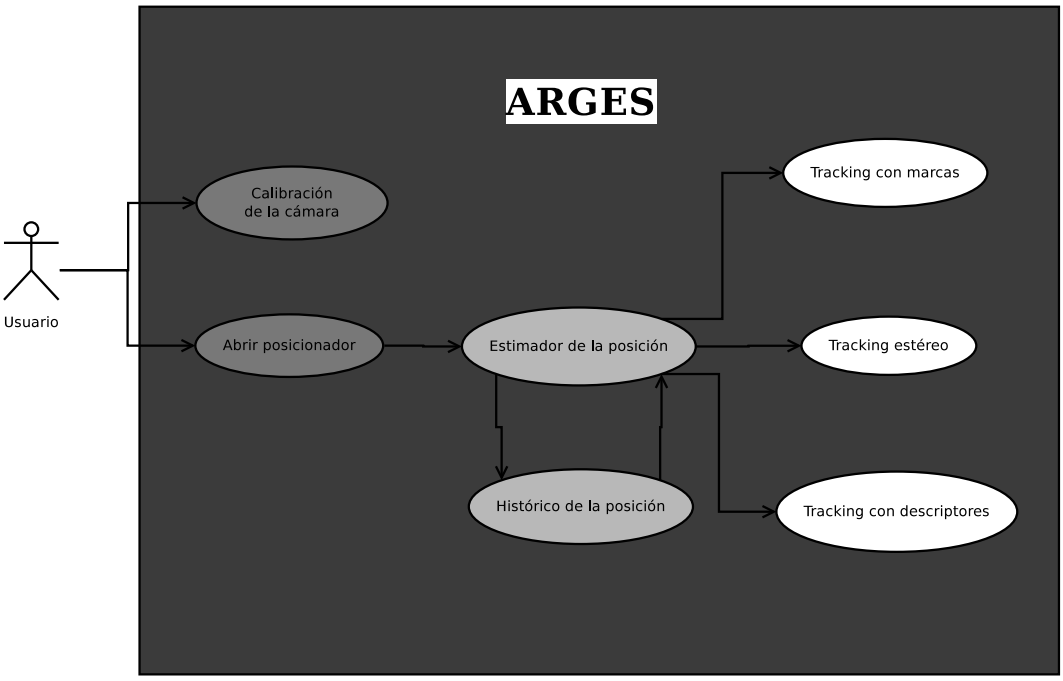


Figura 5.1: Diagramas de casos de uso del proyecto.

En la figura 5.1 se muestran los casos de uso que se contemplan en el sistema. Como se puede observar, el usuario interactúa con el sistema activando el módulo de posicionamiento. Este módulo será el que active todos los sistemas de *tracking*.

En las siguientes secciones se explicarán cada uno de los módulos que componen Arges, lo que da cuenta de la estructura modular del sistema. En la figura 5.2 se puede observar la organización del conjunto de módulos que componen el sistema.

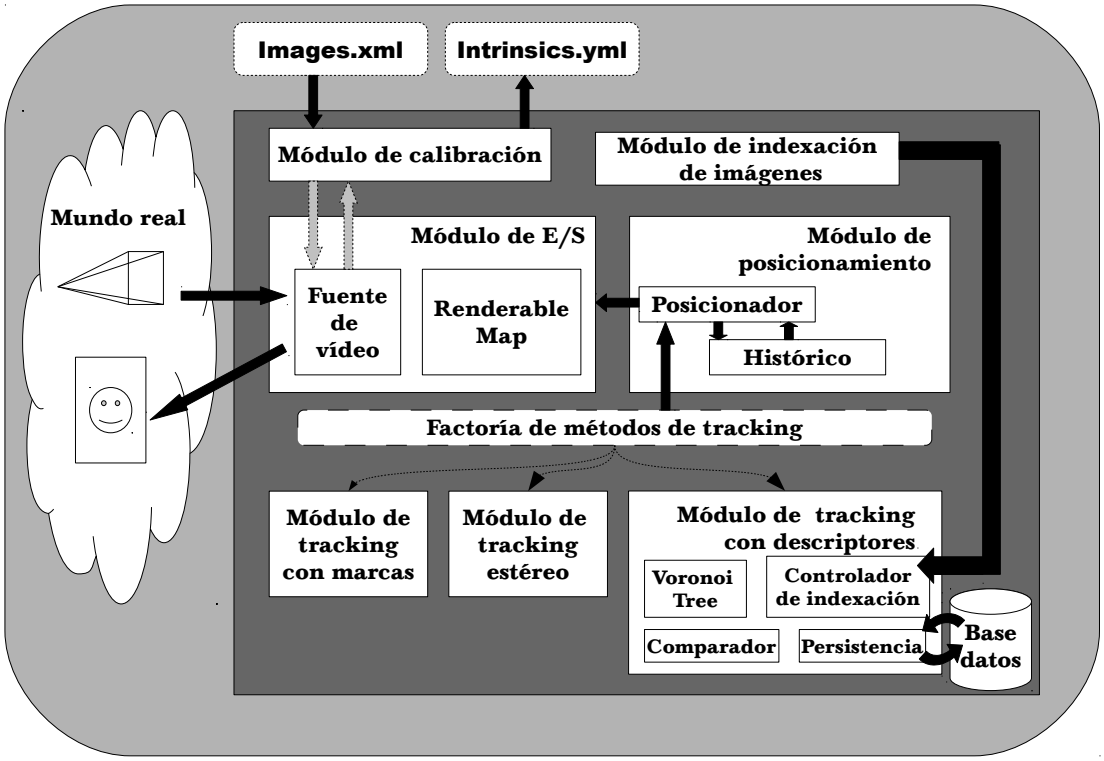


Figura 5.2: Organización de los módulos de Arges

Para un correcto funcionamiento del conjunto de los módulos de *tracking* es necesaria una correcta calibración de la cámara del sistema, por lo que en el proyecto se facilita un módulo aparte para llevar a cabo esta tarea.

5.1. Módulo de calibración de la cámara

En esta sección se explicará el algoritmo a seguir para extraer los parámetros intrínsecos de la cámara, que serán los utilizados por Arges en el *tracking* estéreo [29]. En el Apéndice A se explican las posibles variantes que se pueden encontrar al ejecutar el programa, pero en sí la forma de proceder es siempre la misma.

La entrada al programa siempre será un conjunto de pares de imágenes estéreo en los que se encuentre visible el patrón de calibración, con forma de tablero de ajedrez. Los archivos de imagen se deben listar en un archivo XML, que será el que lea el programa para acceder al directorio correcto (ver Apéndice A para conocer el aspecto de un archivo XML de entrada).

El funcionamiento del algoritmo de calibración se basa en encontrar las esquinas de los cuadrados en el patrón de calibración y realizar una asociación entre ellos. Además, habrá que proporcionar un valor de la medida real del mundo. Se proporciona la medida del lado de un cuadrado del tablero de ajedrez para ello y se utiliza un vector con la medida de las esquinas partiendo de esta base.

Los puntos detectados en las imágenes se almacenan en un vector de objetos *cv::Point2f*. Para el programa se utilizará una matriz que contendrá dos vectores, uno por cada conjunto de puntos en una de las imágenes. Los puntos 3D que miden las distancias de las esquinas del tablero en unidades del mundo se almacenan en un vector de objetos *cv::Point3f*.

Para detectar las esquinas en las imágenes se utiliza la función *cv::findChessboardCorners(...)*. Adicionalmente, para mejorar esta detección se emplea *cv::cornerSubPix*, que se encargará de ubicar estas esquinas en medidas de subpíxels[1].

La función que proporciona *OpenCV* y que calculará los parámetros intrínsecos de la cámara es *cv::stereoCalibrate(...)*. Tras obtener los parámetros deseados, se procederá a su almacenamiento en un fichero de tipo YAML. Para ello se usa un objeto de tipo *cv::FileStorage*.

El algoritmo 2, explica el procedimiento de calibración seguido con este conjunto de funciones.

[1] Un subpíxel consiste en la división del píxel de tal forma que un píxel no se toma como una medida entera, sino como una medida con valores intermedios.

Algoritmo 2 Algoritmo de calibración de Arges

Entrada: $vector < cv :: Point3f >> puntosReales$

Salida: Parámetros intrínsecos en archivo YAML.

1: $vector < vector < cv :: Point2f >> conjuntoEsquinas$;
2: $cv :: Mat\ matrizCamara[2]$;
3: $cv :: Mat\ coeficientesDistorsion[2]$;
4: **para** cada imagen i **hacer**
5: $vector < cv :: Point2f > esquinas$;
6: $cv :: findChessboardCorners(imagen[i], esquinas, ...)$;
7: $cv :: cornerSubPix(imagen[i], esquinas, ...)$
8: Añadir $esquinas$ a $conjuntoEsquinas$
9: **fin para**
10: $cv :: stereoCalibrate(puntosReales, conjuntoEsquinas, matrizCamara[0], matrizCamara[1], coeficientesDistorsion[0], coeficientesDistorsion[1], ...)$;
11: Guardar parámetros con $cv :: FileStorage$.

Se puede observar que en la función *cv::stereoCalibrate(...)* se obtiene la matriz fundamental estimada. Pueden utilizarse sus propiedades dentro de la geometría epipolar para conocer el **error cometido en la calibración**.

Según se ha explicado, puede encontrarse la proyección de un punto en una imagen a lo largo de la línea epipolar dada entre las dos cámaras (restricción epipolar). Además, si la calibración ha sido correcta, con las matrices de cámara extraídas y la matriz fundamental, debe cumplirse la ecuación 2.21 para cada uno de los puntos.

Para medir el error en la calibración, se realiza la computación de las líneas epipolares del par de cámaras utilizando la función *cv::computeCorrespondEpilines(...)*. En un sistema ideal, como se ha dicho, los puntos deben de estar contenidos a lo largo de estas líneas, pero como los cálculos son aproximados en la calibración, existirá una pequeña desviación. Teniendo esto en cuenta, se calcula la desviación acumulada para todos los puntos que se han utilizado en la calibración y finalmente se divide por el total de puntos utilizados. Esto proporciona el error promedio cometido al ejecutar la calibración.

Después de conocer el módulo de calibración, en la siguiente sección se explicará el módulo de Entrada/Salida del sistema.

5.2. Módulo de Entrada/Salida

El módulo de Entrada/Salida es el más "cercano" al usuario dentro del conjunto de módulos del sistema. Es el que se encarga de tomar los datos de entrada y de mostrar la salida al usuario.

5.2.1. Submódulo de entrada

La entrada al sistema está constituida por los frames vistos por la cámara. Estos frames serán capturados por la interfaz proporcionada por *OpenCV*, llamada *cv::VideoCapture*.

Todos los frames capturados por *OpenCV* son guardados en objetos de tipo *cv::Mat*. El código que inicia la captura de frames desde la cámara es el siguiente:

```
cv::VideoCapture vC;

if(!vC.open(0)){
  std::cerr << "Can not initialize video capture" << endl;
  return 0;
}
```

Una vez iniciada la captura de la cámara, se puede guardar el frame actual de la siguiente forma:

```
cv::Mat frameCaptured;
vC >> frameCaptured;
```

Puesto que el número de frames por segundo es elevado con respecto al requerimiento de conocer una nueva posición, se producirá una llamada al módulo de posicionamiento cada cierto número de frames procesados. De esta forma, además se evita una sobrecarga del sistema realizando demasiadas peticiones de cálculo de la posición, lo que iría en detrimento del objetivo de tener un sistema de tiempo real.

5.2.2. Submódulo de salida

El submódulo de salida es el que se encarga de mostrar por pantalla lo que el usuario ve. En la salida se muestra tanto lo que se está viendo a través de la cámara como el mapa indicador de la posición del usuario.

Este mapa indicador es construido utilizando la biblioteca OpenGL. A través de esta biblioteca es posible renderizar cualquier objeto 2D o 3D en la pantalla. Gracias a esta utilidad, se ha realizado el mapa incluyendo un plano en la pantalla, dibujado con OpenGL y cuya textura es el mapa de la figura 5.3, correspondiente al plano de la planta baja de la Escuela Superior de Informática.

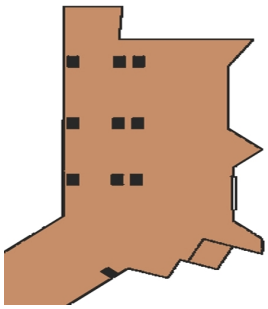


Figura 5.3: Plano de la planta baja de la Escuela Superior de Informática

La posición en tiempo real es indicada a través de un puntero, dibujado empleando primitivas de OpenGL. Este puntero está formado por un punto, que indica la posición actual del usuario, y una flecha, que indica la orientación actual.

En la siguientes secciones se explicarán las gestiones realizadas internamente dentro de Arges para que la posición y orientación estimadas sean recibidas por el submódulo de salida.

5.3. Módulo de Cálculo del Posicionamiento

El módulo de Cálculo de Posicionamiento es el encargado de servir de enlace entre los resultados proporcionados por los métodos de *tracking* y la salida dada en el mapa. Sus funciones son las siguientes:

- Se encarga de recibir las posiciones dadas por los tres métodos de *tracking* y proporciona una posición única basándose en la fiabilidad del método de *tracking* y en la trayectoria dada por el histórico de posiciones.

- Guarda los resultados más actuales en un histórico de posiciones para poder inferir con más exactitud la siguiente posición.

- Inicializa cada módulo de *tracking* empleando la factoría de métodos de *tracking*.

Algoritmo 3 Algoritmo de cálculo de la posición del usuario.

Entrada: PosTrackingConMarcas, PosTrackingEstéreo, PosTrackingDescriptores, PosActual.

Salida: PosiciónUsuario.

```
1: si PosTrackingConMarcas != NULL entonces
2:    PosiciónUsuario = PosTrackingConMarcas;
3: si no
4:    si Distancia(PosTrackingDescriptores, PosActual) <λ entonces
         PosiciónUsuario = PosTrackingEstéreo;
5:    si no
         PosiciónUsuario = Media(PosTrackingEstereo,PosTrackingDescriptores);
6:    fin si
7: fin si
8: Devolver PosiciónUsuario;
```

El cálculo de la posición definitiva se realiza siguiendo el algoritmo 3. Cuando el módulo de *tracking* con marcas proporciona una posición, dicha posición se considera definitiva, y es la que se proporciona como salida. Es cuando se dejan de observar marcas cuando se infiere la posición a partir de los otros dos métodos de *tracking*. Puede ocurrir que el módulo de posicionamiento con descriptores proporcione una distancia muy alejada de la última posición conocida, con lo que esta estimación es considerada inservible y se deshecha, utilizando la posición que ha dado el *tracking* estéreo. En caso de que el posicionador con descriptores de

una posición válida, se hallará una posición intermedia entre ésta y la que dé el posicionador estéreo, de esta forma se proporcionará una mayor exactitud al cálculo.

Cuando el módulo de cálculo tiene la posición definitiva, debe realizar dos acciones: en primer lugar, guardará la nueva posición en el histórico; en segundo lugar, proporcionará la nueva posición al módulo de Entrada/Salida para actualizar el mapa de posicionamiento. En la siguiente sección se explica en qué consiste el histórico de posiciones.

5.3.1. Submódulo de histórico de posiciones

En un sistema de este tipo es interesante realizar un almacenamiento de la evolución que va teniendo la trayectoria del usuario con vistas a una mayor exactitud en el posicionamiento. Durante el proceso de movimiento de la cámara se considera que no es posible un desplazamiento por encima de un determinado valor; es decir, en el funcionamiento normal del sistema el desplazamiento de la cámara no estará por encima de 1 metro por segundo, lo que reduce el rango de posiciones que puede alcanzar la cámara en un segundo (20 frames). Esta restricción es especialmente útil para el *tracking* con descriptores, el cual puede dar posiciones en cualquier lugar del mapa sino existen datos históricos.

En Arges, para el histórico de posiciones, se guarda un vector de las últimas 4 posiciones. Cuanto más reciente sea la posición, mayor peso se dará a la hora de la estimación.

Para terminar esta sección se explicará en qué consiste la factoría de métodos de *tracking* del sistema.

5.3.2. Factoría de métodos de *tracking*

La factoría de métodos de *tracking* ha sido implementada siguiendo el patrón fachada o *facade*. Con este patrón se proporciona una interfaz de alto nivel para un conjunto de clases en un subsistema, haciéndolo más fácil de usar. Este subsistema comprende los tres módulos de *tracking*. De esta forma se simplifica el acceso a estos módulos, ya que el cliente sólo se comunica con ellos a través de una única interfaz. Esta interfaz recibe el nombre de *TrackingMethodFactory*.

Además, en cada módulo de *tracking* se implementará el patrón *Singleton*, de tal forma

que solo pueda existir en ejecución una instancia de cada uno de los métodos de registro. En la figura 5.4 pueden observarse la forma de ambos patrones según su diagrama de clases.

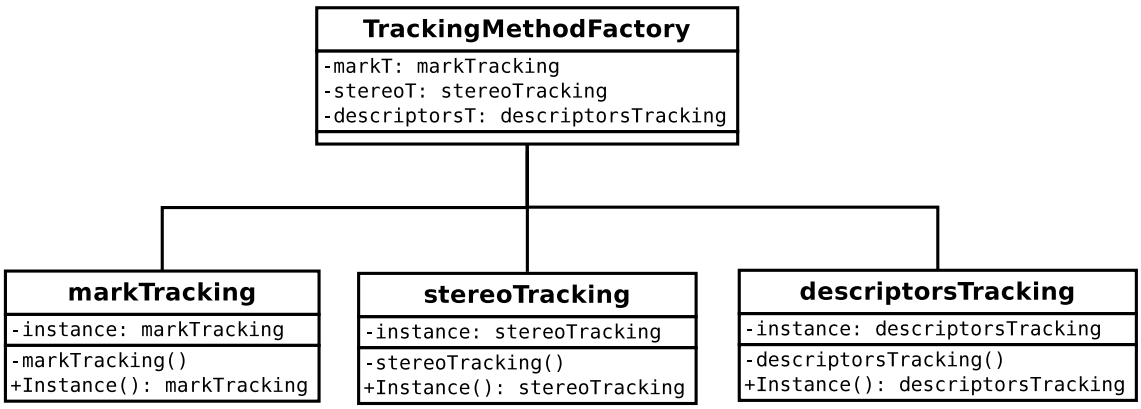


Figura 5.4: Patrones *Facade* y *Singleton* dentro del sistema de cálculo del posicionamiento.

En las siguientes secciones se explicará cómo ha sido implementado cada uno de los métodos de *tracking* dentro de Arges.

5.4. Módulo de *tracking* con marcas

El primero módulo de *tracking* que se ha implementado para este proyecto se basa en las marcas de *ARToolkit*. Se ha decidido incluir este método de *tracking* por tener las siguientes ventajas:

- Proporciona información bastante exacta acerca de la posición y rotación de la cámara con respecto de la marca que está visualizando el sistema.
- Funciona bien aun sin información histórica de la trayectoria que se ha seguido.

Por contra, se tiene que:

- No se desea llenar el escenario de actuación de marcas de *ARToolkit*, con lo que, a menor número de marcas, menos útil resultará este tipo de *tracking*.

- Aunque la cámara esté mirando hacia una marca, se requiere de una cierta cercanía a la misma para que sea detectada correctamente. Este problema puede subsanarse utilizando marcas más grandes, sin embargo, esto iría en contra de la política de modificar lo mínimo posible la estética del escenario.

- El exceso o defecto de luz sobre la marca o cerca de ella puede hacer que la detección no sea correcta.

5.4.1. Inicialización del sistema de *tracking*

Para que el sistema de *tracking* funcione, se deben proporcionar una serie de datos al sistema. Estos datos son:

- Parámetros de la cámara, incluidos en el fichero *"camera_para.dat"*.

- Dimensiones de la entrada de vídeo en píxels.

- Ficheros de los patrones de las marcas. Estos archivos tienen extensión *.patt* y le indican a *ARToolkit* la forma de una determinada marca.

El siguiente fragmento de código muestra cómo se realiza esta inicialización.

```
1  ARParam  wparam, cparam; // Intrinsic parameters of the camera
2
3  // Loading the intrinsic parameters of the camera
4  if(arParamLoad("./Tracking/camera_para.dat",1, &wparam) < 0)
5  std::cerr << "Exception loading the parameters of the camera\n" << endl;
6
7  arParamChangeSize(&wparam, xsize, ysize, &cparam);
8  arInitCparam(&cparam);// Init the camera with cparam
9
10 // Loading the mark
11 if((pattId0=arLoadPatt("./Tracking/patterns/simple.patt")) < 0)
12 std::cerr << "Exception loading the pattern\n" << endl;
```

Téngase en cuenta que las líneas 11 y 12 deben ser repetidas para cada una de las marcas que se utilicen en el sistema.

5.4.2. Detección de una marca

Cuando este módulo de *tracking* recibe un frame de vídeo, debe buscar una de las marcas en él. El proceso de detección de marcas en *ARToolkit* está basado en un algoritmo de detección de bordes. En la figura 5.5 se liustran los pasos que ejecuta internamente *ARToolkit* para detectar los bordes de la marca.

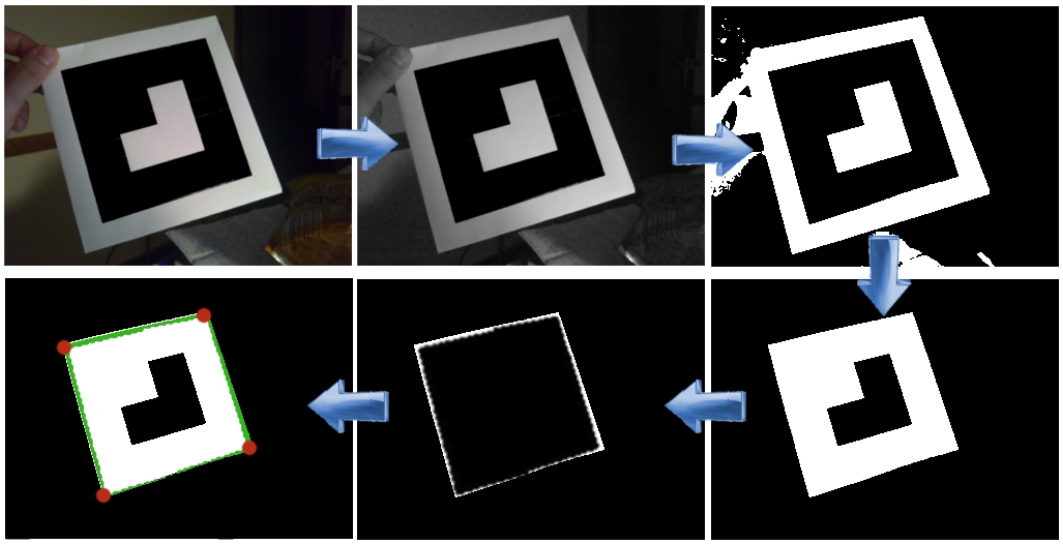


Figura 5.5: Pasos para la detección los bordes de una marca en ARToolkit

Tras convertir la imagen en escala de grises, ésta es binarizada. Posteriormente se extraen aquellas regiones de la imagen susceptibles de contener una marca y sobre ellas se aplica el algoritmo detector de bordes. Una vez se tienen identificada la presencia de una marca en un lugar determinado, se extrae el contenido del centro de la marca para conocer con qué patrón se corresponde la marca que se está visualizando.

Tras la detección de una marca, el programa conoce tanto las posiciones 2D de las aristas y vértices que la definen como el modelo de proyección de la cámara, de esta forma es posible obtener la matriz que transforme la posición de la marca en la posición de la cámara.

En el siguiente código se muestra cómo realizar la detección de una marca:

```
1 ARMarkerInfo *marker_info;
2 int marker_num, j, k;
3 double patt_trans[3][4]; // Mark transformation matrix
4 double p_width     = 120.0; // Wide of the pattern
```

```
5  double p_center[2] = {0.0, 0.0};   // Center of the pattern
6
7  // Detecting the mark in the captured frame
8  arDetectMarker(currentFrame.data, 100, &marker_info, &marker_num);
9
10 // Checking where the pattern is detected
11 // with more reliability
12 for(j = 0, k = -1; j < marker_num; j++) {
13   if(pattId0 == marker_info[j].id) {
14     if (k == -1) k = j;
15     else if(marker_info[k].cf < marker_info[j].cf) k = j;
16   }
17 }
18 if(k != -1) {   // The pattern has been detected...
19   // Obtaining the relative transformation between the mark and
20   // the real camera
21   arGetTransMat(&marker_info[k], p_center, p_width, patt_trans);
22 }
```

5.4.3. Obtención del posicionamiento

Una vez que la marca ha sido detectada, la función *arGetTransMat(...)* proporciona la matriz de transformación entre la cámara y dicha marca. Esta matriz se pasa a la función con el nombre *patt_trans*. En la figura 5.6 se puede ver la correspondencia entre la referencia de los ejes coordenados de la cámara y los de la marca.

La matriz de transformación de la cámara, T_{cam}, proporciona información tanto de rotación como de traslación de tal forma que:

$$T_{cam} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

Cada una de las marcas será ubicada en una posición del escenario, y esa posición se encuentra almacenada dentro del sistema de posicionamiento de Arges en coordenadas globales del mundo. Teniendo como ingredientes la posición 3D del centro de la marca y la matriz

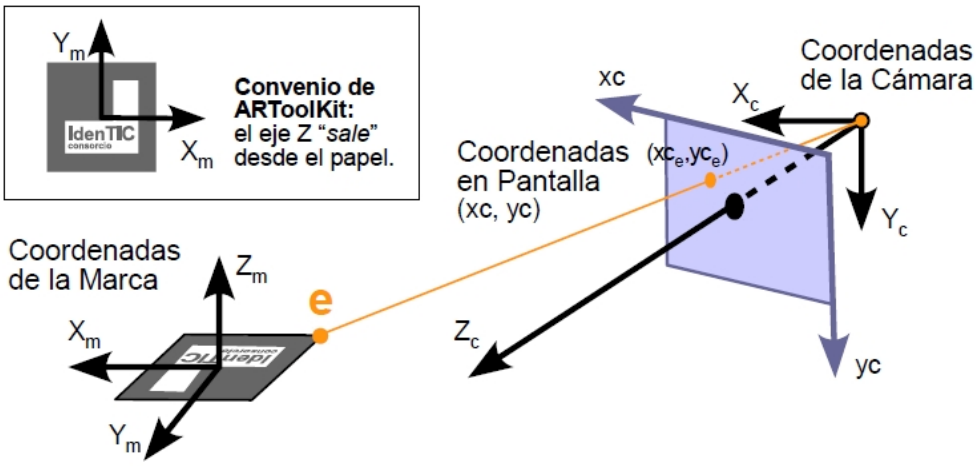


Figura 5.6: Sistema de coordenadas de la marca con respecto de la cámara en ARToolkit. Imagen: Carlos González Morcillo.

T_{cam}, y efectuando una transformación previa del sistema de referencia local cámara-marca al sistema de referencia global, se puede obtener la posición absoluta de la cámara dentro del escenario utilizando la siguiente ecuación:

$$\begin{bmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \\ 1 \end{bmatrix} = T_{cam} \times \begin{bmatrix} X_{marca} \\ Y_{marca} \\ Z_{marca} \\ 1 \end{bmatrix} \tag{5.2}$$

Esta posición será la proporcionada al módulo de cálculo del posicionamiento.

5.5. Módulo de *tracking* con descriptores visuales

Para paliar algunas de las deficiencias del método de *tracking* descrito anteriormente, se ha añadido este módulo, que tiene las siguientes ventajas:

- No requiere de la utilización de marcas, por tanto la estética del escenario no se ve afectada.

- Con la utilización de un histórico del posicionamiento es un sistema bastante exacto.
- Es menos sensible a la incidencia de la luz que el *tracking* con marcas.

Los inconvenientes que tiene este sistema de *tracking* son:

- Requiere hacer un directorio fotográfico relativamente amplio del escenario si se desea una gran exactitud en la estimación.
- Un cambio significativo en la estética del escenario hará que el *tracking* falle y se necesitará realizar un nuevo reportaje fotográfico de la zona que ha cambiado.

Como se ha visto anteriormente, el estándar MPEG-7 proporciona un conjunto muy amplio de descriptores visuales (ver Figura 2.13), de los cuales se han utilizado tres: ***Color Layout***, ***Edge Histogram*** *y* ***Scalable Color***. Para utilizarlos se ha utilizado la biblioteca *MPEG-7FexLib*.

En primer lugar se dará una visión general del funcionamiento de este sistema de *tracking*, cuyo funcionamiento difiere significativamente del de los otros módulos de *tracking* implementados en Arges. Se caracteriza porque funciona siguiendo dos fases: una fase *off-line* y otra *on-line*. La fase *on-line* se corresponde con la estimación de la posición propiamente dicha.

La parte *off-line* es un prerrequisito necesario para que el programa funcione. En primer lugar se debe tener un directorio de imágenes del entorno y una lista de posiciones y orientaciones asignadas a cada imagen. Partiendo de este directorio de imágenes, es necesario realizar la indexación de las características de cada imagen, almacenándolas en una base de datos. Estas características serán las que sean comparadas para realizar el *tracking*. El algoritmo 4 indica los pasos a seguir para realizar esta indexación.

Por otra parte, con la parte *on-line* es con la que se obtiene el posicionamiento estimado para una imagen. El algoritmo 5 muestra el funcionamiento general de este tipo de *tracking*.

Para comprender mejor la arquitectura de este módulo de *tracking*, se presenta un diagrama de clases en la figura B.2 (ver Anexo B).

Algoritmo 4 Algoritmo de indexación de imágenes en base de datos.

Entrada: Directorio con imágenes.

Salida: Base de datos con los valores de los descriptores de cada imagen del directorio almacenados.

1: Abrir directorio.
2: **para** cada imagen contenida en el directorio **hacer**
3: Extraer valores para el descriptor *Color Layout*.
4: Extraer valores para el descriptor *Edge Histogram*.
5: Extraer valores para el descriptor *Scalable Color*.
6: Insertar en la base de datos el nombre de la imagen y los valores extraidos de los tres descriptores.
7: **fin para**
8: Cerrar directorio.

Algoritmo 5 Algoritmo general de *tracking* con descriptores visuales.

Entrada: Imagen para ser comparada.

Salida: Posicionamiento estimado de la imagen.

1: Utilizando los valores almacenados en la base de datos para cada imagen, se crea un árbol de búsqueda.
2: Extraer valores para el descriptor *Color Layout*.
3: Extraer valores para el descriptor *Edge Histogram*.
4: Extraer valores para el descriptor *Scalable Color*.
5: Se buscan en el árbol las imágenes cuya distancia con respecto a la imagen de entrada se encuentre por debajo de un valor λ.
6: Las imágenes seleccionadas se ordenan de menor a mayor distancia.
7: Se selecciona la imagen más parecida según los parámetros de estimación.
8: Se busca en la lista la posición y rotación donde se tomó la imagen seleccionada y se devuelve.

5.5.1. Implementación de los descriptores visuales

La biblioteca MPEG-7FexLib es la encargada de abstraer las tareas de extracción de los valores para los descriptores visuales. Cada uno de los descriptores usados en el *tracking* tiene una serie de características diferentes entre sí. En esta sección se presentan las características propias de cada descriptor.

- **Color Layout** Para el *Color Layout* se utilizan tres vectores para tres tipos de coeficientes: coeficientes Y, coeficientes Cb y coeficientes Cr[2]. Los coeficientes de los tres vectores son números enteros, y se guardan 6 coeficientes Y, 3 coeficientes Cb y 3 coeficientes Cr.

- **Edge Histogram** Para el *Edge Histogram* se utiliza un solo vector con 80 coeficientes de tipo carácter.

- **Scalable Color** Para el *Scalable Color* se utilizan dos vectores de 256 coeficientes de tipo entero. Para asignar valores a los coeficientes se utiliza una matriz de cuantificación de tres posibles valores para cada coeficiente. Uno de los vectores almacena el valor propiamente dicho y el otro contiene en cada posición los valores 1 o 0, en función de si se acepta el valor del otro vector o no.

5.5.2. Almacenamiento en la base de datos

Como se ha visto en la anterior sección, existe cierta heterogeneidad en las estructuras de datos utilizadas para contener los valores de los descriptores. Además, existe el problema de que los vectores no pueden ser almacenados directamente en una base de datos MySQL, con lo que ha sido necesario buscar una forma de transformar estos datos para ser almacenados.

La forma de solucionar estos problemas ha sido almacenar cada uno de los vectores de datos como cadenas de texto, donde cada valor se encuentra separado del siguiente por un espacio en blanco. La figura 5.7 ilustra cómo sería esta transformación para el vector de coeficientes Y del *Color Layout*.

[2]En esta documentación no se entrará en los detalles de implementación de los coeficientes de descriptores. Para profundizar en este tema, ver la documentación del estándar MPEG-7.

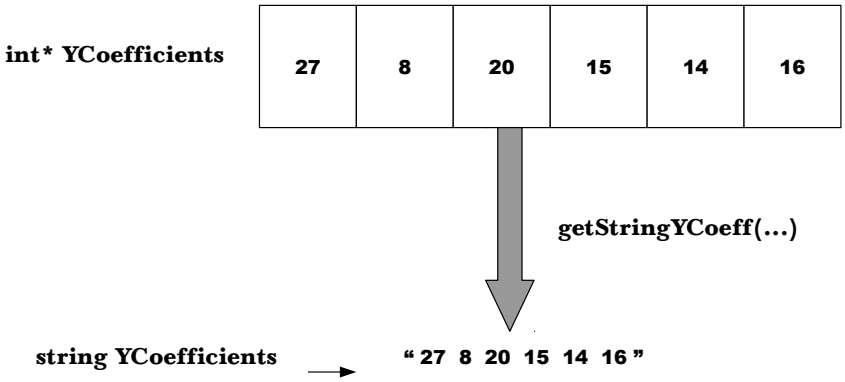


Figura 5.7: Transformación de los vectores de coeficientes en cadenas de texto

La base de datos para este módulo es sencilla, y solo alberga un tipo de tabla, correspondiente a los objetos *Image*. En la figura 5.8 se muestra el esquema de la tabla *Image* de la base de datos.

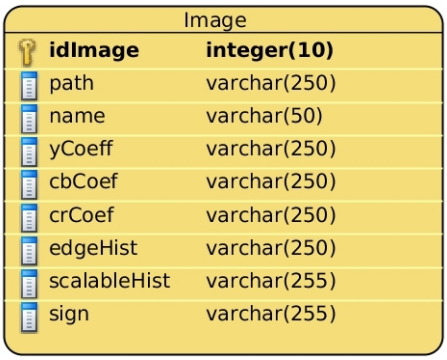


Figura 5.8: Tabla *Image* del esquema de base de datos

Para la implementación de la base de datos han sido utilizados dos tipos de patrones: *Singleton* y *DAO*.

- ***Singleton***: viene representado por la clase *Agent*. Aplicándolo en esta parte, se asegura que solo se tendrá una instancia de comunicación con la base de datos. Resultaría redundante tener más de una clase comunicándose con la base de datos puesto que solo existirá un módulo que trabaja con ella.

- ***Data Access Object(DAO)***: viene representado por la clase *ImageDAO*. Gracias a la aplicación de este patrón se consigue abstraer la tecnología de base de datos utilizada del resto del sistema. Por defecto, el sistema trabaja con base de datos *MySQL*, pero si en un futuro se deseara incluir otro tipo de tecnología, solo habría que hacer pequeños cambios en esta clase para que el resto del sistema funcionase correctamente.

5.5.3. Estructura de datos para agilizar la búsqueda

Para que este módulo de *tracking* funcione correctamente, es necesario tener datos de un conjunto muy grande de imágenes, los cuales están almacenados en una base de datos. Para agilizar la búsqueda, es necesario crear una estructura de datos que albergue toda la información y permita un acceso rápido a ella. Existen diversidad de tipos de estructuras de datos pensadas para contener información multimedia. Una de ellas es el llamado **Árbol de Voronoi**.

Según [30], *"el árbol de Voronoi es una estructura de datos para búsquedas por similitud en espacios métricos"*. A grandes rasgos, el árbol de Voronoi es un árbol n-ario, donde la inserción de los elementos se basa en la distancia existente entre ellos. Dicha distancia en el caso de este proyecto es el valor de similitud entre imágenes dado por los descriptores. Aunque existen versiones dinámicas del árbol de Voronoi, la implementación que se ha realizado es estática. Esto significa que en caso de adición de nuevos elementos, será necesaria la composición completa de dicho árbol.

El árbol de Voronoi está basado en la teoría del diagrama de Voronoi para búsquedas por similitud en espacios métricos. Se construye según el algoritmo del GNAT [31]. Básicamente su construcción consiste en tomar como hijos de un nodo aquellos cuya distancia sea menor con respecto al resto de nodos del mismo nivel, o dicho de otra forma, aquellos que se encuentren dentro del radio cobertor de su padre. De esta forma, a medida que se desciende en

el árbol, la distancia entre la imagen con la que estamos buscando y los nodos del árbol es más pequeña. Así se van eliminando conjuntos de imágenes en la búsqueda que por su similitud sabremos que no nos van a resultar útiles para obtener el resultado.

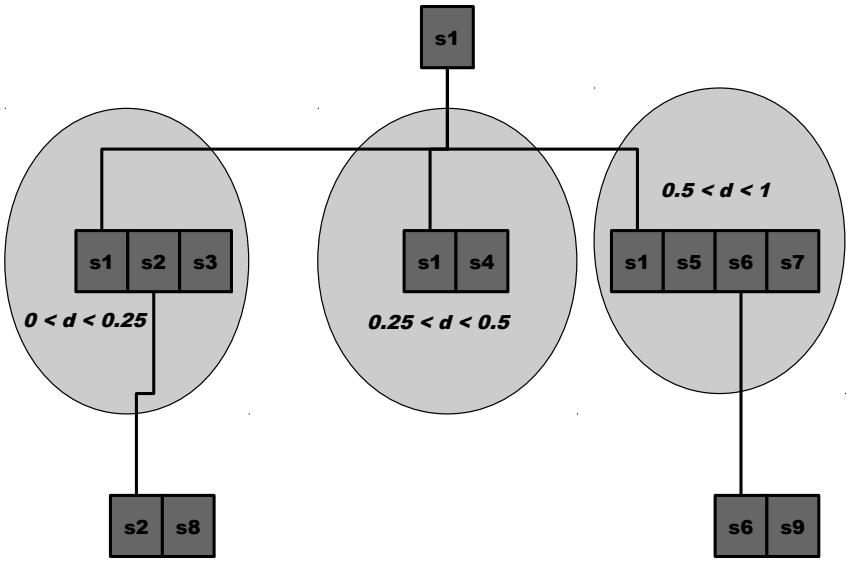


Figura 5.9: Ejemplo de construcción de un árbol de Voronoi

En la figura 5.9 se puede observar el aspecto que tendría un árbol de Voronoi con 9 imágenes. La raiz del árbol es una imagen que se elige como pivote de entre las del conjunto. A partir de aquí se elige el número de hijos que tendrá la raiz. Para el caso de Arges se ha tomado un número de hijos $N_{hijos} = \sqrt{NumeroImagenes}$. Cada hijo constituye un sub-árbol que contendrá el conjunto de imágenes contenido dentro del radio cobertor de la raiz. Se ha establecido que en el árbol de la izquierda se sitúen las imágenes con una distancia a la raiz entre 0 y 0.25. Para el árbol siguiente las situadas a una distancia entre 0.25 y 0.5. El árbol de la derecha guardará el resto. La construcción del árbol se va haciendo de forma recursiva hasta almacenar el conjunto completo de imágenes.

Teniendo esta estructura, la búsqueda se reduce a unas pocas comparaciones, reduciendo notablemente el tiempo de cómputo para buscar en un conjunto muy grande de imágenes.

5.5.4. Obtención de la posición y la orientación

Al buscar en el árbol de Voronoi, se obtiene un conjunto de imágenes que se encontrarán por debajo de un valor establecido por defecto. Ordenando las imágenes de menor a mayor distancia, se obtiene un conjunto de posiciones que el sistema ha estimado como cercanas a la posición actual.

Para obtener la posición y orientación de una imagen, éstas se encuentran almacenadas en un fichero con la siguiente forma:

```
CIMG0014.jpg -4.5 16.0 270
CIMG0015.jpg -6.0 18.0 315
```

En cada línea del fichero se dan los datos referentes a una imagen. En primer lugar el nombre de la imagen; después se proporcionan las coordenadas 2D, primero la coordenada X y después la Y. Por último se indica la rotación en grados.

Es importante tener en cuenta que las posiciones y orientaciones no son relativas a la última posición conocida, como se verá en la siguiente sección con el *tracking* estéreo, sino que son posiciones globales absolutas establecidas en el mapa del entorno previamente.

5.6. Módulo de *tracking* estéreo

El módulo de *tracking* estéreo ha sido implementado para solucionar los problemas encontrados con los dos módulos anteriores. Las principales ventajas de este método de *tracking* son:

- No requiere de marcas especiales en la escena.
- La cantidad de luz sobre la escena no influye en el funcionamiento.
- No requiere de datos preliminares sobre la escena almacenados como el *tracking* con descriptores.

Sin embargo, tiene algunos inconvenientes:

- Es menos preciso que los otros dos tipos de *tracking*.
- Proporciona un posicionamiento relativo, por lo que necesita de información de la posición inicial de la cámara para poder comenzar a funcionar correctamente.

En las siguientes secciones se explicarán los pasos que sigue el módulo para poder realizar el *tracking*. El algoritmo 6 da una panorámica general de su funcionamiento [32].

Algoritmo 6 Algoritmo de *tracking* estéreo.

Entrada: Imagen_izquierda, Imagen_derecha.

Salida: Nueva posición de la cámara.

1: Extraer parámetros intrínsecos de archivo YAML.
2: Extraer *landmarks* de la imagen izquierda.
3: Utilizar *optical flow* para obtener puntos en la imagen derecha.
4: Calcular matriz fundamental.
5: Calcular matriz esencial.
6: Extraer matriz de rotación y vector de traslación entre cámaras.
7: Triangular y proyectar los *landmarks* en 3D.

5.6.1. Extracción de parámetros intrínsecos

Antes de proceder con el cálculo del *tracking*, es necesario disponer de datos previos acerca del sistema. Estos datos se encuentran almacenados en el archivo YAML que produjo el módulo de calibración.

Gracias al uso de la interfaz *cv::FileStorage*, resulta muy sencillo extraer cualquier dato que se haya guardado en el fichero YAML. Para abrir el fichero se utiliza la siguiente orden:

```
1 cv::FileStorage fs(intrinsic_filename, CV_STORAGE_READ);
```

Después de abrir el fichero, la toma de datos se realiza introduciendo en una variable u objeto el elemento que se desea, utilizando el identificador con el que ha sido guardado en el fichero YAML. Un ejemplo de toma de datos puede verse a continuación:

```
cv::Mat cameraMatrix[2], distCoeffs[2];
fs["M1"] >> cameraMatrix[0];
fs["D1"] >> distCoeffs[0];
fs["M2"] >> cameraMatrix[1];
fs["D2"] >> distCoeffs[1];
```

En el ejemplo se ve como se han creado dos vectores con matrices, uno para almacenar las matrices de cámara y otro para almacenar los coeficientes de distorsión. Además de estos elementos, en el módulo es necesario extraer el valor de medida del lado de un cuadrado del patrón de calibración y las dimensiones de los frames que se reciben.

5.6.2. Extracción de *landmarks* y *optical flow*

En cada llamada al módulo de *tracking* estéreo se tiene una imagen de cámara izquierda y otra de cámara derecha. Ocurre que a lo largo del paso del tiempo, lo que en una llamada se consideraba como segunda imagen, pasará a ser la primera imagen en la siguente llamada. Este hecho es aprovechable para reducir el tiempo de cálculo con algoritmos de *optical flow*.

La extracción de *landmarks* se produce en la primera de una serie de imágenes que reproducen el movimiento de la cámara. A partir de la primera extracción, se buscará la presencia de estos puntos en las siguientes imágenes hasta que el número de puntos encontrados está por debajo de un límite establecido.

Para la extracción de *landmarks* se ha optado por utilizar el algoritmo **Shi-Tomasi**.

```
cv::goodFeaturesToTrack(frame1, *features, count, QUALITY_LEVEL, MIN_DISTANCE);
cv::cornerSubPix(frame1,*features,cv::Size(15,15),cv::Size(-1,-1), cv::TermCriteria(
    CV_TERMCRIT_ITER | CV_TERMCRIT_EPS, 20, 0.03));
```

La función *cv::goodFeaturesToTrack(...)* se encarga de aplicar el algoritmo Shi-Tomasi sobre la imagen de nombre *frame1* y de guardar los puntos en el vector de nombre *features*. Para una mayor precisión en las coordenadas de los puntos se utiliza la función *cv::cornerSubPix(...)*. En la imagen 5.10 se puede observar los puntos extraidos en una imagen con el algoritmo Shi-Tomasi.

Por otro lado, para el *optical flow* se utiliza la implementación que hace OpenCV del algoritmo de Lucas-Kanade piramidal.

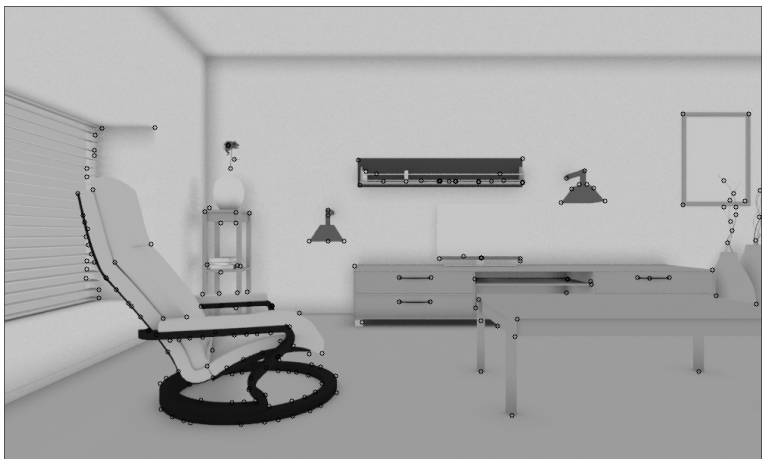


Figura 5.10: Puntos característicos extraidos con el algoritmo Shi-Tomasi

```
cv::calcOpticalFlowPyrLK(frame1,frame2,features1,*features2,       *
    optical_flow_found_feature, *optical_flow_feature_error, optical_flow_window, 5,
    optical_flow_termination_criteria, 0, 0);
```

El algoritmo nos proporciona los puntos del frame1 que han sido vistos en el frame2. El vector *optical_flow_found_feature* contiene un 1 en el caso de que se haya encontrado una asociación para un punto en la posición i, o un 0 en caso contrario. El vector *optical_flow_feature_error* indica el error cometido al encontrar la asociación entre dos puntos.

5.6.3. Cálculo de la matriz fundamental y la matriz esencial

Una vez en posesión de los puntos asociados entre las dos imágenes, se está en condiciones de estimar la matriz fundamental del par de cámaras. Existen un conjunto de algoritmos con el que estimar esta matriz [33]:

- Algoritmo de 7 puntos.
- Algoritmo de 8 puntos.
- Algoritmo RANSAC.
- Algoritmo LMedS.

Para este caso, el algoritmo de 7 puntos no sería válido, puesto que se tiene un conjunto de puntos mucho mayor que 7, que es el número de puntos que necesita el algoritmo. Tras realizar diversas pruebas, se decidió optar por el algoritmo LMedS, que es el que mejor estimación de la matriz fundamental consigue dar. Para ponerlo en marcha es necesario usar la siguiente función:

```
F = cv::findFundamentalMat((cv::Mat)features1, (cv::Mat)features2,CV_FM_LMEDS,0.01, 0.9999);
```

Siendo F la matriz fundamental resultante. Pasando a la función el parámetro *CV_FM_LMEDS* se está calculando la matriz utilizando el algoritmo LMedS.

Una vez se tiene la matriz fundamental, y teniendo las matrices de cámara procedentes de la calibración, es posible calcular la matriz esencial utilizando la ecuación 2.20. Despejando E, quedaría:

$$E = M_r^T F M_l \tag{5.3}$$

Lo que traducido a código sería:

```
cv::Mat E = cameraMatrix[1].t()*F*cameraMatrix[0];
```

5.6.4. Extraer matriz de rotación y vector de traslación

Una vez se tiene la matriz esencial, extraer la matriz de rotación y el vector de traslación es relativamente sencillo. Basta con calcular la descomposición SVD. A través de esta descomposición se obtiene:

$$E = UDV^T \tag{5.4}$$

Utilizando las matrices que nos ha dado la descomposición se tiene que la matriz de rotación es:

$$R = U \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} V^T \tag{5.5}$$

A su vez, es posible obtener una matriz que contiene los valores del vector de traslación:

$$T = V \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} V^T \tag{5.6}$$

Esta matriz tiene la siguiente forma:

$$T = \begin{pmatrix} 0 & -T_z & T_y \\ T_z & 0 & -T_x \\ -T_y & T_x & 0 \end{pmatrix} \tag{5.7}$$

El código OpenCV que realiza estos cálculos es el siguiente:

```
1  cv::SVD::compute(E,D,U,Vt);
2
3  cv::Mat I = cv::Mat::zeros(3,3,CV_64F);
4  I.at<double>(0,1) = -1;
5  I.at<double>(1,0) = 1;
6  I.at<double>(2,2) = 1;
7
8  R = U*I*Vt;
9
10 I.at<double>(0,1) = 1;
11 I.at<double>(1,0) = -1;
12 I.at<double>(2,2) = 0;
13
14 T = Vt.t()*I*Vt;
15
16 cv::Mat Tvect(1,3,CV_64F);
17 cv::Rodrigues(T,Tvect);
```

En la línea 1, puede verse como utilizar la descomposición SVD con OpenCV. Basta con proporcionar la matriz de origen, que en este caso es E, y las matrices fruto de la descomposición serán guardadas en D, U y Vt. Por otra parte, es importante la transformación realizada en la línea 17. La función *cv::Rodrigues(T,Tvect)* permite traducir la matriz de traslación que

se obtiene de la matriz esencial en un vector de traslación de 3 componentes, que serán los valores útiles para saber la traslación entre las dos cámaras.

Teniendo la matriz de rotación, el vector de traslación y la posición inicial de la cámara, el módulo de *tracking* es capaz de proporcionar la nueva posición de la cámara aplicando ambas transformaciones.

Por otra parte, con vistas a la elaboración de un mapa 3D del entorno, es posible calcular las posiciones 3D de los *landmarks* a través de un algoritmo de triangulación, el cual se explica en la siguiente sección.

5.6.5. Triangulación y proyección de los *landmarks*

Según la teoría de la geometría epipolar, es posible hallar las coordenadas 3D de un punto en el espacio a partir de dos vistas diferentes del mismo punto, tal y como se puede observar en la figura 5.11.

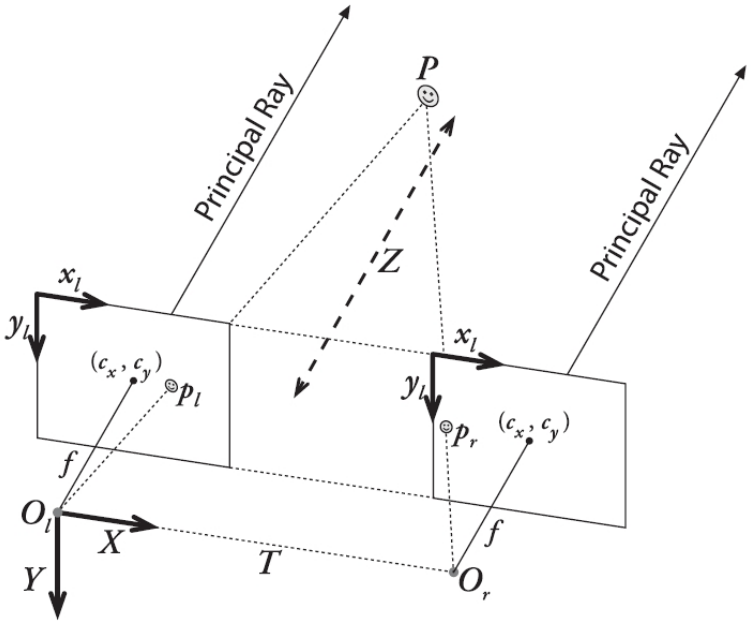


Figura 5.11: Proyección de un punto en el espacio a partir de dos vistas esteroscópicas del mismo. Imagen: *Learning OpenCV* [19].

Para obtener las coordenadas homogéneas del punto p se utilizan las siguientes ecuaciones:

$$p = \begin{pmatrix} x & y & 1 \end{pmatrix} \tag{5.8}$$

$$\frac{x = (u - u_0)}{s_x f_x} \tag{5.9}$$

$$\frac{y = (v - v_0)}{s_y f_y} \tag{5.10}$$

Siendo:

- u y v, las coordenadas del punto en la imagen.
- u_0 y v_0, las coordenadas del punto principal (centro de la imagen). Se obtiene de la matriz de cámara.
- $s_x f_x$ y $s_y f_y$, los productos de distancia focal por tamaño del píxel. Se obtiene de la matriz de cámara.

Para obtener la triangulación, es necesario trazar dos rayos que vayan desde el centro óptico de la cámara hasta el punto de la imagen donde se proyecta el punto 3D. El punto de intersección entre ambos rayos coincidirá entonces con la posición 3D del punto.

Ocurre que, debido a que la alineación de las cámaras no será perfecta, los rayos no intersectarán en la mayoría de ocasiones. Por esta razón hay que buscar algún artificio que aproxime las coordenadas del punto 3D. Se toma entonces aquel punto en el que las rectas están más cercanas entre sí y se traza una recta perpendicular a ambos rayos, w. El punto medio que pasa por la recta w y está entre los rayos trazados es el punto 3D que se estaba buscando. Este método se conoce como método del punto-medio [34]. La figura 5.12 ilustra este escenario.

Siendo r_l y r_r los rayos trazados para las cámaras izquierda y derecha respectivamente, w se hallaría realizando el producto vectorial de los vectores directores de ambas rectas. Teniendo estos tres elementos será necesario resolver el sistema de ecuaciones dado por:

$$r_l + w = r_r \tag{5.11}$$

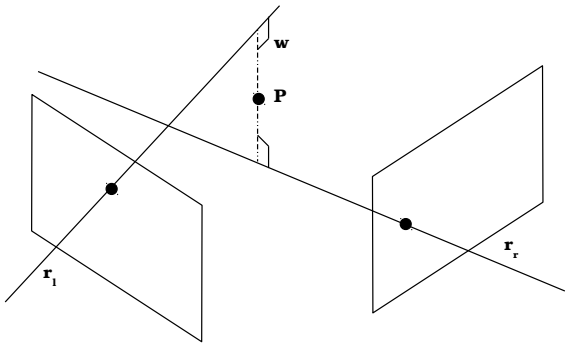


Figura 5.12: Triangulación de un punto 3D con cámaras mal alineadas

Una vez resuelto el sistema, el punto 3D, P se calculará mediante la siguiente ecuación:

$$\frac{P = r_l + w}{2} \tag{5.12}$$

Capítulo 6

Evolución y costes

A lo largo de este capítulo se irán mostrando las etapas por las que ha transcurrido el desarrollo de este proyecto. Se mostrarán los hitos que se han ido cumpliendo y qué obstáculos han ido salvándose hasta llegar al producto final. También se detallará el coste económico estimado.

6.1. Evolución del proyecto

6.1.1. Concepto del software

En el laboratorio *Oreto* se ha venido poniendo en los últimos años especial interés en el desarrollo de proyectos en el ámbito de la RA. Proyectos de este tipo son HESPERIA y ElCano. Este último proyecto ha constituido el motor motivador para el desarrollo de Arges.

La motivación ha venido dada por la necesidad de buscar nuevas formas de posicionamiento para el proyecto ElCano, debido a que, en el momento en el que se establecieron las bases de Arges, el único sistema de *tracking* visual en el que se basaba ElCano era a través de marcas, con las limitaciones que esto conllevaba.

Establecida la motivación del proyecto, hubo una etapa de análisis de requisitos para el mismo, que se explica a continuación.

6.1.2. Análisis de requisitos

Después de establecer los posibles problemas a solucionar en el ámbito en el que se deseaba realizar el proyecto Arges, hubo unos 15 días de análisis para establecer el alcance del mismo.

Fruto de este tiempo de análisis, se realizó una propuesta inicial del proyecto, en la que se establecieron una serie de requisitos básicos que debía cumplir el sistema. Estos requisitos básicos eran:

- Se emplearía un sistema de *tracking* sin marcas, analizando las estructuras planas de la escena.
- Debería constituir un sistema en tiempo real.
- El usuario visualizaría información de posicionamiento en la pantalla.

Tras establecer estos requisitos, hubo un periodo de revisión del estado del arte y de estudio y aprendizaje de las tecnologías de RA y gráficos por computador, tanto de los sistemas de *tracking* con marcas como de las alternativas de *tracking* sin marcas; este periodo de aprendizaje se prolongó durante unos dos meses, ya que los conocimientos en esta materia antes de comenzar el proyecto eran muy escasos.

Así mismo, se profundizó en el funcionamiento de la biblioteca *OpenCV*, puesto que constituiría la base software del proyecto.

Con cada uno de los prototipos creados durante el aprendizaje de las diferentes bibliotecas, se fueron elaborando pequeñas memorias, que posteriormente han sido utilizadas para redactar esta documentación.

Tras esta etapa de análisis de requisitos y aprendizaje, se fueron dando una serie de iteraciones, a través de las cuales ha ido evolucionando el proyecto hasta obtener el producto final.

A continuación se explican con detalle cada una de las iteraciones que se han realizado durante el desarrollo de Arges.

6.1.3. Iteración 1

A lo largo de esta iteración, en primer lugar, se realizó un análisis del funcionamiento de los sistemas de *tracking* con marcas. Para probar su funcionamiento se realizaron diferentes ejemplos con la biblioteca *ARToolkit*, que constituye la tecnología usada en el proyecto ElCano para llevar a cabo este tipo de registro.

Una vez afianzado el funcionamiento de la biblioteca, se creó el módulo de *tracking* con marcas, que constituiría el primero de los sistemas de *tracking* del sistema.

Tras la realización de este módulo se fueron realizando pruebas unitarias del mismo, en primer lugar en entorno virtual y más tarde se pasó a probarlo en el escenario real.

- **Entregable**

  Con esta iteración se obtuvo una primera versión del sistema en la que se recibían imágenes de una cámara y se obtenían datos de posicionamiento sin visualización.

6.1.4. Iteración 2

Para comenzar con la implementación del *tracking* sin marcas, se propuso el desarrollo de un sistema de *tracking* basado en la comparación de las características de la imagen.

- **Hito 1**

  El objetivo de este primer hito consistió en ver hasta qué punto podía resultar útil incorporar tanto a Arges como a ElCano un módulo de registro de este tipo. Por ello se buscó la realización de un prototipo con el que poder realizar pruebas. Se partió de una aplicación de comparación de imágenes realizada en Java con la biblioteca LIRE, en la que se usaban dos tipos de descriptores del estándar MPEG-7: *Color Layout* y *Edge Histogram*. Esta aplicación se adaptó para poder funcionar como posicionador y se procedió a su incorporación con el código de ElCano.

  Debido a la diferencia de lenguajes de programación, fue necesario establecer una jerarquía Cliente-Servidor entre el equipo que ejecutara la aplicación de ElCano (Cliente) y el que ejecutara el posicionador con descriptores (Servidor). Ambos equipos man-

tenían su comunicación via Wifi y las diferentes aplicaciones utilizaban *ZeroC-Ice* como *middleware* para el intercambio de datos .

Tras realizar pruebas de campo, se pudo comprobar la utilidad del sistema, y se decidió continuar con su desarrollo.

- **Hito 2**

  Aunque la primera versión del posicionador con descriptores funcionaba, la precisión obtenida con el mismo no era buena, ya que podía proporcionar posiciones muy alejadas de la posición desde la que se efectuaba la petición.

  Para mejorar el comportamiento del sistema de posicionamiento con descriptores realizado con LIRE, se incorporó un histórico de posiciones que apoyara la estimación.

  Una vez incorporado este histórico, se volvieron a realizar un conjunto de pruebas de campo, donde se observó una sensible mejora en el comportamiento del posicionador con descriptores.

- **Hito 3**

  El hecho de tener que mantener la comunicación entre el Cliente y el Servidor a través de Wifi conllevaba una serie de problemas tales como retrasos a la hora del envío y recepción de datos o pérdida de comunicación entre ambos entes. Por esta razón se propuso buscar una alternativa en el mismo lenguaje que el código de ElCano (C++) para eliminar la arquitectura Cliente-Servidor.

  Después de una revisión de alternativas, se decidió utilizar la biblioteca MPEG-7FexLib, que contenía las funciones necesarias para la extracción y tratamiento de descriptores visuales utilizando *OpenCV* con el lenguaje C++. Con ella se realizó una nueva versión del prototipo de posicionador con descriptores partiendo del que ya se tenía y eliminando la arquitectura Cliente-Servidor. Con la remodelación del posicionador con descriptores se constituyó el segundo método de *tracking* del sistema Arges.

  Tras realizar una serie de pruebas en un entorno virtual para obtener la configuración óptima de los parámetros de funcionamiento del posicionador, se realizaron pruebas en escenario real.

- **Entregable**

  Después de esta iteración, se obtuvo el sistema de *tracking* con descriptores, que funcionaba en un módulo independiente del sistema con marcas. Con él se podían obtener datos de posicionamiento sin visualización.

6.1.5. Iteración 3

Durante esta iteración se buscaron diferentes alternativas para la implementación del *tracking* estéreo. En un principio no existía conocimiento de la vía de implementación de un sistema de este tipo, pues existían pocos sistemas que utilizaran algo similar.

- **Hito 1**

  Se realizó un estudio del sistema PTAM, de George Klein [35] para establecer las bases del sistema de *tracking*. Después de un profundo análisis del sistema, se buscaron formas de utilización de visión estereoscópica para la implementación de un sistema de registro.

  La primera prueba de un sistema de este tipo se basaba en el cálculo de homografías y aún no incorporaba técnicas de visión estereoscópica. Este prototipo no produjo resultados satisfactorios, ya que solo se podía conseguir información de rotación entre imágenes, por lo que se siguieron investigando nuevas formas de implementación.

- **Hito 2**

  Se profundizó en el estudio de la estructura para "ego-motion" mediante la utilización de una única cámara y la simulación de un sistema estereoscópico. La primera aproximación fue realizando un prototipo que calculaba el posicionamiento a través de la reproyección de puntos 3D mediante mapas de disparidad [36]. Este prototipo daba resultados satisfactorios de posicionamiento, pero requería de los parámetros extrínsecos de las cámaras estéreo en tiempo real. Esto suponía un coste computacional muy alto como para ser incorporado en un sistema de tiempo real, por lo que este prototipo también se deshechó.

- **Hito 3**

  Partiendo del prototipo anterior, se buscó la forma de eliminar la necesidad de hallar los parámetros extrínsecos de la cámara en tiempo real. De esta forma, se implementó el módulo de *tracking* estéreo definitivo de Arges, basado en la estimación de la matriz fundamental y en el cálculo del resto de parámetros para la estimación del posicionamiento a partir de la misma solamente utilizando los parámetros intrínsecos y algoritmos de detección de puntos y *optical flow*.

  Tras ejecutar una serie de pruebas en escenario virtual, se probó el sistema en escenario real con resultados satisfactorios.

- **Entregable**

  Tras esta tercera iteración, se consiguió tener el módulo de *tracking* estéreo independiente del resto de módulos ya implementados y probados.

6.1.6. Iteración 4

A lo largo de esta iteración se realizó la integración de los tres módulos en un sistema que constituirá el conjunto de todo el sistema Arges, exceptuando el módulo de Entrada/Salida.

Se decidió crear una *Factoría de Métodos de Tracking* para un mejor control del módulo de cálculo del posicionamiento. Así mismo se incorporó el histórico de posiciones.

El conjunto se probó en escenario real directamente.

- **Entregable**

  Sistema Arges integrado a excepción del módulo de Entrada/Salida.

6.1.7. Iteración 5

Se realizó un estudio de cómo mostrar al usuario el posicionamiento y qué tecnología utilizar para ello. Finalmente se optó por utilizar OpenGL para el submódulo de salida, el cual, se encargaría del dibujado del mapa, junto con el vídeo en tiempo real recibido desde la cámara del sistema y la interfaz de demostración de funcionamiento.

Así mismo, como método de entrada se creó un submódulo que facilitase la apertura de la cámara según el paradigma de *OpenCV*.

Ambos submódulos se integraron e incorporaron al resto del sistema Arges, constituyendo el sistema completo.

- **Entregable**

  Sistema Arges completo, con interfaz de usuario preparada para un funcionamiento normal y la utilización de cada módulo de posicionamiento por separado.

6.2. Recursos y costes

En esta sección se realiza una estimación del coste económico que ha supuesto el desarrollo del proyecto y el tiempo estimado que ha sido necesario para llevarlo a cabo.

Como ha podido observarse en la sección anterior, durante las primeras iteraciones del proyecto se fueron alternando tareas de investigación y aprendizaje con tareas de implementación. Por esta razón es dificil estimar el tiempo dedicado íntegramente a escritura de código.

Recurso	Cantidad	Coste
Ordenador *Acer Aspire 6930*	1	859€
Sueldo programador	1	15000€
Webcam *Logitech Sphere AF*	1	115€

Tabla 6.1: Desglose del coste económico de Arges.

El desarrollo del proyecto ha sido realizado entre octubre de 2010 y enero de 2012, incluyendo el periodo de documentación del mismo. De este lapso de tiempo se estima haber empleado unas 500 horas puramente en materia de programación, divididas en unas 350 horas dedicadas en las tres primeras iteraciones y 150 horas dedicadas en la integración de todo el proyecto. Se ha estimado el sueldo de un programador en 30€/hora[1].

[1] Sueldo medio extraído de la web http://www.infolancer.net.

En la tabla 6.1 se muestra un desglose del coste económico que supondría el desarrollo del proyecto Arges. En ella se ha incluido tanto el sueldo empleado para el pago de el programador o programadores en función de las horas empleadas como el equipo hardware que ha sido requerido para la realización del proyecto.

Capítulo 7

Resultados

A lo largo de este capítulo se irán mostrando diferentes pruebas realizadas sobre cada uno de los módulos de *tracking* desarrollados en este proyecto. Gracias a estas pruebas se podrá observar el comportamiento de los diferentes métodos de posicionamiento en diferentes situaciones y, de esta forma, poder establecer formas de mejorar dicho comportamiento.

Las pruebas han sido divididas en dos grupos. Por un lado se han efectuado pruebas de comprobación de resultados. En ellas se ha ejecutado cada módulo en escenario virtual y en escenario real por separado y se han comprobado y analizado los diferentes resultados obtenidos. Por otra parte, se han efectuado pruebas de *profiling* con el objetivo de analizar diferentes pruebas de rendimiento durante la ejecución de los módulos por separado e integrados.

7.1. Pruebas de *tracking* con marcas

En esta sección se mostrarán y comentarán los resultados obtenidos al utilizar el módulo de *tracking* con marcas por separado. Las pruebas se han hecho en dos escenarios diferentes, uno virtual y otro real. En primer lugar se muestran las pruebas realizadas en el escenario virtual.

7.1.1. Pruebas en entorno virtual

Para estas pruebas se ha utilizado un escenario creado en *Blender*. En este escenario se ha colocado una marca como las que se utilizarán en el entorno real y se han efectuado diversas pruebas de funcionamiento variando algunos parámetros: distancia con respecto de la marca, tamaño de la marca y ángulo de visionado de la marca.

En estas pruebas se ha medido el grado de percepción de la marca que tiene el sistema, puesto que una vez percibida el posicionamiento es absoluto. Con ellas se intenta estimar cuál es la medida idónea de la marca para que el sistema se encuentre reconociéndola el mayor tiempo posible y el ángulo a partir del cual la marca deja de ser percibida para considerar lugares estratégicos del entorno real donde ubicarlas.

7.1.1.1. Pruebas con marca de frente

En esta primera prueba se ha analizado el comportamiento del sistema utilizando diferentes tamaños de marca y para cada tamaño diferentes distancias con respecto a la misma. De esta forma, se podrá comprobar a partir de qué tamaño y distancia el sistema deja de percibir que hay una marca. En estas pruebas no se incluirá la posición que da el *tracking*, solo se verá en qué dimensiones este módulo funcionará. Las pruebas han sido realizadas con diferentes formas de marca para ver cuáles son aquellas marcas que se visualizan mejor.

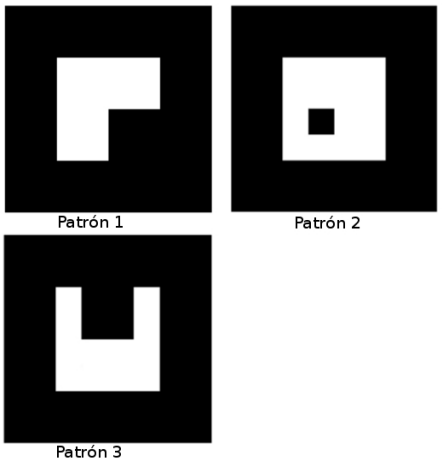


Figura 7.1: Patrones de marca utilizados en las pruebas

Debido a que *Blender* indica las medidas en sus propias unidades, es necesario establecer una correspondencia entre éstas y la medida en el mundo real. Se estima que 10 unidades de *Blender* equivalen a 1 metro del mundo real. En las tablas que contienen los resultados de las pruebas, las medidas se proporcionarán en medidas del mundo real para una mayor claridad.

Los patrones de marca que han sido utilizados en las pruebas pueden observarse en la figura 7.1.

En la tabla 7.1 pueden verse los resultados obtenidos con el patrón número 1.

Ancho(cm)	**Distancia(m)**	**¿Marca encontrada?**
4	3.3	No
4	2.65	No
4	1.68	No
4	0.67	Sí
8	3.3	No
8	2.65	No
8	1.68	No
8	0.67	Sí
12	3.3	No
12	2.65	No
12	1.68	Sí
12	0.67	Sí
16	3.3	No
16	2.65	No
16	1.68	Sí
16	0.67	Sí

Tabla 7.1: Resultados para el *tracking* con marcas en entorno virtual. Patrón 1 visto de frente.

Según los datos, es necesario que la marca con el patrón 1 se encuentre a menos de dos metros y medio para poder ser visualizada teniendo una medida de entre 12cm y 16cm.

En la tabla 7.2 pueden verse los resultados obtenidos con el patrón número 2. En este caso, la marca tiene una peor detección que con el patrón 1. Se necesita que la cámara esté a menos de un metro y medio para tener posibilidades de que la marca sea detectada.

Ancho(cm)	**Distancia(m)**	**¿Marca encontrada?**
4	3.3	No
4	2.65	No
4	1.68	No
4	0.67	No
8	3.3	No
8	2.65	No
8	1.68	No
8	0.67	No
12	3.3	No
12	2.65	No
12	1.68	No
12	0.67	Sí
16	3.3	No
16	2.65	No
16	1.68	No
16	0.67	Sí

Tabla 7.2: Resultados para el *tracking* con marcas en escenario virtual. Patrón 2 visto de frente.

En la tabla 7.3 pueden verse los resultados obtenidos con el patrón número 3. Según estos datos, el patrón 3 se comporta de forma similar al patrón 1 aunque con 12cm de ancho de marca, el sistema no es capaz de detectar el patrón a 1.68m de distancia.

Ancho(cm)	**Distancia(m)**	**¿Marca encontrada?**
4	3.3	No
4	2.65	No
4	1.68	No
4	0.67	Sí
8	3.3	No
8	2.65	No
8	1.68	No
8	0.67	Sí
12	3.3	No
12	2.65	No
12	1.68	No
12	0.67	Sí
16	3.3	No
16	2.65	No
16	1.68	Sí
16	0.67	Sí

Tabla 7.3: Resultados para el *tracking* con marcas en escenario virtual. Patrón 3 visto de frente.

Tras estas primeras pruebas, se puede concluir que la detección de marcas, además de las dimensiones del escenario, depende de la forma de la marca. Cuanto más simple sea la marca, más sencilla de visualizar la resultará al sistema. Por otra parte, es necesario que las marcas tengan al menos 12cm de ancho para asegurar una correcta detección en una distancia inferior a un metro y medio.

Las siguientes pruebas se realizan para comprobar en qué ángulos de visión el sistema deja de visualizar correctamente una marca. Para ello se utilizará un patrón simétrico como el patrón 3 y otro que no tenga simetría, como el patrón 1. Todas estas pruebas se realizarán con un tamaño de marca de 16cm y medio metro de distancia aproximadamente entre la cámara y la marca. Se efectuará una comprobación de detección cada 15º.

La tabla 7.4 muestra las pruebas hechas para el patrón 1. De aquí puede observarse una marca no-simétrica puede ser observada sin problemas con una rotación de hasta 75º a una distancia de medio metro.

Ángulo	**¿Marca encontrada?**
15º	Sí
30º	Sí
45º	Sí
60º	Sí
75º	Sí

Tabla 7.4: Resultados para el *tracking* con marcas en escenario virtual. Patrón 1 visto desde diferentes ángulos.

A continuación se muestran en la tabla 7.5 los datos recogidos con el patrón número 3 y diferentes ángulos de visionado. De aquí puede extraerse que, al igual que ocurría con el patrón 1, la marca es visualizada sin problemas con un ángulo de hasta 75º.

Tras la realización de estas pruebas, se procederá a mostrar algunas pruebas en un entorno real.

7.1.2. Pruebas en entorno real

En estas pruebas se ha ubicado una marca en diferentes lugares de la planta baja de la ESI. Se ha procurado que los lugares de ubicación de la marca tengan diferencias luminosas y diferentes oclusiones, que son las principales causas de fallos en este tipo de *tracking*. Se mostrará si la marca es detectada desde determinados puntos y, de serlo, cuál es la posición que estima que está la cámara.

Ángulo	¿Marca encontrada?
15º	Sí
30º	Sí
45º	Sí
60º	Sí
75º	Sí

Tabla 7.5: Resultados para el *tracking* con marcas en escenario virtual. Patrón 3 visto desde diferentes ángulos.

En la tabla 7.6 se detallan los resultados obtenidos en 5 momentos de la ejecución del módulo de posicionamiento con marcas en las que se podían visualizar alguna marca. En ella pueden observarse los siguientes datos: posición y orientación reales de la marca en el escenario en el momento de la visualización y posición y orientación estimados por el módulo de *tracking*. Aquellos casos en los que no se indica estimación de posicionamiento son en los que la marca no ha sido detectada.

	Pos. marca	Or. marca	Pos. estim.	Or. estim.
1	(6.0,9.0)	90º	(6.31,8.19)	219.21º
2	(4.0,4.0)	180º	-	-
3	(4.0,4.0)	180º	-	-
4	(4.0,9.0)	180º	(4.28,7.31)	9.68º
5	(4.0,9.0)	180º	-	-

Tabla 7.6: Resultados para el *tracking* con marcas en escenario real.

De los resultados de la tabla 7.6 podemos extraer varias conclusiones:

- Cuando la marca es detectada, la estimación de la posición y la orientación son muy exactas.
- La presencia de luz fluorescente dificulta mucho la detección de las marcas, como ocurre en los casos 2 y 3 (ver Anexo C).

- Cuando se está muy alejado de la marca la detección es mucho más complicada, como ocurre en el caso 5 (ver Anexo C).

En resumen, se tiene un sistema de *tracking* muy exacto, pero que requiere de apoyos para el posicionamiento cuando las marcas no pueden detectarse.

7.2. Pruebas de *tracking* estéreo

En esta sección se mostrarán los resultados obtenidos al poner en marcha el *tracking* estéreo. Con este *tracking* se busca obtener un cálculo correcto de la medida de desplazamiento y rotación de la cámara entre su posición inicial y su posición final.

Las pruebas se dividirán en dos partes, al igual que las pruebas del registro con marcas: pruebas en entorno virtual y pruebas en entorno real.

7.2.1. Pruebas en entorno virtual

Para realizar estas pruebas, se ha utilizado la escena realizada en *Blender* del escenario virtual utilizado en la prueba con marcas. Se han eliminado las marcas y se han escogido una serie de puntos fácilmente reconocibles en las imágenes renderizadas. En la figura 7.2 podemos observar las vistas de la cámara izquierda y derecha y los puntos que estudiaremos pintados en negro.

Figura 7.2: Vistas de las cámaras para las pruebas de *tracking* estéreo.

El objetivo de esta prueba ha sido el comprobar cómo se obtienen las coordenadas 3D de los puntos que hemos marcado utilizando la estimación de la rotación y traslación de la cámara a través de operaciones de triangulación. Puesto que conocemos con exactitud las coordenadas 3D de estos puntos gracias a *Blender*, se puede conocer la medida del error que comete el programa al mapearlos. También se realiza una medida del tiempo empleado en realizar los cálculos para el conjunto de puntos.

7.2.1.1. Cálculos intermedios

Antes de mostrar los resultados finales obtenidos por el programa, se mostrará los cálculos que han sido realizados para llegar a ellos. Tomaremos como ejemplo uno de los puntos que hemos seleccionado, y que denominaremos P.

Para este punto, las coordenadas UV[1] para la imagen izquierda y derecha son $p_L = (460, 347)$ y $p_R = (378, 347)$ respectivamente.

A partir de los parámetros intrínsecos obtenidos de la calibración, la matriz de cámara es igual para ambas imágenes:

$$M_l = M_r = \begin{pmatrix} s_x f_x & 0 & u_0 \\ 0 & s_y f_y & v_0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 624{,}99 & 0 & 499{,}85 \\ 0 & 624{,}99 & 299{,}72 \\ 0 & 0 & 1 \end{pmatrix} \tag{7.1}$$

Las matrices fundamental y esencial en este ejemplo tienen los siguientes valores:

$$F = \begin{pmatrix} 3{,}65020375147e-06 & -0{,}0002510465296 & -0{,}618786332147 \\ -0{,}0002068321356 & 0{,}00067043351317 & 79{,}0606291007 \\ 0{,}65557845480 & -79{,}325865272 & 0{,}999999999915 \end{pmatrix} \tag{7.2}$$

[1]Denominaremos coordenadas UV a aquellas que indican la posición 2D de un punto en una imagen utilizando como unidades los píxels de la imagen; se toma como el punto (0,0) la esquina superior izquierda de la imagen.

$$E = \begin{pmatrix} 1{,}42581869331 & -98{,}0621519368 & -432{,}622452709 \\ -80{,}791414817 & 261{,}880349917 & 49473{,}1364149 \\ 372{,}190223387 & -49530{,}9789123 & -106{,}78333838 \end{pmatrix} \tag{7.3}$$

Por último se muestran los valores estimados para la matriz de rotación y el vector de traslación entre las dos cámaras. Cuanto más exacta sea la estimación de estos elementos (calculados a partir de la matriz esencial), más exacto será el *tracking*.

$$R = \begin{pmatrix} 0{,}999867943983 & -0{,}0162471292565 & 0{,}000354097697698 \\ -0{,}0162476687671 & 0{,}999866752576 & -0{,}0015780851547 \\ 0{,}000328411161605 & 0{,}0015836300211 & 0{,}99999869213 \end{pmatrix} \tag{7.4}$$

$$T = \begin{pmatrix} 2{,}513173429878492 & 0{,}02196619055977345 & -0{,}004859477570195453 \end{pmatrix} \tag{7.5}$$

Realizando los cálculos pertinentes para la triangulación, obtenemos que el desplazamiento relativo del punto P con respecto de la cámara izquierda es $(0{,}602408, 0{,}71503, 9{,}44772)$.

7.2.1.2. Resultados finales

Para estas pruebas, en el escenario la cámara izquierda tiene la siguiente posición 3D:

$$Cam = (-5; 21{,}457; 2{,}439) \tag{7.6}$$

En la Tabla 7.7 podemos ver la estimación final de estos datos, correspondientes al punto 3. También podemos observar la estimación para el resto de puntos y el error cometido.

El tiempo empleado en calcular el conjunto de puntos, incluyendo la estimación de las matrices fundamental y esencial es de **0.631413 segundos**.

En general, el error cometido en la triangulación depende en gran medida del éxito a la hora de estimar la matriz fundamental. Los algoritmos empleados en esta estimación son capaces de aproximar los valores en función de los puntos extraidos de las dos imágenes propor-

	p_l	p_r	P_{real}	P_{est}	$Error$
Punto 1	(266,66)	(196,66)	(-12.95,13.53,23.92)	(-6.19,20.24,5.62)	20.632
Punto 2	(420,458)	(264,458)	(-6.18,23.91,12.05)	(-5.64,22.72,7.469	4.763
Punto 3	(460,347)	(378,347)	(-6.12,22.85,20.67)	(-5.53,22.04,10.79)	9.93
Punto 4	(624,379)	(504,379)	(-2.52,23.02,14.9)	(1.94,25.71,32.45)	18.306
Punto 5	(676,538)	(515,538)	(-2.36,25.02,11.79)	(-0.4,27.627,13.9)	4.343
Punto 6	(775,366)	(694,365)	(3.05,23.99,21.32)	(2.83,21.947,7.382)	14.088
Punto 7	(910,479)	(789,479)	(3.174,25.02,14.73)	(2.96,22.337,5.564)	9.553
Punto 8	(895,262)	(825,262)	(8.54,20.16,23.77)	(6.91,21.257,5.469)	18.4

Tabla 7.7: Resultados obtenidos en la triangulación 3D usando visión estereoscópica en escenario virtual

cionadas. Cuanto más exacta sea la estimación de la matriz fundamental, mejores resultados se obtendrán con el *tracking*

7.2.2. Pruebas en entorno real

En estas pruebas se comprobarán los datos de posicionamiento obtenidos con el módulo de *tracking* estéreo. Para ello se ha medido la estimación en 5 momentos de captura diferentes. Se ha intentado utilizar escenarios con movimiento variable para observar el comportamiento del módulo (movimiento de izquierda a derecha, movimiento de derecha a izquierda, movimiento hacia adelante...).

En la tabla 7.8 pueden observarse los resultados obtenidos. En ella podemos observar para cada prueba los siguientes datos:

- **Posición inicial de la cámara**: Posición en el entorno real donde se sitúa la cámara para la primera imagen del par estéreo.
- **Posición final estimada**: Posición en el entorno real donde el sistema estima que se encuentra la posición final.
- **Resultado**: Aquí se comentará si el resultado es correcto en relación al movimiento de

la cámara. Se indicará si el desplazamiento efectuado ha sido a izquierda o derecha, hacia adelante o hacia atrás.

Debido a que es muy dificil proporcionar con precisión una posición final en el entorno real, las pruebas irán encaminadas a la medición del seguimiento del movimiento de la cámara, es decir, si el desplazamiento efectuado por la misma es el que ha estimado el módulo de posicionamiento. De esta manera se asegura que el módulo de *tracking* es capaz de seguir una trayectoria de desplazamiento en el entorno.

	Pos. ini.	**Pos. estim.**	**Resultado**
1	(4.0,5.0)	(3.999999, 5.000005)	Correcto. Desplazamiento izquierda-adelante.
2	(6.0,4.0)	(5.999999,4.000007)	Correcto. Desplazamiento izquierda-adelante.
3	(8.0,8.0)	(7.99999,8.0)	Correcto. Desplazamiento izquierda.
4	(8.0,6.0)	(7.99999,6.0)	Incorrecto.
5	(9.0,7.0)	(6.9999, 9.0)	Incorrecto.

Tabla 7.8: Resultados para el *tracking* estéreo en escenario real.

Según se extrae de la tabla 7.8, los resultados son satisfactorios en cierta medida. El desplazamiento se mide correctamente para los tres primeros casos y se diferencia cuando hay un desplazamiento hacia adelante y cuando no. Sin embargo, los dos últimos casos hay error en la estimación. Estos casos coinciden con las imágenes de entrada que están más distorsionadas debido al movimiento de la cámara.

Hay que tener en cuenta que un sistema de *tracking* de este tipo es muy sensible a distorsiones, por tanto hay que procurar trabajar con cámaras de captura de imágenes fluída para evitar problemas.

7.3. Pruebas posicionamiento con descriptores visuales

En esta sección se presentan los resultados obtenidos al realizar diversas pruebas con el módulo de *tracking* con descriptores visuales implementado en Arges. Primero se mostrarán

un conjunto de pruebas realizadas en entorno virtual. Tras observar el comportamiento de este módulo en entorno virtual, se mostrarán pruebas en escenario real.

7.3.1. Pruebas en entorno virtual

Las pruebas se han realizado sobre la habitación virtual utilizada anteriormente. En ellas se mide el error cometido en la estimación de la posición y la orientación de la cámara al tener un cierto número de imágenes del entorno en la base de datos. Este error se mide como la distancia euclídea existente entre la posición real y la estimada; en el caso de la orientación se mide como la diferencia de ángulo entre la orientación real y la estimada.

Se muestran los resultados en 4, 16 y 64 posiciones diferentes de la habitación. Se ha procurado que la distribución de las tomas fotográficas sean lo más uniformes posible para asegurar una correcta visualización de todos los rincones del escenario.

Por cada posición se han tomado 8 fotografías con una separación de 45º en la orientación relativa entre ellas.

Además se muestra la latencia producida al realizar las búsquedas en el directorio de imágenes para comprobar en que medida aumenta el tiempo de búsqueda al aumentar el número de imágenes que comparar.

Para cada prueba se puede observar una tabla de datos y una gráfica asociada a dicha tabla. En la tabla se mostrarán la posición y orientación reales de los puntos donde se tomaron las fotografías ($X_{real}, Y_{real}, Rot_{real}$), la posición y orientación estimada ($X_{est}, Y_{est}, Rot_{est}$), el error producido en la estimación de ambos parámetros (E_{Pos}, E_{Rot}) y la latencia ($Lat.$). Las gráficas muestran los errores cometidos en las estimaciones. Tenemos por cada tabla una gráfica con una línea de color anaranjado, que indica el error en la estimación de la posición; la otra gráfica tiene una línea de color azul y muestra el error en la estimación de la orientación. Ambas gráficas tienen una línea amarilla horizontal que muestra la media del error cometido para el conjunto de imágenes en la estimación de la posición o la orientación, según la gráfica en la que se encuentre.

Estas pruebas han servido para estimar cuál es la distribución óptima de los pesos asignados a cada uno de los descriptores que intervienen en la comparación de imágenes. Por

esta razón, las primeras pruebas se realizan utilizando los descriptores individualmente para, en función de los primeros resultados, adecuar los pesos al emplear conjuntamente los tres descriptores.

7.3.1.1. Utilización exclusiva del descriptor *Color Layout*

- **Escena con 4 posiciones**

	X_{real}	Y_{real}	Rot_{real}	X_{est}	Y_{est}	Rot_{est}	E_{Pos}	E_{Rot}	$Lat.$
Imagen 1	-5.97	0.04	276.14	-4.5	2	270	2.45	6.14	0.8607
Imagen 2	-9.24	-0.67	260.14	-4.5	2	270	5.44	9.86	0.7176
Imagen 3	-11.2	-4.05	12.65	10.5	16	225	29.54	147.65	0.7171
Imagen 4	-4.1	8.57	82.92	-4.5	2	135	6.58	52.08	0.6496
Imagen 5	-6.11	1.04	313.88	-4.5	2	315	1.87	1.12	0.6852
Imagen 6	1.02	9.45	44.68	10.5	16	90	11.52	45.32	0.712
Imagen 7	-7.9	12.01	151.77	10.5	16	315	18.82	163.23	0.7549
Imagen 8	16.65	10.84	172.37	10.5	16	225	8.02	52.63	0.66
Imagen 9	3.72	7.44	218.65	10.5	2	45	8.69	173.65	0.6791
Imagen 10	-8.99	14.25	285.66	-4.5	16	270	4.81	15.66	0.669
Imagen 11	-4.57	15.76	9.63	10.5	16	90	15.07	80.37	0.6998
Imagen 12	-7.34	4.96	268.92	-4.5	16	225	11.39	43.62	0.6842
Imagen 13	7.47	9.7	108.09	10.5	2	45	8.27	63.09	0.6752
Imagen 14	17.31	15.28	76.27	-4.5	16	0	21.82	76.27	0.6912
Imagen 15	-4.43	14.18	50.59	-4.5	16	45	1.82	5.59	0.685

Tabla 7.9: Resultados con 4 posiciones empleando únicamente *Color Layout*.

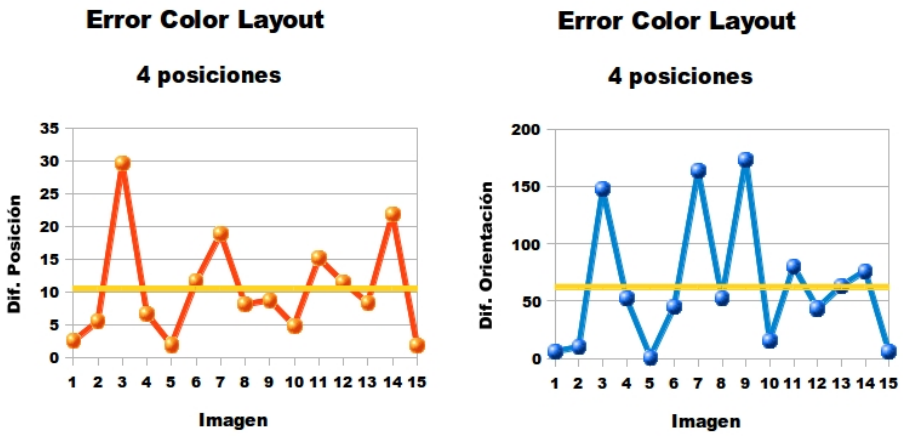


Figura 7.3: Error en posición y orientación con 4 posiciones empleando únicamente *Color Layout*

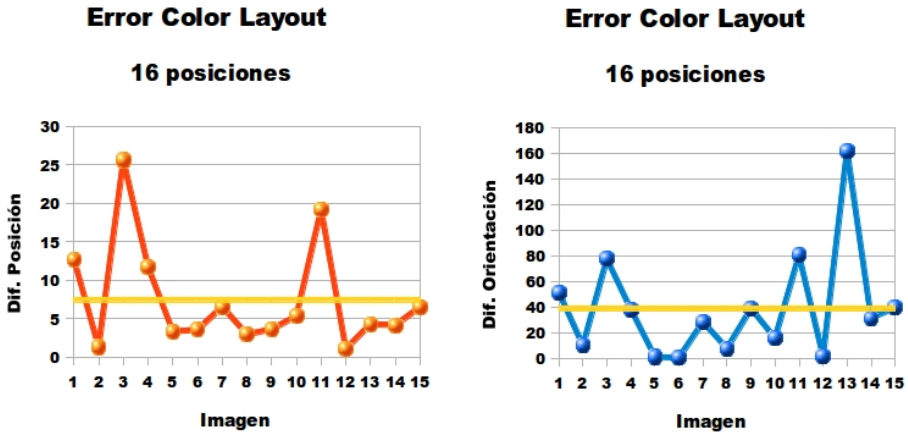


Figura 7.4: Error en posición y orientación con 16 posiciones empleando únicamente *Color Layout*

- **Escena con 16 posiciones**

	X_{real}	Y_{real}	Rot_{real}	X_{est}	Y_{est}	Rot_{est}	E_{Pos}	E_{Rot}	$Lat.$
Imagen 1	-5.97	0.04	276.14	-8.25	12.5	225	12.66	51.14	0.7987
Imagen 2	-9.24	-0.67	260.14	-8.25	-1.5	270	1.29	9.86	0.6834
Imagen 3	-11.2	-4.05	12.65	14.25	-1.5	90	25.57	77.35	0.6956
Imagen 4	-4.1	8.57	82.92	-8.25	19.5	45	11.69	37.92	0.6579
Imagen 5	-6.11	1.04	313.88	-8.25	-1.5	315	3.32	1.12	0.695
Imagen 6	1.02	9.45	44.68	-0.75	12.5	45	3.52	0.32	0.6938
Imagen 7	-7.9	12.01	151.77	-8.25	5.5	180	6.52	28.23	0.6594
Imagen 8	16.65	10.84	172.37	14.25	12.5	180	2.92	7.63	0.6748
Imagen 9	3.72	7.44	218.65	6.75	5.5	0	3.59	38.65	0.6908
Imagen 10	-8.99	14.25	285.66	-8.25	19.5	270	5.3	15.66	0.6814
Imagen 11	-4.57	15.76	9.63	14.25	19.5	90	19.18	80.37	0.6914
Imagen 12	-7.34	4.96	268.92	-8.25	5.5	270	1.05	1.38	0.6841
Imagen 13	7.47	9.7	108.09	6.75	5.5	270	4.26	161.91	0.6863
Imagen 14	17.31	15.28	76.27	14.25	12.5	45	4.13	31.27	0.6824
Imagen 15	-4.43	14.18	50.59	-0.75	19.5	90	6.46	39.41	0.6671

Tabla 7.10: Resultados con 16 posiciones empleando únicamente *Color Layout*.

- **Escena con 64 posiciones**

	X_{real}	Y_{real}	Rot_{real}	X_{est}	Y_{est}	Rot_{est}	E_{Pos}	E_{Rot}	$Lat.$
Imagen 1	-5.97	0.04	276.14	-6.375	10.75	225	10.71	51.14	2.0722
Imagen 2	-9.24	-0.67	260.14	16.125	3.75	0	25.74	80.14	0.7292
Imagen 3	-11.2	-4.05	12.65	16.125	7.25	180	29.56	167.35	0.7359
Imagen 4	-4.1	8.57	82.92	8.625	12.5	270	13.31	172.92	0.6809
Imagen 5	-6.11	1.04	313.88	-6.375	0.25	315	0.83	1.12	0.7175
Imagen 6	1.02	9.45	44.68	1.125	9	45	0.46	0.32	0.7204
Imagen 7	-7.9	12.01	151.77	-6.375	10.75	135	1.97	16.77	0.6814
Imagen 8	16.65	10.84	172.37	8.625	16	225	9.54	52.63	0.7773
Imagen 9	3.72	7.44	218.65	8.625	7.25	45	4.9	173.65	0.7132
Imagen 10	-8.99	14.25	285.66	-6.375	7.25	315	7.47	29.34	0.722
Imagen 11	-4.57	15.76	9.63	-6.375	17.75	0	2.68	9.63	0.7197
Imagen 12	-7.34	4.96	268.92	-10.125	3.75	270	3.03	1.38	0.7138
Imagen 13	7.47	9.7	108.09	8.625	3.75	45	6.06	63.09	0.7315
Imagen 14	17.31	15.28	76.27	-2.625	-3.25	0	27.21	76.27	0.7138
Imagen 15	-4.43	14.18	50.59	-6.375	14.25	45	1.94	5.59	0.7131

Tabla 7.11: Resultados con 64 posiciones empleando únicamente *Color Layout*.

Analizando los resultados de estas tres pruebas para el descriptor *ColorLayout*, se puede observar como el mejor comportamiento se obtiene con 16 posiciones. Esto demuestra que no es un descriptor fiable por sí solo, a pesar de que con 16 posiciones ha tenido una tasa de acierto relativamente buena.

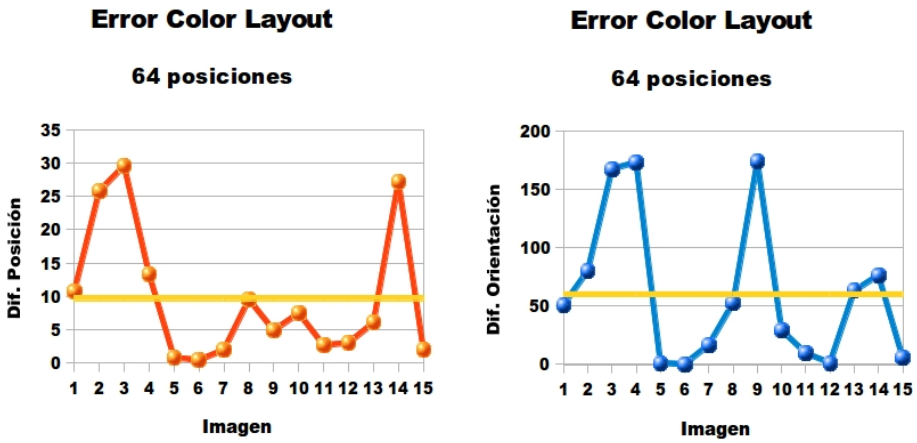


Figura 7.5: Error en posición y orientación con 64 posiciones empleando únicamente *Color Layout*

7.3.1.2. Utilización exclusiva del descriptor *Edge Histogram*

- **Escena con 4 posiciones**

	X_{real}	Y_{real}	Rot_{real}	X_{est}	Y_{est}	Rot_{est}	E_{Pos}	E_{Rot}	$Lat.$
Imagen 1	-5.97	0.04	276.14	-4.5	2	270	2.45	6.14	0.7772
Imagen 2	-9.24	-0.67	260.14	-4.5	16	225	17.33	35.14	0.6829
Imagen 3	-11.2	-4.05	12.65	10.5	2	45	22.52	32.35	0.6942
Imagen 4	-4.1	8.57	82.92	10.5	16	270	16.38	172.92	0.65
Imagen 5	-6.11	1.04	313.88	-4.5	2	315	1.87	178.88	0.6732
Imagen 6	1.02	9.45	44.68	-4.5	16	270	8.56	134.68	0.6901
Imagen 7	-7.9	12.01	151.77	10.5	2	225	20.94	73.23	0.6497
Imagen 8	16.65	10.84	172.37	-4.5	16	270	21.77	97.63	0.7227
Imagen 9	3.72	7.44	218.65	10.5	2	135	8.69	83.65	0.6828
Imagen 10	-8.99	14.25	285.66	-4.5	2	0	13.04	74.34	0.6833
Imagen 11	-4.57	15.76	9.63	-4.5	16	0	0.25	9.63	0.683
Imagen 12	-7.34	4.96	268.92	-4.5	16	225	11.39	43.62	0.681

	X_{real}	Y_{real}	Rot_{real}	X_{est}	Y_{est}	Rot_{est}	E_{Pos}	E_{Rot}	$Lat.$
Imagen 13	7.47	9.7	108.09	10.5	2	90	8.27	18.09	0.7098
Imagen 14	17.31	15.28	76.27	10.5	16	90	6.84	13.73	0.6801
Imagen 15	-4.43	14.18	50.59	-4.5	16	45	1.82	5.59	0.6718

Tabla 7.12: Resultados con 4 posiciones empleando únicamente *Edge Histogram*.

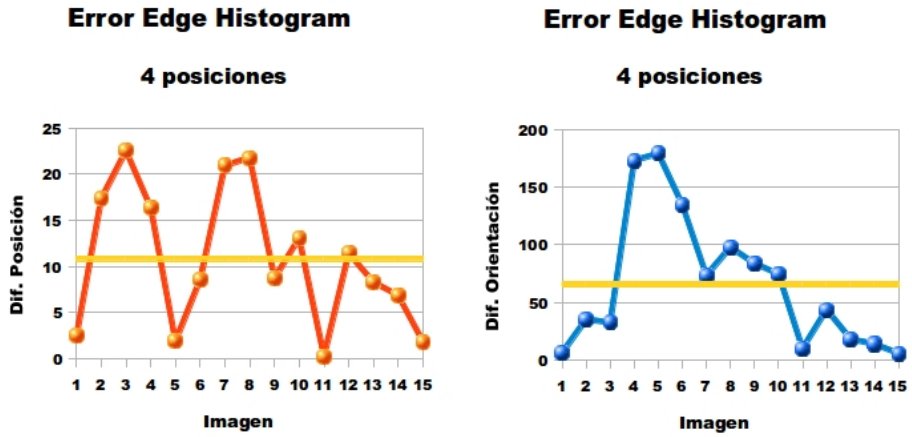


Figura 7.6: Error en posición y orientación con 4 posiciones empleando únicamente *Edge Histogram*

- **Escena con 16 posiciones**

	X_{real}	Y_{real}	Rot_{real}	X_{est}	Y_{est}	Rot_{est}	E_{Pos}	E_{Rot}	$Lat.$
Imagen 1	-5.97	0.04	276.14	-8.25	-1.5	270	2.75	6.14	0.7788
Imagen 2	-9.24	-0.67	260.14	14.25	5.5	0	24.28	80.14	0.6756
Imagen 3	-11.2	-4.05	12.65	6.75	5.5	180	20.33	167.35	0.6857
Imagen 4	-4.1	8.57	82.92	-0.75	5.5	90	4.54	7.08	0.6415
Imagen 5	-6.11	1.04	313.88	-8.25	-1.5	315	3.32	1.12	0.6834

	X_{real}	Y_{real}	Rot_{real}	X_{est}	Y_{est}	Rot_{est}	E_{Pos}	E_{Rot}	$Lat.$
Imagen 6	1.02	9.45	44.68	-0.75	5.5	45	4.32	0.32	0.6891
.Imagen 7	-7.9	12.01	151.77	6.75	5.5	225	16.03	73.23	0.6787
Imagen 8	16.65	10.84	172.37	-0.75	12.5	270	17.47	97.63	0.6691
Imagen 9	3.72	7.44	218.65	-0.75	12.5	135	6.75	83.65	0.6882
Imagen 10	-8.99	14.25	285.66	-8.25	19.5	270	5.3	15.66	0.6793
Imagen 11	-4.57	15.76	9.63	6.75	12.5	315	11.78	125.37	0.6859
Imagen 12	-7.34	4.96	268.92	-8.25	5.5	270	1.05	1.38	0.6768
Imagen 13	7.47	9.7	108.09	6.75	5.5	90	4.26	18.09	0.6975
Imagen 14	17.31	15.28	76.27	14.25	19.5	90	5.21	13.73	0.6818
Imagen 15	-4.43	14.18	50.59	-8.25	19.5	45	6.54	5.59	0.6897

Tabla 7.13: Resultados con 16 posiciones empleando únicamente *Edge Histogram*.

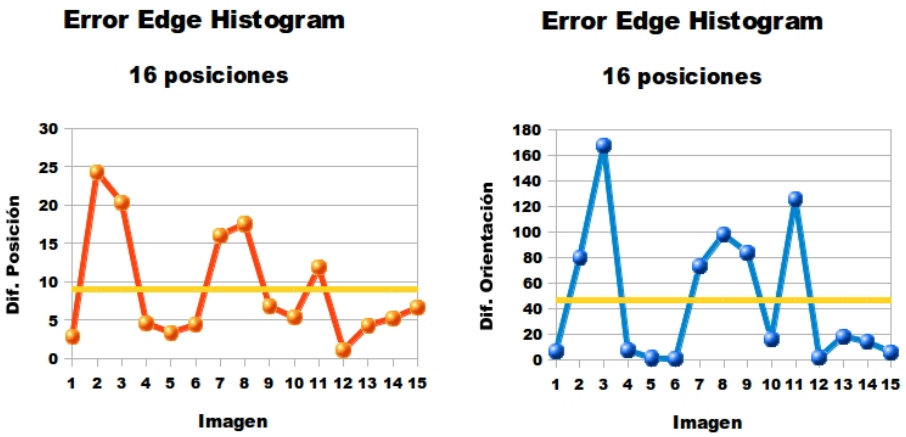


Figura 7.7: Error en posición y orientación con 16 posiciones empleando únicamente *Edge Histogram*

- **Escena con 64 posiciones**

	X_{real}	Y_{real}	Rot_{real}	X_{est}	Y_{est}	Rot_{est}	E_{Pos}	E_{Rot}	$Lat.$
Imagen 1	-5.97	0.04	276.14	-2.625	0.25	270	3.35	6.14	2.1597
Imagen 2	-9.24	-0.67	260.14	-6.375	-3.25	270	3.85	9.86	0.713
Imagen 3	-11.2	-4.05	12.65	4.875	-3.25	45	16.09	32.35	0.7351
Imagen 4	-4.1	8.57	82.92	8.625	16	270	14.73	172.92	0.6904
Imagen 5	-6.11	1.04	313.88	-6.375	0.25	315	0.83	1.12	0.724
Imagen 6	1.02	9.45	44.68	1.125	7.25	45	2.2	0.32	0.7242
Imagen 7	-7.9	12.01	151.77	1.125	9	135	9.51	16.77	0.6953
Imagen 8	16.65	10.84	172.37	16.125	12.5	180	1.74	7.63	0.7211
Imagen 9	3.72	7.44	218.65	12.375	9	270	8.79	51.35	0.7183
Imagen 10	-8.99	14.25	285.66	-10.125	17.75	270	3.67	15.66	0.7183
Imagen 11	-4.57	15.76	9.63	-6.375	14.25	0	2.35	9.63	0.7257
Imagen 12	-7.34	4.96	268.92	-10.125	3.75	270	3.03	1.38	0.7202
Imagen 13	7.47	9.7	108.09	8.625	7.25	135	2.7	26.91	0.731
Imagen 14	17.31	15.28	76.27	16.125	19.5	90	4.38	13.73	0.7296
Imagen 15	-4.43	14.18	50.59	-6.375	14.25	45	1.94	5.59	0.7141

Tabla 7.14: Resultados con 64 posiciones empleando únicamente *Edge Histogram*.

Las pruebas con el descriptor *Edge Histogram* demuestran que puede ser utilizado independientemente, puesto que con un mayor número de imágenes da mejores resultados. Para los tres casos este descriptor proporciona mejores resultados que el *Color Layout*.

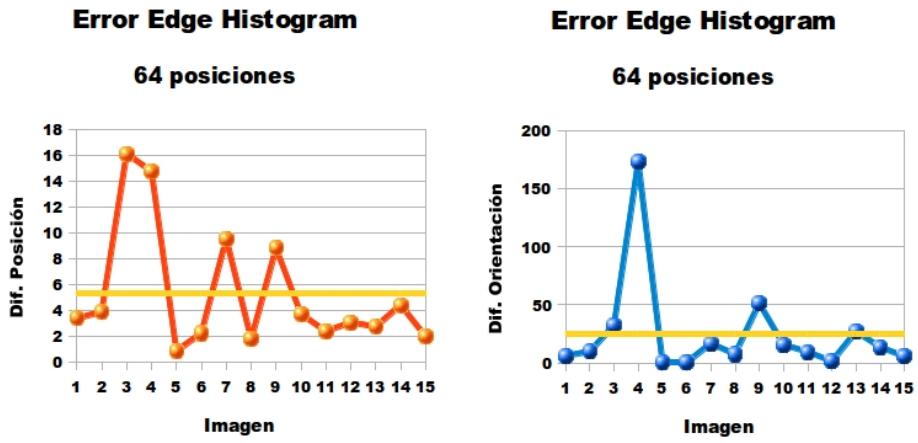


Figura 7.8: Error en posición y orientación con 64 posiciones empleando únicamente *Edge Histogram*

7.3.1.3. Utilización exclusiva del descriptor *Scalable Color*

- **Escena con 4 posiciones**

	X_{real}	Y_{real}	Rot_{real}	X_{est}	Y_{est}	Rot_{est}	E_{Pos}	E_{Rot}	$Lat.$
Imagen 1	-5.97	0.04	276.14	-4.5	16	225	16.02	51.14	0.7264
Imagen 2	-9.24	-0.67	260.14	-4.5	16	225	17.33	35.14	0.6999
Imagen 3	-11.2	-4.05	12.65	10.5	16	90	29.54	77.35	0.6897
Imagen 4	-4.1	8.57	82.92	10.5	16	315	16.38	127.92	0.6958
Imagen 5	-6.11	1.04	313.88	-4.5	2	315	1.87	1.12	0.7117
Imagen 6	1.02	9.45	44.68	-4.5	2	0	9.27	44.68	0.6832
Imagen 7	-7.9	12.01	151.77	10.5	16	315	18.82	163.23	0.6537
Imagen 8	16.65	10.84	172.37	10.5	16	180	8.02	7.63	0.6749
Imagen 9	3.72	7.44	218.65	10.5	2	315	8.69	96.35	0.6971
Imagen 10	-8.99	14.25	285.66	-4.5	16	270	4.81	15.66	0.7109
Imagen 11	-4.57	15.76	9.63	10.5	16	45	15.07	35.37	0.685
Imagen 12	-7.34	4.96	268.92	-4.5	16	225	11.39	43.65	0.6776
Imagen 13	7.47	9.7	108.09	10.5	2	45	8.27	63.09	0.6758
Imagen 14	17.31	15.28	76.27	-4.5	16	315	21.82	121.67	0.6811

	X_{real}	Y_{real}	Rot_{real}	X_{est}	Y_{est}	Rot_{est}	E_{Pos}	E_{Rot}	$Lat.$
Imagen 15	-4.43	14.18	50.59	-4.5	16	90	1.82	39.41	0.6582

Tabla 7.15: Resultados con 4 posiciones empleando únicamente *Scalable Color*.

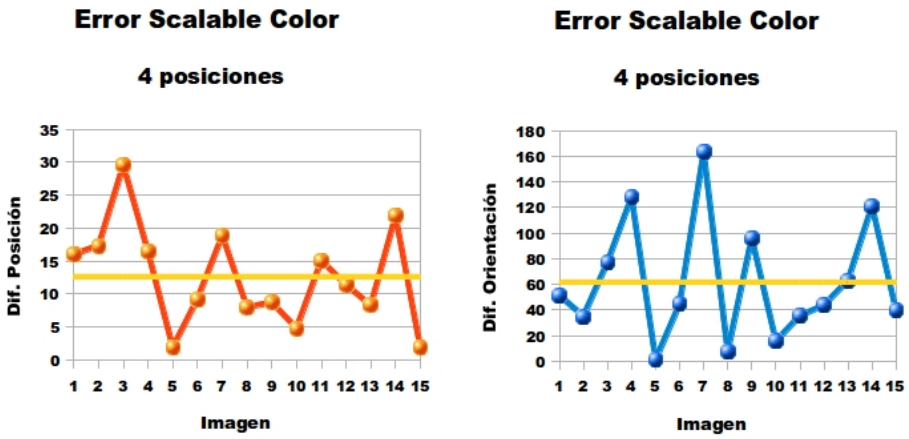


Figura 7.9: Error en posición y orientación con 4 posiciones empleando únicamente *Scalable Color*

- **Escena con 16 posiciones**

	X_{real}	Y_{real}	Rot_{real}	X_{est}	Y_{est}	Rot_{est}	E_{Pos}	E_{Rot}	$Lat.$
Imagen 1	-5.97	0.04	276.14	-8.25	5.5	270	5.91	6.14	0.7722
Imagen 2	-9.24	-0.67	260.14	-8.25	12.5	225	13.2	35.14	0.6722
Imagen 3	-11.2	-4.05	12.65	6.75	12.5	45	24.41	32.35	0.6879
Imagen 4	-4.1	8.57	82.92	14.25	19.5	225	21.35	142.08	0.6524
Imagen 5	-6.11	1.04	313.88	-8.25	-1.5	315	3.32	1.12	0.6869
Imagen 6	1.02	9.45	44.68	-0.75	12.5	45	3.52	0.32	0.6976
Imagen 7	-7.9	12.01	151.77	14.25	19.5	225	23.38	73.23	0.6647

	X_{real}	Y_{real}	Rot_{real}	X_{est}	Y_{est}	Rot_{est}	E_{Pos}	E_{Rot}	$Lat.$
Imagen 8	16.65	10.84	172.37	14.25	12.5	180	2.91	7.63	0.6808
Imagen 9	3.72	7.44	218.65	6.75	12.5	225	5.89	6.35	0.6892
Imagen 10	-8.99	14.25	285.66	-8.25	19.5	270	5.3	15.66	0.6841
Imagen 11	-4.57	15.76	9.63	-0.75	12.5	45	5.02	35.37	0.6882
Imagen 12	-7.34	4.96	268.92	-8.25	5.5	270	1.05	1.38	0.6914
Imagen 13	7.47	9.7	108.09	6.75	5.5	225	4.26	116.91	0.6909
Imagen 14	17.31	15.28	76.27	14.25	19.5	90	5.21	13.73	0.6867
Imagen 15	-4.43	14.18	50.59	-8.25	12.5	45	4.17	5.59	0.6778

Tabla 7.16: Resultados con 16 posiciones empleando únicamente *Scalable Color*.

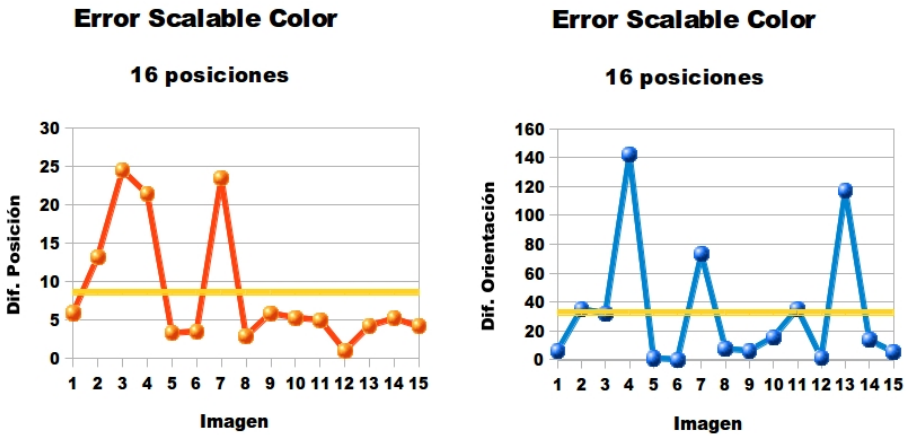


Figura 7.10: Error en posición y orientación con 16 posiciones empleando únicamente *Scalable Color*

- **Escena con 64 posiciones**

	X_{real}	Y_{real}	Rot_{real}	X_{est}	Y_{est}	Rot_{est}	E_{Pos}	E_{Rot}	$Lat.$
Imagen 1	-5.97	0.04	276.14	-6.375	10.75	270	10.71	6.14	2.2124
Imagen 2	-9.24	-0.67	260.14	-10.125	7.25	270	7.96	9.86	0.7036
Imagen 3	-11.2	-4.05	12.65	-10.125	3.75	0	7.87	12.65	0.7196
Imagen 4	-4.1	8.57	82.92	-6.375	10.75	135	3.15	52.08	0.6772
Imagen 5	-6.11	1.04	313.88	-6.375	-3.25	315	4.29	1.12	0.7179
Imagen 6	1.02	9.45	44.68	1.125	9	45	0.46	0.32	0.7162
Imagen 7	-7.9	12.01	151.77	-10.125	14.25	180	3.15	28.23	0.6853
Imagen 8	16.65	10.84	172.37	1.125	12.5	180	15.61	7.63	0.7073
Imagen 9	3.72	7.44	218.65	4.875	0.25	315	7.28	96.35	0.7148
Imagen 10	-8.99	14.25	285.66	-6.375	17.75	270	4.36	15.66	0.7148
Imagen 11	-4.57	15.76	9.63	-6.375	14.25	0	2.35	9.63	0.7199
Imagen 12	-7.34	4.96	268.92	-10.125	3.75	270	3.03	1.38	0.7186
Imagen 13	7.47	9.7	108.09	8.625	3.75	315	6.06	153.09	0.721
Imagen 14	17.31	15.28	76.27	16.125	16	90	1.38	13.73	0.7134
Imagen 15	-4.43	14.18	50.59	-6.375	14.25	45	1.94	5.59	0.7082

Tabla 7.17: Resultados con 64 posiciones empleando únicamente *Scalable Color*.

Al igual que el *Edge Histogram*, el comportamiento del descriptor *Scalable Color* mejora al aumentar el número de percepciones. Los resultados obtenidos con este descriptor son algo mejores que con el anterior descriptor.

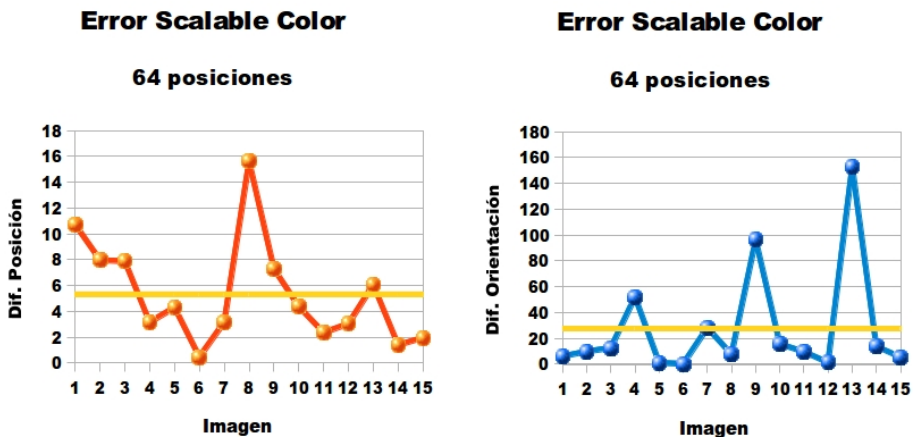


Figura 7.11: Error en posición y orientación con 64 posiciones empleando únicamente *Scalable Color*

7.3.1.4. Primeras conclusiones

A la vista de los resultados obtenidos al separar la utilización de los tres descriptores que intervendrán en la estimación de la posición, podemos extraer una serie de conclusiones:

- A medida que aumentamos el número de muestras, la probabilidad de acierto en la estimación de la posición es mayor. La calidad de la estimación dependerá de la posición en la que fueron tomadas las muestras de imágenes; por ello, es imprescindible tomar las muestras con medidas equitativas entre ellas, de tal forma que minimicemos la distancia entre una muestra y otra.

- El descriptor con mayor índice de exactitud en la estimación de la posición es el *Scalable Color*, seguido del *Edge Histogram* y del *Color Layout*.

- El descriptor con mayor índice de exactitud en la estimación de la orientación es el *Scalable Color*, seguido del *Edge Histogram* y del *Color Layout*.

- La latencia del programa a la hora de mostrar resultados es muy baja, obteniendo los datos al realizar peticiones de imágenes consecutivas en menos de un segundo para cada imagen aunque tengamos una base de datos de más de 500 muestras. La latencia solo se ve afectada en la primera búsqueda, debido a que se emplea tiempo en la creación del árbol de búsqueda.

A continuación, se muestran los resultados obtenidos sobre los mismos datos de imágenes pero combinando los tres descriptores. Se asignará un peso a cada descriptor en función de lo mencionado anteriormente. El *Scalable Color* y el *Color Layout* recibirán más peso que el *Edge Histogram* al proporcionar información más fiable y precisa.

7.3.1.5. Utilización de los tres descriptores

A la vista de la información extraida anteriormente acerca del comportamiento de los descriptores por separado, se ejecutarán tres pruebas más para comprobar el funcionamiento con tres distribuciones de pesos. El hecho de que el descriptor *Edge Histogram* reciba un peso más pequeño en estas pruebas es debido a la implementación en la biblioteca MPEG-7FexLib, que otorga un valor de distancia mayor que los otros dos descriptores. En estas pruebas se verá como otorgando un peso muy alto en este descriptor, los resultados empeoran.

- **Prueba 1**

  La distribución de los pesos en esta prueba ha sido:

  - *Color Layout*: 0.4
  - *Edge Histogram*: 0.1
  - *Scalable Color*: 0.5

	X_{real}	Y_{real}	Rot_{real}	X_{est}	Y_{est}	Rot_{est}	E_{Pos}	E_{Rot}	$Lat.$
Imagen 1	-5.97	0.04	276.14	-6.375	0.25	270	0.45	6.14	2.0261
Imagen 2	-9.24	-0.67	260.14	-6.375	-3.25	270	3.85	9.86	0.7179
Imagen 3	-11.2	-4.05	12.65	-6.375	-3.25	45	4.89	32.35	0.7413
Imagen 4	-4.1	8.57	82.92	8.625	16	270	14.73	172.92	0.6854
Imagen 5	-6.11	1.04	313.88	-6.375	0.25	315	0.83	1.12	0.7306
Imagen 6	1.02	9.45	44.68	1.125	9	45	0.46	0.32	0.7272
Imagen 7	-7.9	12.01	151.77	12.375	16	315	20.66	163.23	0.6907
Imagen 8	16.65	10.84	172.37	8.625	16	225	9.54	52.63	0.701
Imagen 9	3.72	7.44	218.65	8.625	12.5	180	7.04	38.65	0.7255

	X_{real}	Y_{real}	Rot_{real}	X_{est}	Y_{est}	Rot_{est}	E_{Pos}	E_{Rot}	$Lat.$
Imagen 10	-8.99	14.25	285.66	-10.125	17.75	270	3.67	15.66	0.7186
Imagen 11	-4.57	15.76	9.63	-6.375	14.25	0	2.35	9.63	0.726
Imagen 12	-7.34	4.96	268.92	-10.125	3.75	270	3.03	1.38	0.7254
Imagen 13	7.47	9.7	108.09	8.625	9	90	1.35	18.09	0.7244
Imagen 14	17.31	15.28	76.27	16.125	19.5	90	4.38	13.73	0.7476
Imagen 15	-4.43	14.18	50.59	-6.375	14.25	45	1.94	5.59	0.7265

Tabla 7.18: Resultados con 64 posiciones; *Color Layout*(0.4), *Edge Histogram*(0.1), *Scalable Color*(0.5)

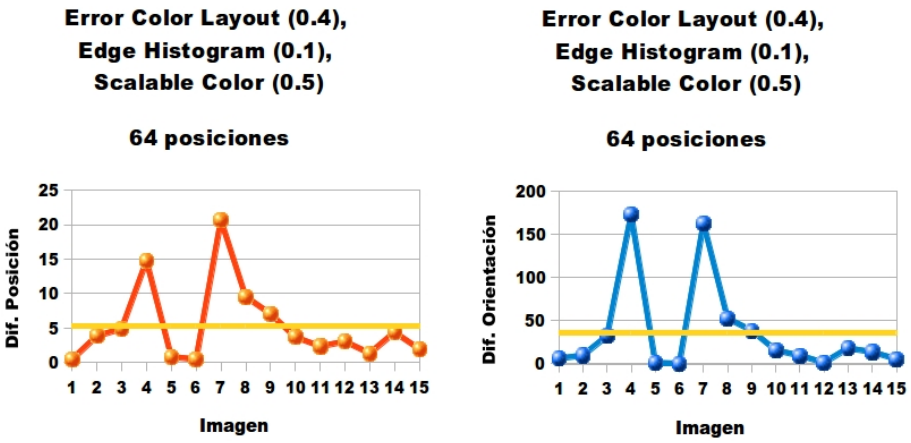


Figura 7.12: Error en posición y orientación con 64 posiciones; *Color Layout*(0.4), *Edge Histogram*(0.1), *Scalable Color*(0.5)

Viendo estos resultados se observa como al juntar los tres descriptores, la estimación mejora con respecto a cada uno de los descriptores individualmente. Las siguientes pruebas realizarán alguna variación sobre los pesos de los descriptores para ver en qué medida mejora la estimación.

- **Prueba 2**

  La distribución de los pesos en esta prueba ha sido:

  - *Color Layout*: 0.3
  - *Edge Histogram*: 0.2
  - *Scalable Color*: 0.5

	X_{real}	Y_{real}	Rot_{real}	X_{est}	Y_{est}	Rot_{est}	E_{Pos}	E_{Rot}	$Lat.$
Imagen 1	-5.97	0.04	276.14	-6.375	0.25	270	0.45	6.14	2.0153
Imagen 2	-9.24	-0.67	260.14	-6.375	-3.25	270	3.85	9.86	0.7159
Imagen 3	-11.2	-4.05	12.65	-6.375	-3.25	45	4.89	32.35	0.7364
Imagen 4	-4.1	8.57	82.92	8.625	16	270	14.73	172.92	0.7061
Imagen 5	-6.11	1.04	313.88	-6.375	0.25	315	0.83	1.12	0.7359
Imagen 6	1.02	9.45	44.68	1.125	9	45	0.46	0.32	0.7104
Imagen 7	-7.9	12.01	151.77	12.375	16	315	20.66	163.23	0.7155
Imagen 8	16.65	10.84	172.37	16.125	12.5	180	1.74	7.63	0.7357
Imagen 9	3.72	7.44	218.65	12.375	7.25	180	8.65	38.65	0.7412
Imagen 10	-8.99	14.25	285.66	-10.125	17.75	270	3.67	15.66	0.7483
Imagen 11	-4.57	15.76	9.63	-6.375	14.25	0	2.35	9.63	0.739
Imagen 12	-7.34	4.96	268.92	-10.125	3.75	270	3.03	1.38	0.7444
Imagen 13	7.47	9.7	108.09	8.625	9	90	1.35	18.09	0.734
Imagen 14	17.31	15.28	76.27	16.125	19.5	90	4.38	13.73	0.7239
Imagen 15	-4.43	14.18	50.59	-6.375	14.25	45	1.94	5.59	0.7252

Tabla 7.19: Resultados con 64 posiciones; *Color Layout*(0.3), *Edge Histogram*(0.2), *Scalable Color*(0.5)

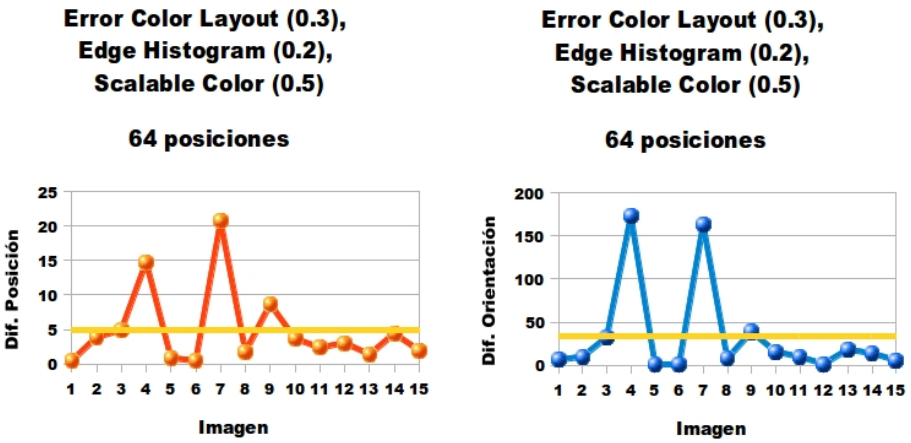


Figura 7.13: Error en posición y orientación con 64 posiciones; *Color Layout*(0.3), *Edge Histogram*(0.2), *Scalable Color*(0.5)

Según los resultados, quitándole peso al *Color Layout* y añadiéndoselo al *Edge Histogram* la estimación mejora sensiblemente. Realizando otra serie de pruebas se ha comprobado como un peso mayor para el *Edge Histogram* empeora los resultados.

En la última prueba, se ha asignado un mayor peso al descriptor *Scalable Color* para comprobar si los resultados pueden ser aun más satisfactorios.

- **Prueba 3**

  La distribución de los pesos en esta prueba ha sido:

  - *Color Layout*: 0.3
  - *Edge Histogram*: 0.1
  - *Scalable Color*: 0.6

	X_{real}	Y_{real}	Rot_{real}	X_{est}	Y_{est}	Rot_{est}	E_{Pos}	E_{Rot}	$Lat.$
Imagen 1	-5.97	0.04	276.14	-6.375	0.25	270	0.45	6.14	1.9743
Imagen 2	-9.24	-0.67	260.14	-6.375	-3.25	270	3.85	9.86	0.7132
Imagen 3	-11.2	-4.05	12.65	-6.375	-3.25	45	4.89	32.35	0.7353
Imagen 4	-4.1	8.57	82.92	8.625	16	270	14.73	172.92	0.6809

	X_{real}	Y_{real}	Rot_{real}	X_{est}	Y_{est}	Rot_{est}	E_{Pos}	E_{Rot}	$Lat.$
Imagen 5	-6.11	1.04	313.88	-6.375	0.25	315	0.83	1.12	0.7136
Imagen 6	1.02	9.45	44.68	1.125	9	45	0.46	0.32	0.7233
Imagen 7	-7.9	12.01	151.77	12.375	16	315	20.66	163.23	0.689
Imagen 8	16.65	10.84	172.37	16.125	12.5	180	1.74	7.63	0.7092
Imagen 9	3.72	7.44	218.65	4.875	0.25	315	7.28	96.35	0.7263
Imagen 10	-8.99	14.25	285.66	-10.125	17.75	270	3.67	15.66	0.7542
Imagen 11	-4.57	15.76	9.63	-6.375	14.25	0	2.35	9.63	0.7226
Imagen 12	-7.34	4.96	268.92	-10.125	3.75	270	3.03	1.38	0.7243
Imagen 13	7.47	9.7	108.09	8.625	9	90	1.35	18.09	0.717
Imagen 14	17.31	15.28	76.27	16.125	19.5	90	4.38	13.73	0.7604
Imagen 15	-4.43	14.18	50.59	-6.375	14.25	45	1.94	5.59	0.7396

Tabla 7.20: Resultados con 64 posiciones; *Color Layout*(0.3), *Edge Histogram*(0.1), *Scalable Color*(0.6)

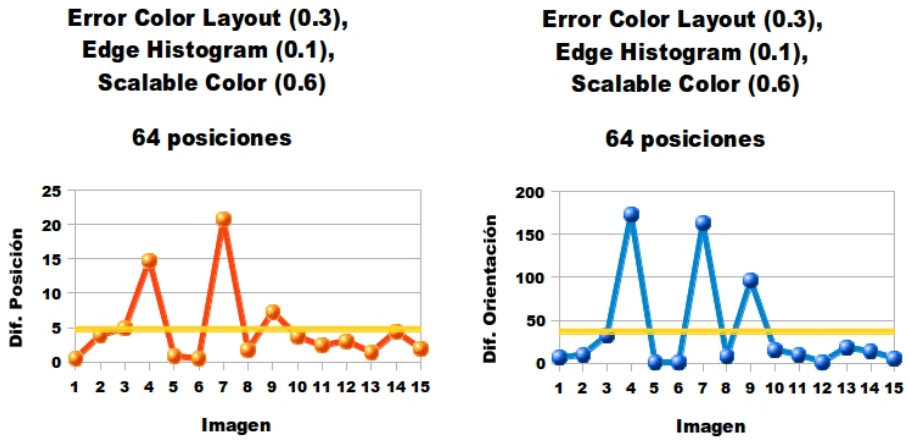


Figura 7.14: Error en posición y orientación con 64 posiciones; *Color Layout*(0.3), *Edge Histogram*(0.1), *Scalable Color*(0.6)

Se puede ver que añadiéndole más peso al *Scalable Color* los resultados no solo no mejoran, sino que son algo peores que en la prueba anterior. En la sección siguiente se puede ver un resumen de los datos que aportan estas pruebas.

7.3.1.6. Resultados finales y conclusiones

Tras las pruebas realizadas con diferentes pesos en cada uno de los descriptores, podemos establecer que la distribución más fiable y precisa en cuanto a la obtención de posición y orientación de las imágenes es la correspondiente a la prueba 2, es decir: *Color Layout* , 0.3; *Edge Histogram*, 0.2; *Scalable Color*, 0.5.

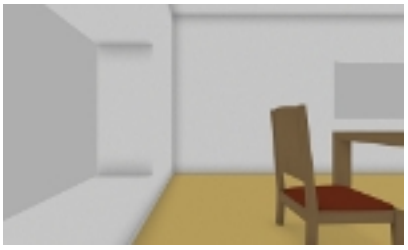
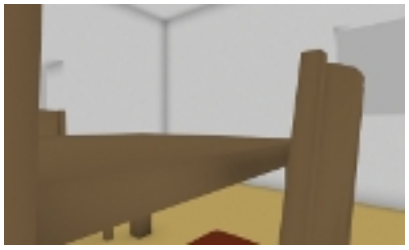
Como puede observarse en las pruebas finales, existen casos en los que a pesar de tener muestras muy parecidas a la imagen real, como es el caso de la Imagen 7 (ver Tabla 7.20 y Tabla 7.21), puede ser que la estimación de la posición y la orientación no sea la correcta. Esto resultará muy común en espacios con cierta homogeneidad en la distribución de los elementos, así como en espacios en los que exista mucha homogeneidad en cuanto a la textura; por ejemplo, espacios con paredes lisas de un solo color y sin elementos decorativos.

A continuación se muestran los resultados visuales obtenidos en la prueba 2 sobre las imágenes de prueba para poder observar la similitud estimada. Recuérdese que en la prueba 2 se empleó una base de datos con 512 muestras del escenario y una distribución de pesos de 0.3 para el descriptor *Color Layout*, 0.2 para el *Edge Histogram* y 0.5 para el *Scalable Color*.

	Imagen real	**Imagen estimada**
Imagen 1.jpg		
Continúa en la siguiente página.		

	Imagen real	Imagen estimada
Imagen 2.jpg		
Imagen 3.jpg		
Imagen 4.jpg		
Imagen 5.jpg		
Imagen 6.jpg		

Continúa en la siguiente página.

	Imagen real	Imagen estimada
Imagen 7.jpg		
Imagen 8.jpg		
Imagen 9.jpg		
Imagen 10.jpg		
Imagen 11.jpg		

Continúa en la siguiente página.

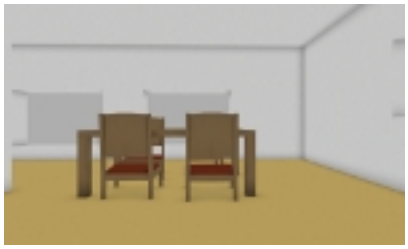
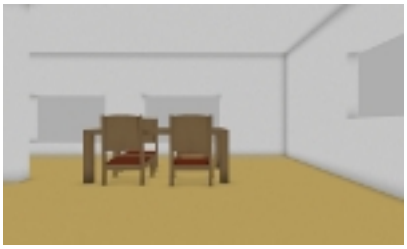
	Imagen real	Imagen estimada
Imagen 12.jpg		
Imagen 13.jpg		
Imagen 14.jpg		
Imagen 15.jpg		

Tabla 7.21: Comparativa entre las imágenes reales y las imágenes estimadas por el sistema de *tracking* con descriptores para la prueba 2 (ver Tabla 7.19).

7.3.2. Pruebas en entorno real

Para estas pruebas se ha empleado una serie de capturas realizadas en el mismo escenario real que el utilizado en las pruebas con los otros dos tipos de posicionamiento. Durante estas pruebas se ha utilizado la configuración más precisa para este módulo, extraida de las pruebas en entorno virtual.

Se ha ejecutado la misma batería de pruebas en dos circunstancias diferentes. En la primera situación, se ha medido la estimación sin la utilización de histórico de posiciones. En la segunda sí se ha utilizado histórico de posiciones. Esto ha servido para observar en qué medida mejoran las estimaciones al incorporar el histórico en el sistema y, de esta forma, justificar su utilización.

7.3.2.1. Pruebas sin histórico de posiciones

La tabla 7.22 muestra los resultados hechos con el módulo de *tracking* con descriptores sin el apoyo del histórico de posiciones. Al comienzo de la ejecución de la aplicación en los que no existe histórico de percepciones, este tipo de posicionador tendrá mucha menos precisión que al cabo de un tiempo en que la aplicación esté en marcha.

	Pos. real	**Or. real**	**Pos. estim.**	**Or. estim.**	**Error pos.**	**Error or.**
1	(5.0,4.0)	0^o	(6.0,16.0)	180^o	12.04	180^o
2	(6.0,8.0)	180^o	(2.0,6.0)	45^o	4.47	135^o
3	(6.0,8.0)	135^o	(6.0,12.0)	45^o	4	180^o
4	(6.0,4.0)	0^o	(6.0,16.0)	180^o	12	180^o
5	(8.0,8.0)	45^o	(6.0,10.0)	45^o	2.82	0^o

Tabla 7.22: Resultados para el *tracking* con descriptores en escenario real sin histórico.

Como puede observarse, existe bastante imprecisión en la estimación, especialmente en la orientación. Esto es debido en gran parte a que el escenario posee una gran homogeneidad en cuanto a su estructura. En la figura 7.15, que corresponde al resultado de la prueba 1 (ver Tabla 7.22), puede observarse como hay un gran parecido geométrico entre la posición

real (imagen izquierda) y la estimada por el posicionador (imagen derecha), sin embargo, pertenecen a posiciones muy diferentes del escenario.

Figura 7.15: Comparación de imágenes para prueba 1 del *tracking* con descriptores en escenario real

7.3.2.2. Pruebas con histórico de posiciones

En esta ocasión se han realizado las mismas pruebas que las realizadas anteriormente, pero en esta ocasión se ha incorporado el dato del histórico de posiciones. Ahora el sistema posicionador tiene datos de la última posición conocida para inferir una estimación mejor.

En la tabla 7.23 pueden observarse los resultados de estas pruebas.

	Pos. real	**Or. real**	**Pos. estim.**	**Or. estim.**	**Error pos.**	**Error or.**
1	(5.0,4.0)	0º	(8.0,12.0)	180º	8.94	180º
2	(6.0,8.0)	180º	(6.0,12.0)	180º	4	0º
3	(6.0,8.0)	135º	(6.0,10.0)	180º	2	45º
4	(6.0,4.0)	0º	(6.0,10.0)	180º	6	180º
5	(8.0,8.0)	45º	(6.0,10.0)	45º	2.82	0º

Tabla 7.23: Resultados para el *tracking* con descriptores en escenario real con histórico.

Como puede comprobarse, se ha conseguido una mejora notoria en las estimaciones en todas las pruebas, lo que demuestra la utilidad de proporcionar al sistema la referencia de

la última posición conocida. De esta manera queda justificada la utilización del histórico de posiciones en el módulo de *tracking* con descriptores.

7.4. *Profiling*

El *Profiling* es una técnica utilizada en ingeniería del software con la que se persigue medir el rendimiento de una aplicación de forma dinámica, es decir, durante su ejecución. De esta forma es posible medir qué partes del programa consumen más tiempo de uso de la CPU para poder conocer los posibles "cuellos de botella" que existan y poder realizar optimizaciones sobre ellos.

Las herramientas utilizadas para realizar el *profiling* son denominadas *profilers*. Para estas pruebas, el *profiler* utilizado ha sido ***Gprof***, herramienta GNU/Linux compatible con los compiladores de la familia *gcc* empleados en este proyecto.

A continuación se mostrarán 4 medidas de rendimiento. En las tres primeras se mostrará el rendimiento obtenido para cada módulo de *tracking* por separado. La última medida integrará la ejecución conjunta de los 3 módulos.

7.4.1. *Profiling* del módulo de *tracking* con marcas

La ejecución con la que se ha medido este módulo ha consistido en 15 peticiones de posicionamiento. Con ello se han obtenido los resultados mostrados en la figura 7.16.

Name (location)	Time	%Time
▼ Summary	70.000ms	100,0%
▶ arLabeling.c	30.000ms	42,86%
▶ arGetTransMat2.c	20.000ms	28,57%
▶ arGetCode.c	10.000ms	14,29%
▶ arGetTransMat3.c	10.000ms	14,29%

Figura 7.16: Resultados *profiling* del *tracking* con marcas

El tiempo de ejecución ha sido aproximadamente de 70ms, repartidos entre las clases indicadas. El resto de clases que han intervenido en la ejecución no se han indicado puesto

que su tiempo de utilización ha sido muy pequeño en relación al total. El conjunto de clases más utilizadas corresponden a la biblioteca *ARToolkit*, ya que son las funciones que realizan la detección de las marcas y las que proporcionan las matrices de transformaciones, que son las operaciones críticas para este módulo de *tracking*.

7.4.2. *Profiling* del módulo de *tracking* estéreo

Para esta ejecución se han realizado 15 peticiones de posicionamiento. La figura 7.17 muestra los resultados.

Name (location)	Time	%Time
Summary	20.000ms	100,0%
??	20.000ms	100,0%
std::basic_string<char, std::char_traits<char>	20.000ms	100,0%

Figura 7.17: Resultados *profiling* del *tracking* estéreo

El tiempo de ejecución utilizado ha sido menor que para el *tracking* con marcas para el mismo número de peticiones de posición (20ms). Prácticamente todo el total de tiempo ha sido empleado en operaciones matriciales.

7.4.3. *Profiling* del módulo de *tracking* con descriptores

Para esta ejecución se han realizado 15 peticiones de posicionamiento. En la figura 7.18 pueden observarse los resultados.

Lo más significativo que puede extraerse de los resultados obtenidos es que el tiempo de ejecución es mucho más elevado en relación a los otros dos módulos (15.55s). La mayoría del tiempo de ejecución es empleado en la extracción de descriptores, siendo el descriptor *Scalable Color* el más lento. El resto son operaciones propias de la biblioteca MPEG-7FexLib para la extracción y comparación de descriptores.

Name (location)	Time	%Time
Summary	15.550s	100,0%
ShiftRightLineLoad (??:-1)	3.700s	23,79%
XM::ColorLayoutExtractionTool::CreateSmallImage(m	3.270s	21,03%
RGB_To_HSV (??:-1)	2.500s	16,08%
XM::EdgeHistogramExtractionTool::extract(momvop*)	1.230s	7,91%
Convert::ipl2momvop(cv::Mat&, momvop*) (??:-1)	850.000ms	5,47%
pixeltocolorhist1 (??:-1)	770.000ms	4,95%
pixeltohistogram1 (??:-1)	700.000ms	4,5%
QuantScalableUniform1(pixel_struct*, pixel_struct*,	570.000ms	3,67%
XM::EdgeHistogramExtractionTool::GetEdgeFeature	360.000ms	2,32%
XM::ScalableColorDescriptor::SetCoefficient(unsigne	360.000ms	2,32%
getuchardata (??:-1)	270.000ms	1,74%
Convert::setMaskValue(momvop*, int) (??:-1)	230.000ms	1,48%
cv::Vec<unsigned char, 3>& cv::Mat::at<cv::Vec<un	180.000ms	1,16%
intra_proc (??:-1)	160.000ms	1,03%
XM::ScalableColorDescriptor::GetCoefficient(unsigne	120.000ms	0,77%
cv::Vec<unsigned char, 3>::operator[](int) (??:-1)	120.000ms	0,77%
Preload (??:-1)	20.000ms	0,13%
RGB_To_YUV (??:-1)	20.000ms	0,13%

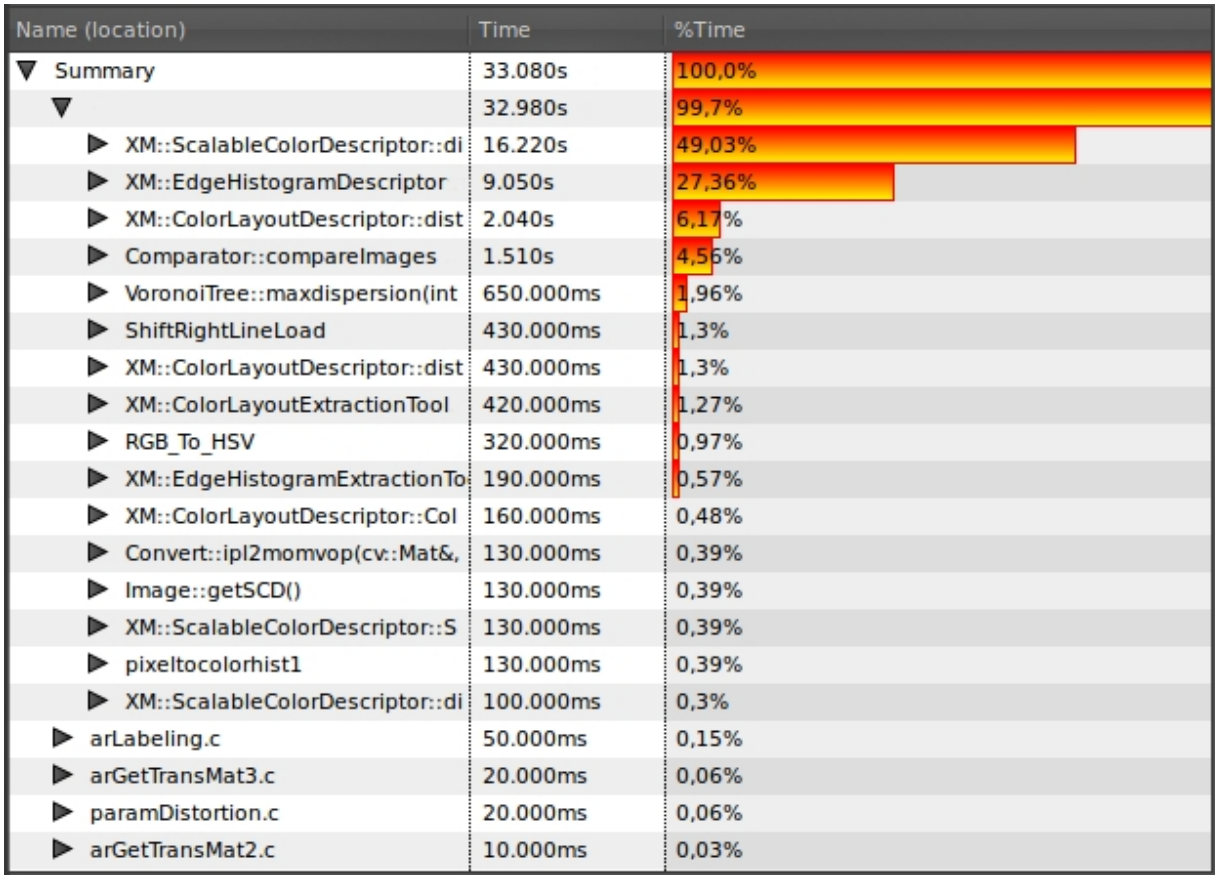
Figura 7.18: Resultados *profiling* del *tracking* con descriptores

7.4.4. *Profiling* con integración de los módulos

Para esta última prueba de rendimiento se ha realizado una ejecución con 15 peticiones de posicionamiento para cada módulo de *tracking*. Los resultados pueden visualizarse en la figura 7.19.

Casi todo el porcentaje de tiempo empleado en esta ejecución ha sido utilizado por el *tracking* con descriptores en la tarea de extracción de los mismos. Solamente un pequeño porcentaje ha sido utilizado en el *tracking* con marcas, como puede observarse al final de la tabla que muestra la figura 7.19. El tiempo empleado por el módulo de *tracking* estéreo es muy pequeño y apenas es reseñable.

Por tanto, habría que buscar formas de obtener una optimización de los algoritmos de extracción de descriptores con el objetivo de reducir los tiempos de ejecución.



Name (location)	Time	%Time
Summary	33.080s	100,0%
	32.980s	99,7%
XM::ScalableColorDescriptor::di	16.220s	49,03%
XM::EdgeHistogramDescriptor	9.050s	27,36%
XM::ColorLayoutDescriptor::dist	2.040s	6,17%
Comparator::compareImages	1.510s	4,56%
VoronoiTree::maxdispersion(int	650.000ms	1,96%
ShiftRightLineLoad	430.000ms	1,3%
XM::ColorLayoutDescriptor::dist	430.000ms	1,3%
XM::ColorLayoutExtractionTool	420.000ms	1,27%
RGB_To_HSV	320.000ms	0,97%
XM::EdgeHistogramExtractionTo	190.000ms	0,57%
XM::ColorLayoutDescriptor::Col	160.000ms	0,48%
Convert::ipl2momvop(cv::Mat&,	130.000ms	0,39%
Image::getSCD()	130.000ms	0,39%
XM::ScalableColorDescriptor::S	130.000ms	0,39%
pixeltocolorhist1	130.000ms	0,39%
XM::ScalableColorDescriptor::di	100.000ms	0,3%
arLabeling.c	50.000ms	0,15%
arGetTransMat3.c	20.000ms	0,06%
paramDistortion.c	20.000ms	0,06%
arGetTransMat2.c	10.000ms	0,03%

Figura 7.19: Resultados *profiling* con integración de los tres módulos de *tracking*

Estos datos también demuestran que el módulo de posicionamiento no debe abusar de las peticiones al módulo de *tracking* con descriptores, puesto que es el que más tiempo requiere y el requisito de tener un sistema de tiempo real puede verse incumplido en caso contrario.

Capítulo 8

Conclusiones y propuestas

En este capítulo se realiza una valoración sobre el cumplimiento de los objetivos formulados en el capítulo 3 a lo largo del tiempo de desarrollo. Se definen un conjunto de propuestas de trabajo futuro para la mejora y ampliación del prototipo funcional que ha sido desarrollado. Por último, se exponen una serie de conclusiones personales sobre lo que ha supuesto la realización de este Proyecto de Fin de Carrera.

8.1. Objetivos alcanzados

Con el desarrollo de Arges se ha conseguido realizar un sistema de posicionamiento para interiores de edificios utilizando técnicas de Visión por Computador y Realidad Aumentada.

En su desarrollo han sido implementados tres tipos de sistemas de *tracking*, que se encargan de proporcionar los datos de posicionamiento a través de diferentes procedimientos.

Se ha elaborado un módulo de *tracking* empleando la biblioteca *ARToolkit*, que es igualmente empleada en el proyecto ElCano. Gracias a este tipo de posicionamiento se consiguen resultados muy exactos al visualizar las marcas.

Des igual modo, se han implementado dos métodos de *tracking* sin marcas. Éstos sirven de apoyo en aquellas ocasiones en las que las marcas empleadas en el sistema de *tracking* comentado anteriormente no son visibles. De esta forma se persigue que el sistema ofrezca resultados de posicionamiento en un mayor rango de situaciones, aunque sean de peor calidad.

El primero de los métodos de posicionamiento sin marcas basa su funcionamiento en el análisis de la geometría epipolar de un sistema estéreo simulado con una sola cámara. Este método, a pesar de ser menos preciso, constituye un modo fiable que permite realizar registro en espacios grandes.

El segundo de los métodos de posicionamiento sin marcas se ha basado en la extracción de descriptores de imagen y su comparación. Se ha conseguido desarrollar un módulo de gran precisión, especialmente con el apoyo de un histórico de posiciones.

Estos métodos de *tracking* constituyen módulos independientes que pueden ser utilizados individualmente o fusionando su salida como en el proyecto ElCano. Existe un bajo acoplamiento entre cada uno de los módulos, de forma que el sistema puede ser ampliado fácilmente, por ejemplo con nuevos métodos de posicionamiento.

Como se muestra en el capítulo de Resultados, el sistema ha sido probado, tanto en entornos controlados (escenarios virtuales), como en el entorno real. Las pruebas se han realizado individualmente en cada módulo para probar el correcto funcionamiento por separado.

Arges constituye un sistema en tiempo real, que permite obtener datos de posicionamiento rápidamente con una tasa de frames por segundo interactiva. En el proyecto ElCano ha sido utilizado en dispositivos portátiles con capacidad de cómputo limitada.

Así mismo, es un sistema multiplataforma, gracias a la utilización de bibliotecas libres y el desarrollo de un código portable.

8.2. Propuestas de trabajo futuro

Arges atesora una gran utilidad, no solo para personas con discapacidad, sino para todo aquellos usuarios que requieren de información precisa de dónde se encuentran dentro de un edificio. Se pretende ampliar y mejorar la funcionalidad del prototipo para hacerlo más accesible y que, de esta forma, su uso pueda estar al alcance de cualquier usuario.

A continuación se detallan algunas propuestas de trabajo futuro para este proyecto:

- **Mejora del sistema de *tracking* estéreo**

  Se pretende buscar formas de mejorar los resultados obtenidos con el sistema de *tracking* estéreo. Una vez comprobada la viabilidad de utilizar las restricciones de la ge-

ometría epipolar para la construcción de un sistema de *tracking*, el objetivo más inmediato sería buscar y aplicar algoritmos que permitan una mayor precisión en la estimación del posicionamiento.

Con las características del registro estéreo es posible la estimación de puntos 3D en la escena, como se ha demostrado en las pruebas. Partiendo de este hecho, se utilizarían algoritmos que permitan la construcción de mapas del entorno a través de la visualización directa y el almacenamiento de los puntos 3D calculados.

Tras una fase de entrenamiento en la que se habrá almacenado en el sistema un mapa 3D completo en forma de nube de puntos del entorno en el que se trabajará, la fase de *tracking* se encargará de utilizar algoritmos que comparen los puntos visualizados en tiempo real con los del mapa almacenado. De esta forma se puede obtener un posicionamiento absoluto, al igual que con el *tracking* con marcas, lo cual, mejoraría en gran medida la precisión de este módulo de posicionamiento.

Estimación de tiempo: 2-3 meses.

- **Desarrollo de métodos de fusión de posicionamiento avanzados**

  Es de suma importancia la gestión que realiza el sistema de cuándo poner a funcionar cada método de posicionamiento. Es necesario buscar un equilibrio en el que siempre se utilice el modo más preciso de obtener posicionamiento y, al mismo tiempo no sobrecargar el sistema haciendo demasiadas peticiones de registro. De otra forma se pierde la capacidad de tener un sistema que funcione en tiempo real.

  Por esta razón, se estima oportuno el desarrollo de heurísticas que permitan la fusión del posicionamiento obtenido con cada método. De esta forma se persiguen dos objetivos: buscar una mejor gestión en las peticiones de registro y desarrollar formas de valorar en cada momento la calidad del posicionamiento obtenido con cada módulo de *tracking*.

  Estimación de tiempo: 3-4 semanas

- **Soporte para Android**

  Arges constituye un sistema de tiempo real que requiere de dispositivos con baja capacidad de cómputo para su funcionamiento. Los dispositivos con baja capacidad de

cómputo por excelencia en los últimos años son los terminales móviles, por esta razón, sería deseable la búsqueda de formas de introducir el sistema en este tipo de aparatos, con el fin de hacerlo más usable para el usuario final.

La biblioteca OpenCV, la cuál constituye la base software de Arges, permite ser utilizada para funcionar con Android por lo que se podría utilizar en cualquier dispositivo móvil que ejecutara este sistema operativo. Esta capacidad puede ser aprovechada para portar el código de Arges a este sistema operativo de forma relativamente sencilla y sin perder su capacidad como sistema de tiempo real.

La cuota de mercado de los móviles con sistema operativo Android fue del 52.5 % en 2011[1]. Esto supone las ventas más altas en el sector de entre todos los sistemas operativos para móviles. Por esta razón, se persigue explotar esta posibilidad para sacar a mercado Arges como aplicación para móviles Android.

Estimación de tiempo: 1-2 meses

- **Creación de un sitio web**

Tras el desarrollo de este proyecto, se ha observado que existe cierta cantidad de información acerca de diversos métodos de posicionamiento, especialmente en cuestión de *tracking* sin marcas, sin embargo esta información se encuentra muy dispersa. Además, existen pocos ejemplos funcionales que implementen los algoritmos que han sido utilizados en Arges.

Por esta razón se estima oportuna y poco costosa, la creación de un sitio web en donde se intente unificar el conocimiento en este área, mostrando los avances que se den en el futuro acerca de este tema y en el que se muestren la mejoras del proyecto, así como las últimas versiones disponibles para ser descargadas del mismo. Desde el sitio web cualquiera podrá descargar el software de Arges bajo licencia GPLv3 (ver Anexo E) y así poder contribuir en el desarrollo de los sistemas de VC y RA.

[1]Dato estimado según la web de Gartner: http://www.gartner.com

En principio se abriría algún espacio web de tipo blog, en el que se muetren pequeños ejemplos funcionales de los aspectos más relevantes de Arges, con el fin de hacer extensible el conocimiento sobre su funcionamiento a todo el mundo. Este blog daría acceso a todo el código de Arges, que se dejaría almacenado y abierto a todo el que desee utilizarlo o ampliarlo en un espacio de Google Code.

Estimación de tiempo: 1-2 semanas

8.3. Conclusión personal

El desarrollo de un Proyecto de Fin de Carrera de esta índole, en el que se combina una gran cantidad de trabajo de investigación y estudio con la implementación del mismo, me ha demostrado la riqueza que existe en el campo de la Ingeniería Informática, no solo a nivel tecnológico, ya que existen multitud de productos software y hardware destinados a la solución de prácticamente cualquier problema, sino a nivel cognitivo. El ingeniero en Informática es aquella persona capaz de extraer conocimiento de cualquier área del saber y utilizarlo para la creación de productos que faciliten las tareas a las personas que trabajan en dicho área.

Durante los años de estudio de la carrera, se ha profundizado en el estudio de un conjunto muy grande de metodologías, lenguajes de programación, algoritmos, conceptos matemáticos, etc. Sin embargo, el abordar el Proyecto de Fin de Carrera, demuestra que al final solo se conoce una pequeña parte del mundo de la computación, que es necesario aprender mucho y que la vida profesional será un contínuo aprendizaje y enriquecimiento del saber. En mi opinión, esto es lo que le da mayor atractivo a la Ingeniería Informática, el poder crear cosas nuevas o mejorar las ya existentes, desde pequeños programas hasta grandes aplicaciones de gestión para empresas.

En este momento de la historia en que nos encontramos en el que la delicada situación económica hace estragos, los ingenieros en Informática debemos ser más que nunca emprendedores, ya que lo que hemos estudiado a lo largo de 5 años nos otorga unas bases sólidas para poder llevar a buen puerto cualquier idea técnologica.

En mi caso, el desarrollo de este proyecto me ha servido para enfocar mi interés en el

campo de los Gráficos por Computador. He aprendido mucho durante este tiempo de creación de Arges en esta materia y deseo seguir ampliando mis conocimientos para poder dedicar mi vida profesional a este área.

Anexo A

Manual de usuario para la calibración de la cámara

Para un correcto funcionamiento del sistema, la aplicación debe tener una correcta parametrización de la cámara, de tal forma que se conozcan lo más aproximadamente posible las equivalencias entre los valores internos del programa y los valores del mundo. De otra forma, el posicionamiento no sería correcto.

Antes de utilizar el programa por primera vez con una cámara con la que nunca se haya utilizado, es necesario efectuar una calibración. Para ello, se facilita un módulo de calibración, cuyo funcionamiento se explica en este apéndice de la forma más clara y didáctica posible.

A.1. Preparativos

El módulo de calibración podemos encontrarlo en el directorio *Calibration* de la aplicación. Previamente, deberemos hacernos con un patrón de calibración en forma de tablero de ajedrez. Se incluye un patrón de este tipo dentro del mismo directorio con el nombre *calibcheckerboard.png*, listo para imprimir.

Una vez tengamos impreso nuestro patrón de calibración, es necesario efectuar una medición. La única medida que necesita el programa es el lado de un cuadrado del tablero de ajedrez. Ésta será la referencia de la que dispondrá el posicionador para conocer las dimensiones del mundo.

Tras disponer de estos dos ingredientes, estaremos en disposición de ejecutar el módulo de calibración de la cámara.

A.2. Ejecución de la calibración

Dentro del directorio *Calibration* podemos encontrar un archivo *Makefile*, que nos permitirá compilar el módulo de calibración.

Tras ejecutar *make* dentro del directorio *Calibration* obtendremos el ejecutable con nombre *"calibration"*. Para ejecutarlo es necesario obligatoriamente pasarle al programa la medida del cuadrado que hemos indicado en la anterior sección. La ejecución básica del programa tendrá la siguiente sintáxis:

```
./calibration -s <medida_del_lado_del_cuadrado>
```

Adicionalmente, existen una serie de parámetros de ejecución opcionales:

- `-w <ancho_tablero>`: esta opción sirve para indicar el número de esquinas existentes en el interior del tablero de ajedrez para el ancho. El número por defecto es 9, que se corresponde con la medida del patrón proporcionado con la aplicación.

- `-h <alto_tablero>`: esta opción sirve para indicar el número de esquinas existentes en el interior del tablero de ajedrez para el alto. El número por defecto es 6, que se corresponde con la medida del patrón proporcionado con la aplicación.

- `-i <archivo_XML>`: con esta opción podremos proporcionar al programa un archivo XML con los nombres de las imágenes ya listas para ser utilizadas en el cálculo de la calibración.

El archivo XML proporcionado al programa debe tener una forma específica para que sea legible. Podemos ver un ejemplo en la figura A.1.

```
<?xml version="1.0"?>
<opencv_storage>
<images>
  "./calib_frames/IMG1L.jpg" "./calib_frames/IMG1R.jpg"
  "./calib_frames/IMG2L.jpg" "./calib_frames/IMG2R.jpg"
  "./calib_frames/IMG3L.jpg" "./calib_frames/IMG3R.jpg"
</images>
</opencv_storage>
```

Figura A.1: Ejemplo de archivo XML para la aplicación

En la figura A.1 podemos observar como las imágenes se van indicando por pares. Este es un ejemplo de fichero XML generado por el programa de calibración. Los nombres de imágenes que terminan en "L" son los correspondientes a la vista de la cámara izquierda; análogamente, los terminados en "R", son los correspondientes a la vista de la cámara derecha.

El modo de proceder a la hora de ejecutar la calibración es proporcionar al programa una serie de capturas de pantalla en las que se visualice el patrón de calibración. Las capturas de pantalla deben realizarse de dos en dos, de forma que se imite a la visión estereoscópica.

Una vez ejecutado el programa, se abrirá la cámara y podremos capturar las imágenes. Para realizar una captura, pulsaremos la tecla "C". Cuando pulsemos la tecla, el programa realizará la captura del frame actual y nos dará dos segundos para mover unos centímetros el patrón hacia un lado, intentando rotarlo lo menos posible. Transcurrido ese tiempo, se realizará una segunda captura. Esta acción la realizaremos más de una vez para hacer la calibración más exacta. Cuantas más muestras tenga el programa, más exacta será la calibración. Una vez que hayamos tomado las suficientes muestras, pulsaremos la tecla "Esc" y el programa realizará la calibración. El conjunto de imágenes serán guardadas en el directorio *Calibration/calib_frames*.

Como resultado de la calibración, obtendremos un archivo YAML con el nombre *calib_parameters.yml*, el cual contendrá los parámetros intrínsecos de la cámara y otros datos útiles para el módulo de posicionamiento.

A.3. Ejemplo de ejecución

Un ejemplo de ejecución podría ser llamando al programa de esta manera:

```
./calibration -s 0.025
```

Esta llamada al programa de calibración implica que un cuadrado de nuestro patrón de calibración tiene 25mm de lado. Tras realizar esta llamada, procederemos a capturar los pares de imágenes. En la figura A.2 vemos cómo se han proporcionado tres pares de imágenes al programa.

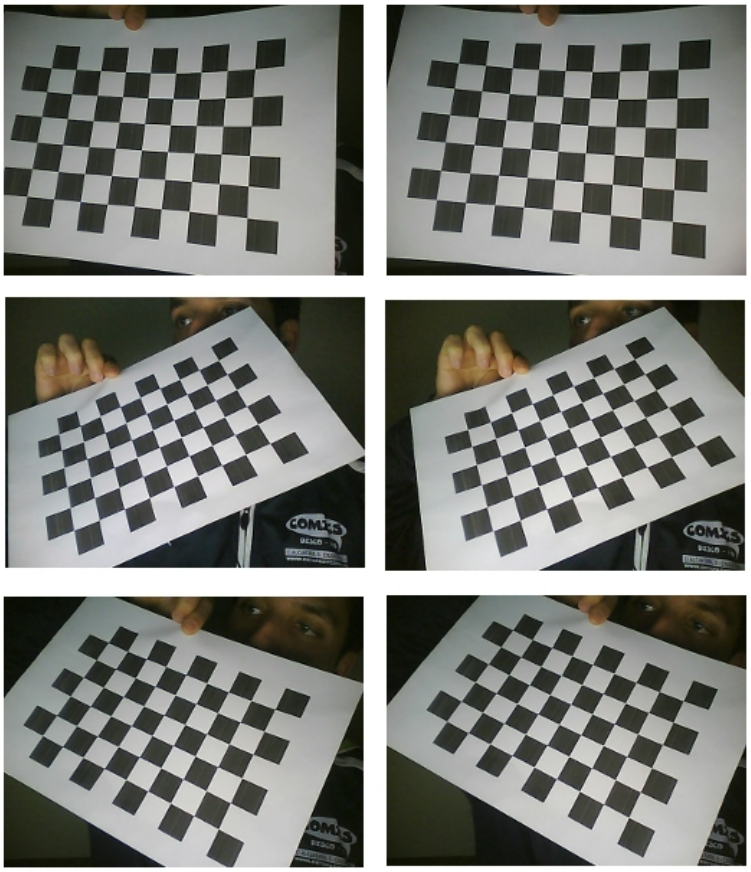


Figura A.2: Capturas realizadas para la calibración

Después de la captura, el programa buscará las esquinas formadas por los cuadrados del patrón de ajedrez y nos mostrará por pantalla dónde las ha encontrado. En caso de que en alguna de las capturas el programa no encuentre las esquinas, la calibración no se ejecutará correctamente. Algunas de las causas por las que el programa pueda no encontrar las esquinas pueden ser una incorrecta iluminación (ya sea por exceso de luz, el cual puede dar lugar a brillos o por una falta de luz, que impedirá al programa distinguir correctamente los colores), o porque el patrón no se coloque correctamente. Se recomienda pegar el patrón de calibración sobre algún soporte rígido para evitar que se doble y obtener resultados indeseados. Si todo ha ido bien el programa reconocerá correctamente las esquinas, como se observa en la figura A.3 y obtendremos los parámetros intrínsecos de la cámara en el archivo YAML.

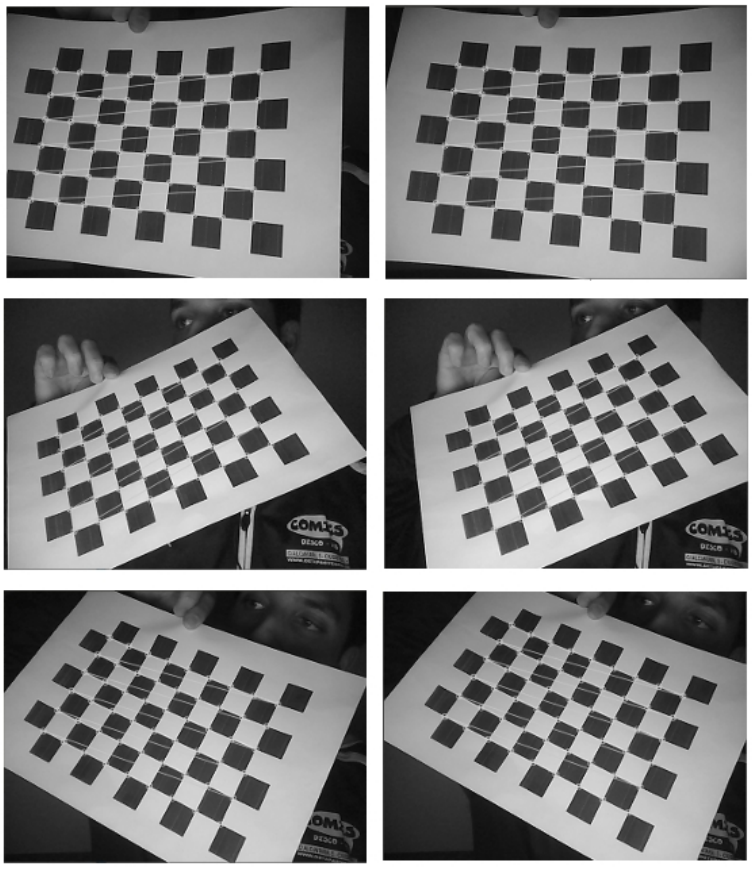


Figura A.3: Detección de esquinas en el patrón de calibración

Anexo B

Diagramas de clases

En este anexo se incluyen los diferentes diagramas de clases UML que describen la arquitectura del sistema.

En primer lugar se muestra el diagrama de clases general que describe la organización del sistema y cómo se relacionan cada uno de los módulos al ser pasados a clases. El segundo diagrama indica la organización de clases para el *tracking* con descriptores, ya que su funcionamiento requiere de una estructura de clases especialmente indicada para realizar el conjunto de operaciones necesario para obtener buenos resultados, en conjunción con la biblioteca MPEG-7FexLib.

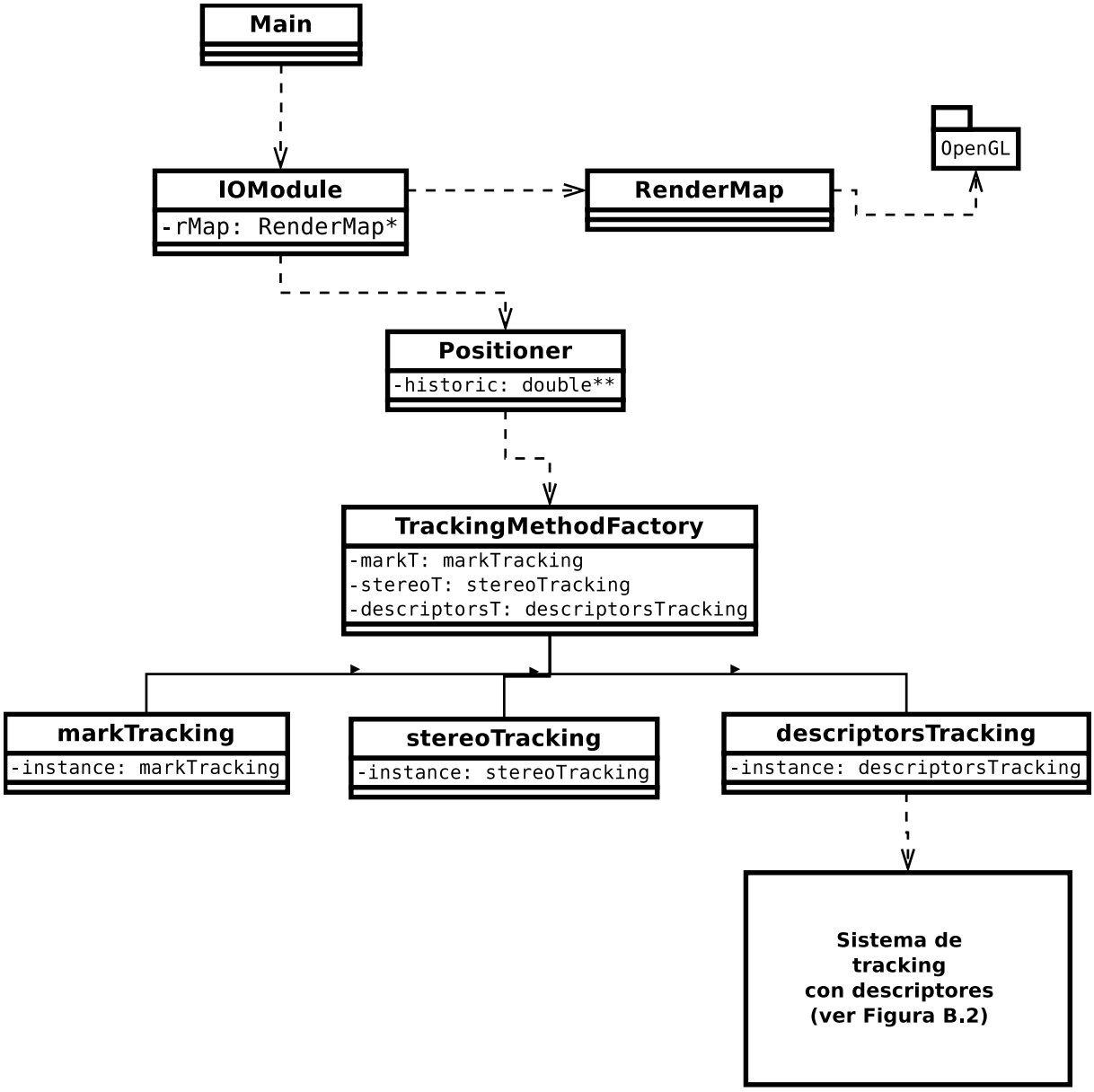


Figura B.1: Diagrama de clases para el sistema ARGES

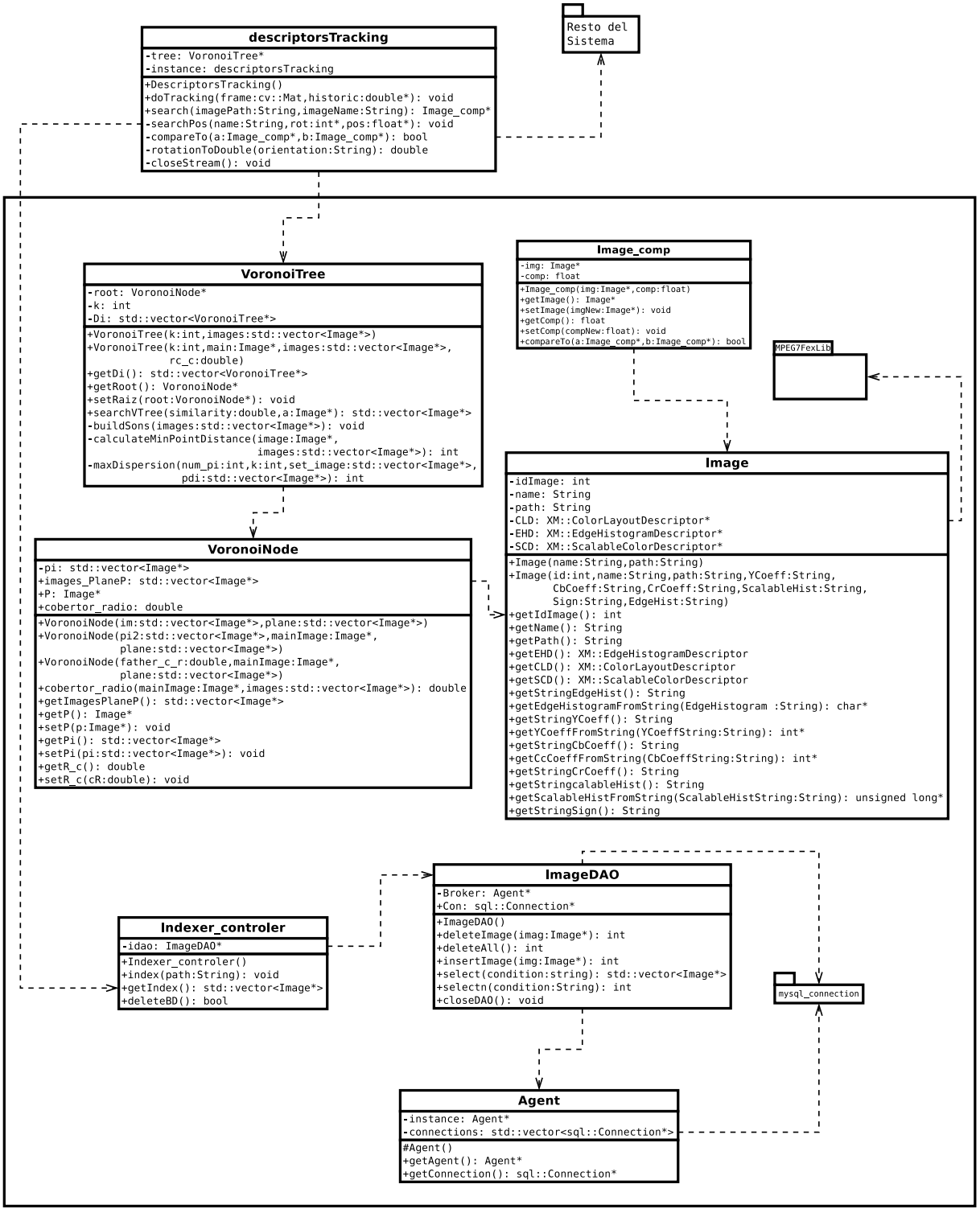


Figura B.2: Diagrama de clases para el módulo de *tracking* con descriptores

Anexo C

Imágenes de las pruebas en entorno real

En este anexo se incluyen aquellas capturas de imagen que han sido utilizadas en las pruebas de los módulos de *tracking* dentro del entorno real, correspondiente a la planta baja de la Escuela Superior de Informática.

Figura C.1: Imágenes utilizadas en las pruebas de *tracking* con marcas en entorno real.

Figura C.2: Imágenes utilizadas en las pruebas de *tracking* estéreo en entorno real.

Figura C.3: Imágenes utilizadas en las pruebas de *tracking* con descriptores en entorno real.

Anexo D

Código fuente

Debido a la extensión del código fuente de Arges, éste se incluye únicamente en su versión electrónica en el CD adjunto a este documento.

La estructura de directorios con la que se ha organizado este código fuente es la siguiente:

- **/Calibration:** contiene todos los archivos necesarios para compilar y ejecutar la utilidad de calibración de la cámara.
    - **/calib_frames:** contiene los frames guardados al ejecutar la utilidad de calibración de la cámara.
- **/Deal:** contiene aquellas clases que se encargan de los procesos de negocio internos del sistema.
- **/Include:** contiene archivos de cabecera necesarios para el correcto funcionamiento del sistema.
- **/Interface:** contiene las clases correspondientes al módulo de entrada/salida.
- **/MPEG7FexLib:** contiene las clases útiles para el sistema de la biblioteca MPEG7-FexLib.
    - **/AddressLib:** contiene clases de apoyo para la extracción de descriptores.

    - **/Descriptors:** contiene las clases que representan cada uno de los descriptores de imagen utilizados en el sistema.
    - **/Extraction:** contienen las clases encargadas de extraer los valores de imagen correspondientes a los descriptores.

- **/Persistence:** contiene las clases que se encargan de la gestión de la base de datos.
- **/Scope:** contiene las clases de dominio del sistema.
- **/Tracking:** contiene aquellas clases que se encargan de gestionar los diferentes métodos de *tracking* del sistema.

    - **/patterns:** contiene los archivos de patrones de marcas para el método de posicionamiento con marcas.

Anexo E

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. http://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this
license document, but changing it is not allowed.

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

   b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

   c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

   d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

   A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

   You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

   a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

   b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

   c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

   d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

   e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

   You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

    A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

    Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

    In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

    If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

    If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

    A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered

work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND /OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does .>

Copyright (C) <textyear>  <name of author>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program>  Copyright (C) <year>  <name of author>

This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `http://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `http://www.gnu.org/philosophy/why-not-lgpl.html`.

Bibliografía

[1] Ronald T. Azuma. *A survey of Augmented Reality*. Presence: Teleoperators and Virtual Environments, 6(4): pp. 355-385, Agosto 1997.

[2] Gerald J. Chader, James Weiland and Mark S. Humayun. *Artificial vision: needs, functioning, and testing of a retinal electronic prosthesis* Progress in Brain Research, Vol. 175, 2009

[3] A. Chow, V. Chow, K. Packo, J. Pollack, G. Peyman, R. Schuchard. *The artificial silicone retina microchip for the treatment of vision loss from retinitis pigmentosa.* Archives of Ophthalmology, 122, pp. 1156-1157, 2004.

[4] D. Marimon Sanjuan. (Tesis) *Advances in Top-down and Bottom-up approaches to video-based camera tracking.*

[5] *People with disabilities: employment and accessibility by stages for people with disabilities in the EU. Post-2010 Lisbon Strategy.* http://eescopinions.eesc.europa.eu/EESCopinionDocument.aspx?identifier=ces\soc\soc363\ces449-2010_ac.doc&language=EN

[6] Escuela Superior de Informática (UCLM). *Normativa del proyecto fin de carrera.* http://webpub.esi.uclm.es/archivos/89/NormativaPFC2007

[7] Roberto García García. (PFC) *Sistema de Odometría Visual para la mejora del Posicionamiento Global de un vehículo.* Escuela Politécnica Superior. Universidad de Alcalá.

[8] Leonard and Durrant-Whyte. *Mobile robot localization by tracking geometric beacons.* IEEE Transactions on Robotics and Automation, Vol. 7, N.3, June 1991.

[9] R. Smith, M. Self and P. Cheeseman. textitEstimating uncertain spatial relationships in robotics. I.J. Cox, G.T. Wilfong, Editors , Autonomous Robot Vehnicles, Springer, Berlin (1990), pp. 167-193.

[10] David G. Lowe. *Object recognition from local scale-invariant features.* Proceedings of the International Conference on Computer Vision. 2. pp. 1150-1157. (1999)

[11] Snavely, N., Seitz, S.M., Szeliski, R. *Photo tourism: exploring photo collections in 3D.* ACM Trans. Graph 25(3), 835-846 (2006)

[12] Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool. *SURF: Speeded Up Robust Features.* Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346–359, 2008

[13] J. Shi and C. Tomasi. *Good features to track.* TR 93-1399, Cornell U., 1993.

[14] G. Bishop and G. Welch. *An Introduction to the Kalman Filter.* SIGGRAPH 2001, Course 8, 2001.

[15] Tomas Akenine-Moller, Eric Haines, and Naty Hoffman. *Real-Time Rendering (Third Edition).* AK Peters, 2008.

[16] T. Camus. *Real-Time Optical Flow* PhD Thesis, Brown University. Technical Report CS-94-36, 1994.

[17] B.K.P. Horn and B.G. Schunck. *Determining optical flow.* Artificial Intelligence, vol 17, pp 185-203, 1981.

[18] Andrew J. Davison et al. *MonoSLAM: Real-Time Single Camera SLAM.* IEEE Trans. Pattern Analysis and Machine Intelligence, 2007.

[19] Gary Bradski and Adrian Kaehler. *Learning OpenCV.* O'Reilly

[20] B. D. Lucas and T. Kanade. *An iterative image registration technique with an application to stereo vision.* Proceedings of Imaging Understanding Workshop, pages 121-130.

[21] R. Martínez, C. Sagués , J.J. Guerrero. *Corrección visual de un robot móvil con homografías.* XXIV Jornadas de Automática, León, Septiembre 2003.

[22] R. I. Hartley. *Theory and practice of projective rectification* International Journal of Computer Vision 35 (1998): 115-127.

[23] J.Y. Bouguet. *The calibration toolbox for Matlab: Stereo rectification algorithm.* http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/example5.html

[24] S.M. Rafizul Haque. *Singular Value Decomposition and Discrete Cosine Transform Based Image Watermarking* Master Thesis, Blekinge Institute of Technology. January, 2008

[25] J. R. Smith, P. Schirling. *Metadata Standards Roundup.* IEEE MultiMedia, 13(2):84-88, IEEE Computer Society Press, Los Alamitos, Ca, USA, April 2006

[26] Mason Woo et. al. *The Red Book: OpenGL Programmin Guide (Third Edition).* Addison Wesley.

[27] Brian W. Kernighan, Dennis M. Ritchie. *El lenguaje de programación C.* Pearson Education, 1991

[28] B. Stroustrup. *C++ Programming Language.* Addison Wesley, 3a edition, 1999.

[29] E. Malis and R. Cipolla. *Camera self-calibration from unknown planar structures enforcing the multi-view constraints between collineations.* IEEE Trans. Pattern Analysis and Machine Intelligence, 24(9):1268-1272, September 2002.

[30] Roberto Uribe, Fabián Granero y Gonzalo Navarro. *Voronoi-Tree: Optimización y Dinamismo.* Revista de la SCCC 6(1), 2005, Chile.

[31] S. Brin. *Near neighbor search in large metric spaces.* Proceedings of the 21st VLDB Conference, pp. 574-584, 1995.

[32] B. Albouy et al. *Fundamental matrix estimation revisited through a global 3D reconstruction framework.* Proceedings of Acivs 2004 (Advanced Concepts for Intelligent Vision Systems), Brussels, Belgium, 2004.

[33] P. H. S. Torr and D. W. Murray. *The Development and Comparison of Robust Methods for Estimating the Fundamental Matrix* International Journal of Computer Vision 24(3), 271-300 (1997).

[34] Cesar Augusto Puente Montejano. *Reconstrucción de objetos presentes en una escena utilizando el esquema de la siguiente mejor imagen.* Reporte del segundo avance: Triangulación masiva y Experimentación, 2005.

[35] G. Klein y T. Drummond. *Robust visual tracking for non-instrumented augmented reality.* 2003.

[36] David Pritchard. *Cloth Parameters and Motion Capture.* 2001.