Plataforma de adquisición de imágenes en Escenarios Virtuales para uso en Sistemas de Visión

Memòria del projecte de final de carrera corresponent als estudis d'Enginyeria Superior en Informàtica presentat per Marc Fenés Vacas i dirigit per Dr. Antonio Lopez.

Bellaterra, 15 de Septiembre de 2009

CERTIFICACIÓ DE DIRECCIÓ

El sotasignat, Dr. Antonio López , Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en Marc Fenés Vacas.

I per tal que consti firma la present.

Bellaterra, 15 de Septiembre de 2009

Signat:

Resumen

En aquest projecte presentem un metode per generar bases de imatges de vianants, requerides per a l'entrenament o validació de sistemes d'aprenentatje basats en exemples, en un entorn virtual. S'ha desenvolupat una plataforma que permet simular una navegació d'una càmara en una escena virtual i recuperar el fluxe de video amb el seu groundtruth. Amb l'ús d'aquesta plataforma es suprimeix el procés d'anotació, necesari per obtenir el groundtruth en entorns reals, i es redueixen els costos al treballar en un entorn virtual.

En este proyecto presentamos un método para generar bases de imágenes de peatones, requeridas para el entrenamiento o validación de sistemas de aprendizaje basados en ejemplos, en un entorno virtual. Se ha desarrollado una plataforma que permite simular una navegación de una cámara en una escena virtual y recuperar el flujo de video junto con su groundtruth. Con el uso de esta plataforma se suprime el proceso de anotación, necesario para obtener el groundtruth en entornos reales, y se reducen los costes al trabajar en un entorno virtual.

In this project we show a method to generate pedestrian image bases, required for training or validation of example-based learning systems, in a virtual environment. We have developed a platform able to simulate a camera navigation and to retrieve video stream with its groundtruth, into a virtual scene. Using this platform annotation process, necessary to obtain groundtruth in real environments, is supressed and as we are working into a virtual environment costs are reduced.

A mi niña, por tener tanta paciencia...

Índice general

Agradecimientos

Quiero agradecer a mi director, Dr. Antonio López, por su supervisión de este proyecto y ayuda, en especial, para la redacción de esta memoria.
Finalmente, quiero también agradecer a David Vázquez y a Javier Marín por su apoyo y atención constantes durante el transcurso de este proyecto y la ayuda ofrecida, tanto en la generación de escenarios como en la realización de distintas pruebas con los sistemas de visión.

Capítulo 1

Introdución

1.1. Motivación

Las técnicas de visión por computador tienen como objetivo extraer un conjunto de propiedades del entorno a través de imágenes. Mediante estas propiedades se pueden construir sistemas inteligentes que realicen determinadas acciones respondiendo a eventos del entorno. Algunas aplicaciones como procesos automáticos de inspección o de control de calidad, sistemas de análisis de imágenes médicas o topográficas, sistemas interactivos o sistemas de detección (video vigilancia), son el resultado de aplicar distintas técnicas de visión por computador.

En visión por computador, los sistemas de detección se basan en el reconocimiento de objetos que se suele afrontar con aprendizaje basado en ejemplos [15]. Existen dos variantes; el aprendizaje supervisado (los ejemplos están etiquetados, por lo que conocemos la clase real del elemento) y, aunque menos común en este tipo de problemas, el no supervisado (los datos no están etiquetados). En estos tipos de aprendizaje, se utilizan un conjunto de ejemplos (llamado conjunto de entrenamiento) para inferir un clasificador. A este podremos facilitarle nuevos ejemplos, y basándose en el parecido con los del conjunto de entrenamiento, podrá tomar una decisión y determinar su clase. En la figura 1.1 se muestra un esquema de este proceso. Debido a que estos sistemas tienen que ser capaces de reconocer ciertos objetos, independientemente de la iluminación, tamaño, forma y color, se necesitan un gran número de muestras para entrenarlos y validarlos debidamente. Este hecho presenta dos problemas. En primer lugar la dificultad de obtener un gran número de muestras variadas, y en segundo lugar, la necesidad de un posterior tedioso proceso de anotación y etiquetado de estas. A través de este último se obtiene el *groundtruth* (clase real de cada elemento), usado en el entrenamiento (con aprendizaje supervisado) y en la validación.

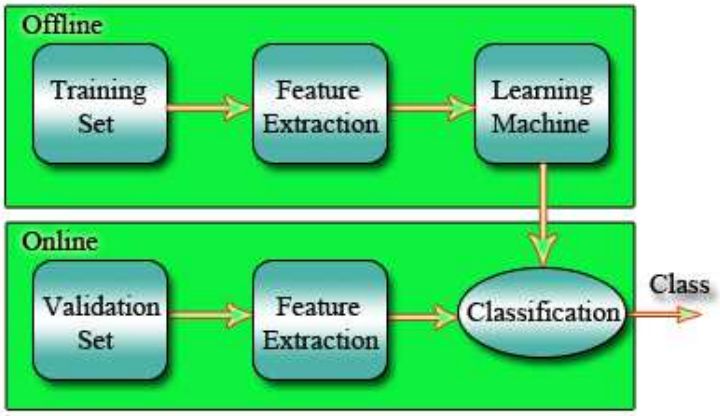

Figura 1.1: Aprendizaje basado en ejemplos.

Afortunadamente, la calidad de imagen y la sensación de realidad que se encuentra hoy en día en el mundo de los gráficos por computador se asemeja mucho al mundo real. Entonces nos preguntamos lo siguiente. ¿Podríamos usar muestras de objetos obtenidas en un entorno virtual para comprobar la robustez de algoritmos de visión? ¿Podríamos entrenar un sistema de detección en un entorno virtual y extrapolarlo con éxito a un entorno real? Las ventajas serian varias. Fácilmente podríamos obtener una gran variedad de modelos o modificar a nuestro antojo otras características, como la iluminación, el ruido, la posición de las cámaras, etc. Dispondríamos de un entorno de experimentación controlada.

Hace relativamente poco tiempo han aparecido los sistemas de conducción inteligente o ADAS (Advanced Driver Assistance Systems). Estos sistemas se implantan en vehículos y mediante distintos sensores y dispositivos de captura, son capaces de analizar el entorno y detectar situaciones de riesgo. Su objetivo es ofrecer información del entorno del vehículo en tiempo real y ofrecer asistencia al conductor (detección de vehículos o peatones, información vial, etc) o llegar a tomar el control en situaciones de peligro (frenada de emergencia en colisiones inminentes).

El fin de este proyecto es el diseño e implementación de una plataforma para la obtención de imágenes en escenarios virtuales, dirigidos al entrenamiento/validación de sistemas de visión por computador para la detección de peatones.

1.2. Objetivos del proyecto

El desarrollo de este proyecto se basará en un proceso iterativo. La idea inicial es llegar a ejecutar un ciclo completo, implementando un conjunto de funcionalidades básicas, y en función del tiempo completar adicionales en iteraciones posteriores. Este conjunto básico implica abordar las tareas necesarias para poder generar videos, obtener el groundtruth y generar resultados mediante el uso de un sistema de visión. A continuación se listan estas tareas.

- Obtención de conocimientos básicos en la edición de escenarios.
- Aprendizaje del uso de la herramienta principal utilizada en este proyecto (ObjectVideo Virtual Video Tool), así como exploración de todas sus funcionalidades.
- Implementación de una plataforma para la obtención de videos simulando un desplazamiento en coche (*travelling*) en escenarios urbanos.
- Desarrollo de aplicación para la obtención del groundtruth.
- Realización de pruebas de sistemas de visión desarrollados en el Centro de Visión por Computador (CVC) de la Universidad Autónoma de Barcelona (UAB) con los videos obtenidos de escenarios virtuales. Realizar comparación del rendimiento de los algoritmos de detección usando videos reales y sintéticos.

1.3. Planificación

A continuación mostraremos la planificación que se ha seguido para el desarrollo de este proyecto. En el diagrama de Gantt (fig. 1.2) podemos ver las distintas fases del proyecto, así como el inicio y fin de estas.

La planificación prevista fue larga debido a que este proyecto se ha realizado

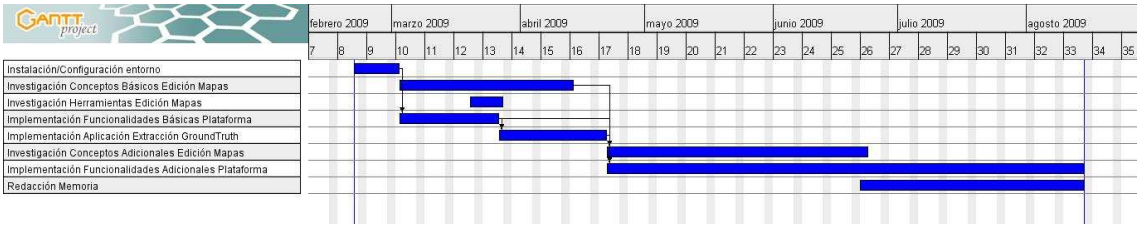

Figura 1.2: Planificación Proyecto.

en conjunción a la vida académica y laboral, por lo que la dedicación diaria no

ha sido completa. Se dedicó una gran cantidad de tiempo para la investigación de conceptos y posibilidades de la edición de escenarios para poder realizar una estimación del alcance y posibles usos que se le podría dar a este proyecto. Esta segunda etapa de investigación se realizaría en paralelo a la implementación y desarrollo de las funcionalidades adicionales de la plataforma, dando mayor prioridad a esta última actividad.

1.4. Organización de la memoria

En el capítulo 2 haremos una introducción al contexto de este proyecto, mostrando aplicaciones de visión por computador destinadas a mejorar la seguridad y eficiencia en los vehículos. Definiremos algunos de los tipos de sistemas de detección que existen, y especialmente abordaremos el problema de la detección de peatones. Comentaremos el proceso de generación de las bases de imágenes, necesarias en este tipo de sistemas, y mostraremos su analogía en un entorno virtual, que es el ámbito en el que se define este proyecto. En este capítulo también se tratará el tema de la visión estereoscópica. Se detallará en qué consiste, cuáles son sus problemas asociados y cuáles se solucionan mediante el uso de esta plataforma, y se comentará las distintas distribuciones de cámaras para este tipo de sistemas. Finalmente, aunque no se ha podido implementar en el marco de este proyecto, se propondrá un método para la generación del groundtruth de este tipo de sistemas en un entorno virtual.

En el capítulo 3 nos centraremos en la plataforma desarrollada. Se expondran los motivos y razones por los que se realizó este proyecto y se comentará la selección de las herramientas principales, mencionando cuales son y sus requisitos. Posteriormente se listarán las funcionalidades de la plataforma y se presentarán los distintos componentes que la forman, así como su arquitectura interna. Para terminar se mostrará el flujo de ejecución de algunos casos de uso principales y se mostrará el proceso completo para generar una base de imágenes mediante la plataforma.

En el capítulo 4 se mostrarán los resultados obtenidos mediante el uso de la plataforma desarrollada. Se hará una introducción al sistema de detección usado y se mostrarán los resultados de validar imágenes reales habiendo entrenando con imágenes sintéticas. Se compararán estos resultados con los obtenidos al entrenar con imágenes reales y se reflexionará sobre ello.

En el capítulo 5 se detallará el proceso seguido en el desarrollo de este proyecto, así como la estimación de tiempo para cada actividad, y se hablará de las dificultades encontradas en las distintas etapas. También se hablará de las distintas

líneas de ampliación de este proyecto, que por cuestiones de tiempo no han podido realizarse. Finalmente se mostrarán distintas aplicaciones para las cuales esta plataforma podría ser de utilidad.

En el apéndice A se expondrán los conceptos necesarios relacionados con la edición de escenarios. Es recomendable que futuros usuarios de la plataforma los asimilen. Se presentarán métodos para instanciar personajes en un mapa o como configurar la iluminación. También se detallarán y se mostrarán ejemplos de uso de algunas herramientas. Se pretende que este capítulo sea una introducción a la edición de escenarios y se enfocará como un manual con los pasos a seguir para realizar distintas tareas. Toda la información detallada ha sido recopilada de distintas fuentes en Internet y organizada para mostrar los puntos más relevantes. Aunque todo el contenido expuesto en este capítulo es público y está al alcance de cualquiera, creemos que el proceso de investigación y síntesis de toda la información obtenida es una aportación importante para el proyecto y futuros usuarios de este.

Finalmente, en el apéndice B se describirá la herramienta principal de la que hace uso la plataforma desarrollada en este proyecto. Se mostrará su arquitectura y los modos de uso de que dispone. Se listarán sus funcionalidades, los comandos proporcionados para comprobar su correcto funcionamiento, y se mostrarán los pasos a seguir para su correcta instalación y uso con la plataforma desarrollada en este proyecto. Por último, comentaremos el proceso seguido para generar el groundtruth y el modo en que se codifica.

Capítulo 2

Tópicos de interés en Visión por Computador

En este capítulo, *introduciremos el contexto de este proyecto presentando un conjunto de aplicaciones de visión por computador para la mejora de la seguridad y eficiencia en vehículos. Especialmente, definiremos el problema de la detección de peatones, ya que es la aplicación principal a la que está dirigido este proyecto. A continuación, mostraremos el proceso de generación de bases de imágenes, necesarias en estos sistemas, y presentaremos el proceso análogo en un entorno virtual compuesto por distintas actividades presentadas en este proyecto, listando las ventajas presentadas.*

Posteriormente se abordará el tema de la visión estereoscópica. Comentaremos la aportación de este proyecto en relación a este problema y presentaremos un método para obtener el groundtruth en este tipo de sistemas.

2.1. Detección de objetos para ADAS

En los últimos 20 años, el elevado índice de accidentes en todo el mundo, ha motivado el desarrollo de vehículos inteligentes. Ámbitos tan diversos como la investigación, la industria automovilística y varias organizaciones relacionadas con los sistemas de transporte, han aunado fuerzas para incrementar la seguridad vial a través del desarrollo de Sistemas de Ayuda a la Conducción (ADAS; de sus siglas en inglés)[2]. La razón principal es que el 90 % de los accidentes se producen por fallo humano, siendo la mayoría de día (60 %), con buen tiempo (94 %), con vehículos en buen estado (98 %) y casi la mitad en un trayecto recto (42,8 %).

Este tipo de sistemas son instalados en los vehículos y son capaces de analizar el entorno y ofrecer información en tiempo real al conductor. El objetivo es incrementar la seguridad, eficiencia y confort del transporte mejorando la funcionalidad de los coches. Algunas de las aplicaciones existentes, aunque aún están en desarrollo o se restringen a vehículos de alta gama o militares son:

- ***Detección de señales de tráfico.*** El objetivo de estos sistemas es advertir al conductor de las prohibiciones o peligros indicados por las señales, o incluso, limitar la velocidad del vehículo a la establecida en la vía.

- ***Detección de líneas de carretera.*** Al igual que los anteriores, este tipo de sistemas suelen ser de señalización y advertencia. Son capaces de detectar y clasificar el tipo de línea de carretera de la vía por la que circula el vehículo y recomendar una velocidad máxima o advertir de una salida de carril.

- ***Detección de vehículos.*** Existen tres grupos de detección: frontal, lateral y trasera. Con detección frontal tenemos sistemas que advierten de la distáncia de otros vehículos, sistemas de control de crucero adaptativo que mantienen automáticamente una distáncia de seguridad, sistemas de minimización de daños en colisiones inminentes o incluso sistemas de control automático de las luces largas. La detección lateral y trasera suele limitarse a sistemas de advertencia, ya sea para cubrir el ángulo muerto de los espejos retrovisores y ofrecer asistencia en cambios de carril o adelantamientos, o para detectar un incumplimiento de la distáncia de seguridad y advertir de una posible colisión trasera.

- ***Detección de peatones.*** Este tipo de sistemas son relativamente nuevos, ya que la mayoria de proyectos de investigación y desarrollo en este campo se han enfocado al conductor o a elementos de la carretera, sin prestar demasiada atención a los peatones. Una de las posibles razones puede ser que la detección de peatones es una tarea compleja debido a la gran diversidad de apariencias que puede presentar un humano, los entornos saturados y las condiciones de iluminación cambiantes. El objetivo de estos sistemas es advertir al conductor de la presencia de peatones o mitigar los daños en caso de colisión inminente [8][6].

- ***Otros (detección de somnolencia o reconocimiento de conductores).*** A diferencia de los mencionados, estos sistemas incorporan los sensores o dispositivos de captura dentro del vehículo y monitorizan al conductor. Son capaces de detectar una posible falta de atención causada por la somnoléncia o reconocer al conductor y ofrecer sistemas antirrobo.

Hoy en día, aún existen distintas posturas sobre cuales son los sensores más adecuados para este tipo de sistemas. Los sensores de distancias, como el radar o

Figura 2.1: Sistema de detección de señales de tráfico.

Figura 2.2: Sistema de control de crucero automático (ACC).

Figura 2.3: Izquierda: Sistema de control automático de luces largas. Derecha: Cobertura del ángulo muerto de espejos retrovisores.

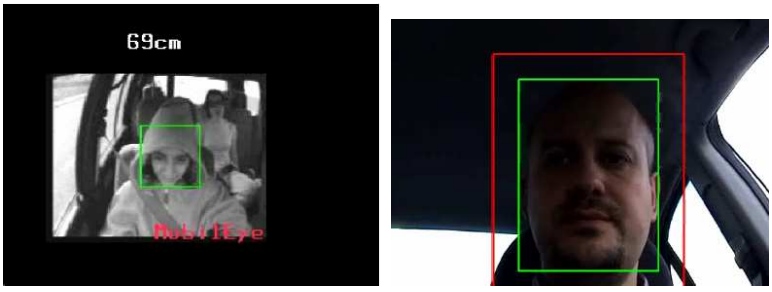

Figura 2.4: Izquierda: Sistema de control de somnolencia e inatención. Derecha: Reconocimiento de conductores

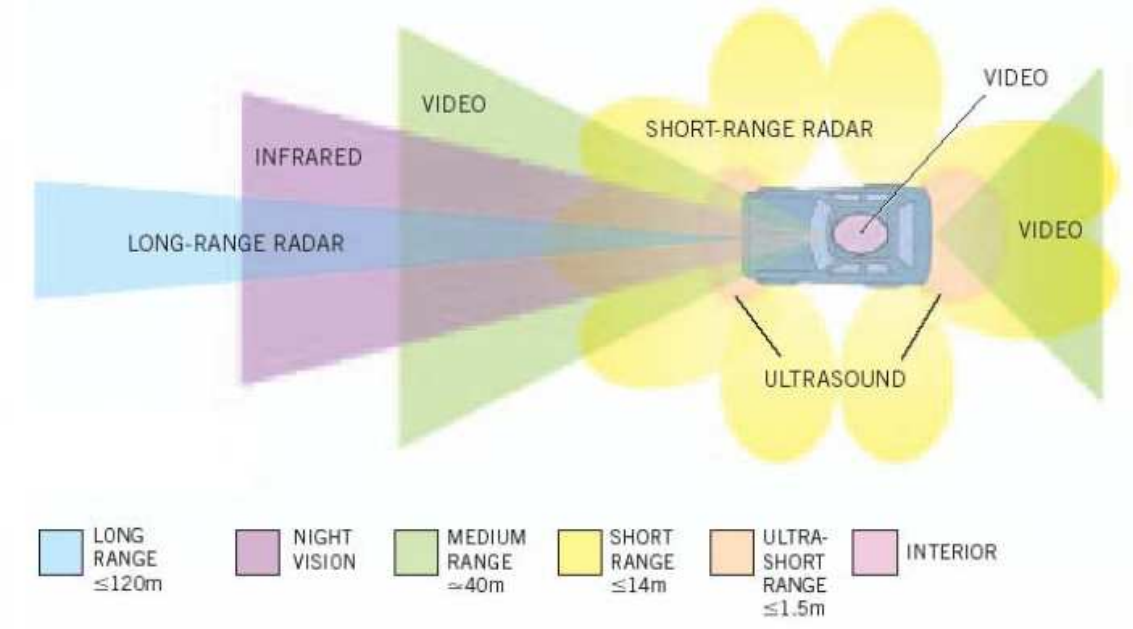

Figura 2.5: Sensores utilizados en vehículos.

el láser, tienen la ventaja de ofrecer medidas de distancias directas. Entre sus desventajas destacan su baja resolución y la tendencia a interferir con otros sensores próximos. Para la detección de peatones en ocasiones se usan cámaras térmicas. Estos dispositivos tienen la ventaja que mantienen los índices de detección en situaciones de poca visibilidad o nocturna, pero suelen ser costosas y solo detectan objetos que irradien calor. En la figura 2.5 se ilustran los distintos tipos de sensores utilizados y el alcance de cada uno de estos.

Por el contrario, la visión por computador proporciona una descripción más completa del entorno pero como contrapartida, exige un procesamiento más complicado de dicha información.

En el problema del reconocimiento de peatones, que es al que se aplicarán los resultados de este proyecto, podemos destacar dos grandes grupos. En el nivel más bajo, están aquellos que buscan rasgos sencillos que definen a una persona. Tienen el inconveniente de que si uno de esos rasgos no está suficientemente presente en la imagen, el peatón no se detecta. Además son propensos a falsos positivos. En el otro extremo de la clasificación, están aquellos métodos que incluyen algún tipo de aprendizaje. En la mayoría de los casos se basan en redes neuronales o en aprendizaje basado en ejemplos. Su mayor desventaja es que requieren de mucho tiempo de entrenamiento, pero son robustos y proporcionan buenos índices de detección.

Como se comentaba en el primer capítulo, este tipo de aprendizaje requiere de un gran conjunto de muestras de lo que se pretende detectar. Estas muestras deben estar etiquetadas y se requieren distintos datos asociados a ellas en función del tipo de sistema que estemos tratando. Este conjunto de datos es lo que se denomina ***groundtruth***.

La traducción literal del término groundtruth significa terreno verdadero. Es decir, el groundtruth nos indica la información verdadera o correcta sobre algo en un contexto concreto. En el campo de la visión por computador, esto se traduce al conjunto de información que nos indica donde están (o cuales son, que forma tienen, etc.) las entidades o objetos que queremos reconocer/identificar en un video o imagen.

El groundtruth tiene una gran cantidad de posibles representaciones y aplicaciones con diferentes fines para las que puede servir, pero su función es la misma en todos los casos. Representar el resultado correcto, el deseable a obtener por la aplicación.

Se genera mediante un proceso manual de anotación, previo al entrenamiento del sistema de detección. Este proceso es lento y tedioso, pero lo es especialmente más en el campo de la visión por computador. En otros campos, como en la inteligencia artificial, donde las muestras de los sistemas de aprendizaje suelen ser conjuntos de datos numéricos representativos de cada entidad, el proceso de anotación deriva en etiquetar y asignar un identificador de clase. En visión por computador, al trabajar con imágenes, no sirve etiquetar la imagen entera si no que hay que etiquetar el contenido de esta. Además, en función del sistema de visión, este proceso será más o menos complejo. Se pueden clasificar en tres grupos:

- ***Clasificación.*** Este es el nivel más bajo de análisis en los sistemas de detección. Se limita a examinar la imagen en busca de la presencia de los elementos a detectar. En caso de éxito, genera una caja contenedora envolviendo al elemento e indicando que está presente. El groundtruth necesario para este tipo de sistemas contiene la información de los píxeles que componen las cajas contenedoras.

- ***Seguimiento.*** Este tipo de sistemas engloban a los anteriores y añaden una restricción; No solo tienen que detectar la presencia de ciertos elementos, si no que también tienen que reconocerlos y distinguirlos entre múltiples instancias. Además de la información anterior, también se requiere un identificador para cada una de las entidades. Se requiere que se tenga en cuenta

Figura 2.6: Tipos de groundtruth. Izquierda: Clasificación. Derecha: Seguimiento. Abajo: Segmentación.

posibles apariciones previas de estas entidades en el momento de asignar los identificadores.

- ***Segmentación.*** Los sistemas de segmentación tienen el mismo objetivo que los de clasificación, detectar la presencia de un cierto elemento, pero estos lo hacen con un grado más alto de precisión. No se basan en generar una caja contenedora que envuelva al elemento, si no que se genera el contorno y se obtienen los píxeles concretos de la imágen que pertenecen a este. Con estos sistemas el groundtruth es un mapa de bits de las mismas dimensiones que la imagen original, dónde se indican los píxeles que forman parte de alguno de los elementos a detectar. Opcionalmente estos sistemas pueden ser también de seguimiento.

A simple vista se puede observar la complejidad del proceso completo y la cantidad de tiempo que habría que invertir para llevarlo a cabo. También hay que tener en cuenta que el hecho de que sea un método muy repetitivo genera que la atención del responsable que lleva a cargo la tarea mengue, con la correspondiente disminución de la fiabilidad y precisión.

El conjunto final de muestras con su groundtruth se llama base de imágenes. En la figura 2.7 se muestra el proceso completo para su obtención. El proceso completo es costoso y requiere de mucho tiempo para llevarse a cabo. Además, las dificultades que presenta el proceso de anotación no son los únicos problemas. Los sistemas de aprendizaje basados en ejemplos deben ser entrenados con conjuntos

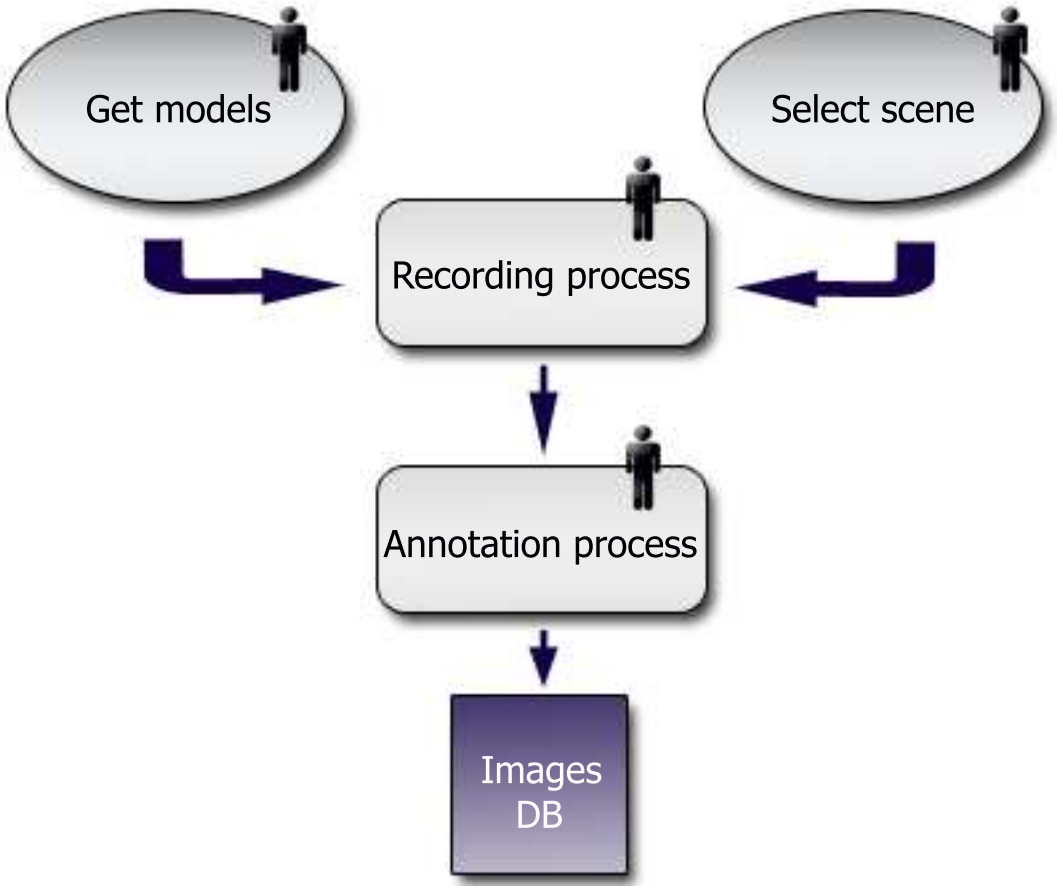

Figura 2.7: Proceso de generación de una base de imágenes para sistemas de visión por computador.

grandes y variados para obtener un buen rendimiento [14]. En un entorno real es costoso cumplir este requisito, además de que hay otros factores que también influyen como la iluminación de la escena o las condiciones climáticas.

En este proyecto se simplifica este proceso y se presenta un método para generar estas bases de imágenes en un entorno virtual. De este modo solucionamos el problema de la variedad y de la variabilidad de la escena, ya que podemos controlar el comportamiento de todos los elementos contenidos, así como de la iluminación o las condiciones climáticas. Y el principal motivo, la generación del groundtruth se automatiza y deja de ser un proceso manual.

En la figura 2.8 se muestra el proceso completo. En este proyecto se ha investigado o se han desarrollado herramientas para cada uno de los pasos:

- ***Obtención de modelos/escenarios.*** Debido a que la herramienta prin-

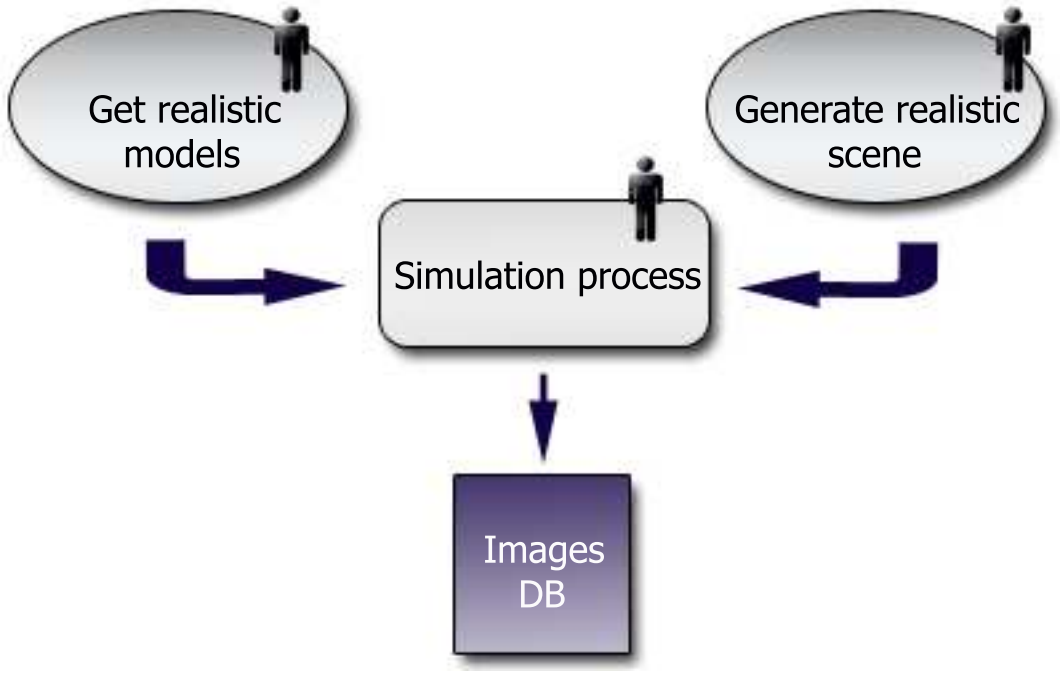

Figura 2.8: Proceso de generación de una base de imágenes sintéticas para sistemas de visión por computador.

cipal de la que hace uso la plataforma desarrollada en este proyecto es un mod (modificación) de un videojuego de Valve, tenemos que usar las herramientas y formatos compatibles con esta. La aplicación que proporciona Valve para generar los escenarios es el editor Hammer. En este proyecto se ha investigado y recopilado la información necesaria para generar un escenario realista con peatones y vehículos. Los modelos pueden ser generados a través de cualquier aplicación de modelado 3D o se pueden obtener de Internet. También se presenta un método para importar estos modelos y poder usarlos con el editor Hammer. Todos los detalles relacionados se ilustran el anexo A.

- ***Simulación.*** Con la plataforma desarrollada en este proyecto se presenta un método cómodo y sencillo para simular una navegación en un escenario obtenido en el paso anterior y recuperar la secuencia de video asociada. El groundtruth se obtiene a la vez que la secuencia de video, pero debido a que está comprimido, se ha implementado una aplicación en Matlab para, posteriormente, recuperar la información original. Los detalles de la plataforma se ilustrarán en el capítulo 3.

2.2. Visión estereoscópica

La visión estereoscópica (o también llamada visión binocular) es la que nos permite apreciar las distancias y volúmenes del entorno que nos rodea. Nos permite ver en tres dimensiones y tener una sensación de profundidad en una escena.

Esta visión en tres dimensiones es resultado de la capacidad del sistema visual de dar aspecto tridimensional a los objetos a partir de las imágenes en dos dimensiones obtenidas en cada una de las retinas de los ojos. Debido a la separación entre estos, las imágenes obtenidas contienen un conjunto de pequeñas diferencias. Estas diferencias son denominadas disparidad. Mediante un proceso llamado *estereopsis*, nuestro cerebro procesa las diferencias entre ambas imágenes y las interpreta de forma que percibimos la sensación de profundidad, lejanía o cercanía de los objetos que nos rodean. En la figura 2.9 se muestra un ejemplo de este tipo de imágenes.

Por lo tanto, a partir de la disparidad se puede realizar una estimación de la

Figura 2.9: Pareja de imágenes e imagen estereoscópica.

distancia de los objetos. Cuando observamos objetos lejanos, los ejes ópticos de nuestros ojos son paralelos y la disparidad es pequeña. Al contrario, al observar objetos cercanos, los ojos giran y se alinean sobre estos (se suele denominar convergencia) y la disparidad es alta. En la figura 2.10 se puede observar este hecho. Para verlo más fácilmente, la imagen ha sido pre-procesada y solo se muestran los contornos de los objetos. Se puede observar como la disparidad del objeto marcado más cercano es mayor que la del que está a más distancia.

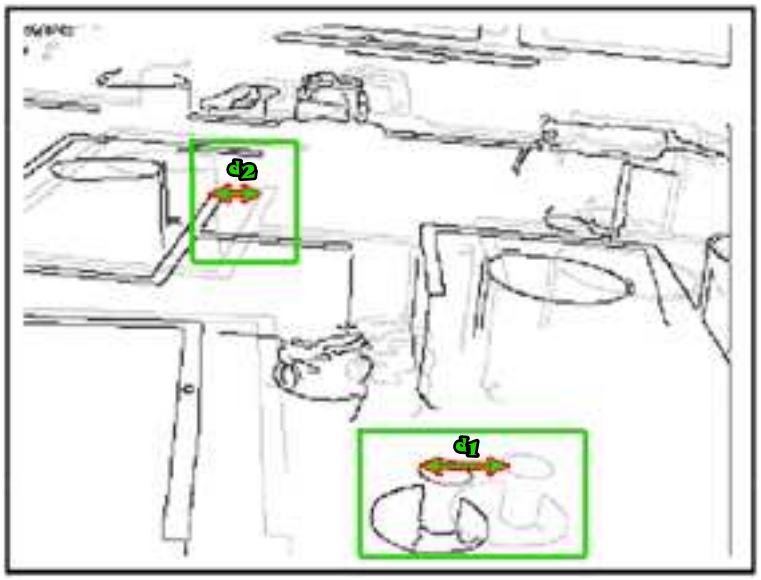

Figura 2.10: Imagen estereoscópica. Muestra de distinta disparidad a distintas distancias.

2.2.1. Cámaras Estereoscópicas

En el mundo de la visión por computador se han realizado distintas investigaciones y proyectos en el campo de la visión estereoscópica simulando el sistema binocular humano, dando paso a películas en tres dimensiones, imágenes volumétricas (utilizadas en la visualización de ultrasonidos y resonancias magnéticas en medicina), o incluso en simuladores. Las imágenes y secuencias de video en este tipo de proyectos se obtienen a través de sistemas de captura compuestos por varias cámaras. En la figura 2.11 se muestran algunos ejemplos de dichos sistemas.

A través de estos dispositivos se obtienen las parejas de imágenes con las que se puede realizar la estimación de la distancia de los objetos a partir de la disparidad. Este tipo de sistemas tienen asociados un par de problemas a resolver; Uno previo a la captura de las imágenes y otro relacionado con el cálculo de la disparidad.

- ***El problema de la reconstrucción.*** A partir de las proyecciones obtenidas por las cámaras se debe determinar su situación dentro de la escena. Esto implica que las cámaras deben ser calibradas y se deben conocer sus parámetros (posición, orientación, distancia entre ellas, etc.). Este punto suele ser una fuente de ruido, ya que es difícil obtener con exactitud toda esta información. En este proyecto este punto es solucionado, ya que en todo momento se dispone de los parámetros exactos de las distintas cámaras.
- ***El problema de la correspondencia.*** Para proceder al cálculo de la dis-

Figura 2.11: Cámaras estereoscópicas.

paridad deben encontrarse las parejas de proyecciones de cada punto visible en las imágenes. Este proceso es posterior a la captura de las imágenes y debe ser realizado por el sistema de visión estereoscópica. En la figura 2.12 se muestra un ejemplo. Este punto no puede cumplirse siempre, ya sea por la aparición de oclusiones (figura 2.13) o porque los objetos no están dentro de la región binocular común a las dos imágenes (figura 2.14).

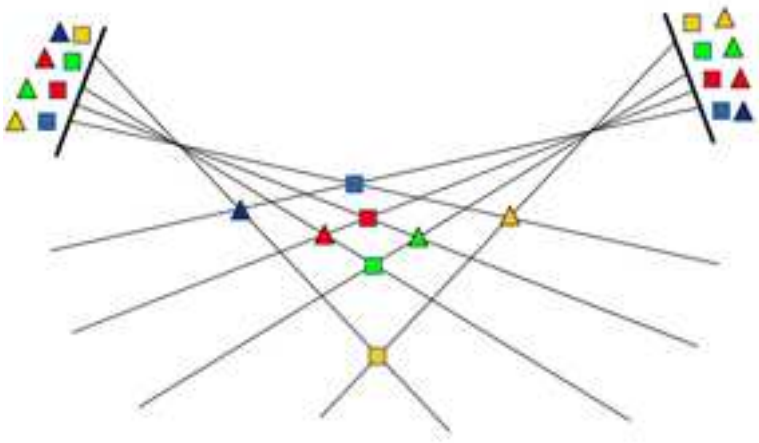

Figura 2.12: Correspondencia de objetos a partir de proyecciones.

Existen dos grupos de configuraciones para las cámaras que componen los sistemas de visión binocular. Cada una de estas presenta un conjunto de ventajas y a la vez tiene sus contras.

- ***Estéreo con ejes paralelos.*** Las cámaras se colocan en paralelo a una cierta distancia y con una orientación de 90 grados. Existen dos variantes

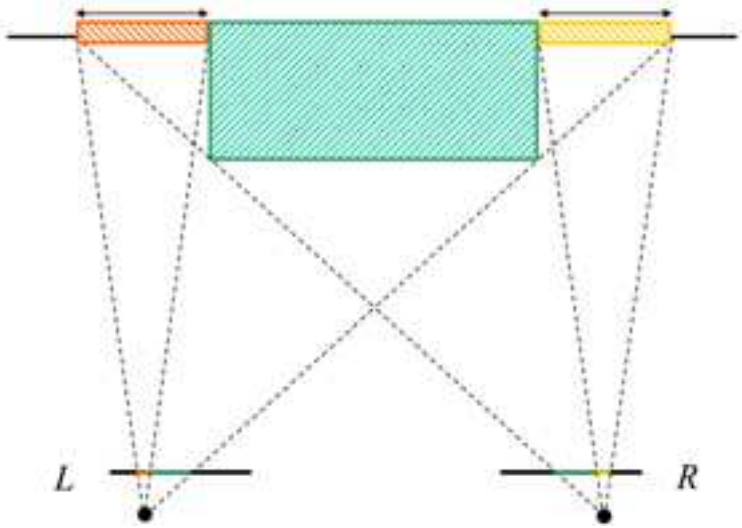

Figura 2.13: Ejemplo de oclusión. El objeto rojo no aparecerá en la imagen R y el amarillo no aparecerá en la imagen L.

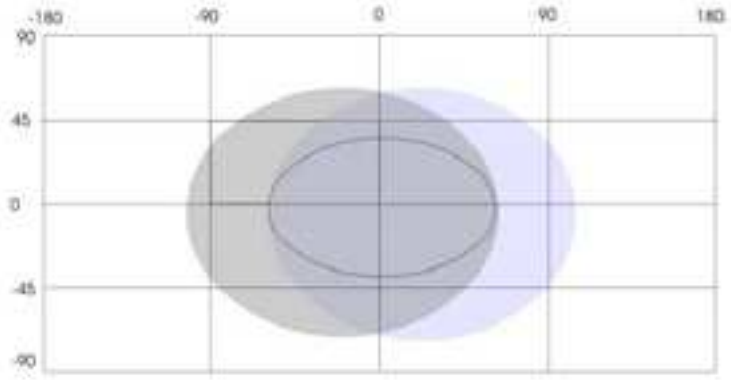

Figura 2.14: Muestra del campo de vista de un sistema de visión estereoscópico. Solo la región de la escena común a las dos imágenes dispondrá de estimación de la profundidad.

para esta configuración:

- *Línea de base corta (figura 2.15).* La región común a las dos imágenes es grande y se abarca gran parte del campo de visión, pero presenta errores de profundidad (cuanto más lejano sea el objeto, menos precisa será la estimación de su distancia).
- *Línea de base larga (figura 2.16).* La región común a las dos imágenes es pequeña aunque aumenta la precisión. En este tipo de configuración aumenta la probabilidad de que aparezcan oclusiones.

- ***Estéreo con ejes convergentes.*** La orientación de las cámaras es variable. El eje óptico de estas converge en el punto de fijación (figura 2.17). En esta configuración de cámaras aumenta el campo de visión común. Aparece el concepto de horóptero, la curva definida por el conjunto de puntos de fijación (figura 2.18). La disparidad ya no se calcula a partir de la distancia, si no a partir de los ángulos de los ejes ópticos. La precisión viene dada por la proximidad del punto de fijación. A mayor distancia de este, la región cubierta de la escena será más profunda, pero el error también será mayor.

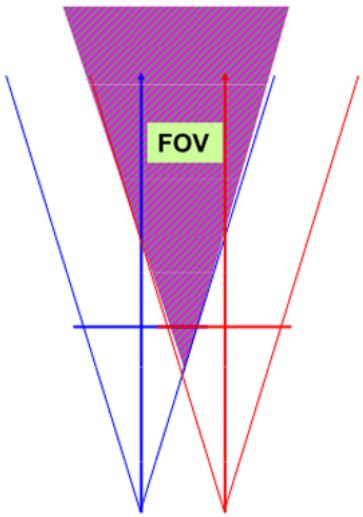

Figura 2.15: Estéreo con ejes paralelos y línea de base corta.

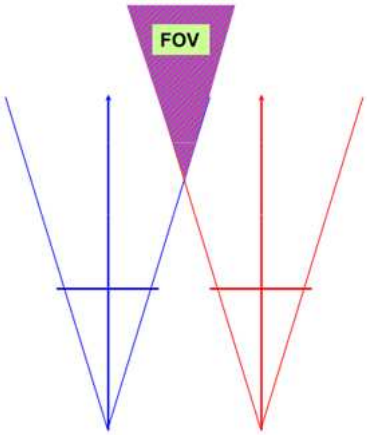

Figura 2.16: Estéreo con ejes paralelos y línea de base larga.

A través de la plataforma desarrollada en este proyecto se pueden simular todas las configuraciones mencionadas. Al activar el modo estereoscópico, se instancian

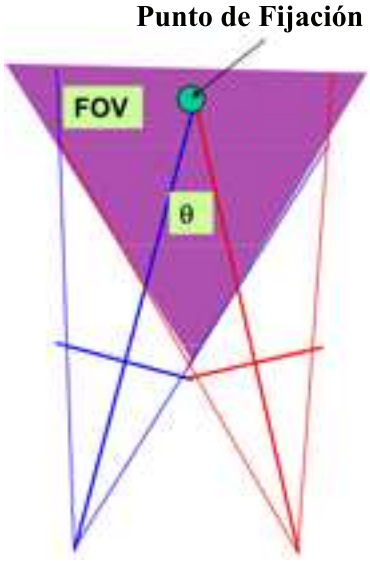

Figura 2.17: Estéreo con cámaras convergentes. Punto de fijación

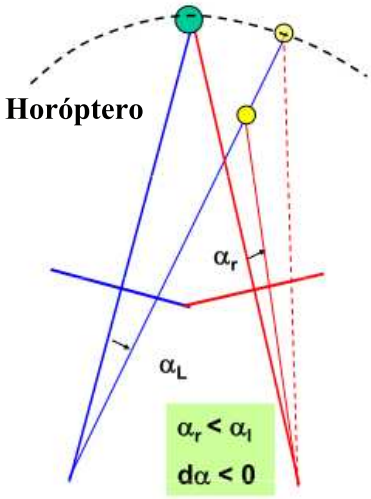

Figura 2.18: Estéreo con cámaras convergentes. Horóptero

dos cámaras en el escenario con una configuración inicial de ejes ópticos paralelos con línea de base corta. El campo de visión inicial de cada cámara será de 45 grados. Los parámetros que se pueden modificar son:

- Orientación de las cámaras.
- Campo de visión de las cámaras.
- Distancia entre cámaras.

Excepto la distancia, que es un parámetro común, el resto de parámetros se pueden modificar para cada una de las cámaras.

2.2.2. Generación del mapa de profundidad

De igual modo que otros sistemas de visión por computador, los estereoscópicos también necesitan disponer del groundtruth de las imágenes. En este caso, el groundtruth no contendrá la información de los targets, como ocurre en los sistemas de detección. El groundtruth en los sistemas de visión estereoscópica son los mapas de profundidad. Estos indican la distancia de cada pixel de la imagen a la cámara dentro de la escena. Actualmente existen sistemas de detección que incorporan la funcionalidad de visión estereoscópica, no solo para estimar la distancia de los objetos a detectar, si no también para filtrar las distintas regiones de la imagen analizando únicamente aquellas que estén relativamente cerca, mejorando así el rendimiento del sistema [10] (figura 2.19);

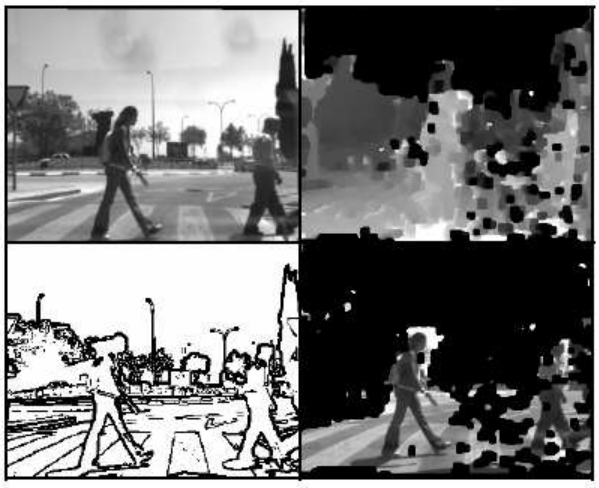

Figura 2.19: [Arriba] Imagen tomada con la cámara estéreo y el correspondiente mapa de disparidad. [Abajo] Detalle de la imagen gradiente y la imagen filtrada en función de las medidas de distancias.

Es fácil darse cuenta de que si en los sistemas de detección el proceso de anotación para obtener el groundtruth era complejo y tedioso, en los sistemas de visión estereoscópica este factor se dispara. Obviamente, este tipo de imágenes deben ser generadas por ordenador. Seria impensable intentar de medir manualmente la distáncia de la cámara a cada uno de los objetos contenidos en una escena. Aun así, la generación de este tipo de imágenes es una tarea ardua y compleja. Para facilitar un poco el trabajo a toda la comunidad, distintos equipos de investigación de diversas universidades suelen ofrecer conjuntos de imágenes ya

etiquetadas (figura 2.20). La web [13] es un ejemplo de ello.

En este proyecto se ha implementado la funcionalidad de captura de parejas

Figura 2.20: Imagen estereoscópica con su mapa de profundidad.

de imágenes para el entrenamiento o validación de sistemas de visión estereoscópica, pero por cuestiones de tiempo no se ha podido realizar la aplicación de generación del groundtruth. Aún así, se ha ideado la manera en que podría desarrollarse dicha aplicación. Este punto es una de las líneas de ampliación de este proyecto.

Al estar en un entorno virtual, tenemos la ventaja que disponemos de toda la información del escenario. Conocemos la posición exacta de cada uno de los objetos que componen el escenario y sus dimensiones. También dispondremos de la información de las cámaras a partir de su groundtruth a medida que definamos su trayectoria por el escenario. Uniendo todos estos elementos tenemos lo necesario para generar los mapas de profundidad de cada imagen capturada por las cámaras.

En primer lugar tendremos que hacer distinción en los elementos que componen el escenario.

- ***Entidad de bloque:*** Este tipo de entidades son creadas desde el editor de escenarios y son sólidos compuestos por seis caras. La información de los vértices de los bloques está contenida en el fichero fuente del escenario tal y como se muestra a continuación.

  solid
  {
  ïd2”
  side
  {
  ïd1”
  ”plane(-224.13766479 -1984.18994141 800) (-224.13766479 -1984.18994141 -

```
384) (-224.13766479 -1616.17016602 -384)"
    "materialTOOLS/TOOLSNODRAW"
    üaxis[0 1 0 25.05960083] 5.75027974"
    "vaxis[0 0 -1 43.24319839] 18.49999955"
    rotation0"
    "lightmapscale512"
    "smoothing_groups0"
   }
   ...
  }
```

- ***Entidad de modelo:*** Este tipo de entidades son creadas mediante una aplicación de modelado 3D y desde el editor de escenarios simplemente son instanciadas. La información de los distintos planos que componen el modelo está contenida en el fichero fuente de dicho modelo. El fichero fuente del escenario contendrán la posición en que se instancian los modelos 3D y la ruta hasta estos.

Llegado a este punto disponemos de la posición y orientación de las cámaras y la de los distintos elementos que componen el escenario. El algoritmo que debería seguir la aplicación para generar el groundtruth es el siguiente:

1. ***Extracción de información.*** Deberá analizarse el fichero fuente del escenario y extraer la información de los planos que componen los bloques y los modelos 3D dentro de dicho escenario. Si el escenario está cargado en exceso de bloques o modelos, el número de planos con los que se deberá tratar será grande y se disparará el consumo de memoria. Se recomienda utilizar escenarios pequeños con pocos objetos y utilizar algún método de ordenación e indexado de los planos para un acceso más rápido y búsquedas más eficientes.

2. ***Proyecciones.*** Desde la posición de la cámara y en la dirección en que este orientada realizar una proyección. Para cada uno de los pixeles de la imagen, buscar la intersección con el plano (ya sea de bloque o modelo) más cercano y calcular la distancia de la cámara hasta este. La imagen estereoscópica para la que deberá generarse el mapa de profundidad, es la equivalente a la que capturaría una cámara colocada en la posición central del sistema estereoscópico. Deberá calcularse esta posición central a partir de la posición de la pareja de cámaras. En la figura 2.21 se muestra una vista aérea del sistema estereoscópico, y se marcan los campos de visión de cada cámara y los objetos que aparecerán en el mapa de profundidad.

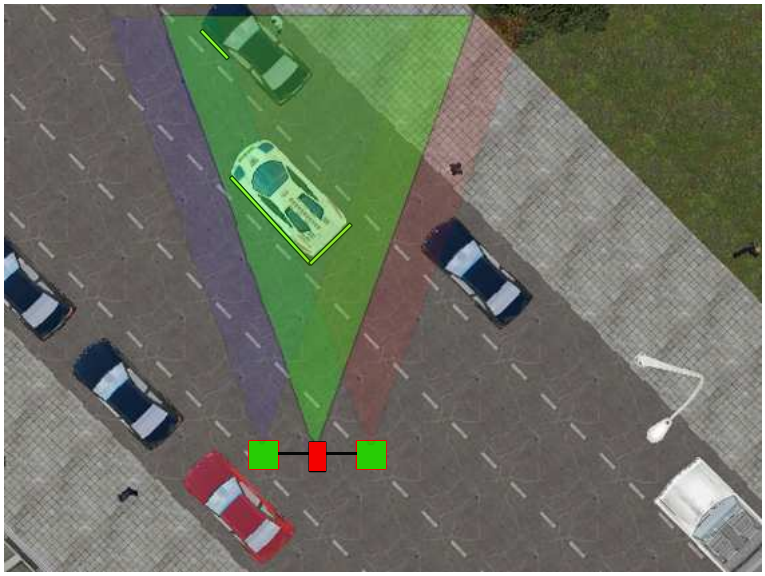

Figura 2.21: Vista aérea con distribución de cámaras estereoscópicas. Las líneas amarillas indican la parte visible de los objetos que se representará en el mapa de profundidad.

A través del método propuesto se pueden generar los mapas de profundidad de las imágenes estereoscópicas en entornos virtuales añadiendo las siguientes ventajas frente a los generados en entornos reales:

- ***Automatización.*** La generación del mapa de profundidad deja de ser un proceso manual. En función de la complejidad del escenario puede tener un coste computacional alto, pero ya que solo deberá generarse una vez por imagen, este no es un factor crítico.
- ***Precisión.*** Los mapas de profundidad en entornos virtuales contendrán la distancia de cada pixel a la cámara con mayor exactitud (en lugar de por regiones o por objetos).

El único requisito para este método es disponer de los ficheros fuente de los escenarios y modelos 3D usados (en el apéndice A se detalla cómo obtenerlos) y que todos los elementos del escenario sean estáticos. El fichero fuente del escenario nos indica las posiciones iniciales de los bloques y de los modelos. Si animamos a los modelos y les definimos trayectorias dejaremos de conocer su posición.

2.3. Resumen

En este capítulo se ha introducido el contexto de este proyecto y se han mostrado algunos sistemas de detección que existen, especialmente, los relacionados con la detección de peatones. Se ha comentado el proceso de generación de

bases de imágenes, mostrando lo costoso y tedioso que es realizar esta tarea, y se ha presentado la solución propuesta por este proyecto.

Finalmente, se ha tratado el tema de la visión estereoscópica, detallando en qué consiste, cuáles son sus problemas asociados y cuáles se solucionan mediante el uso de esta plataforma. Aunque no se ha podido implementar en el marco de este proyecto, se ha propuesto un método para la generación del groundtruth de este tipo de sistemas en un entorno virtual.

Capítulo 3

Plataforma para la generación de bases de imágenes

En este capítulo, *describiremos en detalle la plataforma desarrollada en el marco de este proyecto. En primer lugar se listarán y introducirán las herramientas seleccionadas para el desarrollo de este proyecto. A continuación se presentarán los requisitos de estas herramientas, y por lo tanto, los de la própia plataforma. Finalmente, comentaremos la estructura general de esta, así como su uso y funcionalidades. Para hacer más amena la explicación nos apoyaremos en el uso de diagramas de clase y diagramas de flujo.*

3.1. Introducción

El objetivo principal de este proyecto era encontrar un método para simplificar el proceso de generación de las bases de imágenes, necesarias en los sistemas de visión basados en aprendizaje por ejemplos. Se propuso trasladar este proceso de generación a un entorno virtual, ya que son varias las ventajas de realizar este proceso en este tipo de entornos en comparación al entorno real (gran variedad de objetos y situaciones/escenas, control de la iluminación o condiciones climáticas, reducción de costes, automatización del proceso de anotación necesario en entornos reales, etc). Para llevar a cabo este fin, se seleccionaron las siguientas herramientas:

- ***ObjectVideo Virtual Video Tool (OVVV).*** Herramienta que permite instanciar distintas cámaras en un escenario virtual, y obtener el flujo de video asociado con su groundtruth. La generación del groundtruth se produce de modo automático por la herramienta, por lo que se suprime el

proceso de anotación requerido en entornos reales. Los detalles de esta herramienta se ilustran en el anexo B.

- ***Hammer.*** Editor de escenarios, proporcionado por Valve. Esta herramienta se ha utilizado para diseñar los escenarios utilizados para generar la base de imágenes de este proyecto. Este herramienta es compatible con la herramienta de ObjectVideo.

Aunque la herramienta OVVV de ObjectVideo ya proporciona un método para generar estas bases de imágenes, la interacción con esta (DirectShow o aplicaciones basadas en grafos de filtros) es una tarea ardua y lenta. La única manera de posicionar la cámara dentro de la escena es a través de parámetros, sin la posibilidad de hacerla avanzar o cambiar su orientación sin realizar una nueva reconexión. Esta característica dificulta la interacción del usuario con la herramienta, ya que probablemente deberá realizar varias reconexiones antes de encontrar la posición deseada para la cámara. Además, el hecho de tener que configurar la cámara de este modo obliga al usuario a conocer el conjunto de parámetros.

En este proyecto se ha diseñado e implementado un software front-end que facilita este proceso y se adapta a nuestras necesidades, y ofrece, entre otras funcionalidades, la posibilidad de simular una navegación dentro de un escenario virtual y recuperar el flujo de video con su groundtruth asociado.

3.2. Viabilidad del proyecto

Para realizar este proyecto nos basaremos la herramienta ObjectVideo Virtual Video Tool desarrollada por ObjectVideo, una empresa líder en sistemas de video vigilancia.

La aplicación de ObjectVideo, es una modificación (*mod*) de un popular videojuego de tipo FPS (First Person Shooter) con una gran calidad gráfica y un motor de física muy realista; HalfLife 2. Debido a que está muy orientado al juego en red, la arquitectura de este es de tipo Cliente/Servidor. Por lo tanto, la herramienta OVVV presenta la misma arquitectura e implementa un servidor de cámaras que se comunicará con el servidor del juego, y será capaz de recuperar las imágenes capturadas por estas cámaras en distintos puntos de un escenario. Una de las funcionalidades más interesantes de esta aplicación es que permite obtener el groundtruth de cada imagen.

La aplicación a desarrollar en este proyecto se comunicará con este servidor, creará cámaras en el escenario y guardará las imágenes capturadas por estas.

Por lo tanto, desde el punto de vista económico solo será necesario obtener una licencia para el videojuego HalfLife 2. La herramienta OVVV, así como el resto de herramientas utilizadas en este proyecto disponen de licencia GPL.

Desde el punto de vista de los requisitos técnicos será necesario un ordenador capaz de soportar el juego. Para el desarrollo y validación de este proyecto, así como de la obtención de distintos videos se ha usado un PC con las siguientes características.

- Intel Core 2 Quad 2.8 Ghz
- 4 GB RAM DDR2
- Tarjeta gráfica Nvidia Quadro NVS 290

Es altamente recomendable disponer de una tarjeta gráfica con alta capacidad de procesado. El proceso del cálculo del groundtruth realizado por la herramienta OVVV se realiza en la GPU, por lo que esta debe de ser potente para un rendimiento óptimo.

También será necesario que el equipo tenga instalada la aplicación ObjectVideo Virtual Video Tool, el editor de escenarios Hammer (necesario para crear/modificar escenarios) proporcionado por Valve y un programa de modelado 3D para crear/modificar los modelos de personas/vehículos. La plataforma se ha desarrollado en C++ y debe ser ejecutada en Windows.

3.3. Funcionalidades

A continuación listamos las funcionalidades que ofrece la plataforma. Para un uso más cómodo, se ha diseñado un interficie gráfica de usuario (GUI), con un total de cuatro diálogos; diálogo principal, diálogo de cámara simple, diálogo de cámara estereoscópica y diálogo de configuración. Podemos clasificar las funcionalidades en función de donde se presentan en la plataforma:

- ***Diálogo principal (figura 3.1).*** Es el diálogo que se muestra al iniciar la plataforma.
    - Se pueden seleccionar qué efectos ópticos aplicar a los frames y especificar el valor de estos.
    - Área de mensajes. Aquí se mostrarán al usuario todos los mensajes que genere la plataforma.

- ***Diálogo de cámara simple (figura 3.2).*** Es el diálogo que se muestra cuando nos conectamos al servidor de cámaras. Nos permite explorar el escenario, así como otras funcionalidades.
  - Instanciación de una cámara dentro del escenario. Movimiento libre mediante el ratón sobre un plano definido en los ejes X y Y. Posibilidad de modificar posición y orientación de la cámara.
  - Obtención de frames y archivos de groundtruth asociados. Las imágenes están codificadas en jpeg. La escritura de estos es opcional, de este modo el usuario es libre de explorar el escenario sin consumir espacio en disco.
  - Opción de configurar la cámara (posición y orientación) especificando las coordenadas.
  - Opción de convertir la cámara en una cámara PTZ (Pan, Tilt, Zoom). Se puede establecer la velocidad de *Pan*, los ángulos entre los cuales oscilará la cámara y la posibilidad de hacer zoom.
- ***Diálogo de cámara estereoscópica (figura 3.3).*** Es el diálogo que muestra las imágenes obtenidas a partir de la pareja de cámaras colocadas en paralelo. Dispone de las mismas funcionalidades que el diálogo de cámara simple, además de la opción de modificar distintos parámetros relacionados con la visión estereoscópica (orientación, campo de visión y distancia entre cámaras).
- ***Diálogo de configuración (figura 3.4).*** Permite establecer las opciones principales.
  - Configurar el framerate.
  - Establecer la velocidad de avance de la cámara.
  - Establecer el directorio donde se guardará el fichero de efectos visuales. Este punto se explicará con más detalle en la sección siguiente.
  - Establecer el tamaño de las imágenes. Hay 3 opciones: 320x240, 640x480 y 800x600.

La plataforma desarrollada en este proyecto ha sido implementada en C++, junto con las Microsoft Foundation Classes (MFC) para el desarrollo de la interficie gráfica. A continuación listamos los componentes externos usados en este proyecto:

- Librería para la gestión de la comunicación con la herramienta OVVV. Desarrollada por ObjectVideo.
- Librería para la carga de imágenes jpg.

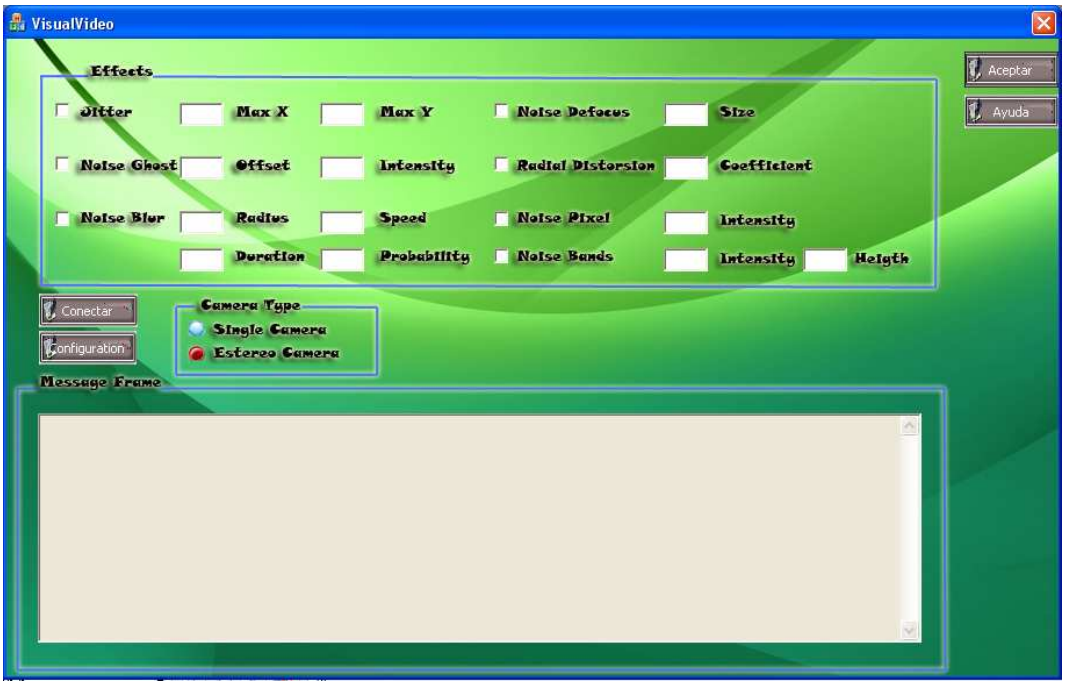

Figura 3.1: Diálogo principal.

- Librería para asociar skins a botones de las MFC.

Todos los componentes externos a este proyecto disponen de licencia GPL.

3.4. Funcionamiento interno

Para hacer al usuario más sencilla la interacción con la plataforma se ha desarrollado una GUI mediante las MFC de Microsoft. Al incluir una GUI, la plataforma se convierte en una aplicación de tipo dirigida por eventos. El esquema general de funcionamiento de este tipo de aplicaciones se muestra en la figura 3.5.

En este tipo de aplicaciones, podemos destacar dos fases; La fase de inicialización y la fase de escucha de los eventos.

En la fase de inicialización, como su propio nombre indica se inicializan las variables y estructuras de datos necesarias para el correcto funcionamiento de la aplicación. Este proceso se realiza una única vez. En este proyecto, en la fase de inicialización se construye el CameraManager, las cámaras, se configuran algunas variables y se inicializa el diálogo principal.

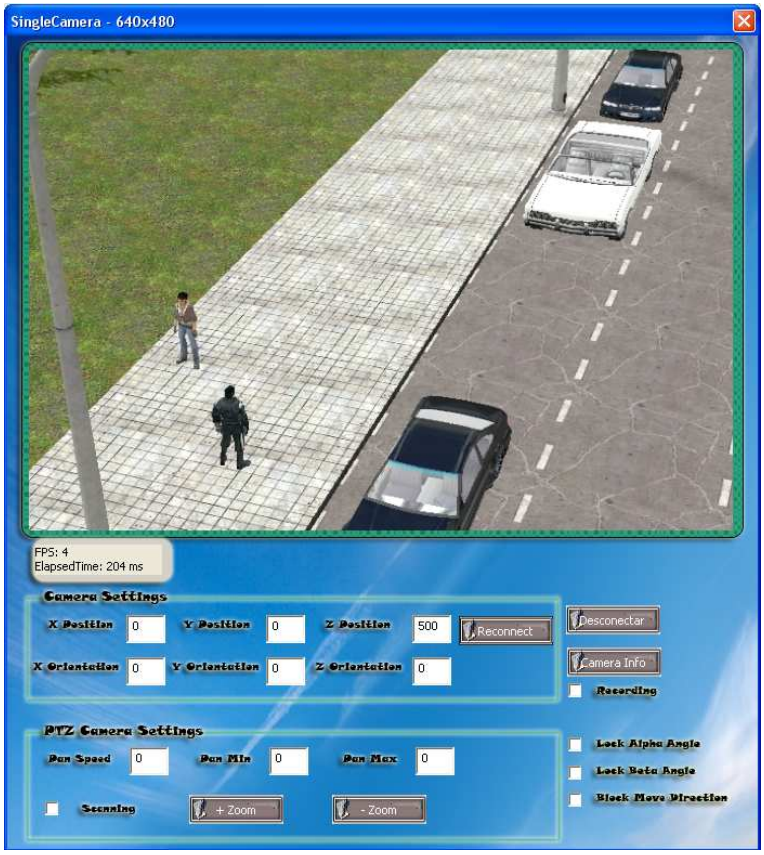

Figura 3.2: Diálogo de cámara simple.

La segunda fase se compone de un proceso de espera a que se produzca un evento, y a continuación el proceso de atender dicho evento. En las MFC, la gestión de eventos se implementa mediante mensajes y los handlers de Windows. Es necesario definir el conjunto de eventos a los que escuchar y asociar una respuesta a cada uno de ellos. Cuando se produzca uno de los eventos especificados se ejecutará la función asociada, que contendrá el código para atender al evento. Una vez se ha atendido al evento se vuelve al punto de espera. Este proceso se repite indefinidamente hasta que se produce un evento que indica el fin de la aplicación.

En las figuras 3.6 y 3.7 se muestran los diagramas de clases de la aplicación. Por cuestiones de espacio, se muestran por separado las clases que pertenecen a la capa de lógica de control, de las que pertenecen a la capa gráfica. En la figura 3.8 se muestra un gráfico de arquitectura de la plataforma junto a la herramienta OVVV.

Las funciones de cada una de las clases de lógica de control son las siguientes:

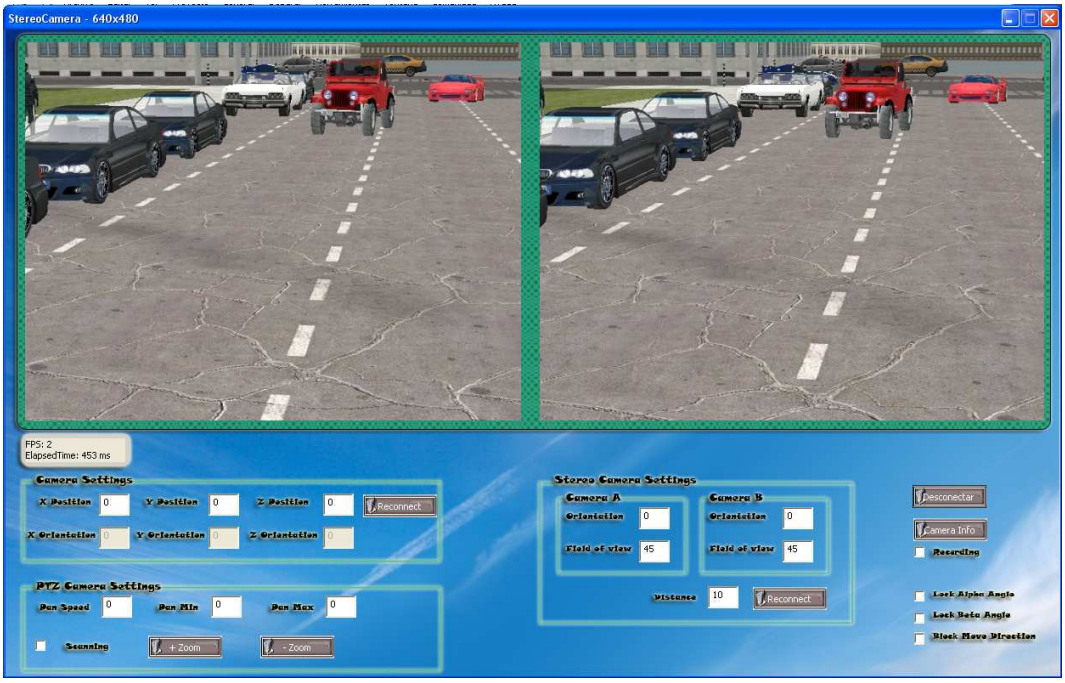

Figura 3.3: Diálogo de cámara estereoscópica.

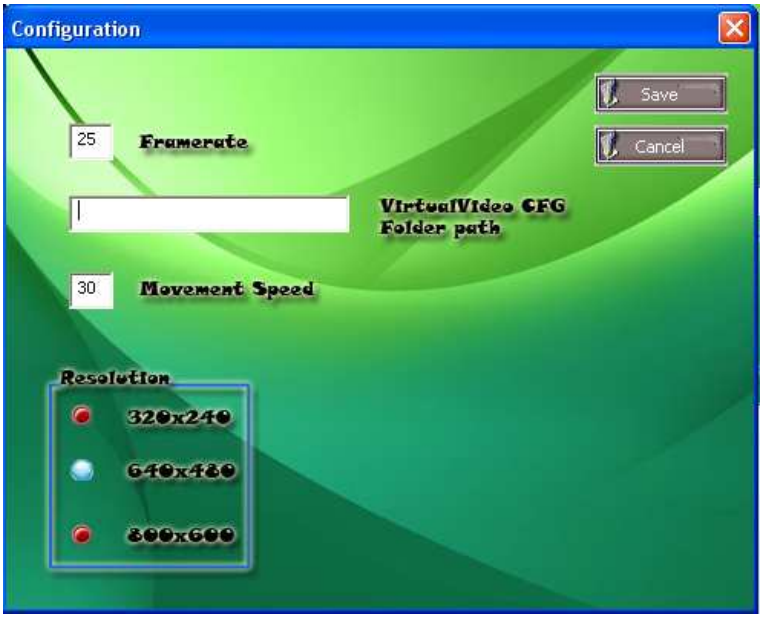

Figura 3.4: Diálogo de configuración.

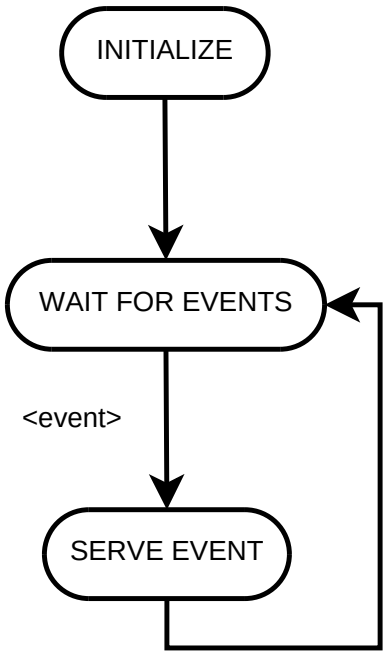

Figura 3.5: Esquema general aplicación dirigida por eventos.

- **CameraManager:** Es el responsable de la creación, mantenimiento y destrucción de las cámaras, así como de todo lo relacionado con estas. Es el elemento que contiene las cámaras y la información relacionada. Mediante este podemos crear una cámara, conectarla con el servidor, actualizar sus parámetros en función de la interacción del usuario, y finalmente desconectarla y destruirla. También se encarga de la gestión de los efectos ópticos, así como la configuración de otros parámetros como el tamaño de las imágenes o el framerate. Para diseñar esta clase se ha utilizado el patrón *Singleton*, puesto que solo va a haber una instancia de esta y sus métodos van a ser llamadas desde distintos puntos de la aplicación.

- **Camera:** Abstracción de una cámara. Es la especificación de los métodos mínimos que debe tener una cámara. Todos sus métodos son virtuales, de manera que la clase que herede de esta deberá implementarlos.

- **CameraSingle:** Representa una cámara. Hereda de *Camera*, por lo que cumple con la especificación de lo que debe hacer una cámara ya que debe implementar sus métodos. Es el componente que interactúa directamente con el servidor de cámaras. Sus funciones son las de conexión, actualización, desconexión y establecimiento de distintos parámetros de configuración.

- **CameraStereo:** Representa una cámara estereoscópica, una pareja de cámaras simples posicionadas en paralelo. Este elemento se compone de dos instan-

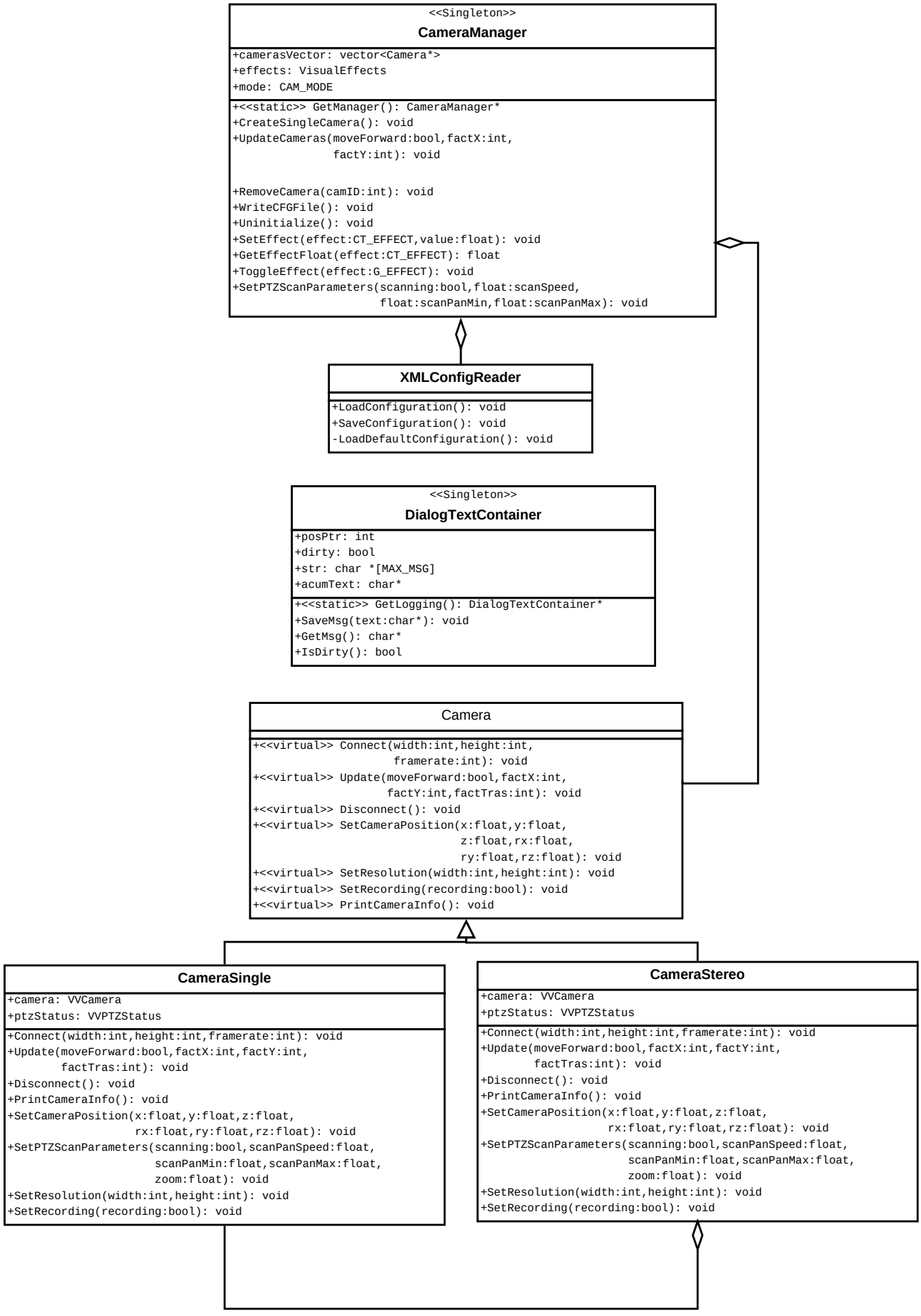

Figura 3.6: Diagrama de de clases capa lógica de control.

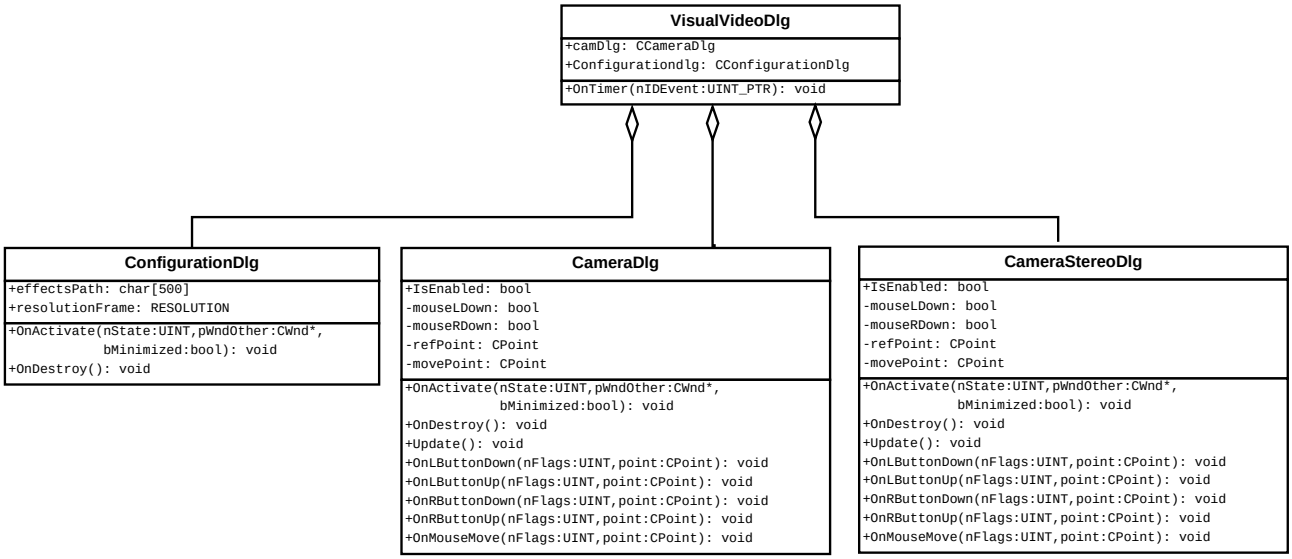

Figura 3.7: Diagrama de clases capa gráfica.

cias de CameraSingle. Está en un nivel intermedio entre el CameraManager y las dos instancias de CameraSingle. La actualización de CameraStereo supone, por parte de este último, la actualización de las dos instancias de CameraSingle.

- **XMLConfigReader:** Responsable de la lectura/modificación del fichero de configuración de la plataforma desarrollada en este proyecto. Cuando la aplicación está en la fase de inicialización, este elemento lee el fichero de configuración XML y establece los distintos parámetros mediante el *CameraManager*. Si el fichero de configuración no existe establece un conjunto de parámetros por defecto. También es el responsable de actualizar dicho fichero cuando el usuario modifica algún parámetro desde el diálogo de configuración.

- **DialogTextContainer:** Elemento responsable de almacenar los mensajes que pueda ir generando la aplicación para ser mostrados al usuario. De modo análogo a la clase *CameraManager* , en esta clase también se ha usado el patrón *Singleton*. Almacena hasta un número máximo de 50 mensajes definido por la constante *MAX_MSG*. La estructura de datos que contiene los mensajes se ha implementado mediante una cola circular, de modo que cuando se añade el mensaje número *MAX_MSG*, este se inserta en la primera posición sobrescribiendo el mensaje más antiguo. Cada vez que se añade un nuevo mensaje se activa un flag indicándolo. Cuando salta el temporizador de la aplicación, se comprueba este flag y si está activo se actualiza el campo de texto en el diálogo principal con los nuevos mensajes.

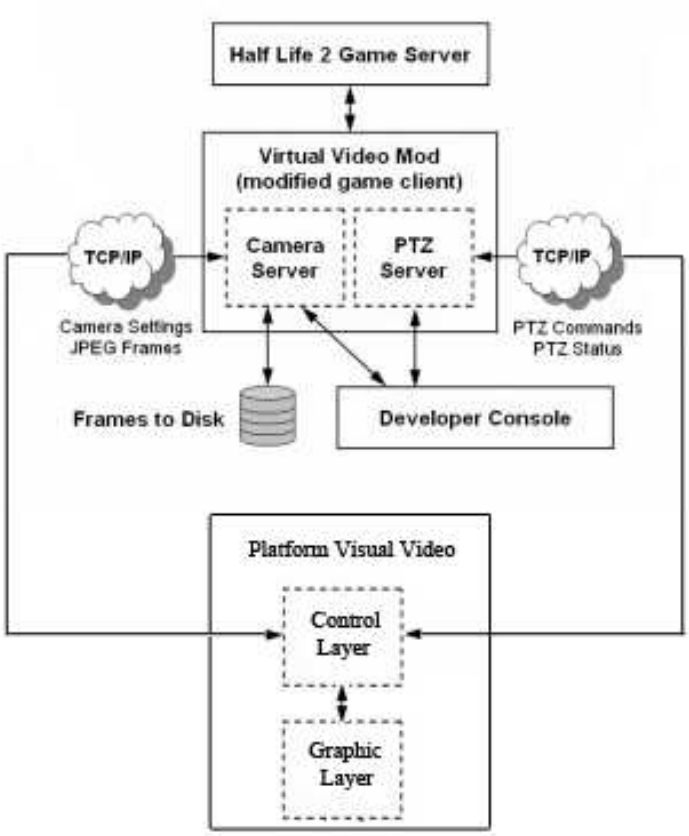

Figura 3.8: Gráfico de arquitectura.

Y a continuación se listan las funciones de cada una de las clases de la capa gráfica. Por razones de espacio de espacio se han obviado los métodos menos relevantes. La mayoría de los métodos contenidos en estas clases son para responder a los eventos generados por el usuario, como pulsar un botón, cambiar el valor de un campo de texto, etc. La respuesta asociada al evento se establece mediante el *CameraManager*.

- **VisualVideoDlg:** Clase asociada al diálogo principal. Gestiona la interacción del usuario con los distintos elementos del diálogo. Tiene asociado un temporizador para actualizar el área de mensajes y las cámaras que pueda haber conectadas. Es la responsable de crear, mostrar y destruir los dos diálogos restantes en función de las acciones del usuario.

- **CameraDlg:** Clase asociada al diálogo de cámara. Gestiona la interacción del usuario con los distintos elementos del diálogo para configurar la cámara.

También atiende a los eventos generados mediante la interacción con el ratón para generar el movimiento de la cámara.

- **CameraStereoDlg:** Clase asociada al diálogo de cámara de estereoscópica. Tiene las mismas funciones que el objeto CameraDlg, pero muestra dos imágenes en lugar de una.
- **ConfigurationDialog:** Clase asociada al diálogo de configuración. Gestiona la interacción del usuario con los distintos elementos del diálogo para configurar la plataforma.

Debido a que el flujo de ejecución de la aplicación depende de la interacción con el usuario este es variable. Mostraremos los escenarios asociados a las acciones más relevantes que se pueden realizar mediante la plataforma.

3.4.1. Escenario 1. Conexión de una cámara simple

A continuación comentaremos como se realiza el proceso de conexión de una cámara simple. Para ello nos apoyaremos en el uso de un diagrama de secuencia (figura 3.9) para poder ver los componentes implicados en el proceso y los mensajes entre ellos, y en un diagrama de flujo (figura 3.10) donde se detalla la estructura de la función que acaba realizando la conexión. En el diagrama de secuencia

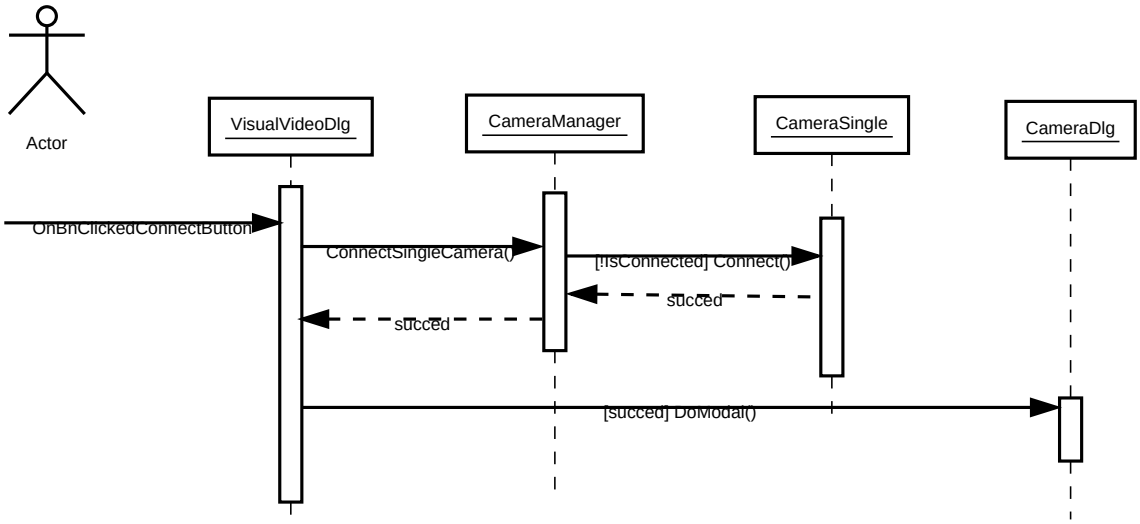

Figura 3.9: Diagrama de secuencia conexión cámara simple.

podemos ver claramente las clases que participan en el proceso de conexión de una cámara. Este empieza cuando el usuario pulsa el botón de conexión. En ese momento se genera un evento y es capturado por la clase *VisualVideoDlg*. Como

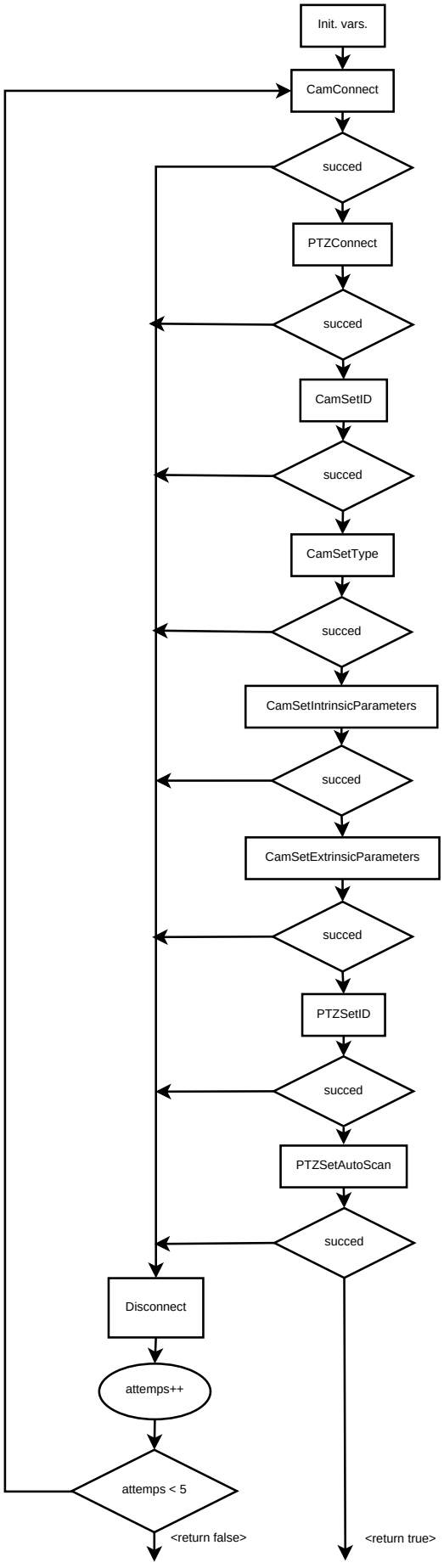

Figura 3.10: Diagrama de flujo conexión cámara simple.

hemos comentado anteriormente, este evento tiene asociado una función, en este caso es *OnBnClickedConnectButton*. Esta función obtiene la instancia del *Singleton CameraManager* y le indica que debe conectar la cámara. El componente *CameraManager* comprueba que la cámara no esté ya conectada, y en este caso ordena a la instancia de *CameraSingle* que se conecte con el servidor de cámaras. La función de la clase *CameraSingle* es quien realmente hace todo el trabajo y gestiona el proceso de conexión, por eso se ha descrito con más detalle en el siguiente diagrama de flujo. Si el proceso de conexión ha tenido éxito se muestra el diálogo de cámara al usuario para que pueda empezar a interactuar con esta.

En el diagrama de flujo podemos observar los distintos pasos que deben realizarse para poder conectarse con el servidor de cámaras. Como hemos comentado anteriormente, la herramienta OVVV está basada en una arquitectura de tipo cliente/servidor y permite conexiones mediante sockets. Para facilitar el proceso de desarrollo de nuevas aplicaciones destinadas a hacer uso y comunicarse con el servidor de cámaras, la empresa *ObjectVideo* proporciona una librería con el conjunto de funciones necesarias para realizar la conexión. Estas funciones son las que se pueden ver en el diagrama de flujo. A continuación detallaremos que realiza cada una de ellas. Para mantener toda la información compactada y actualizada se hace uso de dos estructuras (*VVCamera* y *VVPTZStatus*) que contienen todos los parámetros de configuración de la cámara y del controlador PTZ. Estas estructuras se pasan por referencia y son actualizados en cada una de las llamadas a estas funciones.

- **CamConnect:** Establece una nueva conexión con el servidor de cámaras e inicializa la estructura *VVCamera*.
- **PTZConnect:** Establece una nueva conexión con el servidor de controladores PTZ e inicializa la estructura *VVPTZStatus*.
- **CamSetID:** Conecta la aplicación cliente con la cámara representada por *camID*. Si esta no existe, se crea una nueva cámara con este identificador y los parámetros por defecto.
- **CamSetType:** Establece algunos parámetros de configuración de la cámara, como el tipo (cámara estándar o omni-cámara. Esta última no se ha usado en este proyecto), el identificador del controlador PTZ al que asociar la cámara, el directorio donde el servidor pueda guardar las imágenes (de este modo se guardan las imágenes en modo *recording*. Está explicado con más detalle en la siguiente sub-sección), etc.
- **CamSetIntrinsicParameters:** Establece los parámetros internos de la cámara. Estos son las dimensiones de las imágenes capturadas, el framerate y el campo de visión (en grados).

- **CamSetExtrinsicParameters:** Establece los parámetros externos a la cámara. Estos son la posición y orientación de la cámara en los 3 ejes.
- **PTZSetID:** Hace la misma función que *CamSetID* pero con el controlador PTZ.
- **PTZSetAutoScan:** Permite establecer los distintos parámetros para el escaneo automático, convirtiendo de esta manera una cámara fija en cámara activa. Esta funcionalidad se ha explotado en este proyecto y se ha puesto a disposición del usuario. Permite configurar los ángulos y velocidad de escaneo o el zoom.

En algunas ocasiones y sin motivo aparente, la conexión entre la plataforma y el servidor de cámaras falla. No se ha descubierto la razón de esto, debido a que la herramienta OVVV es externa y no se tiene acceso a detalles de implementación. Para menguar este problema, de modo transparente al usuario se realizan hasta un máximo de 5 intentos de conexión. Cuando alguna de estas conexiones falla, se realizan las llamadas a las pertinentes funciones para desconectar la cámara y deshacer los pasos realizados. Si ninguno de los 5 intentos tiene éxito se informa al usuario del problema.

3.4.2. Escenario 2. Actualización de una cámara simple

A continuación explicaremos en detalle el proceso de actualización de las cámaras, que es el que se realiza cuando el usuario está navegando por la escena. En la figura 3.11 podemos ver el diagrama de secuencia y en la figura 3.12 podemos ver el diagrama de flujo de la función que lleva a cabo esta tarea. El proceso

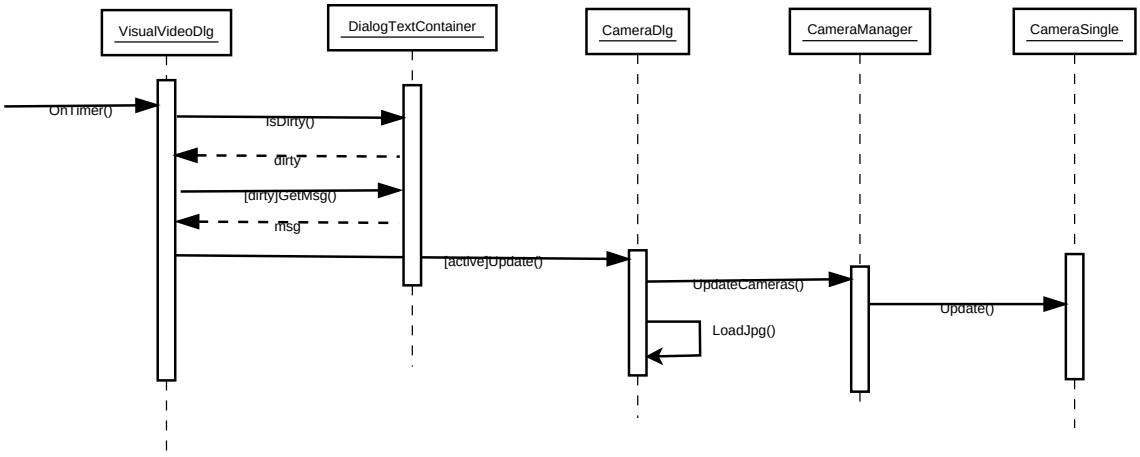

Figura 3.11: Diagrama de secuencia actualización cámara

de actualización de las cámaras es iniciado cuando se activa el temporizador de

la plataforma. Previo a la actualización de las cámaras, se actualizan los mensajes que se deben mostrar al usuario, tal y como se ha comentado anteriormente.

La actualización de la cámara empieza desde la clase asociada al diálogo de cámara (siempre y cuando esté activo). La razón de esto es porque el usuario interactúa con la cámara mediante este diálogo, y es este quien gestiona los correspondientes eventos y tiene la información de las acciones realizadas por el usuario. Se facilita al gestor de cámaras la información de la interacción del usuario y se actualizan las correspondientes al diálogo activo.

Para mostrar al usuario las imágenes capturadas por las cámaras se hace uso de una librería externa para cargar imágenes jpeg. En cada evento de actualización se muestra en el diálogo de cámara la última imagen facilitada por el servidor de cámaras.

El proceso de actualización de una cámara consta de dos partes; La actualización de los parámetros de la cámara (posición y orientación) y la actualización de la imagen que se visualizará en el dialogo correspondiente. La función de actualización recibe 4 parámetros:

- **moveForward:** Indica que la cámara debe avanzar.
- **factX:** Factor de giro en el eje X.
- **factY:** Factor de giro en el eje Y.
- **factTras:** Factor de avance. Este parámetro se puede modificar desde el diálogo de configuración.

Esta información es obtenida a través de un conjunto de funciones que monitorizan la interacción por parte del usuario mediante el ratón.

Los cálculos necesarios para obtener la nueva posición u orientación de la cámara solo se efectuarán si es necesario. De igual manera, solo si hay algún cambio se comunicará al servidor de cámaras la actualización de estas.

Finalmente se obtiene el último frame de la cámara desde el servidor de cámaras. Este frame es guardado en la estructura de tipo *VVCamera* y únicamente se utiliza para mostrarse en el diálogo de cámara correspondiente. Aunque en este punto se podría realizar el guardado de las imágenes en disco, esta tarea se delega al servidor de cámaras.

La razón de esto es debido al alto costo computacional. En este punto, el servidor

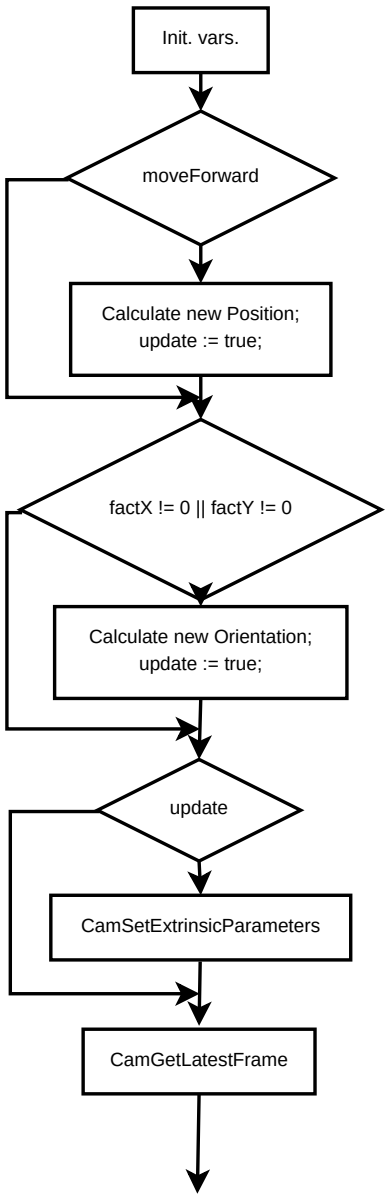

Figura 3.12: Diagrama de flujo actualización cámara simple.

de cámaras transmite mediante sockets a la plataforma el frame y el groundtruth. Cuando activamos el cálculo del groundtruth en la herramienta OVVV el coste computacional incrementa considerablemente. Al añadir esta carga es muy probable, que mientras la plataforma recibe y guarda los datos a disco, se estén generando y perdiendo nuevos frames. En cambio, si es el servidor quien guarda los datos este problema desaparece. Es posible que durante la navegación se sufran ralentizaciones en la plataforma, pero el resultado será el esperado ya que el servidor antes de generar un nuevo frame habrá guardado el anterior.

3.4.3. Escenario 3. Proceso completo

Una vez mostradas dos de las tareas básicas de la plataforma, en la figura 3.13 presentamos el proceso completo con la secuencia de pasos para generar una base de imágenes simulando un viaje en coche . El proceso se compone de tres actividades:

- ***Edición.*** En primer lugar debemos generar el escenario virtual mediante el editor Hammer. Esta es una potente herramienta de diseño, que nos ofrece un gran abanico de posibilidades y permite que tengamos absoluta libertad sobre el escenario, pudiendo generar escenas urbanas, rurales, interiores, etc. También nos permite tener control sobre la iluminación y ofrece mecanismos de inteligencia artificial para los peatones o demás elementos que incluyamos, simulando así un entorno realista.

- ***Simulación.*** Una vez generado el escenario debemos cargarlo en la herramienta de ObjectVideo. Esta herramienta creará el servidor de cámaras y comenzará a simular el escenario, actualizando los distintos elementos de la escena tal y como hayamos indicado a través del paso anterior. A continuación, mediante nuestra plataforma nos conectaremos a dicho servidor de cámaras, y al usuario se le mostrará el diálogo de cámara, ofreciéndole la opción de moverla por la escena, convertirla en una cámara activa, posicionarla en puntos concretos, etc. Si el usuario lo indica, el flujo de video obtenido por la cámara será guardado a disco junto a su groundtruth.

- ***Extracción del groundtruth.*** Debido a que la herramienta OVVV ofrece la opción de trabajar en red y permite conexiones no locales al servidor de cámaras, el groundtruth generado se codifica en binario para optimizar la transmisión. Por lo tanto, el último paso que debemos realizar es analizar los archivos de groundtruth de cada imagen para recuperar la información original. Para esto, se ha implementado una aplicación en Matlab que realiza esta tarea. La información contenida se compone de distintos parámetros de calibración de las cámaras (posición, orientación, etc), las coordenadas de las cajas contenedoras de los peatones presentes en la imágen y del

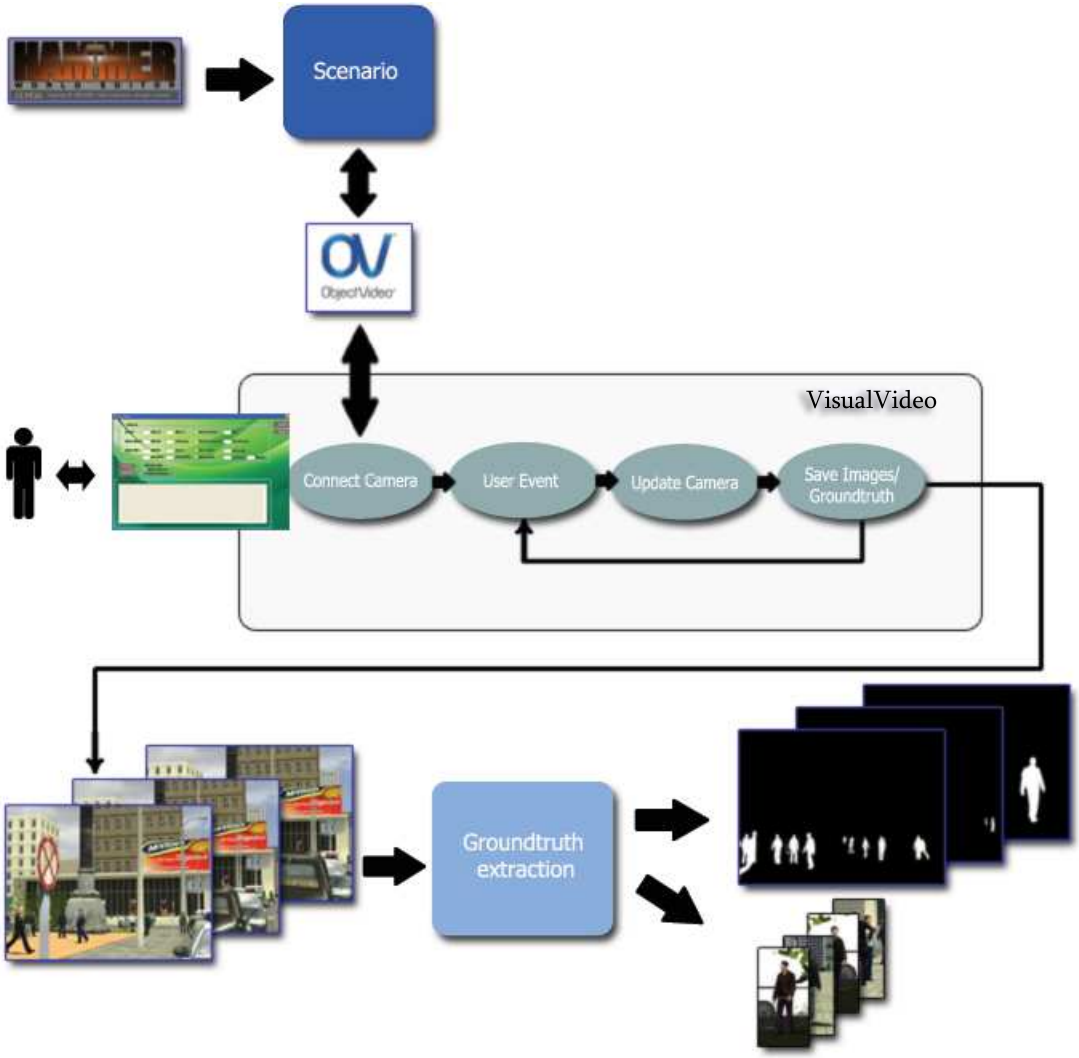

Figura 3.13: Proceso completo para generar una base de imágenes.

foreground label map, que indica que píxeles de la imagen pertenecen a un peatón. Todos los detalles sobre el groundtruth y el proceso de extracción, así como los de la herramienta de ObjectVideo se ilustran en el anexo B.

Los resultados obtenidos a través de este proceso son los recortes de imágenes de peatones que se utilizarán para el entrenamiento de sistemas de aprendizaje basados en ejemplos, las imágenes completas junto con la información que indica si hay peatones presentes y dónde estan, destinadas a la validación de sistemas de detección, o las parejas de imagen - foreground label map útiles para la validación de sistemas de visión basados en segmentación. Aunque no se ha podido realizar

en el marco de este proyecto, con la información obtenida también es posible generar mapas de profundidad destinados a la validación de sistemas de visión estereoscópica a través del método presentado en la sección 2.2.2 de esta memoria.

3.5. Resumen

En este capítulo hemos definido el uso y funcionalidades de la plataforma obtenida mediante este proyecto. También hemos definido su arquitectura interna, las funcionalidades de cada componente y su relación entre ellos. Finalmente se ha detallado el flujo de ejecución y la comunicación entre los componentes para realizar algunas de las acciones básicas de la plataforma y se ha mostrado el proceso completo a seguir para generar una base de imágenes simulando un viaje en coche.

F

Capítulo 4

Detección de peatones

En este capítulo, *mostraremos los resultados de un sistema de detección de personas entrenado con un video generado mediante la plataforma desarrollada en este proyecto. El sistema de detección ha sido desarrollado en el CVC. También compararemos dichos resultados con los obtenidos al entrenar el sistema de detección con un video real. Para acabar mostraremos algunas imágenes de distintos escenarios para observar las posibilidades del editor.*

4.1. Introducción

A continuación mostraremos el proceso seguido para entrenar y validar un sistema de visión por computador. Dicho sistema se constituye en el uso de descriptores de tipo HOG (Histogram of Oriented Gradients) y en la máquina de aprendizaje SVM (Support Vector Machines). El sistema de visión está basado en el método presentado por [7].

Para finalizar se mostrarán los resultados obtenidos al entrenar con imágenes sintéticas, y a su vez los obtenidos entrenando con imágenes reales (figura 4.1). De este modo se podrá observar si el uso de imágenes sintéticas es válido para el entrenamiento/validación de algoritmos de detección.

Tanto el escenario virtual, utilizado para generar la base de imágenes, como el sistema de detección de peatones han sido diseñados y desarrollados por investigadores del Centro de Visión por Computador (CVC).

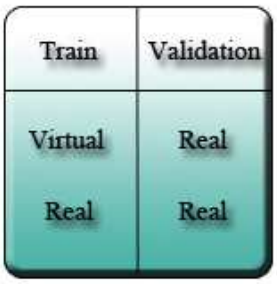

Figura 4.1: Configuración del proceso de validación.

4.1.1. Preparando el conjunto de entrenamiento

Como se comentaba en el capítulo uno, el problema del reconocimiento de objetos suele afrontarse con sistemas de aprendizaje supervisado basado en elementos. Este tipo de sistemas requieren una fase previa (offline) en las que deben realizarse ciertas tareas:

1. Etiquetado de las muestras. Debido a que estamos tratando con un sistema de aprendizaje supervisado es necesario conocer la clase de cada muestra.

2. Extracción de características. En visión por computador las muestras son imágenes de los objetos a clasificar. No se trabaja con la imagen completa, si no que se extraen las características más representativas y discriminantes de cada clase de objeto [16].

3. Aprendizaje. En esta etapa se facilitan las muestras a una máquina de aprendizaje y *se aprende un clasificador.*

En la figura 4.2 se muestra un esquema con el conjunto completo de tareas.

Es habitual dividir el gran conjunto de muestras en 3 subconjuntos disjuntos; Entrenamiento, test y validación. El conjunto de entrenamiento es el que se usa con la máquina de aprendizaje y mediante el cual se induce el clasificador. Es el que hace que el clasificador tome forma, y lo orienta hacia la detección de ciertos elementos. El conjunto de test suele ser un subconjunto pequeño el cual trabaja en paralelo con el conjunto de entrenamiento y sirve para calibrar el clasificador. Suele utilizarse en un segundo ciclo de entrenamiento.

El conjunto de validación sirve para verificar el correcto entrenamiento y funcionamiento del clasificador. Existen multitud de técnicas para realizar este proceso así como para determinar el tamaño de cada uno de los subconjuntos (*Cross-Validation, K-Fold Cross-Validation, Leave one Out, etc.*).

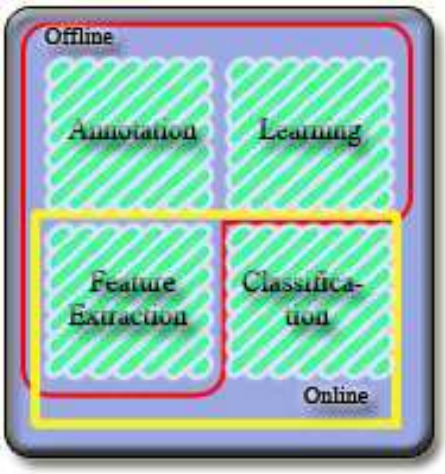

Figura 4.2: Procesos sistema de visión por computador.

Para la obtención de las muestras se ha seguido el proceso siguiente:

1. Grabación de videos mediante la plataforma desarrollada en este proyecto en un escenario virtual.

2. Decodificación del groundtruth de cada frame.

3. Recorte de cada target de cada frame mediante la información proporcionada por las cajas contenedoras.

4. Filtrado de imágenes. Solo se considerarán válidas aquellas imágenes en las que el target sea visible en un 95 % (No nos servirán imágenes de targets ocluidos en exceso) y que su altura en la imagen sea superior a 128 pixeles (targets muy lejanos a la cámara serán muy pequeños y su resolución será demasiado baja). En la figura 4.3 se muestran algunos ejemplos.

5. De las imágenes válidas y teniendo en cuenta que se extraen de una secuencia de video, se seleccionan aleatoriamente un 40 %.

6. De las seleccionadas se genera la imagen simétrica (técnica utilizada por los autores). De este modo se aumenta el conjunto de entrenamiento.

Figura 4.3: Imágenes de targets demasiado pequeños o demasiado ocluidos.

Es necesario mencionar que este proceso ha sido prácticamente automático y realizado en muy poco tiempo. El usuario solo tendrá que invertir tiempo en generar el escenario y grabar los videos. Es la principal ventaja de disponer del groundtruth de cada frame.

Para mostrar algunos datos de ejemplo, en un video de unos 8 minutos de duración con una velocidad de captura de 5 frames por segundo (de este modo hay más variabilidad entre frames) se han obtenido 1388 imágenes válidas de targets. De este total se han seleccionado aleatoriamente 555 imágenes y se ha generado su imagen simétrica obteniendo un conjunto final de 1110 imágenes.

Algunas de estas imágenes son mostradas a continuación (figura 4.4). Para el correcto entrenamiento/validación del algoritmo también son necesarias imágenes negativas, es decir, imágenes en las que no aparezca ningún target. Estas han sido generadas realizando recortes aleatorios y teniendo en cuenta que no hubiera intersecciones con ninguna caja contenedora (figura 4.5). También mostramos algunas de las imágenes completas junto a su foreground label map (figura 4.6).

Figura 4.4: Imágenes de targets.

Figura 4.5: Imágenes negativas.

4.1.2. HOG. Histograma de Orientación de Gradientes

En el campo de la visión por computador aparece a menudo el término descriptor. Un descriptor es usado para aglutinar una o un conjunto de características concretas en una imagen. Existen multitud de descriptores para representar distintos tipos de características que pueden ser útiles para la detección o reconocimiento de objetos (edges, corners, blobs, etc.). La implementación del sistema de detección ha sido realizada por el CVC, de modo externo a este proyecto, por lo que la descripción de este será meramente descriptiva.

Figura 4.6: Imágenes completas con Foreground Label Map.

El descriptor HOG está basado en el análisis de contornos mediante el cálculo de gradientes. La idea subyacente que presenta este descriptor es que la apariencia y forma de un objeto en una imagen puede ser representado por la distribución de la orientación de los gradientes. En la comunidad científica, la detección de personas ha sido desde hace tiempo un reto debido a la gran cantidad de posturas o prendas distintas que puede presentar una persona. El descriptor HOG es invariante ante transformaciones geométricas y fotométricas, por lo que es especialmente útil para la detección de figuras humanas.

La implementación del sistema se compone de los siguientes pasos.

1. **División de la imagen en celdas**. Cada celda es dividida en 9 subceldas.

2. **Cálculo de gradientes mediante derivadas parciales en la dirección horizontal y vertical.** Los gradientes nos proporcionan información sobre los contornos, es decir, nos indican el cambio de intensidad de un pixel al

pixel contiguo. Mediante la combinación de los dos gradientes se obtiene la magnitud y orientación del gradiente en cada pixel.

3. **Cálculo de histogramas.** Cada pixel contribuye a la creación de un histograma de orientación de gradientes. El histograma de la celda puede ser generado mediante los obtenidos en cada subcelda.

4. **Normalización de los histogramas.** Conjuntos de celdas contiguas (llamados bloques) pueden apoyarse para normalizar los histogramas y suavizar posibles cambios de iluminación. El descriptor HOG será el conjunto de histogramas de todas las celdas.

5. **Clasificación.** Una vez disponemos del descriptor debemos obtener el clasificador. Los autores afirman que usando el la máquina de aprendizaje conocida cómo *Lineal SVM* se obtienen buenos resultados en conjunción al descriptor HOG. Esta es capaz de representar una función de decisión mediante la definición de un hiperplano y generar un clasificador.

En la figura 4.7 se muestra el proceso completo tal y como lo describen sus autores.

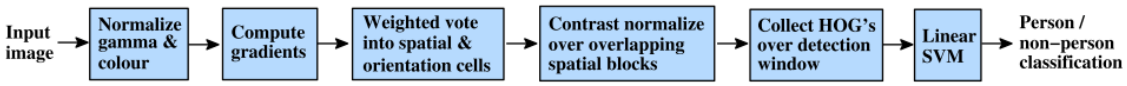

Figura 4.7: Proceso completo reconocimiento de personas (HOG + Linear SVM).

4.1.3. Resultados

Cómo se comentaba al inicio de este capítulo, se pretende averiguar si el uso de imágenes sintéticas es tan válido como el de imágenes reales para el entrenamiento y validación de sistemas de visión. Se han entrenado dos clasificadores mediante el proceso descrito en la sección anterior; uno con imágenes sintéticas y el otro con reales (figura 4.8).

Las imágenes reales usadas para entrenar se han obtenido de la base de imágenes *Daimler*. Se trata de una secuencia de video de 27 minutos en blanco y negro. Las imágenes tienen unas dimensiones de 640x480 pixeles. Las imágenes sintéticas han sido obtenidas mediante la plataforma desarrollada en este proyecto y se ha seguido el proceso descrito en la sección 4.1.1.

Con ambos clasificadores se han usado imágenes reales para la validación. Estas han sido obtenidas de la base de imágenes *Inria*. Se trata de una colección variada, en color y con distintos tamaños de imagen. Para la captura de estas

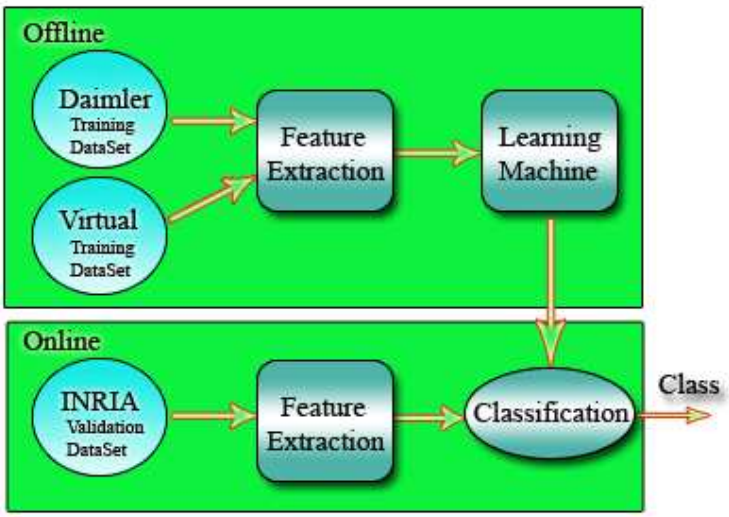

Figura 4.8: Proceso de entrenamiento y validación.

se han utilizado cámaras digitales convencionales. En la figura 4.9 pueden observarse algunas muestras.

A continuación se muestran las curvas DET (Detection Error Trade-Off) obtenidas

Figura 4.9: Imágenes Inria.

del proceso de validación de ambos clasificadores (fig. 4.10 y 4.11). Una curva DET muestra el grado de error de falsos negativos contra el grado de error de falsos positivos, variando el umbral de alguna medida de confianza. La curva nos permite establecer el punto óptimo en el que configurar esta medida, minimizando de este modo el error del sistema.

Para el proceso de detección se hace uso de una ventana que se traslada a lo largo de la imagen a analizar. De este modo se divide la imagen en regiones no disjuntas (el avance de la ventana suele ser menor que el tamaño de esta). Se extraen las características de estas subimágenes y pasan por el clasificador para ver si forman parte (o contienen) de algún target.

El umbral que varía en las curvas DET mostradas afecta al número máximo

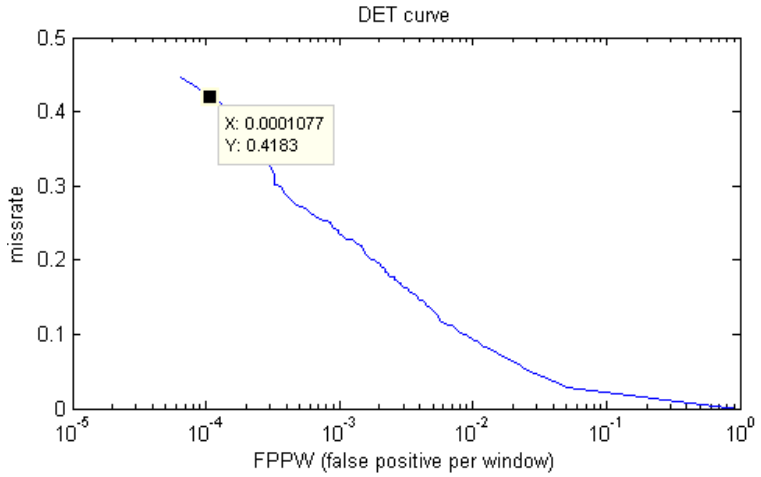

Figura 4.10: Curva DET con entrenamiento mediante imágenes reales.

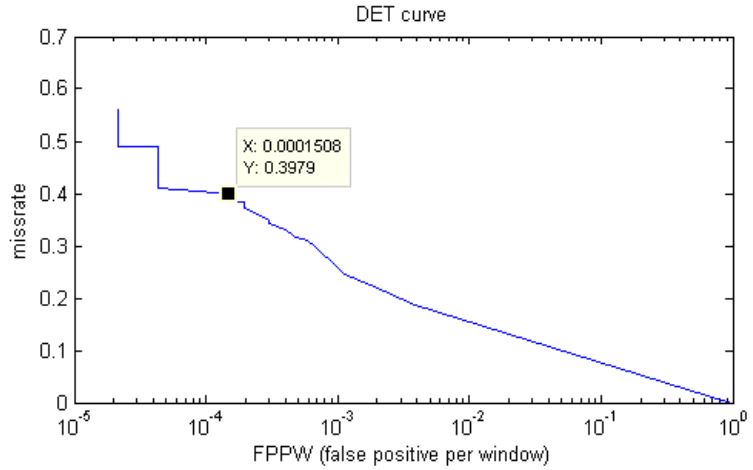

Figura 4.11: Curva DET con entrenamiento mediante imágenes sintéticas.

de falsos positivos por ventana. Permitiendo un falso positivo cada 10000 ventanas, se ha obtenido un grado de detección del 58.17 % del total de los targets entrenando con imágenes reales, y un 60.21 % entrenando con imágenes sintéticas.

En esta primera aproximación, la base de imágenes sintéticas usada no es demasiado grande y puede considerarse pobre en cuanto a variedad (Los videos se han obtenido de un mismo escenario con una iluminación única). Aun así, los resultados obtenidos han sido comparables con los obtenidos mediante la base de imágenes Daimler. Se espera obtener mejores resultados aumentando el tamaño y la variedad en la base de datos sintética. Deberán generarse un conjunto más diverso de escenarios, con distintos tipos de iluminación, y aumentando la diver-

sidad de los modelos (usar modelos con objetos o con distinto *aspect ratio*, con distinta altura o grosor, uso de modelos de niños, etc).

A continuación se muestran algunas imágenes obtenidas del proceso de validación del clasificador entrenado mediante imágenes sintéticas (fig. 4.12). Las cajas azules corresponden al groundtruth y las rojas a la respuesta del sistema de detección. Se puede observar cómo el sistema no es capaz de detectar figuras de niños o imágenes poco iluminadas.

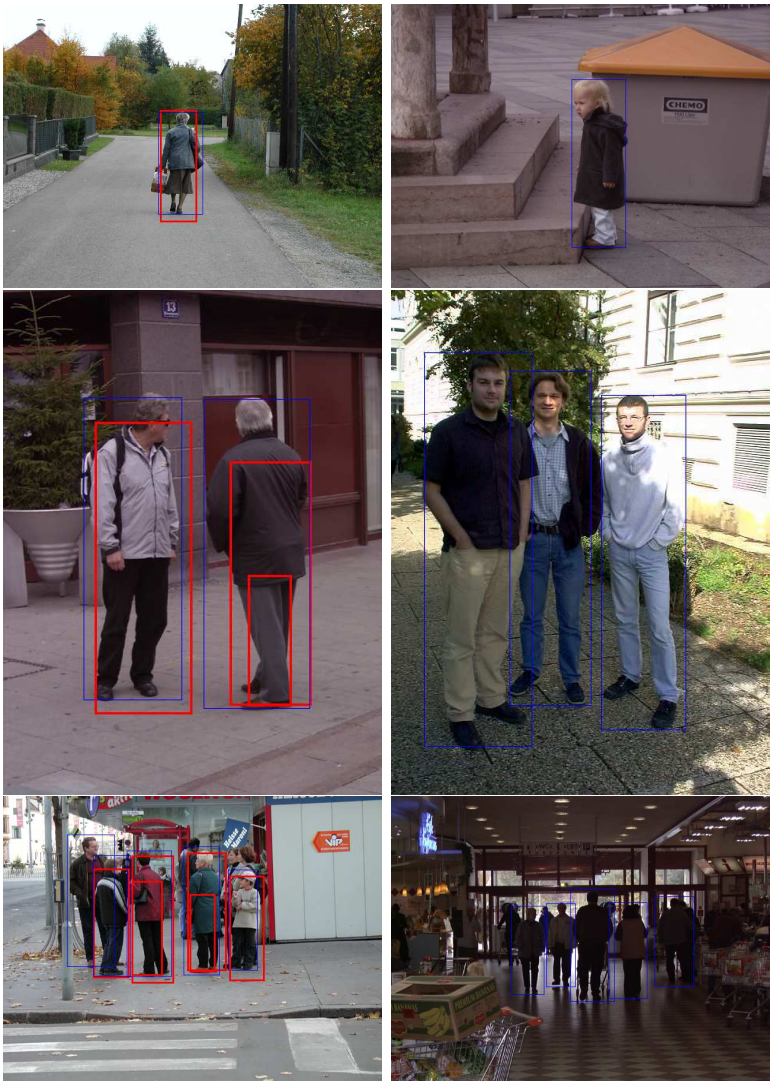

Figura 4.12: Resultados validación base de imágenes Inria.

4.2. Resumen

En este capítulo se han presentado los resultados obtenidos a partir de las imágenes generadas con la plataforma desarrollada en este proyecto. Se han mostrado los resultados de la validación de dos clasificadores, uno entrenado con imágenes sintéticas y el otro con reales, y se ha obtenido porcentajes de acierto parecidos en ambos casos. De este modo se ha demostrado que el uso de imágenes sintéticas para el entrenamiento/validación de sistemas de visión es válido, pudiendo aprovechar así el conjunto de ventajas de trabajar en un entorno virtual descritas en los capítulos anteriores.

Aunque los resultados han sido satisfactorios, se espera obtener mejores índices de detección al aumentar en tamaño y variedad la base de imágenes sintética.

Capítulo 5

Conclusiones

El uso de imágenes sintéticas para el entrenamiento/validación de algoritmos de visión por computador es una práctica que está comenzando a extenderse. En el CVC se quiso comprobar la utilidad de este tipo de imágenes y las posibles limitaciones que podrían presentar.

En el primer capítulo, comentábamos que este proyecto se había basado en un proceso iterativo y habíamos definido un conjunto de actividades básicas que había que completar. En función de los resultados obtenidos en este primer ciclo y del tiempo restante se realizarían actividades adicionales para complementar este proyecto. Debido a que los primeros resultados fueron satisfactorios y que se realizaron segun la planificación inicial, estas tareas adicionales pudieron llevarse a cabo. Las listamos a continuación:

- Implementación de una GUI para el uso de la plataforma. Aunque esta era una tarea adicional, se comenzó a desarrollar en paralelo a las actividades básicas.
- Control libre de la cámara mediante interacción con el ratón. En la planificación inicial se estableció que el recorrido de la cámara fuera prefijado.
- Obtención de métodos para compilar/descompilar modelos 3D propios (no incluidos por Valve en el juego).
- Obtención de métodos para obtener el fichero fuente de mapas ya compilados, siendo posible así su modificación y uso con la plataforma.
- Obtención de conocimientos adicionales en la edición de escenarios para aumentar el grado de realismo, como iluminación o efectos meteorológicos.
- Implementación de funcionalidades adicionales.

- Instanciación de parejas de cámaras dentro del escenario para la obtención de videos para el entrenamiento/validación de algoritmos de visión estereoscópica.
- Grabación del recorrido efectuado por la cámara (*PathTracking*). El usuario podrá definir el camino a seguir por la cámara y obtener videos de modo automático.

Todos los objetivos listados han sido realizados con éxito, a excepción del *PathTracking*. El objetivo de esta funcionalidad era automatizar la captura de videos y predefinir un camino para obtener videos idénticos variando elementos del mapa como la iluminación o efectos meteorológicos, o haciendo uso de los efectos ópticos de la herramienta OVVV. Se descartó por el hecho de que dicha herramienta y la plataforma desarrollada en este proyecto funcionan de modo independiente, y no era posible sincronizarlas. El mismo recorrido de la cámara en instantes distintos capturaría videos distintos, ya que los elementos dinámicos (personas, vehículos, etc.) estarían en distintas posiciones.

A continuación mostramos una estimación de las horas dedicadas a cada una de las fases de este proyecto, y en la figura 5.1 mostramos la planificación inicial. Debido a que este proyecto se ha realizado en conjunción a la vida académica y laboral, la dedicación diaria no ha sido completa. La dedicación a cada fase se aproxima más a las estimaciones detalladas a continuación. El diagrama es meramente orientativo para observar las distintas actividades y apreciar el inicio y fin de cada una.

- **Instalación herramienta OVVV, Hl2, Hammer y pruebas varías:** 6 horas.
- **Iniciación en la edición de mapas. Conceptos básicos:** 40 horas.
- **Investigación herramientas adicionales:** 10 horas.
- **Implementación plataforma. Funcionalidades básicas:** 30 horas.
- **Implementación aplicación para extracción del groundtruth:** 15 horas.
- **Aprendizaje conceptos adicionales de la edición de mapas:** 50 horas.
- **Implementación plataforma. Funcionalidades adicionales:** 80-100 horas.

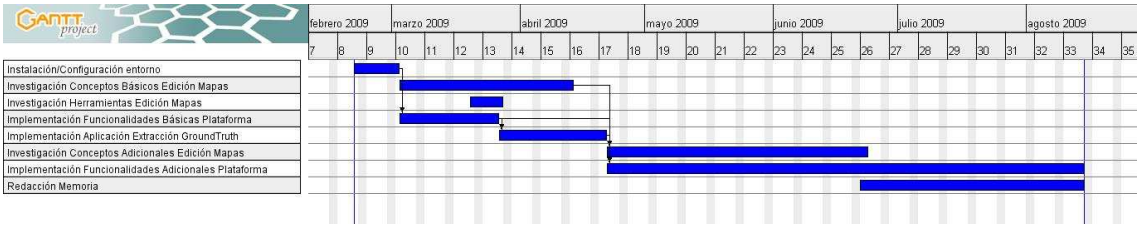

Figura 5.1: Planificación Proyecto.

- **Redacción memoria:** 80-90 horas.

Algunas de las características principales que dispone la plataforma a finales de la realización de este proyecto son:

- Intuitiva interfaz gráfica para un uso más cómodo de la plataforma.
- Libre movimiento por el escenario.
- Uso de los efectos ópticos facilitados por la herramienta OVVV.
- Instanciación de la cámara en posiciones concretas del escenario mediante parámetros. Posibilidad de convertir la cámara a una cámara activa. Uso de zoom.
- Instanciación de cámaras estereoscópicas. Además de las funcionalidades de una cámara simple, posibilidad de establecer parámetros de calibración (distancia entre cámaras, campo de visión, orientación, etc.).

El proceso seguido para el desarrollo de este proyecto se detalla a continuación:

- Lo esencial en la etapa inicial de este proyecto fue investigar en el estado del arte y encontrar las herramientas que podrían ser útiles para los fines de este proyecto. Fue entonces cuando se encontró la herramienta OVVV. Ya que se decidió realizar el proyecto apoyándose en el uso de esta herramienta, la primera tarea era comprobar el uso y correcto funcionamiento de esta. Aunque se obtuvo una idea general de lo que ofrecía la herramienta OVVV, el total de posibilidades se observó cuando se comenzó a implementar la plataforma.
- Una vez comprobado y asimilado el modo de usar la herramienta OVVV, comenzó la fase de investigación en el mundo del *mapping*. Era necesario obtener los conocimientos necesarios para poder generar un mapa sencillo y poder instanciar personajes en este. Durante el transcurso de esta fase se tuvo constancia de la existencia de distintas herramientas que podrían facilitar la generación de escenarios. No se pretendía profundizar en el mundo del mapping, solo abstraer los conceptos básicos para generar un escenario.

- Una vez ya se disponía de un escenario y se conocía el modo de instanciar y animar personajes, se comenzó a implementar la plataforma. La interfaz gráfica de la plataforma se empezó a desarrollar al inicio de esta fase, junto con las funcionalidades básicas de esta. Cuando se finalizó esta etapa la plataforma era capaz de instanciar una cámara en el escenario, moverla y obtener el flujo de video asociado.

- Con las funcionalidades básicas de la plataforma realizadas, se obtuvieron los primeros videos. Se observó que los datos del groundtruth venían codificados en un archivo binario para ofrecer una transmisión más rápida de estos desde el servidor de cámaras. Fue necesario entonces implementar una aplicación para la decodificación de estos datos. Se decidió implementarla en Matlab debido a que es un lenguaje muy práctico para prototipar, y ya que la extracción del groundtruth de un video solo tenía que realizarse una única vez, el tiempo no era un factor crítico.

- Llegado a este punto ya se habían realizado los objetivos básicos definidos en la etapa inicial del proyecto. Se decidió entonces profundizar en nuevas ideas que habían ido surgiendo durante las fases anteriores. Cuando se realizó la iniciación en el mundo del mapping se observó el gran abanico de posibilidades que había detrás de este y se decidió investigar un poco más. Se buscaron formas y soluciones para mejorar el grado de realismo de los escenarios. Se encontraron métodos más realistas para animar a los personajes, cómo definir la iluminación y varios conceptos más.

- Además de profundizar un poco más en el mapping, se decidió aumentar el número de funcionalidades de la plataforma. Fue entonces cuando se desarrolló el uso de los efectos ópticos, la conversión a cámara activa, las cámaras estereoscópicas y demás funcionalidades de la plataforma.

- Finalmente, en paralelo a las últimas dos actividades se redactó la memoria para documentar este proyecto.

Este proyecto ha tenido una fuerte componente en desarrollo y otra en investigación. Se ha invertido mucho tiempo en cada una de ellas. Se ha pretendido aprovechar todas las posibilidades ofrecidas por la herramienta OVVV, así como crear otras que no estaban contempladas en esta.

Los conceptos explicados en esta memoria sobre el mapping podrían ser ampliados. Es posible que el usuario final pueda encontrar aspectos interesantes en la wiki de Valve [12] sobre el diseño de escenarios que no hayan sido comentados en esta memoria. Se ha pretendido exponer los principales elementos sobre el diseño que podrían ser de utilidad para este proyecto.

Observando los resultados presentados en el capítulo 4, se puede afirmar que el entrenamiento de sistemas de visión por computador mediante imágenes sintéticas es igual de válido que con imágenes reales. A través de la plataforma desarrollada en este proyecto y haciendo uso de la herramienta OVVV, se ha obtenido un método cómodo y eficaz para generar dichas imágenes. Se espera que los resultados obtenidos mediante imágenes sintéticas mejorarán al generarlas en un mayor número de escenarios y aumentando el grado de variedad entre los modelos.

Por cuestiones de tiempo, ha habido algunas funcionalidades que no han podido desarrollarse. A continuación comentaremos un par de puntos que podrían complementar la plataforma:

- ***Definición de targets arbitrarios.*** Se ha invertido mucho tiempo investigando el modo en que la herramienta OVVV decide si un modelo es un target a detectar, pero no se ha llegado a obtener resultados concluyentes. Se han realizado pruebas a nivel de la herramienta OVVV, a nivel del editor de escenarios y a nivel de modelo 3D. Si este punto se hubiera podido realizar la plataforma se habría generalizado, aumentando el abanico de aplicaciones para las que podría ser útil. Se habría podido orientar no solo a la detección y seguimiento de personas, si no a la de cualquier clase de objeto.

- ***Aplicación para construir imagen de profundidad para visión estereoscópica, a nivel de bloque y modelo.*** Como se comenta en el capítulo 2, para la validación de sistemas de visión estereoscópica se requiere un mapa de profundidad indicando la distancia de cada pixel en relación a la cámara. Dicho mapa podría construirse realizando una proyección desde la posición de la cámara y hacia la dirección dónde esté orientada dentro del escenario. En los ficheros fuente de los escenarios se indican los distintos planos que definen los bloques. Buscando la intersección con el plano más cercano para cada pixel, podría generarse el mapa de profundidad. Los ficheros fuente de los modelos 3D contienen de igual modo la información de cada uno de los planos que los componen. Debido a que obtenemos la información a partir de ficheros fuente (Conocemos la posición y orientación de los bloques y de los modelos 3D en el instante inicial), todos los modelos insertados en el escenario deberán ser estáticos.

A continuación listamos algunas posibles aplicaciones para los que la plataforma desarrollada en este proyecto podría ser de utilidad.

- Detección de personas con cámara estática.

- Detección de personas con cámara de video vigilancia (cámara activa. PTZ).

- Detección de personas en plataformas móviles (*travelling*).
- Detección de acciones humanas diversas.
- Detección facial.
- Visión estereoscópica.

Este listado es tan solo un conjunto de ejemplo. Probablemente haya más aplicaciones para los que esta plataforma podría ser de utilidad.

Para acabar es importante recordar las ventajas al utilizar escenarios virtuales. El editor de escenarios nos proporciona un control absoluto sobre el entorno. Podremos generar todo tipo de exteriores o interiores, en cualquier instante del día y con cualquier tipo de iluminación. Podremos insertar un gran número de elementos variados en el escenario, ya sean objetivos a detectar o no. El control es completo y el grado de variabilidad tan alto como se quiera.

Bibliografía

[1] [cited at p. -]

[2] *Intelligent vehicle technologies and Trends.* Artech House Inc. [cited at p. 9]

[3] *Multiple View Geometry in Computer Vision.* Cambridge University Press, 2nd Edition. [cited at p. -]

[4] *Pattern Recognition and Machine Learning.* Springer, 2006. [cited at p. -]

[5] K. Murphy A. Torralba and W. Freeman. Sharing visual features for multiclass and multiview object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007. [cited at p. -]

[6] Angel D. Sappa D. Gerónimo, Antonio M. López and Thorsten Graf. Survey on pedestrian detection for advanced driver assistance systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009. [cited at p. 10]

[7] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. [cited at p. 49]

[8] M. Enzweiler and D. M. Gavrila. Monocular pedestrian detection: Survey and experiments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2008. [cited at p. 10]

[9] C. Hilario. *Detección de Peatones en el Espectro Visible e Infrarrojo para un Sistema Avanzado de Asistencia a la Conducción.* PhD thesis, Departamento de ingenierías de sistemas y automática, Universidad Carlos III de Madrid, 2008. [cited at p. -]

[10] C. Hilario, J. M. Collado, J. Mª Armingol, and A. de la Escalera. Detección de peatones para vehículos inteligentes basada en modelos de contornos activos y visión estéreo. [cited at p. 23]

[11] C. Hilario, J. M. Collado, J. Mª Armingol, and A. de la Escalera. Visión por computador para vehículos inteligentes. [cited at p. -]

[12] http://developer.valvesoftware.com/wiki/Category:Level_Design. Hammer level design wiki. [cited at p. 62, 86]

[13] http://vision.middlebury.edu/stereo/index.html. Stereoscopic vision. [cited at p. 24]

[14] B. Schiele P. Dollár, C. Wojek and P. Perona. Pedestrian detection: A benchmark. *IEEE Conf. on Computer Vision and Pattern Recognition*, 2009. [cited at p. 15]

[15] C. Papageorgiou and T. Poggio. A trainable system for object detection. *International Journal of Computer Vision*, 2000. [cited at p. 3]

[16] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. *IEEE Conf. on Computer Vision and Pattern Recognition*, 2001. [cited at p. 50]

[17] Miguel Ángel Sotelo Vázquez. Tratamiento de imagen aplicado a sistemas inteligentes de transporte. [cited at p. -]

Appendices

Apéndice A

Hammer y modelado 3D

En este apéndice, *comentaremos todos los conceptos necesarios para poder crear o modificar un escenario mediante el editor Hammer. El mundo de la edición de mapas es muy amplio, por lo que no se comentarán todas sus características y posibilidades. Se hablará sobre las herramientas básicas así como del entorno del editor Hammer, del proceso a seguir para descompilar mapas, cómo introducir personajes dentro del escenario y cómo definir la iluminación.*

Se pretende presentar este apéndice como un manual con la secuencia de pasos a seguir para generar y configurar diversos aspectos de un escenario virtual. Los pasos detallados sintetizan la información más relevante de toda la obtenida durante la fase de investigación de diversas fuentes en Internet y de la experiencia adquirida. Se recomienda que futuros usuarios de la plataforma desarrollada en este proyecto presten especial atención a este apéndice.

A.1. Introducción a la edición de escenarios. Conceptos Básicos

A.1.1. Qué es un modelo 3D

El término *modelo 3D* está contenido dentro del mundo de los gráficos por computador. Los gráficos por computador pueden considerarse, desde el punto de vista técnico, como un conjunto de fórmulas matemáticas que describen un entorno en 3 dimensiones.

Un modelo 3D es una representación esquemática visible a través de un conjunto de objetos, elementos y propiedades. Suele construirse a partir de objetos poligonales básicos, como triángulos, cuadrados o esferas. Las propiedades con

las que cuenta pueden ser efectos de iluminación, texturizado, transparencias, animaciones, etc. Aunque estos modelos se construyen a partir de propiedades matemáticas, estas suelen dejarse de lado para dar paso a editores visuales, ya que lo que este modelo de representación ofrece es algo mucho más parecido al resultado que se pretende obtener.

Hoy en día, existen multitud de aplicaciones de modelado 3D, ofreciendo un entorno y conjunto de herramientas apropiado a los diseñadores. La obtención de conocimientos en modelado 3D no forma parte de los objetivos del proyecto, por lo que a partir de este punto, se supondrá que ya se dispone de los modelos 3D necesarios o los conocimientos para generarlos.

En el siguiente apartado detallaremos el proceso a seguir para importar modelos propios al editor de escenarios Hammer.

A.1.2. Importación de modelos 3D en el editor Hammer

El proceso que se deberá llevar a cabo se representa en la siguiente figura A.1.

Necesitaremos la aplicación de modelado 3D llamada *XSI Mod Tool*. Esta apli-

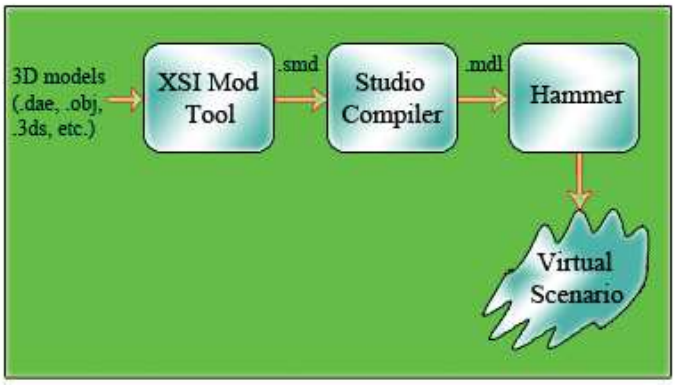

Figura A.1: Proceso de compilación de un modelo 3D.

cación nos será de especial utilidad, ya que incluye un plugin para poder exportar los modelos a formato *smd* (el formato fuente de los modelos 3D que usa Valve). Por lo tanto deberemos instalar esta aplicación, junto con todas sus actualizaciones. No es necesario diseñar los modelos con este editor. Solo hay que tener en cuenta que el editor que se use permita exportar los modelos a un formato que la aplicación XSI Mod Tool sea capaz de importar. Esta aplicación soporta los formatos más habituales, por lo que esto no debería de suponer un problema.

Lo primero que haremos será importar el modelo al editor. Para ello usaremos la herramienta de importación de la que dispone, Crosswalk (File ↦ Crosswalk ↦ Import). Durante el proceso de importación, el modelo pierde su tamaño original, por lo que es probable que tengamos que escalarlo.

Una vez realizadas las modificaciones necesarias, exportaremos el modelo (ValveSource ↦ Export SMD). Obtendremos el modelo en un archivo con formato smd.

A continuación, deberemos compilar el archivo smd para obtener el modelo en formato mdl, el cual es el usado por el editor de escenarios. Para el proceso de compilación utilizaremos una herramienta llamada *StudioCompiler*. Aunque su uso es muy intuitivo mostraremos un ejemplo.

El proceso de compilación lo realizaremos en dos pasos. En primer lugar compilaremos las texturas del modelo, y a continuación el propio modelo.

Para compilar las texturas debemos ir a la pestaña Material Compile. Es necesario que las texturas estén en formato TGA, si no lo están deberemos convertirlas a dicho formato. En la figura A.2 mostramos como debe quedar.

A continuación compilaremos el modelo. Debemos dirigirnos a la pestaña Model Compile. Debemos especificar la ruta hacia el modelo y la ruta hacia el directorio models de la carpeta del Hl2. También especificaremos que el modelo sea de tipo prop_dynamic. Finalmente indicaremos que también utilice el modelo de referencia para generar los archivos de secuencias y de colisiones. En la figura A.3 mostramos el aspecto que debería tener.

Después de realizar todos los pasos ya tendremos compilado nuestro modelo y estará disponible para ser instanciado desde el editor de escenarios. Esta misma herramienta es capaz de realizar el proceso inverso y descompilar un modelo 3D, es decir, obtener el fichero fuente a partir del ya compilado.

A.1.3. Edición de mapas ya compilados

Debido a la gran cantidad de seguidores que hay de los juegos de Valve, y de la versatilidad que ofrece el editor Hammer, existe todo un mundo sobre lo que se conoce como *mapping*. Multitud de jugadores colaboran con la comunidad diseñando mapas para el disfrute de todos. Estos *mappers* suelen tener una gran destreza con el editor de escenarios, por lo que los mapas que generan suelen ser de alta calidad y gran realismo.

Para la correcta realización de este proyecto será necesario disponer de mapas

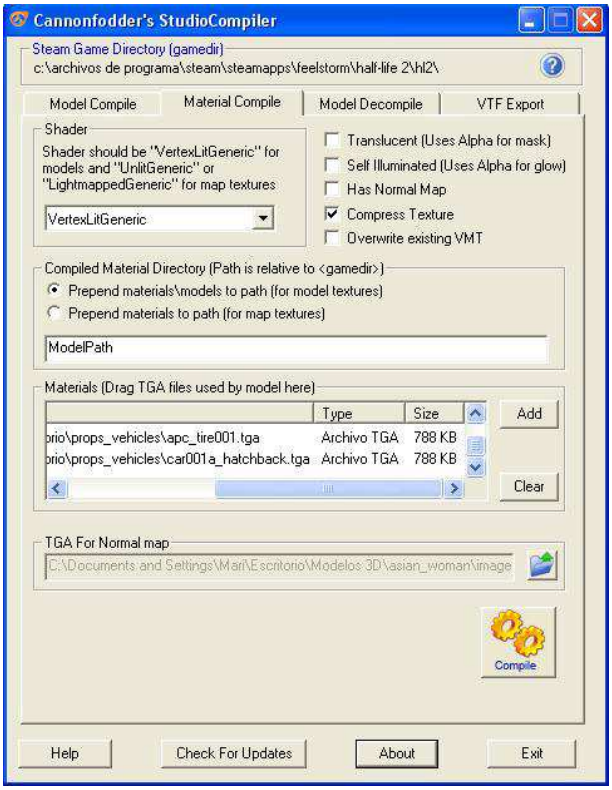

Figura A.2: StudioCompiler - Compilación de texturas.

realistas, pero por razones de tiempo, convertirse en un experto del editor de escenarios no forma parte de los objetivos a cumplir. Por lo tanto sería conveniente poder hacer uso de estos mapas al alcance de todos.

Existen dos tipos de formato para los mapas. El formato *vmf* y el formato *bsp*. El primero es el mapa fuente, editable desde el Hammer. El segundo es el mapa obtenido como resultado de compilar el mapa fuente. El mapa compilado no es modificable desde el Hammer, solo puede ser cargado en el juego.

La mayoría de mappers solo comparten con el resto de la comunidad los mapas compilados, y no los fuentes. Esto se lleva a cabo para evitar que terceras personas se aprovechen del trabajo realizado por estos diseñadores, y aplicando unas mínimas modificaciones se proclamen como autores del mapa. Por suerte, existen herramientas para obtener el mapa fuente a partir del compilado, ya que estos suelen ser una fuente de aprendizaje para editores noveles. En esta sección explicaremos como hacer uso de estas herramientas.

En primer lugar usaremos la herramienta *Vmex*. Esta herramienta está imple-

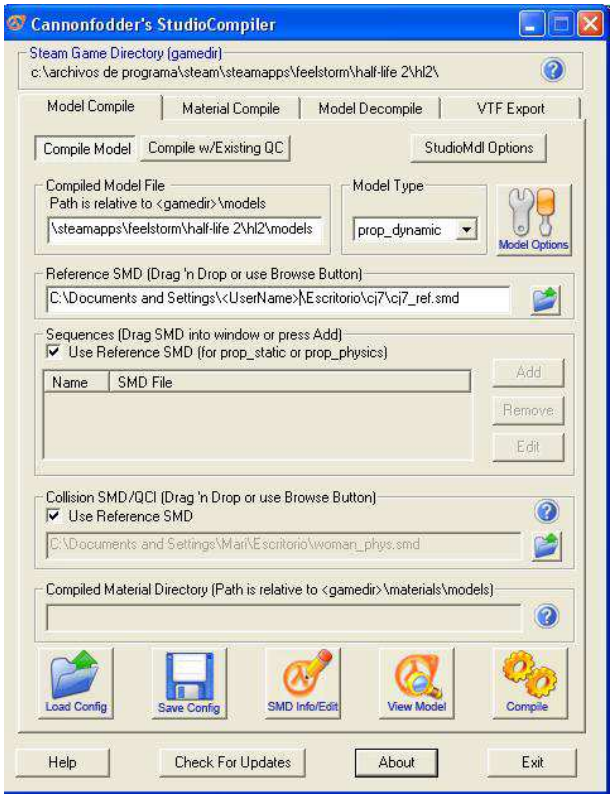

Figura A.3: StudioCompiler - Compilación de modelo.

mentada en Java, por lo que será necesario disponer de la máquina virtual de Java instalada, junto con sus actualizaciones. La configuración por defecto que lleva la herramienta ya nos servirá. Lo único que habrá que hacer es seleccionar el mapa compilado objetivo. La herramienta procesará el fichero binario y extraerá toda la información sobre las entidades y bloques del mapa, generando finalmente el mapa en formato fuente.

En ciertos mapas, después de este primer paso ya podríamos empezar a editarlos a nuestro antojo, pero los mapas más realistas suelen incluir texturas personalizadas. Estas texturas van incrustadas en el mapa compilado y no están de serie en el directorio del juego. Si el mapa contiene este tipo de texturas, al cargar el mapa descompilado en el editor veremos que las texturas no se cargan correctamente.

Para poder añadir estas texturas a nuestra colección, deberemos usar otra herramienta llamada *GCFScape*. Esta herramienta nos permitirá ver todo el contenido incluido en el archivo del mapa, ya sean texturas, modelos o incluso sonidos. Lo único que deberemos hacer será añadir el contenido del mapa al

directorio del juego, de manera que ya dispondremos de todo este nuevo material desde el editor de escenarios.

A.1.4. Añadir una entidad

En este apartado vamos a explicar cómo insertar, configurar y definir un movimiento a nuestros modelos 3D dentro de un escenario. Es necesario mencionar que este es un método general para insertar cualquier modelo. Más adelante se presentarán métodos más específicos para la inserción de personas y vehículos de manera más realista.

Lo primero que debemos hacer es insertar una entidad. Hay 3 entidades que nos serán útiles:

- ***prop_static***
- ***prop_dynamic***
- ***prop_physics***

Estas 3 entidades tienen propiedades comunes a la vez de propias, así que tendremos que usar cada una de ellas en la situación apropiada. Seleccionamos la herramienta de inserción de entidades (Shift+E) y seleccionamos el tipo de entidad, en este caso prop_dynamic.

Una vez hecho esto, mediante la vista 3D insertamos la entidad en el esce-

Figura A.4: Entity Tool y entidad prop_dynamic.

nario. A continuación abriremos la hoja de propiedades de la entidad. Para hacer esto debemos seleccionar la herramienta de selección, y hacer doble click sobre la entidad. Se nos mostrará una ventana como la de la figura A.5.

Lo primero que debemos hacer es elegir el modelo 3D para la entidad. Seleccionamos la entrada World Model y hacemos click en browse. Se nos mostrará el explorador de modelos 3D (fig. A.6), con el que podremos seleccionar el modelo deseado.

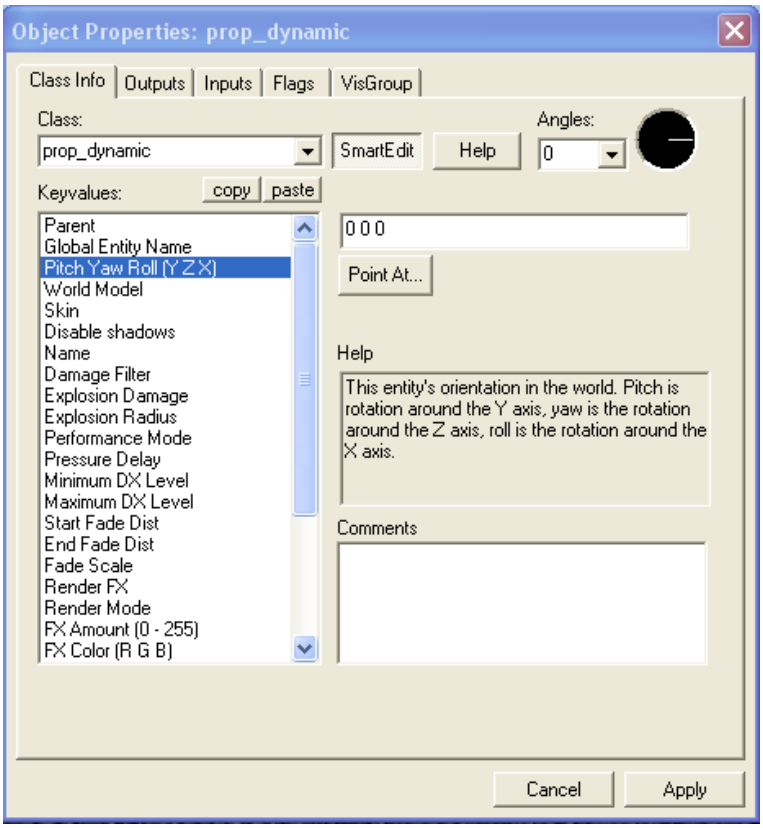

Figura A.5: Hoja de propiedades de una entidad.

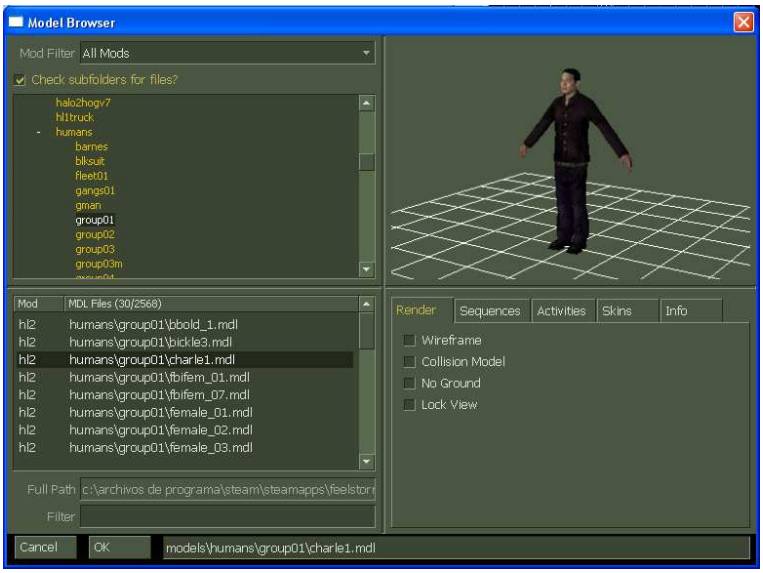

Figura A.6: Explorador de modelos.

En este punto es importante remarcar dos cosas.

1. Comprobar si el modelo 3D está compilado para el tipo de entidad que hemos elegido (prop_static, prop_dynamic o prop_physics). Esto se puede ver seleccionando la pestaña INFO. En caso de que el modelo este compilado para uno de los otros dos tipos de entidad, deberemos cambiar el tipo de nuestra entidad.

2. Averiguar el nombre que tiene la animación que querremos que tenga el modelo. Para ello, seleccionamos la pestaña Sequences y buscamos la animación que nos interese.

Opcionalmente, algunos modelos contienen más de una skin. Esto es algo que también nos podría interesar para aumentar la variedad de modelos en el escenario, por lo que es recomendable comprobar esta pestaña.

A continuación, configuraremos el resto de propiedades.

1. Debemos darle un nombre a la entidad. Seleccionamos la entrada Name y escribimos el nombre que le queramos dar a la entidad. El nombre tiene que ser único. Es recomendable asignar nombres relacionados con la entidad o definir una notación propia (npcX, modelX, etc.) para que sea más sencilla su posterior identificación.

2. Ahora estableceremos la animación para la entidad. Vamos a la entrada Default Animation, e introducimos el nombre de la animación que queremos en el campo de texto.

3. Por último puede interesarnos desactivar las colisiones de la entidad. Para ello seleccionamos la entrada Collisions y establecemos el valor Not Solid.

Una vez hemos llegado a este punto ya tendremos nuestra entidad correctamente configurada, aunque por el momento aun será estática. En los siguientes puntos definiremos el movimiento.

A.1.5. Definición Pathtracks

Ahora lo que debemos hacer es definir el camino que seguirá nuestra entidad. Seleccionamos la herramienta de selección de entidades y elegimos la entidad *path_track*.

Insertamos el primer pathtrack en el escenario mediante la vista 3D. Lo primero que debemos hacer es definir un nombre para el pathtrack. En este caso también

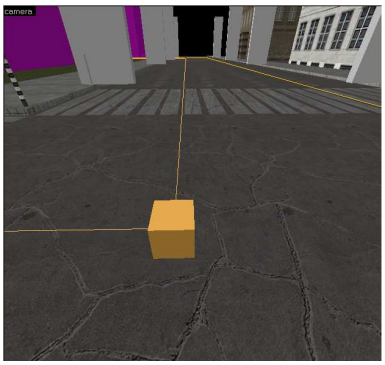

Figura A.7: Entidad path_track.

es recomendable establecer una notación concreta para este tipo de entidades.

Una vez establecido el nombre del pathtrack nos dispondremos a insertar el segundo nodo del camino. Para hacer la tarea más sencilla es recomendable usar la técnica del *drag & drop*. Manteniendo pulsada la tecla Shift, seleccionaremos el pathtrack en una de las vistas (es recomendable la vista de planta) y manteniendo pulsado, arrastraremos el cursor hasta donde queramos establecer el siguiente nodo. De esta manera quedará establecida la relación entre los dos nodos. Si no hemos realizado el paso de establecer el nombre del primer nodo, este método no funcionará. Para el resto de nodos, el nombre se establece automáticamente mediante un contador.

De esta manera iremos definiendo el camino que seguirán los modelos 3D (fig. A.7). Un nodo solo puede tener otro nodo como siguiente, por lo que no podremos definir caminos múltiples. Lo recomendable es formar un circuito cerrado, conectando el último nodo con el primero. Esta última relación deberá establecerse manualmente. Deberemos abrir la hoja de propiedades del último nodo del circuito y establecer la entrada *Next Stop Target* con el nombre del primer nodo.

Es importante mencionar, que aunque la velocidad que tomaran los modelos se definirá en el siguiente apartado, esta se puede cambiar en cuanto se alcanza uno de los nodos del camino. Para realizar esto debemos ir a la entrada *New Train Speed*. Mediante esta entrada se definirá la velocidad que tomará el modelo una vez alcance este nodo. Por defecto el valor esta a 0, indicando que la velocidad no será modificada.

También hay que remarcar que un nodo del camino solo puede ser el punto

inicial de un modelo. Si establecemos que varios modelos tengan el mismo punto de partida estos se solaparán. Por lo tanto, el camino debería contener, como mínimo, tantos nodos como modelos seguirán el recorrido.

A.1.6. Asociar una entidad a un Pathtrack

En este punto es cuando definiremos el movimiento. Lo primero que debemos hacer es seleccionar la herramienta de creación de bloques, y a continuación definir un bloque mediante una de las 3 vistas secundarias. El bloque no tiene porque ser muy grande, podemos considerarlo como una base que soportará al modelo.

Una vez definido el bloque, debemos convertirlo a un tipo de entidad especial. Con el bloque seleccionado, pulsaremos *Ctrl + T*. Se nos mostrará una hoja de propiedades. Debemos seleccionar la opción *func_tracktrain*.

Esta nueva entidad será la que proporcionara el movimiento al modelo. El primer paso que debemos realizar es configurarla correctamente. En primer lugar, mediante la entrada Name, estableceremos un nombre con el que poder referenciar la entidad. A continuación, mediante la entrada Render, estableceremos la opción de *No Render*. Mediante la entrada *First Stop Target*, estableceremos el nodo a partir del cual empezará a recorrer el camino. Debemos introducir el nombre de uno de los pathtracks definidos previamente. Y por último, estableceremos la velocidad que tomará la entidad mediante la entrada *Initial Speed*.

Lo único que queda por hacer es establecer la relación entre el modelo y la entidad func_tracktrain. Para realizar este paso, debemos abrir la hoja de propiedades del modelo y configurar la entrada *Parent* con el nombre de la entidad func_tracktrain. El modelo heredará el movimiento realizado por la entidad padre. Por lo tanto, es recomendable colocar el modelo justo en el centro de la entidad padre, ya que si no el camino que seguirá no será el esperado. De igual modo que con los nodos, si queremos evitar solapamientos, una entidad func_tracktrain solo podrá servir para un modelo.

Si hemos realizado correctamente todos los pasos, cuando carguemos el mapa veremos cómo nuestro modelo sigue el camino que hemos definido.

A.1.7. Inserción de personajes más realistas.

Hay otro método que nos puede interesar a la hora de insertar personas en nuestro escenario. Con este método haremos uso de unas entidades dirigidas a representar peatones en un escenario. Su movimiento será mucho más realista

que con el método anterior.

Podremos usar cualquiera de las 4 entidades siguientes. La razón de usar estas 4 es porqué son las únicas que nos permiten establecer el modelo a usar. Hay algunas pequeñas diferencias entre estas entidades, pero no nos afecta para lo que pretendemos realizar, así que las obviaremos.

- ***npc_alyx***
- ***npc_breen***
- ***npc_eli***
- ***npc_kleiner***

Cuando insertemos alguna de estas entidades solo deberemos preocuparnos de elegir el modelo y asociar el camino. Con estas entidades no servirán los caminos definidos anteriormente. Deberemos usar una entidad distinta llamada *Path Corner* para definir el camino. El funcionamiento es idéntico que con la entidad path_track. Para seleccionar la animación que representará el modelo debemos dirigirnos a la pestaña *Model* de la hoja de propiedades de este (fig. A.8).

A.1.8. Inserción de vehículos

De igual manera que con las personas, también tenemos entidades concretas para el uso de vehículos dentro de nuestro escenario. Nos basaremos en las dos entidades siguientes:

- ***Prop_vehicle_apc:*** Entidad con el modelo del vehículo.
- ***Npc_vehicledriver:*** Entidad que controla el vehículo a través del camino asociado.

Haciendo una analogía con el método explicado anteriormente podemos comparar la entidad prop_vehicle_apc con la entidad prop_dynamic y la entidad npc_vehicledriver con la entidad func_tracktrain.

Los aspectos a configurar son los mismos que los del método anterior, únicamente hay dos excepciones:

- El camino deberá estar formado mediante la entidad Path Corner.
- La entidad prop_vehicle_apc tiene asociado un script para el control del vehículo. Mediante este script se definen un conjunto de parámetros para

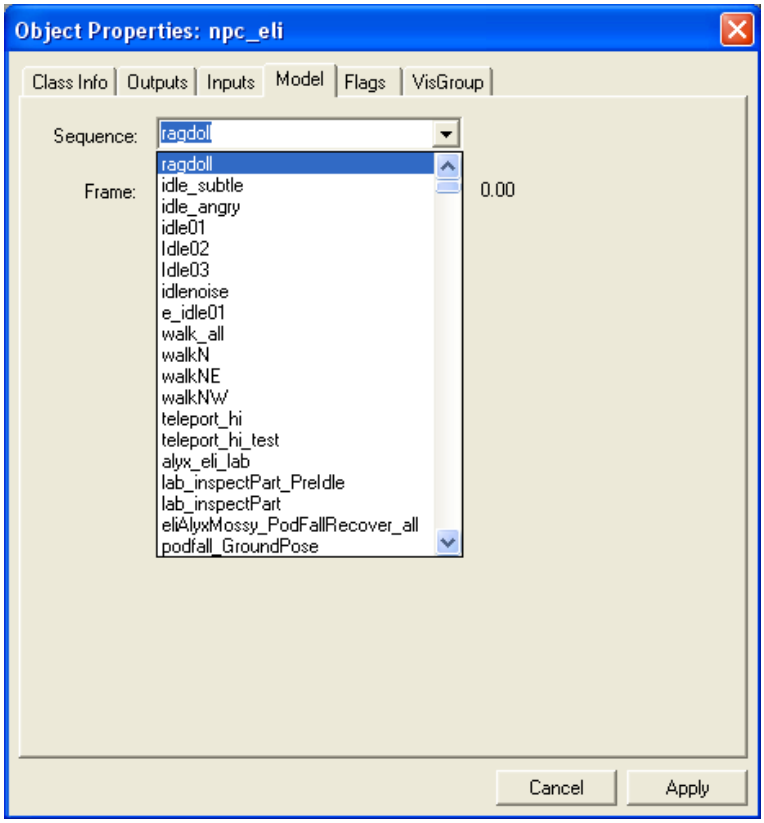

Figura A.8: Hoja de propiedades. Pestaña model.

simular un comportamiento realista y una física apropiada para el vehículo. Aunque Valve ya proporciona un conjunto de scripts para distintos tipos de vehículos (Deportivos, todoterrenos, etc.) estos se pueden modificar para obtener comportamientos más personalizados.

Por último, hay que comentar que es recomendable añadir instancias de la entidad info_node (fig A.9) a lo largo del camino. Estas entidades proporcionan información del camino a las entidades que lo recorren. Es altamente recomendable añadir diversas instancias de estas entidades en las curvas de los caminos para mejorar y obtener trayectorías más realistas.

A.1.9. Iluminación

A continuación comentaremos los pasos a seguir para definir la iluminación en nuestro escenario. Para hacer efectivos los distintos cambios realizados en la iluminación desde el editor de escenarios deberemos activar las opciones *VIS* y *RAD* en el proceso de compilación (fig. A.10). Si estas opciones no son activadas, en la compilación no se realizan los cálculos para la iluminación. Ya que al activar estas opciones el tiempo del proceso de compilación aumenta, se recomienda

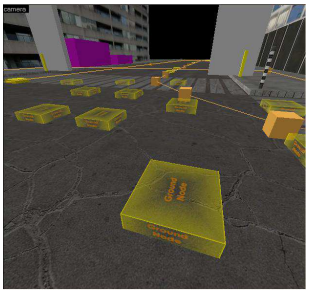

Figura A.9: Entidad info_node.

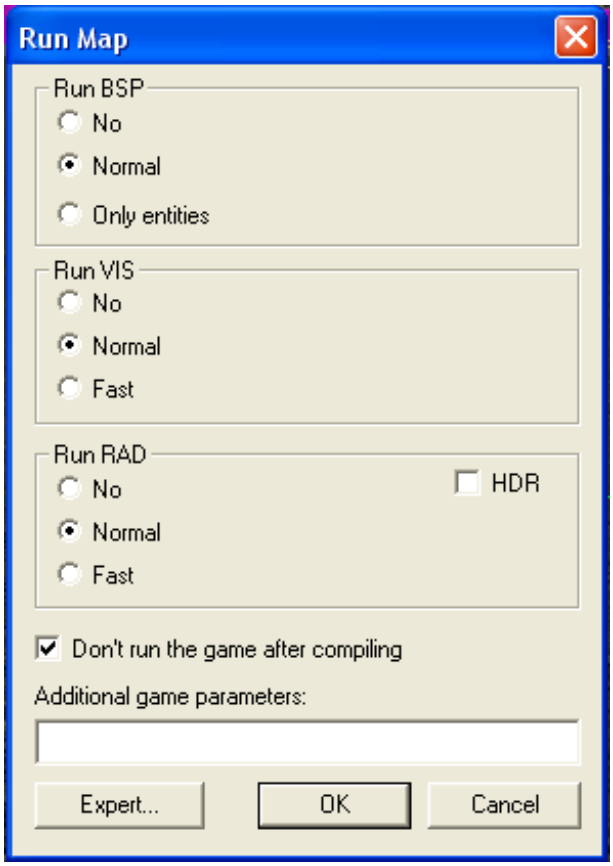

Figura A.10: Opciones de compilación.

utilizar la opción *fast* cuando se estén realizando cambios en la iluminación. Una vez se ha obtenido la iluminación deseada, los cálculos para esta pueden omitirse del proceso de compilación. Podremos realizar cambios en el resto de entidades del escenario manteniendo la iluminación definida.

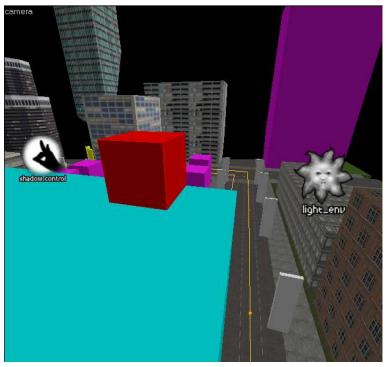

Figura A.11: Entidades light_environment, env_sun y shadow_control.

Iluminación. Luz Ambiente

En este punto vamos a definir como insertar en el escenario un foco de luz ambiente. Este foco será común a todo el escenario y simulará la luz producida por el Sol. Nos apoyaremos en el uso de tres entidades:

- ***light_environment:*** Esta entidad es la responsable del control de la luz y genera un foco de luz difusa. Solo puede haber una entidad de esta clase por escenario.

- ***env_sun:*** Esta entidad coloca un *sprite* simulando el Sol (también está disponible la Luna) dando la sensación de que se encuentra a una gran distancia. Esta entidad no afecta a la iluminación, simplemente es un elemento de decoración para aumentar el grado de realismo.

- ***shadow_control:*** Esta entidad es la encargada de la gestión de las sombras dinámicas de todo el escenario. Este tipo de sombras son las generadas por los modelos animados y se calculan en tiempo de ejecución, a diferencia de las sombras estáticas generadas por edificios u otros objetos inmóviles y calculadas en tiempo de compilación. La entidad permite definir la dirección, color y distancia de atenuación de las sombras.

Una vez definidas las entidades a usar vamos a proceder a instanciarlas en nuestro escenario y a configurarlas debidamente. Se recomienda colocar las tres entidades agrupadas en el centro del escenario a una cierta altura (fig. A.11).

En primer lugar instanciaremos la entidad light_environment. Las propiedades que tendremos que configurar son:

- **Pitch Yaw Roll**: Orientación de la entidad en el escenario. Pitch es el ángulo vertical y Yaw el horizontal. Estos parámetros en conjunción a la entidad *SkyBox* (es una caja que contiene el escenario completo y representa el cielo y el horizonte) sirve para definir la dirección de la luz directa. En nuestro caso deberemos configurarlo para el correcto uso de las sombras.

- **Brightness:** Se especifica el color de la luz directa en RGB seguido de un entero para indicar el nivel de brillo.

- **Ambient:** Se especifica el color de la luz ambiente en RGB seguido de un entero para indicar el nivel de brillo.

A continuación colocaremos la entidad env_sun. Los parámetros que deberemos configurar son:

- **Pitch Yaw Roll:** Debe configurarse con el mismo valor definido previamente.

- **Rendercolor:** Se establece el color del Sol mediante RGB.

- **Material:** Permite seleccionar el sprite que dibujará el Sol.

- **Size:** Podemos seleccionar el tamaño especificando un número entero.

Finalmente instanciaremos la entidad shadow_control. Debemos configurar las propiedades:

- **Pitch Yaw Roll:** Debe configurarse con el mismo valor definido previamente, a excepción del pitch que deberá tener el valor negativo (se recomienda usa un valor negativo para el pitch de ligth_environment y uno positivo para shadow_control, -70 y 70 grados, por ejemplo).

- **Color:** Se define el color de las sombras en RGB.

- **Distance:** Permite establecer la máxima distancia a la que proyectará la sombra. Se especifica en pulgadas y se permite usar decimales.

Una vez establecidos todos estos parámetros y compilado correctamente, podremos observar el Sol alumbrando nuestro escenario y ver como se dibujar las sombras (fig. A.12).

Figura A.12: El Sól en el escenario.

Iluminación. Luz Directa

Una vez definida la luz global en nuestro escenario, puede ser de interés añadir focos de luz direccionales para aumentar el grado de realismo (o diseñar un escenario nocturno e iluminarlo con este tipo de iluminación). Para realizar esto deberemos usar la entidad *light_spot* (fig. A.13). Esta entidad produce un halo de luz unidireccional parecido al de un foco o un faro de coche.

Las propiedades que deberemos configurar son:

- **Pitch Yaw Roll:** Orientación del foco de luz. La orientación de la luz también puede establecerse mediante la propiedad *Entity to point at*, seleccionando la entidad a la que el foco apuntará.

- **Brightness:** Color de la luz. Se especifica en RGB seguido de un entero indicando el nivel de brillo.

- **Inner/Outer:** Parámetros para especificar las dimensiones del cono de luz. Se establece en ángulos y el valor máximo son 90 grados.

Para terminar, a continuación se muestran algunas de las configuraciones típicas para la simulación de distintos tipos de iluminación:

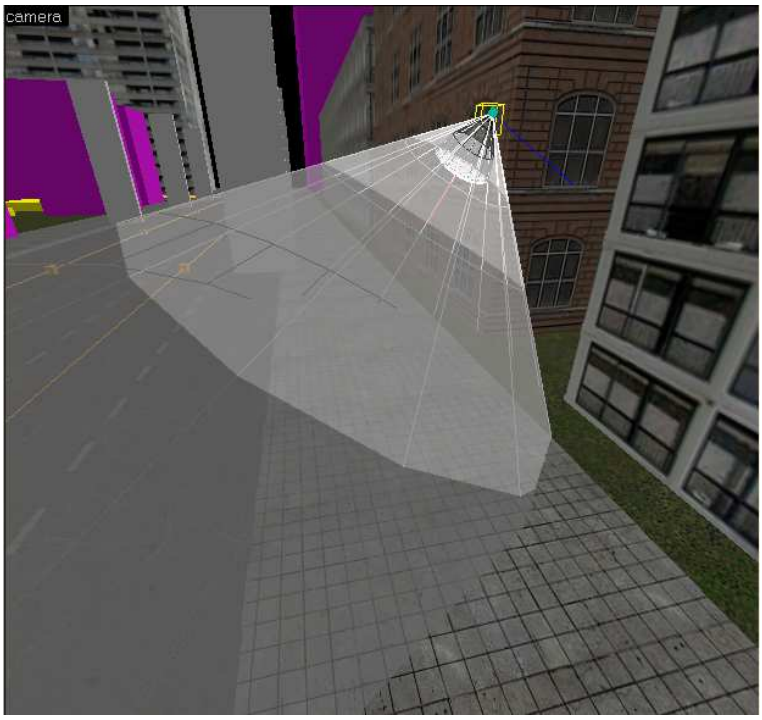

Figura A.13: Entidad light_spot.

- **Luz ambiente. Escenario diurno.**
  Brightness 237 218 143 800
  Ambience 190 201 220 100
- **Luz ambiente. Escenario nocturno.**
  Brightness 175 230 239 50
  Ambience 43 45 57 5
- **Luz directa. Lámpara**
  Brightness 147 226 240 3000
- **Luz directa. Bombilla**
  Brightness 254 216 146 4000
- **Luz directa. Fluorescente**
  Brightness 159 237 215 3500

A.1.10. Otros efectos. Efectos meteorológicos

Finalmente, para aumentar aún más el grado de realismo, el editor nos permite añadir y configurar efectos meteorológicos en nuestro escenario. Aunque el resultado obtenido es espectacular, añadir este efecto al escenario es un proceso relativamente sencillo.

Deberemos usar una entidad de tipo *func_precipitation*. Para instanciar esta entidad deberemos hacerlo del mismo modo que lo hacíamos con la entidad *func_tracktrain*. Únicamente deberemos configurar tres propiedades:

- **Density:** Densidad del efecto. Se especifica en porcentaje (0-100 %).
- **Color:** Color del efecto. Se especifica en RGB.
- **Precipitation type:** Permite establecer el tipo de efecto. Hay cuatro opciones; Lluvia, nieve, ceniza y nevada.

A.2. Resumen

En este apéndice hemos comentado todos los conceptos básicos relacionados con la edición de escenarios necesarios para el correcto uso de la plataforma desarrollada en este proyecto. El futuro usuario de esta plataforma debería prestar una especial atención a este capítulo y asemejar todos los conceptos explicados. En la tabla A.1 se muestran las herramientas descritas en este capítulo.

Es posible que haya aspectos de la edición de escenarios de interés para el usuario

Cuadro A.1: Herramientas edición de escenarios.

Herramienta	**Descripción**
Hammer	Editor de escenarios proporcionado por Valve.
XSI Mod Tool	Entorno de modelado 3D.
Studio Compiler	Compilador/Decompilador de modelos 3D.
Vmex	Descompilador de mapas.
GCFScape	Extractor de ficheros adicionales incrustados en un mapa.

que no estén explicados en este capítulo. Para ampliar la información aquí detallada, se recomienda consultar la wiki ofrecida por Valve sobre el editor Hammer [12].

Apéndice B

ObjectVideo Virtual Video Tool

En este apéndice, *hablaremos de la herramienta principal de la que hace uso la plataforma desarrollada en este proyecto. Definiremos su estructura así como su uso y funcionalidades.*

B.1. Estructura y funcionamiento

La plataforma desarrollada en este proyecto usa la herramienta ObjectVideo Virtual Video Tool (nos referiremos a ella cómo OVVV), desarrollada por la compañía ObjectVideo, empresa líder en el campo de la visión por computador con una gran presencia en el mercado de la video vigilancia.

ObjectVideo Virtual Video Tool es una herramienta gratuita con licencia GPL, basada en una modificación (*mod*) del juego Half Life 2, desarrollado por Valve. La herramienta permite:

- Instanciación y configuración de cámaras en un escenario virtual.
  - Instanciar distintas cámaras y obtener el flujo de video en tiempo real.
  - Configurar los parámetros intrínsecos y extrínsecos de modo independientemente para cada cámara.
- Generación automática de groundtruth.
  - Incluyendo posición de los objetivos, cajas contenedoras y foreground label map.
  - Suprime la tediosa labor de anotación para obtener el groundtruth.
- Configuración flexible del escenario.

- Carga de scripts de escenarios para repetir distintas situaciones a testear.
- Control directo de objetivos en tiempo real. Test interactivo.
- Simulación de efectos ópticos observados en cámaras reales, como desenfoque, ruido o ghosting.
- Otras configuraciones, como el tamaño de las imágenes obtenidas o el framerate de las cámaras.

La herramienta completa está formada por 3 componentes:

- Virtual Video Mod. Es el mod del juego Half Life 2. Gestiona el control y configuración de múltiples cámaras en el escenario virtual.
- Filtro DirectShow. Permite que cualquier aplicación basada en DirectShow pueda crear/configurar cámaras en el escenario, y recuperar los frames capturados por estas.
- Virtual Video C library. Librería en C para desarrollo de aplicaciones que no usen DirectShow. La opción anterior hace uso de esta librería, por lo que las funcionalidades son las mismas.

El componente Virtual Video Mod hace uso de los protocolos de red basados en sockets para recibir comandos o enviar los frames a las aplicaciones cliente. Por lo tanto, podría desarrollarse una aplicación cliente en cualquier lenguaje, en lugar de usar el filtro DirectShow o la librería en C. El único requisito seria que fuera capaz de trabajar con estos protocolos de comunicación.

El juego Half Life 2 usa una arquitectura de cliente/servidor. El servidor mantiene el estado y la lógica del juego, y las aplicaciones cliente son las encargadas de implementar el renderizado y la funcionalidad de la interfaz del usuario. El mod Virtual Video incluye dos servidores, que son los que gestionarán las cámaras dentro del escenario. Las aplicaciones cliente interactuaran con estos dos servidores para enviar los distintos comandos de configuración de las cámaras o para recuperar los frames. En la figura B.1 se muestra la arquitectura completa. Como podemos ver en la imagen, tenemos el Camera Server y el PTZ Server. Cada uno es responsable de un conjunto de tareas.

- **Camera Server:** Es el servidor de cámaras. Se encarga de la creación, configuración y mantenimiento de estas dentro del escenario virtual. Mediante este servidor se recuperan las imágenes capturadas por las cámaras, y es posible modificar los parámetros de estas, así como asociar modos especiales de funcionamiento como control PTZ o escaneo automático (cámara de video vigilancia).

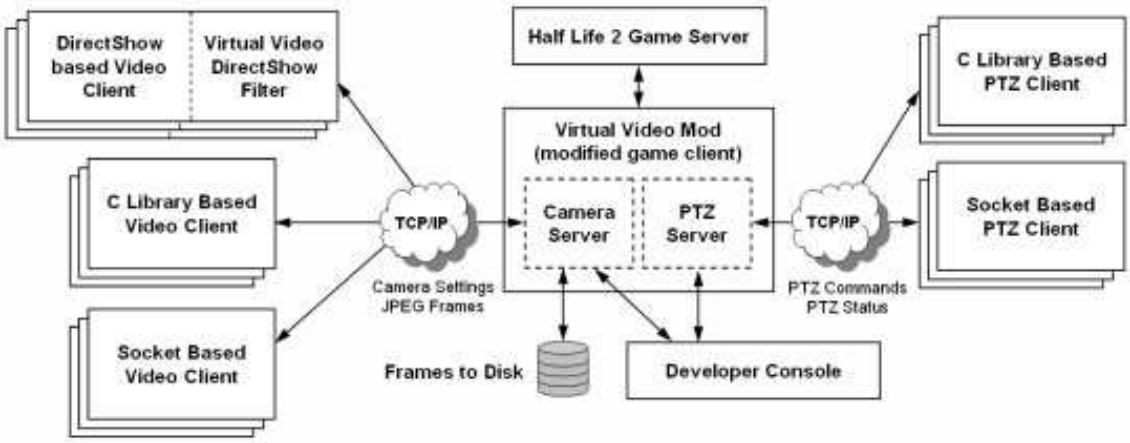

Figura B.1: Estructura OVVV.

- **PTZ Server:** Este servidor se encarga de crear y mantener los controles PTZ. Estos controles se pueden asociar a una o más cámaras del servidor de cámaras, y convertirlas en cámaras activas o PTZ (Pan, Tilt, Zoom). Los controles PTZ calculan la configuración concreta que deberán tener las cámaras asociadas en un instante concreto, y se comunican con el servidor de cámaras para actualizar esta configuración.

ObjectVideo argumenta que la razón de implementar estas funcionalidades en dos servidores separados es para simular la distinción física entre una cámara y un cabezal PTZ (responsable del movimiento de la cámara). Aunque las relaciones entre cámaras y controles PTZ suelen ser de 1 a 1, ambos servidores permiten múltiples conexiones.

B.2. Uso

ObjectVideo proporciona 3 métodos para hacer uso de los servidores.

- **Filtro DirectShow:** Cualquier aplicación basada en DirectShow podrá crear cámaras en el escenario y recuperar el flujo de video. Se accede por URL, donde se especifican el identificador y los parámetros de configuración de la cámara a la que se quiera conectar. Una URL de muestra podría ser *simvideo://localhost:3166?camID=1&x=0&y=0&z=10&rx=0&ry=20&rz=0&framerate=25*, con la que nos conectaríamos a la cámara 1 en la posición (0,0,10) con una orientación de (0,20,0) y con un framerate de 25 frames por segundo. La manera más sencilla de hacer uso de esta opción es mediante el reproductor Windows Media Player.

- **Librería C:** Librería con un conjunto de funciones para interactuar con el servidor de cámaras y el servidor PTZ. Es la opción que se ha usado para el desarrollo de este proyecto.
- **Protocolo de red:** Implementación de una aplicación en cualquier otro lenguaje que no sea C. La aplicación deberá implementar la comunicación con los servidores mediante sockets. La opción anterior hace el papel de interfaz de alto nivel a esta opción.

En la figura B.2 se muestra una posible situación del estado y conexiones en los servidores con varias cámaras creadas. Como se puede observar, en esta situación

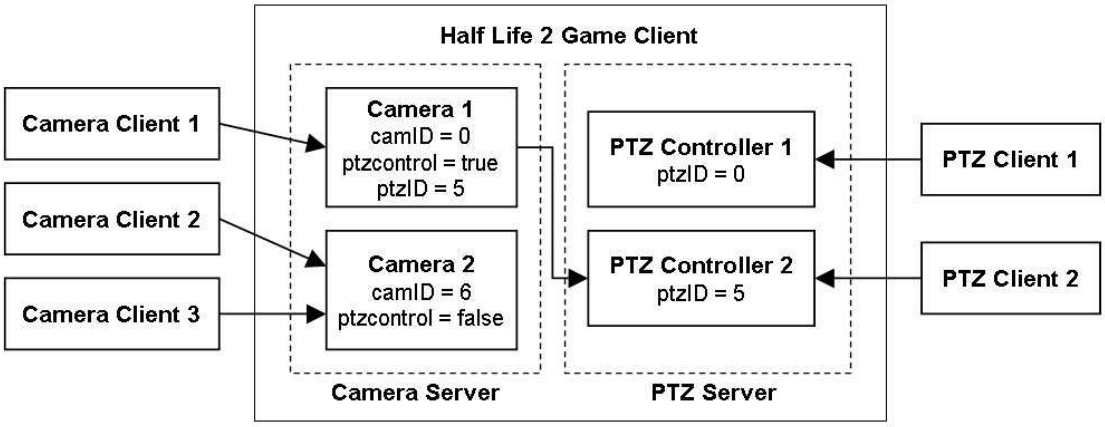

Figura B.2: Ejemplo situación con varios clientes conectados.

tenemos 2 cámaras, dos controladores PTZ y 3 aplicaciones cliente conectadas al servidor de cámaras. Cada cámara y controlador PTZ tiene un identificador único. Al arrancar la herramienta son creadas automáticamente una cámara y un controlador PTZ con el identificador 0.

En el ejemplo mostrado tenemos el cliente 1 conectado a la cámara activa creada por defecto (camID 0), y esta tiene asociado el segundo controlador PTZ. Para convertir una cámara estática en activa, se debe activar el flag de control PTZ y asociar un identificador válido de controlador PTZ.

Las dos aplicaciones cliente restantes están conectadas a la cámara 2. Cualquier cambio realizado en la configuración de la cámara por uno de los dos clientes se verá reflejado también en el otro. Las configuraciones de las cámaras y de los controladores PTZ se mantienen a lo largo de la ejecución de la herramienta, a pesar de las conexiones o desconexiones de los clientes.

Para ofrecer el mejor rendimiento posible, solo las cámaras que tengan clientes

conectados recuperarán el flujo de video del escenario virtual.

A continuación se detallaran los pasos para cargar correctamente la herramienta. La herramienta deberá arrancarse desde Steam, la aplicación de gestión de contenidos de Valve. Debemos seguir los siguientes pasos:

- Cargar el gestor Steam.
- Seleccionar la pestaña Mis juegos. Previa instalación, deberá aparecer la opción Virtual Video.
- Seleccionar Virtual Video y abrir la ventana de propiedades.
- Configurar las opciones de carga de la siguiente manera:

  console -dev -windowed -w 640 -h 480 -nosound

  Esta configuración cargará el juego Half Life 2 en una ventana de tamaño 640x480, con la consola de desarrollar habilitada y el sonido desactivado. Esto solo deberá hacerse una única vez.
- A continuación debemos volver a la ventana anterior y cargar la herramienta Virtual Video.
- Una vez cargada la herramienta se nos mostrará la consola de desarrollador. Ahora deberemos cargar el escenario donde queremos instanciar las cámaras. Para realizar esto deberemos escribir la palabra clave *map*, y a continuación el nombre del mapa. El mapa deberá estar en la carpeta */Steam/steamapps/UserName/half-life 2/hl2/maps/* (si hemos compilado el mapa mediante el editor Hammer, este ya habrá dejado el archivo del mapa en ese directorio).
- Con el mapa cargado ya podremos hacer uso de la plataforma desarrollada en este proyecto. Para abrir o cerrar la consola debemos hacerlo mediante la tecla ESCAPE. Si la consola está abierta, las aplicaciones cliente no son capaces de recuperar el flujo de video de las cámaras, por lo que deberemos cerrarla para poder guardar los datos a disco.

Además de los ofrecidos por Valve, ObjectVideo proporciona un conjunto de comandos adicionales. Estos se clasifican en 3 categorías:

- **Server Commands:** Son los comandos para consultar el estado de las cámaras, aplicaciones cliente conectadas y controladores PTZ.
  - list_cameras
  - list_clients

  - list_clients_ptz
  - list_controllers

- **Visual Effect Commands:** Son los comandos usados para activar y configurar la simulación de los distintos efectos visuales al flujo de video obtenido por las cámaras. Cada comando tiene unos parámetros asociados. Estos comandos son los usados por la plataforma desarrollada en este proyecto. Cada uno de ellos esta descrito con más detalle en la siguiente sección de este apéndice.

  - jitter
  - noise_bands
  - noise_blur
  - noise_defocus
  - noise_ghost
  - noise_pixel
  - noise_distorsion

- **Rendering Options:** El más importante y de obligada mención de estos comandos es el de groundtruth. Debido a que el cálculo del groundtruth supone una carga considerable, este viene desactivado por defecto. Para activarlo deberemos introducir el comando mediante la consola. Si no activamos el groundtruth, aunque al usar la plataforma desarrollada en este proyecto veamos cómo se generan los ficheros .gdt (ficheros binarios que contienen los datos de groundtruth), estos solo contendrán la información relacionada con la cámara y ninguna sobre los personajes presentes en el escenario.

  El comando omni permite simular una omni-cámara (con un ángulo de visión de 360º) y el comando antialias activa/desactiva mejoras gráficas relacionadas con el renderizado.

  - antialias
  - omni
  - groundtruth

B.3. Efectos ópticos

Para poder generar sistemas de visión más robustos la herramienta OVVV proporciona un conjunto de efectos ópticos sufridos a menudo en los dispositivos de captura convencionales y permite simularlos en el flujo de video obtenido en

el entorno virtual. De esta manera se aumenta considerablemente el grado de realismo y es posible mejorar el proceso de validación.

Como esta es otra de las características interesantes de la herramienta OVVV, se ha añadido la opción de poder usar estos efectos mediante la plataforma desarrollada en este proyecto.

El modo de uso que ofrece la herramienta es mediante la consola. Debe introducirse el comando del efecto, seguido de sus correspondientes parámetros. Para facilitar esta tarea y evitar al usuario tener que conocer todos los comandos, este puede crear un fichero de configuración mediante la plataforma desarrollada en este proyecto con el listado de efectos deseados y cargarlos mediante la consola. Es por esta razón que es necesario que el usuario establezca en la pantalla de configuración cual es el directorio donde está instalada la herramienta OVVV. El directorio a establecer debe tener el formato */Steam/steamapps/SourceMods/Virtual Video/cfg*.

El usuario debe introducir los valores deseados en el campo de texto del efecto deseado y a continuación activarlo. Una vez hecho esto se creará el fichero de configuración en el directorio cfg de la herramienta OVVV. Para cargar los efectos deberá introducirse el siguiente comando mediante la consola:

exec effects.cfg

No hay ninguna limitación en cuanto a aplicar distintos efectos simultáneamente. Si se activa un efecto y no se establecen todos los valores de sus parámetros, el valor por defecto es 0. Hay algunos efectos que solo permiten valores enteros como parámetros.

A continuación comentaremos en qué consiste cada uno de ellos y cuáles son sus valores típicos.

B.3.1. Jitter

El jitter es un concepto relacionado con la variación. Este efecto es simulado como una pequeña rotación aleatoria en los ejes Y y Z del frame. La probabilidad de los ángulos está uniformemente distribuida, por lo que la máxima variación de los pixeles en el plano de la imagen estará determinada por *+/- maxJitterX* en el eje X y *+/- maxJitterY* en el eje Y. El jitter aplicado es sensible al zoom, por lo que para un mismo valor de jitter y mayor zoom, mayor será la variación, igual que ocurre en las cámaras convencionales.

- **Max X:** Máximo desplazamiento de los pixeles en el eje X.

- **MaxY:** Máximo desplazamiento de los pixeles en el eje Y.

B.3.2. Noise Bands

Este efecto consiste en la aparición aleatoria de bandas a lo largo del eje Y con mayor intensidad que el resto de la imagen. Los parámetros permiten configurar el tamaño de estas bandas, su intensidad y la probabilidad de aparición.

- **Height:** Altura en píxeles de la banda.
- **Intensity:** Intensidad en niveles de gris.
- **Probability:**Probabilidad de aparición.

B.3.3. Noise Blur

Este efecto simula un reenfoque aleatorio sufrido a menudo en las cámaras que disponen de auto-enfoque. El efecto es simulado mediante un cuadrado de suavizado uniforme, que varía su tamaño desde 0 hasta el máximo y luego vuelve a 0. Se puede configurar el tamaño máximo de este cuadrado, la velocidad de reenfoque, el tiempo en el momento de suavizado máximo y la probabilidad de que ocurra el reenfoque.

- **Radius:** Radio máximo en pixeles que alcanzará el núcleo de suavizado.
- **Speed:** Velocidad en píxeles/sec en que aumentará o encogerá el núcleo de suavizado.
- **Probability:** Probabilidad que ocurra el reenfoque.
- **Duration:** Duración en frames del suavizado máximo.

B.3.4. Noise Defocus

Este efecto simula un desenfoque de la cámara. A diferencia del efecto anterior, este no varía con el tiempo. El desenfoque es estático a través de los frames. El suavizado es aplicado como un núcleo Gaussiano de dimensiones $(2 \cdot size + 1) \cdot (2 \cdot size + 1)$. Este efecto solo permite configurar el tamaño del núcleo de suavizado.

- **Size:** Tamaño máximo en pixeles del núcleo de suavizado.

B.3.5. Noise Ghost

Este efecto simula el *ghosting*, la aparición de zonas de la imagen sobrepuestas sobre el frame original debido a posibles retrasos en la transmisión. Es un efecto observado en la transmisión de video analógico. El parámetro *offset* permite controlar el desplazamiento que se aplicará a la imagen sobrepuesta y el

Figura B.3: [Izquierda] Efecto desactivado. [Derecha] Efecto noise defocus activado.

parámetro *intensity* permite establecer la intensidad de esta (como fracción del frame original).

Figura B.4: [Izquierda] Efecto desactivado. [Derecha] Efecto noise ghost activado.

- **Offset:** Desplazamiento en pixeles de la imagen sobrepuesta.
- **Intensity:** Intensidad de la imagen sobrepuesta. El valor debe estar en el rango 0.0 - 1.0.

B.3.6. Noise Pixel

Este efecto simula la inserción de ruido por parte del sensor óptico, fenómeno sufrido a menudo por cámaras convencionales. El ruido se simula añadiendo a cada pixel un valor según una distribución aleatoria uniforme, entre 0 y el valor máximo especificado por el parámetro *intensity*.

- **Intensity:** Ruido máximo en niveles de gris a añadir a los pixeles.

B.3.7. Radial Distorsion

Este efecto simula una distorsión radial producida por la lente en las cámaras convencionales. El efecto es modelado como:

$$u = u' \cdot (1 + k_2 \cdot |u|^2)$$

Se puede configurar el coeficiente k_2, y su valor debe estar en el rango (0,0 − 0,000002). Aplicando el valor 0 al coeficiente la distorsión será nula.

Figura B.5: Efecto noise pixel.

Figura B.6: [Izquierda] Efecto desactivado. [Derecha] Efecto radial distorsion activado.

- **Coefficient:** Coeficiente de segundo orden k_2. Su valor debe estar en el rango $(0{,}0 - 0{,}000002)$.

B.4. Generación de Groundtruth

Para poder realizar una transmisión más eficiente, la herramienta OVVV genera los datos del groundtruth y los codifica en binario. Para poder hacer uso de estos datos deberemos conocer la estructura del binario, y aplicar un post-procesado para decodificarlos.

En el contexto de este proyecto dispondremos de un archivo de groundtruth por cada frame del video obtenido por la plataforma. Cada archivo contendrá la siguiente información:

- Parámetros extrínsecos e intrínsecos de la cámara (posición, orientación, campo de vista, etc.).
- Información sobre los targets (en nuestro caso, personas). Posición en el escenario y caja contenedora en la imagen.
- Foreground Label Map de los targets.

En todas las aplicaciones de visión por computador, el disponer del groundtruth es un requisito indispensable. Este no solo sirve para validar los distintos sistemas, sino que también sirve para entrenarlos. A alto nivel, podemos ver el proceso de entrenamiento de un sistema (de detección por ejemplo) como una calibración de este. Un ajuste de los distintos parámetros determinantes en el proceso de detección por parte del sistema.

La generación del groundtruth es una de las características más importantes de la herramienta OVVV. Si no dispusiera de esta funcionalidad habría que generar el groundtruth manualmente. La obtención de este es un proceso lento y tedioso, pero lo es especialmente más en el campo de la visión por computador o del procesamiento de imágenes.

Supongamos que disponemos de un sistema de clasificación binario implementado mediante un red neuronal y un conjunto de mil ejemplos. Antes de comenzar el proceso de entrenamiento deberemos obtener el groundtruth. No habrá más remedio que recorrer los mil elementos y, uno a uno, asignarles un identificador indicando a cual de las dos clases pertenecen.

Ahora supongamos que disponemos de un sistema de detección y un video del que se extraerán los ejemplos para entrenar. El proceso que se deberá realizar para cada uno de los frames es el siguiente:

1. Analizar la imagen en búsqueda de los targets que detectará el sistema a entrenar.
2. Mediante alguna herramienta de anotación obtener las coordenadas de la caja contenedora del target y posteriormente recortar esa parte de la imagen.
3. Si el sistema es además de seguimiento (*tracking*) hay que asignar un identificador al target. La restricción de estos sistemas es que a un target presente en distintos frames se le asigne el mismo identificador. Esto exigiría que se tuviera en cuenta posibles apariciones previas de los targets en el momento de asignar los identificadores.

4. Si disponemos de un sistema más preciso quizás habría incluso que segmentar la imagen. Este proceso se basa en recortar únicamente los pixeles del frame que pertenecen a algún target (los recortes obtenidos mediante cajas contenedoras contendrán parte del fondo). En este paso se obtiene el foreground label map (sección 4.3).

A simple vista se puede observar la complejidad del proceso completo y la cantidad de tiempo que habría que invertir para llevarlo a cabo. También hay que tener en cuenta que el hecho de que sea un método muy repetitivo genera que la atención del responsable que lleva a cargo la tarea mengüe, con la correspondiente disminución de la fiabilidad y precisión. Por estas razones, la generación automática del groundtruth es una funcionalidad muy interesante. La comentaremos con más detalle a continuación.

El proceso llevado a cabo por la herramienta OVVV para generar las cajas contenedoras y el foreground label map consta de dos pasos:

- **Un proceso de chroma keying**. Para cada target, se llena un frame buffer de un color distintivo (verde lima por ejemplo), y se renderiza el target. Para calcular la caja contenedora se buscan los pixeles que no forman parte del fondo, aquellos que forman parte del target y no contienen el color distintivo. A continuación se vuelve a rellenar el frame buffer y se repite el proceso con los siguientes targets.

- **Un proceso de chroma keying añadiendo el uso de un z-buffer (para forzar las oclusiones).** Este paso es parecido al anterior, pero se consigue un correcto resultado en función a las oclusiones comprobando si los nuevos pixeles renderizados están más cerca de la cámara que los renderizados previamente. Para comprobar si aparecen oclusiones entre los targets y el resto del escenario, primero se renderiza este. A continuación, se calcula el z-buffer. Este paso simplemente consiste en obtener la distancia de la cámara a cada pixel del frame. Antes de renderizar cualquier pixel de un target se comprobará si este es más cercano a la cámara que el correspondiente en el z-buffer. Todos los pixeles de los targets que sean renderizados serán indicados en el foreground label map.

Es importante mencionar que la fiabilidad y precisión obtenidas son a causa de que la herramienta dispone de la posición exacta de los targets dentro del escenario y del modelo (por tanto de la forma) de estos. El proceso de generación del groundtruth se añade como un elemento más a la cadena de renderizado.

Debido a que la herramienta OVVV es externa a este proyecto, no se dispone de más información al respecto sobre la generación del groundtruth. Se puede

observar que este método es bastante general, y podría usarse para la generación del groundtruth de cualquier otro tipo de objeto. El problema es que se desconoce el modo en que la herramienta OVVV decide si un modelo es un target a detectar o no lo es. Se ha invertido mucho tiempo investigando este hecho pero no se ha obtenido ningún resultado concluyente. Si se pudiera solucionar esto, se podría generalizar y ampliar el número de aplicaciones para los que podría ser útil este proyecto. El editor de escenarios nos ofrece libertad absoluta para diseñar el entorno a nuestro antojo, con lo que, por ejemplo, podríamos enfocar el uso de esta plataforma al reconocimiento y seguimiento de vehículos, así como de cualquier otro tipo de objetos.

B.4.1. Formato

Tal y como se comentaba anteriormente, la herramienta OVVV presenta una arquitectura de tipo cliente-servidor. Las imágenes y su groundtruth son generadas en el servidor de cámaras. Los clientes se conectan al servidor y entonces se inicia la transmisión mediante sockets. Para obtener una transmisión más eficiente y un mejor rendimiento es deseable que se transmita el máximo posible de información utilizando la mínima cantidad de datos. Por esta razón, el fichero que contiene la información del groundtruth no está en texto llano, sino que se codifica en binario y hace uso de la técnica del run-length. Por lo tanto, para poder obtener esta información deberemos realizar un proceso de decodificación.

Para poder decodificar el archivo de groundtruth deberemos conocer su estructura. En las siguientes tablas (B.1, B.2 y B.3) se muestra el orden en que aparecen los distintos campos, así como el tamaño de cada uno de ellos. El groundtruth contiene información sobre los targets presentes en una imagen, por lo que habrá un archivo con este para cada una de las imágenes.

En la tabla B.1 se puede observar la estructura general. Podemos definir 3 secciones:

- ***Información de la cámara.*** Cabecera de 52 bytes con toda la información relacionada con la cámara.
- ***Información de los targets.*** Habrá una instancia de este elemento por cada target visible en el frame. Tiene un tamaño de 56 bytes.
- ***Foreground Label Map.*** Esta codificada mediante la técnica run-length.

Las dos últimas secciones son de tamaño variable. Dependen del número de targets que aparezcan en la imagen. En la tabla se indican con los términos T y L. Está información se encuentra en la cabecera de tamaño fijo.

Cuadro B.1: Estructura General archivo groundtruth.

Byte Length	**Description**
52	Header and Camera groundtruth
Tx56	Target ground truth, T = number of visible targets (from header)
Lx4	RLE label map, L = number of RLE data elements (from header)

A continuación mostramos la estructura de esta cabecera (tabla B.2). Contiene los parámetros extrínsecos e intrínsecos de la cámara en el momento en que se capturó la imagen, así como información relacionada con esta. En la tabla B.3

Cuadro B.2: Estructura General cabecera groundtruth.

Byte Offset	**Data Type**	**Description**
0-3	int	Frame number
4-7	float	Elapsed time in seconds since game started
8-11	int	Frame width
12-15	int	Frame height
16-19	float	Horizontal field of view in degrees
20-23	float	Camera x-coord in world frame in inches
24-27	float	Camera y-coord in world frame in inches
28-31	float	Camera z-coord in world frame in inches
32-35	float	Camera rotation about world x-axis in deg
36-39	float	Camera rotation about world y-axis in deg
40-43	float	Camera rotation about world z-axis in deg
44-47	int	T = Number of ground truth targets
48-51	int	L = Length of RLE label map data in 32-bit elements

se muestra como está estructurada la información relacionada con los targets. Esta contiene el identificador del target, sus coordenadas dentro del escenario y la posición de los extremos de las cajas contenedoras dentro de la imagen. Debido a que la sección del foreground label map está codificada mediante la técnica de run-length, esta no tiene un estructura fija. En el siguiente apartado comentaremos como decodificar esta última parte y definiremos que es el foreground label map.

B.4.2. Foreground Label Map

Cuando mencionamos el término foreground label map nos referimos a una imagen que indica qué pixeles de la imagen forman parte de un target.

A cada target presente en la imagen se le asigna un identificador. El foreground label map es una imagen completamente en negro donde los pixeles que pertenecen a un target contienen como valor el identificador de este. A continuación mostramos un ejemplo (figura B.3).

Cuadro B.3: Estructura Información sobre targets.

Byte Offset	Data Type	Description
0-3	int	Target label (identifies target pixels in label map)
4-7	float	World x-coord of 3D target centroid in inches
8-11	float	World y-coord of 3D target centroid in inches
12-15	float	World z-coord of 3D target centroid in inches
16-19	int	Top y-coord of bounding box around visible target pixels
20-23	int	Bottom y-coord of bounding box around visible target pixels
24-27	int	Left x-coord of bounding box around visible target pixels
28-31	int	Right x-coord of bounding box around visible target pixels
32-35	int	Number of visible foreground pixels in bounding box
36-39	int	Top y-coord of bounding box around all target pixels
40-43	int	Bottom y-coord of bounding box around all target pixels
44-47	int	Left x-coord of bounding box around all target pixels
48-51	int	Right x-coord of bounding box around all target pixels
52-55	int	Number of foreground pixels in bounding box

Debido a que con esta información se nos indica exactamente qué pixeles com-

Figura B.7: Foreground Label Map.

ponen cada target, la precisión es mucho mayor que la que nos ofrecen las cajas contenedoras.

El método de compresión run-length es usado a menudo en la codificación de imágenes. El principio de esta técnica es sustituir secuencias de datos consecutivas con el mismo valor por su número de apariciones y el propio valor. Es especialmente efectivo en áreas suaves de la imagen, donde hay pocas variaciones de color.

Una secuencia consecutiva de un valor será sustituida por el número de apariciones y el valor. Por lo tanto la siguiente secuencia:

AAAAAAAAAAAAAAABBBBBAAAAAABBBB

Será sustituida por:

15 A 5 B 6 A 4 B

De este modo, representamos la secuencia original de 20 caracteres, en tan solo 9. Es importante mencionar que este sistema de compresión es sin perdida. Siempre podremos recuperar la información original.

Para la codificación del foreground label map esta técnica es muy útil. Podremos representar toda la información de la imagen usando pocos datos, ya que la mayoría de regiones de esta, las que pertenecen al fondo, contienen el mismo valor.

La herramienta OVVV lo codifica en L elementos, donde cada uno de ellos ocupa 4 bytes. Están almacenados en parejas, donde cada pareja se compone del valor y el número de repeticiones de este. El siguiente código en C muestra un ejemplo de cómo se puede decodificar la información del foreground label map, donde los datos se encuentran en un buffer *rlebuff* y *width* y *height* son las dimensiones del frame (obtenidas de la cabecera).

```
int index, pix, label, run, endOffset;
int *labelMap;
labelMap = malloc(width*height*4);
rleIndex = 0;
pixOffset = 0;
while ((rleIndex ¡L) && (pixOffset ¡width*height))
{
  label = rlebuf[index++];
  run = rlebuf[index++];

  for (endOffset = pixOffset + run;
  pixOffset ¡endOffset;
  pixOffset++)
  {
    labelMap[pixOffset] = label;
  }
}
```

B.5. Resumen

En este apéndice hemos definido las funcionalidades y la arquitectura de la herramienta usada en este proyecto. Algunas de las funcionalidades que ofrece esta herramienta son:

- Instanciación y configuración de cámaras en escenarios virtuales.
- Generación automática del groundtruth.
- Configuración flexible del escenario.

También hemos definido los conceptos básicos y los comandos necesarios para interactuar y realizar un correcto uso de la consola que proporciona esta herramienta.

Finalmente hemos presentado el modo en que la herramienta OVVV genera el groundtruth, así como la manera de codificarlo. También hemos indicado el proceso a seguir para decodificarlo y obtener la información original.

Firmat: Marc
Bellaterra, 15 de Septiembre de 2009