UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

INGENIERÍA DE TELECOMUNICACIÓN

Trabajo Fin de Carrera

"Técnicas de reconstrucción volumétrica en tiempo real y su aplicación a la detección de personas "

Nombre del Alumno: Raquel Jalvo Peñín

Año: 2009

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

INGENIERÍA DE TELECOMUNICACIÓN

Trabajo Fin de Carrera

"Técnicas de reconstrucción volumétrica en tiempo real y su aplicación a la detección de personas"

Alumno: Raquel Jalvo Peñín

Director: Daniel Pizarro Pérez

Tribunal:

Presidente: D. Alfredo Gardel Vicente

Vocal 1º: D. Elena López Guillén

Vocal 2º: D. Daniel Pizarro Pérez

Calificación: .....

Fecha: .....

A mi padre por haber sido un ejemplo de generosidad y honestidad, a mi madre por su tesón y dedicación infinita a su familia y a mi hermana, por ser un ejemplo de superación e inteligencia.

Agradecimientos

Quiero agradecer en primer lugar a mi tutor Daniel Pizarro por su paciencia, su cercanía y su apoyo demostrado en el transcurso de la realización del proyecto. También deseo agradecer a los demás profesores y compañeros del espacio inteligente que trabajan día a día con mucho esfuerzo para mejorar poco a poco un proyecto global. Gracias a todos ellos, en el laboratorio reina el compañerismo y un buen ambiente de trabajo.

Gracias a mi familia, a la que dedico está obra en su totalidad, y a mis amigos de toda la vida, por estar siempre ahí, cuando se les necesita, en especial a Raul, por el ánimo que me da todos los días.

En último lugar, pero no menos importante, quiero agradecer en este proyecto a mis compañeros durante la carrera. El día que presente este libro estaré cerrando una bonita y dura etapa en la que algunos empezaron siendo solo compañeros y, hoy por hoy, han acabado siendo amigos y espero que por mucho tiempo.

Índice general

Índice de figuras

Índice de tablas

Parte I

Resumen

El presente proyecto propone una solución al problema del la reconstrucción volumétrica de un entorno delimitado y observado por un conjunto de cámaras calibradas y cuya adquisición está sincronizada. La solución presentada se enmarca dentro del concepto de Espacio Inteligente, en el cual una red de sensores (cámaras de vídeo en este caso), es controlada por un sistema automático de análisis con el objetivo de proporcionar un servicio a los usuarios.

La reconstrucción volumétrica planteada consiste en calcular previamente la ocupación tridimensional a partir de las siluetas de N cámaras. El volumen hallado corresponde con el denominado "Visual Hull" que se define como el mayor volumen encerrado por los conos visuales de las siluetas de los objetos que se encuentran dentro del entorno provenientes de las N cámaras.

Para hallar los conos visuales de las siluetas de cada cámara y su intersección, el espacio tridimensional se divide en planos paralelos al suelo y se calcula la intersección de las proyecciones de las siluetas de todas las cámaras sobre dichos planos. Una vez calculada la ocupación se ejecuta el algoritmo de Marching Cubes para realizar una malla formada por triángulos tridimensionales sobre la superficie de la ocupación. La orientación de los triángulos proporciona informarción para asignarles la textura adecuada proveniente de la cámara que mejor grado de visión posea del dicho triángulo.

Parte II

Memoria

Capítulo 1

Introducción

Este proyecto se ubica dentro de un entorno llamado "Espacio Inteligente". Un "Espacio inteligente" es un entorno delimitado por elementos sensoriales no intrusivos que posibilitan la adaptación del sistema al mundo que perciben y además, permiten a los usuarios interactuar con dicho entorno. La percepción y la interacción deben ser procesadas a través de sistemas empotrados con interconexión entre ellos. El objetivo final de este sistema debe ser prestar un servicio a los usuarios de un modo transparente y a través de una comunicación basada en el lenguaje natural.

Los elementos sensoriales son muy variados pero tienen en común que no son invasivos. Algunos ejemplos son micrófonos, ultrasonidos, sensores biométricos, infrarrojos, láser o como en este trabajo cámaras de vídeo, que posibilitan el uso de "Visión artificial". La "Visión Artificial" o "Visión Computacional" consiste en procesar imágenes para hallar características del mundo que se reflejan en ellas.

En la figura 1.1 se observa un ejemplo de "Espacio Inteligente" que percibe la escena a través de cuatro cámaras instaladas en red. En el espacio existen diferentes objetos: personas, robots teledirigidos por un enlace radio y otros obstáculos.

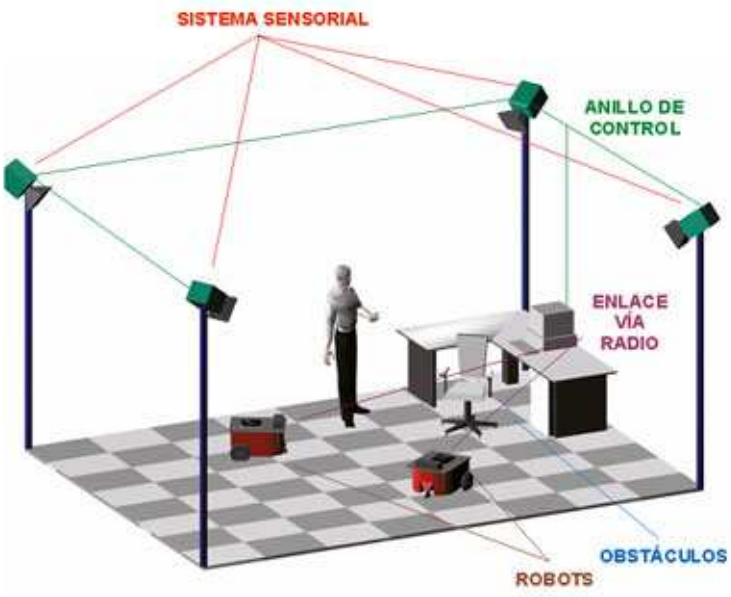


Figura 1.1: Ejemplo de Espacio Inteligente

En este trabajo se ha querido desarrollar un sistema que ubique en el espacio tridimensional diferentes objetos de formas arbitrarias donde las únicas entradas del sistema sean imágenes procedentes de cámaras de vídeo calibradas espacialmente. Para obtener la tridimensionalidad de un objeto con la mayor fidelidad se necesita información de todas sus posibles vistas, por lo que se deben utilizar varias cámaras. Cuantas más imágenes desde diferentes puntos de vista del objeto, la reconstrucción de éste se va a ajustar más a la realidad.

El objetivo principal del proyecto es fundir la información de las N cámaras para obtener una reconstrucción en tres dimensiones de los objetos que se encuentran dentro del entorno. El procesado de imágenes se va a realizar de modo paralelo en N servidores conectados a las N cámaras, para posteriormente fundir la información en un cliente para obtener una representación tridimensional fotorrealista y una ubicación espacial de cada objeto.

1.1. Estado del arte

Existen diferentes laboratorios y universidades que desarrollan multitud de proyectos relacionados con entornos inteligentes. Hallándose, por todo el mundo laboratorios y espacios dedicados exclusivamente a esta labor, por lo que a continuación se va a realizar una introducción de los trabajos más importantes en este campo.

1.1.1. Espacios inteligentes

El inicio de los espacios inteligentes y la "Computación Ubicua" fueron conducidos por Weiser [1] [2]. Este autor realiza una clasificación en función de cuatro elementos básicos: ubicuidad, conocimiento, inteligencia e interacción natural.

- Ubicuidad: Característica de un sistema global formado por otros múltiples sistemas embebidos con total interconexión entre ellos.
- Conocimiento: Habilidad del sistema para localizar y reconocer lo que acontece en el entorno y su comportamiento.
- Inteligencia: Capacidad de adaptación al mundo que percibe.
- Interacción Natural: Capacidad de comunicación entre el entorno y los usuarios.

Uno de los primeros sistemas ubicuos implementados con esta filosofía se propuso en el proyecto Xerox PARC de Computación Ubicua (UbiComp) [3], desarrollado a finales de los 80. La red de sensores desplegada no estaba basada en información visual debido al coste prohibitivo para la época. Hoy en día varios grupos de investigación desarrollan y expanden el concepto de espacio inteligente y la computación ubicua. Algunos de los más notorios se indican a continuación:

- Intelligent Room [4] Desarrollado por el grupo de investigación del Artificial Intelligence Laboratory en el MIT (Massachusetts Institute of Technology). Es uno de los proyectos más evolucionados en la actualidad. Se trata de una habitación que posee cámaras, micrófonos y otros sensores que realizan una actividad de interpretación con el objetivo de averiguar las intenciones de los usuarios. Se realiza interacción mediante voz, gestos y contexto.

- Smart Room [5] Realizado en el Media Laboratory del MIT, utilizando cámaras, micrófonos y sensores se pretende analizar el comportamiento humano dentro del entorno. Se localiza al usuario, identificándolo mediante su voz y apariencia y se efectúa un reconocimiento de gestos. La línea de investigación actual es la capacidad del entorno para aprender a través del usuario que a modo de profesor le enseña nombres y apariencia de diferentes objetos.

- Easy Living [6] Se trata de un proyecto de investigación de la empresa Microsoft con el objetivo de desarrollar "entornos activos" que ayuden a los usuarios en tareas cotidianas. Al igual que otros proyectos comentados se intenta establecer una comunicación entre el entorno y el usuario mediante lenguaje natural. Se hace uso de visión artificial para realizar identificación de personas e interpretación de comportamientos dentro del espacio inteligente.

Muchos de los proyectos mencionados no incluyen robots controlados por el entorno. El uso de agentes controlados por el espacio inteligente es estudiado por un grupo todavía reducido de laboratorios.

- T.Sogo [7] propone un sistema equipado con 16 cámaras fijas llamadas "Agentes de Visión" (VA), las cuales deberán aportar la información necesaria para localizar a un grupo de robots a través de un espacio lleno de obstáculos. Los Agentes de Visión consisten en cámaras omnidireccionales no calibradas. La localización se realiza en dos pasos. En primer lugar, un operador humano debe mostrar al espacio inteligente el camino que deberán seguir los robots. Una vez que el sistema haya aprendido en coordenadas de imagen el camino señalado, puede actuar sobre los robots, con el objetivo de minimizar el camino que recorren en la imagen con respecto al que realizan durante el periodo supervisado.

  La técnica es similar a la utilizada en aplicaciones de servo óptico, por lo que resulta muy difícil estimar la precisión que se obtiene realmente con el sistema. Dependerá en gran medida de la técnica de comparación en el plano imagen y la posición relativa de las cámaras y los robots.

- Un trabajo más evolucionado fue propuesto en la universidad de Tokyo por Lee y Hashimoto [8] [9]. En este caso se dispone de un Espacio Inteligente completo en el que se mueven robots y usuarios. Cada dispositivo de visión es tratado como un dispositivo inteligente distribuido y conectado a una red (DIND). Tiene capacidad de procesamiento por sí solo, por lo que el espacio inteligente posee flexibilidad a la hora de incluir nuevos DIND. Puesto que se cuenta con dispositivos calibrados, la localización se realiza en espacio métrico. Los robots están dotados de balizamiento pasivo mediante un conjunto de bandas de colores fácilmente detectables por cada DIND. En este espacio de trabajo (Ispace), se realizan experimentos de interacción entre humanos y robots y navegación con detección de obstáculos.

- Otra propuesta es el proyecto MEPHISTO [10] del Instituto para el Diseño de Computadores y Tolerancia a Fallos en Alemania. En este proyecto el concepto de inteligencia distribuida se alcanza gracias a las denominadas Unidades de Procesamiento de Área Local (LAPU), similares a los DIND de Lee y Hashimoto. Se define una unidad específica para el control del robot (RCU) que deberá conectar y enviar instrucciones a los diferentes robots. La localización se realiza sin marcas artificiales, usando una descripción poligonal en coordenadas de imagen, la cual es convertida a una posición tridimensional con un enfoque multicámara.

- En el Departamento de Electrónica de la Universidad de Alcalá se desarrolla un proyecto de "espacio inteligente" [11] [12] [13] en el que mediante un conjunto de sensores se pretende realizar posicionamiento de robots móviles. Las alternativas incluyen visión artificial, ultrasonidos e infrarrojos. En dicho proyecto se enmarca el presente trabajo fin de carrera que intenta seguir las líneas de investigación iniciadas, utilizando cámaras como sistema sensorial del espacio.

Los proyectos anteriormente comentados hacen hincapié en el diseño de sistemas distribuidos y flexibles, donde la inclusión de un nuevo sensor al entorno se realiza de forma dinámica, sin afectar al funcionamiento del sistema. Se han desarrollado trabajos, que si bien se diferencian en el planteamiento genérico, las técnicas usadas y el objetivo se relacionan con el trabajo propuesto.

- Destaca entre otros el trabajo realizado por Hoover y Olsen [14], en el que utilizando un conjunto de cámaras con coberturas solapadas, son capaces de detectar obstáculos estáticos y dinámicos. Mediante un mapa de ocupación [15] es posible hacer navegar a un robot dentro del espacio de cobertura conjunta.

  La técnica del mapa de ocupación requiere una fase previa de calibración [16] en la que el espacio se encuentra libre de obstáculos. Se obtienen imágenes de referencia que serán utilizadas para la localización de obstáculos en posteriores capturas.

  Mediante tablas de búsqueda se logra una correspondencia directa entre los puntos de pantalla y las coordenadas tridimensionales de puntos pertenecientes al plano que representa el suelo del espacio. Y así si se realiza un fundido de la información de las distintas cámaras, se consigue un mapa de aquellos puntos del suelo que tienen alta probabilidad de estar ocupados.

  Uno de los principales inconvenientes de este enfoque, es la detección de objetos con bajo contraste con respecto al color del suelo en la imagen de referencia. El sistema desperdicia mucha información tridimensional mediante el mapa de ocupación, que podrá ser utilizada para obtener información tridimensional del entorno y de los objetos que se mueven en él.

- Un proyecto similar al anterior es el presentado por Kruse y Whal [17] denominado MONAMOVE (Monitoring and Navigation for Mobile Vehicles), orientado a la navegación de un vehículo en un entorno industrial. En este trabajo se propone fundir la información de localización obtenida del conjunto de cámaras distribuidas por el entorno, con sensores de ultrasonidos e infrarrojos equipados a bordo del robot. Al igual que el trabajo de Hoover y Olsen se utilizan tablas de búsqueda para crear un mapa de ocupación del entorno.

  Los obstáculos móviles se detectan mediante análisis diferencial en sucesivas imágenes y los estáticos son comparados con dos imágenes de referencia, conseguidas en un proceso previo de calibración. Dichas imágenes de referencia se toman bajo dos condiciones de iluminación diferentes, y se realiza una comparación ponderada con la imagen actual, para obtener regiones correspondientes a objetos que existen en el entorno.

1.1.2. Técnicas de reconstrucción volumétrica a partir de múltiples cámaras

La reconstrucción tridimensional o volumétrica consiste en hallar el volumen que ocupan los objetos en un espacio determinado.

En el campo de la reconstrucción tridimensional a partir de visión artificial en entornos inteligentes, existen varios trabajos interesantes en los que se han utilizado diversas técnicas para optimizar, tanto la plataforma hardware en lo referente a la colocación de los elementos

sensoriales del sistema, como la software incluyendo algoritmos o nuevas técnicas para optimizar los recursos.

La reconstrucción tridimensional mediante múltiples cámaras es un campo científico de gran interés y ha recibido especial atención en la literatura. En concreto, en este proyecto se desea obtener una reconstrucción volumétrica del entorno, donde cada objeto es definido por el volumen 3D que ocupan. Existen diversos trabajos en esta línea bajo el nombre de "SFS".

El término "Shape-From-Silhouette"(SFS) es un método de reconstrucción del volumen tridimensional de un objeto en 3D a partir de los contornos de varias imágenes del mismo. La idea de usar siluetas para una reconstrucción en 3D fue introducida por Baumgart en 1974 [18]. Baumgart estimó los contornos 3D de un muñeco y de un caballo de juguete a partir de 4 imágenes de su silueta. Partiendo de él, han sido propuestas diferentes variaciones del paradigma SFS.

Por ejemplo, Aggarwal [19] [20] usó descripciones volumétricas para representar los objetos reconstruidos. Potmesil [21], Noborio [22] y Ahuja [23] sugirieron usar una estructura de datos octree para acelerar el SFS. Shanmukh y Pujari consiguieron la posición y dirección óptimas de las cámaras para obtener siluetas de un objeto para una reconstrucción del volumen [24]. Szeliski construyó un digitalizador no invasivo usando una mesa giratoria y una sola cámara con el SFS como procedimiento de reconstrucción [25]. En resumen, el SFS se ha convertido en uno de los métodos más populares para la reconstrucción de objetos estáticos.

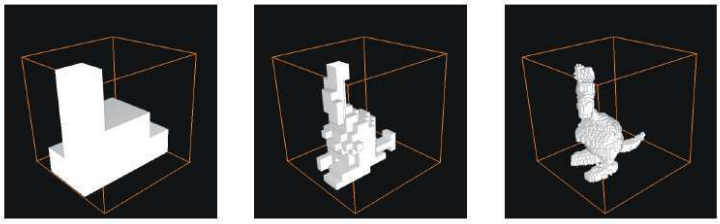


Figura 1.2: Refinado octree

Una de las formas de hallar la reconstrucción tridimensional, a partir de siluetas, es a través del cálculo del "Visual Hull" (V.H.). Este término es usado por los investigadores para denotar el mayor volumen reconstruido usando el principio SFS: El término fue acuñado en 1991 por Laurentini [26] quien también publicó una serie de artículos estudiando los aspectos teóricos del Visual Hull aplicado a objetos poliédricos [27] [28] y curvos [29].

Calcular el volumen utilizando SFS tiene muchas ventajas. La primera es, que las siluetas se consiguen con facilidad, especialmente en entornos cerrados (espacios inteligentes) donde las cámaras son estáticas y el entorno está controlado (iluminación, brillos, sombras...). La aplicación de la mayoría de los métodos SFS, son también relativamente sencillos, en especial si los comparamos con otros métodos de reconstrucción como el multi-baseline stereo [30] o el space carving [31]. Además, la inherente propiedad conservativa del volumen reconstruido usando SFS es particularmente útil en aplicaciones como evitar obstáculos en la manipulación de robots y en el análisis de la visibilidad en navegación. Estas ventajas han provocado que un gran número de investigadores apliquen SFS para solucionar problemas en el ámbito de los espacios inteligentes: digitalización humana virtual [32], estimación de la forma humana [33], tracking/captura de movimiento [34] [35] y renderizado basado en imágenes [36].

Por otro lado, el SFS adolece de la limitación de que el volumen estimado (el Visual Hull) puede ser muy burdo cuando tenemos pocas siluetas,siendo apreciable para objetos de cierta complejidad. Para hacer una estimación mejor de este tipo de escenas hay que incrementar el número de distintas siluetas del objeto. La forma más común de conseguirlo es incrementando

físicamente el número de cámaras utilizadas. Esta opción, muy simple, quizás no puede ser siempre factible debido a situaciones prácticas específicas o limitaciones físicas. En otra línea también hay trabajos [37] para hacer una aproximación "a través del tiempo", incrementando el número de siluetas efectivas, haciendo distintas capturas en distintos instantes de tiempo (cuando el objeto se está moviendo).

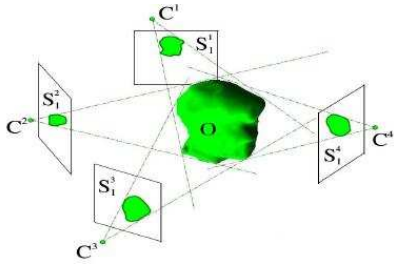


Figura 1.3: Cálculo del Visual Hull, a partir de siluetas

En la mayoría de las soluciones que calculan el Visual Hull, se discretiza el espacio para obtener una ocupación tridimensional que sea fácil de computarizar. El espacio se divide en unidades cúbicas elementales llamadas vóxeles. El método SFS utilizado en el proyecto proporciona como resultado un conjunto de vóxeles con un cierto nivel de ocupación.

Con el objetivo de representar el volumen reconstruido, es muy interesante utilizar una malla a partir del grid de ocupación de los vóxeles. Ésta consiste en una superficie conexa, formada por triángulos, que envuelve a la ocupación tridimensional. De esta manera los triángulos de la malla corresponden con la corteza del Visual Hull y, además, resulta muy sencillo obtener la orientación de dichos triángulos para texturizarlos con la imagen proveniente de la cámara adecuada.

Existen diferentes métodos para mallar un volumen. En este proyecto se ha desarrollado el algoritmo de Marching Cubes, por su sencillez y su baja complejidad computacional.

El método de Marching Cubes propuesto por Lorensen y Cline en 1987 [38] es una poderosa técnica para visualizar una estructura tridimensional formada por vóxeles. Lorensen y Cline presentaron un algoritmo, que consistía en crear modelos formados por triángulos que constituyen superficies equiescalares a partir de datos médicos tridimensionales.

Definieron una tabla de casos posibles para cada disposición de los elementos cúbicos de la estructura donde indicaron la topología de los triángulos adecuada para cada caso, con el objetivo de generar los tramos de conexión de las superficies equiescalares. En la figura 1.4 se representa un cubo, cuyos vértices son los datos tridimensionales discretos, que sirve de unidad para definir los diferentes casos. El algoritmo fue creado para procesar datos médicos tridimensionales en línea para la exploración. Calculaba los vértices de los triángulos mediante interpolación lineal en función del nivel de intensidad . El algoritmo también calculaba el gradiente de los datos originales, y lo normalizaba, para utilizarlo como base para un modelo de sombras. Mostraron resultados con datos proporcionados por una tomografía, una resonancia magnética y una tomografía (SPECT) computarizada de emisión de fotones para ilustrar la calidad y la funcionalidad de Marching Cubes.

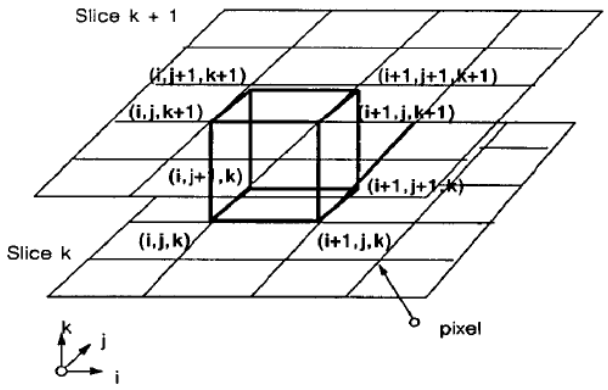


Figura 1.4: Marching Cube

En 1999 surgió un método de MarchingCubes modificado [39] para resolver algunos problemas de ambigüedad que existían en el algoritmo de Marching Cubes original. Para realizarlo examinaron para cada cubo la conectividad de la configuración poliédrica en el sentido de la topología combinatorial. Para los cubos donde las conectividades no son consideradas, modificaron las configuraciones poliédricas para obtener la conectividad y construir superficies equiescalares con la topología correcta.

1.2. Objetivos planteados y solución propuesta

A continuación se van a exponer los objetivos principales del proyecto:

- **Cálculo de las siluetas de los objetos** A partir de las imágenes de las cámaras de vídeo es necesario distinguir las siluetas de los objetos del fondo de la escena.
- **Ubicación espacial de diferentes objetos** A partir de las siluetas de diferentes cámaras se desea fundir la información para obtener la ocupación tridimensional de los objetos.
- **Representación fotorrealista** Se desea representar la reconstrucción tridimensional de los objetos coloreados, a partir de la información obtenida de la ocupación tridimensional y las imágenes a color procedentes de las N cámaras.
- **Discriminación de objetos 3-D** Se requiere obtener el número de objetos de la escena y el "Bounding Box" de cada uno.
- **Restricción de tiempo real** Uno de los principales objetivos de este proyecto es que sea capaz de ejecutarse en tiempo real.

En la figura 1.5 se representa un diagrama con las entradas necesarias para llegar a alcanzar los objetivos y en la figura 1.6 se visualiza un ejemplo con imágenes reales.

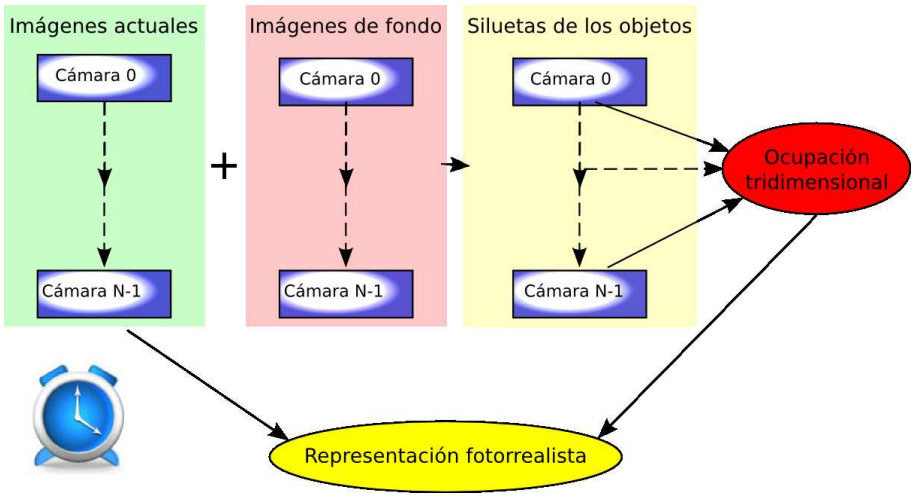


Figura 1.5: Objetivos principales

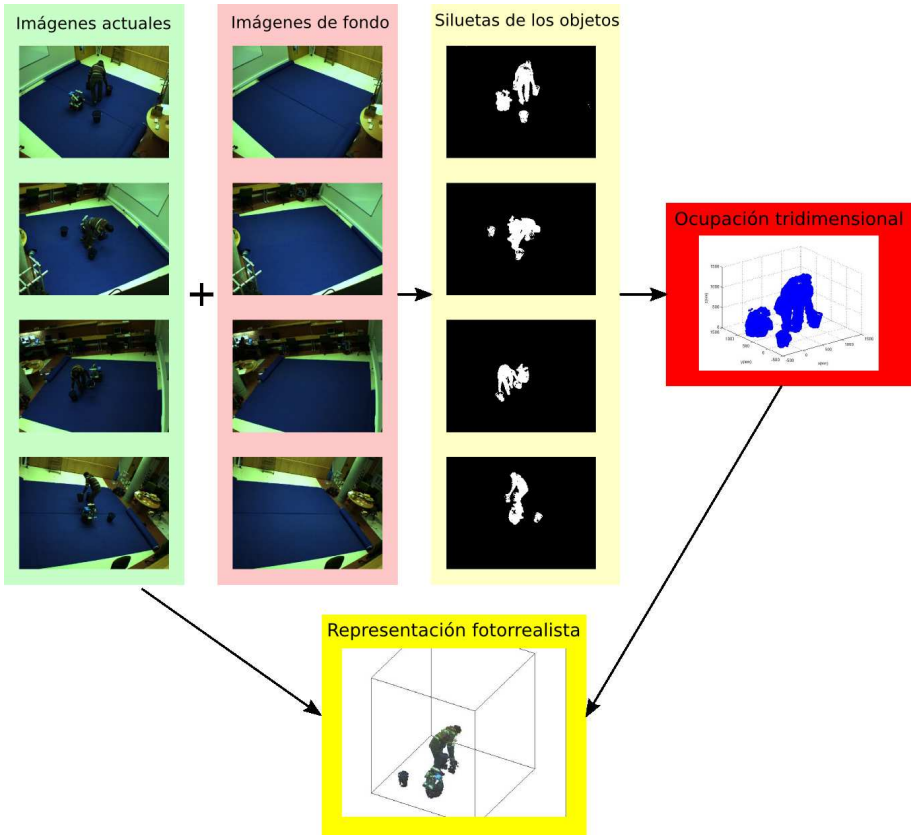


Figura 1.6: Ejemplo de reconstrucción

1.2.1. Solución propuesta

Para alcanzar los objetivos planteados anteriormente se han propuesto las siguientes soluciones:

- **Segmentación de imágenes**

  Para calcular las siluetas de las imágenes se va a proceder a segmentar las imágenes de cada frame a partir de las imágenes de fondo de la escena de cada cámara, para obtener imágenes en blanco y negro, donde los píxeles blancos correspondan a objetos y los píxeles negros correspondan al fondo de la escena.

- **Ocupación tridimensional**

  Para calcular la ocupación tridimensional el espacio se va a dividir en vóxeles 1.7, que es la unidad cúbica del espacio tridimensional, equivalente a píxeles pero en tres dimensiones en vez de en dos.

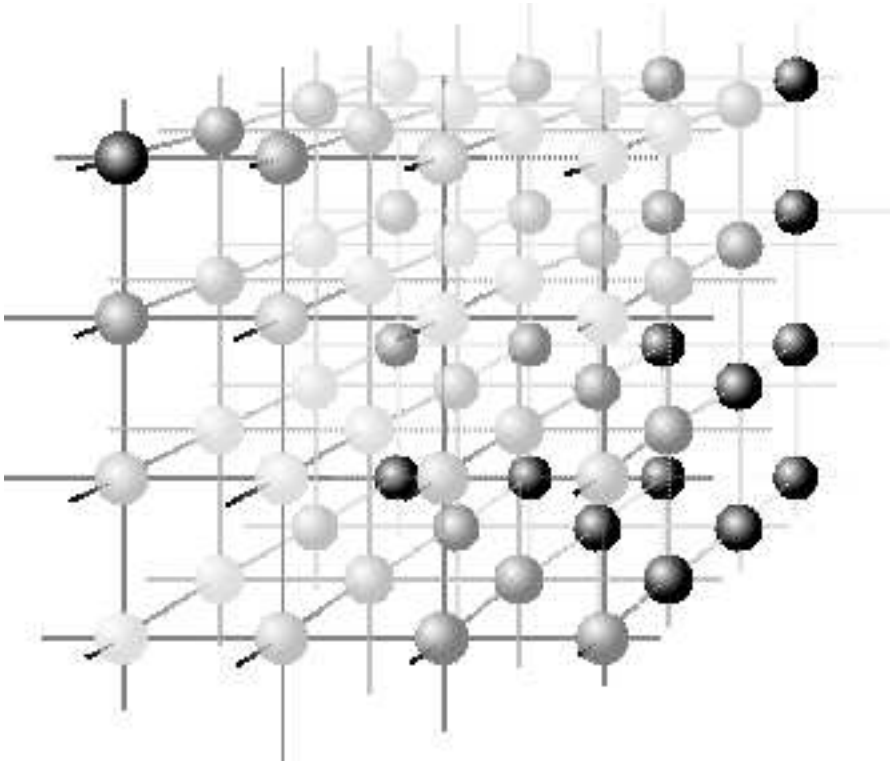


Figura 1.7: Entramado de vóxeles tridimensionales

Dependiendo del tamaño de los vóxeles, la reconstrucción va a ser de mayor o menor calidad. La aproximación de la ocupación real que se va a calcular es el "Visual Hull" que corresponde con la intersección de los conos visuales de las siluetas de los objetos vistas desde N cámaras.

Para calcular los vóxeles que corresponden con el "Visual Hull" se va a dividir el espacio en planos paralelos al suelo. En cada plano se va a calcular la intersección de las N imágenes segmentadas y proyectadas en dicho plano mediante una homografía. Esta solución se encuentra implementada en el proyecto de Javier Rodríguez [40].

- **Mallado de la corteza de los objetos que formen parte de la ocupación tridimensional**

  Para realizar la representación fotorrealista se va a calcular una malla y posteriormente se va a texturizar. El mallado que se realiza va a consistir en calcular una superficie que envuelva a los objetos, formada de triángulos. A partir de los datos de ocupación se va a aplicar el algoritmo de Marching Cubes.

- **Texturizado de la malla**

  El texturizado de los triángulos que formen la malla se realiza a partir de las imágenes en color de las N cámaras. Para elegir la cámara que mejor visualiza al triángulo se va a tener en cuenta la distancia a las cámaras y la orientación de los triángulos.

- **Conectividad tridimensional**

  El cálculo de la conectividad se lleva a cabo para discriminar los objetos que existen entre la ocupación hallada. Se han implementado dos algoritmos, uno para hallar la conectividad de los triángulos y otro para determinar la conectividad de los vóxeles.

1.2.2. Diagrama general

En la figura 1.8 se representa un diagrama esquematizado de los pasos llevados a cabo para realizar la reconstrucción en tres dimensiones.

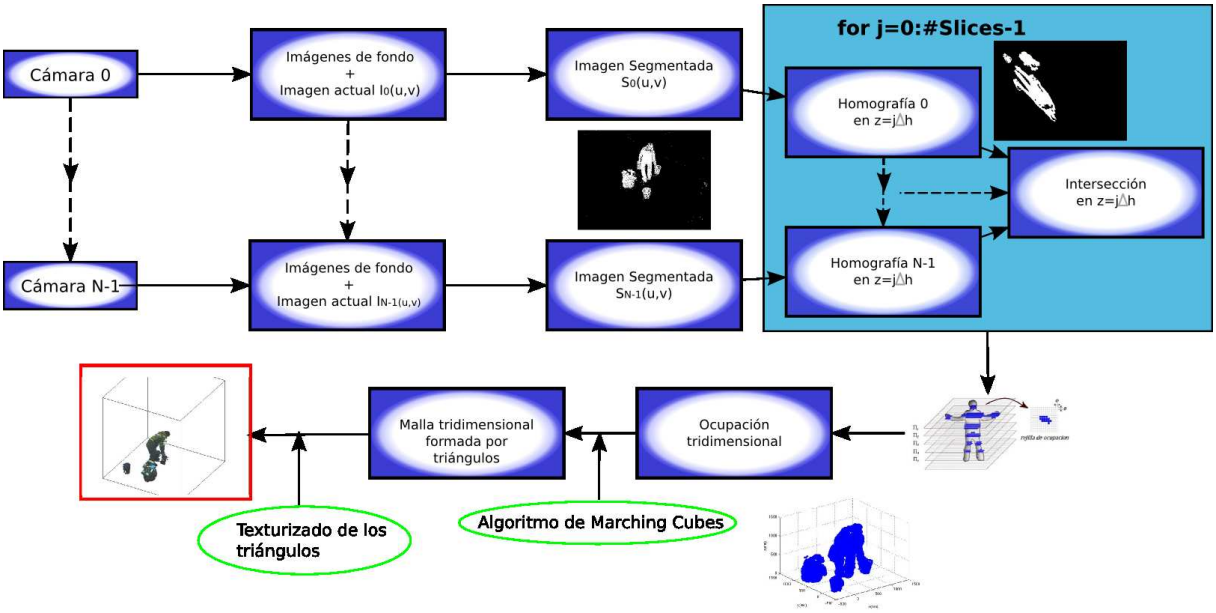


Figura 1.8: Diagrama general de la reconstrucción tridimensional

1.2.3. Estructura del documento

- **Conceptos teóricos de la geometría proyectiva y el espacio tridimensional**

  En esta sección se van a explicar los fundamentos teóricas de la geometría de la formación de la imagen y la relación existente entre la imagen bidimensional y la escena en el espacio. También se va a desarrollar el concepto de Visual Hull y el algoritmo de Marching Cubes para construir una malla de triángulos envolviéndolo.

- **Sistemas de segmentación de fondo en cámaras estáticas**

  En esta sección se van a desarrollar de manera teórica algunos métodos para realizar la segmentación de imágenes para distinguir el fondo de los objetos existentes en el espacio.

- **Sistemas de reconstrucción volumétrica a partir de múltiples cámaras**

  En este apartado se van a desarrollar los algoritmos realizados en el proyecto para hallar el citado Visual Hull y la malla tridimensional y su texturizado.

- **Detección de personas y objetos a partir de la reconstrucción volumétrica** En esta parte del documento se va a explicar el algoritmo de Viola & Jones para detección de caras y su posible integración en la detección de personas, a partir de la reconstrucción volumétrica.

- **Algoritmos de conectividad**

  En esta sección se desarrollan dos algoritmos de conectividad tridimensional, uno de vóxeles y otro de los triángulos que forman una malla.

- **Implementación hardware del sistema**

  En esta sección del documento se describen las características técnicas de los elementos que forman el sistema y su disposición en el espacio inteligente.

- **Implementación software del sistema**

  En esta parte del documento se detallan las herramientas software que se han utilizado para realizar el proyecto y la distribución computacional de las diferentes tareas que se han de ejecutar.

- **Resultados experimentales**

  En este apartado se van a exponer diferentes resultados variando algunos parámetros importantes y de alta sensibilidad en el sistema como son el número de cámaras con el que se realiza la reconstrucción y la diferentes resoluciones usadas en el tamaño de los vóxeles. También se van a detallar análisis temporales de las tareas en función de la resolución y el número de vóxeles ocupados en la escena. Los resultados experimentales se realizarán a partir de reconstrucciones halladas, a partir de imágenes almacenadas en bases de datos, así como en experimentos realizados en tiempo real en el espacio inteligente.

- **Conclusiones y trabajos futuros**

  Como parte final de la memoria se aportarán las conclusiones extraídas de la realización del proyecto, posibles mejoras de los algoritmos descritos y trabajos futuros interesantes de integrar dentro del presente proyecto.

Capítulo 2

Conceptos teóricos de la geometría proyectiva y el espacio tridimensional

La geométrica proyectiva estudia la transformación de los rayos de luz que emiten los puntos que forman una escena tridimensional a una imagen digital bidimensional $I(u, v)$, donde (u, v) son las coordenadas de los píxeles de la imagen e I un dato de intensidad de luz. Si se trata de una imagen en escala de grises tan solo tendrá una componente ,o si por el contrario, es una imagen en color, tendrá tres componentes de luz: roja, verde y azul.

Se va a suponer que cada punto de la escena se comporta como una fuente de luz que radia hacia todas las direcciones.

2.1. Geometría de formación de la imagen

La geometría de la formación de la imagen estudia la trayectoria de los rayos de luz que inciden sobre el plano imagen de una cámara para hallar la correspondencia entre los puntos tridimensionales del espacio y las coordenadas bidimensionales de dicho plano. Existen dos aproximaciones matemáticas utilizadas, la aproximación de las lentes delgadas y el modelo de cámara "pin-hole". Este último modelo es el elegido en el proyecto para realizar la correspondencia tridimensional dada su sencillez.

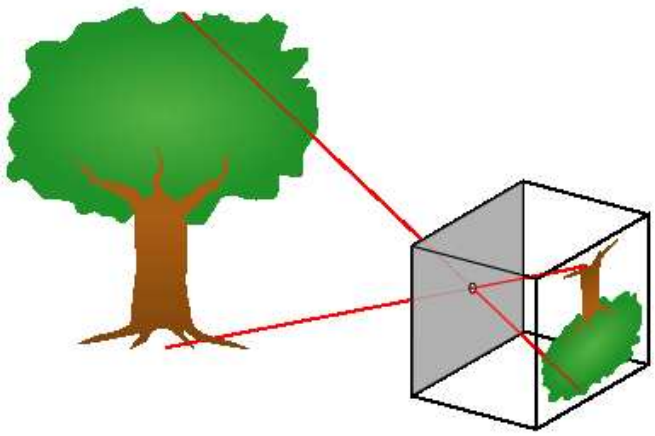


Figura 2.1: Ejemplo de cámara "pin hole"

2.1.1. Modelo de cámara

Las cámaras se encuentran calibradas suponiendo un modelo de cámara "pin hole". En este modelo, se considera que todos los rayos ópticos pasan por el centro óptico, un agujero infinitesimalmente pequeño por el que pasa la luz. La distancia a la que está colocado el plano imagen corresponde con la distancia focal f.

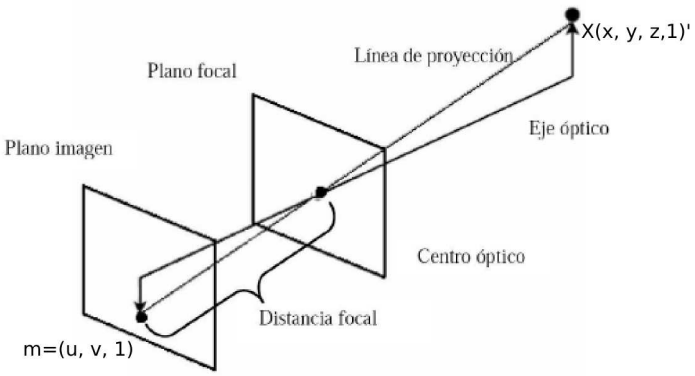


Figura 2.2: Modelo de cámara pin-hole

Los puntos bidimensionales de la imagen en coordenadas homogéneas se anotarán como:

$$m = (u, v, 1)^T \tag{2.1}$$

$$\tilde{m} = (U, V, \lambda)^T \tag{2.2}$$

Las coordenadas (U, V, λ) para cualquier valor de $\lambda \in \Re$ corresponden a los puntos tridimensionales del rayo óptico en coordenadas referenciadas a la cámara. Las coordenadas de la cámara se anotarán como (x_c, y_c, z_c) cuyo origen de coordenadas se encuentra en el centro del plano imagen.

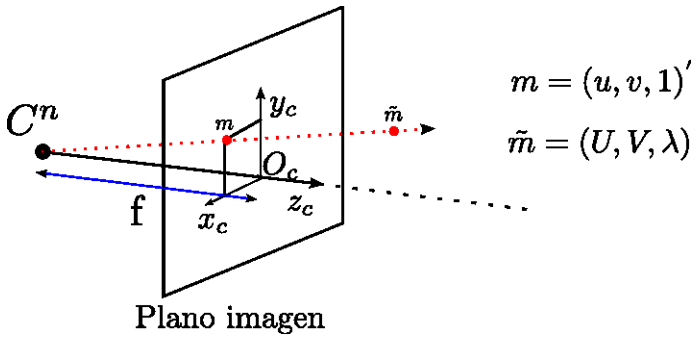


Figura 2.3: Coordenadas referidas al centro de la cámara

Para obtener las coordenadas en la imagen de la cámara es necesario normalizar con la coordenada z de la cámara, por lo tanto, el punto m y $\tilde{m}$ se relacionan de la siguiente forma:

$$m = \frac{1}{\lambda}\tilde{m} = \left(\frac{U}{\lambda}, \frac{V}{\lambda}, 1\right)^T \tag{2.3}$$

Los puntos tridimensionales referenciados a un origen de coordenadas del mundo común para todas las cámaras se expresarán como:

$$X = (x, y, z, 1)^T \tag{2.4}$$

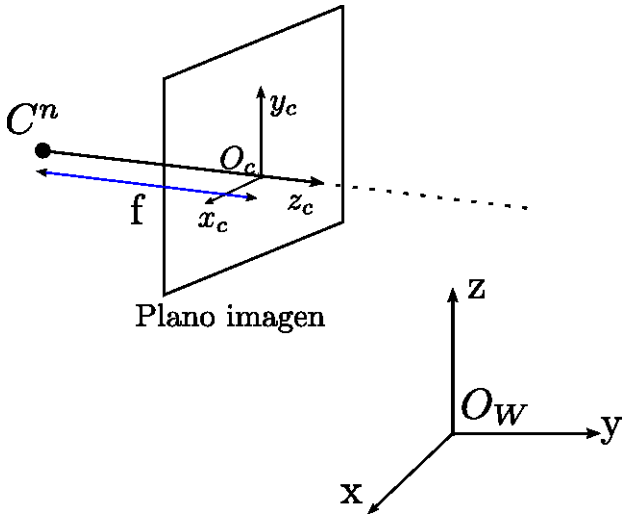


Figura 2.4: Sistema de coordenadas en el modelo de la cámara

Las cámaras están definidas por una matriz de parámetros intrínsecos K y por dos matrices de parámetros extrínsecos, una de rotación R y otra de traslación T. Este modelo se usa para obtener las proyecciones de los puntos 3D sobre el plano de la imagen 2D.

- **Parámetros intrínsecos**

  Los parámetros intrínsecos dependen de las características de la propia cámara y no de factores externos. La matriz K se definirá por la distancia focal, la distorsión, el número de píxeles y su tamaño.

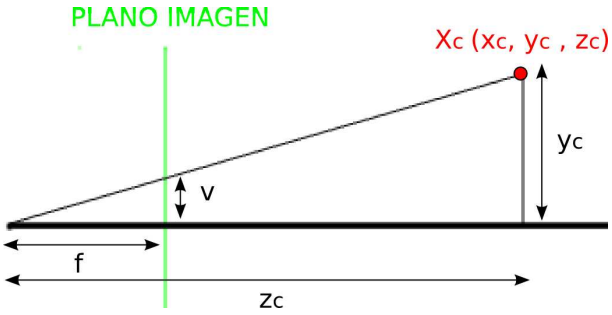


Figura 2.5: Semejanza de triángulos

En la figura 2.5 se puede observar que se forman dos triángulos semejantes entre las coordenadas tridimensionales con respecto del origen de coordenadas de la cámara (x_c, y_c, z_c) y las coordenadas bidimensionales(en distancia) (u_d, v_d) habiéndose fijado el origen de coordenadas de la cámara en el centro óptico. Por semejanza de triángulos se cumple que:

$$\frac{v_d}{f} = \frac{y_c}{z_c} \rightarrow v_d = f\frac{y_c}{z_c} \tag{2.5}$$

De manera análoga las coordenadas u_d también cumplen la siguiente ecuación:

$$\frac{u_d}{f} = \frac{x_c}{z_c} \rightarrow u_d = f\frac{y_c}{z_c} \tag{2.6}$$

Si se tiene en cuenta la longitud de cada píxel d_u y d_v y se divide cada coordenada por dichas longitudes se obtienen las coordenadas en píxeles en vez de en distancia. También hay que sumarles (u_0, v_0) que es el punto en píxeles central del plano imagen. En las ecuaciones anteriores para las coordenadas (u_d, v_d) este punto se ha considerado el $(0, 0)$:

$$u = \frac{1}{d_u} f\frac{x_c}{z_c} + u_0 \tag{2.7}$$

$$v = \frac{1}{d_v} f\frac{y_c}{z_c} + v_0 \tag{2.8}$$

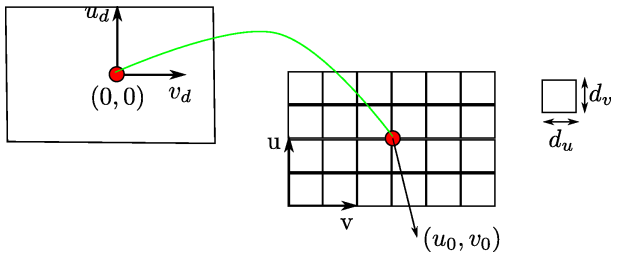


Figura 2.6: Transformación de unidades métricas a unidades en píxeles

En las cámaras existe un parámetro de distorsión S, si se tiene en cuenta las ecuaciones finales resultarían:

$$u = \frac{1}{d_u} f\frac{x_c}{z_c} + u_0 + S\frac{y_c}{z_c} \tag{2.9}$$

$$v = \frac{1}{d_v} f\frac{y_c}{z_c} + v_0 \tag{2.10}$$

Si las ecuaciones 2.9 y 2.10 se escriben de manera matricial se obtiene:

$$\begin{pmatrix} U \\ V \\ \lambda \end{pmatrix} = \begin{pmatrix} \alpha_u & S & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} \tag{2.11}$$

En la ecuación anterior $\alpha_u = \frac{f}{d_u}$ y $\alpha_v = \frac{f}{d_v}$. El factor de escala de la imagen coincide con la coordenada $\lambda = z_c$. El eje z de la cámara coincide con la línea entre el centro óptico y el centro del plano imagen.

Las coordenadas finales son:

$$u = \frac{U}{\lambda} \tag{2.12}$$

$$v = \frac{V}{\lambda} \tag{2.13}$$

Para finalizar la matriz de parámetros extrínsecos es la siguiente:

$$K = \begin{pmatrix} \alpha_u & S & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.14}$$

- **Parámetros extrínsecos**

  Los parámetros extrínsecos dependen del exterior y son relativos a los ejes de coordenadas del mundo común para todas las cámaras con respecto al eje de coordenadas de la propia cámara. La matriz R y la matriz T van a relacionar las coordenadas tridimensionales (x_c, y_c, z_c) con las coordenadas del mundo (x, y, z)

  La matriz R de rotación dependerá de los ángulos de giro (α, β, θ) de los ejes de coordenadas de la cámara respecto de los ejes de coordenadas del mundo.

$$R_\alpha = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & sen(\alpha) \\ 0 & -sen(\alpha) & cos(\alpha) \end{pmatrix} \tag{2.15}$$

$$R_\beta = \begin{pmatrix} cos(\beta) & 0 & -sen(\beta) \\ 0 & 1 & 0 \\ sen(\beta) & 0 & cos(\beta) \end{pmatrix} \tag{2.16}$$

$$R_\theta = \begin{pmatrix} cos(\theta) & sen(\theta) & 0 \\ -sen(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.17}$$

$$R = R_\alpha R_\beta R_\theta \tag{2.18}$$

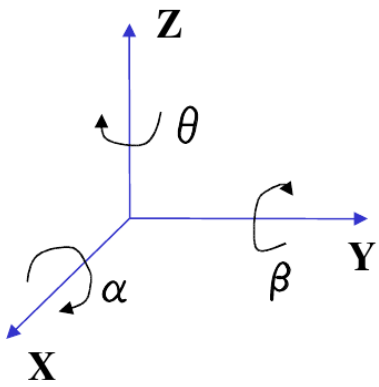


Figura 2.7: Ángulos de euler

La matriz T expresa el lugar donde se encuentra el centro óptico de la cámara (origen de coordenadas de la cámara) con respecto al origen de coordenadas del mundo.

$$T = \begin{pmatrix} T_x \\ T_y \\ T_z \end{pmatrix}$$

- **Matriz de Proyección**

  A partir de las matrices anteriores K, R y T se puede obtener la matriz de proyección P que relaciona un punto espacial $X(x, y, z)$ con su correspondiente proyección (u, v) en la imagen.

$$\begin{pmatrix} U \\ V \\ \lambda \end{pmatrix} = P \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{2.19}$$

  La matriz de proyección está relacionada con las matrices de parámetros extrínsecos e intrínsecos.

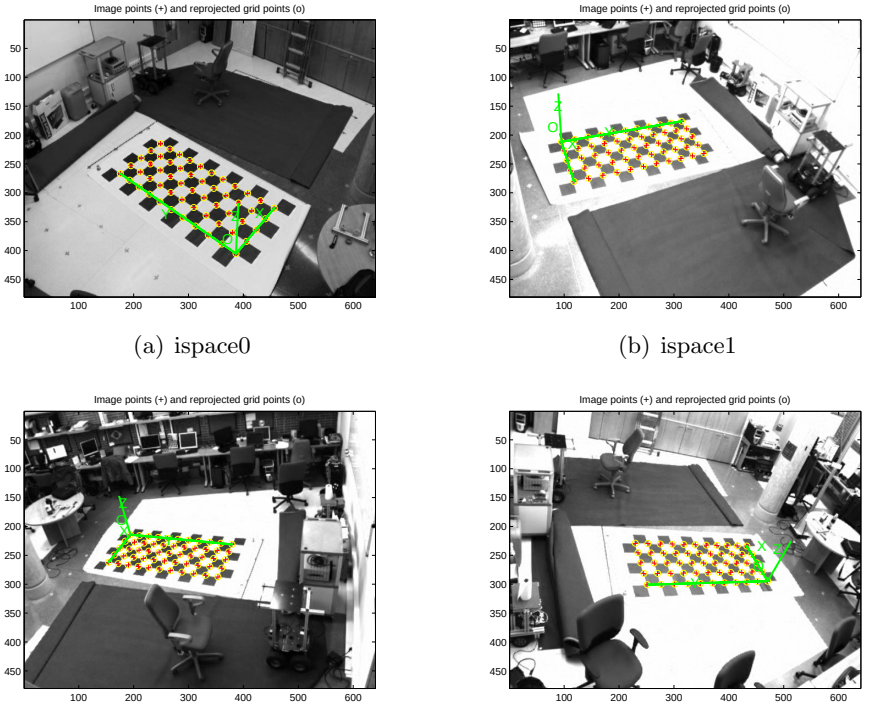
$$P = K(R; T) \tag{2.20}$$

  El punto proyectado en el plano imagen será $u = \frac{U}{\lambda}$ y $v = \frac{V}{\lambda}$. Como el parámetro λ puede tomar cuarquier valor de los números reales, existen infinitos puntos tridimensionales que dan como resultado las mismas coordenadas (u, v) de la imagen, todos los puntos de un mismo rayo óptico. Por esta razón es necesario utilizar varias cámaras para determinar la ocupación tridimensional.

2.1.2. Calibración

Es el proceso de estimar los parámetros extrínsecos e intrínsecos de las cámaras. Para la calibración se ha utilizado una toolbox de Matlab,"Matlab Calibration Toolbox" basada en los artículos [41] y [42].

Para utilizar la herramienta es necesario servirse de un patrón de calibración como el que se observa en la figura 2.8 para obtener imágenes del patrón desde las diferentes cámaras. El patrón está formado por cuadrados negros y blancos con un tamaño de $20cm \times 20cm$. Los cuadros son blancos y negros para detectar las esquinas fácilmente y obtener un entramado de puntos tridimensionales conocidos y su proyección en las cámaras. La herramienta detecta las proyecciones de los puntos 3D en la imagen 2D y resuelve el sistema de ecuaciones correspondiente para hallar los parámetros extrínsecos e intrínsecos. Es necesario indicar el tamaño del cuadrado, y el origen de coordenadas del mundo fijado en una esquina del patrón y los límites del patrón siempre en el mismo orden para todas las cámaras. Esto es así, para fijar un mismo sistema de coordenadas del mundo para todas las cámaras. El número mínimo de puntos necesarios son 12, ya que la matriz de proyección es de 12 elementos.



(a) ispace0 (b) ispace1

(c) ispace2 (d) ispace3

Figura 2.8: Calibración de múltiples cámaras

2.1.2.1. Cálculo de la matriz de proyección

Dado que existen N puntos tridimensionales en coordenadas del mundo conocidos y sus correspondientes proyecciones en la imagen también lo son, se pueden plantear N ecuaciones para cada punto de la imagen $m_i = (u_i, v_i, 1)^T$ junto con el punto tridimensional al que corresponden $X_i = (x_i, y_i, z_i, 1)^T$

$$m_i = P \cdot X_i \tag{2.21}$$

Los elementos de la matriz de proyección P son las incógnitas que se desean obtener:

$$P = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix} \tag{2.22}$$

Las ecuaciones planteadas son:

$$\begin{pmatrix} u_i\lambda_i \\ v_i\lambda_i \\ \lambda_i \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix} \begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} \tag{2.23}$$

Si desarrollamos la ecuación matricial 2.23 se obtiene:

$$\begin{aligned} u_i\lambda_i &= a_{11}x_i + a_{12}y_i + a_{13}z_i + a_{14} \\ v_i\lambda_i &= a_{21}x_i + a_{22}y_i + a_{23}z_i + a_{24} \\ \lambda_i &= a_{31}x_i + a_{32}y_i + a_{33}z_i + a_{34} \end{aligned} \tag{2.24}$$

Sustituyendo el valor de λ_i en las ecuaciones anteriores resulta:

$$\begin{aligned} u_i(a_{31}x_i + a_{32}y_i + a_{33}z_i + a_{34}) &= a_{11}x_i + a_{12}y_i + a_{13}z_i + a_{14} \\ v_i(a_{31}x_i + a_{32}y_i + a_{33}z_i + a_{34}) &= a_{21}x_i + a_{22}y_i + a_{23}z_i + a_{24} \end{aligned} \tag{2.25}$$

$$\begin{aligned} a_{11}x_i + a_{12}y_i + a_{13}z_i + a_{14} - u_i(a_{31}x_i + a_{32}y_i + a_{33}z_i + a_{34}) &= 0 \\ a_{21}x_i + a_{22}y_i + a_{23}z_i + a_{24} - v_i(a_{31}x_i + a_{32}y_i + a_{33}z_i + a_{34}) &= 0 \end{aligned} \tag{2.26}$$

Escribiendo las ecuaciones anteriores de manera matricial para todos los puntos se obtiene la siguiente ecuación:

$$L \cdot a = 0; \tag{2.27}$$

Donde la matriz L es la siguiente:

$$L = \begin{pmatrix} x_1 & y_1 & z_1 & 1 & 0 & 0 & 0 & 0 & -x_1u_1 & -y_1u_1 & -z_1u_1 & -u_1 \\ 0 & 0 & 0 & 0 & x_1 & y_1 & z_1 & 1 & -x_1v_1 & -y_1v_1 & -z_1v_1 & -v_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_i & y_i & z_i & 1 & 0 & 0 & 0 & 0 & -x_iu_i & -y_iu_i & -z_iu_i & -u_i \\ 0 & 0 & 0 & 0 & x_i & y_i & z_i & 1 & -x_iv_i & -y_iv_i & -z_iv_i & -v_i \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_N & y_N & z_N & 1 & 0 & 0 & 0 & 0 & -x_Nu_N & -y_Nu_N & -z_Nu_N & -u_N \\ 0 & 0 & 0 & 0 & x_N & y_N & z_N & 1 & -x_Nv_N & -y_Nv_N & -z_Nv_N & -v_N \end{pmatrix} \tag{2.28}$$

La matriz a es un vector formado por los doce elementos de la matriz de proyección:

$$a = (\ a_{11} \quad a_{12} \quad a_{13} \quad a_{14} \quad a_{21} \quad a_{22} \quad a_{23} \quad a_{24} \quad a_{31} \quad a_{32} \quad a_{33} \quad a_{34}\ )^T \tag{2.29}$$

Existen diferentes métodos para resolver la ecuación 2.27 detallados en la referencia [42].

2.1.2.2. Cálculo de los parámetros intrínsecos

En el año 1999 Zhang propone una técnica flexible para calibrar fácilmente una cámara y obtener los parámetros intrínsecos mediante al menos dos vistas diferentes de un patrón de calibración. También se requiere hallar la relación existente entre los puntos de un plano determinado como el plano del suelo $z = 0$, donde está colocado el patrón y el plano imagen. Esta relación entre planos es también conocida como homografía. El procedimiento consiste en hallar una solución cerrada seguida por una corrección no lineal basada en el criterio de máxima verosimilitud.

Sea la proyección en el plano la siguiente:

$$\tilde{m} = K(R;T)X \longrightarrow \begin{pmatrix} U \\ V \\ \lambda \end{pmatrix} = K(R;T) \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{2.30}$$

La homografía en el plano $z = 0$ se puede expresar como:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \lambda K(r_1 r_2 T) \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2.31}$$

Donde r_1 y r_2 son las columnas 1 y 2 de la matriz de rotación.

Si se llama a la matriz $H = \lambda K(r_1 r_2 T) = (h_1 h_2 h_3)$ donde h_i son las columnas de la matriz H se tiene que:

$$(h_1 h_2 h_3) = \lambda K(r_1 r_2 T) \tag{2.32}$$

Siendo r_1 y r_2 vectores ortonormales se puede llegar a las siguientes expresiones:

$$h_1^T K^{-T} K^{-1} h_2 = 0 \tag{2.33}$$

$$h_1^T K^{-T} K^{-1} h_1 = h_2^T K^{-T} K^{-1} h_2 \tag{2.34}$$

Nota: Para simplificar las expresiones matriciales se va a definir la inversa de la traspuesta como $K^{-T} = (K^{-1})^T$ o $K^{-T} = (K^T)^{-1}$

Sea $B = K^{-T}K^{-1}$ una matriz simétrica se puede escribir como:

$$B = \begin{pmatrix} B_{11} & B_{12} & B_{13} \\ B_{12} & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{pmatrix} \tag{2.35}$$

Si desarrollamos el producto $K^{-T}K^{-1}$ se obtienen las siguientes expresiones:

$$B = \begin{pmatrix} \frac{1}{\alpha_u^2} & -\frac{s}{\alpha_u^2 \alpha_v} & \frac{s \cdot v_0 - u_0 \alpha_v}{\alpha_u^2 \alpha_v} \\ -\frac{s}{\alpha_u^2 \alpha_v} & \frac{s^2}{\alpha_u^2 \alpha_v} + \frac{1}{\alpha_v^2} & \frac{s(s \cdot v_0 - u_0 \alpha_v)}{\alpha_u^2 \alpha_v^2} - \frac{v_0}{\alpha_v^2} \\ \frac{s \cdot v_0 - u_0 \alpha_v}{\alpha_u^2 \alpha_v} & \frac{s(s \cdot v_0 - u_0 \alpha_v)}{\alpha_u^2 \alpha_v^2} - \frac{v_0}{\alpha_v^2} & \frac{(s \cdot v_0 - u_0 \alpha_v)^2}{\alpha_u^2 \alpha_v^2} + \frac{v_0^2}{\alpha_v^2} + 1 \end{pmatrix} \tag{2.36}$$

A partir del cálculo de la matriz B 2.36 se pueden llegar a conocer los parámetros intrínsecos:

Hallando al menos dos matrices de homografía entre los planos imagen de dos vistas diferentes de la misma cámara y el plano del suelo a través de los puntos del patrón se calcula la matriz $B = \lambda K^{-T} K$ y, posteriormente se hallan los parámetros intrínsecos con las siguientes igualdades, siendo λ un valor escalar arbitrario.

$$v_0 = (B_{12}B_{13} - B_{11}B_{23})/(B_{11}B_{22} - B_{12}^2) \tag{2.37}$$

$$\lambda = B_{33} - [B_{13}^2 + v_0(B_{12}B_{13} - B_{11}B_{23})]/B_{11} \tag{2.38}$$

$$\alpha_u = \sqrt{\lambda / B_{11}} \tag{2.39}$$

$$\alpha_v = \sqrt{\lambda B_{11}/(B_{11}B_{22} - B_{12}^2)} \tag{2.40}$$

$$s = -B_{12}\alpha_u^2 \alpha_v / \lambda \tag{2.41}$$

$$u_0 = s \cdot v_0 / \alpha_u - B_{13}\alpha_u^2 / \lambda \tag{2.42}$$

Hay muchas formas para estimar una homografía entre un modelo plano y su imagen . Aquí se presenta una técnica basada en el criterio de máxima verosimilitud. Sean X_i los puntos tridimensionales y m_i los puntos de la imagen. Se asume que m_i está contaminado por ruido gaussiano con media nula y matriz de covarianzas Λ_{m_i}. Para hallar la matriz de homografía H se debe obtener minimizando la siguiente función

$$\sum_i (m_i - \hat{m}_i)^T \Lambda_{m_i}^{-1} (m_i - \hat{m}_i) \tag{2.43}$$

Donde $\hat{m}_i = \frac{1}{\bar{h}_3^T X_i} \begin{bmatrix} \bar{h}_1^T & X_i \\ \bar{h}_2^T & X_i \end{bmatrix}$ con $\bar{h}_i$ es la fila i^{th} de la matriz H

El criterio de máxima verosimilitud consiste en comparar los puntos de las esquinas proyectados en la imagen con las proyecciones calculadas a partir de los puntos 3D correspondientes con los parámetros estimados:

Se debe minimizar la siguiente ecuación:

$$\sum_{i=1}^{N} \sum_{j=1}^{M} \|m_{ij} - \hat{m}_{ij}\|^2 \tag{2.44}$$

Donde $\hat{m}_{ij}$ es:

$$\tilde{m}_{ij} = \hat{K}(\hat{R}_i, \hat{T}_i) X j = (\hat{U}, \hat{V}, \hat{\lambda})^T \tag{2.45}$$

$$\hat{m}_{ij} = \left(\frac{\hat{U}}{\hat{\lambda}}, \frac{\hat{V}}{\hat{\lambda}} \right)^T \tag{2.46}$$

2.1.2.3. Cálculo de los parámetros extrínsecos

Una vez que se tiene calculada la matriz P y K es trivial hallar la matriz R y T:

$$P = K(R;T) \longrightarrow (R;T) = K^{-1}P = P' \tag{2.47}$$

Por lo tanto $R = P'(1:3, 1:3)$ y $T = P'(1:3, 4)$

Existen otros métodos más precisos para hallar la matriz de rotación R definidos en la referencia [43]

2.1.3. Concepto de Homografía

La geometría bidimensional proyectiva estudia las propiedades del plano proyectivo, denominado $\mathcal{P}^2$, que son invariantes bajo un grupo de transformaciones conocidas como transformaciones proyectivas u homografías. El plano proyectivo describe propiedades de puntos expresados en coordenadas homogéneas, como puede ser el punto m_k, resultado de la proyección mediante una cámara "pin-hole". Se dice que una correspondencia $h : \mathcal{P}^2 \longrightarrow \mathcal{P}^2$ es una proyectividad si existe una matriz no singular H, de dimensiones 3×3, tal que, para cualquier punto $m_k \in \mathcal{P}^2$, se cumple que $h(m_k) = Hm_k$ A dicha matriz se la denomina matriz de homografía. Por tanto, una proyectividad u homografía es una correspondencia lineal entre vectores en coordenadas homogéneas de 3 dimensiones:

$$\begin{pmatrix} u^{'} \\ v^{'} \\ 1 \end{pmatrix} = \lambda \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{2.48}$$

Puesto que la matriz de homografía H es no singular, su inversa es a su vez una proyectividad. Si multiplicamos la matriz H por un factor no nulo, la transformación proyectiva no varía porque, al estar trabajando con coordenadas homogéneas, los puntos que difieren en un factor no nulo son equivalentes. Por tanto, si normalizamos la matriz H de forma que $h_{33} = 1$, dicha matriz tendrá 8 grados de libertad.

$$\begin{pmatrix} u^{'} \\ v^{'} \\ 1 \end{pmatrix} = \lambda \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{2.49}$$

En una homografía, en el caso más general, el factor de escala λ depende de las coordenadas (u, v):

$$\lambda(h_{31}u + h_{32}v + 1) = 1 \Longrightarrow \lambda = \frac{1}{h_{31}u + h_{32}v + 1} \tag{2.50}$$

Esta dependencia provoca que no se conserve el paralelismo entre las líneas de un plano a otro. Si $h_{31} = h_{32} = 0$ se trataría de una transformación afín donde sí se conservaría el paralelismo.

Las nuevas coordenadas en el plano de la homografía espacio resultarán:

$$u^{'} = \lambda(h_{11}u + h_{12}v + h_{13}) = \frac{h_{11}u + h_{12}v + h_{13}}{h_{31}u + h_{32}v + 1} \tag{2.51}$$

$$v^{'} = \lambda(h_{21}u + h_{22}v + h_{23}) = \frac{h_{21}u + h_{22}v + h_{23}}{h_{31}u + h_{32}v + 1} \tag{2.52}$$

Mediante una homografía se pueden modelar determinadas relaciones geométricas, entre las que destacan las siguientes:

- **Dos cámaras proyectivas que comparten el origen de proyección:** En este caso se tienen dos cámaras, descritas por sus respectivas matrices P_1 y P_2,en las que el origen de proyección coinciden en ambas, y por lo tanto las matrices de traslación T_1 y T_2 también son coincidentes. En esta configuración existe una matriz de homografía H que relaciona,

de manera biyectiva, la imagen vista por una cámara con la otra. Es decir que, si se considera un punto m_1 visto por la cámara P_1 y otro punto m_2 visto por la cámara P_2, amos correspondientes a la proyección en cada cámara del mismo punto del espacio $X = (x, y, z, 1)'$, entonces se puede encontrar una homografía H tal que cumpla la siguiente expresión:

$$m_1 = \lambda H m_2 \quad \lambda \in \mathcal{R} \tag{2.53}$$

- **Una cámara proyectiva y un plano es en el espacio tridimensional:** Dada una cámara P_1 y un plano en el espacio tridimensional $\Pi : ax + by + cz + d = 0$, existe una homografía H que relaciona puntos definidos en un origen local al plano m_Π(en coordenadas homogéneas), con dichos puntos m_1 vistos en el plano imagen de la cámara.

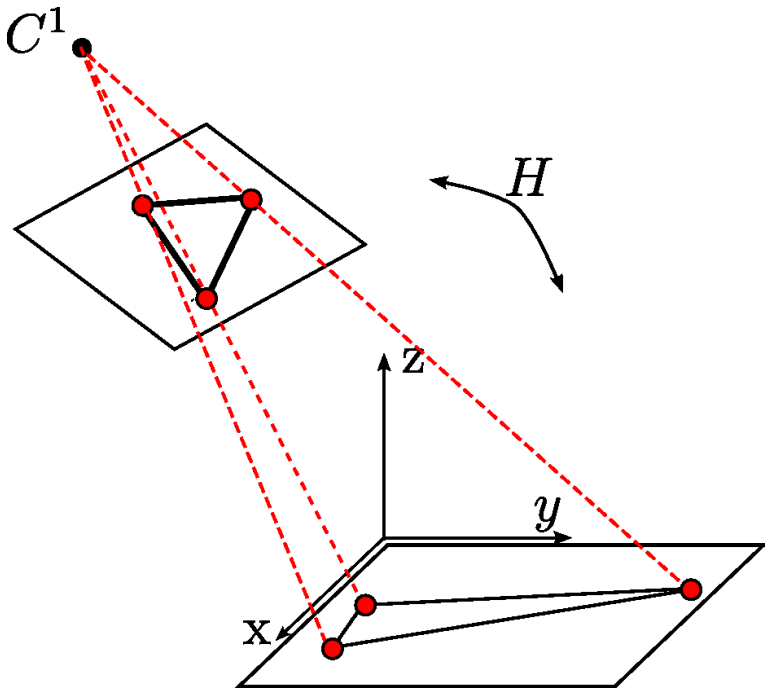


Figura 2.9: Homografía entre una cámara proyectiva y un plano del espacio

Esta relación geométrica es la más importante para el presente trabajo, ya que, para calcular la ocupación tridimensional va a ser necesario realizar homografías entre los planos imagen de cada cámara y planos paralelos al suelo $z = h$, los cuales son divisiones del espacio a reconstruir.

- **Dos cámaras proyectivas y un plano en el espacio tridimensional:** Como extensión del punto anterior, si se tienen dos cámaras P_1 y P_2 y un plano en el espacio tridimensional $\Pi : ax + by + cz + d = 0$, existen dos matrices de homografía H_1 y H_2, que relacionan el plano con cada una de las dos cámaras. Es decir que dados dos puntos m_1 y m_2 vistos por cada cámara y correspondientes al punto m_Π del plano, se cumple que:

$$m_1 = \lambda_1 H_1 m_\Pi \quad \lambda_1 \in \mathcal{R} \tag{2.54}$$

$$m_2 = \lambda_2 H_2 m_\Pi \quad \lambda_2 \in \mathcal{R} \tag{2.55}$$

Puesto que tanto H_1 como H_2 son matrices no singulares, se puede asegurar que existe una homografía H_{12} que relaciona puntos del plano imagen de la cámara P_1 con puntos vistos por la cámara P_2 , resultando H_{12} de la siguiente expresión:

$$m_2 = \lambda_2 H_2 m_\Pi = \lambda_2 \lambda_1^{-1} H_2 H_1^{-1} m_1 = \lambda_{12} H_{12} m_2 \quad \lambda_{12} \in \mathcal{R} \tag{2.56}$$

Siendo la matriz $H_{12} = H_2 H_1^{-1}$

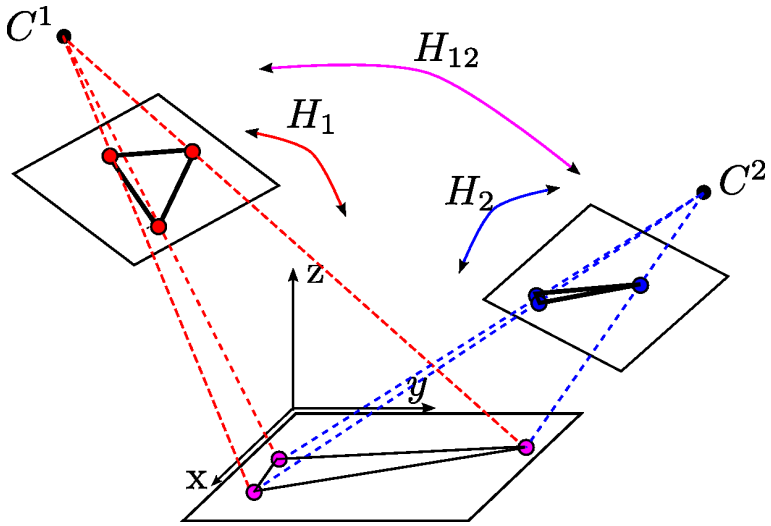


Figura 2.10: Homografía entre dos cámaras proyectivas y un plano del espacio

2.1.3.1. Cálculo de las matrices de homografía

Se va a desarrollar el cálculo de la matriz de homografía para relacionar el plano imagen de una cámara "pin-hole" modelizada por las matrices de parámetros extrínsecos e intrínsecos y un plano en el espacio paralelo al suelo $z = h$.

La correspondencia entre un píxel del plano imagen I con coordenadas (u, v), y el punto X con coordenadas (x, y, h) en el plano Π_h que se caracteriza porque todos los puntos tienen la misma cota $z = h$, se define por la matriz de homografía H:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \lambda \cdot H \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2.57}$$

Los puntos proyectados en el plano imagen I se pueden expresar de la siguiente forma:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \lambda \cdot K(R \cdot X + T) \tag{2.58}$$

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \lambda \cdot K(R \begin{pmatrix} x \\ y \\ z \end{pmatrix} + T) \tag{2.59}$$

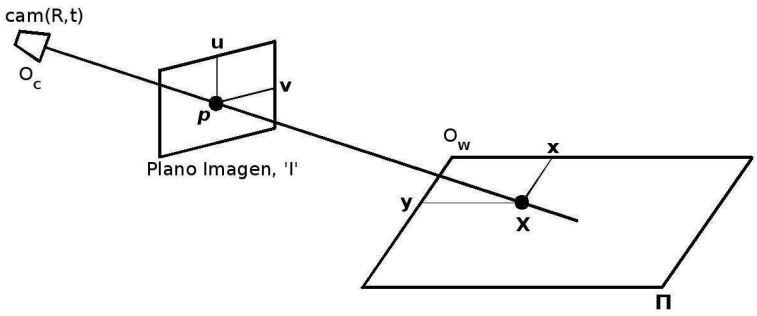


Figura 2.11: Escena cámara-plano Homografía

Particularizando para el plano Π_0 $z = 0$ se obtiene:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \lambda \cdot K(R \begin{pmatrix} x \\ y \\ 0 \end{pmatrix} + T) \tag{2.60}$$

Definiendo $R' = K \cdot R$ y $T' = K \cdot T$ y desarrollando la anterior ecuación se obtiene la siguiente expresión:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \lambda \cdot (R' \begin{pmatrix} x \\ y \\ 0 \end{pmatrix} + T') \tag{2.61}$$

La matriz R' es una matriz 3×3 que se puede escribir como la concatenación de 3 columnas $R' = (c_1 c_2 c_3)$. Operando se obtiene:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \lambda(c_1 \cdot x + c_2 \cdot y + T') \tag{2.62}$$

Escribiendo la ecuación 2.62 de forma matricial resulta:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \lambda(c_1 c_2 T') \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2.63}$$

Concluyendo se obtiene que la matriz de homografía entre un plano imagen y el plano del suelo $z = 0$ es $H = (c_1 c_2 T') = [R'(1:2,:)T']$. Pero en el proyecto es necesario generalizar para cualquier plano Π_h definido con la ecuación $z = h$:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \lambda \cdot K(R \begin{pmatrix} x \\ y \\ h \end{pmatrix} + T) \tag{2.64}$$

Al igual que en la vez anterior se define $R' = K \cdot R$ y $T' = K \cdot T$ y desarrollando la ecuación 2.64 se obtiene la siguiente expresión:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \lambda \cdot (R' \begin{pmatrix} x \\ y \\ h \end{pmatrix} + T') \tag{2.65}$$

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \lambda(c_1 \cdot x + c_2 \cdot y + c_3 \cdot h + T') \tag{2.66}$$

Escribiendo la expresión 2.66 de forma matricial resulta:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \lambda(c_1 : c_2 : c_3 \cdot h + T') \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2.67}$$

La matriz H resultantes es la siguiente:

$$H = (R'(:, 1:2)R'(:, 3) \cdot h + T') \tag{2.68}$$

La función H tiene inversa, por lo que la transformación también se puede expresar en coordenadas del plano Π_h.

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \lambda^{-1} \cdot H^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{2.69}$$

2.2. Concepto de Visual Hull

En el escenario tridimensional se van a disponer de N cámaras que van a dar como resultado un conjunto de siluetas S_i^n para $n = 0, \ldots, N-1$ en el instante t_i. En el figura 4.1 se observa un objeto O desde 4 cámaras diferentes en el instante t_1. Los centros ópticos de las cámaras están anotados en la figura como C^n

En este trabajo aparece un concepto llamado "Visual Hull", del que existen varias definiciones posibles.

- **Definición I de V.H.: Intersección de los conos visuales** El Visual Hull VH_i referenciado a un conjunto de siluetas S_i^n para $n = 0, ..., N-1$ procedentes de N diferentes cámaras se define como la intersección de los N conos visuales formados por el centro de cada cámara C^n y la proyección de la silueta de la imagen S_i^n en el espacio tridimensional.
- **Definición II de V.H.: Volumen máximo** El visual Hull VH_i con respecto a un conjunto de siluetas S_i^n se define como el volumen máximo que encierra S_i^n para $n = 0, 1, \ldots, N-1$. De forma general, para un conjunto de siluetas S_i^n, hay un número infinito de volúmenes (entre los que se encuentra el objeto O) que encierran las siluetas. Con esta segunda definición se fija el V.H como el más grande de estos volúmenes, expresando una de sus cualidades más importantes, el Visual Hull es el volumen convexo máximo que engloba a todas las siluetas, asegurando que toda la geométrica del objeto a representar está incluida dentro del V.H.

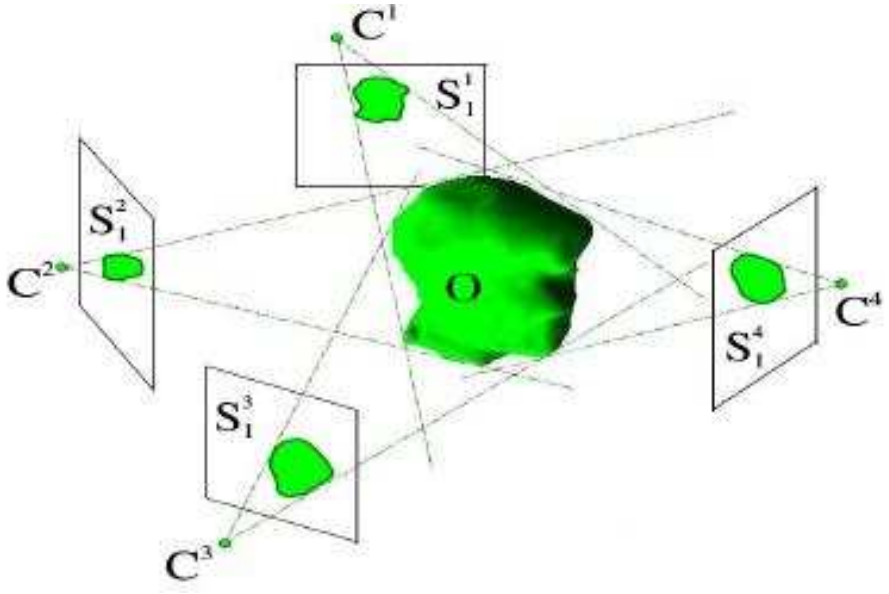


Figura 2.12: Escenario: Objeto O forma la silueta S_1^n en la cámara n en el instante t_1

En la figura 2.13 se representa el Visual Hull de una esfera vista desde tres cámaras. La imagen pertenece a la tesis doctoral de Jean-Sébastian Franco y Edmond Boyer [44].

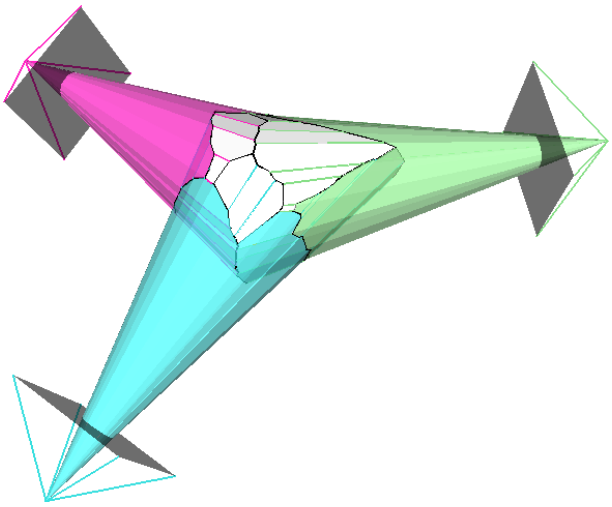


Figura 2.13: El Visual Hull de una esfera

2.2.1. Propiedades

La ocupación tridimensional que se calcula, no es la ocupación tridimensional real, pero se puede asegurar que la contiene. La semejanza entre el Visual Hull y la ocupación tridimensional depende del número de cámaras y de su colocación. Cuanto más cámaras el Visual Hull se ajustará más a la realidad, ya que las intersecciones serán más exigentes. La colocación de las cámaras también es importante, éstas deben de ofrecer diferentes vistas del objeto a detectar.

En la figura 2.14 se puede observar como el objeto detectado es mayor que el objeto real.

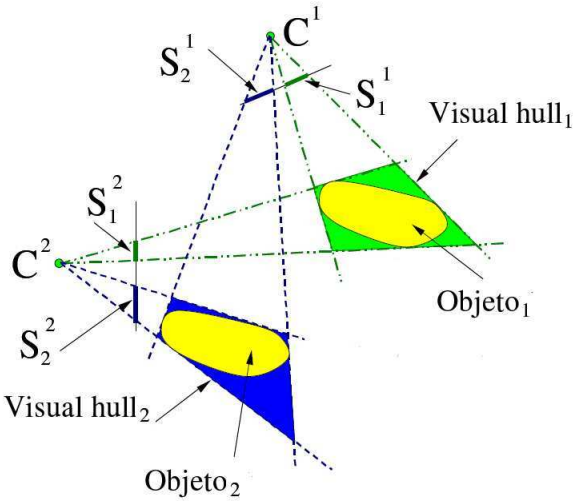


Figura 2.14: Ambigüedad en el Visual Hull en los objetos detectados

En la figura 2.15 se puede analizar que aparecer volúmenes reconstruidos en el Visual Hull que están ocluidos en los conos visuales de las siluetas pero realmente no forman parte de la ocupación real. Cuánto más cámaras se utilicen para realizar la reconstrucción más restrictivo será el Visual Hull.

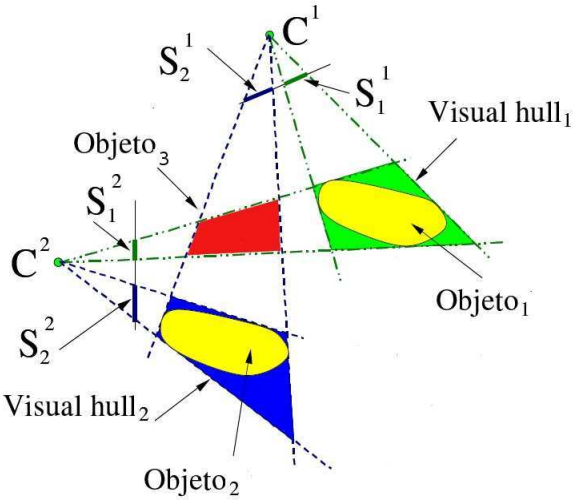


Figura 2.15: Ambigüedad en el Visual Hull, detectando más objetos

En la figura 2.16 se representa un ejemplo de reconstrucción tridimensional de varios objetos con 4 cámaras donde se reconstruye un objeto que realmente no existe:

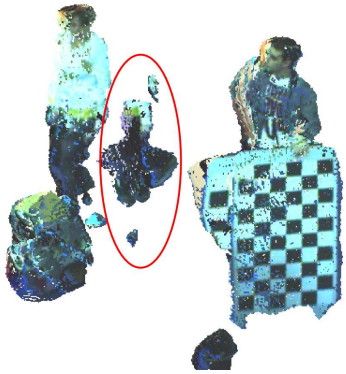


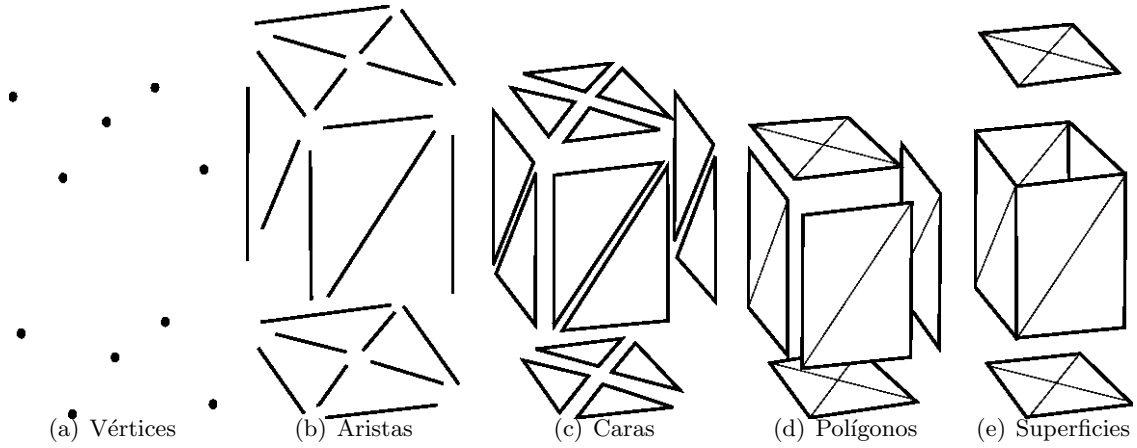
Figura 2.16: Ambigüedad en el Visual Hull, objetos ocluidos

2.3. Mallado tridimensional

Una malla poligonal es un conjunto de vértices, aristas y caras que definen un objeto poliédrico en gráficos 3D y modelos sólidos. Las caras generalmente se tratan de triángulos o cuadriláteros, u otros polígonos convexos (todos los ángulos internos son menores de 180° y todas sus aristas permanecen dentro de los límites del polígono), para simplificar la representación. Aunque también es posible componer la malla con polígonos cóncavos y polígonos con agujeros.

Las operaciones en las mallan pueden incluir operaciones lógicas, filtrado, simplificación, etc.

En la figura 2.17 se representan los elementos que forman una malla, desde los más básicos, hasta los más completos formados por los anteriores:



(a) Vértices (b) Aristas (c) Caras (d) Polígonos (e) Superficies

Figura 2.17: Elementos de una malla tridimensional

El mallado utilizado en el proyecto consiste en construir la superficie del Visual Hull a partir de triángulos, para ello se ha utilizado el algoritmo de Marching Cubes.

Marching Cubes es un algoritmo para representar superficies equiescalares de datos volumétricos. Combina simplicidad y alta velocidad de cómputo ya que puede funcionar con tablas de búsqueda.

Se consideran que los vóxeles del espacio tridimensional son puntos negros o blancos, es decir, ocupados o no ocupados, que están colocados en los vértices de cubos que ocupan todo el espacio 3D.

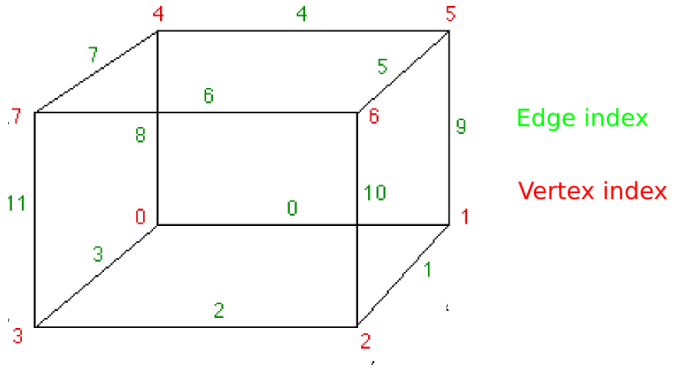


Figura 2.18: Cubo

El problema fundamental es formar una superficie aproximada a través de un campo escalar en una rejilla tridimensional rectangular. Dada una rejilla definida por sus vértices y un valor escalar para cada uno, se deben crear las caras planas que mejor representan la superficie equiescalar en dicha rejilla. La superficie escalar que se halle se construye por cada cubo formado por 8 vértices con los valores de 8 vóxeles del espacio . Cada posibilidad se debe caracterizar por el número de vértices que toman valores por encima o por debajo de la superficie equiescalar, en este caso particular si hay objeto o no. Si un vértice está por encima de la superficie y otro vértice adyacente está por debajo, la superficie hallada intersectará la arista entre esos dos vértices.

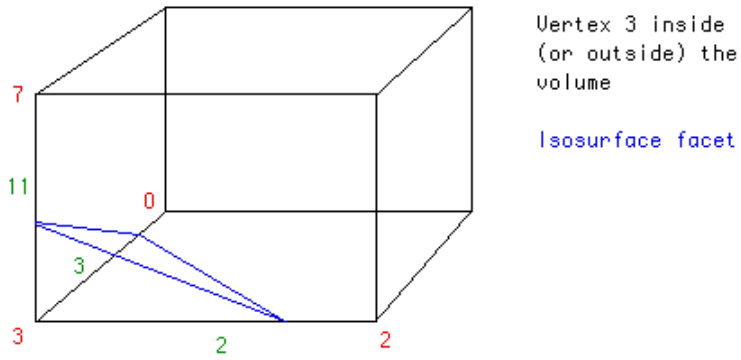


Figura 2.19: Triángulo que intersecta al cubo

En un único cubo de $2\times2\times2$ puntos existen 256 posibles combinaciones de puntos ocupados o no ocupados. En el gran abanico de posibilidades reside la dificultad del algoritmo. Primeramente se usa una tabla que localiza los vértices que se encuentran por debajo del valor que corresponde con la superficie equiescalar. De manera práctica se utiliza un índice de 8 bits donde cada bit corresponde con un vértice , con valor 1 si está ocupado o valor 0 en el caso contrario.

Algorithm 1 Localización inicial

Require: Iniciar el índice $cubeindex = 0$

1: **for all** $vertex = 0 : 7$ **do**
2: **if** $grid.value[vertex] < isolevel$ **then**
3: $cubeindex| = (1 << vextex)$
4: **end if**
5: **end for**

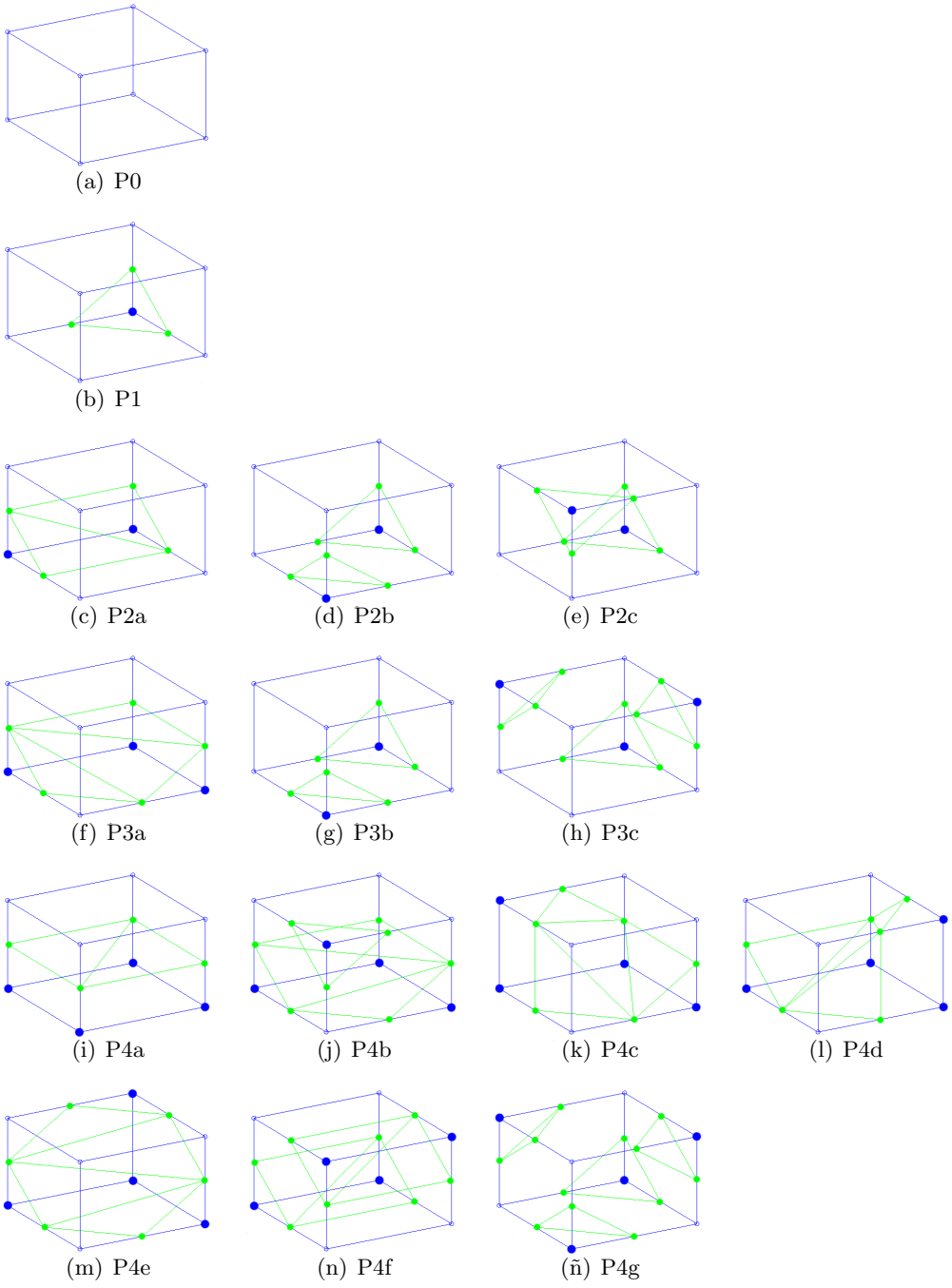
Posteriormente se busca en una tabla de aristas con el índice hallado anteriormente. La tabla indexada devuelve un número de 12 bits donde cada bit corresponde a cada arista del cubo, 1 si en la arista pertenece uno de los vértices de los triángulos intersectándola o 0 en caso contraria. Si no hay aristas interseccionadas se devuelve 0. Esto ocurre cuando el índice $cubeindex = 0$ (todos los vértices están sin ocupar) o $cubeindex = 0xFF$ (todos los vértices están ocupados), puesto que en este caso no existen triángulos porque el cubo estudiado no pertenece a la frontera del objeto.

Los puntos de intersección se deciden por interpolación lineal de los valores de gris entre los puntos blancos y negros. En este caso, los vértices se encuentran en el punto medio, ya que previamente lo que se ha hallado es una ocupación discreta y no existen valores intermedios entre unos puntos y otros.

Por último el algoritmo debe formar las caras de los triángulos desde los puntos donde la superficie interseccionа las aristas de la rejilla. Usando otra tabla de búsqueda indexada con el índice *cubeindex* se hallan los triángulos. Esta tabla devuelve un array con las aristas interseccionadas formando grupos de tres. Como máximo devolverá 5 grupos de 3 aristas, ya que puede haber 5 posibles triángulos.

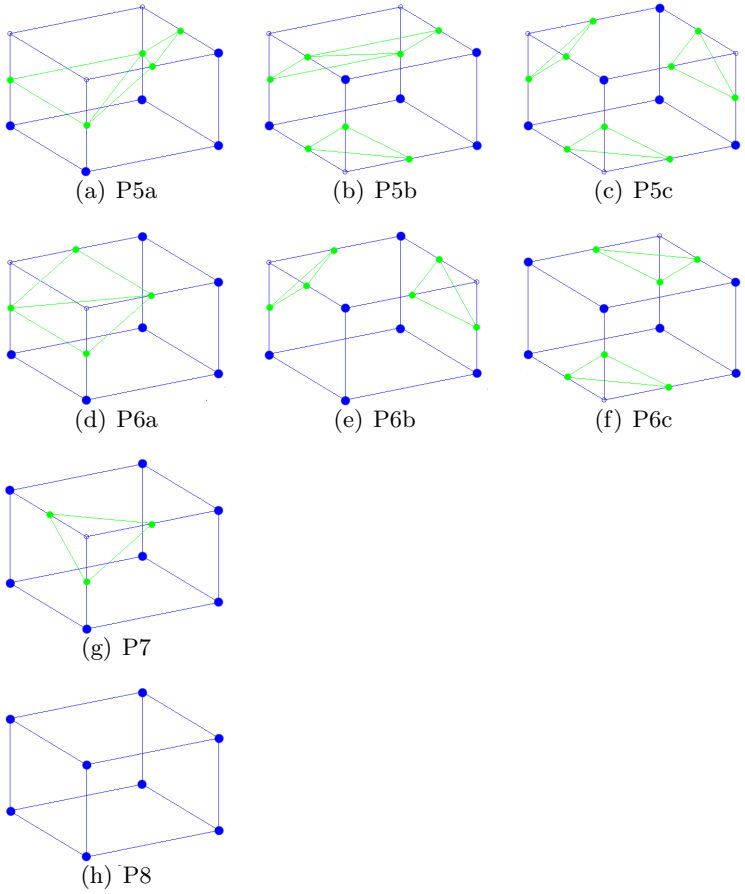
En la figura 2.20 y 2.21 se exponen todos los posibles casos, puesto que el resto de los casos por simetría son similares.

El algoritmo clásico de Marching Cubes causa ambigüedad topológica de las superficies equiescalares. Por ejemplo en el caso P2c, los vértices ocupados son vecinos, pero no va a existir conectividad de la malla construida.



(a) P0 (b) P1 (c) P2a (d) P2b (e) P2c (f) P3a (g) P3b (h) P3c (i) P4a (j) P4b (k) P4c (l) P4d (m) P4e (n) P4f (ñ) P4g

Figura 2.20: Casos de Marching Cubes I



(a) P5a (b) P5b (c) P5c

(d) P6a (e) P6b (f) P6c

(g) P7

(h) P8

Figura 2.21: Casos de Marching Cubes II

Capítulo 3

Sistemas de segmentación de fondo en cámaras estáticas

El problema general de la segmentación de imágenes consiste en un proceso de reconocimiento mediante el cual se pretende identificar y localizar los diferentes elementos u objetos que están presentes en una escena. Por lo tanto, se trata de dividir una imagen digital en regiones homogéneas con respecto a una o más características para facilitar un posterior análisis o reconocimiento.

En un sistema de visión artificial genérico, la fase de segmentación se encuadra entre las fases de preprocesamiento, en la que las imágenes son sometidas a diferentes operaciones de filtrado que ayuden a mejorar la calidad de la imagen o a destacar los objetos que posteriormente queremos segmentar.

La segmentación debe entenderse como un proceso que, a partir de una imagen, produce otra en la que cada píxel se asocia a una etiqueta distintiva al objeto al que pertenece. Así, una vez segmentada una imagen, se podría formar una lista de objetos consistentes en las agrupaciones de los píxeles que tengan la misma etiqueta. En el proyecto, se ha implementado una segmentación muy sencilla en la que los píxeles blancos corresponden con un objeto en el espacio y un píxel negro con el vacío.

En el presente proyecto la segmentación es fundamental para obtener las siluetas de los objetos que se desean reconstruir. El objetivo de la segmentación es diferenciar en las imágenes de cada cámara los objetos que no forman parte del fondo de la escena, y por lo tanto ocupan un espacio.

3.1. Métodos de Segmentación

Existen diferentes métodos de segmentación expuestos a continuación:

- **Segmentación basada en umbralización** En el proceso de umbralización se convierte una imagen en niveles de gris o en color a una imagen binaria, para etiquetar los píxeles que forman los objetos con un valor distinto a los píxeles que pertenecen al fondo. Computacionalmente, es una técnica poco costosa, que puede implementarse durante la captura de imágenes en tiempo real. Por lo general el umbral es calculado en función del histograma, en el que se tiene en cuenta la distribución de grises en la imagen y no la información espacial, por lo que dos imágenes muy diferentes pueden tener histogrmas similares. Por

esta razón, la umbralización, como única técnica de segmentación, resulta un método limitado para solucionar problemas reales. Aunque es bastante útil como complemento de otros métodos más avanzados.

- **Segmentación basada en la detección de contornos**: Existen diferentes técnicas que aprovechan la información proporcionada por las fronteras de los objetos que aparecen en una imagen. Para detectar los objetos distintos objetos de una escena diferenciándolos del fondo es una buena opción diferenciar en primer lugar las fronteras de dichos objetos.

- **Segmentación basada en crecimiento de regiones**: Este método consiste en determinar zonas de la imagen basándose en criterios de similitud y proximidad entre los píxeles. La homogeneidad o la falta de ella entre regiones adyacentes es la pauta que decide la unión o la división de las regiones dentro de una imagen. Los criterios de homogeneidad pueden ser: el nivel medio de gris, el color, la forma, etc. El resultado final debe ser una partición de la imagen en regiones homogéneas.

  A continuación se va a detallar un método basado en umbralización, el cual está implementado en el espacio inteligente para el cálculo de las siluetas de la imagen.

3.2. Modelo del fondo

Es necesario obtener un modelo de fondo para poder discriminar los objetos que existen en la escena. El fondo se entiende como todos los elementos relativamente estáticos que quedan dentro de la zona de trabajo, a este fondo le llamaremos "entrenamiento".

Para discriminar el fondo, utilizando un método sencillo, se parte de una imagen estática tomada por una cámara del fondo de la imagen $F(u, v)$ y una imagen de entrada $I(u, v)$, como se puede observar en la figura 3.1. En la imagen de entrada $I(u, v)$, además del fondo de la imagen, se tiene el objeto a identificar. Tras aplicar el proceso de segmentación obtendremos una imagen de salida $S(u, v)$ en la que se consigue únicamente el objeto a reconocer diferenciado (en negro) de una manera clara del resto de la escena (en blanco).

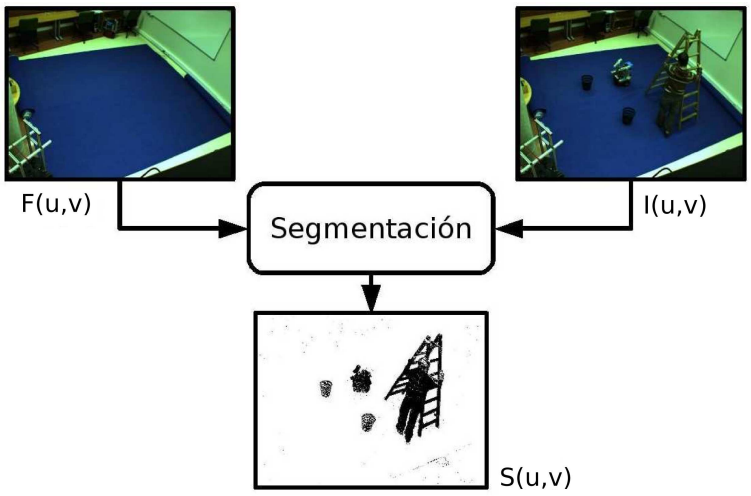


Figura 3.1: Diagrama general de segmentación simple

3.2.1. Método sencillo

Partiendo del diagrama propuesto en la Figura 3.1 se realiza un umbralizado basado en la diferencia entre la imagen de fondo y la imagen de entrada $F(u,v) - I(u,v)$, es decir:

$$\|F(u,v) - I(u,v)\| \geq \tau \longrightarrow S(u,v) = 1 \tag{3.1}$$

$$\|F(u,v) - I(u,v)\| < \tau \longrightarrow S(u,v) = 0 \tag{3.2}$$

El principal problema de este algoritmo es la aparición de ruido en la imagen de salida, para eliminarlo se suelen aplicar operaciones morfológicas (dilatación, erosión...). Dada la escasa robustez del algoritmo, puesto que se ve muy afectado por los cambios de iluminación en la imagen de entrada y la dificultad para ajustar el umbral τ, pues es muy difícil encontrar el balance entre eliminar el ruido completamente excluyendo también el objeto a reconocer, y mantener la forma del objeto a segmentar conservando también el ruido de la escena, se opta por calcular el valor de τ en función del píxel con coordenadas (u,v):

3.2.2. Umbral adaptativo

A diferencia del método sencillo en que el valor de salida de la imagen se calcula con un valor estático del umbral τ, en este método el umbral depende de qué píxel de la imagen se éste segmentando $\tau(u,v)$ obteniéndose el valor de la imagen de la siguiente forma:

$$\|F(u,v) - I(u,v)\| \geq \tau(u,v) \longrightarrow S(u,v) = 1 \tag{3.3}$$

$$\|F(u,v) - I(u,v)\| < \tau(u,v) \longrightarrow S(u,v) = 0 \tag{3.4}$$

3.2.3. Segmentación mediante distancia estadística

Para discriminar los objetos del fondo de la escena es necesario caracterizar estadísticamente dicho fondo, ya que las imágenes estáticas también se ven afectadas por ruido y cambios de iluminación.

Para este método partimos de múltiples imágenes de fondo, donde existen cambios en la iluminación, brillos, sombras, etc... que someterán al sistema a un entrenamiento con el objetivo de obtener la media y la varianza del fondo y mejorar el umbralizado.

La media de un conjunto de N imágenes de fondo se define como:

$$\hat{F}(u,v) = \frac{1}{N}\sum_{i=1}^{N} F_i(u,v) \tag{3.5}$$

La varianza de un conjunto de N imágenes de fondo se define como:

$$VAR_{F(u,v)} = \frac{1}{N}\sum_{i=1}^{N}(F_i(u,v) - \hat{F}(u,v)) \cdot (F_i(u,v) - \hat{F}(u,v))^T \tag{3.6}$$

Si las imágenes son a color la varianza será una matriz de 3×3

$$VAR_{F(u,v)} = \begin{pmatrix} \sigma_R^2 & 0 & 0 \\ 0 & \sigma_G^2 & 0 \\ 0 & 0 & \sigma_B^2 \end{pmatrix} \tag{3.7}$$

Si por el contrario las imágenes fueran de blanco y negro, la varianza solo sería de una dimensión:

$$VAR_{F(u,v)} = \sigma^2 \tag{3.8}$$

Si suponemos que el valor de cada píxel del fondo corresponde con una densidad de probabilidad gaussiana de media $\hat{F}(u,v)$ y varianza $VAR_{F(u,v)}$, se puede definir la distancia de Mahalanobis como:

$$\chi^2(u,v) = (I(u,v) - \hat{F}(u,v)) \cdot VAR_{F(u,v)}^{-1} \cdot (I(u,v) - \hat{F}(u,v))^t \tag{3.9}$$

La distancia de Mahalanobis es una medida que sirve para calcular la similitud entre variables aleatorias multidimensionales. A diferencia de la distancia euclídea, esta medida tiene en cuenta la correlación entre las variables aleatorias, ya que depende de la matriz de covarianza.

En este método se está caracterizando estadísticamente el fondo en un espacio de color RGB, que es variante a la iluminación. Por esta razón la modelización estadística no es capaz de absorver los errores producidos por los cambios de iluminación, ya que los colores se ven muy afectados por ellos. Para evitar los cambios muy bruscos en el color que provoca la aparición de sombras, es necesario trabajar con imágenes definidas en espacios invariantes a la iluminación.

3.3. Espacios de color invariantes a la iluminación

Las condiciones de iluminación provocan sombras en la imagen que pueden derivar en errores en la segmentación por lo que sería una buena opción utilizar métodos invariantes a la iluminación. Estos métodos consisten en utilizar imágenes carentes de sombras tanto para caracterizar estadísticamente el fondo como para las imágenes de entrada en el sistema de segmentación.

En el artículo [45] se define el problema de los cambios de iluminación y la eliminación de las sombras. El objetivo es obtener una imagen en escala de grises carente de sombras, en la que se mantenga la información proporcionada por el resto de la imagen.

En el artículo [46] se propone un método que registra imágenes en un espacio invariante de sombras unidimensional(en escala de grises). La formación de imágenes invariantes a la iluminación es posible gracias a proyectar los colores RGB de las imágenes en un espacio log-cromático, alrededor de una línea intrínseca. La pendiente de la línea depende de una calibración previa de la cámara. Para aplicar el método hay que considerar las siguientes suposiciones:

- **Superficies lambertianas** Una superficie es lambertiana si se comporta como un reflector perfectamente difuso, donde el brillo aparente es igual en todas las direcciones de vista. Bajo esta suposición, la radiación recibida por el sensor se asume que es proporcional al coseno del ángulo de incidencia $cos(i)$, que es el ángulo entre la normal de la superficie y el rayo de luz.
- **Fuente de luz Planckiana** Se considera una distribución espectral de potencia de luz radiada según el modelo de Plank.

- **Sensor de cámara de banda estrecha**. Los sensores de banda estrecha absorben la luz en un ancho de banda muy estrecho centrado en una determinada longitud de onda. Idealmente, su respuesta espectral, puede modelarse por una delta de Dyrac centrada en la longitud de onda correspondiente.

Según las anteriores suposiciones los colores RGB de un píxel de la imagen siguen la siguiente expresión:

$$\rho_k = \sigma S(\lambda_k) E(\lambda_k, T) Q_k \delta(\lambda - \lambda_k) \quad k = 1, 2, 3 \tag{3.10}$$

Donde:

- $\sigma S(\lambda_k)$ Representa la función de la reflectancia espectral de la superficie multiplicada por el factor lambertiano.
- $Q_k \delta(\lambda - \lambda_k)$ Este término expresa la respuesta espectral del sensor para cada canal de color centrada en la longitud de onda λ_k.
- $E(\lambda_k, T)$ Es la distribución de potencia espectral de la luz según el modelo de Plank para un rango de temperaturas bastante amplio $T = [2500°, 10000°]$. Se puede aproximar a la siguiente expresión:

$$E(\lambda, T) = I c_1 \lambda^{-5} e^{-\frac{c_2}{T\lambda}} \tag{3.11}$$

Donde I es la intensidad de luz global y las constantes c_1 y c_2 son fijas. Sustituyendo esta expresión en la ecuación 3.10 se obtiene lo siguiente:

$$\rho_k = \sigma I c_1 \lambda_k^{-5} e^{-\frac{c_2}{T\lambda_k}} S(\lambda_k) Q_k \quad k = 1, 2, 3 \tag{3.12}$$

A partir de los tres canales de color $\rho = (\rho_1, \rho_2, \rho_3)$ descritos en la ecuación 3.12 se definen los ratios de crominancia, en forma de vectores con coordenadas (χ_1, χ_2)

$$\begin{aligned} \chi_1 &= log(\tfrac{\rho_1}{\rho_3}) = log(\tfrac{s_1}{s_3}) + \tfrac{e_1 - e_3}{T} \\ \chi_2 &= log(\tfrac{\rho_2}{\rho_3}) = log(\tfrac{s_2}{s_3}) + \tfrac{e_2 - e_3}{T} \end{aligned} \tag{3.13}$$

Donde $e_k = -c_2/\lambda_k$ solo depende de la respuesta espectral de la cámara y no de la superficie considerada, $s_k = c_1 \lambda_k^{-5} S(\lambda_k) Q_k$, (no depende de la temperatura T).

La relación entre las coordenadas (χ_1, χ_2) forman una línea con dirección $\bar{e} = (e_1 - e_3, e_2 - e_3)$. Una superficie iluminada a distintas temperaturas de color T genera una recta con dirección $\bar{e}$ en un espacio de dos dimensiones con ejes (χ_1, χ_2), véase figura 3.2.

La cantidad de iluminación invariante puede ser expresada con la proyección del vector $\chi = (\chi_1, \chi_2)$ en la línea ortogonal definida por $\bar{e}^{\perp} = (cos(\theta), sen(\theta))$. Dos píxeles provenientes de una misma superficie pero iluminados de distinta manera serán proyectados en el mismo punto, como se representa en la figura 3.3

Para reducir la elección aleatoria de los ratios de crominancia, se puede normalizar cualquiera de las tres componentes de color por la media geométrica de los tres canales de color $\sqrt[3]{\rho_1 \rho_2 \rho_3}$. Así, se obtienen componenetes linealmente dependientes.

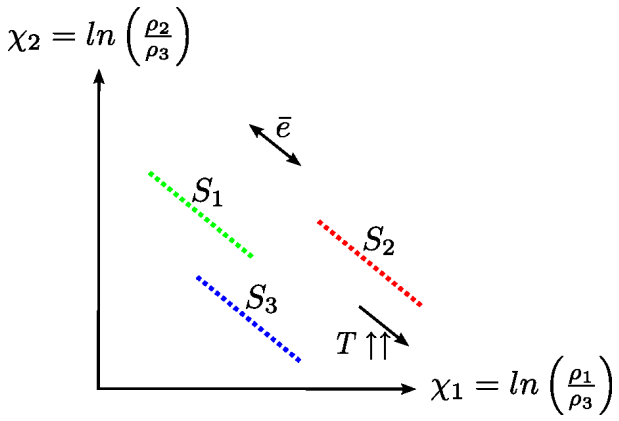


Figura 3.2: Representación de los valores de χ en 3 superficies para diferentes temperaturas de color T

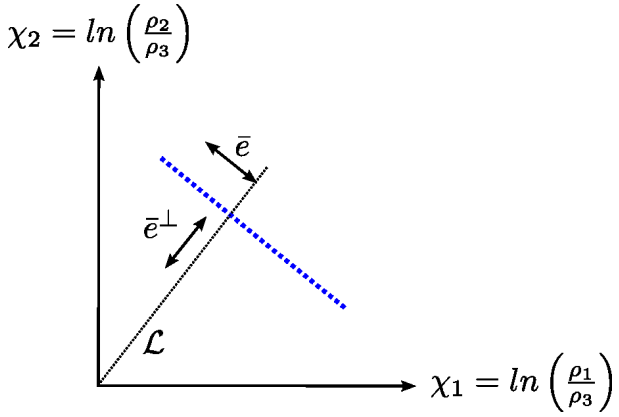


Figura 3.3: Proyección de los puntos de una superficie iluminada a distinta temperatura, sobre la recta definida por el vector $\bar{e}^{\perp}$

Eligiendo una descomposición adecuada, se obtiene un vector bidimensional equivalente a χ en el interior de la recta invariante, caracterizado por el ángulo de su pendiente θ

$$\mathcal{L}(\rho, \theta) = \chi_1(\rho)cos(\theta) + \chi_2(\rho)cos(\theta)sen(\theta) \tag{3.14}$$

Esta proyección equipara dos colores correspondientes a puntos vistos bajo diferentes tipos de iluminación. De esta manera, es posible relacionar un color ρ con su representación invariante a la iluminación.

Si se transforma el valor de cada píxel de una imagen a color $\mathcal{I}$ como el valor de θ en la ecuación 3.14 el resultado es de $\mathcal{L}(\mathcal{I}, \theta)$ es una imagen unidimensional invariante al tipo de iluminación. Por lo tanto se consigue una transformación global e independiente de la posición de los píxeles $q \in \mathcal{R}^2$, solo depende del valor de su color.

En definitiva, los colores RGB en un espacio log-cromático se proyectan alrededor de una línea intrínseca. Como, ya se ha visto, la pendiente de dicha línea depende del parámetro θ que debe ser calibrado con anterioridad. Una buena calibración puede lograrse simplemente grabando una secuencia de imágenes de una escena fija externa a lo largo de todo el día. Después de la calibración, solo con una imagen será suficiente para eliminar la sombra.

En la etapa de calibrado previa, requerida para usar de manera adecuada un espacio inva-

riante, se obtiene el valor del ángulo θ del vector $\bar{e}^{\perp}$. En la figura 3.4 se representa un diagrama modular del proceso en la obtención del espacio invariante, en el cual es necesario el autocalibrado del parámetro θ.

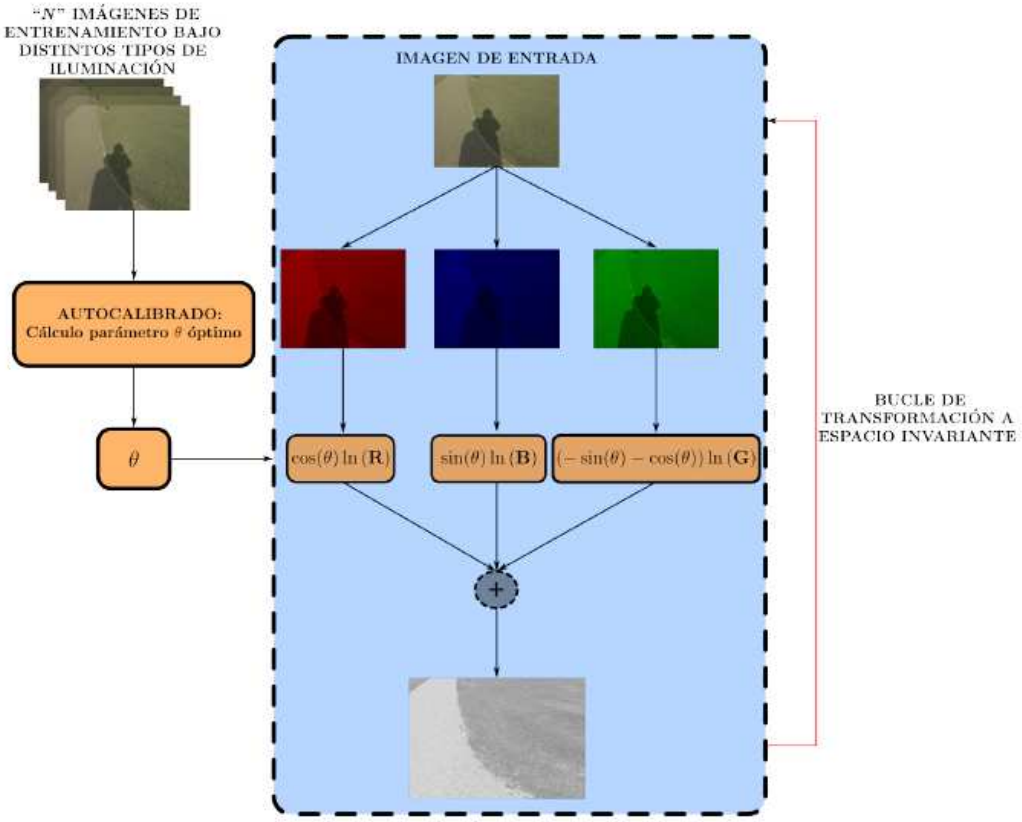


Figura 3.4: Diagrama del proceso de obtención del espacio invariante a la iluminación

El proceso de calibrado del parámetro θ está basado en una resolución por mínimos cuadrados explicada en el artículo [45].

Capítulo 4

Sistema de reconstrucción volumétrica a partir de múltiples cámaras

En el escenario tridimensional se van a disponer de N cámaras que van a dar como resultado un conjunto de siluetas S_i^n para $n = 0, \ldots, N-1$ en el instante t_i. En el figura 4.1 se observa un objeto O desde 4 cámaras diferentes en el instante t_1. Los centros ópticos de las cámaras están anotados en la figura como C^n

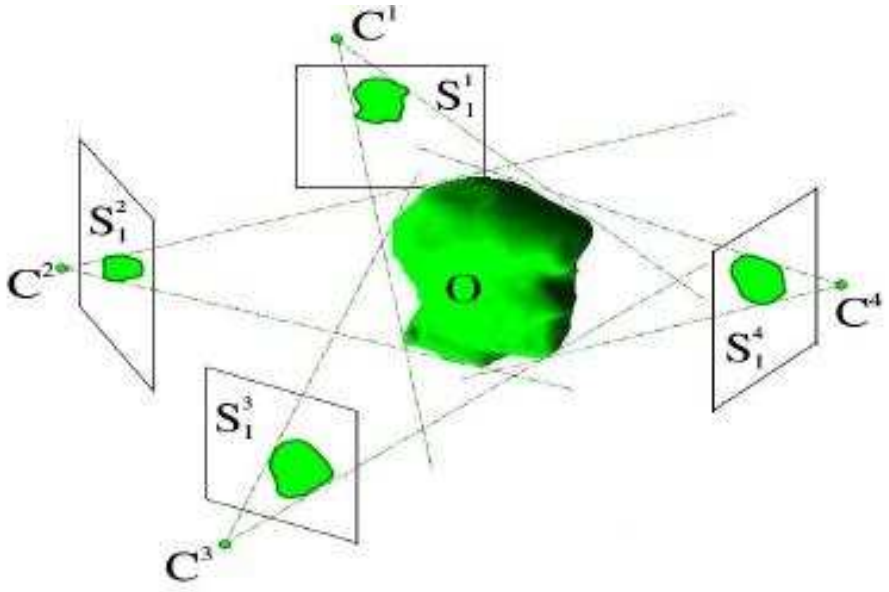


Figura 4.1: Escenario: Objeto O forma la silueta S_1^n en la cámara n en el instante t_1

El Visual Hull calculado es la intersección de los conos visuales que forman el conjunto de siluetas S_i^n de N cámaras con el vértice en el centro óptico de la cámara.

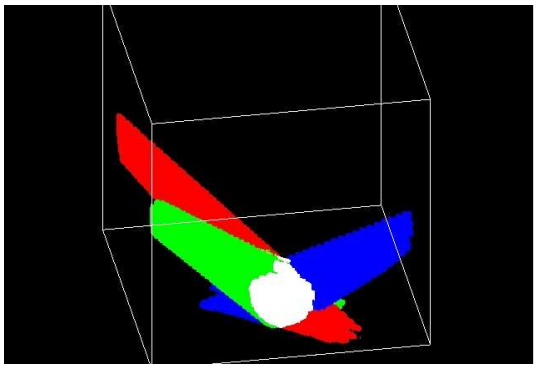


Figura 4.2: El Visual Hull visto como la intersección de los conos visuales de tres cámaras

Para obtener el Visual Hull como un conjunto de puntos discretos de ocupación se va a dividir el espacio en vóxeles y se van a hallar qué vóxeles se encuentran ocupados para conseguir un "grid" de ocupación en tres dimensiones 4.3.

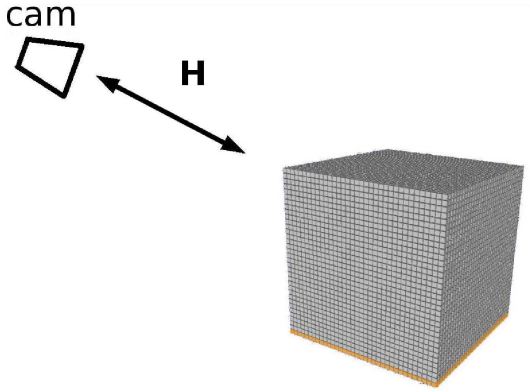


Figura 4.3: Espacio dividido en vóxeles

De manera práctica el grid se va a dividir en planos paralelos al suelo $z = j\Delta h$ o slices representados por imágenes, de dos dimensiones, donde cada píxel corresponde a un vóxel a una altura determinada, que proporciona la tercera dimensión.

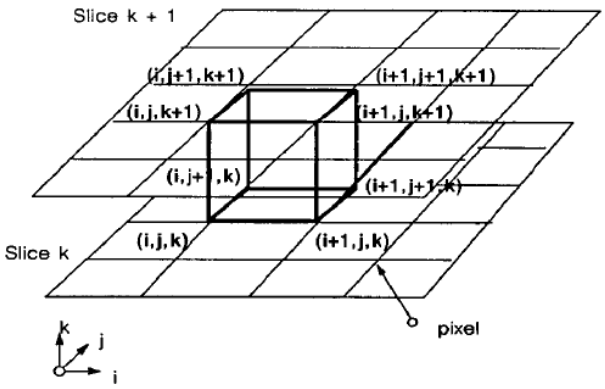


Figura 4.4: Slices

4.1. Aproximación del visual hull mediante homografías en múltiples planos paralelos

Para conseguir calcular, en tiempo real, el visual hull, es necesario utilizar un algoritmo con la menor complejidad computacional posible.

El objetivo previo es hallar los conos visuales de las siluetas, para poder obtener la intersección. Para simplificar el problema se va a hallar la proyección de las siluetas en cada slice. El conjunto de las proyecciones en cada slice es la intersección del cono visual con cada plano paralelo.

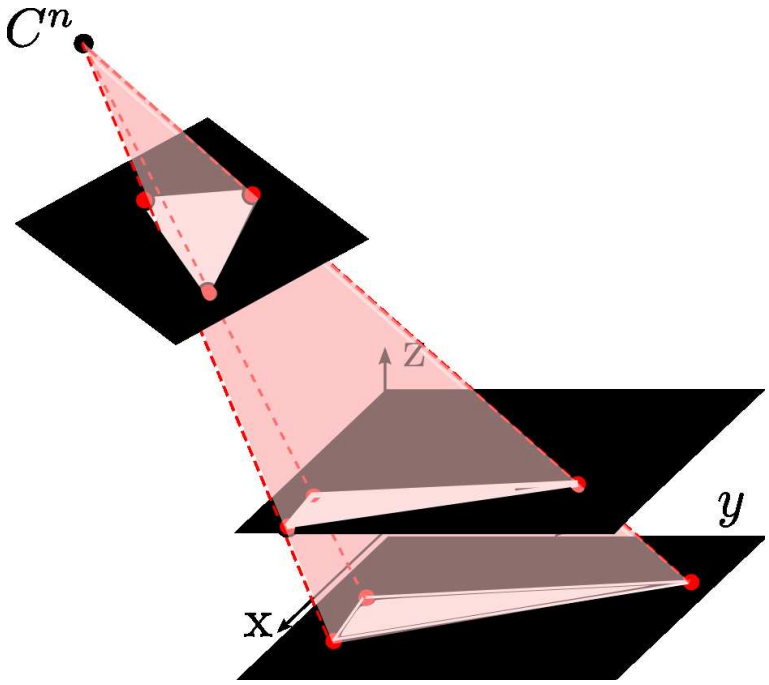


Figura 4.5: La intersección del cono visual con los slices

La ocupación tridimensional final se calcula a partir de la intersección de las imágenes de las siluetas proyectadas en planos paralelos al suelo 4.6. Aproximadamente se trata de la intersección de los conos visuales, ya que las siluetas proyectadas en cada slice pertenecen a dicho cono.

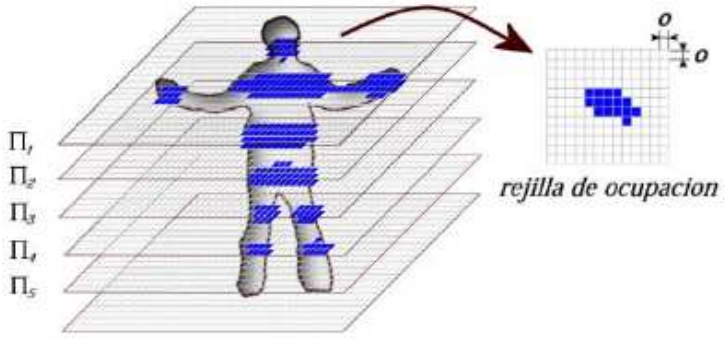


Figura 4.6: Rejilla de ocupación

Los pasos a seguir en la aproximación utilizada en el cálculo del Visual Hull es la siguiente:

- **Cálculo de las siluetas** Para cada cámara C_n se van a calcular el conjunto las siluetas S^n. El cálculo de las siluetas se realiza a través de una segmentación mediante distancia de Mahalanobis respecto a imágenes de fondo.

- **Bucle para cada "slice" o plano paralelo al suelo** j=0:J-1, siendo J el número de planos paralelos en el que dividimos el espacio tridimensional. Los planos corresponden a la expresión $z = j\Delta h$

  Proyección de las siluetas Para cada slice se debe hallar la matriz de homografía que relaciona el plano imagen de cada cámara con el plano paralelo al suelo correspondiente, y proyectar las imágenes segmentadas que contienen las siluetas (con píxeles en color blanco).

  Intersección de las siluetas proyectadas Las N imágenes proyectadas se intersectan para hallar el grid de ocupación en el plano $z = j\Delta h$.

  Almacenamiento de la ocupación Se recorre la imagen para obtener un array de datos de ocupación

En este documento el concepto de resolución está ligado con las longitudes de cada vóxel. En la dimensión x e y es la misma Δxy y en la dimensión z es Δh.

- **Resolución en x e y** La longitud Δxy es la medida en mm, que corresponde un lado de los píxeles de las imágenes donde se guarda el grid de ocupación de cada slice.La anchura y la altura de dichas imágenes dependerá, además de la resolución, de la longitud del espacio a construir:

$$WIDTH = \frac{LONGITUD\_X}{\Delta xy} \tag{4.1}$$

$$HEIGHT = \frac{LONGITUD\_Y}{\Delta xy} \tag{4.2}$$

- **Resolución en z** La longitud Δh es la distancia que existe entre los planos paralelos al suelo que componen el grid de ocupación.El número de planos paralelos o slices depende de la altura del espacio a reconstruir:

$$NUM\_SLICES = \frac{LONGITUD\_Z}{\Delta h} \tag{4.3}$$

La carga computacional de este proyecto es muy alta. La tarea más crítica temporalmente es proyectar a los diferentes planos las siluetas y realizar la intersección.

4.1.1. Cálculo de las matrices de homografía

Todas las homografías que se van a realizar relacionan el plano imagen y planos paralelos al suelo $z = h$.

Recordando del capítulo 2 los puntos pertenecientes al plano $z = h$ proyectados en el plano imagen I se pueden expresar de la siguiente forma:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \lambda \cdot K[\ R(:,1:2) \quad R(:,3) \cdot h + T] \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{4.4}$$

El plano imagen de la cámara se encuentra en una escala de píxeles, la transformación H calculada da como resultado puntos x e y en milímetros, por lo que se necesita una transformación M para obtener la equivalencia entre milímetros y píxeles (p, q) de una imagen que corresponda con el plano del slice correspondiente.

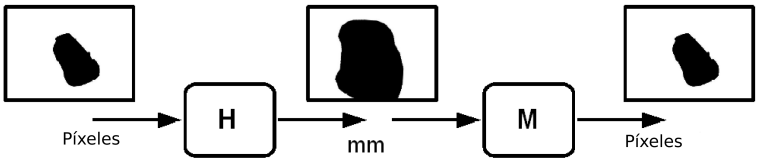


Figura 4.7: Diagrama de escalado

La matriz M es una matriz de escala:

$$M = \begin{pmatrix} \Delta xy & 0 & T_1 \\ 0 & \Delta xy & T_2 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.5}$$

El parámetro Δxy es muy importante ya que va a fijar la resolución tridimensional en el eje X y en el eje Y, es decir, es la medida del vóxel en x e y.

En el proyecto se han intercambiado las coordenadas X con las coordenadas Y para que las coordenadas en el ancho de la imagen correspondan al eje Y y las coordenadas en altura con el eje X. Para ello, la matriz M debe ser de la siguiente forma:

$$M = \begin{pmatrix} 0 & \Delta xy & T_1 \\ \Delta xy & 0 & T_2 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.6}$$

Los parámetros T_1 y T_2 son variables de traslación para ajustar el origen de la reconstrucción con respecto al origen de coordenadas del mundo.

Resumiendo el cálculo de un punto proyectado desde el plano imagen a un plano paralelo al suelo será:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \lambda \cdot (H \cdot M) \begin{pmatrix} p \\ q \\ 1 \end{pmatrix} \tag{4.7}$$

$$\begin{pmatrix} P \\ Q \\ \lambda \end{pmatrix} = (H \cdot M)^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{4.8}$$

4.1.2. Algoritmo de baja carga computacional para el cálculo de una homografía

El cálculo de una homografía de una imagen es una operación muy costosa computacionalmente. En la librería "OpenCV" existe una función llamada "cvWarpPerspective", que combina una función de interpolación con un método que consiste en rellenar la imagen destino de píxeles con un valor determinado excepto si alguno de ellos corresponde con la salida de la proyección de la imagen fuente. Esta función tiene una complejidad de $\mathcal{O}(width \cdot height)$.

En la tabla 4.1 se expresan los valores del tiempo de cómputo para hallar la homografía de una imagen cuyo objetivo es almacenar la proyección de una imagen en un plano en el espacio de $3000mm \times 3000mm$

Tabla 4.1: Tabla del tiempo de cómputo de la función cvWarpPerspective en función de la resolución utilizada

Resolución	Ancho de imagen	Altura de imagen	Tiempo función cvWarpPerspective
10mm	300 px	300 px	5475 μs
20mm	152 px	150 px	1432 μs
30mm	100 px	100 px	659 μs
40mm	76	75 px	411 μs
50mm	60 px	60 px	264 μs
60mm	52 px	50 px	216 μs
100mm	32 px	30 px	110 μs

Al decrecer el número de píxeles exponencialmente, el tiempo de cómputo también lo hace. En la figura 4.8 se comprueba como el tiempo de cómputo decrece exponencialmente en en función de la resolución de la imagen.

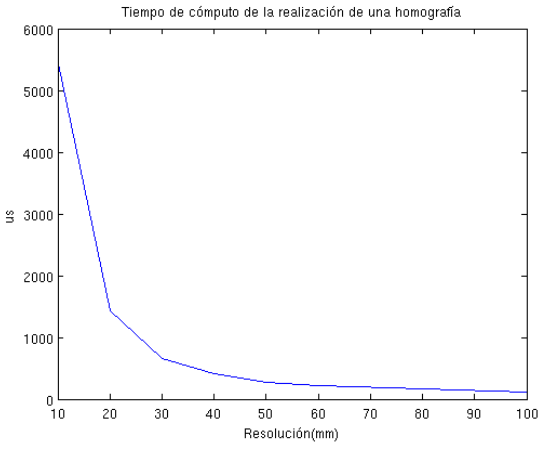
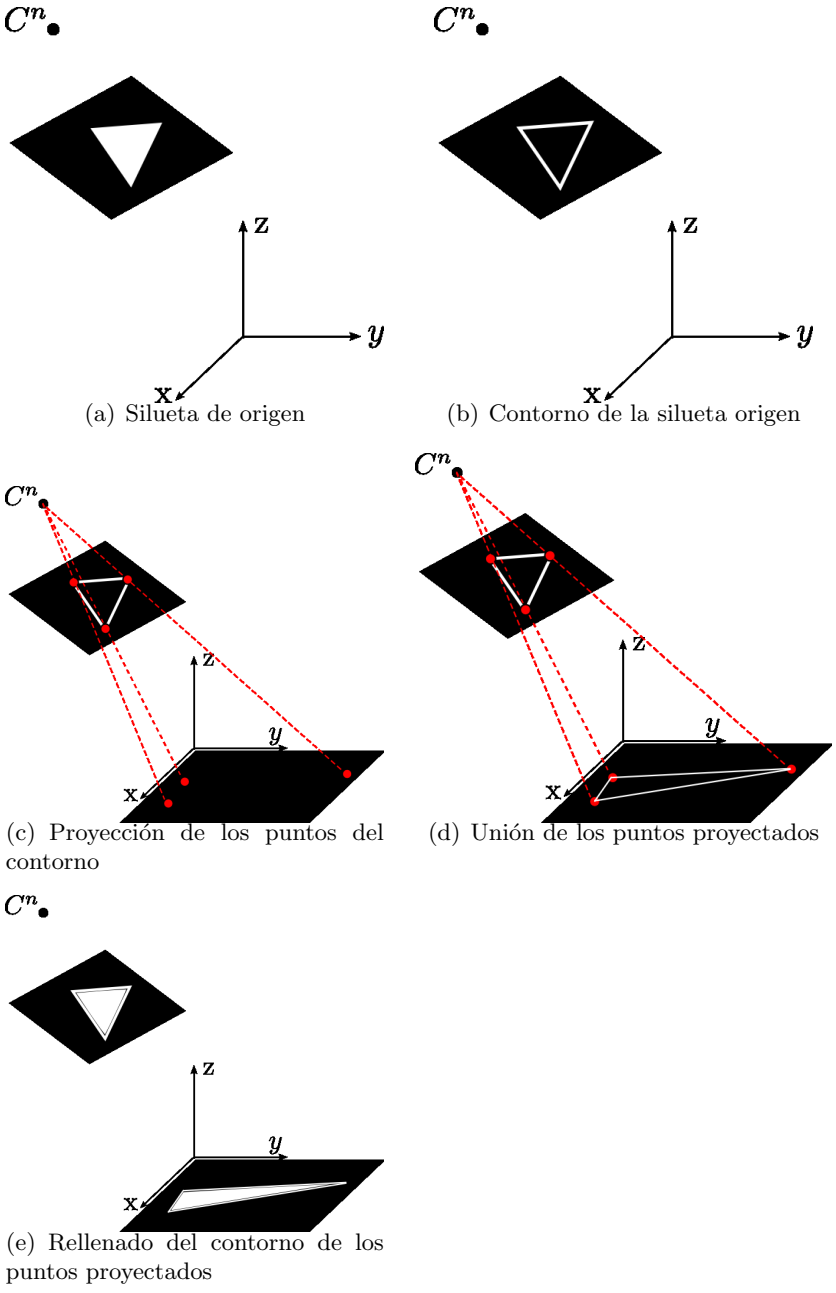


Figura 4.8: Tiempo de cómputo de la función cvWarpPerspective

La operación de la homografía se repite un número de veces igual al número de planos paralelos al suelo en los que se divide el espacio por el número de cámaras que hay, por lo que en

el proyecto se ha buscado una manera de minimizar el tiempo de realización de la homografía.

Para mejorar el tiempo de cómputo se halla el contorno de la imagen segmentada y se proyectan tan solo los puntos del contorno. Esto es posible porque en una transformación proyectiva las líneas rectas equivalen a otras líneas rectas. Los puntos del contorno se proyectan y se unen en la imagen proyectada como se observa en la figura 4.9.



(a) Silueta de origen

(b) Contorno de la silueta origen

(c) Proyección de los puntos del contorno

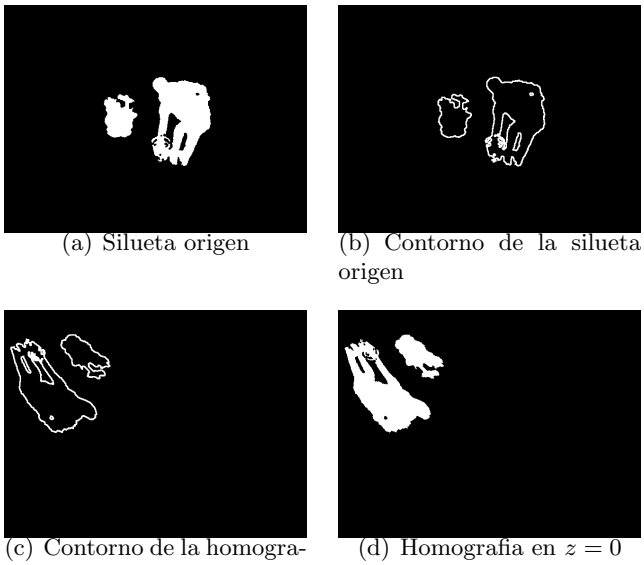
(d) Unión de los puntos proyectados

(e) Rellenado del contorno de los puntos proyectados

Figura 4.9: Pasos en el método del cálculo de la homografía

Entre los contornos calculados hay que distinguir entre contornos externos o internos, ordenándolos de manera jerárquica en dos niveles. Los de primer nivel corresponden a contornos externos y los de segundo nivel a contornos internos correspondientes a agujeros en el interior de los contornos de primer nivel. Por último, los píxeles del contorno de primer nivel se "rellenan" de blanco y posteriormente los contornos de segundo nivel de negro. En la figura 4.10 se

muestra la imagen original segmentada y las tranformaciones que se lleva a cabo para obtener la homografía.



(a) Silueta origen

(b) Contorno de la silueta origen

(c) Contorno de la homografía en $z = 0$

(d) Homografía en $z = 0$

Figura 4.10: Cálculo de homografías

Pueden existir problemas con algunos puntos. Si un plano imagen de la cámara estuviera perpendicular al plano del suelo, los rayos ópticos resultarían paralelos a dicho plano, esta posibilidad es necesaria contemplarla. Para el caso más extremo de rayos ópticos paralelos el valor de escalado será $\lambda = 0$, lo que conlleva a que el punto se proyecte en el infinito $x = \frac{X}{\lambda} \rightarrow \infty$, $y = \frac{Y}{\lambda} \rightarrow \infty$. La solución consiste en limitar a un valor mínimo la coordenada λ evitando valores excesivamente pequeños o incluso overflow en el cálculo de la homografía.

La tabla 4.2 muestra una comparación del tiempo de cómputo de realizar una homografía de manera convencional, y con el método de proyectar tan solo los puntos del contorno. Para estos resultados se han utilizado imágenes de $640px \times 480px$. Si ajustamos el tamaño de la imagen con la resolución el tiempo de cómputo de la función cvWarp es menor, pero de esta manera se puede comprobar que para imágenes de un tamaño considerable la función cvWarp es demasiado costosa temporalmente, siendo mucho más eficiente realizar la homografía tan solo de los puntos del contorno.

Tabla 4.2: Tabla comparativa entre tiempos de cómputo en función del algoritmo de homografía

Resolución	Convencional	Cálculo de Contornos	Homografía
10mm	$18756\mu s$	$1750\mu s$	$573\mu s$
60mm	$15585\mu s$	$1762\mu s$	$324\mu s$

En la figura 4.11 se observa que el método de proyectar tan solo los contornos es constante con la resolución, a diferencia del método convencional utilizando la función cvWarpPerspective que posee una fuerte dependencia que crece exponencialmente.

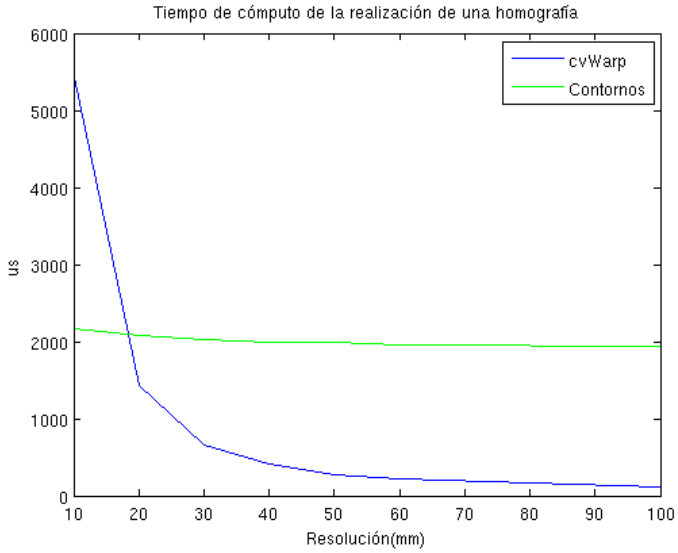


Figura 4.11: Comparación temporal entre la función cvWarp y el método de los contornos

En la figura 4.12 se observan los diferentes tamaños de imagen que son necesarios en función de la resolución:

Figura 4.12: Diferentes tamaños de imagen proyectada. $\Delta xy = 10 : 10 : 60(mm)$

En la figura 4.13 se observa la figura 4.12 ajustando todas las imágenes al mismo tamaño para observar el efecto de la resolución:

Figura 4.13: Diferentes resoluciones de imagen proyectada. $\Delta xy = 10 : 10 : 60(mm)$

4.1.3. Cálculo de los intersecciones de las imágenes proyectadas

La intersección de las homografías de cada cámara se realiza con una operación lógica "and" de las imágenes proyectadas en blanco y negro.

Otra opción ha sido tratar a los contornos de las imágenes como polígonos, y utilizar métodos de intersección de polígonos para así no tener que reconstruir la imagen para realizar la intersección, pero para figuras arbitrarias, no es efectivo, ya que se tratan de polígonos con muchos vértices.

Para el frame 344, mostrado en la figura9.5, se han medido los diferentes tiempos de cómputo necesarios para realizar una intersección de las proyecciones sobre el plano $z = 0$ de dos cámaras:

Tabla 4.3: Tabla temporal para el cálculo de la intersección mediante polígonos

Resolución	Contorno	Proyección	Intersección polígonos	Reconstrucción del polígono
10mm	3201 μs	266 μs	1828 μs	354 μs
60mm	3231 μs	198 μs	712 μs	114 μs

Para realizar la intersección de las imágenes, se realizan con el mismo tamaño que el de las homografías:

El ancho de la imagen será $WIDTH = \frac{LEN_X}{\Delta xy}$ $HEIGHT = \frac{LEN_Y}{\Delta xy}$, entonces el tiempo de cómputo de la intersección depende directamente del tamaño de la imagen que depende a su vez

de la resolución utilizada o el tamaño de los vóxeles.

En la figura 4.14 representa el tiempo de cómputo de la intersección. Decrece exponencialmente en función de la resolución.

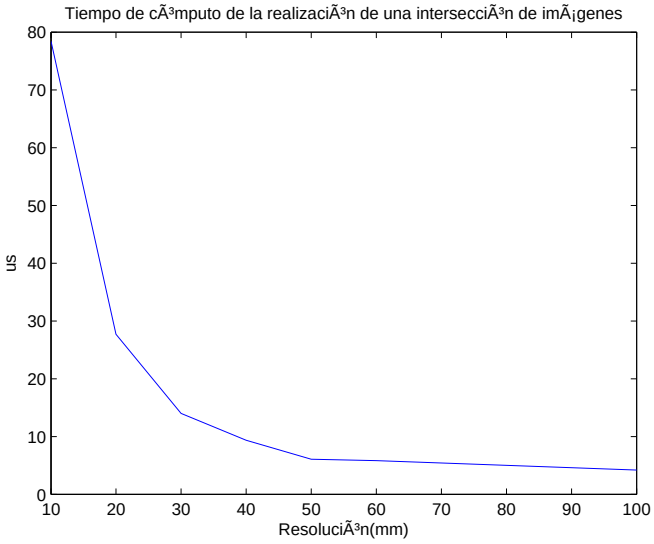


Figura 4.14: Tiempo de cómputo de la función cvAnd

En la siguiente tabla se expresan los tiempos de cómputo de la función cvAnd:

Tabla 4.4: Tabla del tiempo de cómputo de la realización de una intersección entre imágenes

Resolución	Ancho de imagen	Altura de imagen	Tiempo intersección de imágenes
10mm	300 px	300 px	78.4 μs
20mm	152 px	150 px	27.7 μs
30mm	100 px	100 px	14.0 μs
40mm	76	75 px	9.7 μs
50mm	60 px	60 px	6.1 μs
60mm	52 px	50 px	5.8 μs
100mm	32 px	30 px	4.2 μs

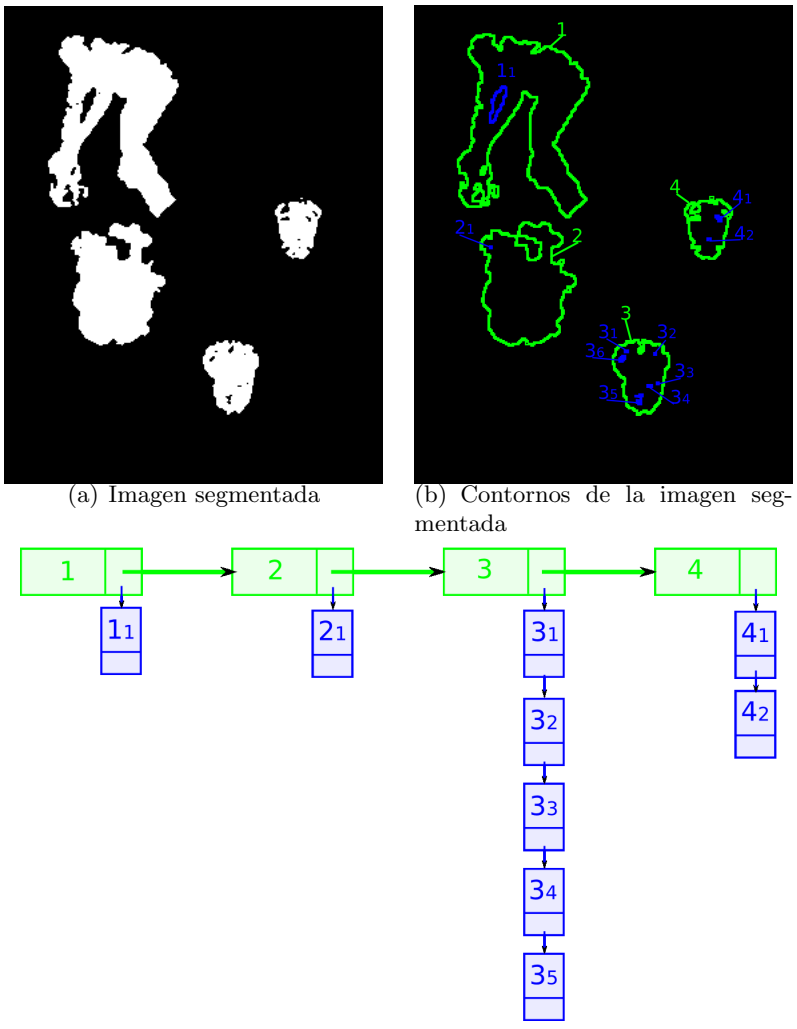
Si se comparan la tabla 4.3 y 4.4 se llega a la conclusión de que no merece la pena realizar la intersección con polígonos, ya que es más costoso computacionalmente utilizar polígonos al tratarse de formas naturales que equivalen a polígonos con muchos vértices.

4.1.4. Algoritmo detallado del cálculo de la ocupación tridimensional

A continuación se va a detallar el algoritmo utilizado en el cálculo de la ocupación tridimensional incorporando el cálculo previo de los contornos para la implementación del nuevo método del cálculo de las homografías para disminuir el gasto computacional:

- **Cálculo de las siluetas** Para cada cámara C^n se van a calcular el conjunto las siluetas S_i^n. El cálculo de las siluetas se realiza a través de una segmentación mediante distancia de Mahalanobis respecto a imágenes de fondo.

- **Cálculo de los contornos** Para cada silueta de la imagen segmentada $S(u, v)$ se hallan los puntos de los contornos externos e internos (si las siluetas poseen agujeros). Al conjunto de los contornos se les anota como F^n. En la figura 4.15 se muestra un ejemplo de lista de contornos donde los contornos verdes son los externos y los contornos azules son internos. Se guardan en una lista de dos niveles donde están enlazados los posibles contornos internos con el contorno externo que los contienen.



(a) Imagen segmentada

(b) Contornos de la imagen segmentada

(c) Lista de contornos

Figura 4.15: Contornos externos e internos

- **Bucle para cada "slice" o plano paralelo al suelo** j=0:SLICES-1, siendo $SLICES$ el número de planos paralelos en el que dividimos el espacio tridimensional. Los planos Π_j corresponden a la expresión $z = j\Delta h$

  Proyección de los contornos Para cada slice se debe hallar la matriz de homografía H_j, y proyectar los puntos de los contornos F^n de cada cámara y así obtener una lista de contornos anotados como $F_j'^n$ proyectados en cada plano Π_j .

  Reconstrucción de los contornos proyectados Para cada cámara C^n Se unen los puntos proyectados del conjunto de contornos $F_j'^n$ y se rellenan de blanco los puntos exteriores e interiores de negro para obtener una imagen $IP(u, v)_j^n$ correspondiente con la homografía de la imagen segmentada.

Intersección de las siluetas proyectadas Las N imágenes proyectadas se intersectan para hallar una imagen $II(u,v)_j$ el grid de ocupación en el plano $z = j\Delta h$.

Almacenamiento de la ocupación Se recorre la imagen $II(u,v)_j$ para obtener un array de datos de ocupación $DatosX[\,]$, $DatosY[\,]$ y $DatosZ[\,]$ de manera que se localicen rápidamente los datos ocupados dentro del grid. También es necesario guardar una estructura llamada $Imagen3D[WIDTH][HEIGHT][SLICES]$ donde $WIDTH$ y $HEIGHT$ es la anchura y la altura de la imagen $II(u,v)$.

Los índices u, v, j corresponden con puntos tridimensionales del espacio que possen una equivalencia en unidades métricas en mm:

- $u \longrightarrow y = u \cdot \Delta xy + T_1$
- $v \longrightarrow x = v \cdot \Delta xy + T_2$
- $j \longrightarrow z = j \cdot \Delta h$

Los puntos tridimensionales en unidades métricas se guardan en una archivo de datos para registrar la información de la ocupación en cada momento t_i.

El pseudocódigo 2 expresa el algoritmo utilizado para cada frame en el instante t_i:

Algorithm 2 Cálculo del visual hull

```
1: for all n = 0 : N_camaras − 1 do
2:    F_n = HallarContornos(S(u,v)_n);
3: end for
4: N_puntos = 0;
5: p = 0;
6: for all j = 0 : SLICES − 1 do
7:    for all n = 0 : N − 1 do
8:       IP_j^n = Proyectar(F^n)
9:       if n = 0 then
10:         II_j = IP_j^n
11:      else
12:         II_j = II_j & IP_j^n
13:      end if
14:   end for
15:   for all u = 0 : WIDTH − 1 do
16:      for all v = 0 : HEIGHT − 1 do
17:         if II_j(u,v) > 0 then
18:            DatosX(p) = u;
19:            DatosY(p) = v;
20:            DatosZ(p) = j;
21:            Imagen3D(u,v,j) = 255
22:            p++;
23:            N_puntos++;
24:            Guardar los datos métricos en un archivo:
25:            fprintf(f_i, '%d%d%d', vΔxy + T_2, uΔxy + T_1, jΔh)
26:         end if
27:      end for
28:   end for
29: end for
```

4.2. Mallado tridimensional y coloreado fotorrealista

Una vez calculada la ocupación 3D y las estructuras de datos que la almacena. Esta estructura le sirve al algoritmo de MarchingCubes para hallar una superficie que englobe a los objetos hallados. La superficie equiescalar que calcula el MarchingCubes coincide aproximadamente con la corteza de los objetos.

Con la estructura de datos $Imagen3D[][][]$ le proviene al algoritmo de un entramado de puntos separados por las las longitudes de los vóxeles donde cada vértice coincide con los valores del grid de ocupación donde los datos ocupados se definen con el valor de 255 y los no ocupados con el valor 0.

La dimensión de los vóxeles viene dada por la resolución $\Delta h > 0$ y $\Delta xy > 0$, véase en la figura 4.16.

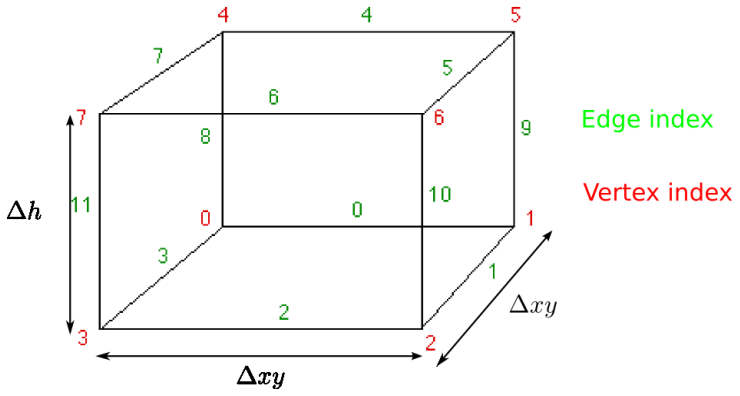


Figura 4.16: Vóxel

Para ejecutar el algoritmo de Marching Cubes de manera eficiente, se debe realizar el algoritmo solo para los puntos ocupados y sus puntos vecinos. Cada punto 3D, posee 27 vecinos, véase la figura 4.17. Es necesario recorrer la estructura de $datosX[]$, $datosY[]$ y $datosZ[]$, más los 27 puntos que existen alrededor del punto ocupado.

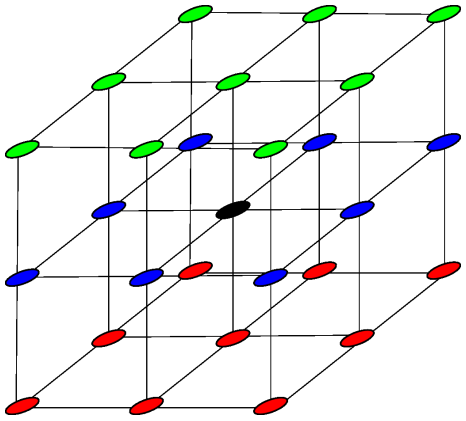


Figura 4.17: Entramado de un punto ocupado y sus vecinos

En el código siguiente se expresa la manera de recorrer los puntos ocupados y sus vecinos. Se utiliza una estructura llamada "cubos_procesados" para controlar que un punto no es procesado múltiples veces, ya que un vecino de un punto ocupado también puede ser estar ocupado.

```
memset(cubos_procesados,0,HEIGHT_FINAL*WIDTH_FINAL*SLICES);
```

```
for(int i=0;i<numero_puntos;i++)
{
  for(int j=0;j<27;j++)
  {
        iX=datosX[i]+indiceX[j];
        iY=datosY[i]+indiceY[j];
        iZ=datosZ[i]+indiceZ[j];
        if(iX<WIDTH_FINAL && iY<HEIGHT_FINAL && iZ<SLICES && iX>=0 && iY>=0 && iZ>=0)
        {
              if (cubos_procesados[iX][iY][iZ]==0)
              {
                    vMarchCube(iX, iY, iZ, fStepSizeX,fStepSizeY,fStepSizeZ);
                    cubos_procesados[iX][iY][iZ]=1;
              }
        }
  }

}
```

Las variables *indiceX*[], *indiceY*[] e *indiceZ*[] origina las 27 posibles posiciones de todos los puntos vecinos de un punto ocupado.

```
unsigned char cubos_procesados[WIDTH_FINAL][HEIGHT_FINAL][SLICES];
indiceX[27]={-1,0,1,-1,0,1,-1,0,1,-1,0,1,-1,0,1,-1,0,1,-1,0,1,-1,0,1,-1,0,1};
indiceY[27]={-1,-1,-1,0,0,0,1,1,1,-1,-1,-1,0,0,0,1,1,1,-1,-1,-1,0,0,0,1,1,1};
indiceZ[27]={-1,-1,-1,-1,-1,-1,-1,-1,-1,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1};
```

Cada punto que se estudia para construir la malla forma parte de la esquina inferior derecha de un cubo, en el cual hay que hallar los triángulos que formen la malla en ese punto. En la figura 4.18 se observan los cubos formados por 27 puntos del espacio.

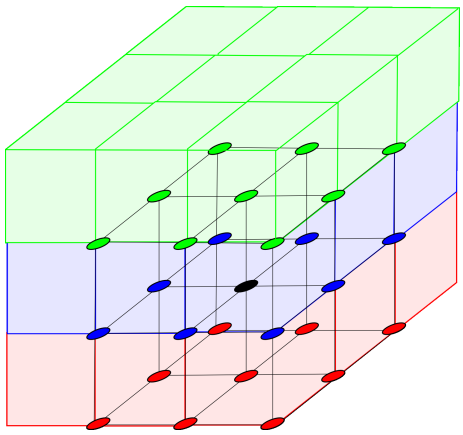


Figura 4.18: Cubos a estudiar alrededor de un punto ocupado

En el pseudocódigo 3 se implementa el algoritmo de Marching Cubes para un solo cubo:

Los arrays que aparecen en el pseudocódigo son utilizados como tablas de búsqueda para acelerar la construcción de la malla:

- **"CubeEdgeFlags"** es una tabla donde se guardan las aristas donde existe un vértice que forme un triángulo. El índice de acceso al array es un número de 0 a 255 que indica un caso determinado del MarchingCubes entre los 256 posibles.

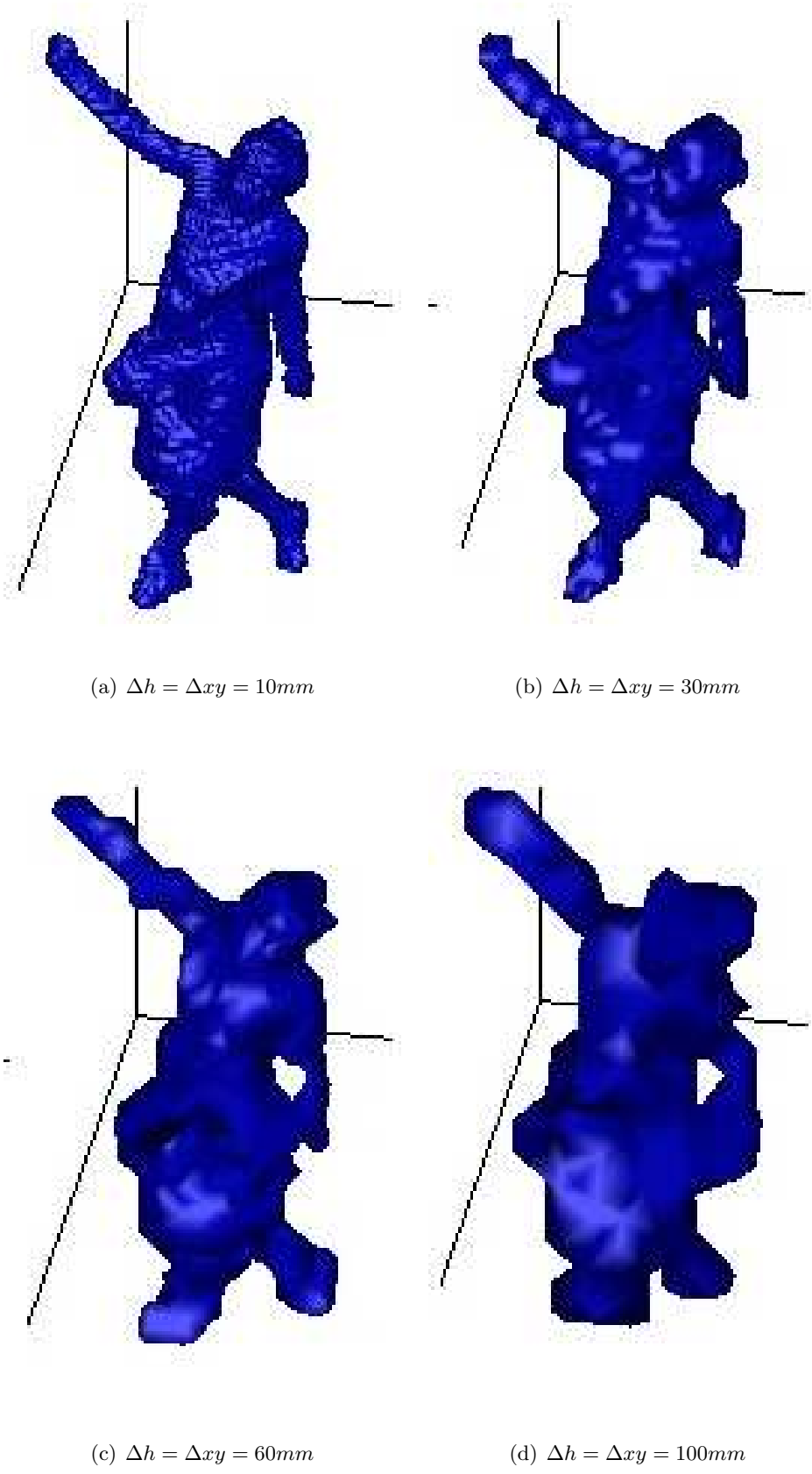
Algorithm 3 Localización inicial

Require: Iniciar el índice $cubeindex = 0$

```
for all vertex = 0 : 7 do
    if grid.value[vertex] > isolevel then
        cubeindex| = (1 ≪ vextex)
    end if
end for
EdgeFlags = CubeEdgeFlags[cubeindex];
for all edges = 0 : 11 do
    if (EdgeFlags&(1 ≪ edges)) == 1 then
        Cálculo de los vértices de los triángulos que pertenecen a la arista edge
        vertex[edges]=grid.position[vertex]+0.5*DirectionOffset[edges]
    end if
end for
for all triangle = 0 : 4 do
    for all corner = 0 : 2 do
        Calcula el orden de los vértices para formar los triángulos
        edge = TriangleConnectionTable[cubeindex][3*triangle+corner];
        El nuevo vértice es ->vertex[edge]
    end for
end for
```

- **"DirectionOffset"** es una tabla donde se guardan las direcciones de cada arista que indica si hay que sumar o restar al vértice del cubo para calcular el vértice del triángulo.
- **"TriangleConnectionTable"** es una tabla donde se guardan el orden de cada arista para formar los 5 posibles triángulos por cada cubo. El índice de acceso al array es un número de 0 a 255 que indica un caso determinado del MarchingCubes entre los 256 posibles.

En la figura 4.19 se muestra el efecto de utilizar diferentes resoluciones, o lo que es lo mismo, diferentes tamaños de cubos:



(a) $\Delta h = \Delta xy = 10mm$

(b) $\Delta h = \Delta xy = 30mm$

(c) $\Delta h = \Delta xy = 60mm$

(d) $\Delta h = \Delta xy = 100mm$

Figura 4.19: Diferentes resoluciones

Una vez se hallan los triángulos se deben calcular sus orientaciones. El grado de visión de cada cámara lo medirá el producto escalar entre el eje z de cada cámara y el vector normal de la superficie de cada triángulo.

Sea un triángulo de vértices V_0, V_1 y V_2, los vectores directores de sus aristas se definen como:

$$\vec{V}_{10} = V_1 - V_0 \tag{4.9}$$

$$\vec{V}_{20} = V_2 - V_0 \tag{4.10}$$

El vector normal a la superficie del triángulo es el producto vectorial de los vectores directores de la arista:

$$\vec{N} = \vec{V}_{10} \times \vec{V}_{20} \tag{4.11}$$

Es necesario tener en cuenta la orientación de la normal, siempre la tenemos que calcular de forma saliente al volumen ocupado.

En las figuras 4.20 se muestran las normales a los triángulos reconstruidos

El criterio para hallar la cámara que mejor visualiza al triángulo es hallar el ángulo existente entre el rayo óptico de la cámaras y la normal del triángulo.

$$\vec{N}_{O_c} = R_c \cdot \vec{N}_{O_w} \tag{4.12}$$

El producto escalar entre la normal en coordenadas de la cámara y el eje z de la cámara aporta información sobre el ángulo de la cámara y el triángulo. El vector director unitario del eje z será $\vec{k}_{Oc} = (0, 0, 1)$. La normal del triángulo se normaliza para que su módulo valga la unidad.

$$\vec{n}_{O_c} = \frac{\vec{N}_{O_c}}{|\vec{N}_{O_c}|} \tag{4.13}$$

$$\vec{n}_{O_c} \cdot \vec{k}_{O_c} = cos(\alpha)|n_{O_c}| \cdot |k_{O_c}| = cos(\alpha) \tag{4.14}$$

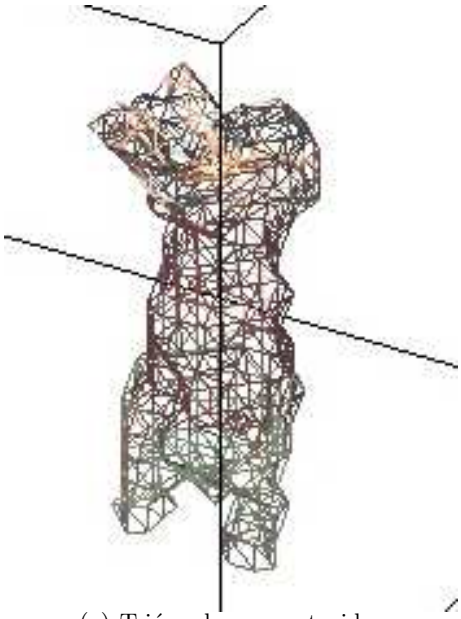
El mejor caso de visualización de la cámara es que ambos vectores fueran antiparalelos, es decir que $\alpha = \pi$ rad . En el proyecto se ha supuesto que existe visión por parte de la cámara si el ángulo que forman ambos vectores cumple que $\frac{\pi}{2} < \alpha < -\frac{\pi}{2}$ por lo que se cumple que:

$$\vec{n}_{O_c} \cdot \vec{k}_{O_c} = cos(\alpha) < 0 \tag{4.15}$$

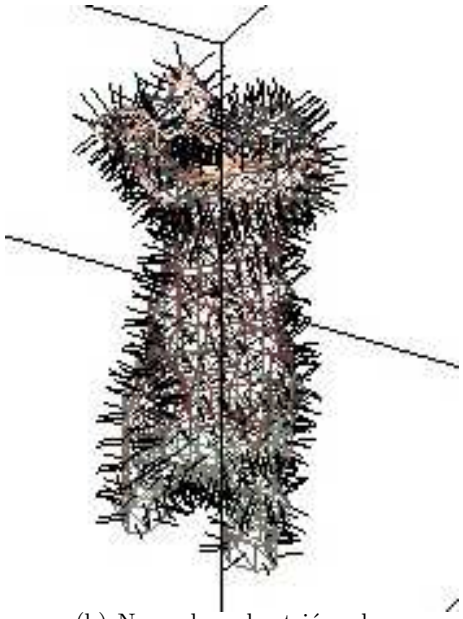
Para hallar la distancia del triángulo a la cámara se ha considerado la distancia del centro del triángulo entre el centro óptico de la cámara.

Para hallar el centro del triángulo se calcula la posición media de los 3 vértices del dicho triángulo $c_T = \frac{V_0+V_1+V_2}{3}$ y la posición del centro óptico de la cámara en coordenadas del mundo viene dada por el vector de traslación de la cámara:

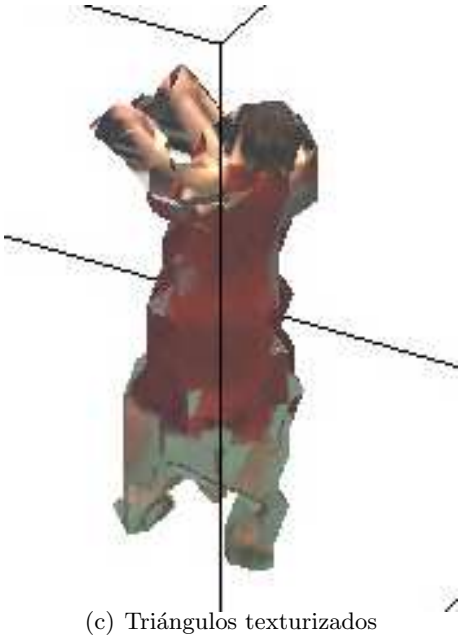
$$d^2 = \|c_T - T\| \tag{4.16}$$



(a) Triángulos reconstruidos



(b) Normales a los triángulos



(c) Triángulos texturizados

Figura 4.20: Triangulos texturizados con una resolución de $\Delta h = 100mm$

Para un mayor rendimiento de la tarjeta gráfica se va a texturizar con una sola textura 4.21 de tamaño máximo $2048px \times 2048px$ que incluya a todas las imágenes de la cámara. El tamaño de la textura siempre debe ser potencia de 2.

Figura 4.21: Una única textura

4.2.1. Algoritmo del pintor

Existe un problema cuando se texturizan triángulos que se encuentran en la misma dirección desde una cámara determinada pero a distancia distancia, porque aunque la orientación indique la texturización con dicha cámara, puede que esa cámara no "vea" al triángulo, porque otro objeto esté delante de él. En la figura 4.22 se representan dos triángulos con la misma orientación, y en la misma línea si se unen los centros de los triángulos y el centro óptico de la cámara, pero a diferentes distancias de la cámara. Según la orientación y la posición va a resultar que los dos triángulos se texturizan con la misma cámara.

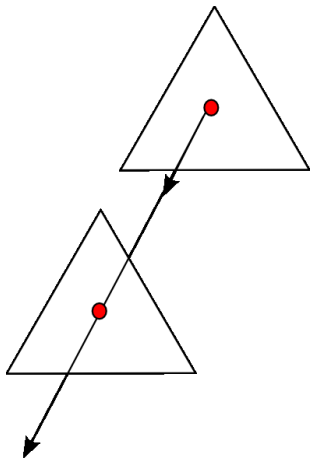


Figura 4.22: Problema del pintor

En la figura 4.23, se representa a un chico con un patrón de calibración delante de él. A causa del problema anteriormente descrito, en vez de texturizar el pantalón con la imagen adecuada, la textura que se le asigna corresponde al patrón de calibración, porque tienen los triángulos que forman el pantalón tienen la misma orientación.

Figura 4.23: Ejemplo del problema que resuelve el algoritmo del pintor

El objetivo del algoritmo del pintor consiste en texturizar los triángulos, en función, no solo de la orientación, sino también, en función de la distancia al objeto. Los pintores pintan primero el fondo y posteriormente los objetos que están más cerca.

Para aplicar el algoritmo al proyecto es más eficiente precalcular las cámaras que van a texturizar los triángulos con el algoritmo del pintor, y posteriormente representar todos los triángulos tan solo una vez.

Para implementar este algoritmo se va a almacenar en una estructura de N veces el tamaño de las imágenes a color e ir guardando el índice del triángulo en las coordenadas bidimensionales de la cámara elegida para texturizar, que correspondan al centro del triángulo y también guardar la distancia a la que se encuentra del centro óptico de dicha cámara. Si posteriormente se encuentra un triángulo con menor distancia y misma orientación a la cámara se sustituirá el valor anterior por los datos del nuevo triángulo y el anterior triángulo no texturizará.

Algorithm 4 Algoritmo del pintor

1: $distancia[num\_camaras][WIDTH][HEIGHT] = -1$;
2: $pixels[num\_camaras][WIDTH][HEIGHT] = -1$;
3: **for all** $iTriangulo = 0 : num\_triangulos - 1$ **do**
4: mejor_camara=Calcular la camara con mejor orientación
5: [u,v]=Calcular las coordenadas del centro del triángulo bidimensionales correspondientes a la mejor cámara
6: distancia =Calcular la distancia del centro del triángulo a la mejor cámara
7: **if** distance[mejor_camara][u][v]==-1 || distance[mejor_camara][u][v]>distancia **then**
8: $distancia[num\_camaras][WIDTH][HEIGHT] = distancia$;
9: $pixels[num\_camaras][WIDTH][HEIGHT] = iTriangulo$;
10: **end if**
11: **end for**

En la figura 4.24 se observa como aplicando el algoritmo, los triángulos que equivalen al pantalón no se han texturizado porque se ha optado en que si hay un error no se texturice.

Figura 4.24: Resultado de la aplicación del algoritmo del pintor

Capítulo 5

Detección de personas y objetos a partir de la reconstrucción volumétrica

5.1. Detección de caras mediante el algoritmo de Viola & Jones

El algoritmo de Viola & Jones describe una máquina de aprendizaje para detectar objetos visuales en tiempo real que introduce tres contribuciones importantes en la detección.

Este algoritmo introduce el concepto de imagen integral que permite que las características de la imagen se computericen mucho más rápido, que a partir de la imagen original.

El algoritmo de aprendizaje se basa en AdaBoost [47] que selecciona un conjunto reducido de las características visuales más críticas de un conjunto más amplio para obtener un clasificador eficiente.

Para acelerar el proceso, se utiliza un método para combinar incrementalmente clasificadores más complejos en cascada que permite que el fondo de la imagen sea descartado rapidamente. La cascada puede ser vista como un objeto específico como un mecanismo previo para descartar regiones de la imagen donde probablemte no exista el objeto a detectar.

La imagen integral se define como:

$$Ii(u,v) = \sum_{u' \leq u, v' \leq v} I(u', v') \tag{5.1}$$

$II(u,v)$ es la imagen integral de $I(u,v)$

Si definimos a $s(u,v)$ como como la suma acumulativa por fila resulta que:

$$s(u,v) = s(u,v-1) + I(u,v) \tag{5.2}$$

$$Ii(u,v) = Ii(u-1,v) + s(u,v) \tag{5.3}$$

En este algoritmo se utiliza una variante de AdaBoost que es usado para seleccionar un pequeño conjunto de características y entrenar al clasificador. Freund y Schapire sostienen que el error entrenamiento del clasificador se aproxima a cero de manera exponencial en función

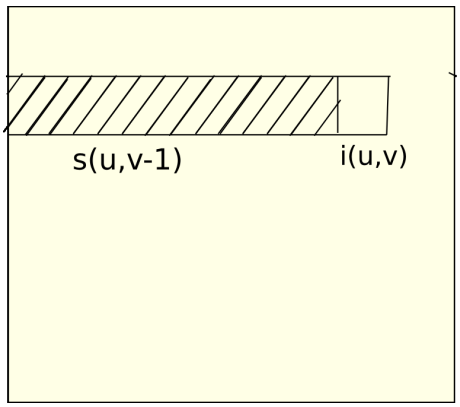


Figura 5.1: Acumulador fila en una imagen integral

del número de iteraciones. Logra buenos resultados rápidamente. Para cada característica, el aprendizaje es diseñado para determinar la función de clasificación según el umbral óptimo.

El algoritmo de aprendizaje está diseñado para seleccionar el rectángulo que mejor separe los ejemplos positivos y negativos. Para cada característica, el aprendiz determina el umbral óptimo para la función de clasificación

$$h_j(x) = \begin{cases} 1 & \text{si } p_j f_j(x) < p_j \theta_j \\ 0 & \text{otro caso} \end{cases} \tag{5.4}$$

Siendo $h_j(x)$ la función del clasificador de la característica f_i, p_j indica la dirección de la inecuación y θ_j el umbral.

Pseudocódigo del algoritmo de Viola & Jones

Algorithm 5 Algoritmo de Viola & Jones

Require: imágenes $(x_1, y_1), \ldots, (x_n, y_n)$ donde $y_i = 0, 1$ para ejemplos negativos y positivos respectivamente.

1: $\omega_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ para $y_i = 0, 1$ donde m y l es el número de negativos y positivos respectivamente
2: **for all** $t = 1, \ldots, T$ **do**
3: Se normalizan los pesos para obtener una función de distribución de probabilidad: $\omega_{t,i} \leftarrow \frac{\omega t,i}{\sum_{j=1}^{n} \omega t,j}$
4: Para cada característica f_j se entrena un clasificador h_j el cual está restringido a usar una sola característica, el error es evaluado con respecto a ω_t como $\epsilon_j = \sum_i \omega_i |h_j(x_i) - y_i$
5: Se elige el clasificador h_t que haya resultado el error ϵ_t menor.
6: Se actualizan los pesos. $\omega_{t+1,i} = \omega_{t,i} \beta_t^{1-e_i}$ donde $e_i = 0$ si x_i es clasificado correctamente y $e_i = 1$ en caso contario, y $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$
7: **end for**

El clasificador final resulta

$$h(x) = \begin{cases} 1 & \text{si } \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otro caso} \end{cases} \tag{5.5}$$

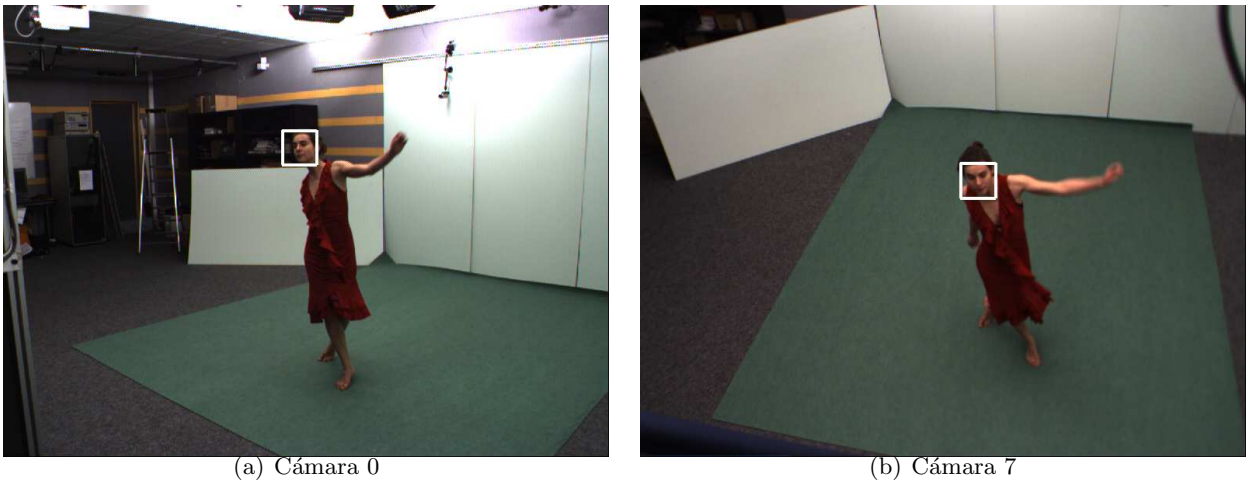
Donde $\alpha_t = log \frac{1}{\beta_t}$

Este algoritmo construye una cascada de clasificadores que logra reducir el tiempo de cómputo. Los clasificadores "boosted" pueden ser construidos para rechazar muchas de las subventanas

de casos negativos mientras detecta casí todos los casos positivos, el umbral de un clasificador "boosted" puede ajustarse para que la probabilidad de falsa alarma esté cercana a cero.

La forma general del proceso de detección es a partir de un árbol de decisión, al que se llama "cascada". Un resultado positivo del primer clasificador dispara la evaluación del segundo, otro resultado positivo del segundo clasificador dispara al tercero y así sucesivamente. Ante una salida negativa en cualquier punto se rechaza la subventana.

Las diferentes etapas en la cascada se construyen con entrenamiento de clasificadores usando "ADABoost", y posteriormente adjustando los umbrales para minimizar la probabilidad de falsa alarma. Por defecto los umbrales están diseñados para obtener una probabilidad de error baja y, en general, los umbrales bajos obtienen probabilidades de detección altas , pero probabilidades de falsa alarma también altas.



(a) Cámara 0 (b) Cámara 7

Figura 5.2: Reconocimiento de caras desde diferentes vistas

```
CvSeq* cvHaarDetectObjects(const CvArr*image,CvHaarClassifierCascade*cascade,
CvMemStorage* storage, double scale_factor=1.1, int min_neighbors=3,
int flags=0, CvSize min_size=cvSize(0,0));
/*Valor retornado (CvSeq*)-> lista dinámica donde se guardan las regiones
donde se detecta una cara.*/
/*const CvArr* image-> imágen fuente*/
/*CvHaarClassifierCascade* cascade -> Entrenamiento previo del clasificador
tipo Haar*/
/*CvMemStorage* storage -> Región de memoria donde se almacena el resultado*/
/*double scale_factor -> Modo de operación*/
/*int min_neighbors -> número mínimo de vecinos rectángulos que compone
un objeto. Todos los grupos de un número menor de rectángulos que
min_neighbors-1 son rechazados.*/
/*int flags=0*/
/*CvSize min_size -> tamaño mínimo de detección*/
```

Puede haber diferentes opciones de detección pero actualmente solo está implementado en las librerías "OpenCV" la opción de "CV_HAAR_DO_CANNY_PRUNING". Con esta opción la función usa el detector de bordes Canny para rechazar las regiones de las imágenes que contienen pocos o muchos bordes y no pueden contener el objeto deseado. Los valores de los umbrales particulares son calculados para la detección óptima de caras y en este caso la discriminación aumenta la velocidad del proceso.

Capítulo 6

Algoritmos de conectividad

Para discriminar los objetos de la escena y distinguirlos se ha optado por estudiar la conectividad tridimensional de la ocupación espacial. Se han implementado dos algoritmos diferentes, uno que estudia la conectividad de los vóxeles y otro la conectividad de los triángulos de la malla. En ambos algoritmos se estudia la conectividad de los puntos o triángulos sin tener en cuenta la contaminación de ruido que contienen las medidas provocada, a su vez por un cálculo de las siluetas corrompido por ruido o errores, o defectos propios del Visual Hull. Por esta razón estos algoritmos deberían complementarse con métodos estadísticos como los filtros de partículas o el algoritmo K-medias, donde se contempla una cierta distancia de margen, calculada por herramientas estadísticas para absorven los errores de ruido.

6.1. Algoritmo de conectividad de triángulos

El objetivo de este algoritmo consiste en agrupar el conjunto de los triángulos que forma la malla en conjuntos conexos que se han llamado clusters. Para ello se usa un array del tamaño del número de triángulos donde se guarda una etiqueta que indica el cluster al que pertenece cada triángulo.

Cada cluster está formado por un conjunto de triángulos conexos. Un triángulo es conexo a otro cuando dos de sus vértices coinciden. También se utiliza otro array para guardar el número de triángulos conectados a cada uno, siendo tres el número máximo. En este caso, al tratarse de una malla cerrada todos los triángulos deben estar conectados a otros tres.

Inicialmente ambos arrays se inicializan a cero, y se recorre el primer array que registra las etiquetas hasta que todos los triángulos poseen una distinta de cero. Para cada triángulo, se llama a una función recursiva que encuentra el siguiente triángulo conectado a él y en la propia función se vuelve a llamar recursivamente a si misma utilizándo el triángulo encontrado como argumento.

A continuación se detalla en un pseudocódigo el algoritmo utilizado:

Algorithm 6 Conectividad de triángulos

```
1: indices_clusters[] = zeros(1, num_triangulos);
2: num_conectados[] = zeros(1, num_triangulos);
3: i = 0;
4: num_clusters = 0;
5: while i = nextZero()! = −1 do
6:    num_clusters + +;
7:    recursive(i, 0)
8: end while
1: recursive(iTriangle, num_conectados_inicial)
2: indices_clusters[iTriangle] = num_clusters;
3: num_conectados[iTriangle] = num_conectados_inicial;
4: for j = nextZero() to j = num_triangles − 1 && num_conectados[iTriangle]<3 do
5:    if indices_clusters[j] == 0 then
6:       if Los triángulos j y iTriangle son conexos then
7:          num_conectados[iTriangle] + +;
8:          recursive(j, 1);
9:       end if
10:   end if
11: end for
```

El algoritmo es muy costoso computacionalmente, ya que un triángulo comprueba la conectividad con todos los triángulos restantes, por lo que para resoluciones altas este método en inviable. Véase en la figura 6.1 el resultado de la aplicación del algoritmo para una resolución de $\Delta xy = 60mm$ y $\Delta h = 60mm$.

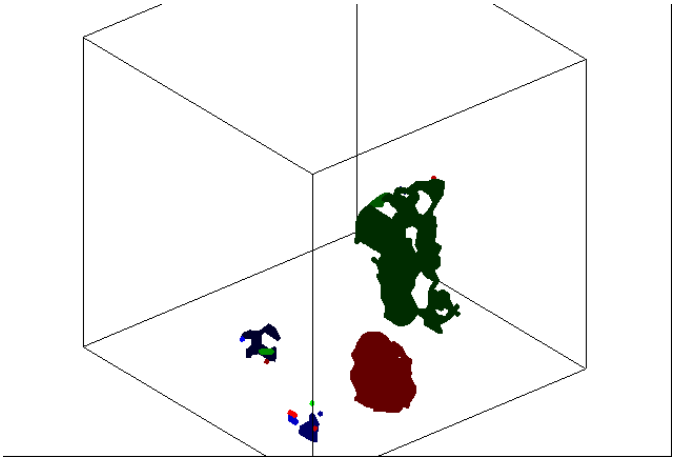


Figura 6.1: Conectividad de triángulos

6.2. Algoritmo de conectividad de cubos

Al igual que en el algoritmo anterior el objetivo es agrupar los vóxeles detectados como ocupados en conjuntos conexos llamados clusters.

Inicialmente los vóxeles detectados se etiquetan con un valor de menor a mayor.

Para hallar la conectividad se deben recorrer los vóxeles etiquetados de sentido positivo y analizar sus 8 vecinos en sentido positivo, tal cual lo hace el algoritmo Marching Cubes y

posteriormente en sentido negativo.

A continuación se expone un ejemplo con una estructura bidimensional:

Inicialmente se parte de la imagen con sus píxeles etiquetados 6.2:

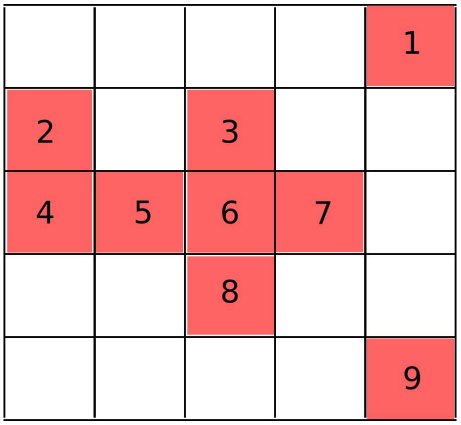


Figura 6.2: Píxeles etiquetados en una imagen bidimensional

El recorrido en la imagen bidimensional en sentido positiva se realiza de la manera siguiente:

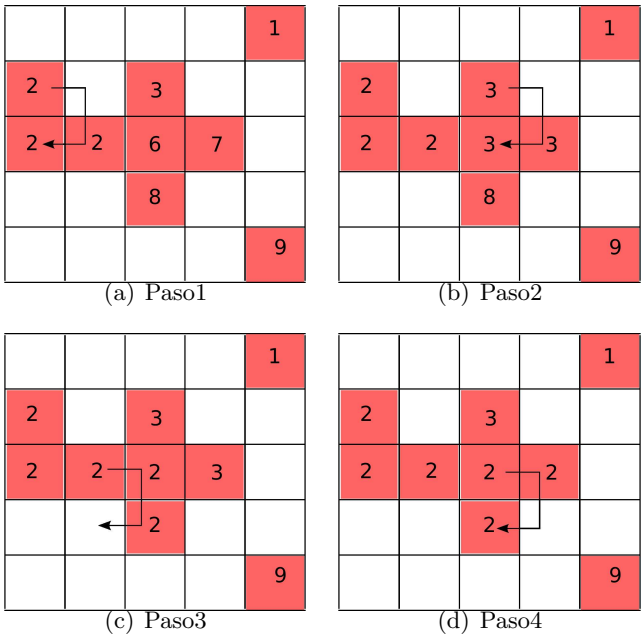


Figura 6.3: Recorrido en sentido positivo

El recorrido en la imagen bidimensional en sentido negativo se realiza de la manera siguiente :

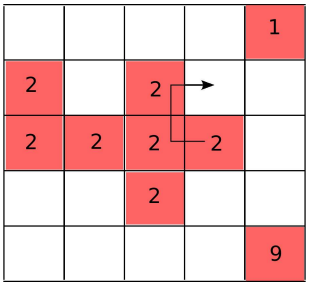


Figura 6.4: Recorrido en sentido negativo

Para extrapolarlo de dos dimensiones a tres dimensiones, se implementa en el código de Marching Cubes, y se analizan por cada punto ocupado las etiquetas de los vértices que corresponden a un cubo, siendo el punto de estudio, el punto de la esquina inferior derecha. Una vez etiquetados los puntos ocupados, se recorren en sentido inverso los puntos y en cada cubo, el punto de estudio es el vértice de la esquina superior de la izquierda.

Los arrays **a2fVertexOffset** y **a2fVertexOffsetNeg** sirven para indexar a partir del punto de estudio las etiquetas vecinas y escoger la mínima.

```
//a2fVertexOffset lists the positions, relative to vertex0,
//of each of the 8 vertices of a cube
static const GLfloat a2fVertexOffset[8][3] =
{
        {0.0, 0.0, 0.0},{1.0, 0.0, 0.0},{1.0, 1.0, 0.0},{0.0, 1.0, 0.0},
        {0.0, 0.0, 1.0},{1.0, 0.0, 1.0},{1.0, 1.0, 1.0},{0.0, 1.0, 1.0}
};

//a2fVertexOffset lists the positions relative to vertex0,
//of each of the 8 vertices of a cube
static const GLint a2fVertexOffsetNeg[8][3] =
{
        {0, 0, 0},{-1, 0, 0},{-1, -1, 0},{0, -1, 0},
        {0, 0, -1},{-1, 0, -1},{-1, -1, -1},{0, -1, -1}
};
```

Algorithm 7 Conectividad de cubos en sentido positivo

1: **for all** i=0:num_cubos_ocupados **do**
2: Hallar la mínima etiqueta de los vértices del cubo formado en sentido positivo si los vértices están ocupados.
3: *min_zona_conexa* = *min*(etiqueta de todos los vértices ocupados)
4: etiqueta de todos los vértices ocupados = *min_zona_conexa*
5: **end for**

Algorithm 8 Conectividad de cubos en sentido negativo

1: **for all** i=num_cubos_ocupados-1:0 **do**
2: Hallar la mínima etiqueta de los vértices del cubo formado en sentido negativo si los vértices están ocupados.
3: *min_zona_conexa* = *min*(etiqueta de todos los vértices ocupados)
4: etiqueta de todos los vértices ocupados = *min_zona_conexa*
5: **end for**

Posteriormente se realiza un filtro por tamaño de los clusters encontrados. En la figura 6.5 se muestra un ejemplo del resultado de aplicar el algoritmo:

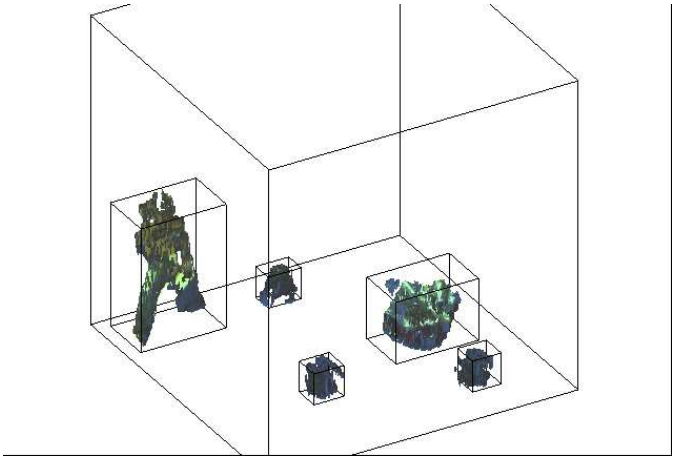


Figura 6.5: Conectividad de vóxeles

Capítulo 7

Implementación hardware del sistema

La implementación hardware del sistema está formada por las cámaras, los servidores de las cámaras y el cliente. Se trata de un sistema distribuido cuyo objetivo es repartir la carga computacional lo máximo posible entre los servidores y el cliente para minimizar el tiempo de cómputo por frame. En la figura 7.1 se representa la configuración del espacio inteligente:

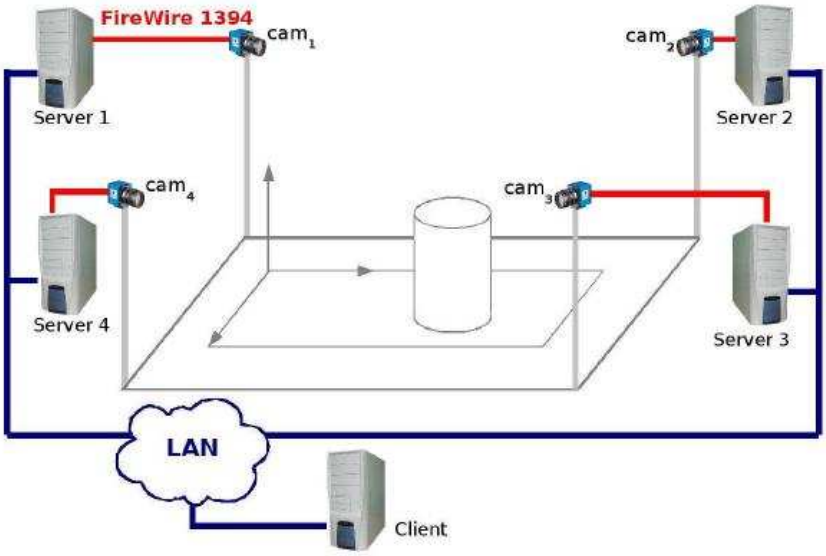


Figura 7.1: Implementación hardware del sistema

- **Elementos sensoriales:** Los elementos sensoriales de este sistema son cámaras de vídeo que trabajan en el espectro visible. Capturan frames en diferentes velocidades configurables: 7.5 fps, 15 fps o 30 fps. Para la aplicación de este proyecto la velocidad de frames adecuada es de 15 fps. La conexión con los servidores se realiza a través de un puerto "Firewire 1394".
- **Servidores:** Los computadores que trabajan como servidores están conectados a las cámaras por el puerto "Firewire 1394" y al computador cliente por una red local Ethernet.
- **Cliente:** Se conectan a la red local Ethernet.

Las características técnicas de los ordenadores son las siguientes:

- **Clientes:**

  Memoria RAM 1.9GB

  Procesador 0 : Intel(R) Core(TM)2 CPU T5600 @ 1.83GHz

  Procesador 1 : Intel(R) Core(TM)2 CPU T5600 @ 1.83GHz

- **Servidores:**

  Intel(R) Pentium(R) 4 CPU 3.00GHz

7.1. El estándar IEEE 1394

El estándar IEEE 1394 es conocido por los usuarios de Apple Finc como "FireWire". Se trata de un estándar multiplataforma que sirve para enviar datos serie a gran velocidad. El ancho de banda disponible en conexiones FireWire es de aproximadamente de 400 Mbps, por esta razón se suelen utilizar para transmitir audio y vídeo.

Las cámaras utilizadas en el laboratorio incorporan una interfaz FireWire permitiendo capturar vídeo digital y convertiéndolo directamente en datos transmitidos por dicha interfaz.

Capítulo 8

Implementación software del sistema

La implementación software del sistema es muy restrictiva, porque se busca la ejecución en tiempo real. En el sistema se ha distribuido la carga computacional entre el cliente y los servidores, bajo una arquitectura hardware mostrada en la figura 8.1

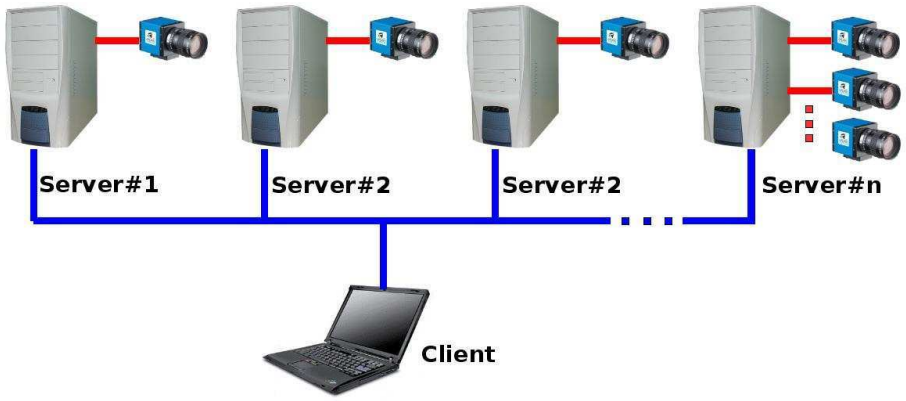


Figura 8.1: Arquitectura hardware de la estructura cliente-servidor

La arquitectura hardware y software responde a la de un sistema distribuido, donde los componenetes se encuentran conectados en red para coordinar sus acciones, mediante una comunicación de mensajes, para alcanzar el objetivo final.

8.1. Algoritmo de baja complejidad para el calculo del visual hull mediante múltiples homografías

En el proyecto se ha jugado con la resolución para llegar a un compromiso entre el tiempo de cómputo por cada frame y la calidad de la representación.

En tiempo real, en el espacio inteligente se ha implementado el sistema con las siguientes características:

- Resolución en altura $\Delta h = 60mm$
- Resolución en x e y $\Delta xy = 12mm$
- Longitud en x $L_x = 2880mm$

- Longitud en y $L_y = 3840mm$
- Altura del espacio $L_z = 2000mm$

Si la matriz inversa de homografía es:

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \tag{8.1}$$

En los servidores las imágenes a segmentar se submuestrean para obtener imágenes más pequeñas para que las operaciones a realizar se hagan con el menor tiempo posible:

La resolución de la cámara es de 640×480, pero las imágenes tratadas son de 320×240. Como la calibración de los parámetros intrínsecos se ha realizado con imágenes de 640×480 es necesario modificar la matriz de homografía utilizada:

$$H = \begin{pmatrix} \frac{h_{11}}{2} & \frac{h_{12}}{2} & h_{13} \\ \frac{h_{21}}{2} & \frac{h_{22}}{2} & h_{23} \\ \frac{h_{31}}{2} & \frac{h_{32}}{2} & h_{33} \end{pmatrix} \tag{8.2}$$

En los **servidores** se realiza el máximo número de tareas posibles relacionadas con el tratamiento individual de las imágenes capturadas de las cámararas de vídeo y la transmisión de la información obtenida hacia el cliente:

- **Operaciones de inicialización**

  Captura de imágenes de fondo En una primera inicialización los servidores capturan varias imágenes que son utilizadas para realizar un modelo estadístico del fondo que captura cada cámara, usado para segmentar las imágenes.

- **Operaciones realizadas por cada frame**

  Segmentación La segmentación dará como resultado una imagen en blanco y negro donde un píxel blanco se referirá a un objeto dentro de la escena y un píxel negro se referirá al fondo.

  Cálculo de los contornos A partir de las imágenes segmentadas se calculan los contornos de cada objeto, diferenciando entre contornos externos e internos. Los contornos internos corresponden a agujeros. Los contornos se crean en una lista dinámica organizada en dos niveles de jerarquía: las componentes externas e internas.

  Cálculo de las homografías Se calculan las proyecciones de cada punto de los contornos en el orden jerárquico adecuado en los planos paralelos al suelo y se guardan en un array de estructuras *c*vContorno":

```
#define EXTERNO 0
#define INTERNO 1

typedef struct contorno
{
        CvPoint *puntos;
        int num_puntos;      //Número de puntos del contorno
        unsigned char tipo; //0->Externo; 1->Interno
}cvContorno;
```

Transmisión de los contornos proyectados Se transmiten el array de estructuras de los contornos proyectados y el número de slice al que corresponde, a través de un socket de comunicación al cliente.

Transmisión de las imágenes a color Se transmiten las imágenes a color capturadas por la cámara hacia el cliente, a través de un socket UDP y una compresión JPEG.

El **cliente** se encarga de recibir la información de los servidores, a través de un socket de comunicaciones con cada uno, y fundirla para realizar la reconstrucción en tres dimensiones

- **Reconstrucción de los contornos proyectados** En el cliente se reconstruyen sobre imágenes los contornos proyectados por cada slice y cada cámara.
- **Intersección de las homografías por slice** Se realiza una operación lógica and sobre las imágenes de cada cámara por cada slice.
- **Construcción de la estructura de datos 3D** Se guardan los datos donde existe ocupación para posteriormente hallar la malla tridimensional, en los puntos donde hay ocupación y los puntos conexos a ellos. También se debe guardar una estructura de 3 dimensiones que va a servir al algoritmo de Marching Cubes para muestrear el espacio.
- **Cálculo de la malla tridimensional** Se aplica el algoritmo de MarchingCubes para obtener una malla tridimensional formada por triángulos.
- **Texturización de la malla y representación** Se calcula la mejor cámara para texturizar el triángulo. Se hallan las coordenadas bidimensionales en la imagén de la cámara correspondientes con las coordenadas tridimensionales de los vértices del triángulo para texturizarlo.

En la figura 8.2 se expone el diagrama de tareas software que realiza cada componente del sistema distribuido:

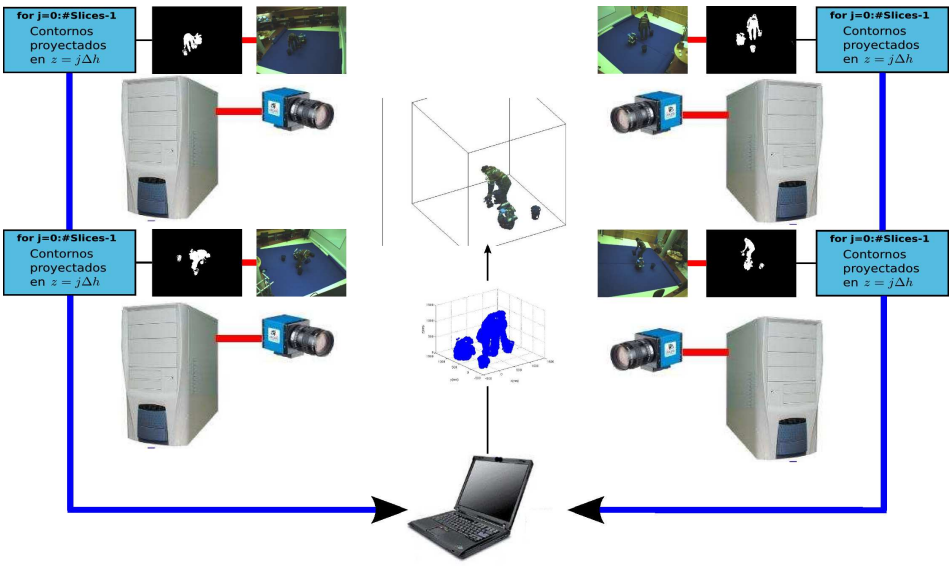


Figura 8.2: Diagrama de la estructura software cliente-servidor

8.2. Características técnicas del software

Las características técnicas del software son las siguientes:

- **Sistema Operativo**El proyecto se ha realizado con una plataforma linux:

  Ubuntu v8.04 (hardy)

  Núcleo Linux 2.6.24-24-generic

  GNOME 2.23.3

- **Tratamiento de imágenes** Se ha usado la biblioteca implementada en C/C++ OpenCV.
- **Representación de gráficos en tres dimensiones** Se ha utilizado la librería implementada en C/C++ OpenGL.

8.2.1. Descripción de la herramienta OpenCV

OpenCV (Open Computer Vision) es una biblioteca de uso libre implementada en C/C++, bajo licencia BSD. Esta orientada para aplicaciones de visión artificial en tiempo real. Realiza operaciones sobre imágenes y otras estructuras de datos definidas en la librería como por ejemplo matrices.

En este proyecto ha sido usada para:

- Captura imágenes
- Procesado de imágenes
- Operaciones matemáticas con matrices

8.2.2. Descripción de la herramienta OpenGL

OpenGL (Open Graphics Library) es una especificación estándar que define una interfaz de programación de aplicaciones (API) multilenguaje y multiplataforma para realizar aplicaciones que visualicen escenas tridimensionales a partir de elementos geométricos simples como puntos, líneas o polígonos.

En este proyecto ha sido usada para:

- Representar escenas en tres dimensiones

8.2.3. Descripción de la herramienta OpenGLUT

GLUT (OpenGL Utility Toolkit) es una biblioteca de utilidades para programación en OpenGL que ofrece funcionalidad para crear ventanas y manejar la interacción con el teclado y el ratón.

En este proyecto ha sido utilizada para crear la ventana de visualización y definir y crear opciones de visualización a través de teclado.

Capítulo 9

Resultados experimentales

En este capítulo se van a mostrar resultados temporales y de calidad de la reconstrucción obtenidos en dos sistemas diferentes: un sistema en el que una aplicación utiliza como fuente de imágenes una base de datos guardada en el propio computador y otro sistema distribuido en tiempo real donde existen servidores que proveen de imágenes al sistema y un cliente que realiza la reconstrucción.

9.1. Resultados en bases de datos de imágenes

Para realizar los experimentos en bases de datos se han usado dos secuencias de imágenes diferentes. Una propia del espacio inteligente de GEINTRA con 4 cámaras y otra secuencia propiedad de "INRIA" (Institud National de Recherche en Informatique et en Automatique) [48] con imágenes procedentes de 8 cámaras.

9.1.1. Análisis del cálculo de la ocupación 3D

En este apartado se estudian los tiempos de cómputo del cálculo de la ocupación tridimensional desplegando los datos temporales en las diferentes tareas necesarias.

En la tabla 9.1 se exponen los resultados para un determinado frame 9.1. En esta tabla se expresan los datos del tiempo de cómputo y el número de puntos ocupados que hay en función de la resolución utilizada:

Tabla 9.1: Tabla comparativa de número de puntos y tiempos de cómputo en función diferentes resoluciones

Δxy	Δz	Número de puntos	Tiempo de cómputo
10mm	10mm	245040	671.0 ms
10mm	30mm	81285	238.6 ms
10mm	60mm	40306	131.5 ms
30mm	30mm	9648	130.4 ms
30mm	60mm	4789	79.7 ms
60mm	60mm	1203	62.0 ms

Observando los resultados, la resolución en z, que perjudica directamente al número de planos paralelos al suelo considerados, es muy sensible al tiempo de cómputo, por lo que una

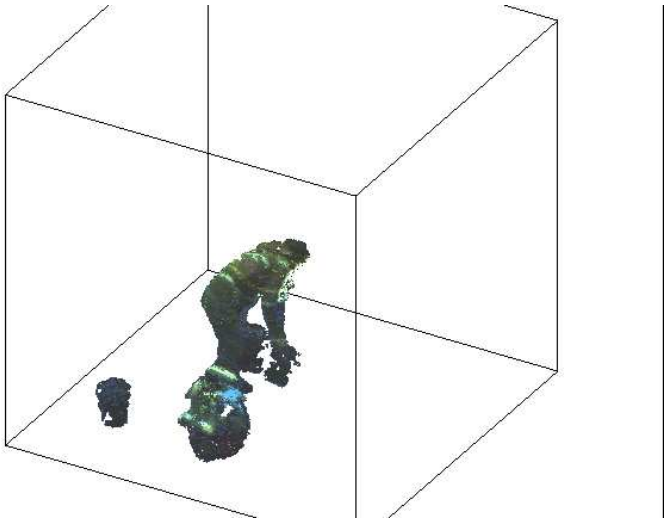


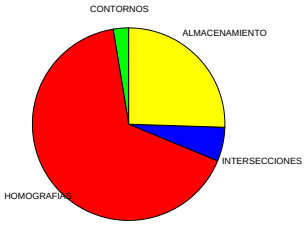
Figura 9.1: Frame de estudio

buena opción es fijar una resolución baja en z y ajustar la resoluxión en x e y para llegar a un compromiso entre calidad de la reconstrucción y el tiempo de cómputo.

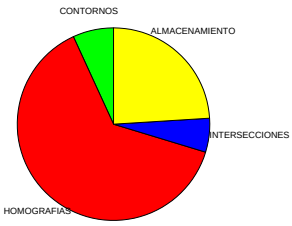
En las figuras 9.2 se muestran los porcentajes de los tiempos de ejecución por cada tarea a realizar en el cálculo de la ocupación en función de la resolución:

Las tareas en las que se ha dividido el estudio son:

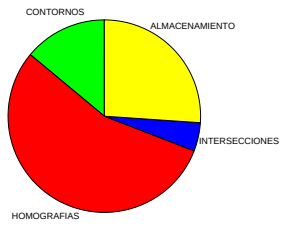
- **Contornos**: Se puede considerar una tarea independiente de la resolución ya que el porcentaje crece en las gráficas en la misma promoción que disminuye el tiempo de cómputo total. Es una tarea que se realiza por cada cámara y no depende de la resolución elegida.
- **Homografías**: Es la tarea más crítica en todos los casos, y depende fuertemente de la resolución en z, puesto que hay que hacer tantas homografías como planos paralelos del suelo se consideren por el número de cámaras que haya.
- **Intersecciones**: Es dependiente de la resolución en z, porque al igual que en el caso de las homografías hay que hacer tantas homografías como planos paralelos del suelo se consideren por el número de cámaras que haya, pero su coste computacional es mucho menor ya que se resumen en operaciones lógicas sencillas que las OpenCV realiza en bloque de 32 bits, minimizando el tiempo de cómputo. El tamaño de las imágenes por cada slice depende también de la resolución en x e y, pero experimentalmente no hay una gran dependencia a Δxy
- **Almacenamiento**: El almacenamiento disminuye notablemente cuando la resolución en todas las dimensiones es menos resctrictiva, ya que el número de puntos también disminuye.



(a) $\Delta xy = 10mm\ \Delta z = 10mm$

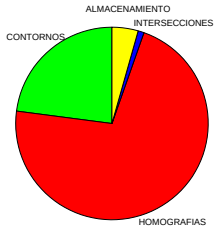


(b) $\Delta xy = 10mm\ \Delta z = 30mm$

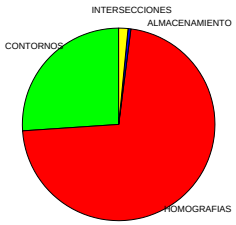


(c) $\Delta xy = 10mm\ \Delta z = 60mm$

(d) $\Delta xy = 30mm\ \Delta z = 30mm$



(e) $\Delta xy = 30mm\ \Delta z = 60mm$

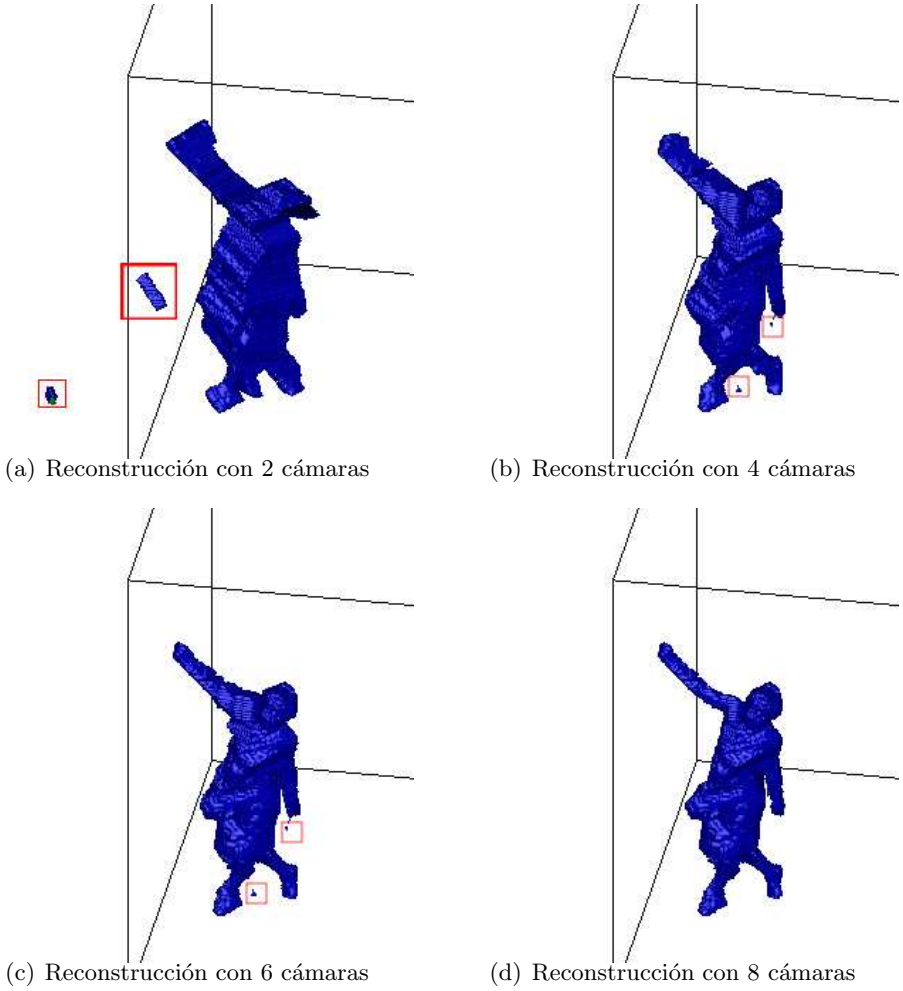


(f) $\Delta xy = 60mm\ \Delta z = 60mm$

Figura 9.2: Análisis temporal del cálculo de la ocupación

9.1.1.1. Efecto del número de cámaras en la ocupación 3D

En las figuras 9.3 se puede observar el efecto de utilizar un número diferente de cámaras.Se llega a la conclusión de que a más cámaras la ocupación es más restrictiva, y el Visual Hull se ajusta más al volumen deseado.



(a) Reconstrucción con 2 cámaras

(b) Reconstrucción con 4 cámaras

(c) Reconstrucción con 6 cámaras

(d) Reconstrucción con 8 cámaras

Figura 9.3: Reconstrucción con diferente número de cámaras

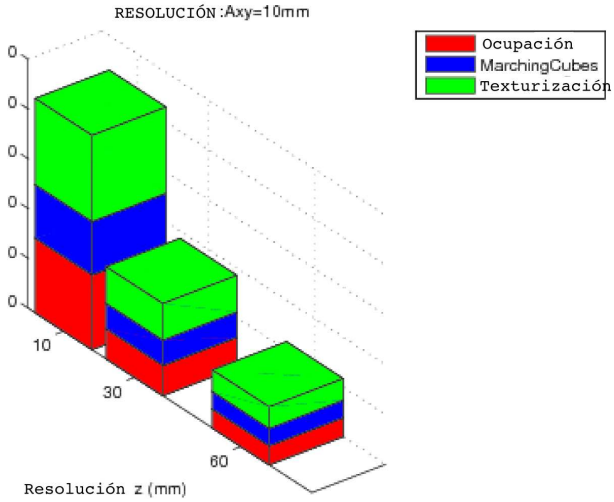
9.1.2. Análisis conjunto del cálculo de la ocupación y de la malla texturizada

En este apartado se estudian los tiempos de cómputo teniendo en cuenta, además del cálculo de la ocupación, el mallado y la texturización.

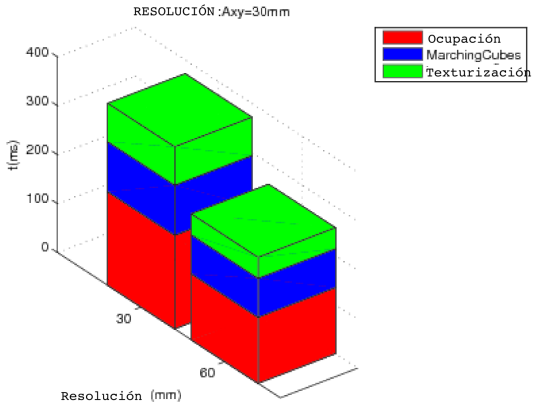
El número de triángulos es mayor que el número de puntos detectados por las características del algoritmo de MarchinCubes utilizado, donde por cada cubo donde haya puntos frontera puede haber de uno hasta cinco triángulos definiendo la superficie equiescalar.

Tabla 9.2: Tabla comparativa de número de puntos y triángulos y tiempos de cómputo en función diferentes resoluciones

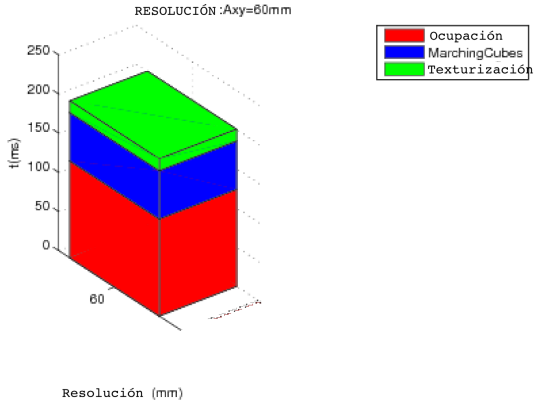
Δxy	Δz	Número de puntos	Número de triángulos	$t_{ocupacion}$	t_{total}
10mm	10mm	245040	295020	749.7 ms	2151.2 ms
10mm	30mm	81285	133504	302.9 ms	925.0 ms
10mm	60mm	40306	79998	190.8ms	582.6 ms
30mm	30mm	9648	30388	191.2 ms	371.8 ms
30mm	60mm	4789	16854	134.7 ms	257.1 ms
60mm	60mm	1203	5984	123.8 ms	200.6 ms



(a) $\Delta xy = 10mm$



(b) $\Delta xy = 30mm$



(c) $\Delta xy = 60mm$

Figura 9.4: Análisis temporal de la representación tridimensional

Las tareas en las que se ha dividido el estudio son:

- Ocupación
- MarchingCubes
- Texturizado

El algoritmo de MarchingCubes depende del número de puntos ocupados detectados y la texturizacion depende del número de triángulos resultantes. Para una resolución baja la tarea más crítica es la del cálculo de la ocupación, pero cuando la resolución mejora el algoritmo de MarchingCubes devuelve un número notablemente mayor de triángulos, por lo que la texturización se vuelve una tarea computacionalmente cara.

En las figuras 9.5 y 9.6 se exponen ejemplos de reconstrucción y representación fotorrealista con 4 cámaras.

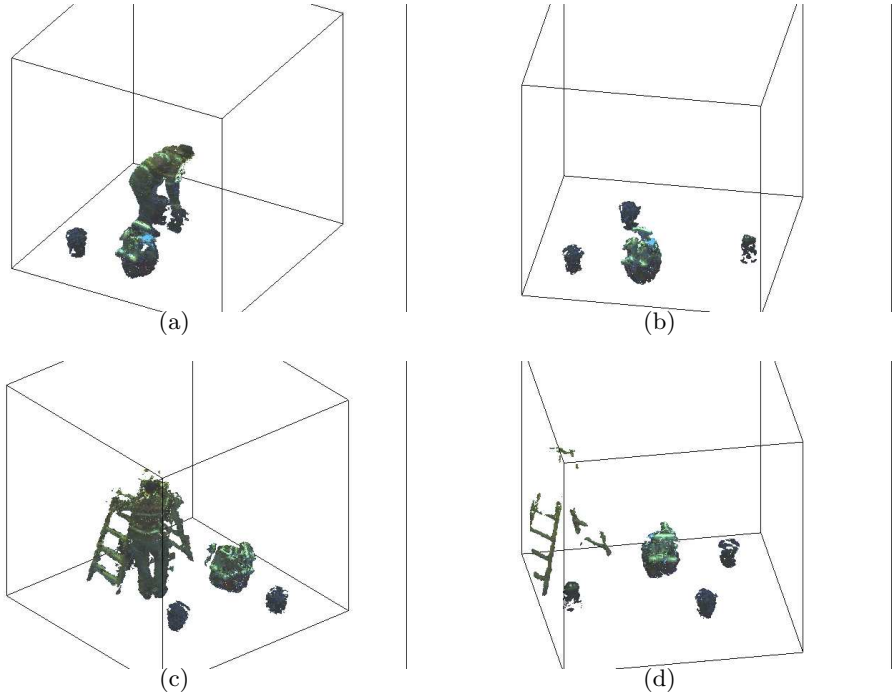
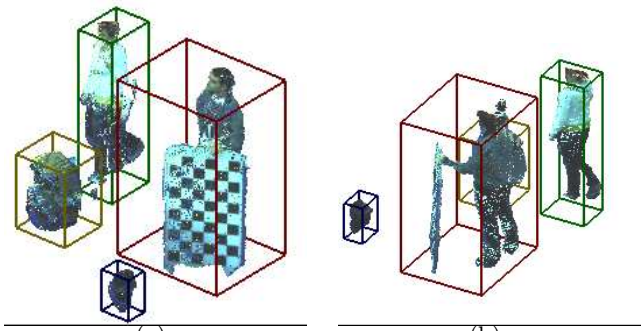


Figura 9.5: Frames



(a) (b)

Figura 9.6: Reconstrucción tridimensional de varios objetos

9.1.3. Análisis temporal del algoritmo de conectividad de cubos

A continuación se va a realizar una análisis temporal comparativo del algoritmo de Marching Cubes sencillo y el mismo algoritmo pero introduciendo el cálculo de la conectividad entre vóxeles.

Tabla 9.3: Tabla comparativa entre el algoritmo de MarchingCubes sin hallar la conectividad y el algoritmo aplicando conectividad de cubos

Δxy	Δz	Número de puntos	$t_{marching}$		$t_{representar}$	
			Solo ocupación	Conectividad	Solo ocupación	Conectividad
10mm	10mm	245040	864.8 ms	1265.7 ms	536.8 ms	1155.4 ms
10mm	30mm	81285	366.9 ms	517.0 ms	255.2 ms	420.9 ms
10mm	60mm	40306	216.0 ms	313.1 ms	175.8 ms	272.6
30mm	30mm	9648	78.7 ms	107.1 ms	86.9 ms	111.0 ms
30mm	60mm	4789	42.8 ms	59.2 ms	79.6 ms	94.4 ms
60mm	60mm	1203	15.1 ms	21.5 ms	61.7 ms	59.7 ms

En las figuras 9.7 y 9.8 se han representado los datos de la tabla 9.3 en función del volumen de un vóxel.

Como el algoritmo de conectividad de cubos está implementado, al mismo tiempo que se recorren los vóxeles para formar una malla, la diferencia temporal es exponencial, entre la ejecución del algoritmo sencillo y el algoritmo de conectividad de cubos. Se puede llegar a la conclusión de que el algoritmo crece exponencialmente con la resolución utilizada.

En la figura 9.8 se observa que tan solo para resoluciones altas la representación de los bounding-box influye bastante, pero para el resto de las resoluciones se incrementa el tiempo de ejecución un valor constante independiente de la resolución.

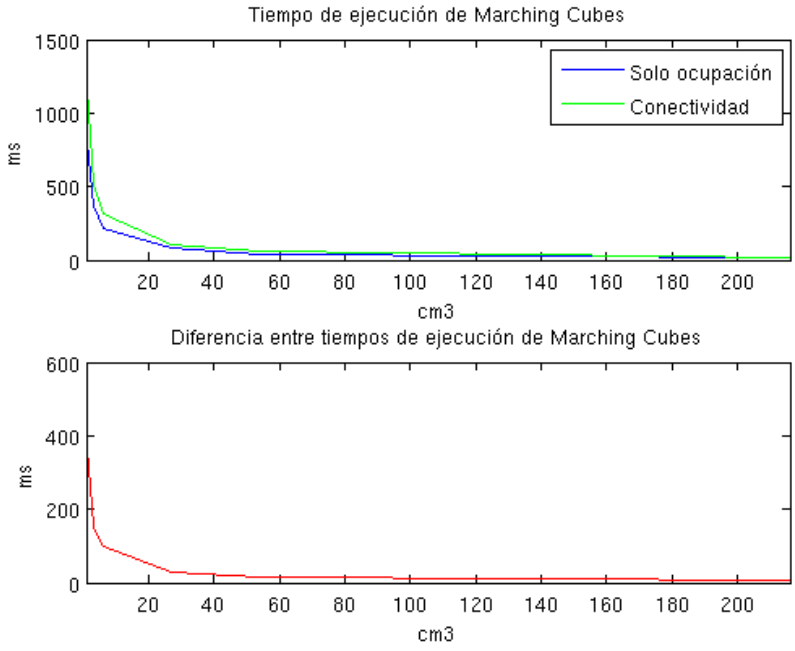


Figura 9.7: Comparación del tiempo de Marching entre el algoritmo sencillo y el algoritmo de conectividad de cubos

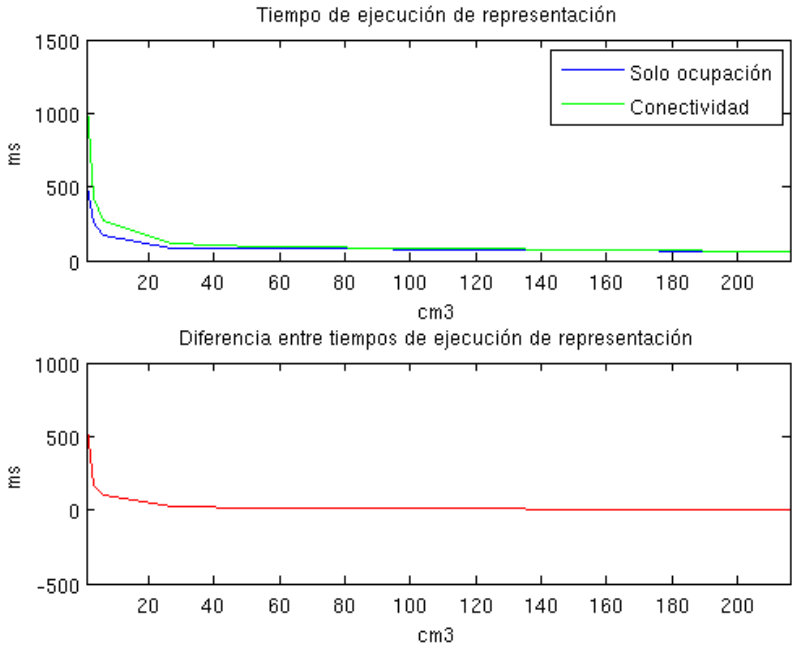


Figura 9.8: Comparación del tiempo de Representación entre el algoritmo sencillo y el algoritmo de conectividad de cubos

9.2. Resultados en un espacio inteligente

La aplicación de reconstrucción volumétrica se ha integrado dentro de otra aplicación que visualiza las imágenes de cada cámara y reconstruye un grid del plano del suelo donde se realiza un seguimiento de la trayectoria de los objetos mediante un filtro de partículas. En la figura 9.9 se observan varios ejemplos de la reconstrucción en tiempo real, realizado con 3 cámaras:

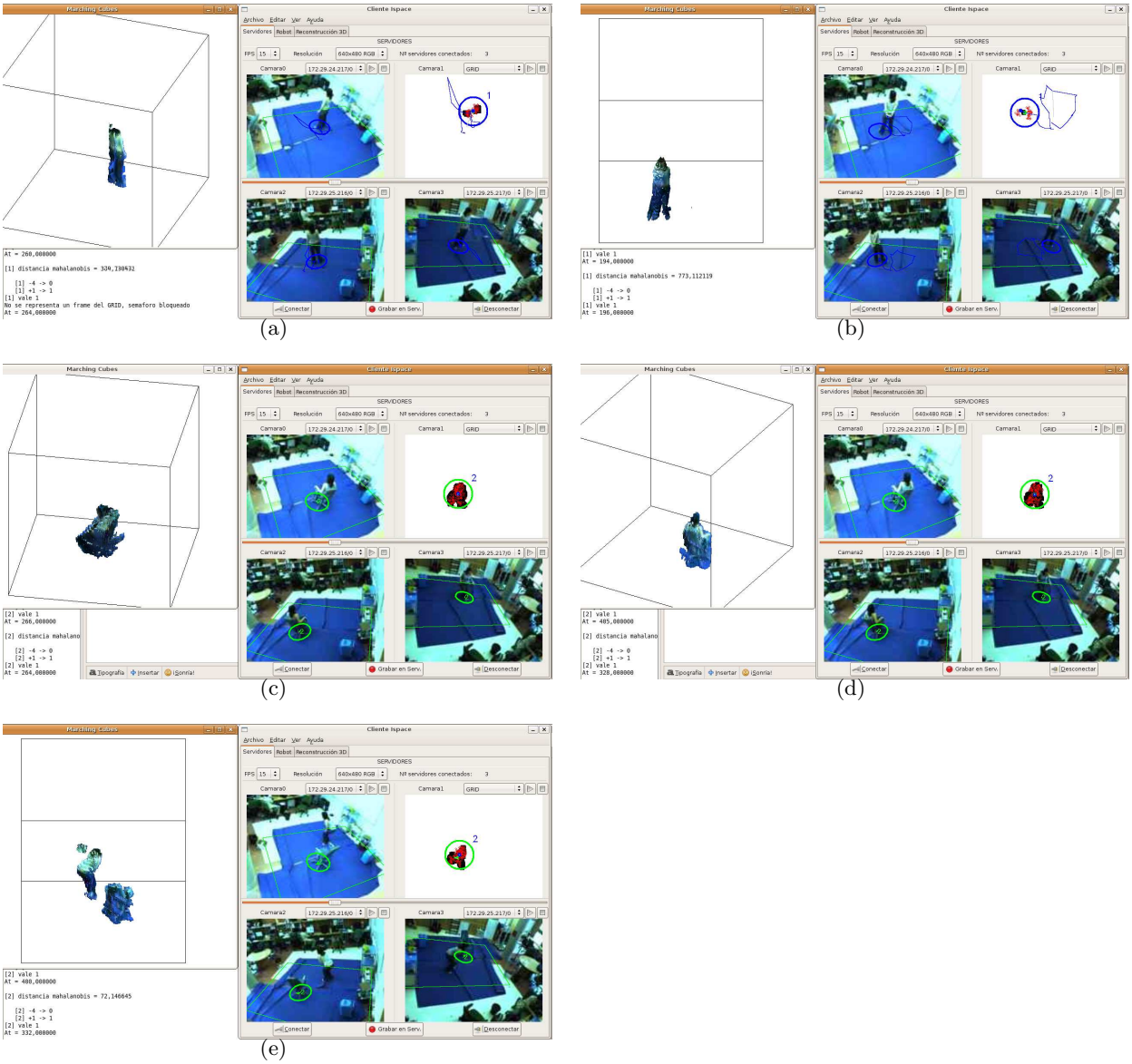


Figura 9.9: Reconstrucción tridimensional en tiempo real

La aplicación consigue ejecutarse a unos 9 frames por segundo aproximadamente.

La resolución usada es de $\Delta xy = 12mm$ y de $\Delta h = 60mm$

Las longitudes utilizadas son de $2280mm \times 3840mm \times 2000mm$

Servidor	Número de contornos	Tiempo de cómputo
Servidor1	14	46.181 ms
Servidor2	7	6.496 ms
Servidor3	3	5.516 ms

Tabla 9.4: Resultados temporales en el cálculo de la proyección de los contornos y su envío en cada servidor

9.2.1. Análisis temporal

Se han medido los tiempos de ejecución en el transcurso de 563 frames.

9.2.1.1. Análisis temporal en los servidores

Los tiempos que se han medido por cada frame son el tiempo de segmentación, el cálculo de los contornos, su proyección y su envio para 33 planos paralelos al suelo.

El tiempo de segmentación de cada frame es de $9{,}763ms \pm 0{,}909ms$ en cada servidor. Este es un tiempo constante y no depende del número de objetos o puntos ocupados:

El tiempo del cálculo de los contornos y la proyección de éstos y su envio depende del número de objetos y del volumen que ocupen. Poseen una gran variabilidad, ya que estos tiempos no solo dependen de su propio coste computacional, son dependientes de la red, de la transferencia de datos entre DMA's, reservas dinámicas de memoria, etc. Por esta razón se van a exponer los tiempos para el peor caso, el frame que sufre más demora en la secuencia de reconstrucción.

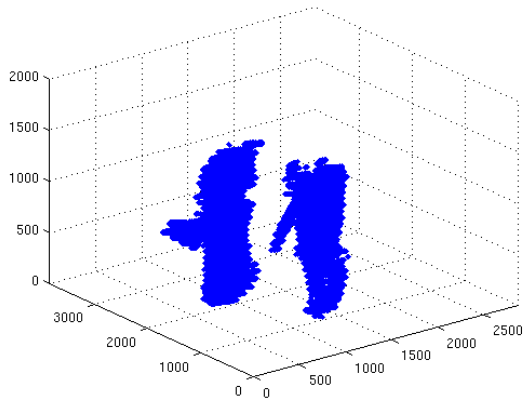


Figura 9.10: Ocupación tridimensional del frame de estudio

9.2.1.2. Análisis temporal en los clientes

Para el mismo frame del estudio anterior en los servidores, se ha realizado la obtención de los datos temporales de manera paralela. Se ha medido el tiempo que se tarda en calcular la ocupación tridimensional y la malla a través del algoritmo de Marching Cubes.

Tiempo de cómputo ocupación	Tiempo marching Cubes	Tiempo total
132.938 ms	191.826 ms	324.764

La frecuencia mínima de frames por segundo será 3 fps, en general, el sistema funciona mucho más rápido.

En las medidas tomadas en este experimento existe mucha variabilidad, por lo que las medidas no son muy fiables. En la figura 9.11 se muestran los valores de los tiempos de cómputo de la ocupación tridimensional y de la construcción de una malla y su representación, en función del número de puntos que se han detectado como ocupados. Se observa que, en valor medio, los tiempos de cómputo dependen del número de puntos.

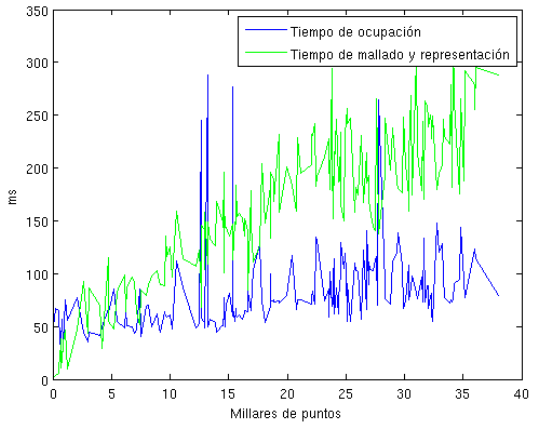


Figura 9.11: Resultado temporal en el cliente en función del número de puntos ocupados

En la figura 9.12 se representan los mismos valores anteriores, pero en el orden en que la secuencia de imágenes ha tenido lugar. También se muestran la variabilidad del número de contornos.

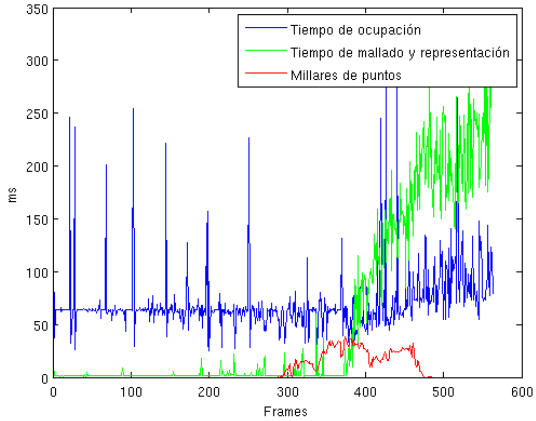


Figura 9.12: Resultados temporales en el cliente

Capítulo 10

Conclusiones y trabajos futuros

Para concluir este trabajo se puede manifestar que llegando a un compromiso entre calidad y tiempo de cómputo es posible realizar una reconstrucción tridimensional en tiempo real, capaz de ubicar en el Espacio Inteligente la ocupación de los objetos con bastante precisión métrica siendo el margen de error alrededor de 6 cm en la coordenada z y 1.2 cm en coordenadas x e y.

En relación con la reconstrucción tridimensional a partir de siluetas SFS (Shape-From-Silhouettes) se puede concluir que es muy dependiente de los errores de ruido producidos en el cálculo dichas siluetas. Es fundamental resolver los problemas de segmentación de fondo para obtener una reconstrucción tridimensional aceptable.

El número de cámaras y su colocación también es muy importante para ajustar el Visual Hull, al volumen real de ocupación. Esto se hace patente sobretodo cuando exiten muchos objetos en la escena donde unos se reflejan en las imágenes ocluidos en otros y es necesario que alguna de las cámaras, dada su colocación sea capaz de registrar la separación de los objetos. Una cámara en el techo del escenacio es muy útil para esta función.

La representación fotorrealista de la ocupación tridimensional, a partir de una malla, se ajusta de manera visual, a una representación mucho más realista que una representación voxélica, ya que la superficie formada por triángulos posee una aspecto más natural que la representación de un conjunto de cubos donde solo existen superficies perpendiculares. Por esta misma razón también es posible texturizar la malla de manera más fidedigna, ya que existe un rango mayor en las orientaciones posibles de las superficies a representar.

10.1. Líneas futuras de investigación

Los trabajos futuros, a partir de este proyecto, consistirían en mejoras en la construcción de la malla e integración de tareas inteligentes a partir de la reconstrucción 3-D.

- **Construcción de una malla no ambigua**: El algoritmo de Marching Cubes utilizado devuelve más triángulos que puntos ocupados, por lo que no es muy eficiente, y hay ciertos casos donde no se considera la conectividad si los vértices no son adyacentes, por el ejemplo en el caso P2b, mostrado en la figura 10.1, se considera que esos dos puntos no están conexos, existe un algoritmo de MarchingCubes modificado donde se resuelve este problema. Una posible mejora sería implementar una nueva versión del algoritmo.
- **Simplificación de la malla**: Dado un conjunto de triángulos conexos con la misma orientación la superficie ocupada se podría simplificar en un número menor de triángulos.

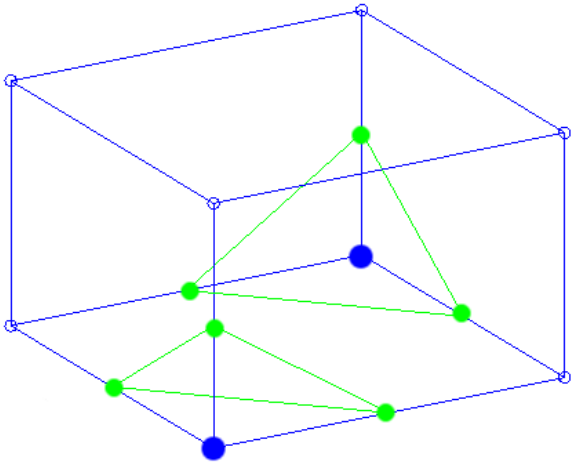
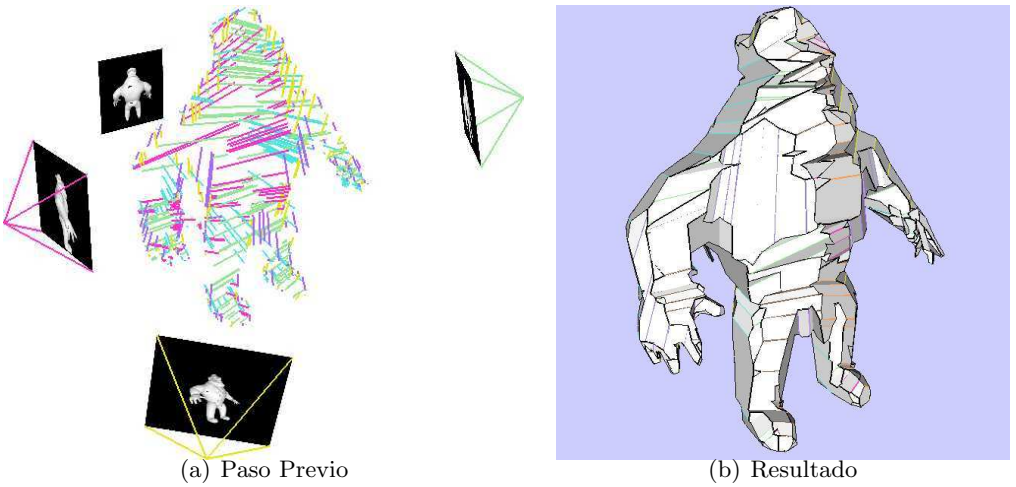


Figura 10.1: P2b

- **Sistema de reconocimiento de personas**: A partir del clusterizado de los objetos reconstruidos se podría detectar cuales de ellos son personas y definir un modelo de apariencia de la persona.
- **Sistema de cálculo del ángulo de la orientación de una cara**: Utilizando el algoritmo de Viola & Jones para detección de caras se podría ubicar la dirección de la cara de una persona estudiando el máximo de detección del algoritmo en un modelo tridimensional sencillo, como por ejemplo un cilindro, construido en una posible situación de la cabeza de la persona.
- **Cálculo del Visual Hull exacto**: Esta solución eliminaría la discretización del espacio en planos paralelos al suelo y calcularía la malla tridimensional directamente buscando la superficie de los conos visuales que intersectan hallando previamente las rectas proyectivas de los puntos de los contornos de las siluetas que formarían parte de dicha superficie. Para realizar estos cálculos es necesario hacer uso de la geometría epipolar. Esta solución está descrita en la tesis doctoral [49].



(a) Paso Previo (b) Resultado

Figura 10.2: Visual Hull exacto

Parte III

Manual de Usuario

Capítulo 11

Manual

11.1. Aplicación en tiempo real de la reconstrucción tridimensional

La aplicación en tiempo real está integrada junto a otra aplicación donde están implementados un filtro de partículas con seguimiento junto con un proceso de identificación de la ocupación bidimensional del plano $z = 0$. En el proyecto [50] se encuentra detallado el proceso de seguimiento llevado a cabo. Esta aplicación consta de 60 archivos fuente en la parte del cliente y de 15 en la del servidor, por lo que en el siguiente apartado se incluye una breve descripción de dichos archivos, antes de explicar las partes más importantes del programa y de la interfaz de usuario.

El sistema utilizado para las pruebas está estructurado con el concepto servidor-cliente. Existe concretamente un cliente y cinco servidores: uno por cámara (hay cuatro), y un robot. El número de servidores asociados a cámaras es fácilmente ampliable, puesto que la diferencia entre dos servidores radica únicamente en su dirección de red y en las matrices de parámetros de la cámara asociada. Como ya se ha mencionado en capítulos anteriores, los servidores se encargan de capturar, segmentar las imágenes, hallar los contornos de las siluetas, además de proyectarlos en múltiples planos paralelos y enviarlos. El cliente recoge esta información para componer la rejilla de Visual Hull final, que sirve como observación al sistema de seguimiento que existe implementado en el plano $z = 0$. El cliente además se encarga, entre otras tareas, de gestionar la comunicación y sincronización entre las cámaras y el robot y joystick, que se utiliza para controlar al robot de una forma cómoda.

11.1.1. Archivos fuente

En esta sección se repasa la funcionalidad de cada uno de los archivos fuente que constituyen tanto la aplicación cliente como la del servidor, de cara a facilitar su comprensión y posible modificación.

11.1.1.1. Aplicación cliente

Está formada por 60 archivos fuente distintos, que se pasan a describir a continuación. Los archivos .c son los siguientes, organizados por funcionalidad:

a) General

- **Main.c** Contiene la función main() del programa. Se encarga de inicializar el motor GTK (interfaz de usuario) y mantenerse a la espera de interrupciones software (callbacks), además de llamar a la función de inicialización de las variables y estado del sistema.
- **Inicializacion.c** Realiza las tareas de inicialización necesarias relativas al interfaz de usuario, hilos, temporizadores, odometría, joystick, robot, servidores, filtro de partículas y semáforos.
- **Timeouts.c** En él aparecen funciones que se ejecutan periódicamente, sobre todo de redibujo.
- **Utils.c** Conjunto de pequeñas funciones de distinta naturaleza.
- **Registro_eventos.c** Función para generar un log de eventos.
- **Ficheros.c** Funciones relativas a la apertura y el guardado de ficheros.

b) Interfaz de usuario

- **Interface.c** Este archivo es generado mediante el traductor de Glade a código C. Glade es el lenguaje utilizado para realizar la interfaz gráfica (GUI o Graphic User Interface).
- **Modifica_interfaz.c** Contiene una serie de funciones que se encargan de completar la interfaz gráfica y obtener información de la misma.
- **Edicion_servidores.c** Similar al anterior, pero relativo a la ventana de edición de servidores que aparece al seleccionar Edición >Servidores en el menú desplegable.
- **Odometria.c** Contiene funciones que modifican las partes del interfaz gráfico relativas a la odometría.
- **Support.c** Funciones GTK adicionales para la interfaz gráfica.
- **Color.c** Cambia el color a través de uno de los elementos que constituye la interfaz gráfica.
- **Callbacks.c** También generado por el traductor de Glade a C, en él están contenidas las definiciones de las distintas callbacks, que son llamadas al interactuar con el interfaz gráfico.

c) Red

- **Hilos.c** Contiene las funciones asociadas a los hilos que se ejecutan de forma paralela en la aplicación: uno asociado al robot, otro a la sincronización entre robot y cámara, otro al sondeo del joystick y un último que se encarga de recibir las imágenes desde los servidores. En la referencia [51] puede verse con más detalle la estructura de los mismos.
- **Captura.c** Funciones relativas a capturar las imágenes de los servidores.
- **Contorno2.c** Funciones para recibir y componer la ocupación tridimensional final y proveer el grid en el plano $z = 0$ al sistema de seguimiento.
- **Estados.c** Funciones que se encargan de controlar el estado de los servidores.
- **Red.c** Contiene las funciones que se encargan de realizar tareas de red y de sincronización entre el robot y servidores.
- **Grabacion.c** Contiene las funciones para envíar las órdenes de comenzar y parar la grabación de imágenes en los servidores.

- **Localiza.c** Función que se encarga de recibir la información relativa a las cámaras disponibles en ese instante.

d) Servidor

- **F_particulas.c** Se encarga periódicamente de obtener las medidas a partir de la imagen del grid, y de llamar a la función del XPFCP y proceso de identificación.
- **Joystick.c** Funciones para la utilización y configuración del joystick para controlar el robot en el espacio inteligente.
- **MiXpfcp.c** Este archivo fuente contiene tanto la definición de la función que implementa el seguidor bidimensional utilizado, el XPFCP, como la del proceso de identificación.
- **MisMath.c** En él aparecen funciones matemáticas relacionadas con el manejo de matrices, algoritmos de selección de partículas, y algunas funciones relativas a la modificación de las distintas imágenes para representar trayectorias, clases, textos, etc.
- **MiCluster.c** Funciones que implementan procesos de clasificación.
- **Robot.c** Contiene funciones de bajo nivel relacionadas con el robot: conexión y desconexión del mismo, modificación de su velocidad y recepción de la odometría.
- **d** Contiene funciones de alto nivel relacionadas con el robot. Como en robot.c, hay funciones para conectar, desconectar y modificar la velocidad, pero programadas a más alto nivel, a partir de las contenidas en robot.c. Además incluye una función para dibujar la odometría en la pestaña "Robot" del interfaz de usuario.

d) Representación tridimensional

- MarchingCubeslib.c Contiene la función del hilo que se encarga de hallar, texturizar y representar la malla que envuelve a la ocupación 3D.

En cuanto a los ficheros cabecera, el principal es ispace.h ya que se encarga de insertar el resto, además de las librerías necesarias para el funcionamiento del programa.

Los siguientes archivos de cabecera contienen las definiciones de constantes asociadas a su .c equivalente:

- **MisMath.h**
- **MiXpfcp.h**
- **Robocom.h**

Estos otros contienen las declaraciones de las funciones que aparecen en sus .c equivalentes:

- **Localiza.h**
- **Modifica_interfaz.h**
- **Odometria.h**
- **Red.h**

- **Registro_eventos.h**
- **Robot.h**
- **Robot_alto_nivel.h**
- **Support.h**
- **Timeouts.h**
- **Callbacks.h**
- **Captura.h**
- **Color.h**
- **Joystick.h**
- **Interface.h**
- **Inicializacion.h**
- **Hilos.h**
- **Grabacion.h**
- **Ficheros.h**
- **F_particulas.h**
- **Estados.h**
- **Edicion_servidores.h**
- **Contorno2.h**
- **MarchingCubeslib.h**

Por otra parte, en:

- **Constantes.h** Se definen constantes generales.
- **Variables.h** Se definen las variables globales.

11.1.1.2. Aplicación del servidor

La aplicación del servidor está formada por 15 archivos distintos: 9 de código fuente y 6 archivos de cabecera.

En cuanto a código fuente:

a) General

- **Main.c** Abre los sockets y se queda esperando conexiones. Es el fichero que contiene la función main.

- **Menu.c** Función para crear ciertos archivos y actuar frente a distintos comandos recibidos por los sockets.
- **Support.c** Funciones varias generadas con Glade.
- **Utils.c** Funciones de diversa naturaleza.

b) Red

- **Localiza.c** Función para localizar las cámaras disponibles y enviar al cliente esa información.
- **Red.c** Funciones relacionadas con la comunicación con el cliente.

c) Tratamiento de imágenes

- **Captura.c** Contiene la función de captura de imágenes.
- **Contorno2.c** Incluye diversas funciones relacionadas con la segmentación, el cálculo de los contornos y las proyecciones a los planos paralelos al suelo y su posterior envío.
- **Funciones.c** Incluye funciones de tratamiento de imágenes de caracter general, y para la realización de diferentes métodos del cálculo de una homografía. Es un archivo que sirve para simplificar el código de "Contorno2.c"

En cuanto a archivos de cabecera, estos contienen las declaraciones de las funciones que aparecen en sus .c equivalentes:

- **Callbacks.h**
- **Contorno2.h**
- **Interface.h**
- **Support.h**

El fichero ispace.h contiene tanto declaración de funciones como de constantes generales para todos los ficheros. Por último, en tablas.h se almacenan valores pre-calculados de la función logaritmo (utilizada para en la función de actualización de la media y desviación típica en imágenes), mejorando así el tiempo de ejecución del proceso de los servidores.

11.1.2. Utilización de la aplicación

Una vez que el cliente y los distintos servidores, robot incluido, están encendidos, el procedimiento a seguir para arrancar la aplicación completa se detalla a continuación.

11.1.2.1. Conexión de las cámaras

Las direcciones IP de los servidores asociados a las cámaras en el espacio inteligente utilizado en las pruebas son:

- ispace0 (172.29.24.216)
- ispace1 (172.29.24.217)

- ispace2 (172.29.24.217)
- ispace3 (172.29.25.217)

El proceso se facilita si se han almacenado previamente en /etc/hosts los nombres asociados a cada IP, ya que así, por ejemplo, basta con teclear ispace0 en lugar de 172.29.24.216.

1. Para poner a cada servidor a la espera de conexiones, tecleamos lo siguiente en consola, para acceder a los servidores por ssh iniciando una instancia del interfaz gráfico de los mismos:

   $ssh\ -X\ servidor@ispaceN$ con $N = 0, 1, 2, 3$

2. A continuación hay que introducir la contraseña correspondiente, que es servidor en todos los casos.

3. Para ejecutar el servidor hay que acceder al directorio correspondiente y ejecutar la aplicación (gladeservi0):

   $cd\ /home/ispace/raquel/servidor/$
   $./src3D/gladeservi0$

   El siguiente mensaje aparecerá por pantalla si todo va bien:

   Socket creado
   Socket enlazado
   Socket creado
   Socket enlazado
   Socket creado
   Socket enlazado

Es necesario repetir los tres pasos anteriores para cada servidor.

El número mínimo de cámaras para obtener una rejilla de ocupación con calidad suficiente como para hacer el seguimiento bidimensional es 3, aunque se recomienda utilizar las 4 para reducir en mayor medida las zonas de la escena no vistas por ninguna cámara.

11.1.2.2. Conexión del robot y del joystick

La conexión con el robot y con el joystick se encuentran explicadas en detalle en el proyecto [50]

11.1.2.3. Aplicación principal

La aplicación cliente se arranca escribiendo la siguiente línea en el directorio correspondiente del ordenador cliente:

./src/proyecto2

Aparecerá la siguiente ventana 11.1:

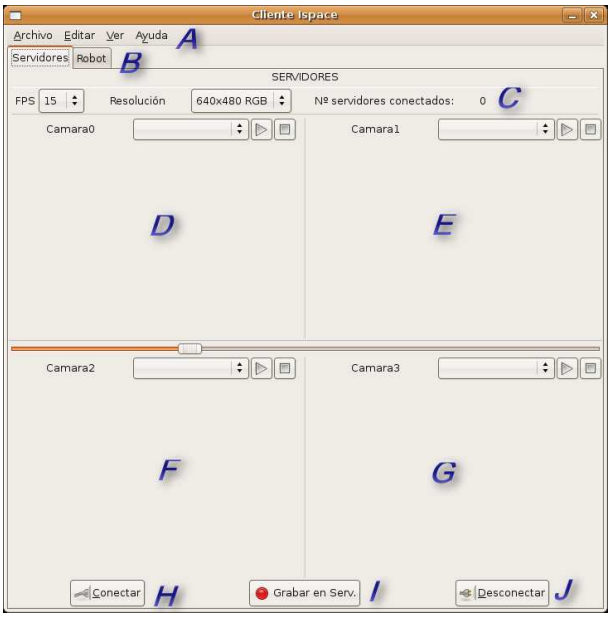


Figura 11.1: Ventana principal de la aplicación

Los elementos que la componen con sus opciones, se enumeran a continuación:

- A: Barra de menús.
  - Archivo
    - Guardar odometría
    - Salir
  - Editar
    - Servidores
    - Joystick
    - Robot
    - Odometría
      - Colores
      - Propiedades odometría
      - Cargar fondo odometría
  - Ver (Versión)
  - Ayuda
- B: Pestañas principales, para pasar de la de análisis de imágenes de servidores a la configuración del robot.
- C: Configuración del modo de captura de los servidores.

- D, E, F, G: Ventanas que muestran las imágenes recibidas por las distintas cámaras o la rejilla de ocupación (llamada GRID), a elegir mediante los menús desplegables situados en la parte superior de cada una, y pulsando el botón de reproducir.
- H: Botón para conectarse con los servidores.
- I: Botón para grabar imágenes capturadas y segmentadas en los servidores.
- J: Botón para desconectarse de los servidores.

Los pasos necesarios para conectar el cliente y los servidores se muestran a continuación. Primero es necesario configurarlos. En Edición >Servidores se especifica las direcciones de los servidores, como se muestra en la figura 11.2:

Figura 11.2: Ventana de edición de servidores

Es posible añadir direcciones personalizadas y guardar esa configuración como predeterminada, así como guardarla en un archivo para utilizarla posteriormente. Para completar la configuración de los servidores, se elegirá la velocidad de captura en FPS (Frames Per Second o Cuadros Por Segundo) y el modo (tamaño y espacio de color, C de la figura 11.1). Por último, se pulsa el botón "Conectar".

Si todo ha ido sin contratiempos aparecerá un mensaje de confirmación en una ventana emergente, así como una contestación por parte de los servidores en la consola de estos, como se muestra en la figura 11.3, en este caso una vez que se han conectado 3 servidores:

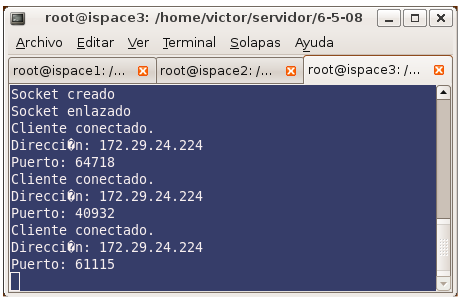


Figura 11.3: Mensajes de conexión de servidores

En este momento la ventana de OpenGL se lanza independientemente del resto de la aplicación, se representa la malla tridimensional texturizando con las imágenes que estén seleccionadas en las ventanas de visualización.

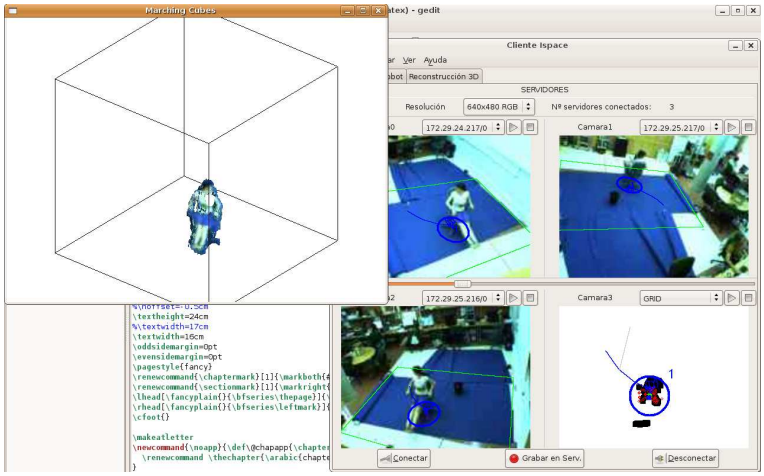


Figura 11.4: Ventana tridimensional integrada en la aplicación

Usando GLUT se ha facilitado una **interfaz de usuario** a través del teclado para modificar algunas características de la visualización tridimensional. A continuación se van a exponer las opciones que se han implementado:

- **Tecla Inicio**

  Gira la cámara virtual

- **Tecla** w

  Se visualizan los triángulos sin "relleno", es decir solo se representar sus aristas.

- **Tecla** $+$

  Se realiza un zoom para que la cámara virtual se "acerque"

- **Tecla** $-$

  Se realiza un zoom para que la cámara virtual se "aleje"

- **Tecla** $\rightarrow$

  La visualización se desplaza hacia la derecha.

- **Tecla ←**

  La visualización se desplaza hacia la izquierda.

- **Tecla ↑**

  La visualización se desplaza hacia arriba.

- **Tecla ↓**

  La visualización se desplaza hacia abajo.

Parte IV

Pliego de condiciones

Capítulo 12

Requisitos de Hardware

- Tarjeta gráfica nVidia GeForce 6600 GT.
- Cámaras ImagingSource DFK 21BF04 (4).
- Soportes omnidireccionales para las cámaras.
- Servidores, PC Intel Pentium D 3.00GHz, 4GB RAM
- Computador cliente, Mac Mini 1.83 Ghz (4).
- Conjunto de cables de red ethernet y FireWire 1394.

Capítulo 13

Requisitos de Software

- Linux Ubuntu v8.04 (hardy)
- Librerías OpenCV 1.0
- Librerías OpenGL

Parte V

Presupuesto

Capítulo 14

Presupueto

En esta parte del documento se va a exponer de manera aproximada el coste del proyecto, desglosándolo en costes de software, costes de equipos y costes de recursos humanos.

14.1. Coste del software

La realización del proyecto se ha llevado a cabo, en su mayoría, con software libre, a excepción de la utilización de Matlab. Matlab se ha utilizado como una herramienta para tareas previas de cálculo, como la calibración de las cámaras, pero no es necesario para ejecutar la aplicación final.

Función	Precio unitario	Cantidad	Subtotal
Sistema operativo: Ubuntu 8.04	0 €	5	0 €
Entorno de programación:Gedit	0 €	1	0 €
Elaboración del documento:Latex	0 €	1	0 €
Edición de gráficos vectoriales: Inkscape	0 €	1	0 €
Edición de imágenes: GIMP	0 €	1	0 €
Plataformas de programación: OpenCV	0 €	1	0 €
Plataformas de programación: OpenGL	0 €	1	0 €
Plataformas de programación: OpenGLUT	0 €	1	0 €
Matlab 7.0	500 €	1	500 €
Total			500 €

14.2. Coste de los equipos

Los equipos utilizados en el proyecto están formados por los ordenadores que realizan la función de servidores y el que realiza la función de cliente, además de las cámaras de vídeo junto con algunos complementos necesarios como los soportes.

Equipo	Precio unitario	Cantidad	Subtotal
Servidores, PC Intel Pentium D 3.00GHz, 4GB RAM	960 €	4	3840€
Cámara ImagingSource DFK 21BF04	340 €	4	1360 €
Soporte cámara omnidireccional	8.30 €	4	25,20 €
Cliente, Mac Mini 1.83 Ghz	480€	1	480€
Impresora HP Laserjet 2100	90€	1	90€
Total			5795,20€

14.3. Coste de recursos humanos

El número de las horas dedicadas a la ingeniería equivalen a seis meses de trabajo a jornada completa.

Función	Precio unitario por hora	Número de horas	Subtotal
Ingeniería	50 €	1056 horas	52800 €
Mecanografía	35 €	240 horas	8400 €
Total			61200 €

14.4. Coste total del proyecto

A continuación se ha contabilizado la suma total del presupuesto:

Concepto	Subtotal
Software	500 €
Equipos	5795,20 €
Recursos humanos	61200 €
Total	**67495,20 €**

Parte VI

Bibliografía

Bibliografía

[1] M. Weiser, "The computer for the 21st century," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 3, no. 3, pp. 3–11, 1999.

[2] ——, "Some computer science issues in ubiquitous computing," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 3, no. 3, p. 12, 1999.

[3] M. M. Waldrop, "PARC Builds a World Saturated With Computation," *Science*, vol. 261, no. 5128, pp. 1523–1524, 1993. [Online]. Available: http://www.sciencemag.org

[4] M. Coen *et al.*, "Design principles for intelligent environments," in *Proceedings of the National Conference On Artificial Intelligence*. John Wiley & SONS LTD, 1998, pp. 547–554.

[5] A. Pentland, "Smart rooms," *Scientific American*, vol. 274, no. 4, pp. 54–62, 1996.

[6] S. Se, D. Lowe, and J. Little, "Vision-based mobile robot localization and mapping using scale-invariant features," in *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA*, 2001.

[7] T. Sogo, H. Ishiguro, and T. Ishida, "Acquisition of qualitative spatial representation by visual observation," in *International Joint On Artificial Intelligence*, vol. 16. LAWRENCE ERLBAUM ASSOCIATES LTD, 1999, pp. 1054–1060.

[8] H. Hashimoto, J. Lee, and N. Ando, "Self-identification of distributed intelligent networked device in intelligent space," in *IEEE International Conference on Robotics and Automation, 2003. Proceedings. ICRA'03*, vol. 3, 2003.

[9] R. Zurawski, "Intelligent space and mobile robots," in *The Industrial Information Technology Handbook*. CRC Press, 2005, pp. 1–15.

[10] P. Steinhaus, M. Ehrenmann, and R. Dillmann, "MEPHISTO: A modular and extensible path planning system using observation," *Lecture notes in computer science*, pp. 361–375, 1999.

[11] J. Villadangos, J. Ureña, M. Mazo, A. Hernandez, F. Alvarez, J. García, C. Marziani, and D. Alonso, "Improvement of ultrasonic beacon-based local position system using multi-access techniques," in *Proc. of IEEE International Symposium on Intelligent Signal Processing, Faro, Portugal*, 2005.

[12] E. B, D. P, and I.Ñ, "Multiple camera calibration using point correspondences oriented to intelligent spaces," in *TELEC-04 International Conference*, 2004.

[13] D. Pizarro, E. Santiso, and M. Mazo, "Simultaneous localization and structure reconstruction of mobile robots with external cameras," in *International Symposium on Industrial Electronics ISIE05, June*, 2005.

[14] A. Hoover and B. Olsen, "Sensor network perception for mobile robotics," in *IEEE International Conference on Robotics and Automation, 2000. Proceedings. ICRA'00*, vol. 1, 2000.

[15] ——, "A real-time occupancy map from multiple video streams," in *1999 IEEE International Conference on Robotics and Automation, 1999. Proceedings*, vol. 3, 1999.

[16] B. Olsen and A. Hoover, "Calibrating a camera network using a domino grid," *Pattern Recognition*, vol. 34, no. 5, pp. 1105–1117, 2001.

[17] E. Kruse and F. Wahl, "Camera-based observation of obstacle motions to derive statistical-data for mobile robot motion planning," in *1998 IEEE International Conference on Robotics and Automation, 1998. Proceedings*, vol. 1, 1998.

[18] B. Baumgart, "Geometric modeling for computer vision." 1974.

[19] W. Martin and J. Aggarwal, "Volumetric descriptions of objects from multiple views," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 5, pp. 150–158, 1983.

[20] Y. Kim and J. Aggarwal, "Rectangular parallelepiped coding: A volumetric representation of three-dimensional objects," *Robotics and Automation, IEEE Journal of [legacy, pre-1988]*, vol. 2, no. 3, pp. 127–134, 1986.

[21] M. Potmesil, "Generating octree models of 3D objects from their silhouettes in a sequence of images," *Computer Vision, Graphics, and Image Processing*, vol. 40, no. 1, pp. 1–29, 1987.

[22] H.Ñoborio, S. Fukuda, and S. Arimoto, "Construction of the octree approximating a three-dimensional objectby using multiple views," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 6, pp. 769–782, 1988.

[23] N. Ahuja and J. Veenstra, "Generating octrees from object silhouettes in orthographic views," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 2, pp. 137–149, 1989.

[24] K. Shanmukh and A. Pujari, "Volume intersection with optimal set of directions," *Pattern recognition letters*, vol. 12, no. 3, pp. 165–170, 1991.

[25] R. Szeliski, "Rapid octree construction from image sequences," *CVGIP: Image Understanding*, vol. 58, no. 1, pp. 23–32, 1993.

[26] A. Laurentini, "The visual hull: A new tool for contour-based image understanding," in *Proc. 7th Scandinavian Conference on Image Analysis*, 1991, pp. 993–1002.

[27] ——, "The visual hull concept for silhouette-based image understanding," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 2, pp. 150–162, 1994.

[28] ——, "How far 3D shapes can be understood from 2D silhouettes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 2, pp. 188–195, 1995.

[29] A. Laurentini and D. e Inf, "The visual hull of curved objects," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 1, 1999.

[30] M. Okutomi and T. Kanade, "A multiple-baseline stereo," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 4, pp. 353–363, 1993.

[31] K. Kutulakos and S. Seitz, "A theory of shape by space carving," *International Journal of Computer Vision*, vol. 38, no. 3, pp. 199–218, 2000.

[32] S. Moezzi, L. Tai, and P. Gerard, "Virtual view generation for 3d digital video," *IEEE multimedia*, vol. 4, no. 1, pp. 18–26, 1997.

[33] I. Kakadiaris and D. Metaxas, "Three-dimensional human body model acquisition from multiple views," *International Journal of Computer Vision*, vol. 30, no. 3, pp. 191–218, 1998.

[34] Q. Delamarre and O. Faugeras, "3D articulated models and multi-view tracking with silhouettes," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 2, 1999.

[35] A. Bottino and A. Laurentini, "Non-intrusive silhouette based motion capture," in *Proceedings of 4th World Multiconference on Systemics, Cybernetics and Informatics*, 2000, pp. 23–26.

[36] C. Buehler, W. Matusik, L. McMillan, and S. Gortler, "Creating and rendering image-based visual hulls," 1999.

[37] K. Cheung, S. Baker, and T. Kanade, "Shape-from-silhouette across time part i: Theory and algorithms," *International Journal of Computer Vision*, vol. 62, no. 3, pp. 221–247, 2005.

[38] W. Lorensen and H. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. ACM New York, NY, USA, 1987, pp. 163–169.

[39] Y. Kenmochi, K. Kotani, and A. Imiya, "Marching cubes method with connectivity," in *Image Processing, 1999. ICIP 99. Proceedings. 1999 International Conference on*, vol. 4, 1999.

[40] J. Rodriguez, "Técnicas de reconstrucción volumétrica a partir de múltiples cámaras. aplicación en espacios inteligentes," Trabajo final de carrera, Departamento de Electrónica, Universidad de Alcalá, Sept. 2008.

[41] Z. Zhang, "Flexible camera calibration by viewing a plane from unknownorientations," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 1, 1999.

[42] J. Heikkila and O. Silven, "A four-step camera calibration procedure with implicit image-correction," in *1997 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1997. Proceedings.*, 1997, pp. 1106–1112.

[43] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[44] J. Franco, "Modélisation tridimensionnelle à partir de silhouettes," 2005.

[45] G. Finlayson, S. Hordley, and M. Drew, "Removing shadows from images," *Lecture Notes in Computer Science*, pp. 823–836, 2002.

[46] D. Pizarro and A. Bartoli, "Shadow resistant direct image registration," *Lecture Notes in Computer Science*, vol. 4522, p. 928, 2007.

[47] K. Okuma, A. Taleghani, N. De Freitas, J. Little, and D. Lowe, "A boosted particle filter: Multitarget detection and tracking," *Lecture Notes in Computer Science*, pp. 28–39, 2004.

[48] [Online]. Available: www.inria.fr

[49] J. Franco, "Modélisation tridimensionnelle à partir de silhouettes," 2005.

[50] Á. Marcos, "Seguimiento de múltiples objetos en espacios inteligentes," Tesis de Máster, Departamento de Electrónica, Universidad de Alcalá, July 2009.

[51] V. Espejo, "Sistema de detección de obstáculos y robots mediante múltiples cámaras en espacios inteligentes," Trabajo final de carrera, Departamento de Electrónica, Universidad de Alcalá, Dec. 2008.