VNIVERSITAT DE VALÈNCIA

Departament d'Informàtica

TESIS DOCTORAL

MEJORA DE PRESTACIONES DE LOS SISTEMAS DE REALIDAD AUMENTADA COLABORATIVA SOBRE DISPOSITIVOS MÓVILES

Presentado por:
D. Víctor Fernández Bauset
Valencia, Mayo 2014

Trabajo dirigido por:
Dr. D. Juan Manuel Orduña Huertas
Dr. D. Pedro Morillo Tena

A mi familia. A los que están, los que ya se fueron,
los que acaban de llegar y los que están por venir.

Agradecimientos

Un documento científico de esta índole no se puede llevar a buen término sin el apoyo de los que te rodean. Por tanto, en las siguientes líneas me gustaría expresar el agradecimiento que merecen todas las personas que han ayudado directa o indirectamente en la realización de esta tesis doctoral.

En primer lugar, me gustaría agradecer a mi familia su apoyo incondicional. Y en especial a mis padres, por permitirme ser estudiante hasta pasados los treinta. ¡Por fin he acabado!

En segundo lugar, quiero dar las gracias a los compañeros del Departamento de Informática de la Universitat de València por compartir su tiempo y conocimientos. Más en detalle, me gustaría agradecer al Dr. Francisco Grimaldo Moreno la confianza depositada en mí, tras tutorizar mi proyecto fin de carrera y proponerme la vía académica. Por volverme a tutorizar en el trabajo fin de máster y por todos sus consejos posteriores. También quería agradecer a Vicente Cavero Millán su ayuda en el desarrollo del simulador. Por las horas que ha pasado resolviéndome dudas y ayudándome en la implementación. Y por compartir sus conocimientos de C, Python, redes y un largo etcétera. Por último, agradecer a mis directores de tesis, Dr. Juan Manuel Orduña Huertas y Dr. Pedro Morillo Tena, por su guía a lo largo de la investigación, sus conocimientos y por las sucesivas revisiones de este documento de tesis doctoral. Cabe destacar el tesón y la disponibilidad constantes del Dr. Juan Manuel Orduña, así como su notable colaboración en la redacción de los artículos derivados de esta investigación. Sin su ayuda y ánimos esta tesis habría sido imposible.

Este trabajo se ha desarrollado dentro del proyecto de investigación TIN 2009-14475-C04-04.

Resumen

Las tecnologías basadas en la denominada *Realidad Aumentada*[1] consisten en la superposición de información virtual sobre la imagen del mundo real que un determinado dispositivo electrónico es capaz de mostrar en su pantalla. Aunque el término hace referencia a tecnologías con más de 50 años de antigüedad, su mayor expansión se ha realizado en los últimos años. Se pueden ver ejemplos de Realidad Aumentada en áreas tan diversas como cirugía avanzada, arquitectura, turismo, sector militar, marketing... . Esta expansión se debe al vertiginoso avance de las tecnologías actuales que se ha producido en el sector de la telefonía móvil, lanzando al mercado dispositivos que incorporan hardware y software con importantes prestaciones. Con ellos se dispone de todo lo necesario para realizar AR: cámara, pantalla a color de alta resolución, procesador... . No obstante, si lo que se pretende es que varios dispositivos interactúen entre sí utilizando la misma aplicación de AR se necesitará un nivel mínimo de prestaciones. A este tipo de aplicaciones se les conoce como aplicaciones de Realidad Aumentada Colaborativa [2] y, para que haya una interacción fluida entre los usuarios de este tipo de aplicaciones, se necesita que respondan en lo que se llama *Tiempo interactivo* (cota actualmente establecida en menos de 250 ms.).

Debido a la gran variedad de dispositivos móviles existentes en el mercado actual, a las distintas gamas y a que ofrecen prestaciones muy distintas unos de otros, es necesario efectuar una caracterización de los más representativos para poder decidir si, por ejemplo, un teléfono móvil de última generación es adecuado o no para ejecutar aplicaciones CAR. Por lo tanto, la primera parte de esta tesis

[1] Más conocida por AR, del inglés “Augmented Reality”.

[2] Conocido usualmente por CAR, del inglés “Collaborative Augmented Reality”.

se centra en la caracterización de los dispositivos móviles actuales.

Tras analizar los dispositivos móviles como elementos individuales de AR es necesario analizar cómo se comportan en conjunto, mientras colaboran en la misma aplicación CAR. Por lo tanto, en la segunda parte de la tesis se va a realizar un sistema CAR que permita, por una parte recrear miles de dispositivos móviles con comportamiento *interactivo* y, por la otra, medir el comportamiento del sistema para evaluar su escalabilidad y dónde están sus posibles cuellos de botella. Tras este análisis se realizarán diversas modificaciones al sistema para tratar de eliminar dichos cuellos de botella y mejorar el rendimiento general del mismo.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1

Introducción

1.1. Motivación

La Realidad Aumentada[1], se define como la tecnología que permite la superposición de información generada sintéticamente a través de un computador sobre la imagen del mundo real que un dispositivo electrónico ofrece en su pantalla. En la Figura 1.1 se muestra un ejemplo de AR. La parte superior muestra el mundo real, lo que se observa a simple vista, en este caso el campanario de la Catedral de Valencia (conocido como "El Miguelete"). En la parte inferior se muestra la misma imagen pero *aumentada* a través de un dispositivo móvil. La información *aumentada* aquí es la superposición de la fecha actual, las coordenadas en las que se está ubicado y un resumen de la historia del campanario.

[1] Conocida como AR por las siglas en inglés de "Augmented Reality".

Figura 1.1: Ejemplo de Realidad Aumentada sobre el campanario de la Catedral de Valencia. Imagen superior: vista del mundo real; imagen inferior: vista con Realidad aumentada

Por lo tanto, para obtener AR se necesita al menos integrar la acción de una cámara que capte el mundo real, un procesador que permita averiguar qué elemento se debe localizar del mundo real, y una pantalla que muestre el mundo real y le superponga la información adicional que se considere oportuna. Existen muchos ejemplos de AR en nuestra vida cotidiana; son muy comunes al ver deportes en la televisión, como los minutos que quedan en un partido de baloncesto, el marcador de un partido de tenis o, incluso, la línea que demuestra que un delantero está en fuera de juego cuando se ve un partido de futbol. Un ejemplo ilustrativo se muestra en la Figura 1.2, donde se observan tres imágenes de deportes vistos por la televisión y que incluyen Realidad Aumentada como la descrita anteriormente. También se pueden observar ejemplos de AR en educación, para visualizar objetos en 3 dimensiones (3D) como partes del cuerpo humano o animales extintos, como los que se observan en la Figura 1.3. Si se añade un GPS (Sistema de Posicionamiento Global) a esos componentes mínimos, se puede usar la AR para obtener información sobre la zona en la que se esté. E incluso se puede obtener información sobre las cosas que existen en la dirección en la que se mira si se añade una brújula, como se muestra en la Figura 1.4.

Figura 1.2: Ejemplos de AR en los deportes que se visualizan por la televisión. De izq. a dcha.: baloncesto, tenis y fútbol

Con todo esto se puede obtener una aplicación de AR para una única persona, pero si se da el caso de que varias personas quieren compartir o visualizar la misma información simultáneamente, se necesita además un medio para que se comuniquen. A este concepto hacen referencia las llamadas tecnologías de AR *Colaborativa*, normalmente representada por las siglas CAR, de sus iniciales en inglés “Collaborative Augmented Reality”[2]. Se define CAR como toda tecnología de AR en la que interviene e interactúa más de un dispositivo simultáneamente. Dicha interacción puede ser local o remota. Se suele denominar interacción

[2]De aquí en adelante, se utilizará esas siglas para hacer referencia a la Realidad aumentada Colaborativa

Figura 1.3: Ejemplos de AR en educación. Observamos huesos virtuales del brazo superpuestos a un brazo real (foto izq.) y dinosaurios en 3D sobre un libro (foto dcha.)

Figura 1.4: Ejemplos de AR con GPS y brújula en los que se obtiene información sobre la zona que se está mirando por la cámara (foto izq.) y sobre la zona en la que se está (foto dcha.)

local cuando todos los participantes se localizan físicamente en el mismo sitio y, evidentemente, será remota cuando al menos un participante no está presente físicamente. Véase un ejemplo en la Figura 1.5, donde en la imagen de la derecha se observa una aplicación para móvil que proporciona información inmobiliaria sobre inmuebles disponibles en la cercanía del usuario, mientras que la foto de la izquierda muestra en funcionamiento una aplicación para turistas que proporciona información que les pueda ser útil, como la ubicación de cajeros, restaurantes... . Por lo tanto, en la colaboración remota, parte de la labor de la aplicación CAR será simular que todos los participantes comparten espacio físico. Un ejemplo de ello es la videoconferencia de (Billinghurst and Kato, 1999), en la que se simula con un sistema CAR una reunión en la que todas las per-

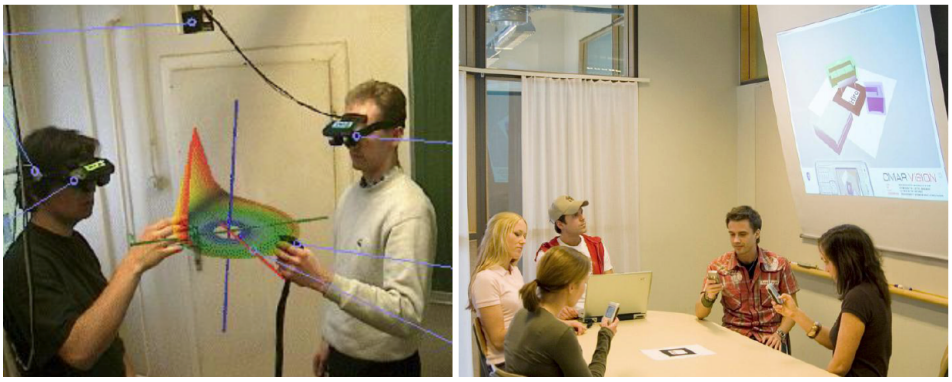

Figura 1.5: Foto izq. ejemplo CAR local. Foto dcha. ejemplo sistema CAR remoto

sonas están en la misma sala. Existen muchos más ejemplos de sistemas CAR, puesto que se utiliza en escuelas, talleres, videojuegos, museos, etc. .

Inicialmente el único dispositivo que permitía englobar todos los elementos mínimos para la existencia de AR o CAR era el ordenador personal (PC). Pese a ser un buen punto de partida, el tamaño y peso del PC impedía su uso en exteriores y limitaba el movimiento. A medida que la tecnología ha ido avanzando, se han ido utilizando dispositivos cada vez más pequeños, pasando por ordenadores portátiles, dispositivos HMD[3] que se montaban en cabeza y manos, etc.. En la actualidad, los más utilizados son los dispositivos móviles, entre los que se incluyen los teléfonos móviles de última generación o inteligentes ("smartphones") y tabletas. Estos dispositivos, pese a tener un tamaño reducido, disponen de todos los recursos necesarios para ejecutar aplicaciones CAR, entre los que se incluyen pantallas a color, procesadores potentes, cámaras con objetivos de varios megapíxeles de resolución, conectividades diversas e incluso GPU.

Como los teléfonos móviles *inteligentes* son relativamente recientes, sólo se han hecho algunas medidas de rendimiento a nivel de tecnologías relacionadas con la AR. Por lo que respecta a las aplicaciones CAR, según nuestro conocimiento en el momento de redacción de este documento de tesis doctoral, no se ha realizado ningún estudio de rendimiento a nivel de parámetros de prestaciones clásicas de los sistemas distribuidos como latencia, productividad, etc. y

[3] Del inglés "Head Mounted Display"

tampoco se ha realizado ningún estudio de escalabilidad. Para que las aplicaciones CAR puedan utilizarse en grandes entornos, es necesario un estudio y/o mejora de la escalabilidad y prestaciones que se pueden conseguir con estos sistemas cuando se usan teléfonos móviles como dispositivos cliente.

1.2. Objetivos

El objetivo principal de esta tesis doctoral consiste en mejorar las prestaciones de los sistemas CAR sobre dispositivos móviles. Este objetivo se puede dividir a su vez en los siguientes subobjetivos:

- Caracterizar los teléfonos móviles de última generación ("smartphones") presentes en el mercado actual, puesto que se ha comentado que son un dispositivo esencial en las aplicaciones CAR actuales.

- Desarrollar un entorno de trabajo completamente instrumentalizado que permita la simulación real de sistemas CAR, es decir, un simulador de CAR que recree cientos de clientes con un comportamiento idéntico al caracterizado en el punto anterior.

- Recrear sistemas CAR a gran escala, estudiando y eliminando los potenciales cuellos de botella que puedan existir.

1.3. Metodología

A la vista del problema planteado y de los sistemas que se pretende mejorar, la metodología a seguir se debe dividir en dos partes: una primera parte para los clientes del sistema CAR y otra para el servidor del mismo.

En la primera parte, centrada en los clientes del sistema CAR, se va a utilizar una selección representativa de los dispositivos móviles del mercado actual, poniéndolos a trabajar sobre aplicaciones reales e instrumentalizadas. Con esta

instrumentalización se medirán las prestaciones de los dispositivos móviles, obteniendo su caracterización. En esta fase no se realizarán mejoras, puesto que se pretende caracterizar los dispositivos móviles con el fin de poder reproducir su comportamiento a través de recreaciones virtuales de los mismos.

En la segunda parte, dirigida al servidor del sistema CAR, se desarrollará un sistema que permita recrear clientes CAR comunicándose a través de un servidor CAR. De esta forma, se podrán simular un gran número de dispositivos móviles, con comportamiento idéntico al caracterizado en la primera parte y, a su vez, que puedan interactuar entre ellos a través del servidor. Por lo tanto, parte de esta metodología consistirá en el desarrollo y validación del servidor de CAR, evaluando sus prestaciones y comprobando la escalabilidad del sistema.

1.4. Estructura de la tesis

El documento de tesis doctoral se organiza en siete capítulos. El primero de ellos corresponde a la introducción, donde se ha explicado la motivación y los objetivos de la tesis. Seguidamente, el Capítulo 2 introduce los conceptos necesarios para entender el funcionamiento general de los sistemas CAR. En él se enunciarán conceptos como las tecnologías AR, los sistemas CAR, los sistemas CAR sobre telefonía móvil y se presentarán los diferentes tipos de dispositivos móvil actuales. En el Capítulo 3 se discutirá el estado de la tecnología actual. En el Capítulo 4 se realizará la caracterización de los dispositivos móviles, eligiendo para ello los más representativos del mercado actual. Se instrumentalizarán aplicaciones reales y se medirán las prestaciones de los dispositivos móviles. En el Capítulo 5 se desarrollará el simulador de sistema CAR, que incluirá la creación del servidor de CAR, su validación y la evaluación de prestaciones del mismo. En el Capítulo 6 se propondrán una serie de modificaciones del sistema CAR desarrollado en el capítulo anterior, con el fin de mejorar sus prestaciones. Por último, en el Capítulo 7 se presentarán las conclusiones, las líneas de investigación abiertas y las publicaciones generadas en el transcurso de esta tesis.

2

Realidad Aumentada Colaborativa

2.1. Realidad Aumentada

2.1.1. Definición

Se define Realidad Aumentada, normalmente representada por las siglas AR, de sus iniciales en inglés “Augmented Reality”[1], como un vista en tiempo real de un entorno físico del mundo real que ha sido aumentado añadiendo información virtual generada por ordenador (AR, 2013). La Realidad Aumentada es la combinación interactiva del mundo real y virtual registrada en tres dimensiones en tiempo real. En (Milgram and Kishino, 1994) se define el Continuo Realidad-Virtualidad de Milgram como una escala continua que oscila entre lo que se pue-

[1] De aquí en adelante, se utilizará esas siglas para hacer referencia a la Realidad Aumentada

Figura 2.1: Representación del espacio continuo de Realidad-Virtualidad de Milgram

de definir como completamente virtual, es decir, la Realidad Virtual (RV del inglés "Virtual Reality"), y lo que es completamente real (la Realidad). Dentro de esta escala se encuentra la Realidad Aumentada y la Virtualidad Aumentada, normalmente representada por las siglas AV, de sus iniciales en inglés "Augmented Virtuality" (de aquí en adelante, se utilizará esas siglas para hacer referencia a la Virtualidad Aumentada). Como es obvio, la AR es la más cercana al mundo real mientras que la AV es la más próxima al mundo virtual. En la Figura 2.1 se representa este espacio continuo.

Las tecnologías de AR pretenden simplificar la vida del usuario aportándole información virtual no sólo de su inmediata proximidad, sino también de cualquier vista indirecta del mundo real, como vídeos en vivo. Las tecnologías de AR mejoran la percepción del usuario durante sus interacciones con el mundo real. Mientras que la tecnología de Realidad Virtual, o Entorno Virtual como se define en (Milgram and Kishino, 1994), sumerge completamente al usuario en un mundo sintético, ciego al mundo real, la tecnología de AR aumenta el sentido de realidad superponiendo objetos virtuales y señales al mundo real en tiempo real. Nótese que, como comentan Azuma et al. (Azuma et al., 2001), no se considera que la AR esté restringida a un tipo particular de tecnología de visionado como las HMD (Head-Mounted Display), ni se la considera limitada al sentido de la vista. La AR puede aplicarse potencialmente a todos los sentidos, aumentando también el olfato, tacto y el oído. La AR también se puede usar para aumentar o sustituir sentidos que no tenga el usuario, como aumentar la vista de los usuario ciegos o con visión deteriorada usando señales acústicas, o aumentando el sonido para usuarios sordos mediante señales visuales.

Azuma et al(Azuma et al., 2001) también consideran la posibilidad de que las aplicaciones de AR eliminen objetos reales del entorno, lo que se conoce

con el nombre de realidad disminuida. Este proceso se lleva a cabo añadiendo información virtual que coincida con el fondo, de forma que se le da al usuario la sensación de que los objetos no se encuentran en el entorno.

Los objetos virtuales que se añaden al mundo real muestran información que el usuario no puede detectar directamente con sus sentidos. Dicha información puede ayudar al usuario a realizar tareas diarias, como guiar a electricistas a través del cableado eléctrico de un avión mediante información digital visualizada con un HMD. O utilizar dicha información para fines lúdicos, como es el caso de Wikitude (Wikitude, 2013) u otras aplicaciones de realidad aumentada. Hay muchas otras clases de aplicaciones de AR, como visualización médica, entretenimiento, publicidad, mantenimiento y reparación, anotaciones, planificación de rutas para robots...

2.1.2. Historia

La primera aparición del término Realidad Aumentada surge en la década de los años '50', cuando Morton Heilig, un director de fotografía, pensó que el cine era una actividad que podría meter al espectador dentro de la acción que se ve en la pantalla si se engañaba a todos sus sentidos eficazmente. Así, en 1962, Helig construyó un prototipo de su idea, que había descrito en 1955 en "The Cinema of the Future", y lo llamó Sensorama, que fue el precursor del procesado digital (Sensorama, 2013). Más tarde, en 1966 Ivan Sutherland inventó el HMD[2] y desarrolló un sistema de Realidad Aumentada que hacía uso de un HMD transparente (historyAR, 2013). Se puede ver este sistema en la Figura 2.2 en la que se observa un hombre de pie en una habitación con el prototipo del casco montado sobre la cabeza. En 1975, Myron Krueger creó el Videoplace, consiguiendo por primera vez que un usuario interactuara con objetos virtuales. Después, Tom Caudell y David Mizell, de Boeing, acuñaron la frase Realidad Aumentada mientras ayudaban a trabajadores a ensamblar cables para un avión (AR, 2013). Fueron ellos los que empezaron a discutir las ventajas de las tecnologías de AR frente a las de VR, como el menor requerimiento de energía, puesto que necesita menos píxeles (Carmigniani and Furht, 2011). En el mismo año, L.B

[2]De sus siglas en inglés "Head Mounted Display", es decir, visor montado sobre la cabeza.

Figura 2.2: Foto del sistema HMD de Ivan Sutherland

Rosenberg desarrolló uno de los primeros sistemas de AR funcionales, al cual llamó Virtual fixtures y demostró que se conseguía una mejora en el rendimiento humano, medido con una batería de pruebas psicomotrices desarrollada por Fitts en (Fitts, 1954). Al mismo tiempo, Steven Feiner, Blair MacIntyre y Doree Seligmann presentaron el primer artículo sobre el prototipo de un sistema AR llamado KARMA (AR, 2013). El continuo Realidad-Virtualidad visto en la Figura 2.1 no se definió hasta 1994, por Paul Milgram y Fumio Kishino como la escala que abarca desde el mundo real al mundo virtual. En 1997, Ronald Azuma escribió el primer ensayo sobre AR, proporcionando la definición más ampliamente reconocida de AR, al identificarla como la combinación interactiva del mundo real y virtual registrada en tres dimensiones en tiempo real (historyAR, 2013). En 2000, Bruce Thomas desarrolló el primer juego de AR móvil funcional incluso al aire libre, ARQuake, que fue mostrado durante el simposio ISWC (del inglés “International Symposium on Wearable Computers”, es decir, Simposio Internacional de ordenadores portables). [3]. En 2005, el informe Horizon (Johnson and Smith, 2005) predijo que las tecnologías de AR tendrían su máximo desarrollo en los cinco años siguientes; y, para confirmar esa predicción, ese mismo año se desarrolló un sistema de cámaras que pueden analizar entornos físicos y relacionar las posiciones entre los objetos y el entorno en tiempo real. Este tipo de sistema con cámara se ha convertido en la base para integrar los objetos virtuales con la

[3]Hasta la fecha todos los juegos de AR se tenían que ejecutar en el entorno controlado de un laboratorio, con unas condiciones óptimas de iluminación.

realidad en los sistemas de AR. En los años siguientes el desarrollo de aplicaciones AR se disparó, sobre todo para dispositivos portátiles, como la guía de viaje de AR Wikitude, lanzada en 2008, y pensada para hacer turismo y acceder sobre la marcha a información del entorno en el que se encuentra el usuario. Con la aparición de los teléfonos móviles inteligentes y las tabletas se ha revolucionado el mundo de la AR y se ha originado un crecimiento exponencial de las aplicaciones AR. En el momento de la redacción de este documento de tesis doctoral, la cantidad total de las aplicaciones AR disponibles para las plataformas Android e iOS supera las 4000 unidades - datos extraidos del listado generado al buscar "Augmented Reality" en las plataformas "Google Play" de Android(Google Play, 2014) y "Apple Store" de iOS(Apple Store, 2014). Así mismo, un estudio realizado por el analista Michael Wiggins muestra que para 2017 se prevé que la cantidad de aplicaciones AR que se instalen en telefonía móvil supere los 2.5 billones al año (Wiggins, 2013).

2.1.3. Técnicas de visualización

La visión por computador renderiza objetos virtuales 3D tomando como origen el mismo punto de vista desde el que las cámaras están capturando las imágenes del mundo real (Carmigniani and Furht, 2011). El procesado de imágenes en AR se lleva a cabo usando diferentes métodos de visión por computador, generalmente relacionados con el análisis de vídeo. Estos métodos suelen consistir en dos fases: seguimiento y reconstrucción/reconocimiento. En primer lugar se detectan las marcas de referencia, las imágenes ópticas o los puntos de interés en las imágenes de la cámara. El seguimiento puede usar detección de características, detección de bordes, u otros métodos de procesado de la imagen para interpretar las imágenes capturadas. La mayoría de las técnicas de seguimiento disponibles en visión por computador se pueden agrupar en dos clases: basadas en modelo (Zhou et al., 2008) y basadas en características (Lee and Chun, 2010). Las técnicas basadas en modelo usan características de modelos de los objetos rastreados, como modelos CAD o plantillas 2D de los objetos basadas en características diferenciadoras. En la Figura 2.3 se muestra un ejemplo ilustrativo de este seguimiento, extraído de (Zhou et al., 2008). En las imágenes superiores se observa la colocación de un pilar y su reconocimiento como cilindro por parte del sistema y en las imágenes inferiores el dibujado del objeto virtual sobre la super-

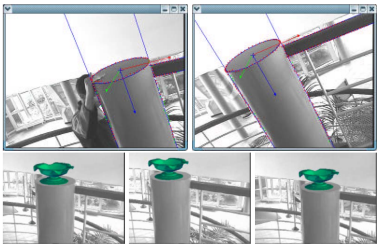

Figura 2.3: Ejemplo de seguimiento de modelos (Zhou et al., 2008)

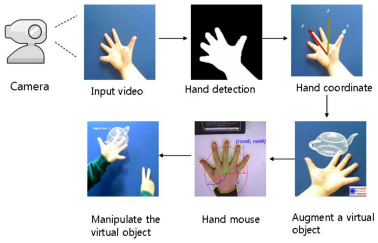

Figura 2.4: Ejemplo de seguimiento de características (Lee and Chun, 2010)

ficie reconocida. Los modelos basados en características consisten en descubrir la relación entre las características propias de la imagen 2D y sus coordenadas 3D en el mundo real. Tras conseguir una conexión entre la imagen 2D y el marco 3D del mundo real, es posible encontrar la posición de la cámara en la que, si se proyectan las coordenadas 3D en las coordenadas de la imagen 2D observada, la distancia que las separa sea mínima. Las restricciones para la estimación de la posición de la cámara se suelen determinar con funciones de punto. La fase de reconstrucción/reconocimiento utiliza los datos obtenidos en la primera fase para reconstruir el sistema de coordenadas del mundo real. Se puede ver un ejemplo del modelo basado en características en la Figura 2.4, extraído de (Lee and Chun, 2010) en el que se observan todos los pasos comentados, desde que se captura una imagen de la cámara (arriba-izquierda) hasta que se detecta y superpone un objeto virtual (abajo-derecha).

Algunos métodos asumen la presencia de marcas de referencia en el entorno u objetos con geometría 3D conocida y hacen uso de estos datos. Otros tienen la estructura 3D del escenario precalculada, como el dispositivo de visión AR de Huang et al (Huang et al., 2009); Sin embargo, el dispositivo tiene que ser fijo y

su posición conocida. Si la escena no se conoce completamente de antemano se utiliza la técnica conocida como SLAM[4] para mapear las posiciones relativas de marcas de referencia o modelos 3D. Si no se puede hacer suposiciones sobre la geometría 3D de la escena se usa el método SfM[5]. SfM puede dividirse en dos partes: rastreo de puntos característicos y estimación de parámetros de la cámara (Carmigniani and Furht, 2011).

En AR los métodos de rastreo/seguimiento dependen principalmente del tipo de entorno en el que se situe el dispositivo, así como del tipo del propio sistema AR. El entorno puede ser de interior, exterior o una combinación de ambos. Así mismo, los sistemas pueden ser móviles o estáticos (con una posición fija). Por ejemplo, si el sistema AR es estático para exteriores, como el propuesto por Huang et al (Huang et al., 2009), los desarrolladores pueden usar rastreo mecánico, puesto que todos los movimientos que tienen que rastrear son mecánicos y la posición del dispositivo es conocida. Este tipo de entornos y sistemas hacen más simple el rastreo para aumentar los alrededores. Por otro lado, si el dispositivo es móvil para exteriores, el rastreo se complica. Una de las técnicas utilizadas es la sugerida por Nilsson et al. (Nilsson et al., 2010), que consiste en construir un sistema de detección de peatones para evitar accidentes de coche usando AR. Aquí la cámara se mueve en un entorno desconocido y el problema para la visión por computador es reconstruir tanto el movimiento de la cámara como la estructura de la escena, usando para ello la imagen y los datos de los sensores. Como no se puede asumir nada de la geometría 3D de la escena, se utiliza SfM para reconstruirla.

Los desarrolladores también tienen la posibilidad de utilizar librerías AR ya existentes, como ARToolKit (Kato, 2011). ARToolKit se desarrolló en 1999 por Hirokazu Kato, del Nara Institute of Science and Tecnology, y se liberó por el laboratorio HIT de la Universidad de Washington. Es una librería de rastreo de visión por computador que permite al usuario crear aplicaciones de AR (Kato, 2011). Usa herramientas de rastreo de vídeo para calcular la posición y orientación relativas de la cámara respecto de marcas físicas en tiempo real. Tras calcular la posición real de la cámara, se coloca una cámara virtual en esa mis-

[4] De sus siglas en inglés *Simultaneous Localization And Mapping*, es decir, mapeo y localización simultáneos

[5] De sus siglas en inglés *Structure from Motion*, es decir, estructura procedente del movimiento

ma posición y, a partir de ese momento, se pueden dibujar modelos 3D sobre las marcas. La versión extendida de ARToolKit es ARToolKitPlus (Wagner and Schmalstieg, 2007), la cual mejora sustancialmente a su predecesora añadiendo, entre otras cosas, las APIs basadas en clases. Sin embargo, pronto le llegó un nuevo sucesor: el rastreador Studierstube (Szalavári et al., 1998).

El concepto de Studierstube es muy similar a ARToolKitPlus, pero su código base es completamente diferente y ya no es código abierto, por lo que no está disponible para descargar. Soporta teléfonos móviles con Studierstube ES, así como ordenadores, y sus requerimientos de memoria son muy bajos (100KB, un 5-10 % de los necesitados por ARToolKitPlus). Este sistema comercial es muy modular y los desarrolladores pueden extenderlo tanto como quieran. Cuando se presentó en (Schmalstieg et al., 2000), los diseñadores tenían en mente una interfaz de usuario que empleara tecnologías CAR como enlace de múltiples interfaces: múltiples usuarios, contextos, entornos, así como aplicaciones, ventanas 3D, plataformas de visualización y sistemas operativos. Se puede encontrar más información sobre Studierstube en (Schmalstieg et al., 2000) y (Schmalstieg et al., 2002).

2.1.4. Dispositivos de AR

Los dispositivos más comunes empleados en sistemas AR son visores, dispositivos de entrada, dispositivos de rastreo y computadores personales.

Los visores utilizados en AR se pueden agrupar en tres categorías: Visores portados en la cabeza (HMD), dispositivos de mano y espaciales. Los HMD se llevan sujetos a la cabeza o como parte de un casco, con lo que se tienen las imágenes del mundo real y virtual a la altura de los ojos del usuario. Se puede ver un ejemplo ilustrativo en la Figura 2.5 en el que se observa el casco propuesto en (Reitmayr and Schmalstieg, 2003) montado sobre la cabeza de un usuario. Los HMD pueden ser dispositivos de visión a través de vídeo u ópticas y pueden tener visores monoculares o binoculares. Los sistemas de visión a través de vídeo (denominados también transparentes de vídeo) son más costosos computacionalmente, puesto que obligan al usuario a cargar con dos cámaras sobre la cabeza y requieren el procesado de las dos cámaras para proporcionar la visión

Figura 2.5: Ejemplo del HMD utilizado en (Reitmayr and Schmalstieg, 2003)

conjunta del mundo real y los objetos virtuales que lo aumentan, mientras que los sistemas ópticos emplean un mecanismo de micro-espejos que les permite ver el mundo real a través de la lente junto con información superpuesta gráficamente que se refleja directamente en su ojo. Tanto la escena, como el mundo real, se perciben de manera mucho más natural que a través del visor. No obstante, en el sistema de visión a través de vídeo se observa directamente la imagen aumentada, lo que proporciona mucho más control sobre el resultado final. Así, en este tipo de sistemas se puede sincronizar la escena real con la virtual de forma que el usuario la perciba sin los retrasos propios del procesamiento o del renderizado, mientras que en los visores ópticos no se pueden retrasar la visión del mundo real, por lo que el usuario percibirá ese retraso. El resultado será una imagen con parpadeos, inestable que parecerá no estar bien fusionada con los objetos reales a los que se supone corresponden.

Por su lado, los visores de mano son pequeños ordenadores con pantalla que el usuario sostiene con sus manos. Actualmente este tipo de dispositivo corresponde a los teléfonos móviles de última generación. Véase un ejemplo en la Figura 2.6. Utilizan técnicas de visión a través de vídeo para superponer los gráficos al mundo real y utilizan sensores, como la brújula digital y el GPS para controlar sus seis grados de libertad, sistemas de marcas como ARToolKit y/o métodos de visión por computador como SLAM. En la actualidad existen dos tipos de visores de mano comercialmente disponibles: los móviles inteligentes (denominados “Smartphones”) y las tabletas (término derivado del inglés “Tablet computer”). Los móviles están extremadamente extendidos y son muy portables, y con los

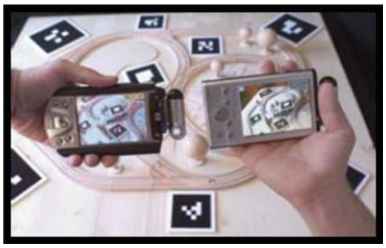

Figura 2.6: Ejemplo de visores de mano extraído de (Wagner and Schmalstieg, 2006)

avances tecnológicos actuales presentan una combinación de procesador potente, cámara, acelerómetros, GPS y brújula, haciendo de ellos una plataforma muy prometedora para AR. Sin embargo, el reducido tamaño de sus pantallas no los hace deseables como interfaz de usuarios 3D. Las antiguas PDAs presentaban muchas de las ventajas y desventajas de los teléfonos móviles pero su uso se ha reducido drásticamente con los avances recientes de los móviles basados en Android e iOS (Carmigniani and Furht, 2011). Las tabletas son mucho más potentes que los teléfonos móviles pero son considerablemente más caras e inicialmente eran muy pesadas para llevarlas en mano, incluso sosteniéndolas con las dos manos. Aunque poco a poco los fabricantes están lanzando dispositivos más ligeros. La solución parece estar a medio camino entre el móvil inteligente y la tableta, por eso los teléfonos móviles se lanzan al mercado cada vez con pantallas más grandes (de hasta seis pulgadas) y las tabletas cada vez son más ligeras y pequeñas. En este sentido, en el momento de redacción de este documento de tesis doctoral, el teléfono móvil con la pantalla más grande corresponde a *Huawei Ascend Mate* con una pantalla de 6.1 pulgadas (Huawei, 2014). Por su parte, la tableta más ligera es *SkyTex SKYPAD SP722* con 226 gramos de peso y una pantalla de 7 pulgadas (SkyTex, 2014).

Por último, la AR espacial (SAR) hace uso de proyectores, elementos ópticos, hologramas, etiquetas de frecuencia de radio y otras tecnologías de rastreo para mostrar información gráfica directamente sobre objetos físicos, sin necesidad de que el usuario cargue con ningún dispositivo. En la Figura 2.7 se muestra un ejemplo de SAR en el que una mujer se prueba ropa virtual gracias a un espejo con esta tecnología. Los visores espaciales separan la tecnología del usuario y la integran en el entorno. Por lo tanto, esto permite a los sistemas SAR un escalado

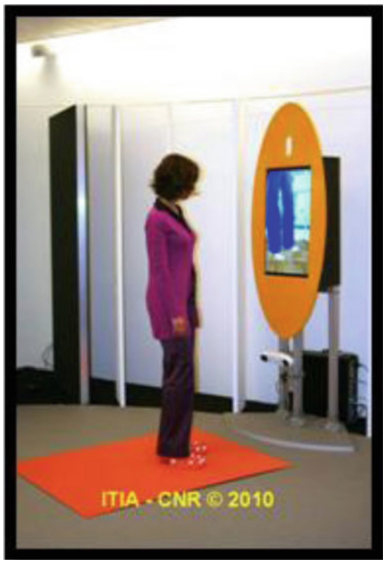

Figura 2.7: Ejemplo de AR espacial (SAR) propuesto en (ITIA-CNR, 2013)

natural hacia grupos de usuarios, permitiendo la colaboración entre ellos, incrementando así el interés por este tipo de sistemas de AR en universidades, laboratorios, museos y la comunidad artística. Existen tres aproximaciones diferentes para las tecnologías SAR que fundamentalmente difieren en cómo aumentan el entorno: visión a través de vídeo, ópticas y aumento directo. Los sistemas basados en visión a través de vídeo utilizan monitores. Se utilizan cuando el sistema no tiene que moverse, puesto que son muy asequibles al usar componentes típicos de ordenador. Los sistemas ópticos generan imágenes que se alinean con el entorno físico. Utilizan mezcladores como espejos planos o curvos divisores de haz, pantallas transparentes u hologramas ópticos. (Bimber et al., 2005). Al igual que los primeros, no son válidos para aplicaciones móviles. Finalmente, los visores espaciales basados en proyectores aplican proyección frontal de imágenes directamente en la superficie de los objetos físicos de la escena, como en (Mistry et al., 2009).

Las tecnologías de AR utilizan una gran variedad de dispositivos de entrada, como el sistema de AR móvil de Reitmayr et al. que utiliza guantes (Reitmayr and Schmalstieg, 2003). Otros sistemas como ReachMedia, propuesto en (Feldman et al., 2005), utilizan pulseras inalámbricas. En el caso de los móviles inteligen-

tes, el propio teléfono móvil se puede utilizar como puntero, como en la aplicación Google Sky Maps, que necesita que el usuario dirija su móvil en la dirección de las estrellas o planetas que quiere identificar. Los dispositivos de entrada dependen fuertemente del tipo de aplicación para el que se han desarrollado, así como del visor elegido. Por ejemplo, la aplicación puede requerir que el usuario tenga las manos libres, como en el caso del dispositivo de interacción por mirada de (Lee et al., 2010a), o las pulseras inalámbricas de (Feldman et al., 2005). Análogamente, si el sistema utiliza un visor de mano, los desarrolladores pueden utilizar una pantalla táctil como dispositivo de entrada.

Los dispositivos de rastreo/seguimiento consisten en una cámara digital y/u otros sensores ópticos, GPS, acelerómetros, brújulas, sensores inalámbricos, etc.. Cada una de estas tecnologías tiene un nivel diferente de precisión y depende del tipo de sistema que se esté desarrollando. En (Yi-bo et al., 2008), los autores identifican las tecnologías de rastreo/seguimiento más comunes como: mecánicas, sensores magnéticos, GPS, ultrasónicos, inerciales, y ópticas. Las principales ventajas del seguimiento mecánico son su exactitud, poco retraso y su inmunidad a perturbaciones visuales o magnéticas, por lo que son recomendables para el seguimiento de objetos pequeños. Pero tiene la desventaja de un rango de uso limitado. Por su parte, los sensores magnéticos son baratos, precisos, no tienen limitaciones visuales y tienen buena inmunidad al ruido, por lo que son ideales para seguimiento en entornos grandes. Tienen la desventaja de que los campos magnéticos y la presencia de objetos metálicos que pueda haber en el entorno afecta a su precisión. En cuanto al GPS, es ideal para entornos grandes al aire libre, pero no es muy preciso y sufre tiempos de espera elevados. En el caso de los ultrasonidos, son baratos, ligeros y no tienen perturbaciones magnéticas, aunque tienen poca precisión en grandes superficies. Por su parte, el sensores de inercia son muy rápidos, no sufren perturbaciones magnéticas ni visuales, son pequeños y baratos, y no tienen limitaciones de distancia. No obstante, sólo controlan tres grados de libertad y no son muy exactos con movimientos lentos. Por último, los sensores ópticos son fáciles de usar, tienen rangos de trabajo largos, son muy rápidos, sin perturbaciones magnéticas y de gran precisión. Sin embargo, tienen limitaciones de visión, son pesados y bastante caros.

Los sistemas de AR necesitan una CPU potente y una cantidad de memoria

RAM considerable para procesar las imágenes de la cámara. Normalmente se utilizaba un ordenador, incluso en los sistemas móviles, que se encargaba de realizar todos los cálculos y transmitr los resultados. Pero con la llegada de los teléfonos inteligentes y las tabletas se ha podido prescindir de ellos, y sólo se utilizan en los sistemas estacionarios.

2.1.5. Interfaces AR

Uno de los aspectos más importantes de la AR es crear técnicas apropiadas para una interacción intuitiva entre el usuario y el contenido sintético de las aplicaciones. Existen cuatro formas principales de interacción en las aplicaciones AR: interfaces tangibles, colaborativas, híbridas y la emergente interfaz multimodal (Carmigniani and Furht, 2011). A continuación se definen cada una de ellas.

Los interfaces tangibles para AR soportan interacción directa con el mundo real explotando el uso de objetos físicos y herramientas reales. Un ejemplo clásico del potencial de las interfaces tangibles es la aplicación VOMAR desarrollada por Kato et al. (Kato et al., 2000), donde se permite a las personas seleccionar y reorganizar el mobiliario de una sala de estar usando una paleta física. Los movimientos de la paleta están mapeados a gestos intuitivos basados en órdenes, como “recoger” un objeto para seleccionarlo y moverlo, o golpear un objeto para hacerlo desaparecer, proporcionando así una experiencia intuitiva.

Un ejemplo más reciente es TaPuMa (Mistry et al., 2008), una interfaz de sobremesa tangible que utiliza objetos físicos para interactuar con mapas digitales proyectados usando objetos de la vida real que el usuario suele llevar consigo como consultas para encontrar ubicaciones o información en el mapa. La principal ventaja de usar objetos como palabras clave es la eliminación de la barrera del lenguaje de las interfaces gráficas convencionales (puesto que aunque algunas tienen varios lenguajes disponibles, normalmente están mal traducidas). Por otro lado, esto puede ser ambiguo, puesto que puede haber más de un mapeo para acciones o informaciones posibles, y personas de diferentes sitios, edades y culturas asignan diferentes significados a los objetos. Aunque este sistema parece fácil de utilizar, abre la puerta al principal problema de las interfaces de usuario: mostrar al usuario cómo utilizar objetos reales para interactuar con el sistema. La

solución de White et al. (White et al., 2007) proporciona pistas visuales virtuales en los objetos reales, mostrando cómo deben ser movidos.

Otro ejemplo de interfaz AR tangible incluye el uso de guantes o pulseras como en (Feldman et al., 2005) y (Cooper et al., 2004), aunque la idea básica es la misma: añadir sensores a esa clase de objetos, de forma que se asignen comportamientos a determinados movimientos que se hacen con esos objetos en la vida cotidiana y que sirvan como interfaz para manejar una aplicación.

Los interfaces de AR colaborativas incluyen el uso de múltiples visores para llevar a cabo actividades tanto locales como remotas. La colaboración local se lleva a cabo con interfaces 3D que mejoran el espacio de trabajo físico. Para la colaboración remota, la AR es capaz, sin mucho esfuerzo, de integrar múltiples dispositivos desde diversas ubicaciones para mejorar las teleconferencias. Un ejemplo de colaboración local se puede ver en Studierstube (Schmalstieg et al., 2000) y (Schmalstieg et al., 2002). La primera vez que se presentó Studierstube (Schmalstieg et al., 2000), los desarrolladores tenían en mente una interfaz de usuario que "usara la Realidad Aumentada Colaborativa como unión entre una gran variedad de dimensiones dentro de los interfaces de usuario: múltiples usuarios, contextos y ubicaciones así como aplicaciones, ventanas 3D, anfitriones, plataformas de visualización y sistemas operativos".

La colaboración remota se puede usar para mejorar las teleconferencias como en (Barakonyi et al., 2004). O integrarse con aplicaciones médicas para realizar diagnósticos, cirugías, o incluso rutinas de mantenimiento.

Los interfaces de AR híbridas combinan un surtido de interfaces diferentes, aunque complementarias, así como la posibilidad de interactuar a través de un amplio abanico de dispositivos de interacción (Zhou et al., 2008). Proporcionan una plataforma flexible para interacciones sin planificación, que se hacen en la vida cotidiana, donde no se sabe a priori qué tipo de visor o dispositivo de AR se utilizará. En (Sandor et al., 2005), Sandor et al. desarrollaron un interfaz híbrido de usuario utilizando visores HMD para superponer la AR y proporcionar realimentación visual y auditiva. Su sistema de AR está implementado para permitir a los usuarios asignar interacciones físicas a operaciones así como para asignar objetos virtuales sobre los que realizar los procedimientos, y reconfigurar el ma-

peo entre dispositivos, objetos y operaciones conforme el usuario interacciona con el sistema.

Finalmente, los interfaces de AR multimodal combinan entradas de objetos reales con lenguaje natural y comportamientos como el habla, tacto, gestos naturales de las manos, o la mirada. Este tipo de interfaz es el más reciente. Algunos ejemplos incluyen la interfaz de gestos del "sexto sentido" del MIT (Mistry et al., 2009), llamada WUW. En este interfaz se le proporciona la información al usuario proyectándola en superficies, muros y objetos físicos a través de gestos de las manos, movimiento de brazos, e interacción con los propios objetos. Otro ejemplo de interacción multimodal es el trabajo de Lee et al. (Lee et al., 2010a), que utiliza miradas y parpadeos para interactuar con objetos. Este tipo de interacción se está desarrollando ampliamente y seguramente será el preferido en las aplicaciones de AR futuras, puesto que proporciona una relativamente robusta, eficiente, expresiva y muy portable interacción humano-ordenador que representa el estilo de interacción preferido por el usuario. Soporta la combinación flexible de modalidades o la conmutación entre modos de entrada en función de la tarea o configuración. Adicionalmente, las interfaces multimodales ofrecen la libertad de elegir el modo de interacción preferido por el usuario dependiendo del contexto; por ejemplo, sitios públicos, museos, bibliotecas, etc.. Esta libertad para escoger el modo de interacción es vital para una aceptación masiva (Encyclopidia, 2013).

2.1.6. Sistemas de AR

Los sistemas de AR se pueden dividir en cinco categorías: Sistemas fijos de interior, sistemas fijos de exterior, sistemas móviles de interior, sistemas móviles de exterior, y sistemas móviles de interior/exterior (Carmigniani and Furht, 2011). Se entiende por sistema móvil aquel que no restringe al usuario a una única habitación y que le permite moverse libremente sin usar cables. Por lo tanto, los sistemas fijos son aquellos que no permiten moverse al usuario y le obligan a usarlos en el lugar donde hayan sido colocados, sin flexibilidad para desplazarse, a menos que reubiquen todo el sistema. La elección de qué tipo de sistema hay que utilizar es la primera decisión que debe realizar el desarrollador, y le ayuda en la elección del sistema de rastreo, tipo de visor e interfaz que usará. Por ejemplo,

los sistemas fijos no usarán el rastreo por GPS, a diferencia de los sistemas móviles de exterior.

En (Carmigniani and Furht, 2011), los autores realizan un estudio sobre diversos sistemas de AR usando para ello 25 artículos clasificados en función del tipo de sistema, y determinan qué tipo de técnicas de rastreo, visores e interfaces usar en cada uno. Se puede encontrar otro estudio similar en (Papagiannakis et al., 2008), aunque con artículos más antiguos. Los usados para ese estudio se publicaron entre los años 2002 y 2010, aunque la mayoría (17 de 25) se publicaron entre los años 2005 y 2010. Aunque los resultados no se pueden usar como una regla general al construir sistemas de AR, sirven como indicador del tipo de técnicas de rastreo, visión, o interfaz más populares para cada tipo de sistema. Los desarrolladores deben tener en mente que estas elecciones dependen también del tipo de aplicación.

De este estudio se puede concluir que para sistemas fijos es preferible el rastreo óptico, mientras que para los móviles se prefiere una aproximación híbrida. Por otra parte, los visores HMD suelen ser los preferidos. En cuanto a los interfaces, los tangibles son la elección más popular, aunque la tendencia de mercado parece acercarse hacia los interfaces multimodales.

2.1.7. Aplicaciones

Aunque existen muchas posibilidades de uso para las tecnologías de AR, el presente estudio se enfocará en los cuatro tipos de aplicación en los que se suele centrar la investigación sobre AR: comerciales y publicitarias, educativas y de entretenimiento, médicas, y aplicaciones para móvil. Más abajo se estudiará por qué la AR podría proporcionar una mejor solución en algunas áreas, una solución más barata en otras, o simplemente crear un nuevo servicio. También se comentarán los desafíos que están afrontando las tecnologías de AR para pasar de los laboratorios a la industria.

En cuanto a las aplicaciones comerciales y publicitarias, la AR se utiliza muchas veces para promoción online de productos nuevos. La mayoría de las técnicas incluyen unas marcas que el usuario pone frente a su cámara web y, a través

Figura 2.8: Anuncio de vehículo Mini con modelo 3D superpuesto con AR, extraído de (Cool; Augmented Reality Advertisements, 2013)

de aplicaciones específicas o de la propia web de la compañía que hace la publicidad, visualiza la publicidad superpuesta sobre dichas marcas (normalmente modelos 3D animados del producto). Por ejemplo, en diciembre de 2008, Mini (Mini, 2013), la conocida marca de coches propiedad del grupo BMW, lanzó una campaña de publicidad con AR en muchas revistas de coches alemanas (Cool; Augmented Reality Advertisements, 2013). El lector sólo tenía que ir a la página web de Mini, mostrar el anuncio de la revista frente a su cámara web, y un modelo en 3D del vehículo aparecía en su pantalla, como puede observarse en la Figura 2.8.

La AR soluciona también el problema de la construcción de prototipos. Las industrias se enfrentan a menudo con la costosa necesidad de realizar productos antes de su comercialización para averiguar si necesitan efectuar algún cambio o si el producto cumple con las expectativas iniciales que propiciaron su desarrollo. Si se decide que hay que hacer cambios, se tiene que realizar otro prototipo, con el consiguiente gasto de tiempo y dinero. Un grupo del Instituto de Tecnologías Industriales y Automatización (ITIA) del Consejo Nacional de Investigación italiano (ITIA-CNR, 2013) en Milán, trabaja con sistemas de AR y VR como herramienta de soporte para prototipos virtuales. Se puede ver un ejemplo en la Figura 2.9, en el que se ve el prototipo virtual de una moto junto a su versión física. Otros ejemplos aparecen en las Figuras 2.10 y 2.11, que muestran el acabado de una fábrica y oficina virtuales, o la Figura 2.12, que muestra una simulación virtual de iluminación.

Otro ejemplo es el Espejo Mágico, también desarrollado por el ITIA en Milán (ITIA-CNR, 2013). Este espejo se combina con un calzado tecnológico para

Figura 2.9: Foto izquierda: imagen de un prototipo de moto virtual (a la derecha) junto a un prototipo físico (a la izquierda) en un entorno real. Foto derecha: prototipo de moto virtual en un entorno virtual (ITIA-CNR, 2013)

Figura 2.10: Prototipo virtual de una fabrica (ITIA-CNR, 2013)

Figura 2.11: Prototipo virtual de una oficina (ITIA-CNR, 2013)

Figura 2.12: Prueba virtual de un sistema de alumbrado (ITIA-CNR, 2013)

mediciones y permite al usuario “probarse” virtualmente el calzado. Con este sistema el usuario puede ver cómo le queda el calzado a través del Espejo Mágico, que visualiza su reflejo junto con un modelo virtual del calzado que se quiera probar, como muestra la Figura 2.13.

Las aplicaciones AR educativas y de entretenimiento incluyen aplicaciones culturales para turismo o guías de museo, juegos tradicionales usando interfaces de AR y algunas aplicaciones para móviles inteligentes que hacen uso de la AR con fines educativos o lúdicos. En las aplicaciones culturales, existen unos pocos sistemas que utilizan AR para reconstruir virtualmente ruinas antiguas, como en (Huang et al., 2009) donde superponen un modelo 3D de la arquitectura original de Dashuifa a las ruinas que existen en la actualidad, como se ve en la Figura 2.14. Este tipo de sistema también se utiliza para instruir virtualmente a usuarios sobre la historia de algún lugar concreto, como en (Malaka et al., 2004). Existen otros sistemas que utilizan las tecnologías de AR para hacer de guía virtual en museos, como se muestra en (Miyashita et al., 2008) y (Bruns et al., 2007), de los que observamos ejemplos en las Figuras 2.15 y 2.16, respectivamente. La Figura 2.15 muestra a un usuario que obtiene información de una vitrina de museo a través de su móvil. Por su parte, la Figura 2.16 muestra una

Figura 2.13: Usuario probándose un zapato virtual frente al Espejo Mágico (ITIA-CNR, 2013)

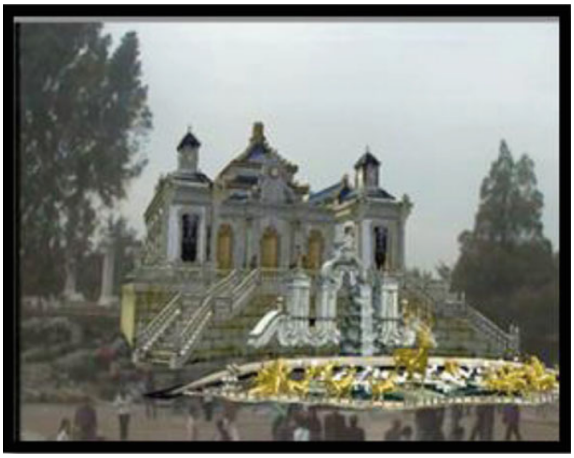

Figura 2.14: Vista aumentada de Dashuifa de (Huang et al., 2009)

tableta haciendo de guía virtual de un usuario dentro de un edificio.

La AR se puede utilizar como herramienta de apoyo en el aprendizaje, en materias como historia, matemáticas... . Por ejemplo, en (Billinghurst et al., 2001) desarrollaron el "Libro Mágico", un libro en cuyas páginas se encuentra tecnología simple de AR que proporciona una lectura más cautivadora. O (Kretschmer et al., 2001), donde construyen un sistema de AR móvil de exterior que hace uso de su proyecto anterior (GEIST) para proporcionar asistencia en el aprendizaje de historia a través de un juego cuenta-cuentos donde el usuario puede liberar fantasmas del pasado.

Los juegos que emplean tecnologías basadas en AR presentan muchas ventajas frente a las típicas alternativas de tablero físico con, por ejemplo, la habilidad de introducir animaciones y otras presentaciones multimedia. En (Cooper et al., 2004), los autores crearon un juego de damas chino, llamado ARCC, que usa tres marcadores de referencia y una cámara fija en el techo para rastrear dichos marcadores. Dos de los marcadores se usan para averiguar la posición del tablero y la tercera para manipular las piezas del juego. La Figura 2.17 muestra un ejemplo de este juego en el que se aprecia el tablero de juego virtual.

La mayoría de las aplicaciones médicas usan la guía por imágenes y la cirugía asistida por robots. Como resultado, se ha realizado una gran investigación

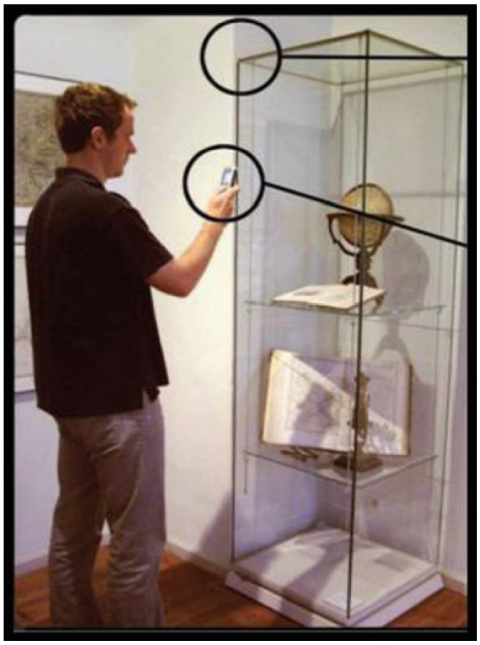

Figura 2.15: Guía a través de teléfono móvil en un museo de (Bruns et al., 2007)

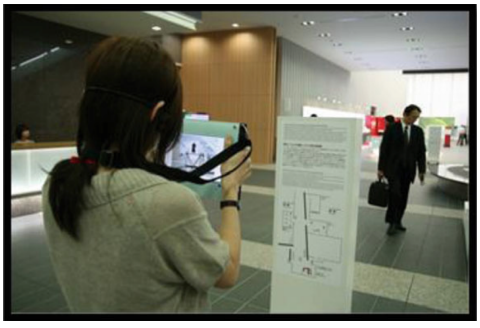

Figura 2.16: Visitante con sistema de guía de (Miyashita et al., 2008)

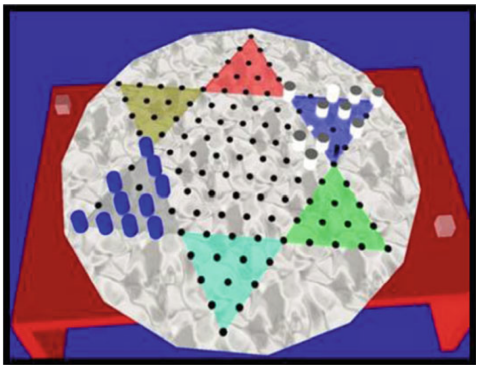

Figura 2.17: Ejemplo del juego ARCC propuesto en (Cooper et al., 2004)

para incorporar la AR con imágenes médicas e instrumentos para incorporar las habilidades intuitivas de los médicos. Se han hecho muchos progresos con este tipo de herramientas como visualizar las imágenes grabadas por una cámara de endoscopias sobre el propio paciente, a través de un monitor. Aunque este progreso también limita la vista 3D natural, directa e intuitiva que el cirujano tiene del cuerpo humano, puesto que ahora tiene que utilizar un monitor (Bichlmeier et al., 2007). El equipo médico puede usar la AR para observar imágenes en tiempo real durante el procedimiento. En (Bichlmeier et al., 2007) crearon un sistema de AR para ver debajo de la piel real la anatomía virtual, usando modelos de superficie poligonal para la visualización en tiempo real. Se puede ver un ejemplo en la Figura 2.18 en la que se observa la cara de una paciente y la superposición virtual de los huesos del cráneo que utiliza el médico para su diagnóstico.

En cuanto a aplicaciones para móviles, un ejemplo significativo es Wikitude-Drive (Wikitude, 2013), que es una aplicación similar a las clásicas aplicaciones de geo-referenciación basadas en GPS que permiten al usuario mantener su mirada en la carretera mientras observa el móvil, como muestra la Figura 2.19 en la que se observa un teléfono móvil fijado a un parabrisas y proporcionando indicaciones de navegación al usuario. Otro ejemplo sería la aplicación “Le Bar Guide” que tiene una función de navegación que guía al usuario hasta el bar más próximo que sirva cerveza “Stella Artois”. Se puede ver un ejemplo ilustrativo en la Figura 2.20 en la que se observa el bloque de edificios que está visualizando el usuario en su móvil y la ubicación del bar más próximo al que puede ir a degustar la cerveza.

Figura 2.18: Sistema para ver a través de la piel propuesto en (Bichlmeier et al., 2007)

Figura 2.19: Sistema de navegación propuesto en WikitudeDrive (Wikitude, 2013)

Figura 2.20: Ejemplo de utilización de la aplicación “Le Bar Guide” propuesta en (Mashable, The Social Media Guide, 2013)

Pese a que las aplicaciones AR sobre móvil llevan funcionando una década, no existe un gran variedad de librerías que faciliten a los desarrolladores la inclusión de AR. Entre las más conocidas se encuentran ARToolKit Plus (Wagner and Schmalstieg, 2007), Studierstube ES (Schmalstieg et al., 2002) y Vuforia (Qualcomm, 2012), ordenadas cronológicamente. No se entrará en detalles aquí, puesto que se describirán en el Capítulo 3.

2.2. Realidad Aumentada Colaborativa

Hasta ahora se ha visto en detalle las tecnologías involucradas en la denominada AR y las aplicaciones que la utilizan, cuyo ciclo de trabajo se puede dividir en tres fases: Captura de la imagen a través del flujo de entrada de la cámara, obtención de la posición y orientación de la cámara, y dibujado de los objetos

3D que *aumentan* la imagen percibida por el usuario, en la posición y orientación calculados en la etapa anterior. En este sentido, para convertir la AR en AR Colaborativa, normalmente representada por las siglas CAR, de sus iniciales en inglés “Collaborative Augmented Reality”[6], es necesario añadir una nueva fase, que se suele denominar de *comunicaciones*, en la que los usuarios intercambian algún tipo de información. Dicha información se puede transmitir por el canal de comunicación inalámbrica que mejor convenga, pero los usuales suelen ser: Wi-fi, Bluetooth o 3G. De estos, el menos usado es el Bluetooth, puesto que limita el rango de transimisión (Schmeil and Broll, 2007). La Figura 2.21 muestra las fases en las que se descompone el ciclo de trabajo de las aplicaciones CAR.

Una de las primeras aplicaciones CAR consiguió una mejora significativa de los sistemas de conferencia, dando la sensación de presencia real de los colaboradores remotos (Billinghurst and Kato, 1999). El sistema de Transvision de Rekimoto mostró cómo compartir objetos virtuales a través de dispositivos de mano (Rekimoto, 1996). A su vez, Schmalstieg creó una arquitectura de software para desarrollar aplicaciones CAR con facilidad (Szalavári et al., 1998); mientras que Billinghurst mejoró la Computación Colaborativa en (Billinghurst et al., 2000).

[6]De aquí en adelante, se utilizará esas siglas para hacer referencia a la Realidad Aumentada Colaborativa

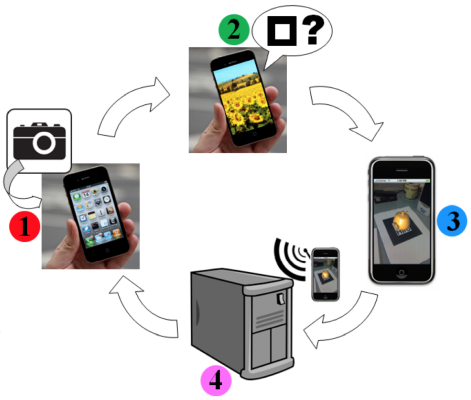

Figura 2.21: Descripción de las etapas presentes en las aplicaciones CAR consistentes en: 1) captura de la imagen, 2) detección de marcas y extracción de coordenadas, 3) dibujado de la información virtual, 4) envío de la información al servidor

2.3. Realidad Aumentada Colaborativa sobre teléfonos móviles

Cuando se desarrollan aplicaciones CAR, se desea ejecutarlas sobre un dispositivo de tamaño mínimo y lo más ligero posible, de forma que permita desplazarse líbremente por el entorno e interactuar con otros usuarios. Uno de los dispositivos más extendidos actualmente es el teléfono móvil de última generación ("smartphone"). Los teléfonos móviles actuales tienen una cámara de más de 5 megapíxeles, al menos un procesador, con una potencia de al menos 500MHz (llegando incluso a 8 núcleos de más de 2 GHz), pantalla a color de 6 pulgadas o más, GPS (Sistema de Posicionamiento Global), Brújula digital, acelerómetros, GPU (Unidad de Procesamiento Gráfico)... . En resumen, una gran variedad de componentes en el peso de unos pocos cientos de gramos. Existe una gran variedad de teléfonos móviles, con diferentes sistemas operativos y capacidades, abanderados por diferentes grandes compañías líderes en el sector de las TIC (siglás de las Tecnologías de la Información y la Comunicación). Los sistemas operativos más extendidos entre los teléfonos móviles son: Apple iOS, Google Android OS, Microsoft Windows Mobile/Phone 8, Nokia Symbian, RIM/Blackberry

y Samsung Bada (Ahonen, 2010).

Como se ha comentado anteriormente, el ciclo de trabajo de las aplicaciones CAR se dividían en cuatro fases: la captura de la imagen, la detección de marca/características, el dibujado (3D) y el envío de información. Cabe recordar aquí que la segunda fase del ciclo tiene dos variantes: la basada en detección de marcas y la basada en vector de características, aunque ya se explicaron en el primer punto del presente capítulo. Para los teléfonos inteligentes se ha optado mayoritariamente por la primera variante, puesto que para la segunda variante se necesita más capacidad de procesamiento. Otro factor a favor de esta variante es que la librería ARToolKit es libre, por lo que la realización de aplicaciones AR con ella se facilita en gran medida.

Existen algunas soluciones basadas en detección de marcas sobre teléfonos inteligentes. En 2003 ARToolKit (Kato, 2011) se desarrolló para Windows CE, y se desarrolló la primera aplicación de AR auto-contenida para teléfono móvil (Wagner and Schmalstieg, 2003). Este software evolucionó después a la librería ARToolKitPlus (Wagner and Schmalstieg, 2007). Después se desarrolló una propuesta para teléfonos móviles que funciona con marcas 3D codificadas con color (Mohring et al., 2004), y una versión de ARToolKit para el sistema operativo Symbian, parcialmente basada en el código fuente de ARToolKitPlus (Henryssson et al., 2005). El equipo de investigación que hay detrás de estos trabajos se centra en detección de marcas, pero no desde el punto de vista colaborativo.

En cuanto a la segunda variante, la detección de características o sistema sin marcas (en inglés “markerless”), existen algunos trabajos, como el realizado en (Srinivasan et al., 2009) donde se consigue triplicar el rendimiento del modelo MAR (*Mobile Augmented Reality*). En (Wagner et al., 2008b) se presentan dos técnicas orientadas a la descripción de características en tiempo real sobre móviles que consigue un rendimiento de 20 FPS [7]. Por su parte, en (Wagner et al., 2009) muestran un método de seguimiento y estimación de posición en tiempo real para móviles que permite seguir a seis objetos simultáneamente a 23 FPS. Finalmente, en (Wagner et al., 2010) se consigue optimizar los modelos de descripción de características SIFT y Ferns obteniendo tiempos interactivos, con una frecuencia de más de 30 FPS.

[7] Siglas en inglés de *Frames por segundo*, es decir, imágenes por segundo

Se comentará en detalle estas librerías en el capítulo del estado de la tecnología (Capítulo 3).

3

Estado de la tecnología

A lo largo de este trabajo se ha comentado brevemente la historia de las tecnologías de Realidad Aumentada y los sistemas CAR y se han enumerado los trabajos de investigación más representativos de las mismas. Si bien, como el presente es un caso muy concreto de CAR, centrada en teléfonos móviles inteligentes, cabe la pena profundizar en las investigaciones que tienen como foco estos dispositivos. Por lo tanto, en este capítulo se comentarán las publicaciones más destacadas que sirven de base para el presente trabajo de investigación.

El proyecto MARS[1], presentado en 1999 por la Universidad de Columbia (Höllerer et al., 1999) fue el primer sistema de CAR que realmente se podía considerar como *móvil*, puesto que su configuración permitía al usuario moverse líbremente por el entorno, llevando todo el equipo necesario sobre sus hombros. En

[1] Del inglés “Mobile Augmented Reality Systems”

este trabajo se propone un banco de pruebas con cuatro interfaces de usuario, dos de exterior y dos de interior. Las interfaces de exterior se usan a la vez y se componen de un casco con pantalla, cámara, GPS y acelerómetros que permiten al usuario visualizar el mundo real y información 3D virtual superpuesta, así como una interfaz de mano, compuesta por una pantalla para ver objetos 2D y un puntero para interactuar con ella. La aplicación propuesta se utiliza para desplazarse por el campus de la universidad y dispone de dos vistas: si se contempla por la pantalla del casco se observan los posibles destinos a los que dirigirse, y si se contempla la pantalla de la mano se visualiza un plano de la universidad con información detallada de la misma. En cuanto a las interfaces de interior se puede elegir entre un entorno 3D de escritorio, para señalar ubicaciones del mapa dinámicamente y llevar un registro de actividad del usuario que se encuentra en el exterior o su versión inmersiva, con un caso y pantalla como el exterior. Como se comentó anteriormente, es el primer ejemplo de colaboración móvil con AR, puesto que el usuario de interior puede añadir rutas u objetivos al mapa y el usuario que deambula por el campus las visualizará automáticamente en sus interfaces pudiendo actuar en consecuencia. El sistema está un poco restringido, puesto que emplean un GPS con precisión de centímetros, pero que sólo funciona en el área delimitada por el campus, al igual que la red de interconexión entre los usuarios. Así mismo, la latencia en la actualización de las posiciones del usuario de exterior es tal que no pueden dibujar los movimientos que realiza en tiempo real.

Pronto aparecieron múltiples plataformas examinando las áreas de aplicación para este sistema, como BARS (Yohan et al., 2000), Tinmith (Piekarski and Thomas, 2001), Studierstube (Schmalstieg et al., 2002) o el sistema Bat[2] (Newman et al., 2001).

BARS[3] es un sistema realizado en colaboración con la Universidad de Columbia (EEUU) con el que se pretende facilitar las operaciones militares en territorios urbanos. El sistema consiste en un ordenador portátil, un sistema de comunicaciones inalámbrico y un visor HMD. Con este sistema se le proporcionan datos al usuario, como ubicaciones conocidas de francotiradores, en forma de iconos en la pantalla o mapas de la zona con los objetivos de las misiones. Pese a que

[2]Murciélago en inglés
[3]Del inglés Battlefield Augmented Reality System

en trabajos anteriores Yohan et al. indicaban que las líneas de investigación se debían dirigir a mejorar el seguimiento, el diseño de la interfaz de usuario y la interacción del usuario (Julier et al., 1999), con este prototipo sólo abordan parcialmente las dos primeras. No obstante, Yohan et al. aportan ideas interesantes como el sistema de información selectiva. Este sistema selectivo utiliza filtros que priorizan la información de forma que sólo transmite a los usuarios los datos que les afectan y, en caso de duda, se les muestra la información más útil en función de su perfil. También aportan un sistema de calibrado, de forma que cada usuario final puede adecuar el equipo a sus necesidades específicas.

La plataforma Studierstube estaba operativa a principios del año 2000; su diseño incluía un ordenador colgado en la espalda, un casco de visión a través (cámara más pantalla) y una pantalla táctil en la muñeca, junto con marcas ARToolKit y un sensor de inercia. Con todos estos componentes se crea un guía virtual que indica el camino a utilizar para desplazarse de un sitio a otro dentro de edificios, sin ayuda de GPS[4]. El sistema incorpora un esquema muy elaborado de marcas, que permite la reutilización de las mismas en habitaciones que no están en línea, por lo que con muy pocas marcas pueden abarcar todo el edificio. Han hecho todas las marcas del mismo tamaño (20x20 cm) y utilizan el tamaño de la marca captado por la cámara para saber a qué distancia se encuentran de la misma y poder ubicarse mejor en el mapa. El sistema depende de la correcta identificación de las marcas y no se reconocen por igual en todas las habitaciones del edificio, puesto que las condiciones de iluminación son diferentes en cada una. Por lo tanto, se encuentran con un problema que de momento no puede abordar su infraestructura y que debería solventarse con umbrales lumínicos adaptables. Por su parte, el algoritmo para encontrar el camino de un sitio a otro se basa en grafos adyacentes, lo que permite encontrar eficazmente el camino más corto posible.

En 2001 surgió la plataforma Tinmith, diseñada como una posible metodología para desarrollar aplicaciones AR complejas. Piekarski y Thomas proponen una arquitectura multicapa basada en un modelo de flujo de datos, descomponiendo las aplicaciones en cientos de objetos que realizan acciones específicas. En su nivel más bajo, el modelo de flujo de información se implementa usando punteros de “callback” dinámicos que pueden añadirse o eliminarse en tiempo de

[4] Abreviatura de Sistema de Posicionamiento Global, del inglés “Global Position System”

ejecución, pero al hacer uso de estos punteros para comunicarse en niveles superiores se produce una pequeña sobrecarga si se comunican muchos objetos. Siempre que se puede, el sistema se ejecuta secuencialmente, aunque ofrecen la posibilidad de separar los objetos en diferentes procesos o hilos, lo que repercute negativamente en el rendimiento del sistema. Con esta arquitectura software pretendían proporcionar soporte software para el desarrollo de entornos 3D y así lo hace siempre que el sistema sea pequeño, aunque para sistemas a gran escala el rendimiento decrece. En los niveles más altos de la arquitectura proponen un sistema de dibujado muy elaborado, completamente jerarquizado, compuesto por un grafo de escena de los objetos y nodos de transformación, así como un motor de construcción de geometría sólida en tiempo real. Otra curiosidad son los guantes con seguimiento 3D que incluyen para su interfaz de usuario, lo que permite construir modelos de forma fácil e intuitiva tanto en entornos abiertos como cerrados.

Por último, Newman et al. desarrollan en 2001 el sistema Bat para la compañía AT&T, con una visión diferente; para los autores las marcas en el entorno son *contaminación visual*, por lo que proponen un sistema con un entorno al que llaman *sensible*, puesto que comparte la percepción del mundo que tienen los usuario, cuyo modelo implementa una colección de objetos software desarrollados empleando CORBA[5]. En el momento de realizar las pruebas, el model del entorno tenía más de 1900 objetos software que incluían al personal, teléfonos, ordenadores, muros, ventanas, ... lo que supone una gran cantidad de trabajo inicial. En cuanto al sistema de localización, utilizan dispositivos inalámbricos de tamaño reducido a los que llaman *murciélagos*, que colocan por todo el edificio y sobre las personas. Dichos *murciélagos* emiten los ultrasonidos que captará el sistema sensor para obtener la posición y orientación del usuario. En las pruebas han utilizado más de 200 *murciélagos* y se obtienen ubicaciones con un error menor de tres centímetros el 95 % de las veces. Cabe mencionar que ninguna de las variantes incluye sensor de inercia, por lo que la posición y orientación del usuario se actualizan pocas veces por segundo. Pese a ser un sistema novedoso y proponer una idea interesante, conlleva un coste de desarrollo nada despreciable si se compara, por ejemplo, con imprimir las marcas de ARToolKit. Proponen dos variantes de este sistema, una inmersiva con un casco de visión a

[5]Lenguaje de desarrollo de alto nivel

través (como los vistos hasta ahora) y otra que sólo utiliza una PDA[6]. Esta última variante trata la PDA como un cliente *ligero*[7], por lo que el cálculo se hace en un ordenador de sobremesa y a la PDA se le manda directamente el resultado. La principal ventaja de la variante en PDA es que es muy ligera de transportar, pero a cambio tiene la sensación de ver una pantalla más estrecha, a la que han dado un efecto de lente de *ojo de pez*, por lo que se pierde bastante precisión en la información mostrada.

Todos estos sistemas considerados como *móviles*, pese a decir que tenían todo lo necesario para realizar AR equipado en la espalda, utilizaban el procesador de un ordenador de sobremesa para realizar los cálculos más intensivos. Por lo tanto, actuaban como clientes *ligeros*, lo que significa que no eran autosuficientes. La comunicación entre el cliente y el servidor podía ser cableada o inalámbrica. Un ejemplo del primer caso es el prototipo *Chamaleon* propuesto por George W. Fitzmaurice, de la universidad de Toronto (Fitzmaurice, 1993), consistente en un pequeño dispositivo dotado de pantalla que permite acceder, modificar e interactuar con la información almacenada en el ordenador. Este sistema es precursor de las PDA y usaba un ordenador de sobremesa para realizar todos los cálculos y mostrar los resultados. Dichos resultados llegaban a la pequeña pantalla a color (de sólo cuatro pulgadas) a través de un cable conectado a una videocámara, que estaba grabando todo lo que salía del monitor del ordenador, por lo que era un sistema muy primitivo. Otro ejemplo de sistema cableado es el *AR Pad* propuesto en (Mogilev et al., 2002), donde parten del trabajo realizado por Rekimoto en *Transvision* (Rekimoto, 1996) y del realizado por Regenbrecht en mPARD (Regenbrecht and Specht, 2000) para proporcionar un sistema compuesto por una pequeña pantalla, una cámara de vídeo y un mando analógico que les permite interactuar con la información obtenida del ordenador al que va conectado. Por lo tanto, Regenbrecht y Specht son de los primeros que profundizan en la interacción con el usuario, permitiendo que los objetos virtuales se puedan rotar y trasladar usando el mando analógico. En la Figura 3.1 se pueden ver imágenes de la utilización de los resultados de los sistemas Chamaleon y AR-Pad.

En cuanto a los sistemas *sin cables*, se puede mencionar el propuesto en

[6] Asistente personal digital, de sus siglas en inglés: “Personal Digital Assistant”
[7] También conocido por su nombre en inglés: “thin client”

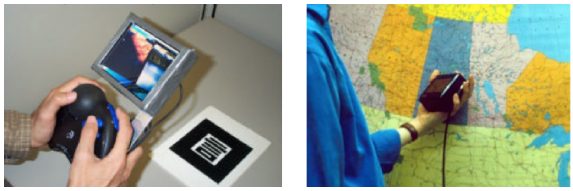

Figura 3.1: A la izquierda ejemplo de uso de Chamaleon y a la derecha ejemplo de uso de AR Pad

(Gausemeier et al., 2003) por Gausemeier et al., donde desarrollan un método de reconocimiento de objetos basado en imágenes, en tiempo real, dentro del proyecto AR-PDA (AR-PDA, 2009), que utilizan junto a una PDA o móvil de tercera generación (con cámara incorporada) para conectarse inalámbricamente a un ordenador de sobremesa. El proyecto AR-PDA surge de una iniciativa del Ministerio de Investigación alemán en el año 2000, en la que pretendían desarrollar sistemas hardware y software para que las PDAs se usasen, junto con tecnología de AR, como asistentes tecnológicos en las tareas diarias de cualquier usuario. Gausemeier propone un sistema de reconocimiento basado en geometrías 3D, que intenta encontrar las formas 3D que mejor cuadren con los objetos que se observan a través de la cámara. Aunque es una alternativa interesante desde la perspectiva del reconocimiento, resulta ser computacionalmente costosa. De hecho, se ven obligados a utilizar heurísticas para reducir la cantidad de objetos 3D posibles para cada forma detectada y, aunque aseguran que limitando las posibilidades a 63 se conseguiría un tiempo de respuesta interactivo (funcionando a 16 FPS), al final sólo consiguen limitarlas a 100 posibilidades (obteniendo 10 FPS). Por lo tanto, era un sistema muy primitivo que funcionaba en tiempo real sólo en entornos muy controlados y limitados. En la Figura 3.2 se puede ver un ejemplo de utilización de este sistema en el que se hace uso de una PDA para extraer información de un mueble de cocina con un horno incorporado.

Dado que las prestaciones de los computadores personales mejoran rápidamente, llegó un momento en el que el ordenador portátil que lleva consigo el usuario de AR ya tenía potencia de cálculo suficiente para no tener que comunicarse con otro de sobremesa. Esto se muestra en el *Touring Machine* propuesto por investigadores de la Universidad de Columbia (Feiner et al., 1997) donde, haciendo uso de hardware comercial, desarrollaron una aplicación que hacía de

Figura 3.2: Ejemplo de utilización de AR-PDA

guía turístico del campus universitario. Este hardware incluía un casco con pantalla 3D, GPS y cámara, un ordenador cargado en la espalda y una pantalla 2D de mano, con puntero y trackpad. La pantalla de mano sirve a la vez para interactuar con el usuario y para mostrar los objetos que requieren mejor resolución, puesto que la pantalla del casco es un poco limitada. Aunque es un prototipo no colaborativo, se comenta aquí porque es de los primeros que no necesita ordenador de sobremesa para realizar el seguimiento. También incorpora ideas interesantes en cuanto a las interfaces AR, como el hecho de moverse por los menús de la interfaz utilizando el movimiento de la cabeza. Aunque se trata sólo de un prototipo, tiene algunas carencias que cabe destacar como son los problemas en el rastreo (al utilizar hardware poco preciso), la calidad de las pantallas, que no tienen suficiente brillo para trabajar en exteriores, y la pérdida de señal del GPS, que no ha sido gestionada.

Otro ejemplo es ARQuake (Thomas et al., 2000), donde sus creadores presentan una versión del famoso juego de ordenador Quake[8], al que se puede jugar tanto en entornos exteriores como en interiores. En el artículo anterior presentan una arquitectura barata, basada en el ya comentado Tinmith, con seis grados de libertad[9], bastante precisa, que hace el seguimiento mediante GPS y

[8] Videojuego de disparos en primera persona publicado en 1996, que introdujo algunos de los mayoures avances en el género de los videojuegos en 3D, como la utilización de modelos tridimensionales para los jugadores y los monstruos en vez de imágenes bidimensionales. Así como el diseñar en tres dimensiones el munde donde el juego tiene lugar (Quake, 2005)

[9] Se suele expresar como *6DOF*, acrónimo del inglés “Six Degrees of Freedom”, y referido al

brújula digital cuando el usuario está ubicado en un entorno exterior alejados de edificios. Cuando el usuario está en un entorno exterior, pero cerca de edificios, hace uso del GPS y marcas (prevaleciendo el valor de la marca siempre que esté disponible). Por último, cuando el usuario está en el interior de edificios se hace uso exclusivo de marcas, colocadas en pilares y suelos, de forma que siempre tenga a la vista al menos una marca. En cuanto al aspecto colaborativo, disponen de interfaces de comunicación por voz y cursores 3D que permiten a los usuarios indicar posiciones dentro del entorno 3D. Los usuarios pueden colaborar con otros usuarios con el equipo portátil, o con usuarios que jueguen en ordenadores de sobremesa, todo ello inalámbricamente. Como muchos otros prototipos, la implementación de este sistema no era completa en el momento de la escritura del artículo y aún quedan cosas por resolver. Sin embargo, la principal aportación que hacen los creadores de ARQuake es la combinación de periféricos tipo GPS/brújula con el seguimiento de marcas para obtener una herramienta válida tanto para entornos interiores como para exteriores, a un precio relativamente bajo. Para tratar las marcas hacen uso de la librería ARToolKit. Dicha librería fue desarrollada por Hirokazu Kato en 1999 (Kato and Billinghurst, 1999) y publicada por el HIT Lab de la Universidad de Washington (Kato, 2011). La aparición de ARToolKit orientó la expansión de las aplicaciones CAR hacia el seguimiento óptico usando cámaras de bajo coste. Como se comentó en el Capítulo 2, una de las grandes características de ARToolKit fue la de su lanzamiento basado en licencia de código abierto, puesto que desde ese momento los desarrolladores que se decidían por las tecnologías basadas en AR tenían a su disposición una librería con buenos resultados y gratuita, hecho que impulsó el crecimiento de este área de conocimiento.

Al mismo tiempo que se desarrollaban estos primeros sistemas móviles, Schmalstieg (Szalavári et al., 1998) y Billinghurst (Billinghurst et al., 1998) exploraban los inicios de los interfaces AR Colaborativos (CAR).

Billinghurst, con su trabajo del *Espacio Abierto*, mostró cómo se puede hacer uso de las tecnologías de AR para mejorar la colaboración de tipo cara a cara (Billinghurst et al., 2000). Su sistema utiliza unas gafas transparentes ópticas,

movimiento en un espacio tridimensional, es decir, la capacidad de moverse hacia delante/atrás, arriba/abajo, izquierda/derecha (traslación en tres ejes perpendiculares), combinados con la rotación sobre tres ejes perpendiculares.

conectadas a un ordenador, que hace uso del algoritmo propuesto por Hirokazu Kato para reconocer marcas (comentado anteriormente). Lo más novedoso de este sistema es la forma de interactuar de los usuarios, ya que si éstos quieren mover un objeto virtual, lo único que tienen que hacer es mover su marca física (las marcas están impresas en tarjetas). Así como la asignación de distintos significados al hecho de tener más de una marca junta. También se saca partido de la libertad ofrecida por las aplicaciones AR, que permiten visualizar objetos virtuales mientras se expone el mundo real, por lo que consigue mostrar las expresiones y movimientos de los otros usuarios, a la vez que ofrecen comunicación entre usuarios. Con esta misma filosofía presentó su trabajo "Conferencia AR" donde muestra cómo crear con tecnologías basadas en AR la sensación de que un colaborador remoto está presente en el espacio local, proporcionando mayor realismo que las videoconferencias tradicionales (Billinghurst and Kato, 1999).

Por su parte, la arquitectura Studierstube de Schmalstieg (Szalavári et al., 1998) está pensada para facilitar el desarrollo de aplicaciones AR colaborativas distribuidas. Para reducir la complejidad del problema, limitan el sistema a una única habitación en la que, de momento, sólo pueden interactuar dos usuarios. Dichos usuarios van equipados con un casco de Realidad Virtual con visión 3D y seguimiento magnético, así como un PIP (Panel de Interacción Personal), que es un dispositivo con pantalla y puntero que se sujeta con las dos manos. El PIP también proporciona seguimiento de posición y orientación, y es el principal elemento de interacción con el sistema. Se utiliza una arquitectura cliente-servidor donde se dispone de un servidor para el entorno en el que se almacena una base de datos que incluye registros, objetos móviles y datos de la aplicación y de visionado; además se provee de un servidor de seguimiento para llevar un control de todos los marcadores de la habitación. Como la calidad del seguimiento es crucial para garantizar la calidad de la experiencia, este servidor se ejecuta en un ordenador diferente. Por su parte, el renderizado se realiza en estaciones de trabajo con librerías Open Inventor (Strauss and Carey, 1992). Todo el intercambio de información se realiza a través de Ethernet, usando protocolos TCP/IP y tecnología multicast. Aunque es un sistema con buena acogida por parte del público, incluso de aquel que no tiene conocimientos técnicos sobre las tecnologías que rodean a la AR, está muy restringido. Por una parte, al exigir que el sistema se monte en una única habitación y, por otra parte, a que sólo haya dos usuario colaborando entre sí. A esto hay que añadir que el sistema de seguimiento se

podría mejorar con una versión híbrida, como la que se mostró en ARQuake.

Usando Studierstube, Reitmayr (Reitmayr and Schmalstieg, 2001a) estudió cómo compartir objetos 3D con una interfaz CAR móvil (cargada en la espalda). Este sistema fue el primero en permitir que múltiples usuarios equipados con sistemas AR portátiles colaboraran juntos de manera espontánea, tanto si estaban cara a cara, como si estaban en distintas ubicaciones. Propuso ideas interesantes como que el número de participantes puede cambiar en tiempo de ejecución y la aplicación sigue funcionando, mandándole a los nuevos usuarios la aplicación que se está ejecutando y su estado actual. Del mismo modo, da a los usuarios la opción de almacenar el estado actual de la aplicación cuando abandonan el espacio compartido. Propone también la idea de tener versiones de la misma aplicación, con diferentes *calidades*, de forma que se instale en cada dispositivo la más adecuada a sus prestaciones. El prototipo que proponen está formado con hardware que se puede encontrar en el mercado, con lo que se evitan el tiempo requerido para desarrollar el suyo propio. Esto aporta una gran facilidad para el cambio de configuración del sistema pero, por contra, no les permite tener un equipo lo más pequeño y ligero posible. Para poder manejar el seguimiento y los datos de entrada multimodal, desarrollaron una arquitectura de software nueva, que recibe el nombre de “OpenTracker” (Reitmayr and Schmalstieg, 2001b), basada en el metalenguaje de etiquetado XML, que acababa de aparecer. La idea principal de esta arquitectura es segmentar la manipulación de datos en pasos individuales y construir una red de flujo de datos de las transformaciones. Como permite la integración de distintos tipos de seguimiento y plataformas con diferentes prestaciones, se echa de menos la implementación de algún tipo de negociación para elegir la que mejor se adapte a todos los usuarios.

En 2003 se consiguió desarrollar la primera aplicación AR autocontenida para PDA, es decir, que incluía todo lo necesario para el funcionamiento de la aplicación dentro del propio dispositivo. Este logro vino de la mano de Daniel Wagner, investigador de la Universidad de Tecnología de Viena, que portó ARToolKit para que funcionara en ordenadores de bolsillo (Wagner and Schmalstieg, 2003). Este proyecto propone dos alternativas, una en la que todo se hace desde la PDA y otra en la que el seguimiento de la marca se hace en un ordenador de sobremesa. También permite cambiar dinámica y transparentemente de una alternativa a la otra. En cuanto al rendimiento obtenido, se vio que la detección

de marcas en la PDA era casi diez veces más lenta que la detección en el ordenador, y que leer los datos de la cámara ocupaba más del 40 % del tiempo. Un dato sorprendente fue que, aunque la resolución de la cámara usada en la PDA era muy baja (320x240 píxeles), la precisión obtenida era muy buena, de forma que se consideró su utilización como casco transparente óptico *monocular*, con alguna especie de soporte que permitiera mantener la PDA a la altura de la cara. Aunque se obtuvieron buenos resultados, el prototipo estaba optimizado solucionando cuellos de botella muy específicos del hardware elegido, por lo que debería generalizarse e incluir la posiblidad de añadir otros tipos de sensores como GPS.

Seguidamente se migró Studierstube, consiguiendo así la primera aplicación CAR autocontenida para dispositivos tipo PDA (Wagner et al., 2005). Con este proyecto pretenden dejar atrás los sistemas anteriores (con portátiles cargados en la espalda), puesto que son caros, delicados y pesados, e impedían la difusión de aplicaciones en el mercado, ya que poca gente se podía permitir comprar estos equipos. Del mismo modo limitaban la escala de colaboración a cuatro o cinco usuarios a la vez. Con este mismo objetivo en mente, el equipo de investigación propone una arquitectura interactiva, independiente de la infraestructura y multiusuario, con la idea de que funcione sobre cualquier PDA del mercado. Para ello, desarrollaron una arquitectura basada en un Studierstube *ligero*, al que eliminaron todas las funcionalidades no esenciales, de forma que se pueda ejecutar sin un gran requerimiento de software. Cabe destacar que con Studierstube se consiguió una plataforma de desarrollo de aplicaciones CAR fácil e intuitiva. También incorporan la librería OpenTracker (comentada anteriormente), para gestionar el seguimiento de activos en la escena. Como las librerías software que proporcionan mecanismos de abstracción hardware desarrolladas para PDA estaban pensadas para juegos 2D, desarrollan su propia librería, a la que llaman “PocketKnife”. Del mismo modo, como no existía ninguna librería gráfica compatible con OpenGL para dispositivos móviles, tuvieron que desarrollar una propia, a la que llamaron “Klimt”, con una API[10] muy similar a OpenGL y OpenGL/ES. Para las comunicaciones utilizan un wrapper[11] de ACE[12](ACE, 2013) que proporciona

[10] Del inglés “Application Programming Interface”, es decir, interfaz de programación de aplicaciones

[11] Del inglés “envoltorio”, consiste en una capa de código simple que traduce el interfaz de una librería ya existente en otra interfaz compatible

[12] Siglas en inglés del Entorno de Comunicación Adaptable

acceso a los recursos del sistema independientemente de la plataforma que se use, de forma que se pueda utilizar con cualquier tipo de PDA.

Uno de los problemas más habituales a la hora de realizar aplicaciones para dispositivos móviles es la depuración de errores, puesto que suele ser una tarea tediosa, incluso cuando se realiza en emuladores. En este caso, todos los componentes de este sistema funcionan tanto para Windows CE (el sistema operativo de la PDA) como para Windows 2000/XP, con lo que los programadores pueden depurar las aplicaciones en un ordenador de sobremesa y, a continuación, lanzarlas en la PDA, lo que facilita considerablemente la tarea. Otro beneficio adicional de esta arquitectura es su lanzamiento bajo una licencia "Open Source", lo que permite a los desarrolladores obtenerla de manera gratuita.

La aplicación ejemplo con la que dieron a conocer esta arquitectura se llamó "The Invisible Train"[13] que consiste en una aplicación colaborativa con la que se controla un tren virtual sobre unas vías reales en miniatura. Evaluaron la aplicación en lugares que iban desde la convención de gráficos por ordenador de Los Ángeles (EEUU) (SIGGRAPH 2004), hasta museos y reuniones para estudiantes de secundaria, con la idea de que el público que la usara tuviese diferentes conocimientos de AR. En total, consiguieron probarla cerca de 5000 usuarios, convirtiéndola en la primera aplicación para móviles que se evaluaba por una cantidad suficiente de usuarios. Las conclusiones de las pruebas fueron las previsibles, como que los usuarios aceptaban mejor el uso de PDA que el sistema anterior (portátil en la espalda más casco de visión a través), o que los usuarios eran capaces de manejar fácilmente el interfaz, independientemente de sus conocimientos en AR. Por lo que obtuvieron unos resultados satisfactorios e impulsaron a su vez el uso de PDAs en las aplicaciones CAR.

La AR sobre telefonía móvil siguió un desarrollo similar que el seguido por las PDAs., a las que acabó eclipsando con el lanzamiento de los "smartphones" (o teléfonos inteligentes). Los primeros móviles no tenían suficiente capacidad de cómputo, por lo que los investigadores volvieron a utilizar la estrategia de los *clientes ligeros*. Por ejemplo, el proyecto AR-Phone (Assad et al., 2003) usaba el sistema de comunicación inalámbrico Bluetooth para enviar las imágenes de la cámara del móvil a un servidor remoto, donde se realizaba el procesamiento y

[13]Cuya traducción del inglés es el tren invisible

se preparaba la capa de dibujado, necesitando varios segundos por imagen. Se enviaban imágenes en vez de capturarlas del flujo de vídeo de la cámara porque aún no se contemplaba esa posibilidad, a nivel de sistema operativo, con el consiguiente retraso operativo. Si bien, se intentaba paliar este hecho umbralizando la imagen antes de enviarla, con lo que se conseguía transmitir menos datos. No obstante, la aplicación resultante en AR-Phone mandaba la información en mapas de bits sin comprimir, por lo que aún podían haber optimizado más el envío. Por otro lado, se pretendía que la interfaz para el móvil fuese muy simple, con la idea de que se pudiese ejecutar en móviles de diferentes gamas y lo que consiguieron es una interfaz tan simple que el móvil realiza sencillas tareas de visor, por lo que la interacción es prácticamente nula. Al utilizar Bluetooth el alcance del sistema es muy limitado y lento, por lo que deberían utilizar otras conexiones inalámbricas de más ancho de banda, como la ofrecida por Wifi. Pese a todas las carencias y errores de diseño, se comenta este sistema porque fue el primero en considerar el teléfono móvil como un dispositivo válido para aplicaciones CAR.

Pronto surgieron soluciones más prometedoras, como la propuesta de Henrysson, que portó ARToolKit para que funcionase sobre un teléfono con Symbian[14] (Henrysson and Ollila, 2004), o la de Moehring, que desarrolló su propia librería para el procesado de imágenes y seguimiento (Mohring et al., 2004). Estos trabajos permitieron ejecutar aplicaciones AR simples sobre móviles con una productividad cercana a los 7 FPS. Cabe comentar que los móviles utilizados para estas pruebas ya tenían contemplada la posibilidad de obtener la imagen del flujo de vídeo de la cámara, con lo que se consigue un ahorro significativo en la obtención de los datos a analizar, y que tenían la potencia de cálculo suficiente para empezar a dibujar en 3D.

Tras la llegada del teléfono móvil de altas prestaciones a las aplicaciones CAR, parte de la investigación se centró en adaptar librerías ya existentes. Así ocurrió con ARToolKit, que se optimizó y extendió para su uso sobre telefonía móvil. A esta mejora se le conoce como ARToolKit Plus (Wagner and Schmalstieg, 2007) y está accesible desde 2007. En ella Wagner y Schmalstieg suprimieron, por ejemplo, las operaciones en punto flotante, puesto que los primeros móviles carecían de unidades en coma flotante. También se añadió un detector automático de umbral de oscuridad que mejora sustancialmente las imágenes

[14] Sistema operativo propietario de los teléfonos Nokia

a analizar. Tras acabar la librería los autores consideraban que ARToolKit Plus había alcanzado un nivel sobre el que poco más se podía optimizar y cambiaron el enfoque, centrándose en aspectos que pudieran mejorar la calidad del rastreo, como la detección de bordes (considerada como la mayor debilidad de ARToolKit y ARToolKit Plus). También empezaban a pensar en la posibilidad de detecciones híbridas, con marcas artificiales y naturales.

Ese mismo año, Wagner y Schmalstieg presentaron Studierstube Tracker (Schmalstieg and Wagner, 2007), una nueva libería, creada desde cero, pensada exclusivamente para móviles. Dicha librería soporta seis tipos diferentes de marcas, aporta dos estimadores de posición y propone tres algoritmos de umbralización, de forma que se utilice uno u otro en función de cada aplicación, para obtener el mejor rendimiento posible. Los requerimientos de memoria son un orden de magnitud más bajos que los necesarios en ARToolKit Plus y están en torno a los 150 KBytes. No obstante, pese a mejorar las prestaciones de sus predecesoras, tiene un problema, y es que ya no es código abierto ni gratuita, por lo que su difusión es más lenta.

Al año siguiente de la publicación de esta librería mejorada, estos mismos autores, junto con T. Langlotz presentaron una serie de mejoras para la librería Studierstube Tracker (Wagner et al., 2008a), entre las que se incluían tres tipos nuevos de marcas, menos molestas desde el punto de vista visual y un par de algoritmos que permiten reaccionar a posibles pérdidas de marcas u oclusiones. Uno de los algoritmos propuestos utiliza seguimiento de características naturales en la sección de imagen capturada alrededor de la marca artificial, de forma que cuando se deja de percibir algún borde de la marca, se puede seguir estimando la posición de la misma. Para poder realizar el seguimiento de las características del contorno de la marca, se van extrayendo datos de la misma mientras se detecte con normalidad, y se almacenan en una base de datos para cuando llegue el momento de necesitarlas. El otro algoritmo utiliza el flujo de vídeo proporcionado por la cámara como sensor de inercia, de forma que se puede hacer uso del mismo aunque el teléfono móvil no disponga de dicho sensor. Aquí se utilizan dos aproximaciones, una primera versión del algoritmo, menos precisa pero con un consumo de recursos muy ligero, que se utiliza mientras no haya problemas y la segunda versión del algoritmo, mucho más robusta, que se usa sólo cuando falla la primera. Ambos algoritmos ofrecen buenos resultados mientras el entorno sea

plano o casi plano, por lo que sólo se sacará provecho de ellos en aplicaciones con marcas sobre superficies como mesas o paredes.

Ese mismo año los autores de ARToolKit Plus, en colaboración con investigadores de la Universidad de Cambridge, presentaron dos nuevas técnicas de seguimiento sin marcas en tiempo real para móviles (Wagner et al., 2008b). Fue la primera vez que se hablaba de seguimiento sin marcas con seis grados de libertad en teléfonos móviles. Esto se debe a que la variante del seguimiento sin marcas es mucho más costosa computacionalmente hablando, puesto que se tienen que extraer vectores de características de la imagen y compararlas después con una base de datos de objetos previamente almacenados. Las técnicas presentadas son adaptaciones de dos de los algoritmos más conocidos de descripción de características, SIFT[15] (Lowe, 2004), conocido por ser un descriptor de características robusto, pero computacionalmente caro, y FERNS (Ozuysal et al., 2007), que transforma la detección en una clasificación rápida, pero requiere una gran cantidad de memoria. Por estos motivos, estos algoritmos no son integrados en teléfonos móviles, pues tienen poca memoria y poca potencia de CPU. En (Wagner et al., 2008b) se describe en detalle las modificaciones que realizan a ambas tecnologías, con las que consiguen obtener una productividad de 15 FPS. A la hora de realizar las pruebas se encontraron con un problema que no han sabido explicar, relacionado con que el algoritmo de Fern se implementó inicialmente para Symbian. Al evaluarlo en dispositivos con sistema operativo Windows Mobile, el rendimiento es bastante peor (el doble de lento) que en dispositivos móviles provistos con Symbian. La mayor limitación de ambos algoritmos es que no permiten que la inclinación de la imagen sea superior a 50 grados, mientras que con las marcas se seguía funcionando con ángulos cercanos a los 90 grados. Pese a todo, consiguen ejecutar ambos algoritmos en tiempo interactivo, como se pretendía inicialmente.

A partir de 2008 ha habido otros artículos que analizan la viabilidad de las tecnologías de AR sin marcas sobre teléfonía móvil, como es el caso de (Srinivasan et al., 2009), donde se presenta un análisis de la sobrecarga que imponen estas aplicaciones en los móviles y de los posibles cuellos de botella en su diseño. Los autores también presentan una caracterización detallada de esos cuellos

[15] Siglas procedentes del inglés “Scale Invariant Feature Tracking”, es decir, seguimiento de características invariante a la escala

de botella e implementan diversas optimizaciones, analizando sus beneficios. No obstante, se observa que este análisis está realizado desde el punto de vista de investigadores de la compañía Intel que no están acostumbrados a trabajar con Realidad Aumentada sobre teléfonos móviles, ya que centran su análisis de cuellos de botella únicamente en la detección y emparejamiento de características, aludiendo a que la captura de imagen y el dibujado de la información en la pantalla son despreciables en tiempo de ejecución, según su *análisis inicial*. En el capítulo de caracterización de los teléfonos móviles (Capítulo 4) se ha demostrado que esto no es cierto, ya que la captura de la imágen no sólo no es despreciable, sino que representa un porcentaje considerable del ciclo de aplicación. El problema anterior queda patente a lo largo del artículo, como en las imágenes que insertan de ejemplo de sus pruebas sobre móviles, donde se puede apreciar montajes fotográficos. Por lo tanto, esta investigación se podría considerar una optimización de algoritmos de caracterización de descriptores más que un análisis de prestaciones de AR sobre telefonía móvil.

Como venían haciendo año tras año, D. Wagner y D. Schmalstieg, en este caso junto a H. Bischof, en 2009 presentaron una nueva investigación en la que demostraban que es posible realizar seguimiento de varios objetivos (sin marcas) simultáneamente sobre teléfonos móviles en tiempo real (Wagner et al., 2009). El sistema que presentan consigue seguir múltiples objetivos conocidos además de detectar nuevos elementos a rastrear. También presentan un método para balancear automática y dinámicamente la calidad de la detección y el seguimiento de forma que pueden asegurar un rendimiento constante. Realizan pruebas de este sistema sobre móviles reales y consiguen seguir a seis objetos planos simultáneamente, mientras proporcionan AR con seis grados de libertad, con una productividad de 23 FPS.

Normalmente, las investigaciones se centran en aspectos software de los sistemas CAR, pero en 2009, un grupo de investigadores de Intel, compuesto por la mayoría de los participantes del primer artículo -(Srinivasan et al., 2009)-, propuso un par de aceleradores hardware que pretenden mejorar algunas limitaciones de las plataformas x86 para móviles (Lee et al., 2009). En este trabajo presentan el diseño detallado de un reconocedor de objetos (MAR-HA) y otro de procesamiento de coincidencias (MAR-MA). También cuantifican el rendimiento y área de eficiencia de los aceleradores. Con estas dos propuestas consiguen que los

cuellos de botella detectados funcionen 20 veces más rápido, produciendo un tiempo de respuesta 7 veces mejor. No obstante, se observa el mismo problema comentado anteriormente, donde el beneficio que se obtiene con estos aceleradores es *virtual* puesto que, de momento, no se han implementado en teléfonos móviles reales.

En la línea de los investigaciones de CAR sin marcas, investigadores del laboratorio GIST U-VR (Korea) propusieron en 2010 su propio sistema AR sobre móviles en el que la intervención del usuario es mínima (Lee et al., 2010b). En este sistema al usuario le basta con apuntar con la cámara del móvil a la zona que quiere aumentar y desde entonces cada vez que la cámara enfoque ese lugar lo detectará y aumentará. Este sistema se basa en otro trabajo reciente centrado en el reconocimiento de parches en perspectiva (Hinterstoisser et al., 2011) sobre ordenadores de sobremesa. Este trabajo es destacable ya que funciona bien incluso sobre superficies con pocas texturas. En este trabajo muestran cómo modificar ese sistema para obtener mejores prestaciones de CAR sobre móviles. En particular, sustituyen el paso de detección de puntos de características por parches largos para mejorar la robustez. También reemplazan la forma en que se computan las plantillas, puesto que consumen una gran cantidad de memoria, mediante operaciones de dibujado y emborronamiento, que es mucho más eficaz sobre móviles y necesita sólo unos segundos. Y, por último, explotan el uso de los acelerómetros del móvil para relajar la captura de imágenes en el seguimiento de la escena. Adicionalmente, este trabajo permite compartir la información sobre los puntos que se aumentan de forma inalámbrica, construyendo así un espacio AR compartido.

Por su parte, ese mismo año, Wagner y Schmalstieg, junto con A. Mulloni presentan el estado de su investigación de AR sin marcas en tiempo real sobre móviles (Wagner et al., 2010). En esta investigación proponen una nueva técnica de seguimiento sin marcas de seis grados de libertad con la que consiguen frecuencias de más de 30 Hz sobre móviles disponibles en el mercado. Esta técnica, que recibe el nombre de "PatchTracker"[16], se basa un modelo de movimiento, en el que se estima qué buscar, dónde buscarlo y qué transformaciones afines se puede esperar. En contraste con SIFT y FERNS, no intenta ser invariante a cambios locales, sino que los busca activamente. Esta aproximación es más eficaz

[16] Traducido como *Seguidor de Parches*

que la detección por seguimiento porque usa el hecho de que la escena y la posición de la cámara no cambian significativamente de una imagen a la siguiente, por lo que la posición de las características se puede predecir con facilidad. Por lo tanto, utiliza una imagen de referencia como único dato de entrada. Esta imagen es almacenada con diferentes tamaños para evitar efectos de "aliasing"[17] que se sufren durante los cambios de escala a medida que el usuario se mueve por el entorno real. Con esta nueva técnica y las dos que propusieron en el artículo anterior (versiones para móvil de SIFT y Fern) realizan un híbrido que funciona cinco veces mejor que las versiones presentadas hasta el momento y, a la vez, solventan la limitación que tenían SIFT y Fern respecto de detecciones con la imagen girada más de 50 grados.

A finales del 2011, se realiza el lanzamiento de una nueva librería llamada Vuforia, por parte de la compañía Qualcomm (comentada en detalle en el Capítulo 2). Esta librería es la continuación lógica del trabajo realizado por Wagner y Schmalstieg sobre AR sin marcas. El único documento científico que se ha encontrado con información sobre esta librería es un trabajo de fin de máster desarrollado por Josep Paredes Figueras (Paredes Figueras, 2013), que concluyó en Marzo del 2013 en la Universidad Politécnica de Cataluña. En él se analizan las características y funcionalidades del SDK de Vuforia, así como su arquitectura y elementos. Después se habla de los posibles campos de aplicación de esta librería y de los pasos típicos para utilizarla en aplicaciones AR, junto con la herramienta Unity 3D. Como se ha comentado anteriormente, esta librería se puede descargar gratuitamente, pero no es código abierto, por lo que no se puede analizar su código fuente. Pese a que no se ha encontrado otro documento científico sobre ella, sí que se sabe que uno de los encargados del equipo de desarrollo que hay detrás de ella es Daniel Wagner que, como se ha mostrado a lo largo del presente capítulo, es uno de los grandes representantes de la AR, cuyas investigaciones junto a Dieter Schmalstieg han guiado el desarrollo de la AR a lo largo de más de 10 años.

Hasta el momento se han mostrado librerías desarrolladas para programadores, por lo que los usuarios sin conocimientos de programación no les pueden sacar mucho partido. Partiendo de esta premisa, se han desarrollado otras libre-

[17] El aliasing es el artefacto gráfico característico que hace que en una pantalla ciertas curvas y líneas inclinadas presenten un efecto visual tipo *sierra* o *escalón*

rías para usuarios sin conocimientos en programación, como diseñadores gráficos o expertos en multimedia, entre las que se pueden destacar BuildAR, DART, AMIRE y MARS, que se comentan a continuación.

BuildAR (BuildAR, 2009) permite a los usuarios asociar modelos 3D con marcas a las que se les efectúa seguimiento visual. Es una librería muy básica, con un conjunto de funcionalidades algo escaso. De hecho, sólo permite escalar, trasladar y posicionar objetos *sobre* las marcas y obtener una vista AR en vivo. Sin embargo, no dispone de soporte para interacción entre objetos o comportamientos complejos.

Una de las primeras librerías orientada a usuarios sin conocimientos a nivel de programación que proporcionaba interacción es DART[18] (MacIntyre et al., 2004), desarrollada sobre la aplicación Macromedia Director[19]. DART permite a este tipo de usuarios crear experiencias AR usando los servicio AR de bajo nivel proporcionados por el Director Xtras (Wang et al., 2009), e integrarlas con los comportamientos y conceptos ya existentes en el Macromedia Director. Esta librería soporta tanto programación visual como escrita, permitiendo a los diseñadores especificar relaciones complejas entre el mundo físico y el virtual, y permite, entre otras cosas, manejar actores 3D animados así como el seguimiento de marcas en los vídeos en tiempo real (usando para ello ARToolKit).

Por su parte, AMIRE (Grimm et al., 2002), proporciona un interfaz para cargar y reemplazar una librería en tiempo de ejecución y usa técnicas de programación visual para desarrollar aplicaciones AR interactivas. Está diseñada para permiter a los expertos en contenido construir fácilmente aplicaciones sin conocer los detalles que subyacen en las tecnologías base que, de nuevo, utilizan ARToolKit para la parte de reconocimiento de marcas.

Todas estas librerías se crearon para el desarrollo de aplicaciones orientadas a computadores de sobremesa, pero existen algunas que también se han diseñado para aplicaciones AR *móviles*, aunque no sean específicas para telefonía

[18] El nombre proviene de sus siglas en inglés: “Designer’s AR Toolkit”, es decir, la herramienta de realidad aumentada de los diseñadores

[19] Un entorno de desarrollo multimedia desarrollado por la empresa Macromedia y distribuido más tarde por Adobe Systems Incorporated. Fue precursor del actual Adobe Flash (Macromedia Director, 2014)

móvil. Un ejemplo de estas librerías es MARS[20] (Sinem and Feiner, 2003). Esta librería usa un interfaz gráfico 3D de usuario que permite crear aplicaciones AR móviles para entornos de exterior usando sistemas AR de los que se cargan en la espalda. Permite previsualizar los resultados en un ordenador de sobremesa, en un sistema de realidad virtual o uno de AR.

Aunque la librería de tipo "markerless" (del inglés "sin marcas") Vuforia ha marcado un hito importante, en el transcurso de esta tesis el campo de la AR para móviles ha seguido avanzando en cuanto a investigaciones. Ejemplo de ello es la investigación realizada en (Hofmann et al., 2012) donde se hace uso de la GPU de los teléfonos móviles actuales. El disponer de GPU habilita la utilización del descriptor de características SURF[21] que, hasta el momento, se había considerado imposible de implementar sobre móviles, por su elevada carga computacional. Esta adaptación de SURF recibe el nombre de "uSURF-ES", está escrita en C++ y utiliza OpenGL ES 2.0. Los cambios más notables frente a la versión original son la utilización de "mipmaps"[22] para conseguir conocimiento de la escala, muestreo con precisión de sub-píxel con *wavelets de Haar*[23] y la codificación en punto-fijo de ajuste dinámico a la precisión en punto-flotante de la GPU. Pese a la mejora obtenida, es sólo una primera aproximación al potencial de explotar la GPU que se podría optimizar. Por ejemplo, reformando las texturas que almacenan los descriptores sin normalizar para que almacenasen cada descriptor en un texel[24] cuadrático en vez de en columnas, así se aprovecharía la localidad espacial de las texturas y se incrementaría la utilización de la caché de texturas.

Otras investigaciones se han centrado en mejorar la interacción con los móviles haciendo uso de las manos, como es el caso de (Chun and Höllerer, 2013)

[20] De sus siglas en inglés "Mobile Augmented Reality Systems", es decir, sistemas de realidad aumentada móvil

[21] Siglas en inglés del algoritmo "Speeded-Up Robust Features", es decir, descriptor robusto de características acelerado(Bay et al., 2008)

[22] Los mapas MIP (comúnmente llamados mipmaps), son colecciones de imágenes de mapas de bits que acompañan a una textura principal para aumentar la velocidad de dibujado

[23] El wavelet de Haar es una cierta secuencia de funciones, propuesta en 1909 por Alfred Haar, para ofrecer un ejemplo de un sistema ortonormal contable para el espacio de las funciones de cuadrado integrable en la recta real(Wavelet Haar, 2013)

[24] Un texel (contracción del inglés "texture element", o también "texture pixel") es la unidad mínima de una textura aplicada a una superficie

y (Sodhi et al., 2013). En el primero de estos dos artículos, Chun et al. proponen una aproximación para interactuar con el contenido virtual de entornos AR usando los gestos de la mano, solucionando el problema del movimiento de la cámara del móvil durante la realización del gesto con mapeado de texturas proyectivas y eliminación del fondo de pantalla, consiguiendo resultados aceptables para escenas simples mientras mantienen la productividad. Tras la implementación realizaron una evaluación del sistema con pruebas con usuarios y los resultados no fueron muy satisfactorios. Por una parte, la idea es nueva y los usuarios están muy acostumbrados a los gestos sobre la pantalla (como toques y deslizamientos) por lo que los movimientos de la mano no resultaron tan intuitivos como se pensaba. Por otro lado, la precisión alcanzada aún no es la óptima y los usuarios percibían poca respuesta del sistema a movimientos delicados. Por lo tanto, es interesante como una nueva aproximación a la colaboración en aplicaciones AR y, aunque se queda un poco limitada, abre la puerta a futuras investigaciones en un campo con muchas posibilidades. Un ejemplo se puede ver en la Figura 3.3, donde aparecen cuatro imágenes en las que se aprecian los diferentes tipos de interacción que se puede tener con este sistema. En el segundo artículo -(Sodhi et al., 2013)- se presenta *BeThere*, un sistema diseñado para explorar los gestos 3D y entradas espaciales que permite a los usuarios remotos interaccionar con los usuarios locales como si estuviesen en el mismo espacio físico. Proporciona pues, ayudas visuales en 3D para tareas colaborativas inherentemente en 3D. No obstante, para conseguir la posición y coordenadas de la mano no basta con un teléfono móvil sino que necesitan una serie de sensores añadidos, entre los que se encuentra Kinect[25] y varios sensores táctiles. Por lo tanto, no es un sistema que se pueda utilizar con los teléfonos móviles actuales sin más y, aunque proporciona buenos resultados en los estudios realizados con el prototipo, se podría considerar como una interfaz posible si los teléfonos móviles futuros incorporasen nuevos sensores. En la Figura 3.4 se muestra un ejemplo de uso de este sistema. La primera imagen de la izquierda muestra la interacción de cada usuario: Bob proporcionará el escenario con piezas y Alice producirá un gesto 3D con la mano. En la imagen etiquetada como “a” se muestra lo que el sistema consigue simular, que ambos usuarios estén en el mismo espacio físico. En la imagen “b” se observa al usuario local (Bob) haciendo gestos con la mano. Por último, la imagen “c” muestra que el usuario local observa una segunda mano (virtual) generada por el sistema y que se corresponde con la mano del usuario

[25] Sensor de movimiento desarrollado por Microsoft para la consola Xbox 360(Kinect, 2013)

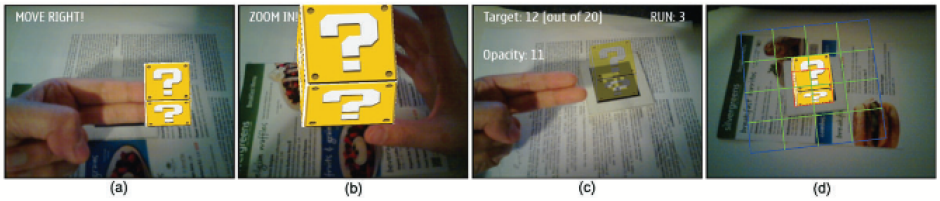

Figura 3.3: Ejemplo de uso del sistema de interacción con mano detallado en (Chun and Höllerer, 2013)

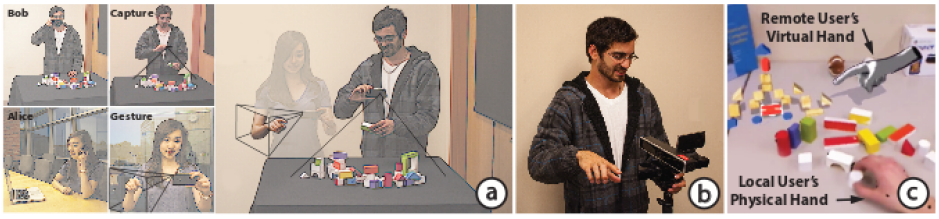

Figura 3.4: Ejemplo de uso del sistema BeThere de interacción por gestos 3D propuesto en (Sodhi et al., 2013)

remoto (Alice).

Como resumen se tiene que, a día de hoy, se pueden ejecutar aplicaciones AR sobre teléfonos móviles en tiempo real, tanto si se utiliza la versión de seguimiento de marcas, como si se usa la de detección de características del mundo real. Estas mismas aplicaciones contemplan la posibilidad de que varios usuarios colaboren para realizar tareas en equipo, aunque esta colaboración, de momento, implica sólo a cuatro o cinco usuarios a la vez. No obstante, la gran difusión del teléfono móvil junto con las considerables prestaciones que poseen los dispositivos actuales, abren la posibilidad de expandir la colaboración de forma que intervengan muchos más usuarios a la vez, lo que puede provocar una carga considerable en los sistemas de AR. Por lo tanto, habría que realizar una caracterización de los sistemas de CAR para ver cómo se comportan conforme se aumenta la cantidad de usuarios que intervienen en la colaboración. De ahí surge la necesidad de la presente investigación, que intenta cubrir ese hueco del estado de la tecnología y servir de guía a las aplicaciones CAR que están por venir, en las que se podría encontrar un uso masivo de la colaboración entre usuario AR.

4

Caracterización de los dispositivos móviles al ejecutar aplicaciones CAR

4.1. Cuota de mercado

Desde el lanzamiento de los primeros sistemas basados en tecnologías de Realidad Aumentada (AR[1]), el potencial de los sistemas de Realidad Aumentada Colaborativa (CAR[2]) se explotó con actividades como la Computación Colaborativa (Billinghurst et al., 2000) o la Teleconferencia (Billinghurst and Kato, 1999). Los dispositivos portátiles se usaron para proporcionar movilidad a los sistemas CAR y permitir que los usuarios equipados con dispositivos AR portátiles pudie-

[1] De sus siglas en inglés “Augmented Reality”

[2] De sus siglas en inglés “Collaborative Augmented Reality”

sen colaborar con los usuarios de ordenadores de sobremesa (Höllerer et al., 1999; Piekarski and Thomas, 2002).

Al juntar la continua mejora de la tecnología del silicio con la evolución en las metodologías de diseño, se consigue integrar computación realmente compleja dentro de único chip (SoCs[3]). Como resultado, existe un gran número de dispositivos con sistemas de computación empotrados presentes en nuestra vida diaria, y prácticamente todos se han utilizado en los sistemas CAR. Uno de estos dispositivos es el teléfono móvil (Henrysson and Ollila, 2004; Mohring et al., 2004). Se ha visto que es el dispositivo idóneo para aplicaciones CAR, puesto que es el dispositivo más pequeño en cumplir con todos los requisitos. Dichos requisitos incluyen una pantalla a color de buena resolución, cámaras integradas de al menos dos megapíxeles, procesadores con hasta ocho núcleos, procesadores gráficos (GPUs), GPS, acelerómetros y un largo listado de periféricos (Hall and Anderson, 2009).

Con la gran expansión de la telefonía móvil, existe una gran variedad de marcas, prestaciones y Sistemas Operativos (S.O.) entre los que elegir. Los S.O. más extendidos para teléfonos móviles son Symbian de Nokia, Android de Google, RIM de Blackberry, iOS de Apple, Windows Mobile de Microsoft y Bada de Samsung (Ahonen, 2010). Este estudio se centrará en dos de ellos, Android e iOS, porque juntos acaparan la práctica totalidad de la cota de mercado (Hall and Anderson, 2009). Pese a limitar a dos los sistemas operativos y pese a que iOS sólo está disponible para móviles de la marca Apple, con Android se tiene una gran variedad de marcas y, dentro de cada una de ellas, móviles de gamas alta, media y baja. Esta variedad puede tener efectos significativos en las prestaciones de los sistemas CAR de gran escala, en términos de latencia y productividad. Por ejemplo, el tiempo de respuesta de cada cliente, multiplicado por el número de clientes, puede tener efectos significativos en el tiempo de respuesta del sistema.

A fecha de este trabajo de investigación (2013), Apple proporciona iPhone 3GS y iPhone 4 como dispositivos de gama media y baja, respectivamente. Por lo tanto, se seleccionarán otros dos móviles Android de gama media y baja. De entre las posibilidades existentes en el mercado nacional de telefonía móvil se

[3]Sistemas-en-Chip, del inglés “Sistems on Chip”

OS	**Android**		**iOS**	
Model	**Milest.**	**Nex. One**	**iPh 3GS**	**iPh 4**
CPU	TI OMAP 3430	Qual. QSD8250	Samsung S5PC100	Apple APL0398
Freq. (MHz)	550	998	412	800
GPU	SGX530	Qual. Adreno200	SGX535	SGX535
RAM (MB)	256	512	128	512
Camera (MP)	5.02	4.92	1.92	4.92

Tabla 4.1: Características Hardware de los móviles considerados

ha escogido Motorola Milestone y HTC Nexus One, que se corresponden con terminales de gama media y alta de Android, respectivamente. El motivo de esta elección es que son teléfonos móviles disponibles en el grupo de investigación, puesto que son los teléfonos personales de algunos de los integrantes. La Tabla 4.1 muestra las principales características de estos móviles, incluyendo CPU, modelo de GPU, memoria RAM de que disponen y resolución de las cámaras. Empezando por el S.O. Android, el terminal Motorola Milestone tiene una frecuencia de 550 MHz, mientras que el terminal Nexus One tiene el doble (998 MHz). El Motorola Milestone ejecuta la versión 2.0 del S.O., mientras que el Nexus One ejecuta la 2.1. Para los dispositivos de Apple, el terminal iPhone 3GS tiene una CPU de 412 MHz, mientras que el iPhone 4 tiene el doble de frecuencia (800 MHz). Ambos dispositivos ejecutan la versión del S.O. 4.1.3. Todos ellos incluyen una cámara de 5 Megapíxeles de resolución, excepto el iPhone 3GS, que tiene equipada una de 2 Megapíxeles. En cuanto a la GPU, El Milestone y los dos dispositivos de Apple incluyen el procesador gráfico PowerVR SGX, mientras que el Nexus One tiene una GPU Qualcomm.

En cuanto a los tipos de pruebas a realizar, las aplicaciones de CAR son una expansión de las aplicaciones de AR, a las que se le añade la etapa de envío, es decir, se inserta un subsistema de comunicaciones para que varios usuarios de AR compartan escenarios 3D aumentados en tiempo real.

El ciclo de trabajo de las aplicaciones CAR puede dividirse en cuatro etapas, como se explicó en el Capítulo 2. La primera etapa recibe el nombre de *Adquisición de imagen*, y consiste en obtener una imagen del flujo de la cámara, es decir, no se captura una imagen cada vez que se necesita, puesto que ese sería un proceso muy lento. En su lugar, se deja la cámara en modo vídeo y cada vez que se necesita una imagen se hace una copia de la información almacenada en el buffer en ese momento. En la segunda etapa se realiza la detección de la marca (ya sea artifical o natural) de la imagen obtenida en la primera etapa. Una vez detectada, se obtiene la posición y orientación relativa respecto de la cámara. Cabe comentar aquí que existe la posibilidad de realizar el seguimiento de varias marcas simultáneamente pero, como la mayoría de las aplicaciones CAR sólo detectan una, la presente investigación se centrará en una única marca (artificial o natural). En la tercera etapa se usa la información obtenida para dibujar los objetos 3D necesarios para *aumentar* la escena y ponerlos así en la posición y orientación necesarios para que se vean bien desde nuestro punto de vista actual. Por último, en la cuarta fase se realiza el intercambio de información con otros nodos a través de algún medio de comunicación que, generalmente es Wifi, 3G o Bluetooth (Schmeil and Broll, 2007). En nuestro caso, se ha decidido que la información a compartir será la posición de la marca obtenida en la segunda etapa, es decir, un vector con las coordenadas X e Y, con dos cifras decimales. Por otro lado, aunque se ha comentado tres posibles canales de comunicación, se descartará el Bluetooth porque su rango de transmisión es muy limitado.

A la hora de realizar la caracterización, nuestro objetivo es analizar el tiempo que necesita cada etapa para ejecutarse, el consumo de CPU que realiza, la cantidad de memoria que requiere, y el retardo obtenido en la transmisión de los datos.

La primera implementación que se verá es la implementación con marcas que, como se dijo anteriormente, es la que más aceptación tuvo al inicio de la AR. Otro de los motivos de su popularidad es que existen librerías de código abierto que facilitan mucho la realización de aplicaciones. El caso más famoso es la librería ARToolKit y su sucesora ARToolKitPlus, como se comentó en el Capítulo 2, que serán las utilizadas en nuestra estudio. Por su parte, la segunda implementación, correspondiente a la AR sin marcas, hará uso de una nueva librería llamada Vuforia, que publicó Qualcomm a finales del 2011. El uso de esta

librería se ha propagado rápidamente, convirtiéndola en un nuevo estándar de facto. Esta librería hace un mejor uso de los procesadores actuales (multinúcleo) y ha salido con soporte para más de 400 teléfonos y tabletas diferentes, y sigue objetos reales con una gran fluidez (Qualcomm, 2012).

4.2. Implementaciones con marcas

Existen marcas de tipos muy diversos, como ARToolKit, ARToolKitPlus, ARTag, ARSTudio, QR-Code, ShotCode, etc. (Fiala, 2005). Un ejemplo de ellas se puede ver en la Figura 4.1. Sin embargo, las más extendidas son las dos primeras, porque el código fuente de sus librerías es abierto, como se mencionó anteriormente. Por ese motivo se ha seleccionado la librería ARToolKitPlus, que es la versión para teléfonos móviles de ARToolKit que, entre otras cosas, elimina el uso del punto flotante aritmético, puesto que los primeros móviles no disponían de unidad de cálculo en coma flotante.

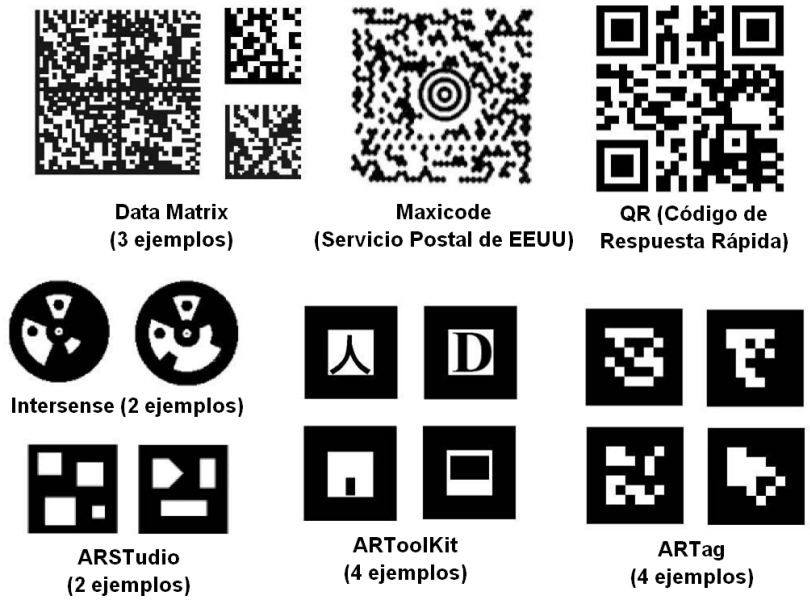

Figura 4.1: Ejemplos de los diferentes tipos de marcas que se utilizan en la etapa de reconocimiento de las aplicaciones de AR

A la hora de realizar la implementación se realizó una investigación para ver si existían desarrollos (comerciales o libres) sobre plataformas móviles que utilizasen esta librería. Se encontraron implementaciones que hacían uso de ella para aplicaciones AR, por lo que se utilizaron como punto de partida. Concretamente, para la implementación de Android se usará el código proporcionado por la librería NyARToolKit (nyartoolkit, 2011).

NyARToolKit es una versión de ARToolKitPlus portada a Java; es una librería de funciones orientada a la interpretación visual e integración de datos Virtual con el mundo real. Se usará en su versión 2.5.2. A modo de resumen, se puede decir que la librería incluye captura de imágenes en tiempo real, dibujado de objetos 3D, uso de Open GL e integra ambas en el flujo de salida. Tras analizar el código, se ha delimitado cada una de las tres primeras etapas vistas en la Figura 2.21 que se corresponden con las etapas de la AR clásica. Tras identificar dichas etapas en el código, se les ha puesto marcas de tiempo, para saber lo que tardan en ejecutarse. Acto seguido se le ha añadido la etapa de envío, creando un socket TCP que envía la información (la posición de la marca comentada anteriormente) a un servidor “echoServer” realizado expresamente. Este servidor se limita a escuchar por un puerto constantemente y contestar mensajes de reconocimiento. El cliente completa su etapa de envío tras recibir este mensaje del servidor. El código completo de esta implementación figura en el Apéndice G.

Android proporciona diferentes resoluciones para elegir, a la hora de tomar las capturas de imágenes con la cámara. Pero como nuestra aplicación intenta ser lo más rápida posible, se seleccionará la resolución más pequeña posible, la de peor calidad, para que la búsqueda de la marca sea lo más rápida posible. Concretamente, se utilizará una resolución de 320 x 240 píxeles para ambos teléfonos móviles, el Nexus One y el Motorola Milestone. Tras obtener la posición y orientación de la marca, se dibujará un objeto simple en 3D en la pantalla, como se puede ver en la Figura 4.2 en la que se superpone una tetera a la marca encontrada.

Para realizar el proyecto con el S.O. Android se ha hecho uso de la plataforma de desarrollo *Eclipse* en su versión v21.0.0519525 junto con el “Android Developer Tools” (ADT), que incluye el SDK de Android (Google, 2011) desde la versión 1.5 a la 4.2 de dicho SDK. Se pueden ver las modificaciones del código de la

Figura 4.2: Captura de pantalla del dibujado de implementación Android

implementación de Android en el Apéndice A. Y, como es un estudio sobre la implementación con marcas, se ha utilizado la marca que aparece en la imagen de la izquierda de la Figura 4.3.

En cuanto a la versión de iOS, se ha utilizado la implementación proporcionada por Benjamin Loulier (Loulier, 2011). Para la etapa de detección de la marca, se ha hecho un *wrapper* en objetive-c de la librería ARToolKitPlus en su versión 2.1.1. Para la carga de los objetos 3D se usan ficheros XML y archivos ".h". Se probó con archivos ".obj"(más usuales en dibujado 3D), pero el parseado era

Figura 4.3: Marcas utilizadas en ambas implementaciones: NyARToolkit para Android (izda.) y ARToolKit Plus para iOS (dcha.)

muy lento, por lo que sólo soporta el dibujado de una textura. Se dibuja usando OpenGL ES. En cuanto al procedimiento seguido, ha sido análogo al de la versión Android, es decir, se ha analizado el código y delimitado con marcas de tiempo las tres etapas que se corresponden con la aplicación AR. A continuación se ha añadido la etapa de envío, creando un socket TCP que envía al EchoServer la posición obtenida en la segunda etapa, convirtiéndo así la aplicación en CAR.

Al contrario que Android, iOS proporciona sólo dos resoluciones: *completa* y *media*. En la resolución media se obtiene la misma resolución que en la completa, pero sólo se analizan uno de cada dos píxeles. Con el fin de efectuar la comparación lo más justa posible, se ha elegido la resolución *media*, lo que proporciona imágenes de 400 x 304 píxeles para el iPhone 3GS e imágenes de 1280 x 720 píxeles para el iPhone 4.

Por su parte, para la implementación de iOS se utiliza la plataforma de desarrollo *Xcode* en su versión 4.6.1 para versiones de iOS desde la 4.3 a la 5.1. Las modificaciones del código de la implementación de iOS se pueden encontrar en el Apéndice B. Al tratarse de una implementación con marcas, se ha hecho uso de la marca que aparece en la imagen derecha de la Figura 4.3.

En cuanto a la toma de datos de las pruebas, se ha comentado que cada etapa queda delimitada por dos marcas de tiempo. Dichas marcas de tiempo quedan impresas en el archivo de registro[4] de cada ejecución, por lo que tras realizar las pruebas sólo queda analizar los ficheros de log para realizar la evaluación de prestaciones. Un ejemplo de la traza de ejecución de una prueba se puede ver en el Apéndice C. Para tratar los ficheros de “log” se ha hecho una aplicación en python que se encarga de obtener los valores con los que se realizará el análisis. Dicho fichero también se puede encontrar en el Apéndice D.

[4]Conocido usualmente por su nombre inglés: “log”

4.3. Evaluación de prestaciones de la implementación con marcas

Esta sección muestra la evaluación de prestaciones de los teléfonos móviles de Android e iOS al usarlos en aplicaciones CAR con detección de marcas. Se ha medido la latencia y el número de imágenes por segundo[5] que el sistema CAR puede proporcionar en función del modelo de teléfono que se utilice. La latencia, medida en milisegundos (ms), se entiende aquí como el tiempo necesario para enviar cada nueva posición al servidor. Sin embargo, en sistemas distribuídos, no se puede calcular de forma precisa la latencia de los datos intercambiados entre dispositivos, debido a los posibles desfases entre los relojes de emisor y receptor. En estos casos lo que se utiliza es el tiempo de ida y vuelta o RTT[6], puesto que permite que el tiempo de ida y el de vuelta se midan por el mismo reloj. La Tabla 4.2 muestra estos tiempos junto con el tiempo total para completar un ciclo y su valor inverso (FPS).

Etapas (ms)	Cámara	Detección	Dibujado	Envío	Total	FPS
Milestone	248,64	288,53	30,42	14,14	698,34	1,43
Nexus One	40,25	78,08	13,23	5,54	167,11	5,98
iPhone 3GS	33,29	58,07	28,26	15,42	398,00	2,51
iPhone 4	17,66	182,17	23,34	7,06	523,26	1,91

Tabla 4.2: Tiempos de ejecución (ms) de cada etapa CAR para cada uno de los móviles considerados

En la Tabla 4.2 las primeras cuatro columnas muestran la duración media de cada una de las etapas de los sistemas CAR para cada móvil empleado, medidas en milisegundos. Por su parte, la quinta columna muestra el tiempo necesario para concluir un ciclo. Este valor se obtiene calculando el tiempo transcurrido entre dos marcas de tiempo cualesquiera, es decir, se mira cuándo se inicia la primera etapa en un ciclo y cuándo se inicia esa etapa de nuevo en el ciclo siguiente. De la resta de ambos valores sale el tiempo de ciclo. Los FPS que se muestran en la columna de la derecha se obtienen convirtiendo el tiempo total

[5] Conocido usualmente como FPS, de sus siglas en inglés: “Frames Per Second”

[6] De sus siglas en inglés: “Round-Trip-Time”

de milisegundos a segundos e invirtiendo el valor resultante.

El objetivo de esta caracterización es encontrar la etapa que más tiempo consume y ver cómo se comportan los teléfonos móviles en cada etapa, así como saber qué S.O. hay que elegir para obtener el mayor rendimiento. El Motorola Milestone proporciona la peor productividad (1,43 FPS), debido a que es seis veces más lento en obtener la imagen y casi el doble de lento en detectar la marca que cualquiera de los otros teléfonos de la tabla. Este dispositivo es más lento tanto en la etapa de captura de la imagen, como en la de detección de marca, e incluso en la etapa de dibujado, aunque es cierto que en esta última etapa existe poca diferencia respecto al segundo más lento (iPhone 3GS). Por contra, el dispositivo Nexus One proporciona la mejor productividad (5,98 FPS), aunque es una conclusión obvia, puesto que es el mejor en cada una de las etapas, excepto en la primera. En la etapa de envío la latencia obtenida en los distintos clientes es muy similar (del más lento al más rápido sólo se llevan 10 milisegundos). Este era un resultado previsible, puesto que esta etapa depende más de las características de la red que de la capacidad de cómputo de los teléfonos móviles considerados.

La Tabla 4.2 también muestra que los dispositivos Android son más rápidos que los iOS. Esto se debe a que las imágenes capturadas por los móviles Android son cuatro veces más pequeñas que la imagen capturada por el iPhone 4. Por su parte, el iPhone 3GS necesita menos tiempo que los teléfonos Android en completar la etapa de detección de marca, y eso que no sólo tiene un tamaño de imagen similar al de los Android, sino que el Nexus One tiene un procesador el doble de potente. Otro dato curioso es que la captura de imagen del iPhone 4 es casi el doble de rápida que la del iPhone 3G. Sin embargo, la detección de marca en el iPhone 4 es tres veces más lenta que la del iPhone 3GS. Y eso que la CPU del iPhone 4 es el doble de rápida que la del 3GS. Esto se puede explicar sabiendo que la imagen capturada por el dispositivo iPhone 4 es seis veces mayor que la capturada por el iPhone 3GS. Sin embargo, el teléfono móvil iPhone 4 es más rápido que el 3GS en las etapas de dibujado y envío. Aunque ambos son más lentos que el más rápido de los dispositivos Android en la última etapa.

Un detalle que llama la atención es el hecho de que el tiempo total de ejecución (penúltima columna) debería coincidir con la suma de las cuatro columnas

que representan las cuatro etapas CAR. De hecho, esto es lo que ocurre con los móviles Android. Sin embargo, el tiempo total de ejecución de los dispositivos iOS es casi el doble que la suma de las cuatro columnas. Este comportamiento anómalo se podría deber al hecho de que las aplicaciones Android no necesitan manejar directamente la memoria, puesto que se ejecutan sobre máquinas virtuales, que hacen una gestión automática de la memoria. También se podría deber al hecho de que en la implemetación de Android alguna de las etapas se ejecutan en hilos independientes y no se ejecutan de manera bloqueante. Como ejemplo, se tiene la etapa de dibujado, que está desacoplada del resto de etapas CAR (ejecutada en un hilo aparte), y no espera a tener una nueva posición para realizar el dibujado. Si llegado el momento de dibujar no tiene una posición nueva, lo que hace es dibujar en la posición anterior.

Estos resultados parecen indicar que los dispositivos Android realizan más rasterizados de la escena que los dispositivos iOS. Para confirmar esto, se ha medido el número de etapas de renderizado completadas en comparación con el número de veces que se ejecutan el resto de etapas. Para ello se ha modificado la aplicación que leía los “logs” de las ejecuciones (ver Apéndice D), para que cuente para cada ciclo cuántas veces se ejecuta cada etapa. Dicha aplicación se puede ver en el Apéndice A. Como resultados de estas pruebas se ha obtenido que el Motorola Milestone realiza, de media, 6’28 dibujados por ciclo mientras que el Nexus One realiza 5’55 dibujados por ciclo. Con lo que se confirmaron nuestras suposiciones, pues mientras cada etapa se ejecuta una sola vez en un mismo ciclo CAR, el dibujado se ejecuta más de cinco veces.

El número extra de dibujados depende del número de polígonos y de la cantidad de datos de texturas que haya en el modelo 3D. La Tabla 4.3 muestra la relación existente entre el número de etapas de dibujado por ciclo y la complejidad de la escena 3D para ambos dispositivos Android. Para cambiar la complejidad de la escena a dibujar, se cambiará el tipo de objeto que se dibuja, alternando entre dos modelos clásicos: el cilindro (modelo simple) y el avión (modelo complejo), como se puede ver en la Figura 4.4

La primera columna de la Tabla 4.3 muestra el número de milisegundos que necesita la etapa de dibujado para cada uno de los dispositivos y modelos 3D. La siguiente columna (RRender), indica el cantidad de dibujados que se hacen

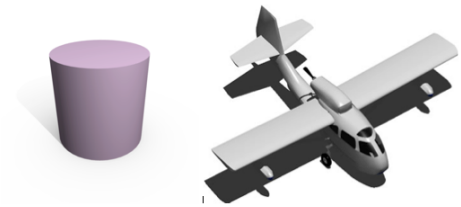

Figura 4.4: Modelos dibujados en la pruebas de complejidad de dibujado para android. A la izquierda modelo simple(cilindro) y a la derecha modelo complejo(avión)

	Render (ms)	**R_Render**	**FPS**
Milestone (simple model)	21,40	18,24	1,23
Milestone (complex model)	48,32	9,90	1,25
Nexus One (simple model)	5,48	6,27	5,80
Nexus One (complex model)	24,46	4,47	5,87

Tabla 4.3: Relación entre el número de dibujados por ciclo y la complejidad de la escena 3D para los diferentes modelos de Android

en un ciclo CAR (repeticiones de la etapa de dibujado) y, finalmente, la última columna (FPS) muestra la productividad de la aplicación, expresada en fps. La Tabla 4.3 muestra que a medida que se aumenta la complegidad del objeto a dibujar, los móviles Android requieren más tiempo para completar la etapa de dibujado. En este experimeno, la media de tiempo necesaria para completar el dibujado del modelo complejo es el doble de la necesaria para el modelo simple. Esta variación es más evidente si se observa la segunda columna de la Tabla 4.3 (R_Render). Si se analiza, por ejemplo, el caso del Motorola Milestone, al pasar del modelo simple al complejo la etapa de dibujado cuesta el doble (de 21,40ms a 48,32), por lo que el número de repeticiones de esta etapa se reduce a la mitad (de 18,24 a 9,9) para poder mantener los mismos FPS, que pasan de 1,23 a 1,25.

Por último, se ha medido el consumo de CPU y la cantidad de memoria uti-

lizada por el teléfono elegido. Los valores obtenidos se pueden observar en al Tabla 4.4. Esto se ha conseguido a través de las plataformas de desarrollo de aplicaciones, puesto que proporcionan herramientas que permiten obtener los valores con cierta facilidad. Para ver un ejemplo de la información obtenida al realizar la traza de una ejecución se puede consultar el Apéndice E.

Dispositivo	Memoria (MB)	CPU (%)
Milestone	25	97
Nexus One	11	96
iPhone 3GS	7	95
iPhone 4	4	97

Tabla 4.4: Memoria y consumo de CPU de los teléfonos considerados

La Tabla 4.4 muestra el consumo de memoria y la utilización de la CPU para los teléfonos considerados en el experimento. En la primera columna aparece la memoria utilizada en la ejecución de la aplicación y medida en Megabytes, mientras que en la segunda columna aparece el porcentaje de utilización de la CPU de la máquina cuando ejecuta la aplicación CAR correspondiente.

La Tabla 4.4 muestra que todos los móviles usados en la prueba están cerca de la saturación (100 % de CPU), por lo que el rastreo de marcas de AR se puede considerar como un proceso intensivo de CPU, ya que requiere el máximo de los recursos de los microprocesadores tanto de la versión de Android (desacoplada) como de la de iOS (normal). En cuanto al consumo de memoria, se observa que los dispositivos Android consumen más memoria que los iOS al ejecutar aplicaciones CAR. Este incremento de la memoria consumida puede explicarse por la gestión que se hace de la memoria en los dispositivos Android, dado que, a diferencia de los dispositivos iOS, donde el programador es el encargado de controlar la cantidad de memoria reservada para la aplicación así como de liberarla tras su uso, en Android se hace uso de la *Máquina Virtual* (VM[7]). Dicha VM es la que gestiona la memoria en todas las aplicaciones Java, haciendo transparente para el desarrollador todo lo referente al manejo de memoria. Y, precisamente por hacerlo transparente al programador, no siempre se comporta de la manera

[7] De sus siglas en inglés: “Virtual Machine”

más eficiente para la aplicación.

4.4. Implementaciones sin marcas

Como se comentó al principio de este capítulo, el ciclo de trabajo de las aplicaciones CAR se puede descomponer en cuatro etapas: Adquisición de imagen, Seguimiento de marcas, Dibujado y Envío. En la etapa de seguimiento había dos vertientes: la de marcas artificiales y la de vectores de características. Esta segunda opción es la que se verá en la presente sección. Aunque se comentó que la detección de marcas era el sistema de seguimiento más extendido, durante la realización de este trabajo de investigación, ha surgido un nuevo sistema sin marcas llamado Vuforia, de la empresa Qualcomm (Qualcomm Vuforia, 2011). Dicho sistema se ha extendido con gran rapidez y ya se utiliza como un estándar de facto. Por lo tanto, no se podía considerar como terminada la caracterización de los teléfonos móviles para su uso sobre CAR si no se extendía la evaluación de prestaciones incluyendo este nuevo sistema.

Vuforia es un SDK[8] de Realidad Aumentada para teléfonos móviles y tabletas que permite ejecutar aplicaciones AR en tiempo real. Su software utiliza la tecnología de visión por computador para reconocer y realizar el seguimiento individual de los objetos capturados por la cámara de vídeo en tiempo real. Se usará el SDK en su versión 2.0, pues es la más reciente, que soporta diferentes objetivos, tanto 2D como 3D, incluyendo configuraciones multi-objetivos. El SDK tiene otras características, como la detección de oclusiones usando botones virtuales, selección de imágenes objetivo en tiempo real y la posibilidad de reconfigurar y crear conjuntos de objetivos en función del escenario (Paredes Figueras, 2013). El SDK también proporciona un API para los lenguajes de programación Java, C++ y .Net, a través de una extensión de Unity3D[9] (Unity, 2014). Dicho motor se comentará más en detalle en el capítulo del servidor, donde se utilizará para realizar un interfaz gráfico para el sistema CAR piloto. En cuanto a la librería en sí y a cómo funciona poco se puede comentar aquí, puesto que es código cerrado,

[8] Kit de desarrollo de aplicaciones, del inglés “Software Development Kit”

[9] Motor de juegos multiplataforma que proporciona su propio entorno de desarrollo o IDE (Integrated Developer Environment)

a diferencia de las usadas en la versión con marcas (NyARToolKit y ARToolKit Plus). Se sabe que las imágenes objetivo no necesitan regiones especiales en blanco y negro o códigos para ser reconocidos. El SDK de Vuforia usa un conjunto de algoritmos para detectar y seguir las características presentes en una imagen, reconociéndolas mediante la comparación de dichas características con una base de datos de objetos conocidos. Tras detectarla, Vuforia seguirá la imagen mientras permanezca en el flujo de la cámara. Las imágenes objetivo se crean on-line con el sistema de manejo de objetivos (TMS[10]) a partir de imágenes en JPG o PNG. Las características que se extraen de esas imágenes son las que se almacenan en la base de datos y se usan para comparaciones en tiempo real (Paredes Figueras, 2013).

Como se comentó anteriormente, las aplicaciones AR desarrolladas con Vuforia son compatibles con una gran variedad de teléfonos móviles como los iPhone (3GS, 4, 4S y 5), el iPad y los teléfonos y tabletas Android que tengan una versión de S.O. igual o superior a la 2.2. Por lo tanto, no existe ningún problema para ejecutar las pruebas sobre los terminales utilizados en la implementación anterior, que eran el teléfono móvil Motorola Milestone y el HTC Nexus One para Android, y el iPhone 3GS y el iPhone 4 para Apple. De esta forma se puede realizar una comparación lo más justa posible entre las implementaciones.

Las aplicaciones de partida para nuestro desarrollo, tanto de Android como de iOS, son las que proporciona el SDK de Vuforia como aplicaciones de ejemplo de AR. Ambas implementaciones son muy similares, de hecho, primero se desarrolló la aplicación para iOS en Objetive-C y, a continuación, se utilizó JNI para portarla a la versión Java de Android. De nuevo, se utilizará Eclipse para desarrollar la aplicación para Android y xCode para desarrollar la de iOS. Para el dibujado de ambas implementaciones se utilizará OpenGL, como en la anterior implementación. Como las aplicaciones que proporciona Vuforia son de AR, se les añadirá la etapa de envío de información para convertirlas en aplicaciones CAR. Para ello se implementará un socket TCP como el utilizado en las versiones de la implementación con marcas. Al igual que antes, se analizará en profundidad el código proporcionado por Vuforia con el fin de delimitar las etapas CAR y se pondrá marcas de tiempo al inicio y final de cada etapa, para saber cuánto tiempo consume el terminal en cada una de ellas. Las modificaciones del código

[10] De sus siglas en inglés: “Target Management System”

Figura 4.5: Imagen de referencia para las implementaciones de Vuforia

se pueden ver en el Apéndice F. Como esta implementación es sin marcas, la imagen almacenada en la base de datos que sirve de referencia es la que se muestra en la Figura 4.5.

En cuanto a la toma de datos de las pruebas, tras cada ejecución el teléfono móvil genera un log. En dicho log se encuentran las marcas de tiempo correspondientes a cada etapa del ciclo CAR, para cada uno de los ciclos que haya durado la simulación. Por lo tanto, se hará uso de la aplicación empleada en la implementación anterior para analizar los ficheros de "log" y obtener las estadísticas de cada ejecución. Dicha aplicación se puede consultar en el Apéndice D.

4.5. Evaluación de prestaciones de la implementación sin marcas

En la presente sección se verá la evaluación de prestaciones de los teléfonos móviles al usarlos en aplicaciones CAR que no hacen uso de marcas. Se medirá la latencia y los fps que proporciona el sistema en función del teléfono móvil utilizado. De nuevo, se entenderá por latencia el tiempo que transcurre desde que el cliente emite el mensaje hasta que el sistema le contesta, medido en milisegundos. El resumen de las características de los móviles utilizados se puede ver en la Tabla 4.1. Con el fin de comparar ambas implementaciones, en la Tabla 4.5 se verán los resultados obtenidos en ambas. La parte superior de la tabla, etiquetada como NyARToolKit & ARToolKitPlus, corresponde a la primera implementación y la etiquetada como Vuforia corresponde a la implementación

NyARToolKit & ARToolKit Plus					
Etapas (ms)	**Cámara**	**Detección**	**Dibujado**	**Envío**	**Total**
Milestone	248,64	288,53	30,42	14,14	698,34
Nexus One	40,25	78,08	13,23	5,54	167,11
iPhone 3GS	33,29	58,07	28,26	15,42	398,00
iPhone 4	17,66	182,17	23,34	7,06	523,26
Vuforia					
Etapas (ms)	**Cámara**	**Detección**	**Dibujado**	**Envío**	**Total**
Milestone	19,58	3,95	2,38	15,06	86,41
Nexus One	9,97	18,03	1,76	8,48	56,68
iPhone 3GS	11,06	8,23	14,29	17,38	145,53
iPhone 4	8,15	6,21	17,85	9,68	105,91

Tabla 4.5: Tiempos de ejecución (ms) de cada etapa CAR para cada uno de los móviles considerados en cada una de las implementaciones propuestas

sin marcas.

Cada fila de la Tabla 4.5 presenta los resultados para un teléfono móvil diferente. Las dos primeras filas muestran los móviles de Android (ejemplos representativos de las gamas alta y baja de los dispositivos encontrados en el mercado), y las dos últimas filas muestran el equivalente para iOS. Los mismos dispositivos se usan en la parte baja de la tabla. Por su parte, las cuatro primeras columnas representan cada una de las etapas de las aplicaciones CAR y, la última columna representa el tiempo necesario para completar un ciclo. Todos los valores se representan en milisegundos (ms), y son valores promediados de decenas de pruebas. Como puede observarse, el sistema sin marcas es mucho más rápido que el sistema con marcas en cada una de las etapas CAR, a excepción del envío, cosa previsible puesto que esta etapa sólo depende de los parámetros y estado de la red, y se ha usado la misma en ambas caracterizaciones.

La Tabla 4.5 muestra que el tiempo requerido por la adquisición de imagen en la implementación sin marcas es, al menos, la mitad del requerido por la

implementación con marcas (como es el caso del iPhone 4), mientras que en el caso del iPhone 3GS funciona el triple de rápido, de 33'29ms. a 11'06ms.. Sin embargo, la mejora más drástica es la conseguida por el Motorola Milestone, pasando de los 248'64ms. de la implementación con marcas a tan solo 19'58ms. en la implementación sin marcas.

En cuanto a la segunda etapa, las diferencias de rendimiento aún son más claras. El tiempo necesario para completar esta etapa por la implementación con marcas es, como mínimo, cuatro veces más lento que el requerido para la implementación sin marcas. El motivo de esta mejora tan drástica es difícil de descubrir, puesto que Qualcomm no proporciona el código fuente de Vuforia, sino sólo su librería compilada. Vale la pena recordar que cada teléfono proporciona unas resoluciones diferentes, y que el tamaño de las imágenes proporcionadas por los dispositivos Android es muy diferente del de las proporcionadas por los dispositivos iOS. No es lo mismo analizar una imagen de 320x480 píxeles buscando una marca que hacerlo en una imagen de 1280x720 píxeles, puesto que se necesita mucho más tiempo en esta última.

Por lo que respecta a la tercera etapa, la Tabla 4.5 muestra que la implementación si marcas no mejora drásticamente en todos los terminales, de hecho en el iPhone 4 la mejora es de menos de 6ms. En el caso del iPhone 3GS, funciona el doble de rápido. Por su parte, los dispositivos android funcionan mucho más rápido, el Nexus One se ejecuta ocho veces más rápido y el Milestone más de 12 veces. Esta mejora tan sustancial en los dispositivos Android se justifica, en parte, en el hecho de que el renderizado de la implementación sin marcas no está en un hilo aparte, como pasaba en la anterior implementación. Por lo tanto, no ejecuta más de un renderizado por ciclo.

Por último, la columna de la derecha representa el tiempo necesario para concluir el ciclo CAR en cada uno de los dispositivos. Comparando ambas implementaciones se puede concluir que todos los dispositivos funcionan, al menos, tres veces más rápido al usar la implementación sin marcas. Por lo que esta implementación es la mejor opción tanto para dispositivos de gama alta como baja. Una de los motivos que contribuyen a obtener estos resultados es que las librerías de la versión con marcas no fueron diseñadas explícitamente para los móviles actuales, por lo que no pueden sacarle partido a todo el hardware de que

disponen los dispositivos actuales, como la GPU. No obstante, se recuerda que el código fuente de la librería Vuforia es código cerrado y no se puede analizar en profundidad, a diferencia de las librerías empleadas en la implementación con marcas.

Para mostrar el efecto práctico de las prestaciones obtenidas con ambas implementaciones, la Tabla 4.6 muestra los valores obtenidos en la Tabla 4.5, pero expresados en parámetros típicos de evaluación de sistemas como los FPS, que es la cantidad de ciclos CAR hechos en un segundo, y el RTT[11], que es el tiempo de ida y vuelta de un mensaje. De nuevo, cada fila representa el dispositivo utilizado. Los FPS se muestran en la primera columna y el RTT en la segunda (medidos en ms), y corresponde con la cuarta etapa mostrada en la Tabla 4.5.

En la Tabla 4.6 se observa que el dispositivo más lento usando la implementación de Vuforia es el iPhone 3GS, trabajando a 7 FPS. Este dispositivo *lento* en Vuforia es más rápido que cualquiera de los otros móviles en la implementación con marcas(etiquetada como ARTookKit), donde el dispositivo más rápido era el Nexus One con casi 6 FPS.

	ARToolKit		**Vuforia**	
	FPS	**RTT(ms)**	**FPS**	**RTT(ms)**
Milestone	1,43	14,14	11,57	15,06
Nexus One	5,98	5,54	17,64	8,48
iPhone 3GS	2,51	15,42	6,87	17,38
iPhone 4	1,91	7,06	9,44	9,68

Tabla 4.6: productividad (FPS) y latencia (RTT) de los dispositivos en cada una de las implementaciones

[11] De sus siglas en inglés: “Round Trip Time”

4.6. Siguiente generación de terminales

Los teléfonos móviles mejoran sus prestaciones constantemente, por lo tanto, no es de extrañar que lo que se entendía gama alta al principio de una anualidad, se convierta en gama media o baja al año siguiente. Durante el transcurso de esta investigación se ha sufrido este efecto constantemente, por lo que se centró el estudio en unos teléfonos móviles en concreto y no se actualizó la caracterización con cada nueva actualización de dispositivos. No obstante, al realizar la segunda caracterización con Vuforia, se completó el estudio con nuevos dispositivos para que resultasen un poco más acordes al mercado de ese momento. Por lo tanto, se ha realizado las mismas pruebas con teléfonos móviles de gama alta del mercado de finales del 2012, como son el dispositivo Samsung Galaxy SIII para Android y el terminal iPhone 4S para iOS. Con el fin de tener una vista más completa de la caracterización, se pondrán las tablas de la evaluación de prestaciones anterior pero incluyendo los nuevos dispositivos. En la Tabla 4.7 se pueden ver las características de los teléfonos móviles escogidos.

OS	**Android**			**iOS**		
Model	**Milest.**	**Nex. One**	**S.G.SIII**	**iPh 3GS**	**iPh 4**	**iPh 4S**
CPU	TI OMAP	Qual.	32 bit Samsung	Samsung	Apple	Apple A5
	3430	QSD8250	Exynos 4412	S5PC100	APL0398	APL0498
Freq. (MHz)	550	998	1400	412	800	800
GPU	SGX530	Qual.	ARM	SGX535	SGX535	PowerVR
		Adreno200	Mali-400			SGX543MP2
RAM (MB)	256	512	1024	128	512	512
Camera (MP)	5.02	4.92	7.99	1.92	4.92	7.99
Núcleos	1	1	4	1	1	2

Tabla 4.7: Datos del Hardware de los móviles usados en la caracterización

En la Tabla 4.7 se observa que el terminal Samsung Galaxy SIII tiene un procesador de 1400 MHz con 4 núcleos y una cámara de 8 megapíxeles, mientras que el teléfono iPhone 4S tiene un procesador de 2 núcleos con 800 MHz y también una resolución de cámara correspondiente a 8 megapíxeles.

Con estos dispositivos se ha repetido la caracterización de las implementaciones con marcas (con NyARToolKit para Android y ARToolKit plus para iOS) y sin marcas (con Vuforia), obteniendo los resultados mostrados en la Tabla 4.8.

NyARToolKit & ARToolKit Plus						
Etapas (ms)	**Cámara**	**Detección**	**Dibujado**	**Envío**	**Total**	**FPS**
Milestone	248,64	288,53	30,42	14,14	698,34	1,43
Nexus One	40,25	78,08	13,23	5,54	167,11	5,98
iPhone 3GS	33,29	58,07	28,26	15,42	398,00	2,51
iPhone 4	17,66	182,17	23,34	7,06	523,26	1,91
iPhone 4S	1,19	114,64	6,37	8,30	182,46	5,48
Samsung G SIII	60,05	9,34	5,90	7,90	128,02	7,81
Vuforia						
Etapas (ms)	**Cámara**	**Detección**	**Dibujado**	**Envío**	**Total**	**FPS**
Milestone	19,58	3,95	2,38	15,06	86,41	11,57
Nexus One	9,97	18,03	1,76	8,48	56,68	17,64
iPhone 3GS	11,06	8,23	14,29	17,38	145,53	6,87
iPhone 4	8,15	6,21	17,85	9,68	105,91	9,44
iPhone 4S	2,73	2,10	3,40	9,06	43,58	22,95
Samsung G SIII	18,30	9,20	0,46	4,67	34,11	29,32

Tabla 4.8: Tiempos de ejecución (ms) de cada etapa CAR incluyendo dispositivos multi-núcleo

En la Tabla 4.8 se pueden ver siete columnas. La primera de ellas indica el teléfono móvil elegido para la simulación. Las cuatro siguientes se corresponden a cada una de las etapas CAR, medidas en milisegundos. La siguiente columna, con el rótulo “Total”, se corresponde al tiempo necesario para completar el ciclo CAR, también en milisegundos. Por último, la columna de los FPS representa la cantidad de ciclos que se llevan a cabo en un segundo. Como ya se comentó los resultados de los dispositivos mono-núcleo en las secciones 4.3 y 4.5, se comentarán aquí los resultados de los dispositivos multi-núcleo.

En la implementación con marcas se observa que el dispositivo Android, pese a tener el doble de núcleos que el dispositivo iOS, obtiene un rendimiento ligera-

mente superior que éste último. Por otra parte, el hecho de añadir más núcleos no permite al iPhone 4S conseguir mejor rendimiento que el mejor de los dispositivos mono-núcleo (el Nexus One). Así mismo, los tiempos de envío del Nexus One y el iPhone 4 son mejores que sus respectivos multi-núcleo. Estos resultados indican que el uso de dispositivos multi-núcleo con la implementación con marcas mejora muy poco las prestaciones. Esta mejora es más significativa en los dispositivos iOS, puesto que el iPhone 4 y el iPhone 4S trabajan a la misma frecuencia (800MHz) y, añadiendo un único núcleo, el iPhone 4S obtiene más del doble de FPS, y eso a pesar de tener que analizar una imagen más grande que la de su antecesor (de cinco megapíxeles a ocho). Por su parte, los dispositivos Android obtienen una mejoría más leve, puesto que el Samsung Galaxy SIII, con más potencia y tres núcleos extra sólo proporciona un incremento de dos FPS (de 5'98 a 7'81). Por lo que se puede decir que iOS hace un mejor uso de los núcleos extra.

Por su parte, la implementación sin marcas sí que obtiene una mejora de prestaciones considerable con la adición de mejores prestaciones, puesto que ambos dispositivos multi-núcleo sobrepasan la productividad (FPS) alcanzada por los mejores mono-núcleo, llegando casi a los 30 fps. Esto se debe, como se comentó en el apartado anterior, a que Vuforia es una librería mucho más reciente que ARToolKit/ARToolKit Plus (más de diez años más joven que ARToolkit y más de seis que ARToolKit Plus) y, por tanto, ha sido diseñada para aprovechar las prestaciones de los teléfonos móviles actuales. Se observa, por ejemplo, como en iOS al añadir un núcleo extra (de iPhone 4 a iPhone 4S) la productividad se duplica, pasando de 9,44 fps a 22,95 fps. De nuevo, el incremento de núcleos en Android no obtiene unos beneficios proporcionales, puesto que pasa de 17,64 fps a 29,32 fps tras añadir tres núcleos al procesado. En esta implementación las fases de detección y dibujado mejoran sustancialmente tras la agregación de núcleos, pasando, por ejemplo, en iOS de los 17,85ms. obtenidos en el iPhone 4 a tan solo 3,40ms. en el iPhone 4S y en Android consiguiendo dibujados de menos de medio milisegundo.

Como se comentó anteriormente, en la implementación con marcas la etapa de dibujado está desacoplada en los dispositivos Android, es decir, se ejecuta en un hilo aparte. Este hecho provoca que el hilo de dibujado se ejecute independientemente, de forma que, cuando renderice la escena 3D, si aún no tiene la

nueva posición, dibuja en la posición anterior. Por lo tanto, esta etapa se ejecuta más veces que las demás. El número de veces que se repite por cada ciclo dependerá del número de polígonos y la cantidad de texturas del modelo a dibujar. La Tabla 4.9 muestra la relación entre el número de dibujados por ciclo y la complejidad de la escena para cada dispositivo Android (véase los objetos a dibujar en la Figura 4.4.

	Render (ms)	**R_Render**	**FPS**
Milestone (simple model)	21,40	18,24	1,23
Milestone (complex model)	48,32	9,90	1,25
Nexus One (simple model)	5,48	6,27	5,80
Nexus One (complex model)	24,46	4,47	5,87
Samsung G SIII (simple model)	3,26	5,73	7,95
Samsung G SIII (complex model)	9,83	3,11	7,96

Tabla 4.9: Relación entre el número de dibujados por ciclo y la complejidad del modelo 3D para los dispositivos Android

La primera columna de la Tabla 4.9 muestra el tiempo en milisegundos que necesita la etapa de dibujado para terminar. La siguiente columna muestra la cantidad de dibujados que se hacen mientras se completa el ciclo CAR y, la última muestra la productividad, medida en FPS.

La Tabla 4.9 demuestra que a medida que aumenta la complejidad del modelo a dibujar, la etapa de dibujado consume más tiempo. En este experimento, el tiempo necesario para completar el dibujado complejo es el doble que el requerido para el dibujado simple. Esta variación es más evidente para el número de repeticiones de la etapa de dibujado. Como se necesita más tiempo para dibujar el modelo, el número de repeticiones dentro de un mismo ciclo CAR se reduce para mantener constante la productividad de la aplicación. En este caso, el uso de dispositivos multi-núcleo no cambia significativamente el rendimiento obtenido.

Adicionalmente, se ha medido el consumo de CPU y la memoria consumida

por los móviles usados en las pruebas. La Tabla 4.10 muestra estos resultados. En la primera columna aparece la memoria consumida (en MB) y en la segunda columna se tiene el porcentaje de CPU utilizado en las pruebas. Anteriormente se vio que los dispositivos mono-núcleo están cerca de saturación (100 % de CPU), pero en la Tabla 4.10 se observa que el proceso de seguimiento de marcas utiliza solo un pequeño porcentaje de la CPU de los dispositivos multi-núcleo, bajando del 95 % a un 14 % en el dispositivo iOS con doble núcleo. También se observa que existe muy poca variación en la utilización de CPU de un dispositivo con dos núcleos a otro con cuatro (tan solo un 3 %). Y, que las aplicaciones CAR no tienen por qué consumir una gran cantidad de memoria.

Dispositivo	Memoria (MB)	CPU (%)
Milestone	25	97
Nexus One	11	96
iPhone 3GS	7	95
iPhone 4	4	97
iPhone 4S	7	14
Samsung G SIII	9	11

Tabla 4.10: Consumo de CPU y memoria utilizada por los dispositivos mono-núcleo y multi-núcleo

4.7. Conclusiones

En este capítulo se ha propuesto la caracterización de los teléfonos móviles para aplicaciones CAR con seguimiento de marcas y sin marcas, un proceso esencial que se lleva a cabo en toda aplicación CAR. Con el fin de realizar un análisis robusto de los móviles del mercado, se ha considerado teléfonos móviles basados en Android e iOS. En ellos, se ha ejecutado una aplicación CAR simple mientras se medían sus prestaciones.

Los resultados de la evaluación de prestaciones de la implementación *con*

marcas muestran que cuando se ejecuta la misma aplicación CAR sobre los diferentes móviles, la mejor productividad (medida en fps), la obtienen los móviles basados en Android. Sin embargo, para móviles de gama baja, son los dispositivos iOS los que obtienen mejores resultados.

También se ha estudiado las diferentes etapas en las que se descomponen las aplicaciones CAR de seguimiento (tanto de marcas como sin ellas). Los resultados de la implementación con marcas muestran que la etapa que más tiempo consume es la de detección de marcas, seguida de la captura de imágenes, la etapa de dibujado y, por último, la etapa de transmisión. En cuanto a los S.O., se ha visto que la etapa de dibujado de los dispositivos Android está desacoplada, de forma que se ejecuta a la vez que las otras tres etapas. Aunque esto no impide que las mejores prestaciones se obtengan con este S.O.. Es más, esta etapa se puede re-programar para que se ejecute junto con las otras realizando una implementación *ad hoc*.

Por otra parte, los resultados también muestran que algunos móviles como el terminal iPhone 4, sólo funcionan con resoluciones grandes de imagen y, como resultado, consiguen una gran calidad de imagen pero a costa de emplear una gran cantidad de tiempo en la etapa de detección.

La implementación sin marcas tiene una productividad mucho mejor que la implementación con marcas. La mejor productividad (medida en fps) la siguen obteniendo los móviles basados en Android, esta vez tanto los de gama alta como los de gama media. Por su parte, el tiempo consumido por cada etapa del ciclo CAR es muy similar, necesitando todas menos de 20 ms para finalizar. Como las implementaciones para Android e iOS son transcripciones idénticas del mismo código, los móviles Android no sufren de desacoplamiento de la etapa de dibujado, por lo que su rendimiento es aún un poco mejor. No obstante, sigue habiendo partes del ciclo de los dispositivos iOS en las que el S.O. pierde tiempo, puesto que la suma de los tiempos consumidos en cada etapa sigue siendo muy diferente del tiempo total de ciclo CAR.

Finalmente, los dispositivos con varios núcleos no proporcionan una productividad lineal con el número de núcleos, e incluso hay un dispositivo que con un único núcleo obtiene mejores resultados tanto en productividad como en latencia.

La explicación es que las cuatro etapas de los sistemas CAR son inherentemente secuenciales y, por lo tanto, no se puede sacar beneficio a los núcleos extra.

4.8. Impacto de las nuevas generaciones de dispositivos móviles

En el momento de la redacción y experimentación de esta tesis los teléfonos móviles de gama alta correspondiente a los terminales Samsung Galaxy S III de Android y Apple iPhone 4S. No obstante, en el momento de la redacción de la tesis, los últimos lanzamientos de las compañías de móviles más importantes son el teléfono móvil Google Nexus 5 de Android y el iPhone 5S de Apple, cuyas características hardware se muestran en la Tabla 4.11, comparadas con las de los dispositivos de la última generación analizada, es decir, el Samsung Galaxy S III y el iPhone 4S.

OS	**Android**		**iOS**	
Model	**S.G.SIII**	**Nexus 5**	**iPh 4S**	**iPh 5S**
CPU	32 bit Samsung	32 bit Qualcomm	Apple A5	64 bit Apple A7
	Exynos 4412	Snapdragon 800 MSM8974AA	APL0498	APL0698
Freq. (MHz)	1400	2265	800	1300
GPU	ARM	Qualcomm	PowerVR	PowerVR
	Mali-400	Adreno 330	SGX543MP2	G6430MP4
RAM (MB)	1024	2048	512	1024
Camera (MP)	7.99	7.99	7.99	7.99
Núcleos	4	4	2	2

Tabla 4.11: Datos del Hardware de los móviles usados en la caracterización

En la Tabla 4.11 se observa que el teléfono móvil Nexus 5 tiene un procesador Qualcomm que funciona a 2265 MHz frente a los 1400 MHz del Samsung Galaxy S III. En cuanto a la GPU, el Nexus 5 utiliza la Adreno 330, también de Qualcomm, primera clasificada en el ranking de la web “Notebookcheck” a 11 de Diciembre de 2013 (Ranking GPUs, 2014), frente a la Mali-400 considerada décimo cuarta según ese mismo ranking. Por lo que respecta a la memoria RAM, el teléfono Nexus 5 duplica la del Samsung Galaxy S III con 2048 MB. Ambos tienen una cámara de 8 megapíxeles y cuatro núcleos. Por su parte, el iPhone 5S dispone

de un procesador de 64 bits que funciona a 1300 MHz, frente al procesador de 32 bits del iPhone 4S que funciona a 800 MHz. El iPhone 5S sigue utilizando una GPU PowerVR, pero cambia al modelo “G6430MP4” que está en el segundo puesto del ranking de “Notebookcheck”, frente a la GPU del iPhone 4S que figura en la novena posición. Al igual que ocurre con los dispositivos Android, el último teléfono móvil de Apple duplica la RAM pero mantiene la misma resolución de cámara y cantidad de núcleos que el iPhone 4S, quedándose con 8 megapíxeles y 2 núcleos.

A efectos prácticos y para el desarrollo de la tesis, las nuevas características hardware de los dispositivos móviles se reflejan en que los tiempos de computación de las cuatro etapas en las que se descompone el ciclo de trabajo de las aplicaciones CAR se reducirán. En primer lugar, la adquisición de la imagen será más rápida, puesto que los buses actuales te permiten obtener la imagen procedente del “buffer” de la cámara más eficientemente. En segundo lugar, la detección de marca se realizará más rápido debido al incremento de la capacidad de cómputo proporcionado por los nuevos procesadores. En tercer lugar, el renderizado de las imágenes en pantalla se acelerará por las mejoras en la GPU. Y, por último, el envío de la información será más rápido por la mejora de las interfaces de red y por la utilización de normas nuevas como la 802.11ac (Norma 802.11ac, 2014). Los efectos de tales mejoras se traducirán en que el ciclo de actuación de los clientes del sistema CAR se reduzca y, por lo tanto, actúen con mayor frecuencia. A su vez, esto supondrá una modificación de las cotas superior e inferior de frecuencia de actuación de los dispositivos móviles. Como se demostró con las dos generaciones anteriores de dispositivos móviles, donde se vio que la caracterización de los teléfonos de la segunda generación se podía extrapolar de la caracterización de los teléfonos móviles de la primera generación, la aparición de una nueva generación de dispositivos móviles no invalida la caracterización de los mismos realizada en el presente capítulo. Por el contrario, los resultados previsibles para la nueva generación de terminales de telefonía móvil se pueden extrapolar de la caracterización de las anteriores generaciones.

5

Simulador del sistema CAR

5.1. Introducción

En el capítulo anterior se analizó el comportamiento de los dispositivos cliente basados en telefonía móvil cuando actúan individualmente frente a un servidor. En este capítulo se va a analizar el comportamiento y las prestaciones del sistema distribuido compuesto por el servidor de la aplicación y los distintos clientes conectados a él al intercambiar información utilizándolo de intermediario. Como se opera en un sistema *colaborativo* y distribuido, cada dispositivo cliente simulado enviará al resto de clientes pertenecientes a su grupo de trabajo[1] la posición donde ha encontrado la marca buscada. Esta tarea se realizará en cada uno de sus ciclos de actuación, y a su vez se contestará a los vecinos un mensaje de

[1] A los que se hará referencia como vecinos

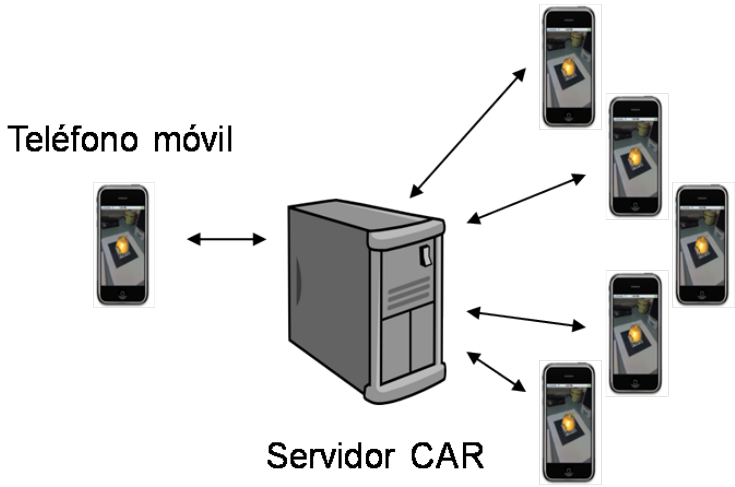

Figura 5.1: Arquitectura cliente-servidor para el sistema CAR piloto propuesto

reconocimiento (mensajes conocidos usualmente como ACK[2]) por cada mensaje de actualización de posición que llegue de ellos, indicando así que le ha llegado la información. Para ello, se necesita una arquitectura de cliente-servidor, en la que los teléfonos móviles hagan de clientes y se comuniquen entre ellos a través de un servidor. La Figura 5.1 muestra un ejemplo de esta arquitectura monoservidor.

Con el fin de recrear un sistema lo más real posible, se simulará teléfonos móviles reales mediante hilos ejecutándose en varios computadores cliente. Cada hilo simulará un dispositivo cliente con el comportamiento análogo al caracterizado en el capítulo anterior. Este comportamiento está basado en el ciclo de actuación descrito en la Figura 5.2, compuesto de: El primer paso del ciclo de actuación de cada cliente consistirá en esperar un tiempo equivalente a la obtención de la posición de la marca en la imagen capturada. En el segundo paso, y antes de enviar la nueva posición al servidor, el cliente esperará un máximo de 20 segundos a haber recibido los mensajes ACK de sus vecinos en el grupo de trabajo[3]. Si pasado ese tiempo falta algún mensaje se dará por perdido para ase-

[2]Del inglés “ACKnowledge”

[3]Se han realizado pruebas con diferentes cantidades de tiempo y se ha visto que todos las respuestas ACK que tienen que llegar al cliente origen lo hacen *siempre* antes de que concluya ese tiempo.

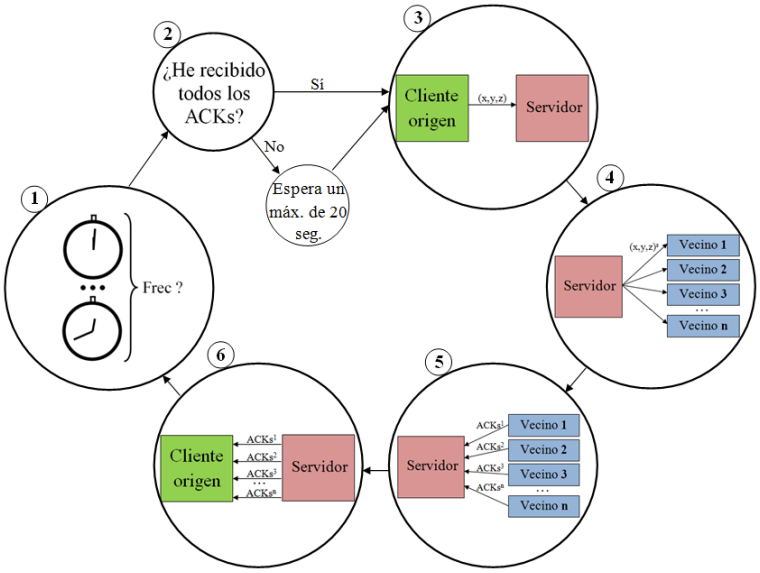

Figura 5.2: Pasos del ciclo de actuación de cada teléfono móvil en la simulación

gurar que la simulación siempre termina. En el tercer paso del ciclo, se entiende que todos los mensajes ACK se han recibido y se envía la posición al servidor. En el cuarto paso el servidor retransmite la posición del cliente emisor a todos los clientes vecinos de su grupo de trabajo. En el quinto paso cada cliente contesta a esa posición enviando un mensaje ACK al servidor. Y, por último, en el sexto paso el servidor retransmite cada respuesta ACK recibida al cliente emisor.

5.2. Diseño del simulador

Tal y como se mostró en la Figura 5.1, la arquitectura que se utiliza para los sistemas CAR corresponde a una organización de tipo *cliente/servidor*. La razón por la que se utiliza esta arquitectura reside en que la cantidad de clientes que generalmente se ven involucrados en los sistemas CAR es pequeña, del orden de la decena de clientes. Y, para esa cantidad, un único servidor es capaz de gestionar todo el sistema. Uno de los objetivos de este trabajo consiste en investigar hasta dónde llegan las prestaciones de esta arquitectura y qué se puede hacer para mejorarlas o, por el contrario, ver si es necesario plantear otras

arquitecturas cuando el número de clientes crece.

Dado que se asume el modelo clientes-servidor se tendrá un único computador servidor y un gran número de clientes, agrupados en grupos de trabajo(WG[4]). Dichos grupos de trabajo representan agrupaciones de dispositivos móviles que comparten un mismo espacio, como pueden serlo un grupo de operarios viendo el mismo motor, o un grupo de turistas observando el mismo cuadro. Por lo tanto, los clientes pertenecientes a un determinado WG sólo intercambiarán información dentro de ese grupo. El tamaño de los WG es un parámetro del simulador que se comentará más adelante.

De entre las opciones posibles, se ha decidido que la aplicación que simula el comportamiento de un servidor CAR ejecute un único *proceso servidor*, simplificando así la gestión del mismo. La Figura 5.3 muestra el esquema de dicho proceso servidor, encargado de calcular el consumo de CPU en el computador que ejecuta el servidor y de generar un *hilo servidor* por cada WG. Estos hilos servidor son los encargados de gestionar cada WG, y básicamente realizan dos tareas: operar como intermediario entre los clientes del WG y almacenar los tiempos de respuesta de cada comunicación. Para llevar a cabo la primera de ellas, generan un *hilo receptor* y un *hilo procesador* por cada cliente del WG. El hilo receptor asocia un socket TCP a cada cliente y, conforme recibe datos por ese socket, los almacena en una cola asignada a ese cliente. Por su parte, el hilo procesador se dedica a sacar la información de la cola del cliente y a procesarla. Puede recibir mensajes de dos tipos: actualizaciones de posición de otros clientes o acuses de recibo de que una posición propia ha llegado a algún vecino (ACK). Si el cliente recibe una actualización de posición de un cliente vecino le contestará con un mensaje ACK y si le llega un ACK de un vecino lo contabilizará para calcular el tiempo de respuesta de cada posición enviada. Estos tiempos de respuesta se enviarán al hilo servidor, tras finalizar la prueba, para que realice su segunda tarea, es decir, el almacenamiento de esos tiempos de respuesta para calcular datos de la simulación como el tiempo de respuesta medio y la desviación estándar de ese tiempo de respuesta.

A la hora de diseñar los clientes, se ha decidido agruparlos en procesos, de forma que cada proceso tenga 50 hilos cliente. Se ha elegido esta cantidad en

[4]Del inglés “Working Group”

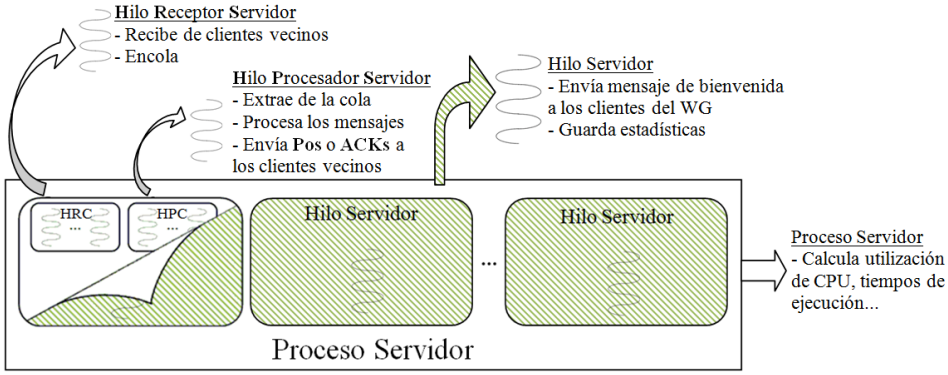

Figura 5.3: Esquema general del proceso servidor del simulador CAR

busca de un compromiso entre la cantidad de hilos y procesos del sistema, de forma que sea lo suficientemente eficiente para reducir el número de procesos en el sistema y no sature al computador que ejecuta a los clientes para que no desvirtúe la simulación. Asimismo, se pretende que las pruebas del sistema se ejecuten para una cantidad de clientes que vaya desde 100 hasta 1000 unidades con incrementos de 100 clientes (como se comentará en el apartado de parámetros de las pruebas), y se busca un valor que sea divisor de esas cantidades para que sea más fácil lanzar las pruebas, por lo que el valor de 50 clientes sirve. Por otra parte, como el valor máximo de clientes va a ser de 1000 unidades, puesto que es una cantidad que sobrepasa con creces la cantidad de clientes presentes en las aplicaciones CAR actuales, se ha decidido ejecutarlos repartidos homogéneamente en dos computadores. Se hace el reparto en dos computadores porque experimentalmente se ha comprobado que así el consumo de CPU en ambos nunca sobrepasa del 50 %. De esta forma, si se quiere lanzar la prueba de 1000 clientes se ejecutarán 10 procesos cliente en cada máquina cliente, teniendo 500 (10x50) clientes por computador.

La Figura 5.4 muestra el esquema general del *proceso cliente*. Dicho proceso es el encargado de calcular el consumo de CPU en el computador que ejecuta el cliente, pero como puede haber varios procesos cliente en el mismo computador, sólo el primero de ellos realizará el cálculo. Como se ha comentado, el proceso cliente se compone de 50 *hilos cliente*. Cada uno de estos hilos cliente genera un *hilo receptor* y un *hilo procesador*, que tienen un comportamiento análogo al descrito en el caso del servidor. El hilo receptor tiene un socket TCP asociado y

se dedica a encolar todos los mensajes que recibe por él en la cola del cliente. Por su parte, el hilo procesador se dedica a desencolar los mensajes que recibe y a procesarlos según su procedencia: si recibe un mensaje de posición de un vecino le responde con un ACK, y si recibe un ACK lo contabiliza para saber cuántos vecinos han recibido su posición.

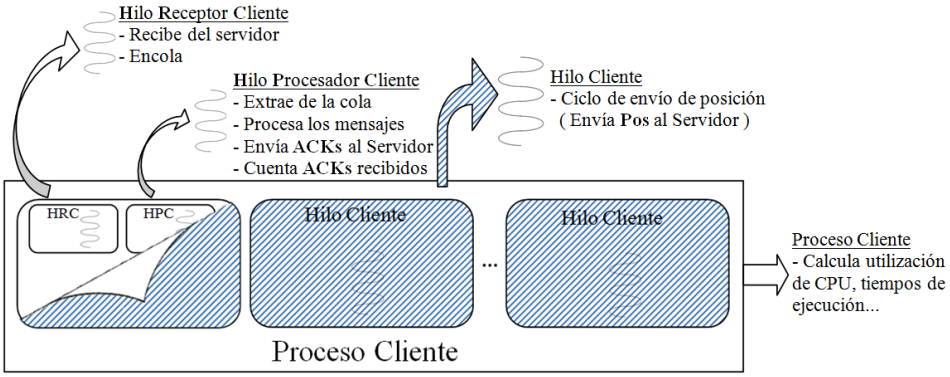

Figura 5.4: Esquema general del proceso cliente del simulador CAR

Cada hilo cliente simula el comportamiento de un teléfono móvil en el sistema CAR, y sigue el ciclo descrito en la Figura 5.2. El primer paso de este ciclo es obtener la nueva posición de la cámara, determinado por la frecuencia de actuación del cliente (parámetro del sistema[5]). Antes de enviar esta nueva posición se comprueba que se han recibido todos los mensajes ACK de la anterior posición o que se ha alcanzado el tiempo límite. Tras enviar la posición y mientras se espera a que un nuevo ciclo de actuación se complete, el hilo cliente se dedica a contar los reconocimientos (ACKs) que se han recibido, y a responder con sus propios mensajes ACK a posiciones de los vecinos. La cantidad de veces que se repite este ciclo es un parámetro de la simulación, y los hilos cliente llevan la cuenta del mismo para saber cuándo deben terminar. Cuando los procesos cliente completan todas las iteraciones envían un mensaje especial a su hilo servidor y finalizan. El proceso cliente lleva la cuenta de cuántos hilos cliente tiene activos y cuando este número llega a cero finaliza. Por su parte, el hilo servidor cuenta cuántos clientes de su WG han acabado y cuando lo hacen todos él también finaliza. El proceso servidor lleva la cuenta de cuántos hilos servidor tiene activos, y cuando ese valor llega a cero escribe los valores de todos los tiempos

[5]El valor elegido para esa frecuencia será equivalente al obtenido en el capítulo de caracterización de los teléfonos móviles (Capítulo 4)

de respuesta calculados en un fichero y finaliza. Antes de terminar la simulación, un “script” calcula el tiempo de respuesta medio y su desviación típica, calcula el consumo medio de CPU en el computador servidor y en los computadores cliente y el tiempo total de la simulación, almacenando el resultado final en un fichero para su posterior análisis.

5.3. Metodología de evaluación

En este apartado se describe la metodología empleada para evaluar el sistema CAR de tipo generalista desarrollado en esta tesis.

Al igual que en los sistemas DVE, la metodología que se utilizará para validar nuestro sistema se basa en la evaluación de prestaciones en un sistema real. Las principales medidas de prestaciones que se va a considerar son la productividad y la latencia, entendida aquí como el tiempo que transcurre desde que un cliente manda una posición al servidor hasta que recibe el último ACK de sus vecinos. Ambos son parámetros típicos de la evaluación de redes de interconexión de distintos sistemas distribuidos, en general, y de DVEs en particular (Duato et al., 1997).

Para poder realizar las pruebas, se ha implementado un sistema CAR instrumentalizado compuesto por el servidor y cliente comentados en el apartado de diseño del simulador (5.2) empleando *C++* como lenguaje de programación. El código fuente está accesible en la siguiente dirección:

https://disco.uv.es/bauset/Simulador/

5.3.1. Parámetros de las pruebas

Para evaluar el sistema se ha planteado un modelo de evaluación de prestaciones atendiendo a la variación de una serie de parámetros del sistema. Esta variación de los parámetros conseguirá toda una gama de niveles de carga en el sistema. Los parámetros que utilizarán son:

- **La cantidad de posiciones enviadas por cada cliente**. Puesto que se realizan pruebas con centenares de clientes y se promedian los resultados obtenidos en cada una de ellas, se considera que 100 posiciones por cliente es suficiente para obtener unos resultados representativos. Por lo tanto, los clientes realizarán 100 iteraciones o ciclos completos como el descrito en la Figura 5.2 antes de finalizar su ejecución.

- **La frecuencia de actuación de los clientes**. La caracterización de los clientes realizada en el Capítulo 4 mostró que el HTC Nexus One era el dispositivo más rápido, con un ciclo de actuación de 167.11 milisegundos, mientras que el Motorola Milestone era el dispositivo más lento, con un ciclo de actuación de 698.34 milisegundos. Se ha considerado estos dos valores como la cota superior e inferior en cuanto a ciclo de actuación, por lo que serán los utilizados como parámetro para representar clientes rápidos y lentos (respectivamente). De esta forma, si se quiere que el sistema esté lo más cargado posible, se utilizará un gran número de clientes tipo *Nexus One*, mientras que si se quiere que esté poco cargado, se emplearán clientes tipo *Motorola Milestone*. Al definir las cotas superior e inferior de actuación se valida todo el rango intermedio de resultados que se generarían al elegir cualquier otro dispositivo para la simulación, puesto que los valores obtenidos por este último estarán dentro del intervalo generado en el análisis de prestaciones de ambas cotas.

- **La cantidad de clientes dentro de un Grupo de Trabajo (WG)**. Anteriormente se comentó que cada cliente está asignado a un grupo de trabajo que incluye a los vecinos a los que tiene que informar de su nueva posición y de los que tiene que recibir un mensaje ACK por cada posición que envíe. Por lo tanto, a medida que se aumente el tamaño del WG las comunicaciones realizadas por los clientes del mismo se incrementarán sustancialmente. Para analizar la escalabilidad del sistema se han realizado pruebas con un rango de tamaños de grupo de trabajo entre 5 y 25 (diferencia del cuadrado) para ver cómo afecta el volumen de intercambio de información entre clientes al rendimiento total del sistema. Consecuentemente, los valores utilizados son: 5, 10, 20 y 25.

- **La cantidad de clientes en el sistema**. Se ha optado por un rango de clientes que va desde 100 hasta 1000 unidades. Se parte de la base de que las aplicaciones CAR actuales nunca tienen a más de un centenar de clientes

actuando a la vez. Por ejemplo, un sistema CAR típico puede ser los alumnos de una clase, o los turistas viendo un cuadro, como el sistema mostrado en (Bruns et al., 2007). En estos casos, el tamaño no suele pasar de unas decenas. Por ello, llegar hasta el millar de clientes excede claramente del peor caso que se puede encontrar en este tipo de aplicaciones.

5.3.2. Características técnicas de los equipos y de la red local

Para realizar la evaluación de prestaciones se ha empleado dos tipos diferentes de computador. El primero con mejores prestaciones computacionales para el servidor, y otros más básicos, de propósito general, para los clientes. El computador donde se ejecutará el servidor incorpora un procesador Intel Core 2 Duo E8400 con una CPU de 3.00 GHz y 4 Gbytes de memoria RAM, ejecutando una distribución Ubuntu de Linux con el núcleo de sistema operativo 3.0.0-14-genericx86_64. Por su parte, los ordenadores destinados a ejecutar los clientes incorporan un procesador Intel Dual-Core E6600 de 3.06 GHz y 2 Gbytes de memoria RAM, que ejecutan una distribución OpenSuse de Linux con el núcleo 3.7.10-1.16-desktopx86_64.

La red que conecta todos los equipos es una red de área local IEEE 802.3u de 100 Mbps que pertenece a la Universitat de València (UV). Nuestras pruebas compartirán red con el resto de usuarios habituales de la UV, por lo que no se tiene control preciso del tráfico de red durante el tiempo de realización de las pruebas. No obstante, los valores aquí mostrados son valores promedio de 30 pruebas realizadas a diferentes horas del día, con lo que se elimina en gran manera el efecto de cualquier pico de utilización de la misma.

5.4. Resultados de las simulaciones

En este apartado se muestran los resultados de la evaluación realizada. Estos resultados han sido obtenidos al variar cada uno de los parámetros de la simulación, para ver cómo afecta cada uno de ellos al rendimiento del sistema. Para ello se ha medido el tiempo de respuesta (en milisegundos) como el tiempo

que transcurre desde que un cliente manda una posición hasta que recibe el último de los mensajes ACK provenientes de sus vecinos. Se medirá la desviación estándar de ese tiempo de respuesta (también en milisegundos), para evaluar la variación de los valores respecto a la media obtenida. También se medirá el porcentaje de utilización medio de la CPU del computador que ejecuta el servidor, para evaluar el consumo de recursos del sistema al ejecutar la simulación. Tal y como se ha comentado anteriormente, los valores mostrados son promedios de 30 pruebas, por lo que representan más fielmente el comportamiento del sistema.

En el capítulo de caracterización de los dispositivos CAR basados en telefonía móvil (Capítulo 4), se analizaron cuatro dispositivos. De esos cuatro dispositivos se ha seleccionado el que más ha tardado en realizar las pruebas y el que menos, puesto que así se tiene una cota inferior y superior de la carga que se puede esperar en el sistema, respectivamente. Tal y como se indicó en la Sección 4.3, el dispositivo más rápido resultó ser el teléfono móvil Google Nexus One, con una frecuencia de actuación de 167.11 ms., mientras que el dispositivo más lento resultó ser el Motorola Milestone, con un ciclo de 698.34 ms. Se hará referencia a ellos como *One* y *Milestone*, respectivamente.

	One			**Milestone**		
WG 5	**RT**	**Dev**	**CPU**	**RT**	**Dev**	**CPU**
100	62,37	22,68	9,9	64,92	23,49	8
200	63,77	22,21	15	64,51	23,32	9,1
300	66,71	22,66	22	67,47	23,14	26,3
400	68,68	22,5	32,7	70,84	23,81	65
500	71,04	23,56	45	71,89	24,53	65
600	71,5	24,18	48,6	70,23	23,91	21,8
700	72,37	25,01	59	74,24	24,44	44
800	72,85	26,01	68	70,34	23,81	28
900	75,01	28,98	79,2	71,36	24,4	30,7
1000	147,33	101,71	85	70,77	24,16	34

Tabla 5.1: Rendimiento del sistema para un grupo de trabajo de 5 vecinos

En la Tabla 5.1 se muestra los resultados de la prueba para un WG de 5 vecinos por cliente cuando todos los clientes del sistema son uno de estos dos modelos de teléfono móvil. La primera fila de la columna de la izquierda, con etiqueta “WG”, indica el tamaño del grupo de trabajo, y los siguientes valores de esa columna indican la cantidad de población que interviene en las pruebas (de 100 a 1000 clientes). A continuación, se muestran dos grupos de tres columnas cada uno, que indican los resultados del teléfono móvil más rápido (HTC Nexus One, etiquetado como “One”) y el teléfono más lento (Motorola Milestone, etiquetado como “Milestone”). La primera columna de cada grupo muestra el tiempo de respuesta medio, medido en milisegundos y la segunda columna representa su desviación típica, también en milisegundos. La tercera columna muestra el porcentaje de tasa de utilización de CPU en la máquina que ejecuta el servidor.

La Tabla 5.1 muestra un comportamiento del sistema muy similar al ejecutar clientes de los dos tipos de móvil. Si se compara el tiempo de respuesta (columna RT[6]) en ambos dispositivos, se observan unos valores muy similares para cualquier tamaño de población, a excepción de la prueba con 1000 clientes, donde el Nexus One duplica el tiempo de respuesta con respecto a las pruebas con 900 clientes (de 75.01 ms. a 147.33 ms.), mientras que el Motorola Milestone sigue prácticamente constante. Un comportamiento similar se produce con la desviación estándar de ese tiempo de respuesta (columna Dev[7]), donde los valores permanecen prácticamente constantes en ambos dispositivos (en torno a los 23-25 ms.) hasta que la población llega a 1000 clientes, momento en el que la desviación estándar del Nexus One se quintuplica. En cuanto al parámetro de utilización de CPU del sistema (columna CPU), este valor en el Nexus One crece más rápidamente que el Motorola Milestone conforme se aumenta el tamaño de la población, lo que le lleva a alcanzar una ocupación del 85 % al simular la población de 1000 clientes, mientras que el Motorola Milestone sigue estando en un 34 %. Por lo tanto, se puede decir que el sistema está poco cargado cuando se utiliza un grupo de trabajo de 5 vecinos y que se queda cerca de la saturación al simular una población de 1000 clientes pero sólo para el caso del Nexus One.

La Tabla 5.2 muestra los resultados para la misma configuración del sistema, pero esta vez con un tamaño de grupo de trabajo de 10 vecinos.

[6] Del inglés “Response Time”.
[7] Del inglés “Deviation”.

	One			Milestone		
WG 10	**RT**	**Dev**	**CPU**	**RT**	**Dev**	**CPU**
100	78,23	30,41	16	77,03	27,63	9,9
200	74,02	25,06	25,2	78,16	26,07	26
300	79,45	24,57	37	83,6	28,61	25
400	81,02	24,05	59,6	82,54	23,76	23,8
500	93,61	28,56	68,7	88,27	26,76	31,3
600	147,73	100,48	83,8	87,66	27,45	33
700	195,47	76,24	83,2	88,45	28,58	33,7
800	237,71	91,09	85,9	94,43	34,05	62
900	270,25	104,97	84,2	84,8	25,7	69,3
1000	300,96	132,04	84,2	87,72	27,56	46,5

Tabla 5.2: Rendimiento del sistema para un grupo de trabajo de 10 vecinos

La Tabla 5.2 muestra que el sistema tiene un comportamiento diferente en función del cliente simulado. Si se compara las columnas del tiempo de respuesta en ambos dispositivos, se observa que el RT se mantiene casi constante para el Milestone conforme aumenta la población, yendo de 77.03 ms. a 87.72 ms. Sin embargo, en el caso del Nexus One, cuando la población alcanza los 600 clientes el RT se dispara (de 93.61 ms. a 147.43 ms.). Un comportamiento similar ocurre con la desviación estándar, donde para el Motorola Milestone permanece prácticamente constante mientras que en el One se dispara al alcanzar los 600 clientes. A partir de esa población, el tiempo de respuesta y su desviación típica sufren un crecimiento más acentuado para el Nexus One. Por su parte, el consumo de CPU va creciendo paulatinamente conforme se aumenta el tamaño de la población; de nuevo este crecimiento es más exagerado para el Nexus One, alcanzando un valor máximo de 85.9 %. Mientras que el Milestone no supera el 70 % de ocupación. Esto se debe a que el Motorola Milestone, al tener un ciclo de actuación más lento, no sobrecarga tanto al sistema, puesto que sus actualizaciones de posición ocurren con menos frecuencia que las del Nexus One. De hecho, en la configuración con clientes Nexus One se puede afirmar que el punto de saturación se alcanza al llegar a los 600 clientes, donde tiempo de respuesta y desviación se disparan, por lo que se considerará ese porcentaje (83.8 %)

como el límite de ocupación del servidor. Más adelante se comentará por qué ese porcentaje no llega al 100 % de utilización de la CPU y sin embargo parece alcanzarse la saturación del servidor.

Desde el punto de vista de los límites aceptables para el tiempo de respuesta, se suele establecer el valor de 250 ms. como el tiempo máximo para considerar las aplicaciones como interactivas (Henderson and Bhatti, 2003). Por lo tanto, se puede decir que el servidor es capaz de ejecutar 800 clientes tipo Nexus One con comportamiento interactivo (de media) y más de 1000 clientes tipo Motorola Milestone.

En la Tabla 5.3, se muestra la simulación para una configuración de sistema con tamaño de grupo de trabajo de 20 vecinos. De nuevo, la tabla muestra como ambos dispositivos van aumentando su tiempo de respuesta, desviación y consumo de CPU a medida que se incrementa la población del sistema. No obstante, el sistema con clientes tipo Nexus One sufre un crecimiento más pronunciado a partir de una población de 400 clientes, donde el porcentaje de utilización de la CPU llega al 84.2 %, momento en el que el tiempo de respuesta y su desviación estándar se disparan, alcanzando un valor máximo de 473.99 ms. y 286.28 ms., respectivamente. El Motorola Milestone sigue sin llegar a saturar el sistema y obtiene un consumo de CPU máximo de tan solo el 74.7 %. En este caso, para un grupo de trabajo de 20 vecinos, el Nexus One puede simular hasta 400 clientes con un comportamiento interactivo (su RT es de 195.92 ms. frente al umbral de 250 ms.), mientras que el Motorola Milestone simula más de 1000 clientes puesto que, llegados a este tamaño de población, aún tiene un tiempo de respuesta medio de 106.7 ms.

Por último, en la Tabla 5.4, se evalúa el sistema con un tamaño de grupo de trabajo de 25 clientes. Análogamente a lo que ocurría para los anteriores grupos de trabajo, ambos dispositivos obtienen unos valores similares pero con el sistema un poco más cargado, de forma que el Motorola Milestone utiliza el 81 % de CPU durante la simulación, con un tiempo de respuesta de 131.5 ms. y una desviación estándar de 63.24 ms. para una población de 1000 clientes. Por su parte, los valores correspondientes a las tres columnas del Nexus One crecen uniformemente hasta la población de 300 clientes, momento en el que la ocupación del sistema llega al 86 % y su tiempo de respuesta y desviación

	One			Milestone		
WG 20	**RT**	**Dev**	**CPU**	**RT**	**Dev**	**CPU**
100	88,22	32,16	27,6	93,88	26,9	20,2
200	94,57	29,06	45,5	94,98	25,74	21,2
300	128,16	28,88	73,3	98,01	24,79	29,7
400	195,92	63,42	84,2	129,2	30,8	37,7
500	273,58	89,08	84	102	30,91	43
600	326,79	106,94	86,2	121,5	39,5	50,5
700	400,45	142,76	87	148,8	66,52	83,9
800	447,02	155,46	87	110,4	39,07	65,3
900	473,99	223,98	86	101,4	32,33	67,3
1000	467,04	286,28	86,9	106,7	37,8	74,7

Tabla 5.3: Rendimiento del sistema para un grupo de trabajo de 20 vecinos en ambos teléfonos móviles

se disparan. En este último caso se consiguen respuestas interactivas con una población de 300 clientes para el Nexus One y 1000 clientes para el Motorola Milestone.

A lo largo de este estudio se ha observado que la saturación del sistema se alcanza al llegar a una ocupación de CPU cercana al 85 %, momento en el que el tiempo de respuesta y su desviación típica se disparan. Cabe destacar que sistemas distribuidos similares, como los sistemas DVE suelen alcanzar la saturación cuando el servidor llega a una ocupación cercana al 95-98 % (Morillo et al., 2005). La diferencia entre el 85 % y ese 98 % se puede explicar por la arquitectura de memoria compartida de los procesadores actuales (como el utilizado para ejecutar el servidor), que difieren de los utilizados en (Morillo et al., 2005), donde son procesadores multi-núcleo. Nuestra hipótesis es que la sincronización de las llamadas del núcleo[8], junto con la sincronización interna de los hilos dentro de los procesos de la aplicación, impiden al sistema CAR explotar el poder computacional de todos los núcleos del procesador a la vez, alcanzando la saturación con valores más bajos de carga. Con el fin de demostrar que este razonamiento

[8]Se le suele llamar por su nombre en inglés: Kernel

	One			Milestone		
WG 25	**RT**	**Dev**	**CPU**	**RT**	**Dev**	**CPU**
100	96,03	25,91	60,4	96,19	25,12	25
200	103,19	38,71	83	96,3	24,83	47,1
300	167,58	50,19	86,1	120,5	26,36	43,6
400	250,53	77,68	85	125,2	39,86	53,3
500	357,13	126,19	92	144,9	64,99	56
600	496,62	218,79	94,1	135,7	52,91	57
700	480,87	156,16	85,3	112,3	36,48	60,7
800	531,93	210,02	86	184,2	107,4	77,8
900	505,89	300,11	89	126,6	61,98	78
1000	524,96	372,62	87,1	131,5	63,24	81

Tabla 5.4: Rendimiento del sistema para un grupo de trabajo de 25 vecinos en ambos teléfonos móviles

es correcto, se repetirá las pruebas del dispositivo móvil Nexus One con grupos de trabajo de 25 clientes, pero ejecutando el servidor sobre un computador de 8 núcleos. Dicho computador incorpora un procesador Intel(R) Core(TM) i7 960 CPU, con una potencia de 3.20 GHz y 8 Gbytes de RAM, ejecutando una versión de sistema operativo Linux SuSE 2.6.37.6-0.5-desktop. La Tabla 5.5 muestra los resultados obtenidos para un grupo de trabajo de 25 clientes basados en el ciclo de simulación del dispositivo cliente Nexus One, que supone la carga máxima del sistema (el dispositivo más rápido con el grupo de trabajo más grande).

La Tabla 5.5 muestra que el tiempo medio de respuesta se incrementa significativamente cuando la población llega a los 500 clientes para el caso del computador de 8 núcleos, pasando de 98.65 ms. a 228.98 ms., mientras que su desviación estándar se duplica al llegar a esa población. Por lo que el punto de saturación del sistema para este servidor se alcanza con una población de 500 clientes, y este punto de saturación se alcanza con un porcentaje de ocupación de la CPU de 43.5 % (cuarta columna desde la izquierda en la Tabla 5.5). Sin embargo, el resultado obtenido con el servidor de dos núcleos muestra que la saturación se alcanza para una población de 300 clientes, momento en el que

	8 núcleos (i7)			2 núcleos (Core Duo)		
WG 25	**RT**	**Dev**	**CPU**	**RT**	**Dev**	**CPU**
100	90.91	26.91	24.1	96.03	25.91	60.4
300	98.65	20.63	33.6	167.58	50.19	86.1
500	228.98	42.36	43.5	357.13	126.19	92,0
700	357.95	75.79	45.3	480.87	156.16	85.3
900	478.95	103.34	48.3	505.89	300.11	89.0

Tabla 5.5: Rendimiento del sistema para computadores con diferente cantidad de núcleos

el tiempo de respuesta medio salta de 96.03 ms. a 167.58 ms. Para este tiempo de respuesta, el porcentaje de utilización de la CPU es del 86.1 %. Como cabría esperar, la productividad se incrementa al utilizar un procesador más potente para ejecutar el servidor, pero el punto de saturación del sistema CAR no es un valor fijo del porcentaje total de utilización de la CPU, y dicho punto depende del número de procesadores disponibles.

5.5. Interfaz Gráfico del sistema CAR

En esta sección se presenta una propuesta de interfaz gráfico para el sistema CAR desarrollado. Para su realización se ha hecho uso de la herramienta Unity 3D, que es un motor de juegos multiplataforma que proporciona su propio entorno de desarrollo (Unity, 2014). De entre las opciones disponibles, se ha seleccionado un modelo 3D compuesto por un almacén lleno de estanterías que contienen cajas de diferentes tamaños y formas. En él los clientes moverán simultáneamente las piezas existentes en el stock. Los grupos de trabajo estarán formados por los clientes que estén visualizando las mismas estanterías, por lo que la cantidad de las mismas dependerá de la cantidad de clientes simulados. La Figura 5.5 muestra las diferentes vistas que se tiene del interfaz gráfico propuesto. En la imagen de la izquierda se observa una vista cenital del almacén que hace de interfaz del sistema CAR propuesto. En la imagen superior derecha se

muestra la visión que tiene cada cliente del almacén y un ejemplo de interacción con el mismo, en el que el cliente selecciona una caja en particular. Por último, en la imagen inferior derecha se observa cómo percibe el servidor la interacción del cliente de la imagen anterior con el almacén, resaltando la caja seleccionada por el cliente.

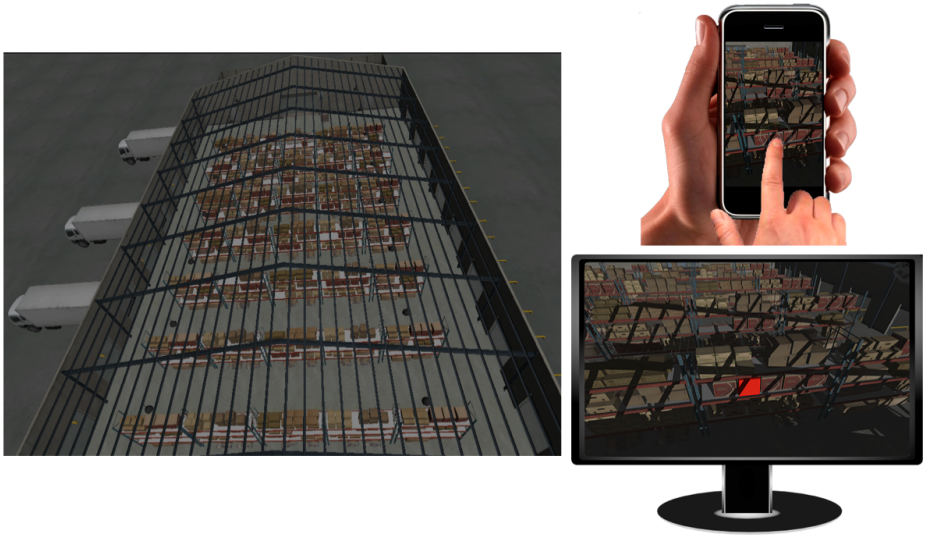

Figura 5.5: Diferentes vistas del modelo propuesto. Imagen izquierda: Vista cenital del almacén que hace de interfaz gráfico del sistema CAR. Imagen superior derecha: Vista del cliente y ejemplo de interacción con el sistema CAR. Imagen inferior derecha: Ejemplo de cómo percibe el servidor la interacción del cliente de la imagen superior derecha.

Se han realizado pruebas con este interfaz funcionando en la misma máquina que ejecuta el servidor del sistema CAR, y se ha visto que la adición de este interfaz gráfico no repercute en las prestaciones del servidor, que obtiene resultados similares a los mostrados en las secciones anteriores del presente capítulo, por lo que no se repetirán aquí. No obstante, sirven para validar el uso de este interfaz cuya ejecución no impacta en las prestaciones del sistema CAR.

5.6. Conclusiones

En este capítulo se ha desarrollado el simulador del sistema CAR y se han analizado sus prestaciones cuando los dispositivos cliente son los teléfonos móviles simulados. En las pruebas aquí analizadas se ha simulado un sistema CAR donde todos los clientes son el dispositivo móvil más lento y también un sistema CAR donde todos los clientes son el dispositvo móvil más rápido, puesto que representan la cota inferior y superior de demanda de prestaciones al sistema, respectivamente. Se ha visto que el rendimiento del sistema depende significativamente del tipo de dispositivo elegido para la simulación. De forma que, si se elige el teléfono más rápido (Nexus One), el sistema se carga más rápidamente puesto que demanda muchos más recursos. Para este móvil se consigue simular más de 1000 clientes en tiempo interactivo con un grupo de trabajo reducido (5 clientes), y hasta 300 clientes para un grupo de trabajo grande (25 clientes). Por otro lado, si se escoge el teléfono más lento (Motorola Milestone), el sistema se carga menos, puesto que su frecuencia de actuación es cuatro veces más lenta que la del Nexus One (698.34 ms. del Motorola Milestone frente a 167.11 ms. del Nexus One). Con este cliente, se pueden simular más de 1000 clientes dentro de unas cotas más que razonables de interactividad con cualquiera de los grupos de trabajo propuestos.

También se ha visto que el punto de saturación del sistema depende del porcentaje total de utilización de CPU en el computador que actúa como servidor, aunque no es un valor fijo, sino que depende del número de núcleos procesador del computador que actúa como servidor.

Por último, se ha presentado un interfaz gráfico basado en un almacén 3D, realizado con Unity 3D y se ha evaluado que su ejecución no tiene impacto en las prestaciones del servidor.

6

Mejora de Prestaciones del sistema CAR

Tras realizar el simulador del sistema CAR y caracterizar el funcionamiento de dicho sistema, se va a intentar mejorar su rendimiento, de forma que se pueda simular más clientes respetando el tiempo de respuesta del mismo. Por lo tanto, en este capítulo se van a realizar tantas mejoras como sea posible a nuestro sistema inicial para conseguir optimizar sus prestaciones. Todas las mejoras se evaluarán sobre la misma configuración inicial, es decir, se ejecutarán en las mismas máquinas descritas en la Sección 5.3.2, de forma que la comparación sea lo más justa posible.

6.1. Servidor *activo* frente a servidor *pasivo*

Tal y como se comentó en el capítulo centrado en el simulador generalista de sistemas CAR (Capítulo 5), el intercambio de mensajes tras la actualización de una posición cambia sustancialmente en función del grupo de trabajo empleado, de forma que para un grupo de trabajo de 5 clientes, una única posición equivale a 13 mensajes: 1 mensaje del cliente origen al servidor enviando la nueva posición, 4 mensajes del servidor a los clientes destino retransmitiendo dicha posición, 4 mensajes de los clientes destino al servidor contestando con un mensaje de reconocimiento (ACK) y 4 mensajes del servidor al cliente origen retransmitiendo ese mensaje de reconocimiento. Mientras que en un grupo de trabajo de 25 clientes, equivale a 73 mensajes. Por lo tanto, para tratar de mitigar este incremento de mensajes se modificará el servidor para provocar que su comportamiento sea más *activo* en el intercambio de mensajes. En lugar de dedicarse a retransmitir mensajes entre clientes, y que sea el cliente origen el que lleve la cuenta de los mensajes de reconocimiento recibidos de sus vecinos, el mismo servidor es el que llevará esta cuenta y enviará un único mensaje ACK al cliente origen cuando se hayan recibido todos los mensajes ACK de sus vecinos. De esta forma, cuando el cliente mande un mensaje en un grupo de trabajo de 5 vecinos, se intercambiarán sólo 9 mensajes: 1 mensaje del cliente origen al servidor con la nueva posición, 4 mensajes del servidor a los clientes destino retransmitiendo dicha posición, 4 mensajes de los clientes destino al servidor contestando con un mensaje de reconocimiento y 1 único mensaje del servidor al cliente origen avisando de que ya ha recibido todos los mensajes de reconocimiento de los clientes vecinos. Análogamente, si el grupo de trabajo es de 25 vecinos, se intercambiarán 49 mensajes. Con este nuevo servidor se reducirán más de un 30 % los mensajes intercambiados en cada actualización de posición. Se ha evaluado el sistema CAR con las mismas configuraciones que las descritas en el Capítulo 5, pero ahora con el servidor activo en lugar del servidor meramente pasivo. En la Tabla 6.1 se observa cómo afecta esta modificación al rendimiento del sistema para un grupo de trabajo de 10 clientes; las tablas para grupos de trabajo de 5, 20 y 25 se encuentran en el Apéndice I.

La Tabla 6.1 muestra las prestaciones del sistema al simular grupos de trabajo de 10 clientes. En esta tabla se puede apreciar, en la primera fila de la primera

	One			Milestone		
WG 10	**RT**	**Dev**	**CPU**	**RT**	**Dev**	**CPU**
100	70,27	23,61	10,9	72,89	28,02	7,1
200	68,85	23,14	21,8	69,35	23,92	13,0
300	73,27	18,07	36,4	73,67	18,43	17,8
400	74,81	17,48	44,0	78,13	23,07	18,8
500	79,71	21,34	57,0	75,9	19,04	26,0
600	83,58	25,59	70,0	80,35	26,04	31,7
700	83,85	27,61	82,3	81,73	26,06	31,0
800	179,75	89,37	84,2	78,69	23,82	40,8
900	222,94	102,97	84,0	76,32	20,47	43,9
1000	255,67	116,19	85,2	78,29	21,99	45,1

Tabla 6.1: Rendimiento del sistema para un grupo de trabajo de 10 vecinos en ambos teléfonos móviles con el servidor *activo*

columna, el grupo de trabajo simulado (con la etiqueta WG), en este caso 10, seguido de las poblaciones empleadas en cada prueba, que van desde 100 hasta 1000 clientes. A continuación se muestran dos grupos de tres columnas cada uno, que representan los resultados para el teléfono móvil Nexus One y para el Motorola Milestone. La primera columna de cada grupo es el tiempo de respuesta (RT) medido en milisegundos, entendido aquí como el tiempo que transcurre desde que el cliente manda una posición hasta que recibe el ACK del servidor, indicando así que llegó el último mensaje ACK de los vecinos. La segunda columna muestra la desviación estándar (Dev) de ese tiempo de respuesta, también en milisegundos. Y, por último, se muestra el porcentaje de CPU utilizado por el ordenador que simula al servidor durante la prueba (CPU).

Como se puede ver en la Tabla 6.1, el sistema sigue comportamientos distintos en función del tipo de dispositivo móvil simulado. Mientras que el Motorola Milestone mantiene prácticamente constantes el RT y su desviación estándar, el Nexus One va incrementándolas conforme aumenta la población simulada, hasta llegar al 84 % de consumo de CPU, donde el sistema vuelve a entrar en saturación y se disparan los valores medios de tiempo de respuesta (RT). No obstante,

si se compara la Tabla 6.1 con la Tabla 5.2, se puede observar que se ha conseguido una mejora de rendimiento en ambos dispositivos al reducir la cantidad de mensajes intercambiados entre el cliente origen y el servidor. Ahora el Nexus One es capaz de simular más de 900 clientes con comportamiento interactivo, frente a los 800 clientes que podía simular con el servidor *pasivo*. Y, por su parte, el Motorola Milestone sigue siendo capaz de simular más de 1000 clientes, rebajando su tiempo de respuesta máximo de los 94,43 ms. anteriores (para 800 clientes) a los 81,73 ms. (para 700 clientes).

En la Figura 6.1 se puede ver, a modo de resumen, la evaluación del tiempo de respuesta en ambas implementaciones del servidor, para cada uno de los grupos de trabajo y teléfonos móviles simulados. Cada una de las subgráficas mostradas en la Figura 6.1 tiene como título el tamaño del grupo de trabajo que se ha utilizado en la simulacióin. El eje de abscisas representa la cantidad de clientes simulados y el eje de las ordenadas muestra el tiempo de respuesta, medido en milisegundos. Por su parte, los valores con la leyenda “a-One” corresponden a la simulación de servidor *activo* para el teléfono Nexus One, por lo que la línea con la leyenda “p-One” corresponde al servidor *pasivo* para el mismo teléfono móvil. Análogamente, “a-Miles” y “p-Miles” corresponden a las pruebas con el teléfono móvil Motorola Milestone para el servidor *activo* y *pasivo*, respectivamente.

En todas las gráficas incluidas en la Figura 6.1 se observa que el tiempo de respuesta obtenido con la versión pasiva del servidor está por encima del obtenido con la versión activa. Asimismo, se puede observar que el tiempo de respuesta del Motorola Milestone permanece prácticamente constante durante toda la simulación, mientras que el Nexus One crece conforme se aumenta el tamaño de la población. Las gráficas también muestran a qué altura queda el umbral de tiempo interactivo, representado en las gráficas por una línea roja horizontal a la altura de los 250 ms. Por lo tanto, con la mejora del servidor *activo* se puede simular en tiempo interactivo a más de 1000 clientes para cualquier grupo de trabajo que utilice el teléfono móvil Motorola Milestone y sobre el Nexus One se pueden simular más de 1000 para un grupo de trabajo de 5, más de 900 clientes para un grupo de trabajo de 10, 600 clientes para el grupo de trabajo de 20 y 500 para el grupo de trabajo de 25 vecinos.

Como conclusión de esta mejora se tiene que el tiempo de respuesta del sis-

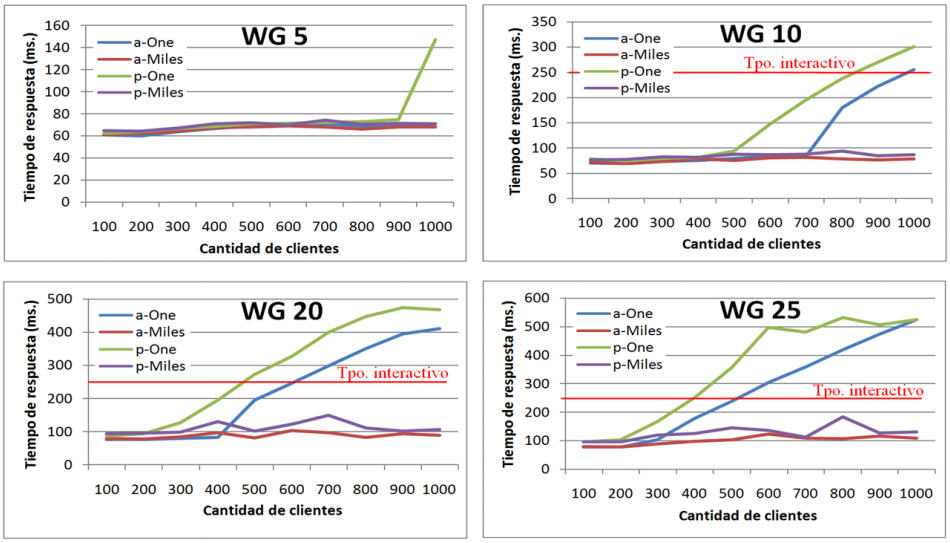

Figura 6.1: Gráficas del tiempo de respuesta de los servidores Activo y Pasivo, para todos los grupos de trabajo y ambos teléfonos móviles

tema se reduce ligeramente al reducir el tráfico intercambiado entre el cliente origen y el servidor. Esto se consigue a expensas de un incremento de complejidad del funcionamiento del servidor, pues tiene que contabilizar los mensajes de reconocimiento que recibe cada cliente de sus vecinos y avisarle cuando hayan llegado todos. Tarea que realizará para cada una de las posiciones que se envían. Con el fin de encontrar una justificación a tan discreta mejoría del tiempo de respuesta medio obtenido, se añadirá un nuevo parámetro a las pruebas: *el tiempo de respuesta en el servidor*, entendido como el tiempo que transcurre desde que el servidor envía un mensaje a cualquier cliente hasta que obtiene su contestación. La Tabla 6.2 muestra los mismos resultados obtenidos para el teléfono móvil Nexus One con el servidor *activo* (Tabla 6.1), pero añadiendo ahora el correspondiente *tiempo de respuesta en el servidor* (SRT) y su valor máximo (MaxSRT), ambos medidos en milisegudos. El resto de tablas con estos nuevos parámetros, tanto para el servidor pasivo como para el activo sobre ambos teléfonos móviles se encuentran en el Apéndice I.

La Tabla 6.2 muestra que el tiempo de respuesta medio en el servidor (columna SRT) no supera los 50 milisegundos para el teléfono móvil Nexus One,

	Nexus One activo				
WG 10	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	70,27	23,61	10,9	19,97	22,38
200	68,85	23,14	21,8	19,69	26,93
300	73,27	18,07	36,4	26,87	31,57
400	74,81	17,48	44	27,48	32,04
500	79,71	21,34	57	26,65	33,94
600	83,58	25,59	70	28,43	36,14
700	83,85	27,61	82,3	28,45	39,14
800	179,75	89,37	84,2	36,25	42,22
900	222,94	102,97	84	42,39	49,52
1000	255,67	116,19	85,2	49,14	58,82

Tabla 6.2: Tiempo de respuesta en el servidor (*activo*) para un grupo de trabajo de 10 vecinos para el teléfono móvil Nexus One

independientemente del tamaño de la población simulada, mientras que el valor máximo de ese tiempo de respuesta del servidor (columna MaxSRT) no supera los 60 milisegundos. Estos valores indican que el cuello de botella del sistema (la razón por la que el tiempo de respuesta es tan elevado, tanto para el servidor *activo* como el *pasivo*) se encuentra en el servidor, puesto que se ha visto que el tiempo que emplean los mensajes fuera del servidor es reducido. Por ejemplo, este valor es de tan solo 60 ms. frente a los 255,67 ms. de tiempo de respuesta obtenidos para la población de 1000 clientes. No obstante, aunque el modelo de servidor *activo* desempeña más tareas que el modelo de servidor *pasivo*, se reduce el intercambio de información con cada cliente, consiguiendo mejorar las prestaciones.

6.2. Servidor con función “select”

La implementación básica del simulador maneja una gran cantidad de ficheros para gestionar los hilos de cada proceso: los sockets abiertos para cada clien-

te, los ficheros de estadísticas, etc.. En esta nueva versión se intentará reducir esa cantidad de ficheros abiertos mediante la utilización de la función “select” de los sockets BSD (Jones, 2003). Dicha función permite tratar diversos sockets con un único proceso, de forma que si se tiene varios de ellos abiertos y se dispone de sus descriptores, se le pueden pasar a la función “select” y será ella la que genere avisos cada vez que reciba información, momento en el que facilitará los datos transmitidos y el cliente del que proceden. De esta forma, se cambiará la versión anterior del *hilo servidor*, que creaba un *hilo receptor servidor* y un *hilo procesador servidor* por cada cliente y se generará un único par de estos últimos hilos para todo el grupo de trabajo. Con ello se tendrá un único hilo gestionando todos los sockets de los clientes de ese grupo de trabajo, que apilará los mensajes en una única cola de recepción. Como es lógico, sólo se necesita un hilo que procese esa cola de recepción. Por lo tanto, si se tienen 1000 clientes con grupos de trabajo de 5 vecinos, en la versión inicial se necesitaría 200 hilos servidor, que generarían 5 hilos de recepción cada uno (uno para cada vecino del grupo de trabajo), que almacenarían datos en 5 colas distintas, y 5 hilos de procesamiento, que leerían de esas colas. Esta configuración requiere un total de 2000 hilos (1000 hilos de recepción y 1000 hilos de procesamiento). En la versión actual, los 200 hilos servidor sólo crearán un hilo de recepción cada uno (uno para todo el grupo de trabajo), con su cola de mensajes, y un hilo de procesamiento para tratar esa cola. Por lo que se ahorran al menos 1600 hilos (800 hilos de recepción y 800 de procesamiento) y 800 colas de mensajes. El resto de arquitectura del simulador permanece idéntica a la implementación inicial. Se puede observar la modificación gráficamente en el esquema de la Figura 6.2, en la que se muestra que ahora cada hilo servidor sólo genera un hilo receptor (SRT) y un hilo procesador (SPT).

La evaluación de prestaciones con esta nueva implementación se muestra a continuación. Se indican los resultados para el teléfono móvil Nexus One, puesto que es el que más carga genera al sistema. Se han seleccionado aquí los resultados para las configuraciones con grupos de trabajo de 5 y 25 clientes, para ver los resultados con el sistema poco congestionado y muy congestionado, respectivamente. El resto de tablas se encuentran en el Apéndice I.

Con el fin de mostrar resultados comparativos entre distintas implementaciones, la Tabla 6.3 muestra las pruebas sobre el teléfono móvil Nexus One con un

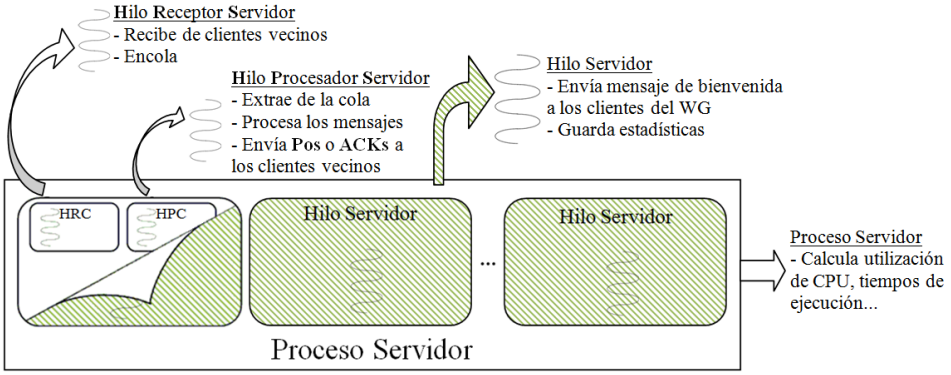

Figura 6.2: Esquema general del proceso servidor del simulador CAR versión “select”

grupo de trabajo de 5 clientes, tanto en la implementación descrita en el Capítulo 5 (denominada aquí con la etiqueta “TCP Base”), como en la implementación basada en la función “select” (etiquetada como “Select”). Como en anteriores tablas, la primera columna muestra el tamaño de las poblaciones simuladas, que van desde 100 hasta 1000 clientes. Después se tiene dos grupos de 5 columnas cada uno, que se corresponden con la implementación “base” a la izquierda, y la implementación con “select” a la derecha. Cada uno de estos grupos de columnas incluye el tiempo de respuesta (RT) medido en milisegundos, su desviación estándard también en milisegundos, el porcentaje de utilización de CPU del ordenador que ejecuta el servidor (CPU), el tiempo de respuesta en el servidor (SRT) en milisegundos, y su valor máximo (MaxSRT) también en milisegundos.

En la Tabla 6.3 se observa que cuando el sistema recrea un escenario con pocos clientes por grupo de trabajo (WG de 5 clientes), el tiempo de respuesta medio se mejora ligeramente (columna RT) respecto de la implementación “base” (“TCP base”). Lo mismo ocurre con la desviación estándar de ese tiempo de respuesta (columna Dev). Si se compara valor a valor ambas columnas (RT y Dev) para ambas implementaciones, se observa una mejoría de unos pocos milisegundos en cada uno de los tamaños de población simulados, a excepción de la población de 1000 clientes (última fila). Esto se explica analizando la tasa de utilización de CPU (columna CPU) de la implementación “select”, que con un valor máximo del 67 % de utilización de CPU para el tamaño de población de 1000 clientes (última fila) no alcanza el *umbral de saturación* comentado anteriormente

ONE	TCP base					Select				
WG 5	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	62,37	22,68	9,9	19,36	20,9	65,77	21,56	8	20,44	24,96
200	63,77	22,21	15	20,12	22,43	67,12	22,71	11,2	21,5	23,54
300	66,71	22,66	22	25,15	28,28	67,52	22,6	19,8	23,67	26,62
400	68,68	22,5	32,7	27,14	29,56	67,64	22,88	28	23,21	27,28
500	71,04	23,56	45	27,14	30,9	69,12	23,23	31	25,31	29,34
600	71,5	24,18	48,6	26,58	31,95	69,14	23	39,6	25,28	28,74
700	72,37	25,01	59	27,17	35,42	69,37	23,45	47	24,53	30,72
800	72,85	26,01	68	27,87	34,54	75,75	26,63	54,5	26,96	35,16
900	75,01	28,98	79,2	28,61	35,89	70,24	24,81	59,6	24,02	31,87
1000	147,33	101,71	85	43,95	48,66	71,05	27,53	67	22,64	35,04

Tabla 6.3: Rendimiento del sistema para el Nexus One con un grupo de trabajo de 5 vecinos. Implementación “base” y mejora “select”

(cercano al 84 % de utilización de CPU). Por su parte, el tiempo de respuesta en el servidor (columna SRT) se mantiene prácticamente constante para la implementación “select”, en torno a los 24 ms. y su valor máximo (columna MaxSRT) no supera los 35.16 ms (obtenidos para el tamaño de población de 800 clientes).

La Tabla 6.4 muestra la diferencia entre la implementación “base” y la implementación “select” cuando el sistema recrea un escenario con grupos de trabajo de 25 clientes. En esta tabla se puede observar que cuando el sistema soporta mayor carga los beneficios de la implementación “select” se hacen más evidentes. Los tiempos de respuesta (columna RT) obtenidos en la implementación “select” son ligeramente inferiores a los obtenidos en la implementación “base”, consiguiendo simular más de 400 clientes en tiempo interactivo (con RT menor de 250 ms.), frente a los 300 clientes que se podía simular con la versión “base”. Por su parte, la desviación estándar de ese tiempo de respuesta (columna Dev) reduce sustancialmente su crecimiento, alcanzando un valor máximo de 133.33 ms. para el tamaño de población de 1000 clientes, frente a los 372.62ms. obtenidos para la misma población en la implementación “base”. La utilización de CPU (columna CPU) presenta un rango dinámico más grande para la implementación “select”, de forma que pese a tener un valor inicial más bajo (23.2 % frente al 60.4 % anterior para el primer tamaño de población) llega a valores de utilización superiores al 90 % para las cuatro últimas poblaciones (correspondientes a las cuatro últimas filas de la columna CPU). Por último, tanto el tiempo de respuesta

en el servidor (columna SRT) como su valor máximo (columna MaxSRT), sufren un crecimiento más pronunciado que en la implementación “base”, llegando a duplicar el valor de ésta para el tamaño de población de 1000 clientes (última fila de ambas columnas). El valor de SRT pasa de 82.39 ms. en la implementación “base” a 166.79 ms. en la implementación “select” y el parámetro MaxSRT se incrementa desde los 96.23 ms. a los 175.88 ms.

ONE	TCP base					Select				
WG 25	RT	Dev	CPU	SRT	MaxSRT	RT	Dev	CPU	SRT	MaxSRT
100	96,03	25,91	60,4	24,26	26,28	90,8	24,7	23,2	19,35	20,96
200	103,19	38,71	83	30,98	34,53	89,95	21,13	47	33,4	37,51
300	167,58	50,19	86,1	40,24	43,24	123,95	32,36	72	54,7	59,24
400	250,53	77,68	85	49,89	54,97	209,2	35,88	87,2	85,55	88,44
500	357,13	126,19	92	74,41	78,1	268,17	44,44	86	112,07	118,26
600	496,62	218,79	94,1	96,25	104,29	331,41	50,55	87	143,62	150,76
700	480,87	156,16	85,3	93,42	100,18	383,96	70,6	93,1	151,56	164,57
800	531,93	210,02	86	97,47	107,15	454,57	125,21	90,1	151,56	178,35
900	505,89	300,11	89	93,45	102,97	491,59	106,18	91	147,56	154,16
1000	524,96	372,62	87,1	82,39	96,23	566,44	133,33	93,1	166,79	175,88

Tabla 6.4: Rendimiento del sistema para el dispositivo móvil Nexus One con un grupo de trabajo de 25 vecinos. Implementación “base” y mejora “select”

La variación en el comportamiento del sistema observado en la Tabla 6.4 se justifica como sigue. En primer lugar, el hecho de que la desviación estándar del tiempo de respuesta se reduzca se debe a que en esta nueva configuración hay muchos menos hilos que compiten por la CPU, de forma que las esperas que tienen que efectuar cuando están preparados para ejecutarse, pero no se les concede un núcleo, son más cortas y menos frecuentes, por lo que los tiempos de respuesta obtenidos son más similares, lo que reduce su desviación estándar. En segundo lugar, el hecho de que se consiga una utilización de la CPU superior al 90 % en las cuatro últimas poblaciones tiene una justificación similar: al existir menos hilos de la aplicación ejecutándose, menos de ellos compiten por un núcleo y, por tanto, las ejecuciones son más seguidas y se aprovecha mejor la capacidad de cómputo del ordenador que ejecuta el servidor. Por último, el tiempo de respuesta en el servidor aumenta debido a que los clientes ahora contestan todos a un mismo hilo, con la función “select”, que permite gestionar todos sus sockets juntos. De forma que, en cierto modo, se serializan. Así, aunque todos los clientes contesten a la vez, sus mensajes irán apareciendo de uno

en uno y cuando el último llegue realmente a la cola del grupo habrá sufrido una penalización. También se puede justificar con el hecho de que el servidor ahora sólo tiene una cola de procesamiento, a la que llegan los mensajes desde la función “select”, de forma que tiene que esperar más tiempo para recibir el mensaje de respuesta (este tiempo se mide desde que el servidor envía un mensaje a un cliente hasta que éste le responde). Antes, cada cliente tenía una cola, por lo que se sabía rápidamente que se le había contestado. Ahora las respuestas de todos los clientes se apilan en la misma cola, por lo que cuesta un poco más de tiempo que se procese y cuente como contestado. Esto último sugiere que se podría sacar más rendimiento del sistema, si se tuviesen más hilos vaciando la cola de cada hilo servidor (actualmente sólo hay uno). Por lo tanto, en la Figura 6.3 se verá cómo afecta el número de hilos de procesamiento al rendimiento del sistema.

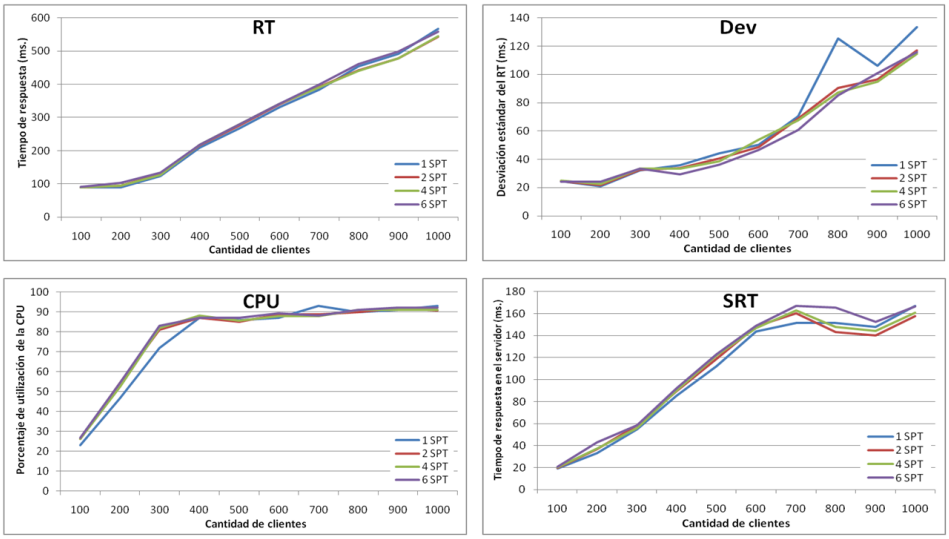

Figura 6.3: Gráficas comparativas de cada parámetro de la simulación del Nexus One con WG de 25 clientes sobre la implementación “select”, variando el número de SPT: 1, 2, 4 y 6

En la Figura 6.3 se puede observar cuatro gráficas correspondientes al tiempo de respuesta (RT), su desviación estándar (Dev), la utilización de CPU (CPU) y el tiempo de respuesta en el servidor (SRT) al variar la cantidad de hilos que

gestionan la cola de cada grupo de trabajo (SPT[1]) con los siguientes valores: 1, 2, 4 y 6. Todos ellos medidos en milisegundos, menos la gráfica "CPU" que representa el porcentaje de utilización de CPU. Las pruebas siguen siendo sobre el teléfono móvil Nexus One, para un grupo de trabajo de 25 clientes. No se van a analizar en detalle los resultados presentados aquí porque la primera serie de cada gráfica (con la leyenda "1 SPT") se corresponde a la Tabla 6.4 analizada en el párrafo anterior. Como se puede ver en cada una de las gráficas, el hecho de tener más hilos desencolando información de la cola del grupo de trabajo no supone ninguna mejora en el rendimiento del sistema. Por lo tanto, se seguirá utilizando un único hilo SPT, dejando así el número de hilos del simulador al mínimo. Las pruebas realizadas con los restantes grupos de trabajo (5, 10 y 20) han dado un resultado idéntico, por lo que no se muestran aquí, aunque se pueden ver en el Apéndice I.

Como conclusión de esta mejora se tiene que la utilización de la función "select" de los sockets BSD mejora el rendimiento del sistema, puesto que permite reducir la cantidad de hilos necesarios para ejecutar el servidor CAR. Se ha demostrado que cuantos menos hilos se utilice mejor, puesto que menos de ellos intentan obtener la CPU, el sistema pierde menos tiempo cambiando de contexto y puede sacar mejor partido a sus núcleos, consiguiendo elevar la utilización de los mismos. También se ha visto que con un único hilo gestionando la cola de los grupos de trabajo es más que suficiente independientemente de la cantidad de clientes que participen en el grupo. Por lo tanto, cualquier otra optimización del simulador tendrá que enfocarse en otro aspecto del mismo.

6.3. Servidor UDP

En esta implementación se va a evaluar la efectividad de las comunicaciones no orientadas a conexión cuando se ejecutan en entornos distribuidos como los sistemas CAR. Nuestra primera implementación utilizaba sockets TCP para comunicar a unos dispositivos con otros. A ese tipo de comunicación se le denomina *orientada a conexión* debido al protocolo TCP/IP que utilizan y con el que se asegura que todos los paquetes enviados llegarán a su destino, incluso reenviará

[1] Del inglés "Server Processor Thread"

paquetes que se hayan perdido. No obstante, para conseguir esta seguridad, se necesita que los paquetes lleven algún mecanismo de control y unas cabeceras que añaden una sobrecarga a la información que se quiere enviar. Por todo esto se considera interesante evaluar los sockets UDP que, al ser no orientados a conexión, reducen al mínimo la información intercambiada. Aunque este tipo de protocolo puede perder paquetes, se ha demostrado que puede obtener mejores prestaciones para tráfico de red de poco tamaño y mucha frecuencia (Duato et al., 1997).

La implementación UDP es muy similar a la implementación “select” (TCP) vista en la sección 6.2. La única diferencia es que esta nueva implementación utilizará sockets UDP. Dado que esta implementación puede perder paquetes (que no se reenviarán), se ha añadido un contador de paquetes perdidos para cada uno de los mensajes que se envían de servidor a clientes y viceversa. Los mensajes que tienen como origen los clientes incluyen el envío de nuevas posiciones, mensajes de respuesta a esas posiciones (ACK), mensajes con el tiempo de respuesta del sistema para cada ciclo de actuación de los clientes y el envío de la posición final (que indica que el cliente origen ha terminado su simulación). Por su parte, los mensajes que tienen como origen el servidor se corresponden con el reenvío de las nuevas posiciones de los clientes a sus vecinos del grupo de trabajo, el envío de un mensaje de control al cliente origen (con el que el servidor avisa de que ya ha reenviado a todos los vecinos su nueva posición), el reenvío de los mensajes ACK de cada cliente vecino y el reenvío de la posición final de cada cliente. Como se sabe la cantidad de mensajes que tienen que llegar de cada tipo, se obtiene el porcentaje de pérdida de cada uno de ellos a partir de los mensajes realmente recibidos.

A continuación, en la Tabla 6.5, se muestran los resultados obtenidos con esta nueva implementación para grupos de trabajo de 5 clientes y se comparan con los resultados obtenidos en la implementación “Select-TCP", puesto que era la que mejores resultados obtenía hasta el momento. Dicha tabla se encuentra dividida en tres subtablas. En la primera de ellas se encuentran los datos obtenidos con la implementación “selectz que sirven como referencia. En las dos siguientes se encuentran los datos obtenidos en la implementación UDP, que se han tenido que segmentar por problemas de espacio. Como en las anteriores tablas, la primera columna de cada subtabla indica el dispositivo utilizado (en este caso el

One	Select				
WG 5	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	65,77	21,56	8,00	20,44	24,96
200	67,12	22,71	11,20	21,50	23,54
300	67,52	22,60	19,80	23,67	26,62
400	67,64	22,88	28,00	23,21	27,28
500	69,12	23,23	31,00	25,31	29,34
600	69,14	23,00	39,60	25,28	28,74
700	69,37	23,45	47,00	24,53	30,72
800	75,75	26,63	54,50	26,96	35,16
900	70,24	24,81	59,60	24,02	31,87
1000	71,05	27,53	67,00	22,64	35,04
	UDP				
	RT	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	1,53	0,17	14,50	0,70	0,72
200	1,67	0,87	26,00	0,74	0,81
300	1,83	1,21	33,30	0,79	1,07
400	1,99	1,32	39,80	0,83	1,12
500	2,09	2,33	46,70	0,84	1,46
600	2,11	2,43	54,70	0,87	1,55
700	2,25	3,58	60,70	0,91	1,62
800	2,97	4,27	73,40	1,15	2,33
900	3,83	6,69	80,20	1,38	2,95
1000	35,37	21,36	90,70	14,86	30,17

	% pérdidas servidor				% pérdidas cliente			
	POS	**FIN**	**ACK**	**TRC**	**POS**	**FIN**	**ACK**	**CTR**
100	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
200	0,00	0,00	0,00	0,01	0,00	0,00	0,00	0,00
300	0,00	0,00	0,00	0,05	0,00	0,00	0,00	0,00
400	0,00	0,00	0,01	0,05	0,00	0,00	0,01	0,00
500	0,00	0,00	0,01	0,06	0,00	0,00	0,01	0,00
600	0,00	0,00	0,02	0,08	0,00	0,00	0,02	0,00
700	0,00	0,00	0,08	0,17	0,00	0,00	0,08	0,00
800	0,00	0,00	0,09	0,21	0,00	0,00	0,09	0,00
900	0,00	0,00	0,21	0,29	0,00	0,00	0,21	0,00
1000	0,00	0,00	0,45	12,89	0,00	0,00	0,45	0,00

Tabla 6.5: Rendimiento del sistema para el Nexus One con un grupo de trabajo de 5 clientes. Comparando las implementaciones “select” y UDP

teléfono móvil Nexus One), la cantidad de clientes por grupo de trabajo (WG 5, es decir, 5 clientes por grupo de trabajo) y los tamaños de población, que oscilan de 100 a 1000 clientes. Dentro de cada subtabla se encuentran los resultados de la simulación para el tiempo de respuesta (RT) y su desviación estándar (Dev) (medidos en milisegundos). A continuación se muestra la tasa de utilización de

CPU en la máquina que ejecuta el servidor (columna CPU) (mostrado en porcentaje). Por último, se indica el tiempo de respuesta en el servidor (SRT), con su valor máximo (MaxSRT), también medidos en milisegundos. Como la nueva implementación puede perder paquetes se ha añadido una nueva subtabla (la subtabla inferior) con una columna para cada uno de los diferentes mensajes que se espera recibir en el servidor, con las que se expresa el porcentaje de pérdida de dichos mensajes, que incluyen: mensajes de posición (etiquetados como POS), mensajes de reconocimiento de posición recibida (etiquetados como ACK), mensajes de posición final (etiquetados como FIN) y mensajes que expresan el tiempo de respuesta del sistema para cada ciclo de actuación de los clientes (etiquetados como TRC[2]). Análogamente, se han añadido cuatro columnas que representan el porcentaje de pérdida de paquetes producidas en el cliente, correspondientes a mensajes de posición (etiquetados como POS), reconocimiento (ACK), posición final (FIN) y el mensaje de control que envía el servidor al cliente origen tras cada retransmisión de su posición a los vecinos del grupo de trabajo (etiquetada como CTR).

Como se observa en la Tabla 6.5, al utilizar la implementación UDP el tiempo de respuesta (columna RT) se reduce un factor de 60 para los primeros tamaños de población simulados (de 100 a 400 clientes) y disminuye hasta llegar a un factor de 2 con respecto a la implementación “select"(para la población de 1000 clientes). La desviación estándard del tiempo de respuesta se reduce un factor de 20 (para los cuatro primeros tamaños de población) y va aproximándose al valor obtenido con la implementación “select", en la población de 1000 clientes (obteniendo 21.36 ms frente a los 27.53 ms de los que se partía). De forma similar, los valores del tiempo de respuesta en el servidor (columna SRT) y su valor máximo (columna MaxSRT) se reducen un factor de 20 con respecto a los obtenidos por la implementación “select"; aunque esta mejora se reduce para la última población, llegando prácticamente a igualar el resultado obtenido en la anterior implementación. Por su parte, la utilización de CPU (columna CPU) de UDP se incrementa con respecto a la implementación “select", empezando con un valor del 14.5 % de utilización de CPU para 100 clientes, frente al 8 % de utilización de CPU obtenido en la implementación “select", y llegando a un valor de 90.7 % para la implementación de 1000 clientes (frente al 67 % obtenido anteriormente). Este incremento en la utilización de la CPU se debe al hecho de que UDP

[2]TRC se corresponde con las siglas de Tiempo de Respuesta del Cliente.

no es orientado a conexión; al no utilizar TCP, no se retransmiten los paquetes, ni se asegura su orden de llegada, por lo que el sistema funciona más fluido al estar menos tiempo parado en la gestión y sincronización de esos paquetes. Por lo que respecta a los paquetes perdidos, el sistema no pierde ninguno para la población de 100 clientes. No obstante, en la simulación con 200 clientes ya se tiene una pérdida del 0,01 % de los mensajes de tiempo de respuesta por ciclo (columna TRC), esta pérdida se produce cuando el sistema no está saturado. Se ha descartado que dichas pérdidas se produzcan por un tamaño de búffer de lectura/escritura inadecuado de los sockets, puesto que se ha ampliado su tamaño al máximo que permite el sistema operativo. Con este aumento se redujo la cantidad de paquetes perdidos sustancialmente, aunque sigue habiendo pérdidas. También se ha calculado el ancho de banda necesario para realizar las pruebas (se puede ver en la Tabla 6.7), por lo que se sabe que para esa prueba sólo se necesita un ancho de banda de 10.55 Mbps y disponemos de una red de 100 Mbps. Por lo tanto, las pérdidas deben producirse porque los tamaños de búffer de los routers que hay entre los ordenadores que alojan al servidor y los clientes no son lo suficientemente grandes para soportar los picos de tráfico que se pueden dar a lo largo de las pruebas. Cuando esto ocurre, el sistema se comporta como si no hubiese búffer y esos paquetes se pierden; como UDP no reenvía paquetes, no habrá forma de recuperarlos.

Por otro lado, el hecho de que se pierdan paquetes no implica necesariamente que la aplicación vea afectadas sus prestaciones. En este sentido, se observa que el sistema es capaz de terminar las simulaciones sin perder mensajes de posición y posición final (columnas POS y FIN de servidor y cliente). Lo mismo ocurre con los mensajes de control que envía el servidor al cliente (columna CTR), que consiguen llegar sin pérdidas para todas las poblaciones simuladas. No ocurre lo mismo con los mensajes de reconocimiento (columna ACK, tanto de servidor como de cliente) que, pese a empezar sin pérdidas, a partir de las población de 400 clientes se empiezan a perder mensajes (0.01 %), aunque dicha pérdida no sobrepase el 0.45 % (obtenido para la población de 1000 clientes). Puesto que se pierde la misma cantidad de mensajes de reconocimiento en el servidor y en el cliente, se puede concluir que dichos mensajes se pierden en su camino al servidor, quien obviamente no es capaz de reenviarlos al cliente que mandó la posición (cliente origen). Por este mismo motivo, la cantidad de mensajes TRC (de tiempo de respuesta por ciclo de actuación) perdidos se incrementa,

puesto que el cliente origen está esperando recibir el último mensaje ACK de sus vecinos para calcular el tiempo de respuesta que luego mandará al servidor. Al igual que pasaba con los mensajes ACK, la pérdida de mensajes TRC se incrementa con la cantidad de clientes simulados, llegando a un 12.89 % de mensajes perdidos para la población de 1000 clientes. Cabe mencionar que para nuestra aplicación los mensajes más importantes son los de la posición de los clientes, puesto que son los que indican la posición de la pantalla del cliente que necesita ser *aumentada*. A estos mensajes les sigue en importancia los mensajes de reconocimiento (ACK), puesto que indican si los clientes vecinos son conscientes de nuestra posición. El resto de mensajes son importantes para llevar un control del sistema pero no repercuten en el funcionamiento de la aplicación.

Con un grupo de trabajo reducido (cinco clientes por grupo) las prestaciones de la implementación basada en sockets UDP son mucho mejores que cualquiera de las anteriores. En la Tabla 6.6 se muestra el comportamiento del sistema al simular grupos de trabajo de 25 clientes. Como en los apartados anteriores, el resto de tablas (para WG de 10 y 20, y las pruebas con el Motorola Milestone) se pueden encontrar en el Apéndice I.

En la Tabla 6.6 se observa que, cuando el nivel de carga del sistema es elevado (por usar grupos de trabajo de 25 clientes), la nueva implementación consigue tiempos de respuesta por debajo del tope interactivo de 250 ms. hasta llegar a la población de 800 clientes, momento en el que la columna RT supera este valor con 253.46 ms. . A partir de ese momento su valor crece de forma más pronunciada conforme aumenta el tamaño de población, llegando a los 450.06 ms. para la población de 1000 clientes, frente a los 566.4 ms. que se obtenía en la implementación “select". La desviación estándar del tiempo de respuesta (columna Dev) sigue siendo ligeramente inferior a la de la implementación “select"hasta llegar a la población de 300 clientes, momento en el que su valor sobrepasa al obtenido en la implementación anterior. A partir de esa población su valor sigue creciendo y se encuentra por encima del obtenido en la implementación anterior para todas las poblaciones simuladas, llegando a un valor de 393.40 ms. para la población de 1000 clientes (frente a los 133.3 ms. de la implementación “select"). El porcentaje de utilización de CPU (columna CPU) de la implementación UDP obtiene valores por encima de los conseguidos en la anterior implementación para todas las poblaciones simuladas, partiendo de un valor de 39 % para la población

One	Select				
WG 25	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	90,8	24,7	23,2	19,35	20,96
200	89,95	21,13	47	33,4	37,51
300	124	32,36	72	54,7	59,24
400	209,2	35,88	87,2	85,55	88,44
500	268,2	44,44	86	112,1	118,26
600	331,4	50,55	87	143,6	150,76
700	384	70,6	93,1	151,6	164,57
800	454,6	125,2	90,1	151,6	178,35
900	491,6	106,2	91	147,6	154,16
1000	566,4	133,3	93,1	166,8	175,88
	UDP				
	RT	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	4,89	4,98	39,00	1,68	2,20
200	6,30	5,67	67,40	2,17	2,99
300	82,24	49,48	92,10	81,03	87,91
400	83,39	49,64	92,90	87,47	94,09
500	85,11	54,72	93,00	88,75	96,35
600	94,39	56,41	93,40	89,06	95,98
700	115,77	87,94	93,80	89,96	99,13
800	253,46	178,95	94,80	97,88	108,75
900	328,79	224,95	96,00	115,24	150,08
1000	450,06	393,40	99,00	145,67	169,05

	% pérdidas servidor				% pérdidas clientes			
	POS	**FIN**	**ACK**	**TRC**	**POS**	**FIN**	**ACK**	**CTR**
100	0,00	0,00	0,00	0,01	0,00	0,00	0,00	0,00
200	0,00	0,00	0,07	0,14	0,00	0,00	0,07	0,00
300	0,00	0,00	6,75	90,99	0,00	0,00	6,75	0,00
400	0,00	0,00	23,05	97,41	0,00	0,00	23,05	0,00
500	0,00	0,00	33,68	98,19	0,00	0,00	33,68	0,00
600	0,00	0,00	41,25	98,56	0,00	0,00	41,25	0,00
700	0,00	0,00	47,32	99,01	0,00	0,00	47,32	0,00
800	0,00	0,00	51,89	99,12	0,00	0,00	51,89	0,00
900	0,00	0,00	55,67	99,16	2,53	77,10	57,37	3,21
1000	0,00	0,00	56,95	99,51	7,61	95,18	62,85	8,50

Tabla 6.6: Rendimiento del sistema para el Nexus One con un grupo de trabajo de 25 clientes. Comparando las implementaciones “select” y UDP

de 100 clientes y pasando del 92 % desde la población de 300 clientes. Por su parte, el tiempo de respuesta en el servidor y su valor máximo (columnas SRT y MaxSRT, respectivamente), se reducen un orden de magnitud hasta la población de 200 cliente y pasan de los 80 ms. a partir de la población de 300 clientes, momento que coincide con el porcentaje de utilización de la CPU superior al 92 %.

El valor de ambos parámetros continúa con un crecimiento uniforme hasta llegar a la población de 1000 clientes, en la que obtienen un valor de 145.67 ms. para el SRT y de 169.05 ms. para el MaxSRT. El hecho de que el tiempo de respuesta y su desviación estándar se disparen para los últimos tamaños de población se justifica observando la cantidad de paquetes perdidos a partir de la población de 300 clientes; al llegar a los 300 clientes simulados la columna TRC muestra unas pérdidas del 90.99 % de los mensajes, lo que significa que el cálculo del tiempo de respuesta y su desviación estándar se hace sólo con un 9 % del total de mensajes que se esperaban. Por lo tanto, el valor obtenido en ambas columnas (RT y Dev) no se puede considerar significativo. De hecho, para una población de 1000 clientes, el cálculo del ambas columnas se hace con menos de un 0.49 % de los valores que debería, puesto que se pierden un 99.51 % de los mensajes TRC. Así mismo, se observa que la pérdida de mensajes de posición y posición final (columnas POS y FIN) se mantiene a cero para el servidor pero no ocurre lo mismo con los clientes. Al llegar a los dos últimos tamaños de población, se empiezan a producir pérdidas de paquetes de posición y posición final en el cliente, lo cual indica que al servidor le han llegado dichos mensajes pero no es capaz de retransmitirlos todos. Este efecto vuelve a constatar que el servidor está al límite de sus posibilidades. Por último, los mensajes de control que envía el servidor a los clientes (columna CTR) también comienzan a sufrir pérdidas a partir de la población de 900 clientes, llegando a perder el 8.50 % de los mensajes para la población de 1000 clientes.

El hecho de perder tal cantidad de mensajes pone en tela de juicio la viabilidad de la mejora del simulador basada en UDP. Para intentar justificar su uso se va a analizar el ancho de banda necesario para realizar las pruebas. Si se tiene en cuenta la cantidad de paquetes que se mandan en cada ciclo de actuación de los clientes y que cada uno de ellos dura 167.11 ms. (puesto que el teléfono móvil simulado es el Nexus One), el ancho de banda necesario para que se ejecute cada prueba es el que aparece en la Tabla 6.7.

En la Tabla 6.7 se observa el ancho de banda necesario para ejecutar las pruebas con el teléfono móvil Nexus One, medido en Mbps. La primera columna representa la cantidad de clientes simulados en cada prueba, que van desde los 100 clientes hasta los 1000 clientes. Las cuatro siguientes columnas representan el ancho de banda necesario para simular las pruebas con los grupos de trabajo

One	WG 5	WG 10	WG 20	WG 25
100	5,277	10,553	21,106	26,383
200	10,553	21,106	42,213	52,766
300	15,830	31,660	63,319	79,149
400	21,106	42,213	84,426	105,532
500	26,383	52,766	105,532	131,915
600	31,660	63,319	126,639	158,299
700	36,936	73,873	147,745	184,682
800	42,213	84,426	168,852	211,065
900	47,490	94,979	189,958	237,448
1000	52,766	105,532	211,065	263,831

Tabla 6.7: Ancho de banda (Mbps) necesario para ejecutar las pruebas con el teléfono móvil Nexus One con el sistema ARToolKit

de 5, 10, 20 y 25 clientes. Como se comentó en la sección 5.3.2, la red utilizada para las pruebas es una red de área local IEEE 802.3u de 100 Mbps que pertenece a la Universitat de València (UV). Por lo tanto, todos los valores de la Tabla 6.7 que pasen de los 100 Mbps tendrán una gran cantidad de pérdidas, puesto que representan pruebas que necesitarán más ancho de banda del que se le puede proporcionar. Por ejemplo, en la Tabla 6.6 se vio que a partir de la población de 400 clientes la pérdida de paquetes se disparaba, obligando a hacer el cálculo del tiempo de respuesta y su desviación con tan solo un 4 % de los datos que se esperaban. Si se observa la celda correspondiente a esa prueba en la Tabla 6.7, se ve que el ancho de banda necesario para ejecutar esa prueba es de 105.532 Mbps, por lo que la pérdida de mensajes a partir de esa prueba está justificada.

Cabe comentar que el hecho de perder un paquete de información en el simulador se percibe como una actualización de posición que no llega a su destino o un mensaje de reconocimiento perdido. Al no llegar la posición, el dibujado de la información 3D que se muestra en la pantalla para producir el efecto de realidad *aumentada* tendrá dos alternativas: o mostrarse en la posición anterior, o no mostrarse. En nuestro simulador se utiliza la primera opción porque la segunda deja de percibir la información en la pantalla. No obstante, incluso con la peor al-

ternativa, sólo se percibiría como un pequeño parpadeo de la información puesto que el ciclo de actuación se lleva a cabo varias veces por segundo.

Como conclusión de esta mejora se tiene que la implementación UDP proporciona el mejor rendimiento obtenido hasta el momento por el simulador. Al prescindir de la orientación a conexión se consigue que los mensajes se intercambien con mayor fluidez. Tanto es así que se llega a saturar la red de 100 Mbps que se utiliza para hacer las pruebas, convirtiéndose en el factor limitante del sistema. Mientras no se llegue a demandar un ancho de banda excesivo para la red, el rendimiento del sistema es muy superior al de las demás implementaciones. Por lo tanto, si se dispone de una red superior, se puede simular más de 1000 clientes respetando el tiempo interactivo (250 ms.) para cualquiera de los grupos de trabajo considerados. Esta mejora se consigue a expensas de perder un porcentaje de los mensajes enviados. Sin embargo, el efecto de esta cantidad de paquetes perdidos se percibirá como un pequeño parpadeo en un número reducido de clientes del sistema CAR. Por lo tanto, estos resultados validan la propuesta basada en UDP como la implementación más eficiente para diseñar sistemas CAR de gran escala sobre dispositivos móviles.

6.4. Configuración del sistema operativo para adaptarlo al simulador

La forma en que el sistema operativo realiza los cambios de contexto y todo el mecanismo de planificación del mismo puede influir negativamente en las prestaciones de las aplicaciones que se ejecutan sobre él. Teniendo en cuenta que el sistema CAR que se ha desarrollado está basado en hilos, y que el servidor del mismo se va a ejecutar sobre un computador con sistema operativo Linux, se pretende adaptar el sistema de planificación de procesos del sistema operativo para que no perjudique a los hilos del servidor y se puedan ejecutar de la forma más óptima posible.

El planificador es la parte del núcleo del sistema operativo que decide qué proceso ejecutable será el siguiente en ejecutarse. El planificador de Linux im-

plementa una serie de políticas de planificación que determinan cuándo y por cuánto tiempo se ejecuta un hilo en un núcleo de CPU particular. Dichas políticas se dividen en dos categorías principales: Políticas de tiempo real y políticas normales. Las políticas en tiempo real se utilizan para tareas críticas que deben completarse sin interrupciones y no se ejecutan por largos periodos de tiempo (Planificación CPU, 2014). Por su parte, las políticas de planificación normal están pensadas para administrar grandes cantidades de hilos y generlamente proporcionan mejor rendimiento que las políticas de tiempo real, debido a que permiten al planificador ejecutar hilos de forma más eficiente. Como el sistema aquí desarrollado utiliza una gran cantidad de hilos durante su ejecución, hará uso de las políticas normales. Existen tres políticas de planificación normal: SCHED_OTHER, SCHED_BATCH y SCHED_IDLE. Sin embargo, las dos últimas son para trabajos de una prioridad muy baja, por lo que se analizará SCHED_OTHER, que es la política de planificación predeterminada de Linux.

SCHED_OTHER utiliza el planificador de reparto justo (CFS[3]) para proporcionar periodos de acceso justo a todos los hilos del sistema. Para ello, CFS mantiene el balance (equidad) en el tiempo de procesador que se asigna a los hilos. Cuando un hilo está *fuera de balance*, se le asigna una cierta cantidad de tiempo de ejecución en el procesador, denominada “Virtual Runtime”. Para llevar el control de cuánto “Virtual Runtime” lleva ejecutado cada hilo, el CFS mantiene un árbol de tipo *rojo-negro* ordenado por tiempo. De esta forma, el proceso con menor “Virtual Runtime” es el más próximo a ser ejecutado (CFS, 2014).

Los parámetros que proporciona el CFS para poder configurar el planificador son: “sched_min_granula rity_ns”, “sched_latency_ns” y “sched_wakeup_granularity_ns”. Todos ellos se pueden modificar desde la línea de comandos y se encuentran en el directorio “/proc/sys/kernel/”. A continuación se describe cada uno de ellos.

sched_min_granularity_ns decide el tiempo mínimo que una tarea podrá ejecutarse en la CPU. Por defecto está puesta a 4ms., por lo que cualquier tarea se ejecutará como mínimo 4 ms. antes de marcarla como lista para expulsar de la CPU. Se probará valores que vayan desde los 1.5 ms. hasta los 6 ms., con incrementos de 0.5 ms..

[3]Del inglés “Completely Fair scheduler”

sched_latency_ns decide el periodo del planificador, es decir, el periodo en el que todas las tareas listas para ejecutar se ejecutarán al menos una vez. Su valor por defecto es 20 ms. y se probarán valores desde 12ms. hasta 24 ms., con incrementos de 2 ms..

sched_wakeup_granularity_ns es la habilidad que tienen las tareas para despertarse y sustituir a la tarea en ejecución. Cuanto más pequeño es su valor, más fácil le resulta sustituir a la tarea en ejecución. El valor por defecto es 2 ms. y se probarán valores desde 1 ms. hasta 4 ms. con incrementos de 0.5 ms..

Tras realizar pruebas con todas las combinaciones posibles de los parámetros comentados anteriormente, el resultado obtenido en cada una de ellas es prácticamente idéntico. Por lo tanto, no se mostrará ninguna tabla aquí, puesto que no se puede realizar ninguna comparación entre ellas, al obtener siempre los mismos resultados.

La conclusión obtenida de este estudio es que no importa la combinación de parámetros elegida del CFS, puesto que no afectan al rendimiento del sistema. Por tanto, se mantendrán los valores que proporciona por defecto.

6.5. Simulador con teléfonos móviles sobre Vuforia

Como se comentó en la caracterización de clientes (Capítulo 4), mientras se realizaban las últimas mejoras del simulador apareció una plataforma y entorno de desarrollo de aplicaciones de realidad aumentada *sin marcas* llamado Vuforia, de la empresa Qualcomm (Qualcomm, 2012). Debido a su sencillez, esta plataforma se ha extendido con gran rapidez y ya se utiliza como un estándar de facto entre la comunidad de desarrolladores. Por lo tanto, se caracterizaron los teléfonos móviles con este nuevo sistema para analizar las diferencias de rendimiento. Los resultados de esa caracterización se encuentran el capítulo de caracterización (en la Sección 4.4), donde se vio que con este sistema la frecuencia de actuación de todos los teléfonos móviles se incrementaba. Por lo tanto, ahora se analizará cómo afecta ese incremento de velocidad al utilizar los teléfonos móviles en el simulador.

Se utilizará la implementación UDP para realizar las pruebas, puesto que se ha demostrado que es la que mejores resultados ofrece. En cuanto a los teléfonos móviles, se volverá a elegir el dispositivo más rápido y el más lento para realizar las pruebas, puesto que representan la cota superior e inferior de actuación. El teléfono más rápido vuelve a ser el Nexus One, pero con este sistema su frecuencia de actuación es de 56.68 ms. (frente a los 167.11 ms. anteriores). Mientras que el teléfono más lento pasa a ser el iPhone 3GS, con una frecuencia de actuación de 145.53 ms. (frente a los 398.00 ms. anteriores). Las pruebas seguirán siendo para poblaciones de 100 a 1000 clientes y los grupos de trabajo considerados serán 5, 10, 20 y 25.

La Tabla 6.8 muestra el rendimiento del sistema cuando todos los clientes son del tipo iPhone 3GS. En esta tabla se pueden apreciar cuatro subtablas que representan los cuatro grupos de trabajo contemplados (5, 10, 20 y 25 clientes). Cada subtabla contiene los cinco parámetros analizados en las pruebas anteriores: el tiempo de respuesta (columna etiquetada como RT) medido en milisegundos, su desviación estándar (Dev) medida también en milisegundos, la utilización de CPU medida en porcentaje (CPU), el tiempo de respuesta en el servidor (SRT). Acto seguido figuran dos grupos de cuatro columnas cada uno (que llamaremos central y derecho) que representan el porcentaje de paquetes de cada tipo que se pueden perder tanto en el servidor como en el cliente (recordemos que la implementación es UDP y, por tanto, puede perder paquetes). Los tres primeros parámetros de cada grupo son equivalentes y se corresponden con las posiciones enviadas (POS), las posiciones finales enviadas (FIN), los mensajes de reconocimiento (ACK). El último parámetro del grupo central, que se corresponde con el servidor, representa el porcentaje de pérdidas de los mensajes de tiempo de respuesta por ciclo (TRC), mientras que el último parámetro del grupo de la derecha representa el porcentaje de pérdidas de los mensajes de control que manda el servidor al cliente (CTR).

En la Tabla 6.8 se observa que el tiempo de respuesta para el grupo de trabajo de 5 clientes permanece prácticamente constante, en torno a 4 milisegundos, hasta llegar a la población de 900 clientes; momento en el que pasa a 9.5 ms. y de ahí a los 49.16 ms. que obtiene para 1000 clientes. Su desviación estándar se incrementa ligeramente conforme aumenta el tamaño de la población, desde 0.2 ms. (para 100 clientes) hasta 44.5 ms. (para la población de 1000 clientes); tam-

						% pérdidas servidor				% pérdidas clientes			
WG 5	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**	**POS**	**FIN**	**ACK**	**TRC**	**POS**	**FIN**	**ACK**	**CTR**
100	1,54	0,20	17,20	0,69	0,73	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
200	1,64	0,61	27,70	0,73	0,80	0,00	0,00	0,00	0,01	0,00	0,00	0,00	0,00
300	1,86	1,46	35,80	0,80	0,94	0,00	0,00	0,01	0,06	0,00	0,00	0,01	0,00
400	2,33	1,96	44,10	0,95	1,52	0,00	0,00	0,03	0,08	0,00	0,00	0,03	0,00
500	3,04	4,76	52,60	1,20	1,85	0,00	0,00	0,16	0,09	0,00	0,00	0,16	0,00
600	3,44	4,76	61,30	1,33	2,43	0,00	0,00	0,04	0,11	0,00	0,00	0,04	0,00
700	3,17	6,03	70,60	1,56	2,52	0,00	0,00	0,05	0,13	0,00	0,00	0,05	0,00
800	3,96	6,72	80,40	2,45	3,37	0,00	0,00	0,12	0,23	0,00	0,00	0,12	0,00
900	9,50	17,30	89,90	3,45	10,70	0,00	0,00	0,24	0,89	0,00	0,00	0,24	0,00
1000	49,16	44,50	93,00	25,83	48,65	0,00	0,00	0,87	39,58	0,00	0,00	0,87	0,00
WG 10	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**	**POS**	**FIN**	**ACK**	**TRC**	**POS**	**FIN**	**ACK**	**CTR**
100	1,98	0,49	25,50	0,86	0,94	0,00	0,00	0,00	0,01	0,00	0,00	0,00	0,00
200	2,47	1,17	41,00	0,97	1,23	0,00	0,00	0,01	0,05	0,00	0,00	0,01	0,00
300	7,32	8,70	57,00	2,58	3,27	0,00	0,00	0,07	0,15	0,00	0,00	0,07	0,00
400	7,95	9,75	70,50	3,15	3,94	0,00	0,00	0,10	0,24	0,00	0,00	0,10	0,00
500	9,77	10,55	87,50	3,45	8,15	0,00	0,00	0,16	0,30	0,00	0,00	0,16	0,00
600	51,97	29,42	94,00	61,60	66,86	0,00	0,00	3,97	86,46	0,00	0,00	3,97	0,00
700	53,25	45,43	94,00	75,93	84,42	0,00	0,00	13,81	96,11	0,00	0,00	13,81	0,00
800	53,77	48,44	94,00	81,65	91,37	0,00	0,00	22,75	97,73	0,00	0,00	22,75	0,00
900	57,56	58,85	94,30	84,98	103,74	0,00	0,00	29,42	97,32	0,00	0,00	29,42	0,00
1000	58,33	72,30	95,00	93,47	111,12	0,00	0,00	34,57	97,30	0,00	0,00	34,57	0,00
WG 20	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**	**POS**	**FIN**	**ACK**	**TRC**	**POS**	**FIN**	**ACK**	**CTR**
100	3,13	1,46	36,80	1,14	1,34	0,00	0,00	0,01	0,06	0,00	0,00	0,01	0,00
200	4,84	3,10	65,30	1,75	2,82	0,00	0,00	0,05	0,13	0,00	0,00	0,05	0,00
300	72,53	43,74	92,00	65,68	80,51	0,00	0,00	1,97	68,67	0,00	0,00	1,97	0,00
400	66,29	44,21	92,90	73,56	81,31	0,00	0,00	19,05	96,78	0,00	0,00	19,05	0,00
500	68,60	45,17	92,90	75,34	84,69	0,00	0,00	29,24	98,07	0,00	0,00	29,24	0,00
600	62,00	48,33	92,90	81,35	90,73	0,00	0,00	39,23	98,39	0,00	0,00	39,23	0,00
700	77,83	57,13	92,00	81,50	91,24	0,00	0,00	45,07	98,88	0,00	0,00	46,35	0,00
800	239,75	188,38	92,00	89,35	101,42	0,00	0,00	49,63	99,04	0,00	0,00	49,56	0,00
900	447,75	222,11	93,00	105,37	128,77	0,00	0,00	53,46	99,16	0,00	0,00	54,57	0,00
1000	535,37	275,97	94,00	132,48	275,33	0,00	0,00	54,86	99,29	1,03	24,68	55,70	1,24
WG 25	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**	**POS**	**FIN**	**ACK**	**TRC**	**POS**	**FIN**	**ACK**	**CTR**
100	3,95	2,15	42,10	1,34	1,65	0,00	0,00	0,00	0,05	0,00	0,00	0,00	0,00
200	11,56	9,27	80,00	4,14	4,64	0,00	0,00	0,13	0,24	0,00	0,00	0,13	0,00
300	69,94	42,43	91,00	72,18	77,36	0,00	0,00	14,55	96,42	0,00	0,00	14,55	0,00
400	74,09	43,85	91,80	75,71	81,85	0,00	0,00	29,66	97,99	0,00	0,00	29,66	0,00
500	75,75	42,73	91,90	77,64	83,96	0,00	0,00	40,09	98,53	0,00	0,00	40,09	0,00
600	98,76	62,43	91,90	77,80	85,98	0,00	0,00	47,40	98,96	0,00	0,00	47,40	0,00
700	331,50	92,88	92,00	82,56	89,22	0,00	0,00	52,46	99,12	0,00	0,00	52,46	0,00
800	375,81	185,17	93,20	87,55	97,60	0,00	0,00	57,48	99,34	0,00	1,45	57,64	0,45
800	499,20	200,94	94,00	107,84	139,42	0,00	0,00	58,78	99,35	0,00	9,11	64,07	8,79
1000	523,54	480,76	99,90	222,83	284,47	0,00	0,00	62,39	99,73	12,66	59,64	68,19	13,51

Tabla 6.8: Rendimiento del sistema para el iPhone 3GS con el sistema Vuforia (145.53 ms.) sobre la implementación UDP en todos los grupos de trabajo contemplados

bién con crecimiento más pronunciado a partir de los 900 clientes . El consumo de CPU también se incrementa, desde el 17.20 % para la población de 100 clientes hasta el 93 % de la población de 1000 clientes. El tiempo de respuesta en el servidor y su valor máximo también permanecen constantes, en torno a 2 ms. y 4 ms. hasta la población de 800 clientes, respectivamente, y crece significativa-

mente para las dos últimas poblaciones, llegando a 25.83 ms. y 48,65 ms. para la población de 1000 clientes. En cuanto a los mensajes perdidos, los mensajes de posición y posición final (columnas POS y FIN, respectivamente) no se pierden para ninguna población, tanto en el servidor como en el cliente. Lo mismo ocurre con los mensajes de control del servidor (CTR), que logran mantenerse para cualquier tamaño de población. Por su parte, los mensajes ACK se empiezan a perder a partir de la población de 300 clientes, con un valor de 0.01 % y llegando al 0.87 % para la población de 1000 clientes (con idéntico resultado para servidor y cliente). Por último, los mensajes de tiempo de respuesta en el cliente (TRC) empiezan a perderse a partir de los 200 clientes, con un valor del 0.01 % y siguen perdiéndose con un crecimiento uniforme hasta llegar a la última población, momento en el que su crecimiento se dispara, llegando al 39.58 % de pérdidas.

Con el grupo de trabajo de 10 clientes se consigue simular todos los tamaños de población con un tiempo de respuesta (columna RT) interactivo, obteniendo un máximo de 58.33 ms. para la última prueba. La desviación estándar de ese tiempo de respuesta (columna Dev) y la utilización de CPU (CPU) crecen uniformemente alcanzando un valor máximo de 72.3 ms. y del 95 % para la prueba de 1000 clientes. Por su parte, el SRT y su valor máximo (MaxSRT) tienen de nuevo un crecimiento uniforme hasta llegar a la población de 600 clientes, momento en el que crecen un orden de magnitud, llegando a 61.6 ms. y 66.86 ms. respectivamente. A partir de ese tamaño de población, su crecimiento vuelve a ser uniforme. Por lo que respecta a las pérdidas, su comportamiento es análogo al del grupo de trabajo de 5 clientes, aunque con un crecimiento más pronunciado. Sigue sin haber pérdida de paquetes de POS y FIN en servidor y cliente, la cantidad de mensajes ACK perdidos llega al 34.57 % para 1000 clientes y el porcentaje de mensajes TRC pasa del 86 % a partir de los 600 clientes simulados, por lo que los valores de tiempo de respuesta (RT) y desviación (Dev) obtenidos no son significativos, puesto que su cálculo se realiza con menos del 14 % de los datos totales.

Cuando se simulan grupos de trabajo de 20 clientes, se consigue simular más de 800 clientes dentro del tiempo de respuesta interactivo, obteniendo un RT de 239.75 ms., y alcanzando un valor máximo de 535.37 ms. para el último tamaño de población. La desviación estándar de ese RT (columna Dev) sufre un crecimiento escalonado obteniendo valores cercanos a 3 ms. para las primeras

dos pruebas, pasando por valores de entre 40-50ms. hasta la población de 700 clientes y llegando a 275.97 ms. para la población de 1000 clientes. Por su parte, el porcentaje de utilización de CPU está por encima del 92 % para todas las poblaciones superiores a 200 clientes. El tiempo de respuesta en el servidor (SRT) y su valor máximo (MaxSRT) sufren un crecimiento análogo al de la desviación comentada (Dev). Por lo que respecta a la pérdida de mensajes, las posiciones y posiciones finales (columnas POS y FIN, respectivamente) siguen llegando sin pérdidas al servidor, aunque empiezan a perderse las retransmisiones hacia los clientes, como se muestra en la población de 1000 clientes, momento en el que se pierden un 1.03 % de posiciones y un 24.68 % de posiciones finales. La cantidad de mensajes ACK perdidos llega al 54.86 % en el servidor y es ligeramente superior en los clientes, llegando al 55.7 % para la población de 1000 clientes. Por su parte, los mensajes de tiempo de respuesta del cliente (TRC) superan el 96 % desde la población de 400 clientes, por lo que los valores de RT y Dev no son significativos. A partir de este tamaño de grupo (20 clientes) se empieza a perder mensajes de control (CTR), lo que corrobora la pérdida de mensajes hacia los clientes, ya que hasta ahora los mensajes se perdían en el servidor.

Para el último tamaño de grupo simulado (25 clientes), se consigue simular hasta 600 clientes dentro del tiempo de respuesta interactivo, alcanzando un RT de 98.76 ms.. No obstante, tanto el tiempo de respuesta (RT) obtenido como su desviación (Dev) no son representativos de la prueba a partir de 300 clientes, puesto que su cálculo se realiza con menos del 4 % de los valores totales, como indica la columna TRC con su valor del 96.42 % de pérdidas. El tiempo de respuesta en el servidor y su valor máximo (SRT y MaxSRT) tienen un comportamiento análogo al obtenido para el grupo de trabajo de 20 clientes, con un crecimiento escalonado que alcanza los 222.83 ms. y 284.47 ms., respectivamente. Por último, las pérdidas de mensajes en los clientes se hacen más evidentes, puesto que desde los 700 clientes simulados los valores de las columnas FIN y ACK del cliente difieren respecto de las mismas columnas en el servidor. Este efecto se observa, por ejemplo, en la columna FIN de servidor y cliente para la última población, con valores del 0 % y 59.64 %, lo cual indica que mientras que todas las posiciones finales llegan al servidor, más de la mitad no consiguen ser retransmitidas a los clientes.

En la Tabla 6.9 se muestra el rendimiento del sistema cuando todos los clien-

tes son del tipo Nexus One, el teléfono móvil más rápido, con tan solo 56.68ms. de ciclo de actuación. En esta tabla se pueden apreciar dos subtablas que representan los dos primeros grupos de trabajo contemplados (5 y 10 clientes), como se comentará a continuación, no se han incluido los dos últimos grupos porque sus valores no son representativos debido a la gran cantidad de pérdidas que se obtienen. Cada subtabla contiene los cinco parámetros analizados en las pruebas anteriores: el tiempo de respuesta (columna etiquetada como RT) medido en milisegundos, su desviación estándar (Dev) medida también en milisegundos, la utilización de CPU medida en porcentaje (CPU), el tiempo de respuesta en el servidor (SRT). Acto seguido figuran dos grupos de cuatro columnas cada uno (que llamaremos central y derecho) que representan el porcentaje de paquetes de cada tipo que se pueden perder tanto en el servidor como en el cliente (recordemos que la implementación es UDP y, por tanto, puede perder paquetes). Los tres primeros parámetros de cada grupo son equivalentes y se corresponden con las posiciones enviadas (POS), las posiciones finales enviadas (FIN), los mensajes de reconocimiento (ACK). El último parámetro del grupo central, que se corresponde con el servidor, representa el porcentaje de pérdidas de los mensajes de tiempo de respuesta por ciclo (TRC), mientras que el último parámetro del grupo de la derecha representa el porcentaje de pérdidas de los mensajes de control que manda el servidor al cliente (CTR).

La Tabla 6.9 muestra un comportamiento análogo al obtenido en la Tabla 6.8 pero con tiempos de respuesta (columna RT) superiores, debido al ciclo de actuación más corto del Nexus One. Con los grupos de trabajo de 5 y 10 clientes el tiempo de respuesta sigue siendo interactivo para todos los tamaños de población simulados. No obstante, los valores de ese tiempo de respuesta y su desviación estándar (Dev) dejan de ser representativos cuando el porcentaje de mensajes TRC perdidos supera el 90 %. Para el grupo de trabajo de cinco clientes esto ocurre al llegar a 500 clientes; a partir de ese momento los valores de tiempo de respuesta y su desviación no son representativos, puesto que se calculan con muy pocos valores (menos del 11 %). Lo mismo ocurre con el tiempo de respuesta en el servidor y su valor máximo (SRT y MaxSRT), que disparan sus valores al llegar a ese tamaño de población. El porcentaje de utilización de CPU comienza con un 29.4 % para la simulación de 100 clientes y crece cerca de un 20 % en las sucesivas simulaciones hasta llegar a la simulación de 400 clientes, momento en el que su valor alcanza el 91,9 %; a partir de ese momento su

						% pérdidas servidor				% pérdidas clientes			
WG 5	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**	**POS**	**FIN**	**ACK**	**TRC**	**POS**	**FIN**	**ACK**	**CTR**
100	1,59	0,24	29,40	0,69	0,71	0,00	0,00	0,00	0,00	0	0	0,00	0,00
200	1,79	0,69	55,90	0,77	1,02	0,00	0,00	0,00	0,02	0	0	0,00	0,00
300	2,08	0,86	74,20	0,86	1,05	0,00	0,00	0,06	0,14	0	0	0,06	0,00
400	4,70	6,68	91,90	2,04	6,33	0,00	0,00	0,26	0,95	0	0	0,26	0,00
500	12,08	15,13	93,10	30,34	56,59	0,00	0,00	11,08	89,83	0	0	11,08	0,00
600	14,21	15,35	93,00	32,39	46,60	0,00	0,00	19,56	95,44	0	0	19,56	0,00
700	17,27	17,64	93,90	32,61	50,18	0,00	0,00	29,30	97,55	0	0	29,30	0,00
800	21,51	25,74	93,00	33,30	45,54	0,00	0,00	36,10	97,94	0	0	36,10	0,00
900	35,42	45,58	94,00	36,38	54,14	0,00	0,00	41,16	97,99	0	0	41,16	0,00
1000	38,80	53,20	95,00	52,60	74,38	0,00	0,00	44,46	98,52	0	0	44,46	0,00
WG 10	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**	**POS**	**FIN**	**ACK**	**TRC**	**POS**	**FIN**	**ACK**	**CTR**
100	2,20	0,64	47,80	0,85	0,90	0,00	0,00	0,00	0,04	0	0	0,00	0,00
200	3,51	2,17	84,70	1,39	1,88	0,00	0,00	0,17	0,33	0	0	0,17	0,00
300	23,41	18,28	93,00	31,44	36,01	0,00	0,00	15,99	96,57	0	0	15,99	0,00
400	29,04	17,30	93,00	32,14	38,88	0,00	0,00	30,81	97,82	0	0	30,81	0,00
500	18,44	16,26	93,00	31,08	36,45	0,00	0,00	41,54	98,33	0	0	41,54	0,00
600	44,36	13,86	92,00	31,64	43,54	0,00	0,00	49,20	98,95	0	0	49,20	0,00
700	57,41	32,22	92,00	39,14	50,45	0,00	0,00	54,04	99,13	0	0	54,04	0,00
800	60,22	40,01	93,00	52,48	53,50	0,00	0,00	57,23	99,32	0	0	57,23	0,00
900	49,53	47,73	94,00	103,38	106,17	0,00	0,00	57,37	99,58	0	0	57,37	0,00
1000	81,29	141,70	94,00	130,81	194,90	0,00	0,00	58,80	99,65	0	0	58,80	0,00

Tabla 6.9: Rendimiento del sistema para el Nexus One con el sistema Vuforia (56.68 ms.) sobre la implementación UDP en todos los grupos de trabajo contemplados

crecimiento se atenúa, llegando a un valor máximo del 95 % para la simulación con 1000 clientes. Para el grupo de trabajo de 10 clientes el crecimiento es similar, aunque el porcentaje de pérdidas de TRC supera el 96 % ya en la simulación con 300 clientes, por lo que los valores de tiempo de respuesta (RT) y su desviación (Dev) dejan de ser representativos, aunque siguen siendo interactivos. En ambos grupos de trabajo el sistema aún es capaz de retransmitir posiciones (POS), posiciones finales (FIN) y mensajes de reconocimiento (ACK) desde el servidor a los clientes, por lo que las tres columnas tienen los mismo valores tanto en el servidor como en el cliente. La pérdida de mensajes POS, FIN y CTR es nula para ambos grupos de trabajo y la cantidad máxima de ACKs perdidos es del 58.8 %, obtenida para 1000 clientes y el grupo de trabajo de 10 clientes. No se mostrará los resultados obtenidos para los grupos de trabajo de 20 y 25 clientes, puesto que el porcentaje de pérdida de mensajes es muy elevado desde el primer tamaño de población, con valores de TRC superiores al 90 % y valores de ACK de hasta el 70 %, lo cual desvirtúa los valores obtenidos en las medias mostradas. Para justificar esta decisión, en la Tabla 6.10 se muestra el ancho de banda requerido por cada una de las pruebas y tamaños de grupo al utilizar el

Nexus One con el ciclo de actuación de la caracterización de Vuforia (56.68 ms.).

One	WG 5	WG 10	WG 20	WG 25
100	15,460	30,919	61,838	77,298
200	30,919	61,838	123,677	154,596
300	46,379	92,757	185,515	231,893
400	61,838	123,677	247,353	309,191
500	77,298	154,596	309,191	386,489
600	92,757	185,515	371,030	463,787
700	108,217	216,434	432,868	541,085
800	123,677	247,353	494,706	618,383
900	139,136	278,272	556,544	695,680
1000	154,596	309,191	618,383	772,978

Tabla 6.10: Ancho de banda (Mbps) necesario para ejecutar las pruebas con el teléfono móvil Nexus One con el sistema Vuforia

En la Tabla 6.10 se observa el ancho de banda necesario para ejecutar las pruebas con el teléfono móvil Nexus One con el sistema Vuforia, medido en Mbps. La primera columna representa la cantidad de clientes simulados en cada prueba, que van desde los 100 clientes hasta los 1000 clientes. Las cuatro siguientes columnas representan el ancho de banda necesario para simular las pruebas con los grupos de trabajo de 5, 10, 20 y 25 clientes. Como se comentó en la sección 5.3.2, la red utilizada para las pruebas es una red de área local IEEE 802.3u de 100 Mbps que pertenece a la Universitat de València (UV), por lo que todos los valores de la tabla que se acerquen a los 100 Mbps representan pruebas en las que las pérdidas de mensajes serán elevadas. A la vista de la tabla, la mayoría de las pruebas con esta configuración necesitan más ancho de banda del que se les puede proporcionar, llegando a un máximo de 772.978 Mbps para el grupo de trabajo de 25 clientes y una población de 1000 clientes. Por lo tanto, para poder simular sin problemas todos los grupos de trabajo y tamaños de población haría falta una red de 1000 Mbps.

Estos resultados muestran que al utilizar el rastreo sin marcas de Vuforia, el sistema CAR sigue proporcionando tiempos de respuesta interactivos, excepto

cuando el ancho de banda necesario para ejecutar la prueba sobrepasa la capacidad de la red. Cuando esto pasa se pierde una cantidad de paquetes tal que malogra los resultados obtenidos, puesto que se obtienen con una cantidad ínfima de mensajes, que no son representativos de la prueba. Mientras esto no ocurra, el sistema con la mejora UDP proporciona los mejores tiempos de respuesta y la mejor utilización de la CPU, consiguiendo simular poblaciones de 1000 clientes en tiempo interactivo.

6.5.1. Conclusiones

A partir de los resultados de la caracterización del sistema CAR, se ha abordado el problema de la escalabilidad del mismo. Se han presentado diferentes aproximaciones para mejorar el rendimiento del sistema. En primer lugar, se intentó reducir al mínimo la cantidad de mensajes intercambiados entre el cliente origen y el servidor, dado que era uno de los factores limitantes de la productividad del sistema. En este sentido, se modificó el funcionamiento del servidor para que no retransmitiese cada uno de los mensajes de los clientes vecinos al cliente origen, sino que se le añadió la lógica necesaria para que llevase la cuenta de dichos mensajes ACK y respondiese al cliente origen con un único mensaje cuando se hubiesen recibido todas las respuestas ACK. Con este servidor, llamado *activo* por realizar más cálculos que el anterior, se obtiene una mejora en el rendimiento del sistema, consiguiendo mejores tiempos de respuesta y demostrando que el intercambio de mensajes entre clientes origen y servidor era realmente un factor limitante del sistema.

Por otra parte, como se partía de un sistema con sockets TCP que necesitaba un gran número de ficheros abiertos para su funcionamiento, se utilizó la función “select” de los sockets BSD para reducir los hilos necesarios en el servidor. Con ello se consiguió una ligera mejora en el rendimiento global del sistema y se obtuvo la implementación que serviría de base para la auténtica mejora. Dicha optimización llegó al prescindir de la comunicación orientada a conexión que proporciona el protocolo TCP/IP. En su lugar se utilizó sockets UDP, con lo que se redujo la sobrecarga de información transmitida con cada mensaje, puesto que las cabeceras UDP son mucho más reducidas que las TCP. Con esta última implementación se obtuvo un incremento de prestaciones muy significativo, con-

siguiendo reducir el tiempo de respuesta medio de los clientes hasta un orden de magnitud en algunos de los grupos de trabajo contemplados. Al mismo tiempo se demostró que la utilización de UDP no representa ningún riesgo para el rendimiento global del sistema, puesto que el porcentaje total de paquetes perdidos es despreciable siempre que el ancho de banda necesario para ejecutar la prueba no sobrepase la capacidad de la red. Si esto se cumple, la pérdida de mensajes se percibe como un ligero parpadeo en las pantallas de los dispositivos clientes, puesto que actúan varias veces por segundo.

Tras obtener la mejor implementación posible del sistema, se intentó modificar parámetros del sistema operativo para acomodarlo a nuestro tipo de aplicación, tratando de sacar beneficio de *quantos* más grandes, de menos cambios de contexto, etc.. No obstante, no se consiguió que ninguna de estas modificaciones mejorase el rendimiento del sistema. Por lo que se descarta cualquier modificación del sistema operativo.

Por último, se ha demostrado que el sistema seguirá proporcionando tiempos de respuesta interactivos con la llegada de nuevos teléfonos móviles, puesto que la mejora de prestaciones de los mismo se traducirá en un ligero incremento de su ciclo de actuación. Ese incremento obliga al sistema a funcionar más rápido pero es capaz de reaccionar y dar servicio a más de 1000 dispositivos basados en telefonía móvil con tiempos de respuesta inferiores al umbral de interactividad (250 ms.), usando como único servidor un computador personal con un hardware estándar. Esa gran cantidad de dispositivos móviles supera con creces las necesidades actuales de cualquier aplicación CAR actual.

7

Conclusiones y Trabajo Futuro

Este capítulo recoje las conclusiones generales que se extraen del trabajo realizado, y que se han ido comentando a lo largo de cada uno de los capítulos anteriores. Por otra parte, se comentan las líneas de investigación que surgen como continuación del trabajo realizado en esta tesis, dentro del campo de los sistemas CAR. Finalmente, se incluyen las publicaciones realizadas para difundir las aportaciones de la tesis.

7.1. Conclusiones y aportaciones

Esta tesis se ha centrado en el estudio de la mejora de prestaciones de los sistemas CAR. Para ello se empezó caracterizando el comportamiento y las pres-

taciones de distintos modelos de teléfonos móviles disponibles en el mercado en el momento de la evaluación, puesto que los teléfonos móviles son los dispositivos que mejor se adaptan a las necesidades de los usuarios de sistemas CAR. El sistema de realidad aumentada más común, al inicio de la investigación, era ARToolKit (basado en marcas). Por ello, la caracterización inicial se hizo sobre este sistema. No obstante, a finales de la investigación surgió un sistema sin marcas denominado Vuforia, que se extendió rápidamente. Este sistema se ha incluido en la caracterización con el objetivo de que ésta fuese lo más completa posible. Finalmente, tras caracterizar los teléfonos móviles, se ha desarrollado un sistema CAR que nos permite simular miles de teléfonos móviles comunicándose a través de un único servidor. Con este simulador se ha caracterizado el sistema CAR completo y se han realizado diversas mejoras sobre el mismo. Así pues, las aportaciones más importantes en los diferentes aspectos que cubre esta tesis son:

Acerca de la caracterización de dispositivos móviles con ARToolKit. Se ha demostrado que la mejor productividad (medida en FPS) la obtienen los teléfonos móviles de última generación basados en el sistema operativo Android. Sin embargo, para teléfonos de gama baja, son los dispositivos basados en el sistema operativo iOS los que obtienen mejores resultados. En cuanto a las diferentes etapas en las que se descomponen las aplicaciones CAR, se ha demostrado que la etapa que más tiempo consume es la de detección de marcas, seguida de la captura de imágenes, la etapa de dibujado y, por último, la etapa de transmisión.

Sobre la caracterización de dispositivos móviles con Vuforia. Se ha mostrado que la mejor productividad la obtienen los teléfonos móviles basados en el sistema operativo Android, tanto de gama alta como baja. En esta implementación ninguna de las etapas del ciclo CAR destaca sobre las demás, durando todas menos de 20 ms.

Respecto de la caracterización de teléfonos móviles con varios núcleos. Se ha demostrado que no proporcionan una productividad lineal con el número de núcleos, debido a que las etapas CAR son inherentemente secuenciales, por lo que no se pueden paralelizar. También se ha demostrado que los teléfonos móviles que están por llegar seguirán teniendo un comportamiento análogo y que sólo acelerarán ligeramente cada una de las etapas CAR, por lo que esta

caracterización seguirá siendo válida.

Acerca de la caracterización de los sistemas CAR. Se ha desarrollado un sistema CAR completo que se ha validado y caracterizado. Los resultados han demostrado que es capaz de simular más de 1000 teléfonos móviles con un comportamiento interactivo (respondiendo en menos de 250 ms.).

Respecto de la escalabilidad de los sistemas CAR basados en teléfonos móviles. Se han simulado diversas alternativas de implementación de los sistemas CAR y se ha demostrado que la mejor opción es la de implementar el sistema con sockets UDP, puesto que con una red suficientemente potente la cantidad de paquetes perdidos es despreciable y los beneficios son sustanciales en comparación con la implementación TCP en términos de latencia o tiempo de respuesta. Así mismo, se ha demostrado que el sistema se beneficia de una cantidad de hilos reducida, a expensas de que el servidor CAR sea más complejo. También se ha probado que no se obtiene ningún beneficio de la modificación de parámetros del sistema operativo como el tamaño del "quantum" o de la afinidad de procesador. Por último, se ha demostrado que el sistema seguirá proporcionando tiempos de respuesta interactivos con los teléfonos móviles que están por venir, puesto que la mejora de prestaciones de los mismos se traducirá en un ligero incremento de su ciclo de actuación. Ese incremento obliga al sistema a funcionar más rápido pero es capaz de reaccionar y dar servicio a más de 1000 teléfonos móviles con tiempos de respuesta inferiores al umbral de interactividad (250 ms.), usando un único servidor ejecutándose en un computador personal con un hardware estándar. Esa gran cantidad de dispositivos móviles supera con creces las necesidades actuales de cualquier aplicación CAR.

7.2. Líneas de investigación abiertas

La realización de esta tesis deja abiertas varias líneas de trabajo que se podrían desarrollar para mejorar el sistema.

- **Empleo de la arquitectura P2P para la comunicación entre dispositivos móviles.** Todas nuestras implementaciones y variantes del sistema han coin-

cidido en la utilización del modelo cliente/servidor, de forma que los teléfonos móviles se comunicaban entre ellos a través del servidor. En la caracterización del sistema se ha visto que la comunicación entre el cliente origen y el servidor era un factor limitante del sistema, por lo que sería interesante caracterizar el sistema si se elimina el servidor y los teléfonos móviles se comunican entre ellos sin intermendiarios.

- **Prescindir del mensaje de reconocimiento (ACK).** Por cada mensaje de actualización de posición que genera un cliente recibe tantos mensajes ACK como vecinos tiene en su grupo de trabajo. Esto genera una gran cantidad de tráfico que asegura que todos los vecinos han recibido dicha actualización. Sería interesante comprobar cómo funciona el sistema si se prescinde de dichos mensajes ACK, de forma que cada teléfono móvil se dedicaría a transmitir sus posición y visualizar las que le lleguen de los demás sin tener que contestarlas.

- Combinando las dos líneas anteriores, **realizar un sistema sin servidor y sin mensajes de reconocimiento**. De esta forma se tendría un sistema en el que las comunicaciones ya no serían un factor limitante. En él los teléfonos móviles se limitarían a difundir sus posiciones sin preocuparse de quién pueda oirlas, mientras toman nota de las posiciones que vayan escuchado.

- **Aplicación en el mercado actual.** Aunque se ha conseguido simular más de 1000 teléfonos móviles con comportamiento interactivo, las aplicaciones CAR del mercado actual están pensadas para tan solo unas decenas de dispositivos. Habría que buscar aplicaciones prácticas que involucrasen a un gran número de dispositivos móviles; pensadas para ejecutarse en sitios en los que haya una gran cantidad de personas con teléfonos móviles susceptibles de interactuar entre ellos, como estadios de fútbol, aeropuertos, congresos...

7.3. Publicaciones relacionadas

Las publicaciones originadas hasta el momento por el trabajo que se ha presentado en esta tesis doctoral, dentro del proyecto TIN 2009-14475-C04-04, son las siguientes (por orden de publicación):

Víctor Fernández-Bauset, Juan M. Orduña, Pedro Morillo. “Performance characterization on mobile phones for collaborative augmented reality (CAR) applications”. In Proceedings of the 2011 IEEE/ACM 15th International Symposium on Distributed Simulation and Real Time Applications (pp. 52-53). 2011, September. IEEE Computer Society. Conf. Intern. en ranking CORE, tipo B.

Víctor Fernández-Bauset, Juan M. Orduña, Pedro Morillo. “On the Characterization of CAR Systems Based on Mobile Computing”. In High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on (pp. 1205-1210). 2012, June. IEEE. Conf. Intern. en ranking CORE, tipo B.

Víctor Fernández-Bauset, Juan M. Orduña, Pedro Morillo. “Performance Characterization of Mobile Phones in Augmented Reality Marker Tracking”. In 12th International Conference on Computational and Mathematical Methods in Science and Engineering (CMMSE’12) (pp. 537-549). 2012.

Víctor Fernández-Bauset, Juan M. Orduña, Pedro Morillo. “How Mobile Phones Perform in Augmented Reality Marker Tracking?”. In Proceedings of XXIII Jornadas de Paralelismo, Elche (Alicante), 19-21 Septiembre 2012 (pp. 574-579). 2012, September.

Víctor Fernández-Bauset, Juan M. Orduña, Pedro Morillo, Vicente Cavero. “Characterization of CAR Servers for Augmented Reality Marker Tracking”. In Proceedings of XXIII Jornadas de Paralelismo, Elche (Alicante), 19-21 Septiembre 2012 (pp. 598-604). 2012, September.

Víctor Fernández-Bauset, Juan M. Orduña, Pedro Morillo. “On the Implementation of Servers for Large Scale CAR Systems based on Mobile Phones”. In Proceedings of the International Conference on Computer Graphics Theory and Applications (GRAPP 2013) (pp. 381-384). 2013, February. Conf. Intern. en ranking CORE, tipo A.

Víctor Fernández-Bauset, Juan M. Orduña, Pedro Morillo. “How mobile phones perform in collaborative augmented reality (CAR) applications”. The Journal

of Supercomputing, 1-13. Volume 65, Issue 3, (pp. 1179-1191). 2013, April. The Journal of Supercomputing. DOI: 10.1007/s11227-013-0925-8. Revista indexada en JCR (2012). Factor de Impacto 0'917, área "COMPUTER SCIENCE, THEORY & METHODS", posición 39 de 100, 2ndo cuartil.

Víctor Fernández-Bauset, Juan M. Orduña, Pedro Morillo. "On the Characterization of Markerless CAR Systems Based on Mobile Phones". In 13th International Conference on Computational and Mathematical Methods in Science and Engineering, CMMSE 2013 24–27 June, 2013. (pp. 618-629).

Víctor Fernández-Bauset, Juan M. Orduña, Pedro Morillo. "Improving the performance of CAR systems based on mobile phones". In High Performance Computing and Simulation (HPCS), 2013 International Conference on (pp. 671-673). 2013, July. IEEE. Conf. Intern. en ranking CORE, tipo B.

Víctor Fernández-Bauset, Juan M. Orduña, Pedro Morillo. "Improving the Server Performance of CAR Systems Based on Mobile Phones". In Proceedings of XXIV Jornadas de Paralelismo, Madrid , 17-20 Septiembre 2013 (pp. 343-348).

Víctor Fernández-Bauset, Juan M. Orduña, Pedro Morillo."Comparative performance evaluation of CAR systems based on mobile phones and feature tracking". 2014, January. The Journal of Supercomputing. DOI: 10.1007/s11227-013-1082-9. Revista indexada en JCR (2012). Factor de Impacto 0'917, área "COMPUTER SCIENCE, THEORY & METHODS", posición 39 de 100, 2ndo cuartil.

Ya aceptado y pendiente de publicación:

Víctor Fernández-Bauset, Juan M. Orduña, Pedro Morillo."Server Implementations for Improving the Performance of CAR Systems Based on Mobile Phones". Journal of Network and Computer Applications. Revista indexada en JCR (2012). Factor de Impacto 1'467, área "COMPUTER SCIENCE, HARDWARE & ARCHITECTURE", posición 11 de 50, 1er cuartil.

A

Código de la implementación Android

A continuación se exponen los fragmentos de código que se han tenido que modificar del código fuente original.

Marcas de tiempo para la cámara: Fichero HT03ACamera.java

```
1
2  private void startPreview() {
3    if (mPausing || mMainActivity.isFinishing()) {
4      return;
5    }
6    ensureCameraDevice();
7
8    // If we're previewing already, stop the preview first (this will blank
9    // the screen).
10   if (mPreviewing) {
11     stopPreview();
12   }
13   setPreviewDisplay(mSurfaceHolder);
14   setCameraParameters();
```

```
15
16    Log.d(TAG, "Camara1");
17    mCameraDevice.setOneShotPreviewCallback(mOneShotPreviewCallback);
18
19    try {
20      Log.v(TAG, "startPreview3x3");
21      mCameraDevice.startPreview();
22    } catch (Throwable ex) {
23      closeCamera();
24      throw new RuntimeException("startPreview failed", ex);
25    }
26    mPreviewing = true;
27    mStatus = IDLE;
28  }
29
30  /**
31   * Callback classes
32   */
33  private final class OneShotPreviewCallback implements
34  android.hardware.Camera.PreviewCallback {
35
36    @Override
37    public void onPreviewFrame(byte[] data, android.hardware.Camera camera) {
38      Log.d(TAG, "OneShotPreviewCallback.onPreviewFrame3x3");
39      Log.d(TAG, "Camara2");
40      if (data != null) {
41        Log.d(TAG, "data exist3x3");
42        cb.onPreviewFrame(data, null);
43      }
44      mHandler.sendEmptyMessage(NyARToolkitAndroidActivity.RESTART_PREVIEW);
45    }
46  };
```

Marcas de tiempo para la detección, etapa de envío y marcas de tiempo para el envío: Fichero ARToolkitDrawer.java

```
1
2   public void draw(byte[] data) {
3
4     if(data == null) {
5       Log.d("AR draw", "data= null");
6       return;
7     }
8     Log.d("AR draw", "data.length= " + data.length);
9     int width = 320;
10    int height = 240;
11
12    Bitmap bitmap = null;
13    if (!isYuv420spPreviewFormat) {
14      BitmapFactory.Options options = new BitmapFactory.Options();
15          options.inSampleSize = 4;
16          bitmap = BitmapFactory.decodeByteArray(data, 0, data.length, options);
17          if (bitmap == null) {
```

```
18            Log.d("AR␣draw", "data␣is␣not␣BitMap␣data.");
19            return;
20          }
21
22          if(bitmap.getHeight() < 240) {
23            bitmap = BitmapFactory.decodeByteArray(data, 0, data.length);
24            if (bitmap == null) {
25              Log.d("AR␣draw", "data␣is␣not␣BitMap␣data.");
26                  return;
27            }
28          }
29
30          width = bitmap.getWidth();
31          height = bitmap.getHeight();
32          Log.d("AR␣draw", "bitmap␣width␣*␣height()=␣" + width + "␣*␣" + height);
33
34          mRenderer.setBgBitmap(bitmap);
35
36      } else if (!mTranslucentBackground) {
37        // assume YUV420SP
38        int[] rgb = new int[(width * height)];
39        try {
40              bitmap = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB_8888);
41        } catch (Exception e) {
42              Log.d("AR␣draw", "bitmap␣create␣error.");
43              return;
44        }
45        long time1 = SystemClock.uptimeMillis();
46        // convert YUV420SP to ARGB
47        decodeYUV420SP(rgb, data, width, height, 2);
48        long time2 = SystemClock.uptimeMillis();
49        Log.d("ARToolkitDrawer", "ARGB␣decode␣time:␣" + (time2 - time1) + "ms");
50            bitmap.setPixels(rgb, 0, width, 0, 0, width, height);
51
52            mRenderer.setBgBitmap(bitmap);
53      }
54
55      // start coordinates calculation.
56      byte[] buf = new byte[width * height * 3];
57
58      if (!isYuv420spPreviewFormat) {
59        int[] rgb = new int[(width * height)];
60
61            bitmap.getPixels(rgb, 0, width, 0, 0, width, height);
62
63            // convert ARGB to RGB24
64            for (int i = 0; i < rgb.length; i++) {
65              byte r = (byte) (rgb[i] & 0x00FF0000 >> 16);
66              byte g = (byte) (rgb[i] & 0x0000FF00 >> 8);
67              byte b = (byte) (rgb[i] & 0x000000FF);
68              buf[i * 3] = r;
69              buf[i * 3 + 1] = g;
70              buf[i * 3 + 2] = b;
71            }
```

```
72    } else {
73      // assume YUV420SP
74          long time1 = SystemClock.uptimeMillis();
75          // convert YUV420SP to RGB24
76          decodeYUV420SP(buf, data, width, height, 1);
77          long time2 = SystemClock.uptimeMillis();
78          Log.d("ARToolkitDrawer", "RGB decode time: " + (time2 - time1) + "ms");
79    }
80
81    float[][] resultf = new float[MARKER_MAX][16];
82    int found_markers;
83    int ar_code_index[] = new int[MARKER_MAX];
84
85    Log.d("AR deteccion marca", "Marca1");
86    createNyARTool(width, height);
87    // Marker detection
88    try {
89      Log.d("AR draw", "Marker detection.");
90          raster = new NyARRgbRaster_RGB(width, height);
91          raster.wrapBuffer(buf);
92          found_markers = nya.detectMarkerLite(raster, 100);
93    } catch (NyARException e) {
94      Log.e("AR draw", "marker detection failed", e);
95          return;
96    }
97    Log.d("AR deteccion marca", "Marca2");
98
99    // An OpenGL object will be drawn if matched.
100   if (found_markers > 0) {
101         Log.d("AR draw", "!!!!!!!!!!!exist marker." + found_markers + "!!!!!!!!!!!");
102         // Projection transformation.
103         float[] cameraRHf = new float[16];
104         ar_util.toCameraFrustumRHf(ar_param, cameraRHf);
105
106         if (found_markers > MARKER_MAX)
107           found_markers = MARKER_MAX;
108         for (int i = 0; i < found_markers; i++) {
109           if (nya.getConfidence(i) < 0.60f)
110             continue;
111           try {
112             ar_code_index[i] = nya.getARCodeIndex(i);
113                 NyARTransMatResult transmat_result = ar_transmat_result;
114                 //
115                 NyARDoublePoint2d centro;
116                 //
117                 centro=nya.getTransmationMatrix(i, transmat_result);
118
119                 Log.d("CENTRO", "["+i+"]: ("+centro.x+","+centro.y+")");
120
121                 sendServer(i,Math.rint(centro.x*100)/100, Math.rint(centro.y*100)/100);
122
123                 //nya.getTransmationMatrix(i, transmat_result);
124                 ar_util.toCameraViewRHf(transmat_result, resultf[i]);
125
```

```
126             } catch (NyARException e) {
127           Log.e("AR draw", "getCameraViewRH failed", e);
128                   return;
129             }
130           }
131           mRenderer.objectPointChanged(found_markers, ar_code_index,
132                   resultf, cameraRHf);
133           if(mVoiceSound != null)
134             mVoiceSound.startVoice();
135     } else {
136       Log.d("AR draw", "not exist marker.");
137           if(mVoiceSound != null)
138             mVoiceSound.stopVoice();
139           mRenderer.objectClear();
140     }
141   }
142
143   public void sendServer(int i, double x, double y) {
144
145     try {
146       InetAddress serverAddr = InetAddress.getByName(IP);
147
148       Socket socket = new Socket(serverAddr, 4444);
149       String message = new String("["+i+"]=("+x+","+y+")");
150
151       try {
152             Log.d("SendServer","C: Sending"+message);
153
154             PrintWriter out = new PrintWriter( new BufferedWriter(
155              new OutputStreamWriter(socket.getOutputStream())),true);
156             Log.d("sendServer", "Envio1");
157             out.println(message);
158
159       } catch(Exception e) {
160             Log.e("SendServer","C: Error", e);
161       }
162
163       try {
164             BufferedReader in = new BufferedReader(
165                       new InputStreamReader(socket.getInputStream()));
166             int c;
167
168             while ((c = in.read()) != -1) {
169             }
170             Log.d("sendServer", "Envio2");
171
172       } catch (Exception e) {
173             Log.e("SendServer","C: Error", e);
174       }
175
176     } catch (Exception e) {
177       Log.e("SendServer","C: Error", e);
178     }
179   }
```

Marcas de tiempo para el dibujado: Fichero ModelRenderer.java

```
1   public void onDrawFrame(GL10 gl) {
2     if (modelChangep) {
3       mCube = new Cube();
4       initModel(gl);
5       reloadTexturep = false;
6     } else if (reloadTexturep) {
7       Log.d("ModelRenderer","in reloadTexturep:");
8       for (int i = 0; i < PATT_MAX; i++) {
9         if (model[i] != null)
10          model[i].reloadTexture(gl);
11      }
12      reloadTexturep = false;
13    }
14    if (bgChangep) {
15      Log.d("ModelRenderer","in loadBitmap:");
16          loadBitmap(gl);
17    }
18
19    /*
20     * Usually, the first thing one might want to do is to clear
21     * the screen. The most efficient way of doing this is to use
22     * glClear().
23     */
24
25    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
26
27    // Bg
28    if (bgBitmap != null && bgTextureName != -1) {
29      // Log.i("Renderer", "bgTextureName: "+bgTextureName);
30          bg.draw(gl, bgTextureName, bgBitmap);
31    }
32    if (drawp) {
33      // camera
34      if (useRHfp) {
35            gl.glMatrixMode(GL10.GL_PROJECTION);
36            gl.glLoadMatrixf(cameraRHf, 0);
37      } else if (cameraChangep) {
38            cameraSetup(gl);
39      }
40
41    if (mCube != null) {
42          Log.d("renderizado", "Renderizado1");
43      int patt[] = new int[PATT_MAX];
44      for (int i = 0; i < found_markers; i++) {
45            gl.glMatrixMode(GL10.GL_MODELVIEW);
46        if (useRHfp) {
47              gl.glLoadMatrixf(resultf[i], 0);
48                  //posicionamiento
49          gl.glTranslatef(0.0f, 0.0f, 0.0f);
50          //
51              gl.glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
52        } else {
```

```
53                gl.glLoadIdentity();
54            }
55            if (patt[ar_code_index[i]] != -1) {
56                Log.d("ModelRenderer", "onDrawFrame: " + i + ",model: "+ ar_code_index[i]);
57                patt[ar_code_index[i]] = -1;
58                if (lightp)
59                    lightSetup(gl);
60                model[ar_code_index[i]].enables(gl, 1.0f);
61                model[ar_code_index[i]].draw(gl);
62                model[ar_code_index[i]].disables(gl);
63                if (lightp)
64                    lightCleanup(gl);
65              } else {
66                Log.d("ModelRenderer", "onDrawFrame: " + i + ", "+ ar_code_index[i]);
67                gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
68                gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
69                mCube.draw(gl);
70              }
71              makeFramerate();
72              //drawp = false;
73            }
74            Log.d("renderizado", "Renderizado2");
75          }
76        } else {
77              gl.glMatrixMode(GL10.GL_PROJECTION);
78              gl.glLoadMatrixf(cameraRHf, 0);
79              gl.glEnable(GL10.GL_DEPTH_TEST);
80        }
81    }
```

B

Código de la implementación iOS

A continuación se exponen los fragmentos de código que se han tenido que modificar del código fuente original.

Marcas de tiempo para la cámara: Fichero VRViewController.m

```
1  #pragma mark -
2  #pragma mark AVCaptureSession delegate
3  - (void)captureOutput:(AVCaptureOutput *)captureOutput
4  didOutputSampleBuffer:(CMSampleBufferRef)sampleBuffer
5                fromConnection:(AVCaptureConnection *)connection
6  {
7          NSLog(@"Camara1");
8      CVImageBufferRef imageBuffer = CMSampleBufferGetImageBuffer(sampleBuffer);
9          NSLog(@"Camara2");
10
11 //     /*We lock the buffer and compute its width and height*/
12 //       CVPixelBufferLockBaseAddress(imageBuffer,0);
13 //       size_t width = CVPixelBufferGetWidth(imageBuffer);
14 //       size_t height = CVPixelBufferGetHeight(imageBuffer);
15 //       /*We unlock the pixel buffer, we are done with it*/
```

```
16  //      CVPixelBufferUnlockBaseAddress(imageBuffer,0);
17
18      NSLog(@"size=%f,%f",CVImageBufferGetCleanRect(imageBuffer).size.width,
19                              CVImageBufferGetCleanRect(imageBuffer).size.height);
20  //    NSLog(@"color=%d",CVImageBuf);
21
22          if (self.detectionEnabled) {
23                  if (!self.wrapper) {
24                          self.wrapper = [[ARToolKitPlusWrapper alloc] init];
25                          self.wrapper.delegate = self;
26                          [self.wrapper setupWithImageBuffer:imageBuffer];
27                  }
28                  [self.wrapper detectMarkerInImageBuffer:imageBuffer];
29          }
30  }
```

Marca inicial de tiempo para la detección: fichero TrackerSingleMarkerImpl.cxx

```
1   ARSM_TEMPL_FUNC int
2   ARSM_TEMPL_TRACKER::calc(const unsigned char* nImage, int nPattern, bool nUpdateMatrix,
3           ARMarkerInfo** nMarker_info, int* nNumMarkers)
4   {
5       ARMarkerInfo    *marker_info;
6       int             marker_num;
7
8           if(nImage==0)
9                   return 0;
10
11          PROFILE_BEGINSEC(profiler, SINGLEMARKER_OVERALL)
12
13          confidence = 0.0f;
14
15          NSLog(@"Marca1");
16
17      // detect the markers in the video frame
18          //
19      if(arDetectMarker(const_cast<unsigned char*>(nImage), this->thresh,
20         &marker_info, &marker_num) < 0)
21          {
22                  PROFILE_ENDSEC(profiler, SINGLEMARKER_OVERALL)
23          return -1;
24          }
25
26      // find best visible marker
27      int j, k = -1;
28      for(j = 0; j < marker_num; j++)
29          if(marker_info[j].id!=-1 && (nPattern==-1 || nPattern==marker_info[j].id))
30                  {
31              if(k == -1)
32                              k = j;
33                      else
34                      if(marker_info[k].cf < marker_info[j].cf)
35                              k = j;
```

```
36          }
37
38          if(nMarker_info)
39                  *nMarker_info = marker_info;
40          if(nNumMarkers)
41                  *nNumMarkers = marker_num;
42
43          // nothing found ?
44          //
45      if(k == -1)
46          {
47                  PROFILE_ENDSEC(profiler, SINGLEMARKER_OVERALL)
48          return -1;
49          }
50
51          confidence = marker_info[k].cf;
52
53      // get the transformation between the marker and the real camera
54          //
55          if(nUpdateMatrix)
56          {
57                  executeSingleMarkerPoseEstimator(&marker_info[k],
58                                          patt_center, patt_width, patt_trans);
59                  convertTransformationMatrixToOpenGLStyle(patt_trans, this->gl_para);
60          }
61
62          PROFILE_ENDSEC(profiler, SINGLEMARKER_OVERALL)
63          return marker_info[k].id;
64  }
```

Marca final de tiempo para la detección, código etapa envío y marcas de tiempo para el envío: Fichero arDetectMarker.cxx

```
1   // marker detection using tracking history
2   //
3   AR_TEMPL_FUNC int
4   AR_TEMPL_TRACKER::arDetectMarker(ARUint8 *dataPtr, int _thresh,
5         ARMarkerInfo **marker_info, int *marker_num)
6   {
7       ARInt16                 *limage=NULL;
8       int                     label_num;
9       int                     *area, *clip, *label_ref;
10      ARFloat                  *pos;
11      ARFloat                  rarea, rlen, rlenmin;
12      ARFloat                  diff, diffmin;
13      int                     cid, cdir;
14      int                     i, j, k;
15
16          autoThreshold.reset();
17          checkImageBuffer();
18
19  //      FILE* fp = fopen("imgdump.raw", "wb");
20  //      fwrite(dataPtr, 1, 320*240*2, fp);
```

```
21  //      fclose(fp);
22
23      *marker_num = 0;
24
25          for(int numTries = 0;;)
26          {
27                  limage = arLabeling(dataPtr, _thresh, &label_num, &area, &pos,
28                  &clip, &label_ref);
29                  if(limage)
30                  {
31                          marker_info2 = arDetectMarker2(limage, label_num, label_ref,
32                              area, pos, clip, AR_AREA_MAX,
33                          AR_AREA_MIN, 1.0, &wmarker_num);
34                          assert(wmarker_num <= MAX_IMAGE_PATTERNS);
35                          if(marker_info2)
36                          {
37                                  wmarker_info = arGetMarkerInfo(dataPtr, marker_info2,
38                                    &wmarker_num, _thresh);
39                                  assert(wmarker_num <= MAX_IMAGE_PATTERNS);
40                                  if(wmarker_info && wmarker_num>0)
41                                          break;
42                          }
43                  }
44
45                  if(!autoThreshold.enable)
46                          break;
47                  else
48                  {
49                          _thresh = thresh = (rand() %230) + 10;
50                          if(++numTries>autoThreshold.numRandomRetries)
51                                  break;
52                  }
53
54          }
55
56          if(!limage || !marker_info2 || !wmarker_info)
57                  return -1;
58
59      for( i = 0; i < prev_num; i++ ) {
60          rlenmin = 10.0;
61          cid = -1;
62          for( j = 0; j < wmarker_num; j++ ) {
63              rarea = (ARFloat)prev_info[i].marker.area / (ARFloat)wmarker_info[j].area;
64              if( rarea < 0.7 || rarea > 1.43 ) continue;
65              rlen = ( (wmarker_info[j].pos[0] - prev_info[i].marker.pos[0])
66                     * (wmarker_info[j].pos[0] - prev_info[i].marker.pos[0])
67                     + (wmarker_info[j].pos[1] - prev_info[i].marker.pos[1])
68                     * (wmarker_info[j].pos[1] - prev_info[i].marker.pos[1]) )
69                     / wmarker_info[j].area;
70              if( rlen < 0.5 && rlen < rlenmin ) {
71                  rlenmin = rlen;
72                  cid = j;
73              }
74          }
```

```
75          if( cid >= 0 && wmarker_info[cid].cf < prev_info[i].marker.cf ) {
76              wmarker_info[cid].cf = prev_info[i].marker.cf;
77              wmarker_info[cid].id = prev_info[i].marker.id;
78              diffmin = 10000.0 * 10000.0;
79              cdir = -1;
80              for( j = 0; j < 4; j++ ) {
81                  diff = 0;
82                  for( k = 0; k < 4; k++ ) {
83                      diff += (prev_info[i].marker.vertex[k][0]
84                            - wmarker_info[cid].vertex[(j+k)%4][0])
85                            * (prev_info[i].marker.vertex[k][0]
86                            - wmarker_info[cid].vertex[(j+k)%4][0])
87                            + (prev_info[i].marker.vertex[k][1]
88                            - wmarker_info[cid].vertex[(j+k)%4][1])
89                            * (prev_info[i].marker.vertex[k][1]
90                            - wmarker_info[cid].vertex[(j+k)%4][1]);
91                  }
92                  if( diff < diffmin ) {
93                      diffmin = diff;
94                      cdir = (prev_info[i].marker.dir - j + 4) %4;
95                  }
96              }
97              wmarker_info[cid].dir = cdir;
98          }
99      }
100
101     for( i = 0; i < wmarker_num; i++ ) {
102         if( wmarker_info[i].cf < 0.5 ) wmarker_info[i].id = -1;
103     }
104
105
106 /*-------------------------------------------------------------*/
107
108     for( i = j = 0; i < prev_num; i++ ) {
109         prev_info[i].count++;
110         if( prev_info[i].count < 4 ) {
111             prev_info[j] = prev_info[i];
112             j++;
113         }
114     }
115     prev_num = j;
116
117     for( i = 0; i < wmarker_num; i++ ) {
118         if( wmarker_info[i].id < 0 )
119                         continue;
120
121         for( j = 0; j < prev_num; j++ ) {
122             if( prev_info[j].marker.id == wmarker_info[i].id )
123                                 break;
124         }
125                 if(j<MAX_IMAGE_PATTERNS)
126                 {
127                         prev_info[j].marker = wmarker_info[i];
128                         prev_info[j].count  = 1;
```

```
129                             if( j == prev_num )
130                                     prev_num++;
131                     }
132             }
133
134             int jAux;
135
136             for( i = 0; i < prev_num; i++ ) {
137                     for( j = 0; j < wmarker_num; j++ ) {
138                             rarea = (ARFloat)prev_info[i].marker.area
139                                     / (ARFloat)wmarker_info[j].area;
140                             if( rarea < 0.7 || rarea > 1.43 ) continue;
141                             rlen = ( (wmarker_info[j].pos[0] - prev_info[i].marker.pos[0])
142                                     * (wmarker_info[j].pos[0] - prev_info[i].marker.pos[0])
143                                     + (wmarker_info[j].pos[1] - prev_info[i].marker.pos[1])
144                                     * (wmarker_info[j].pos[1] - prev_info[i].marker.pos[1]) )
145                                     / wmarker_info[j].area;
146                             if( rlen < 0.5 ) break;
147                     }
148                     if(j==wmarker_num && wmarker_num<MAX_IMAGE_PATTERNS) {
149                             wmarker_info[wmarker_num] = prev_info[i].marker;
150                             wmarker_num++;
151                                     assert(wmarker_num <= MAX_IMAGE_PATTERNS);
152                     }
153     //NSLog(@"--->x:%f y:%f", wmarker_info[j].pos[0],   wmarker_info[j].pos[1]);
154                     jAux = j;
155             }
156
157
158             *marker_num   = wmarker_num;
159
160             *marker_info = wmarker_info;
161
162                     assert(*marker_num <= MAX_IMAGE_PATTERNS);
163
164                     if(autoThreshold.enable)
165                             thresh = autoThreshold.calc();
166
167                     NSLog(@"Marca2");
168
169             NSLog(@"Envio1");
170             // This will be the read stream.
171             CFReadStreamRef myReadStream = NULL;
172
173             // This will be the write stream.
174             CFWriteStreamRef myWriteStream = NULL;
175
176             NSString *host = @"147.156.80.67";//192.168.0.17";
177             int port = 4444;
178
179             // Create socket.
180             CFStreamCreatePairWithSocketToHost(kCFAllocatorDefault,
181                                                (CFStringRef) host,
182                                                port,
```

```
183                                             &myReadStream,
184                                             &myWriteStream);
185
186     // Open write stream.
187     if (myWriteStream != NULL && CFWriteStreamOpen(myWriteStream))
188     {
189         // Write a byte.
190         //NSLog(@"Te lo mando...");
191
192         NSString *buffer1 = [NSString stringWithFormat:
193          @"X: %f, Y: %f\n", wmarker_info[jAux].pos[0],  wmarker_info[jAux].pos[1]];
194         CFWriteStreamWrite(myWriteStream, (const UInt8 *) [buffer1 UTF8String],
195           [buffer1 lengthOfBytesUsingEncoding:NSUTF8StringEncoding]);
196     }
197
198     // Open read stream.
199     if (myReadStream != NULL && CFReadStreamOpen(myReadStream))
200     {
201         int cont = 0, bytesRead;
202         UInt8 readBuffer[30];
203         //bool c_read = true;
204
205         do {
206             bytesRead = CFReadStreamRead(myReadStream, readBuffer, 30);
207             if (bytesRead > 0) {
208                 cont++;
209             }
210         } while (bytesRead > 0);
211
212         NSLog(@"caracteres leidos = %d",cont);
213     }
214     NSLog(@"Envio2");
215     // Be a good citizen.
216     CFReadStreamClose(myReadStream);
217     CFWriteStreamClose(myWriteStream);
218     CFRelease(myReadStream);
219     CFRelease(myWriteStream);
220
221         NSLog(@"x:%f y:%f", wmarker_info[jAux].pos[0],  wmarker_info[jAux].pos[1]);
222
223     return 0;
224 }
```

Marcas de tiempo para el dibujado: Fichero EAGLView.m

```
1  - (void)drawView
2  {
3          NSLog(@"Render1");
4
5          if (_modelViewMatrix&&self.object) {
6                  // Make sure that you are drawing to the current context
7                  [EAGLContext setCurrentContext:context];
8
```

```
 9                  glBindFramebufferOES(GL_FRAMEBUFFER_OES, viewFramebuffer);
10                  glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
11                  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
12
13                  //Setup model view matrix
14                  glLoadIdentity();
15                  glLoadMatrixf(_modelViewMatrix);
16                  glScalef(self.object.scaleFactor, self.object.scaleFactor,
17                           self.object.scaleFactor);
18                  //User defined rotation
19                  glRotatef(self.object.zRotation + spinZ, 0.0, 0.0, 1.0);
20                  glRotatef(self.object.yRotation, 0.0, 1.0, 0.0);
21                  glRotatef(self.object.xRotation, 1.0, 0.0, 0.0);
22                  glTranslatef(0.0, 0.0, self.object.zTranslation);
23                  glRotatef(90.0f, 1.0, 0, 0);
24                  glTranslatef(0, 0.5, 0);
25                //
26                  glDrawArrays(GL_TRIANGLES, 0, self.object.numberOfVertices);
27
28                  glBindRenderbufferOES(GL_RENDERBUFFER_OES, viewRenderbuffer);
29                  [context presentRenderbuffer:GL_RENDERBUFFER_OES];
30
31                  if (animating) spinZ+=2;
32          }
33          NSLog(@"Render2");
34  }
```

C

Traza ejemplo de un cliente CAR

Aquí se puede ver un ejemplo de la información proporcionada por el cliente cuando se ejecuta el código de la caracterización de clientes CAR sobre un iPhone 4. Este fichero es el que se analiza para obtener la información de tiempos de todas las etapas de ciclo CAR.

```
1  2012-10-08 10:20:36.663 ImageTargets[22489:4407] Camara1
2  2012-10-08 10:20:36.667 ImageTargets[22489:4407] setFramebuffer
3  2012-10-08 10:20:36.677 ImageTargets[22489:4407] Camara2
4  2012-10-08 10:20:36.681 ImageTargets[22489:4407] Marca1
5  2012-10-08 10:20:36.686 ImageTargets[22489:4407] Marca2
6  2012-10-08 10:20:36.689 ImageTargets[22489:4407] active trackables: 1
7  2012-10-08 10:20:36.692 ImageTargets[22489:4407] Renderizado1
8  2012-10-08 10:20:36.694 ImageTargets[22489:4407] USA OpenGL 2
9  translatePoseMatrix
10 scalePoseMatrix
11 multiplyMatrix
12 checkGLError
13 2012-10-08 10:20:36.700 ImageTargets[22489:4407] Renderizado2
14 2012-10-08 10:20:36.711 ImageTargets[22489:4407] Envio1
15 2012-10-08 10:20:36.714 ImageTargets[22489:4407] Recibiendo...
16 2012-10-08 10:20:36.717 ImageTargets[22489:4407] Leido:S: SendiX: 112.953827, Y: 21.704109
```

```
2012-10-08 10:20:36.721 ImageTargets[22489:4407] Envio2
2012-10-08 10:20:36.724 ImageTargets[22489:4407] presentFramebuffer
2012-10-08 10:20:36.751 ImageTargets[22489:4407] renderFrameQCARx2
2012-10-08 10:20:36.754 ImageTargets[22489:4407] Camara1
2012-10-08 10:20:36.757 ImageTargets[22489:4407] setFramebuffer
2012-10-08 10:20:36.760 ImageTargets[22489:4407] Camara2
2012-10-08 10:20:36.762 ImageTargets[22489:4407] Marca1
2012-10-08 10:20:36.766 ImageTargets[22489:4407] Marca2
2012-10-08 10:20:36.773 ImageTargets[22489:4407] active trackables: 1
2012-10-08 10:20:36.780 ImageTargets[22489:4407] Renderizado1
2012-10-08 10:20:36.783 ImageTargets[22489:4407] USA OpenGL 2
translatePoseMatrix
scalePoseMatrix
multiplyMatrix
checkGLError
2012-10-08 10:20:36.788 ImageTargets[22489:4407] Renderizado2
2012-10-08 10:20:36.795 ImageTargets[22489:4407] Envio1
2012-10-08 10:20:36.798 ImageTargets[22489:4407] Recibiendo...
2012-10-08 10:20:36.807 ImageTargets[22489:4407] Leido:S: SendiX: 113.116776, Y: 22.608152
2012-10-08 10:20:36.814 ImageTargets[22489:4407] Envio2
2012-10-08 10:20:36.817 ImageTargets[22489:4407] presentFramebuffer
2012-10-08 10:20:36.832 ImageTargets[22489:4407] renderFrameQCARx2
2012-10-08 10:20:36.840 ImageTargets[22489:4407] Camara1
2012-10-08 10:20:36.847 ImageTargets[22489:4407] setFramebuffer
2012-10-08 10:20:36.850 ImageTargets[22489:4407] Camara2
2012-10-08 10:20:36.853 ImageTargets[22489:4407] Marca1
2012-10-08 10:20:36.856 ImageTargets[22489:4407] Marca2
2012-10-08 10:20:36.860 ImageTargets[22489:4407] active trackables: 1
2012-10-08 10:20:36.862 ImageTargets[22489:4407] Renderizado1
2012-10-08 10:20:36.865 ImageTargets[22489:4407] USA OpenGL 2
translatePoseMatrix
scalePoseMatrix
multiplyMatrix
checkGLError
2012-10-08 10:20:36.884 ImageTargets[22489:4407] Renderizado2
2012-10-08 10:20:36.892 ImageTargets[22489:4407] Envio1
2012-10-08 10:20:36.895 ImageTargets[22489:4407] Recibiendo...
2012-10-08 10:20:36.900 ImageTargets[22489:4407] Leido:S: SendiX: 108.835564, Y: 21.742920
2012-10-08 10:20:36.906 ImageTargets[22489:4407] Envio2
```

D

Aplicación de Phyton que analiza el "log" del cliente CAR

Los siguientes ficheros de código se corresponde a los programas que analizan el log del cliente y obtienen de él las estadísticas necesarias para la caracterización del cliente.

El fichero que hay que ejecutar para que se realice el análisis del log del cliente es: caracterizacion_android.py.

```
1  #! /usr/bin/python
2  import os, sys, math, commands, datetime
3  from datetime import timedelta
4
5  def getDate(texto):
6      fechaHora = texto.split('␣')
7      fechaG = fechaHora[0].split('-')
8      anyo = 2011
9      mes = int(fechaG[0])
10     dia = int(fechaG[1])
```

```
11
12      horaG = fechaHora[1].split(':')
13      hora = int(horaG[0])
14      minuto = int(horaG[1])
15
16      segundoG = horaG[2].split('.')
17      segundo = int(segundoG[0])
18      milis = int(segundoG[1])
19
20      return anyo, mes, dia, hora, minuto, segundo, milis*1000
21
22  original = raw_input('Nombre del fichero a tratar: ')
23
24  direct = 'datos_' + original.split('.')[0]
25  os.system('mkdir ' + direct)
26  dirAux = 'datos_' + original.split('.')[0]+'_aux'
27  os.system('mkdir ' + dirAux)
28
29  etiquetas = ['Camara','Marca','Render','Envio']
30
31  fich3 = open(direct+'/medias.txt','w')
32  fich3.write("Nombre fichero: "+original+'\n\n'+
33  "Media de milisegundos por etapa:\n")
34  fich3.write("Camara"+'\t'+"Marca"+'\t'+"Renderizado"+
35  '\t'+"Envio"+'\t'+"Media\n")
36  for i in range(0,len(etiquetas)):
37      #print ("cat "+original+" |grep "+etiquetas[i]+"|cut -d' ' -f1,2 >> "+
38  dirAux+'/'+etiquetas[i])
39      os.system("cat "+original+" |grep "+etiquetas[i]+"|cut -d' ' -f1,2 >> "+
40  dirAux+'/'+etiquetas[i])
41
42      fich = open(dirAux+'/'+etiquetas[i], 'r')
43      fich2 = open(direct+'/'+etiquetas[i]+'2.txt','w')
44      first = 0
45      for line in fich:
46          if first == 0:
47              anyo,mes,dia,hora,minuto,segundo,milis = getDate(line)
48              #print 'anyo %d, mes %d, dia %d, hora %d, minuto %d, segundo %d, milis %d'
49  %(anyo,mes,dia,hora,minuto,segundo, milis)
50              first = 1
51              fecha1 = datetime.datetime(anyo,mes,dia,hora,minuto,segundo,milis)
52          else:
53              anyo,mes,dia,hora,minuto,segundo,milis = getDate(line)
54              fecha2 = datetime.datetime(anyo,mes,dia,hora,minuto,segundo,milis)
55              result = fecha2 - fecha1
56              fich2.write(str(result).split(':')[2]+'\n')
57              #fich2.write(str(result).split(':')[2].replace('.',',')+'\n')
58              first = 0
59
60      fich2.close()
61      fich.close()
62
63      #calculo las medias y lo pongo en milisegundos
64      fich = open(direct+'/'+etiquetas[i]+'2.txt','r')
```

```
65      fich2 = open(direct+'/'+etiquetas[i]+'.txt','w')
66      suma=0
67      cont=0
68      first = 0
69      for line in fich:
70          if first == 0:
71              first = 1
72          else:
73              aux = float(line)*1000
74              suma = aux + suma
75              cont = cont+1
76              fich2.write(str(aux).replace('.',',')+'\n')
77      fich2.close()
78      fich.close()
79      os.system('rm '+direct+'/'+etiquetas[i]+'2.txt')
80
81      fich3.write(str(suma/cont).replace('.',',')+'\t')
82
83  #se calcula los ciclos y su media
84  os.system("./ciclos.py "+original+" Camara1")
85  os.system("mv tpo_Camara1_"+ original +" ./"+direct)
86  fich = open(direct+"/tpo_Camara1_"+original,"r")
87  cont = 0
88  suma = 0
89  first = 0
90  for line in fich:
91      if first == 0:
92          first = 1
93      else:
94          suma = float(line.replace(',','.')) + suma
95          cont = cont + 1
96
97  fich.close()
98  fich3.write(str(suma/cont).replace('.',',')+'\t')
99
100 #se calcula las apariciones y las medias
101 os.system("./apariciones.py "+original)
102 os.system("./apariciones2.py "+original)
103 os.system("mv apa_"+ original +" ./"+direct)
104 os.system("mv apa2_"+ original +" ./"+direct)
105
106 fich3.write("\n\nApariciones por cada camara:\nMarca\tRender\tEnvio\n")
107 fich = open(direct+"/apa2_"+original,"r")
108
109 c1 = 0
110 c2 = 0
111 c3 = 0
112 cont = 0
113
114 for line in fich:
115     col = line.split('\t')
116     c1 = c1 + int(col[0])
117     c2 = c2 + int(col[1])
118     c3 = c3 + int(col[2])
```

```
119     if int(col[0]) != 0 or int(col[1]) != 0 or int(col[2]) != 0:
120         cont = cont + 1
121 
122 fich.close()
123 
124 fich3.write(str(float(c1)/cont)+'\t'+str(float(c2)/cont)+'\t'+str(float(c3)/cont))
125 
126 fich3.close()
127 os.system('rm␣-R␣'+dirAux)
```

A continuación se listan los programas a los que llama el programa principal. El primero al que llama es: ciclos.py

```
1  #!/usr/bin/python
2  import os, sys, math, commands, datetime
3  from datetime import timedelta
4  
5  def getDate(texto):
6      fechaHora = texto.split('␣')
7      fechaG = fechaHora[0].split('-')
8      anyo = 2011
9      mes = int(fechaG[0])
10     dia = int(fechaG[1])
11 
12     horaG = fechaHora[1].split(':')
13     hora = int(horaG[0])
14     minuto = int(horaG[1])
15 
16     segundoG = horaG[2].split('.')
17     segundo = int(segundoG[0])
18     milis = int(segundoG[1])
19 
20     return anyo, mes, dia, hora, minuto, segundo, milis*1000
21 
22 os.system("cat␣"+sys.argv[1]+"␣|grep␣"+sys.argv[2]+"|cut␣-d':'␣-f1,2,3␣>␣"+
23 sys.argv[2]+'_'+sys.argv[1])
24 
25 fich = open(sys.argv[2]+'_'+sys.argv[1], 'r')
26 fich2 = open('tpo_'+sys.argv[2]+'_'+sys.argv[1],'w')
27 first = 0
28 for line in fich:
29     if first == 0:
30         anyo,mes,dia,hora,minuto,segundo,milis = getDate(line)
31         #print 'anyo␣%d,␣mes␣%d,␣dia␣%d,␣hora␣%d,␣minuto␣%d,␣segundo␣%d,␣milis␣%d'
32  %(anyo,mes,dia,hora,minuto,segundo, milis)
33         first = 1
34         fecha1 = datetime.datetime(anyo,mes,dia,hora,minuto,segundo,milis)
35     else:
36         anyo,mes,dia,hora,minuto,segundo,milis = getDate(line)
37         fecha2 = datetime.datetime(anyo,mes,dia,hora,minuto,segundo,milis)
38         result = fecha2 - fecha1
39         fich2.write(str(float(str(result).split(':')[2])*1000).replace('.',',')+'\n')
40         fecha1 = fecha2
```

```
41
42 fich2.close()
43 fich.close()
44
45 os.system('rm␣'+sys.argv[2]+'_'+sys.argv[1])
```

Después se llama a la aplicación: apariciones.py

```
1  #! /usr/bin/python
2  import os, sys, math, commands, datetime
3  from datetime import timedelta
4
5  def getDate(texto):
6      fechaHora = texto.split('␣')
7      fechaG = fechaHora[0].split('-')
8      anyo = 2011
9      mes = int(fechaG[0])
10     dia = int(fechaG[1])
11
12     horaG = fechaHora[1].split(':')
13     hora = int(horaG[0])
14     minuto = int(horaG[1])
15
16     segundoG = horaG[2].split('.')
17     segundo = int(segundoG[0])
18     milis = int(segundoG[1])
19
20     return anyo, mes, dia, hora, minuto, segundo, milis*1000
21
22 #os.system("cat␣"+sys.argv[1]+"␣|grep␣ms␣>␣aux_"+sys.argv[1])
23
24 fich = open(sys.argv[1], 'r')
25 fich2 = open('apa_'+sys.argv[1],'w')
26 first = 0
27 for line in fich:
28     if line.find("Camara1") >= 0:
29         fich2.write("\nCamara\t")
30         #fich2.write(line)
31     if line.find("Renderizado1") >= 0:
32         fich2.write("Render\t")
33         #fich2.write(line)
34     if line.find("Marca1") >= 0:
35         fich2.write("Marca␣\t")
36         #fich2.write(line)
37     if line.find("Envio1") >= 0:
38         fich2.write("Envio␣\t")
39         #fich2.write(line)
40
41 fich2.close()
42 fich.close()
43
44 #os.system('rm␣'+'aux_'+sys.argv[1])
```

Y, por último, se llama a la aplicación: apariciones2.py

```
#! /usr/bin/python
import os, sys, math, commands, datetime
from datetime import timedelta

def getDate(texto):
    fechaHora = texto.split(' ')
    fechaG = fechaHora[0].split('-')
    anyo = 2011
    mes = int(fechaG[0])
    dia = int(fechaG[1])

    horaG = fechaHora[1].split(':')
    hora = int(horaG[0])
    minuto = int(horaG[1])

    segundoG = horaG[2].split('.')
    segundo = int(segundoG[0])
    milis = int(segundoG[1])

    return anyo, mes, dia, hora, minuto, segundo, milis*1000

#os.system("cat "+sys.argv[1]+" | grep ms > aux_"+sys.argv[1])

fich = open(sys.argv[1], 'r')
fich2 = open('apa2_'+sys.argv[1],'w')
first = 0

cont_r=0
cont_m=0
cont_e=0

for line in fich:
    if line.find("Camara1") >= 0:
        fich2.write(str(cont_r)+'\t'+str(cont_m)+'\t'+str(cont_e)+'\n')
        cont_r=0
        cont_m=0
        cont_e=0
    if line.find("Marca1") >= 0:
        cont_r = cont_r + 1
    if line.find("Renderizado1") >= 0:
        cont_m = cont_m + 1
    if line.find("Envio1") >= 0:
        cont_e = cont_e + 1

fich2.write(str(cont_r)+'\t'+str(cont_m)+'\t'+str(cont_e)+'\n')

fich2.close()
fich.close()

#os.system('rm '+'aux_'+sys.argv[1])
```

E

Herramientas para medir la utilización de CPU y memoria consumida en los clientes CAR

Para calcular el consumo de CPU y memoria se ha hecho uso de las utilidades que proporciona cada uno de los entornos de desarrollo. En el caso de Android, se hace uso de la herramienta ”Dalvik Debug Monitor“ presente en la instalación del SDK de Android. Dicha herramienta proporciona información en tiempo real del teléfono móvil en cuanto lo conectas al ordenador. Por lo tanto, se conecta el móvil[1] al ordenador, comprobando que no se ejecuta ninguna aplicación sobre él (estado de reposo) y se lanza la herramienta para ver el consumo de CPU y de memoria inicial. Una vez obtenida la referencia inicial, se ejecuta la aplicación y se observa cómo afecta a los consumos iniciales. Se muestra inicialmente el

[1] Las figuras que se muestran son de la caracterización del Motorola Milestone

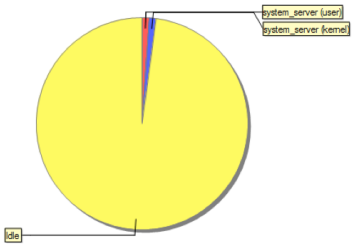

Figura E.1: Consumo de CPU del Motorola Milestone en estado de reposo

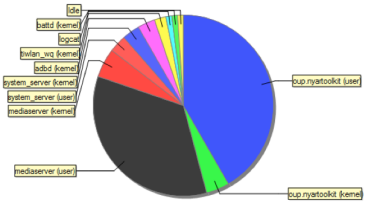

Figura E.2: Consumo de CPU del Motorola Milestone en estado de ejecución

consumo de CPU, que se obtiene al comparar las Figuras E.1 y E.2.

Las siguientes figuras corresponden al consumo de memoria inicial y final. Figuras E.3 y E.4, respectivamente.

Análogamente, comparando las gráficas de consumo de memoria, se ve que inicialmente había un 35 % de memoria libre, que se convierte en un 5 % de memoria libre cuando la aplicación se está ejecutando.

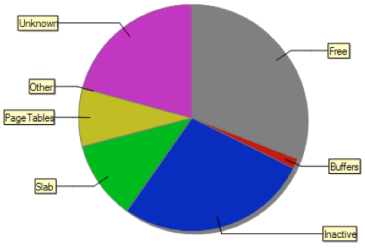

Figura E.3: Consumo de memoria del Motorola Milestone en estado de reposo

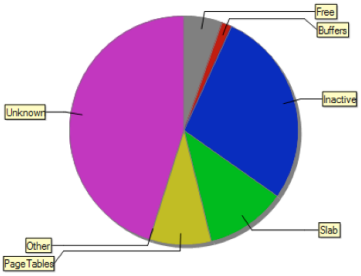

Figura E.4: Consumo de memoria del Motorola Milestone en estado de ejecución

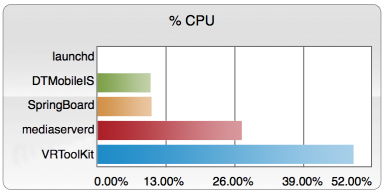

Figura E.5: Consumo de CPU del iPhone 3GS en estado de ejecución

Por su parte, la plataforma de desarrollo de Apple, conocida como xCode, proporciona la herramienta *Instruments*. Dicha herramienta te proporciona información de consumo de memoria y CPU (entre otras) en tiempo real del teléfono iOS que tengas conectado al ordenador. Pero, a diferencia de la herramienta de Android, no te permite obtener información si no está ejecutando ninguna aplicación, por lo que sólo se dispone de información de consumo durante la prueba. Por lo tanto, en la Figura E.5 se muestra el consumo de CPU y en la Figura E.6 el consumo de memoria[2].

[2]Estos datos son de la ejecución de la aplicación sobre el iPhone 3GS

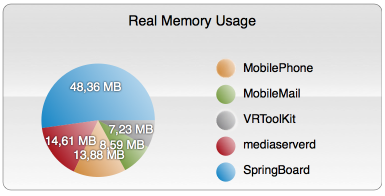

Figura E.6: Consumo de memoria del iPhone 3GS en estado de ejecución

F

Código de las implementaciones para Vuforia

Vuforia te suministra códigos AR de ejemplo tanto para Android como para iOS. El código que se ha utilizado es el proporcionado con el nombre *ImageTargets*. El único fichero que hay que modificar en la implementación sin marcas para iOS es: EAGLView.mm, que se adjunta a continuación. En él se ha añadido la etapa de envío y las marcas de tiempo para saber cuánto ocupa cada etapa CAR.

```
1  /*==============================================================================
2   Copyright (c) 2012 QUALCOMM Austria Research Center GmbH.
3   All Rights Reserved.
4   Qualcomm Confidential and Proprietary
5   ==============================================================================*/
6
7  #include <sys/socket.h>
8  #include <netinet/in.h>
9  #include <arpa/inet.h>
10 // Subclassed from AR_EAGLView
```

```
11  #import "EAGLView.h"
12  #import "Teapot.h"
13  #import "Texture.h"
14
15  #import <QCAR/Renderer.h>
16
17  #import "QCARutils.h"
18
19  #ifndef USE_OPENGL1
20  #import "ShaderUtils.h"
21  #endif
22
23  namespace {
24      // Teapot texture filenames
25      const char* textureFilenames[] = {
26          "TextureTeapotBrass.png",
27          "TextureTeapotBlue.png",
28          "TextureTeapotRed.png"
29      };
30
31      // Model scale factor
32      const float kObjectScale = 3.0f;
33  }
34
35
36  @implementation EAGLView
37
38  - (id)initWithFrame:(CGRect)frame
39  {
40      self = [super initWithFrame:frame];
41
42          if (self)
43      {
44          // create list of textures we want loading - ARViewController will do this for us
45          int nTextures = sizeof(textureFilenames) / sizeof(textureFilenames[0]);
46          for (int i = 0; i < nTextures; ++i)
47              [textureList addObject: [NSString stringWithUTF8String:textureFilenames[i]]];
48      }
49      return self;
50  }
51
52  - (void) setup3dObjects
53  {
54      NSLog(@"setup3dObjectsx2");
55      // build the array of objects we want drawn and their texture
56      // in this example we have 3 targets and require 3 models
57      // but using the same underlying 3D model of a teapot, differentiated
58      // by using a different texture for each
59
60      for (int i=0; i < [textures count]; i++)
61      {
62          Object3D *obj3D = [[Object3D alloc] init];
63
64          obj3D.numVertices = NUM_TEAPOT_OBJECT_VERTEX;
```

```
65          obj3D.vertices = teapotVertices;
66          obj3D.normals = teapotNormals;
67          obj3D.texCoords = teapotTexCoords;
68
69          obj3D.numIndices = NUM_TEAPOT_OBJECT_INDEX;
70          obj3D.indices = teapotIndices;
71
72          obj3D.texture = [textures objectAtIndex:i];
73
74          [objects3D addObject:obj3D];
75          [obj3D release];
76      }
77  }
78
79
80  // called after QCAR is initialised but before the camera starts
81  - (void) postInitQCAR
82  {
83      //variables del envio
84  //    float x, y, z;
85      //
86
87      NSLog(@"postInitQCARx2");
88      // These two calls to setHint tell QCAR to split work over multiple
89      // frames.  Depending on your requirements you can opt to omit these.
90      QCAR::setHint(QCAR::HINT_IMAGE_TARGET_MULTI_FRAME_ENABLED, 1);
91      QCAR::setHint(QCAR::HINT_IMAGE_TARGET_MILLISECONDS_PER_MULTI_FRAME, 25);
92
93      // Here we could also make a QCAR::setHint call to set the maximum
94      // number of simultaneous targets
95      //QCAR::setHint(QCAR::HINT_MAX_SIMULTANEOUS_IMAGE_TARGETS, 2);
96  }
97
98  // modify renderFrameQCAR here if you want a different 3D rendering model
99  //////////////////////////////////////////////////////////////////////////////
100 // Draw the current frame using OpenGL
101 //
102 // This method is called by QCAR when it wishes to render the current frame to
103 // the screen.
104 //
105 // *** QCAR will call this method on a single background thread ***
106 - (void)renderFrameQCAR
107 {
108     NSLog(@"renderFrameQCARx2");
109
110 NSLog(@"Camara1");
111
112     [self setFramebuffer];
113
114     // Clear colour and depth buffers
115     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
116
117 NSLog(@"Camara2");
118
```

```
119 NSLog(@"Marca1");
120
121     // Render video background and retrieve tracking state
122     QCAR::State state = QCAR::Renderer::getInstance().begin();
123     QCAR::Renderer::getInstance().drawVideoBackground();
124
125 NSLog(@"Marca2");
126
127     NSLog(@"active trackables: %d", state.getNumActiveTrackables());
128
129 NSLog(@"Renderizado1");
130
131     if (QCAR::GL_11 & qUtils.QCARFlags) {
132         glEnable(GL_TEXTURE_2D);
133         glDisable(GL_LIGHTING);
134         glEnableClientState(GL_VERTEX_ARRAY);
135         glEnableClientState(GL_NORMAL_ARRAY);
136         glEnableClientState(GL_TEXTURE_COORD_ARRAY);
137     }
138
139     glEnable(GL_DEPTH_TEST);
140     glEnable(GL_CULL_FACE);
141
142     for (int i = 0; i < state.getNumActiveTrackables(); ++i) {
143         // Get the trackable
144         const QCAR::Trackable* trackable = state.getActiveTrackable(i);
145         QCAR::Matrix44F modelViewMatrix = QCAR::Tool::convertPose2GLMatrix(
146 trackable->getPose());
147         QCAR::Matrix44F modelViewMatrix2 = QCAR::Tool::convertPose2GLMatrix(
148 trackable->getPose());
149         QCAR::Matrix44F modelViewMatrix3 = QCAR::Tool::convertPose2GLMatrix(
150 trackable->getPose());
151         QCAR::Matrix44F modelViewMatrix4 = QCAR::Tool::convertPose2GLMatrix(
152 trackable->getPose());
153
154         // Choose the texture based on the target name
155         int targetIndex = 0; // "stones"
156         if (!strcmp(trackable->getName(), "chips"))
157             targetIndex = 1;
158         else if (!strcmp(trackable->getName(), "tarmac"))
159             targetIndex = 2;
160
161         Object3D *obj3D = [objects3D objectAtIndex:targetIndex];
162
163         // Render using the appropriate version of OpenGL
164         if (QCAR::GL_11 & qUtils.QCARFlags) {
165
166             NSLog(@"USA OpenGL 11");
167             // Load the projection matrix
168             glMatrixMode(GL_PROJECTION);
169             glLoadMatrixf(qUtils.projectionMatrix.data);
170
171             // Load the model-view matrix
172             glMatrixMode(GL_MODELVIEW);
```

```
173             glLoadMatrixf(modelViewMatrix.data);
174             glTranslatef(0.0f, 0.0f, -kObjectScale);
175             glScalef(kObjectScale, kObjectScale, kObjectScale);
176
177             // Draw object
178             glBindTexture(GL_TEXTURE_2D, [obj3D.texture textureID]);
179             glTexCoordPointer(2, GL_FLOAT, 0, (const GLvoid*)obj3D.texCoords);
180             glVertexPointer(3, GL_FLOAT, 0, (const GLvoid*)obj3D.vertices);
181             glNormalPointer(GL_FLOAT, 0, (const GLvoid*)obj3D.normals);
182             glDrawElements(GL_TRIANGLES, obj3D.numIndices, GL_UNSIGNED_SHORT,
183 (const GLvoid*)obj3D.indices);
184         }
185 #ifndef USE_OPENGL1
186         else {
187             // OpenGL 2
188
189             NSLog(@"USA OpenGL 2");
190             QCAR::Matrix44F modelViewProjection;
191
192             ShaderUtils::translatePoseMatrix(50.0f, 0.0f, kObjectScale,
193 &modelViewMatrix.data[0]);
194 //------DATOS ENVIO
195             x = modelViewMatrix.data[12];
196             y = modelViewMatrix.data[13];
197             z = modelViewMatrix.data[14];
198 //------
199             ShaderUtils::scalePoseMatrix(kObjectScale, kObjectScale, kObjectScale,
200 &modelViewMatrix.data[0]);
201             ShaderUtils::multiplyMatrix(&qUtils.projectionMatrix.data[0],
202 &modelViewMatrix.data[0], &modelViewProjection.data[0]);
203
204             glUseProgram(shaderProgramID);
205
206             glVertexAttribPointer(vertexHandle, 3, GL_FLOAT, GL_FALSE, 0,
207 (const GLvoid*)obj3D.vertices);
208             glVertexAttribPointer(normalHandle, 3, GL_FLOAT, GL_FALSE, 0,
209 (const GLvoid*)obj3D.normals);
210             glVertexAttribPointer(textureCoordHandle, 2, GL_FLOAT, GL_FALSE, 0,
211 (const GLvoid*)obj3D.texCoords);
212
213             glEnableVertexAttribArray(vertexHandle);
214             glEnableVertexAttribArray(normalHandle);
215             glEnableVertexAttribArray(textureCoordHandle);
216
217             glActiveTexture(GL_TEXTURE0);
218             glBindTexture(GL_TEXTURE_2D, [obj3D.texture textureID]);
219             glUniformMatrix4fv(mvpMatrixHandle, 1, GL_FALSE,
220 (const GLfloat*)&modelViewProjection.data[0]);
221             glDrawElements(GL_TRIANGLES, obj3D.numIndices, GL_UNSIGNED_SHORT,
222 (const GLvoid*)obj3D.indices);
223
224             ShaderUtils::checkGlError("EAGLView renderFrameQCAR");
225         }
226 #endif
```

```
227         }
228
229         glDisable(GL_DEPTH_TEST);
230         glDisable(GL_CULL_FACE);
231
232         if (QCAR::GL_11 & qUtils.QCARFlags) {
233             glDisable(GL_TEXTURE_2D);
234             glDisableClientState(GL_VERTEX_ARRAY);
235             glDisableClientState(GL_NORMAL_ARRAY);
236             glDisableClientState(GL_TEXTURE_COORD_ARRAY);
237         }
238 #ifndef USE_OPENGL1
239         else {
240             glDisableVertexAttribArray(vertexHandle);
241             glDisableVertexAttribArray(normalHandle);
242             glDisableVertexAttribArray(textureCoordHandle);
243         }
244 #endif
245
246         QCAR::Renderer::getInstance().end();
247
248 NSLog(@"Renderizado2");
249
250 //------Envio
251         int sock;
252         struct sockaddr_in dir;
253
254         sock = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
255
256         dir.sin_family = PF_INET;
257         dir.sin_port = htons(4444);
258
259         if (inet_aton("192.168.0.17", &dir.sin_addr)==0) {
260             NSLog(@"Error: inet_aton");
261         }
262
263         if (connect(sock, (struct sockaddr *) &dir, sizeof(struct
264                                                             sockaddr_in))<0) {
265             NSLog(@"Error: connect");
266         }
267
268         char buffer[128];
269         sprintf(buffer,"X: %, Y: %\n", x, y);
270
271         NSLog(@"Envio1");
272         if(write(sock, buffer, 128)<0){
273             NSLog(@"Error: write");
274         }
275
276
277         char bufferRcv[8];
278
279         NSLog(@"Recibiendo...");
280
```

```
281         if (read(sock, bufferRcv, 8)>=0) {
282             NSLog(@"Leido:%s",bufferRcv);
283         }
284         NSLog(@"Envio2");
285 
286     //------
287 
288         [self presentFramebuffer];
289 
290     }
291 
292     @end
```

Como se comentó en el capítulo de la caracterización del clientes (Capítulo 4), el código de Android es una portación del realizado para iOS, a través de JNI. Por lo tanto, pese a ser java, parte del código modificado se corresponde a código C++. En concreto, se han modificado dos ficheros del total de los suministrados en el proyecto *imageTargets* para Android: “ImageTargetsRenderer.java” e “ImageTargets.cpp”. El código del primero se muestra a continuación, donde se ha añadido la etapa de envío y las marcas de envío y la primera de la cámara.

```
1   /*==============================================================================
2               Copyright (c) 2012 QUALCOMM Austria Research Center GmbH.
3               All Rights Reserved.
4               Qualcomm Confidential and Proprietary
5 
6   @file
7       ImageTargetsRenderer.java
8 
9   @brief
10      Sample for ImageTargets
11 
12  ==============================================================================*/
13 
14 
15  package com.qualcomm.QCARSamples.ImageTargets;
16 
17  import javax.microedition.khronos.egl.EGLConfig;
18  import javax.microedition.khronos.opengles.GL10;
19 
20  import java.io.BufferedReader;
21  import java.io.BufferedWriter;
22  import java.io.InputStream;
23  import java.io.InputStreamReader;
24  import java.io.OutputStreamWriter;
25  import java.io.PrintWriter;
26  import java.net.InetAddress;
27  import java.net.Socket;
28  import java.util.ArrayList;
29 
30  import android.opengl.GLSurfaceView;
31 
32  import com.qualcomm.QCAR.QCAR;
```

```
33
34
35  /** The renderer class for the ImageTargets sample. */
36  public class ImageTargetsRenderer implements GLSurfaceView.Renderer
37  {
38          /**
39           * Server address
40           */
41          private static final String IP = new String("147.156.80.67");
42
43      public boolean mIsActive = false;
44
45      /** Native function for initializing the renderer. */
46      public native void initRendering();
47
48
49      /** Native function to update the renderer. */
50      public native void updateRendering(int width, int height);
51
52
53      /** Called when the surface is created or recreated. */
54      public void onSurfaceCreated(GL10 gl, EGLConfig config)
55      {
56          DebugLog.LOGD("GLRenderer::onSurfaceCreated");
57
58          // Call native function to initialize rendering:
59          initRendering();
60
61          // Call QCAR function to (re)initialize rendering after first use
62          // or after OpenGL ES context was lost (e.g. after onPause/onResume):
63          QCAR.onSurfaceCreated();
64      }
65
66
67      /** Called when the surface changed size. */
68      public void onSurfaceChanged(GL10 gl, int width, int height)
69      {
70          DebugLog.LOGD("GLRenderer::onSurfaceChanged");
71
72          // Call native function to update rendering when render surface
73           // parameters have changed:
74          updateRendering(width, height);
75
76          // Call QCAR function to handle render surface size changes:
77          QCAR.onSurfaceChanged(width, height);
78      }
79
80
81      /** The native render function. */
82      public native String renderFrame();
83
84
85      /** Called to draw the current frame. */
86      public void onDrawFrame(GL10 gl)
```

```
87      {
88          if (!mIsActive)
89              return;
90
91          // Call our native function to render content
92          //DebugLog.LOGD("haciendo el renderFrame...");
93          String aux;
94          //DebugLog.LOGD("buscando coordenadas...");
95          aux = renderFrame();
96          //DebugLog.LOGD("encontrando coordenadas...");
97          DebugLog.LOGD(aux);
98
99          sendServer(aux);
100     }
101
102     public void sendServer(String message)
103         {
104                 long time1=0, time2=0, result=0;
105                 try {
106             InetAddress serverAddr = InetAddress.getByName(IP);
107
108             Socket socket = new Socket(serverAddr, 4444);
109             //String message = new String("["+i+"]=("+x+","+y+")");
110
111                 try {
112                         DebugLog.LOGD("C: Sending"+message);
113                     DebugLog.LOGD("Envio1");
114                     PrintWriter out = new PrintWriter( new BufferedWriter( new
115  OutputStreamWriter(socket.getOutputStream())),true);
116                     //time1 = SystemClock.uptimeMillis();
117                     out.println(message);
118
119                 } catch(Exception e) {
120                         DebugLog.LOGD("C: Error"+e);
121                 }
122                 try {
123                         BufferedReader in = new BufferedReader(
124 new InputStreamReader(socket.getInputStream()));
125                     Boolean lee = true;
126                     String str = "";
127
128                         while(lee){
129                                 str = in.readLine();
130                                 if (str != null)
131                                         lee = false;
132                         }
133
134                     DebugLog.LOGD("recibi ["+str+"]");
135                     DebugLog.LOGD("Envio2");
136                     DebugLog.LOGD("Camara1");
137
138                         } catch (Exception e) {
139                                 DebugLog.LOGD("C: Error"+e);
140                         }
```

```
141
142                 } catch (Exception e) {
143                         DebugLog.LOGD("C: Error"+e);
144                 }
145         }
```

Por último, se muestra el fragmento que se ha modificado del fichero “ImageTargets.cpp”, en el que se han añadido las marcas de tiempo de las etapas de detección, dibujado y la segunda marca de la etapa de cámara.

```
1
2   // Coordenadas
3   float coordX    = 0.0;
4   float coordY    = 0.0;
5   float coordZ    = 0.0;
6
7   JNIEXPORT jstring JNICALL
8   Java_com_qualcomm_QCARSamples_ImageTargets_ImageTargetsRenderer_renderFrame(
9   JNIEnv *env, jobject obj)
10  {
11      //LOG("Java_com_qualcomm_QCARSamples_ImageTargets_GLRenderer_renderFrame");
12      // Clear color and depth buffer
13      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
14  LOG("Camara2");
15  LOG("Marca1");
16      // Get the state from QCAR and mark the beginning of a rendering section
17      QCAR::State state = QCAR::Renderer::getInstance().begin();
18
19      // Explicitly render the Video Background
20      QCAR::Renderer::getInstance().drawVideoBackground();
21  LOG("Marca2");
22  LOG("Render1");
23  #ifdef USE_OPENGL_ES_1_1
24      // Set GL11 flags:
25      glEnableClientState(GL_VERTEX_ARRAY);
26      glEnableClientState(GL_NORMAL_ARRAY);
27      glEnableClientState(GL_TEXTURE_COORD_ARRAY);
28
29      glEnable(GL_TEXTURE_2D);
30      glDisable(GL_LIGHTING);
31
32  #endif
33
34      glEnable(GL_DEPTH_TEST);
35      glEnable(GL_CULL_FACE);
36
37      // Did we find any trackables this frame?
38      for(int tIdx = 0; tIdx < state.getNumActiveTrackables(); tIdx++)
39      {
40          LOG("Dibujando %...",tIdx);
41          // Get the trackable:
42          const QCAR::Trackable* trackable = state.getActiveTrackable(tIdx);
43          QCAR::Matrix44F modelViewMatrix =
44              QCAR::Tool::convertPose2GLMatrix(trackable->getPose());
```

```
45              QCAR::Matrix44F modelViewMatrix2 =
46              QCAR::Tool::convertPose2GLMatrix(trackable->getPose());
47              QCAR::Matrix44F modelViewMatrix3 =
48              QCAR::Tool::convertPose2GLMatrix(trackable->getPose());
49              QCAR::Matrix44F modelViewMatrix4 =
50              QCAR::Tool::convertPose2GLMatrix(trackable->getPose());
51
52          // Choose the texture based on the target name:
53          int textureIndex;
54          if (strcmp(trackable->getName(), "chips") == 0)
55          {
56              textureIndex = 0;
57          }
58          else if (strcmp(trackable->getName(), "stones") == 0)
59          {
60              textureIndex = 1;
61          }
62          else
63          {
64              textureIndex = 2;
65          }
66
67          const Texture* const thisTexture = textures[textureIndex];
68
69  #ifdef USE_OPENGL_ES_1_1
70  LOG("Usa OpenGL 1.1");
71          // Load projection matrix:
72          glMatrixMode(GL_PROJECTION);
73          glLoadMatrixf(projectionMatrix.data);
74
75          // Load model view matrix:
76          glMatrixMode(GL_MODELVIEW);
77          glLoadMatrixf(modelViewMatrix.data);
78          glTranslatef(0.f, 0.f, kObjectScale);
79          glScalef(kObjectScale, kObjectScale, kObjectScale);
80
81          // Draw object:
82          glBindTexture(GL_TEXTURE_2D, thisTexture->mTextureID);
83          glTexCoordPointer(2, GL_FLOAT, 0, (const GLvoid*) &teapotTexCoords[0]);
84          glVertexPointer(3, GL_FLOAT, 0, (const GLvoid*) &teapotVertices[0]);
85          glNormalPointer(GL_FLOAT, 0,  (const GLvoid*) &teapotNormals[0]);
86          glDrawElements(GL_TRIANGLES, NUM_TEAPOT_OBJECT_INDEX, GL_UNSIGNED_SHORT,
87                         (const GLvoid*) &teapotIndices[0]);
88  #else
89  LOG("Usa OpenGL 2");
90
91          QCAR::Matrix44F modelViewProjection;
92
93          SampleUtils::translatePoseMatrix(50.0f, 0.0f, kObjectScale,
94                                           &modelViewMatrix.data[0]);
95
96          coordX = modelViewMatrix.data[12];
97          coordY = modelViewMatrix.data[13];
98          coordZ = modelViewMatrix.data[14];
```

```
99
100 //LOG("Coordenadas [%f,%f,%f]",coordX,coordY,coordZ);
101
102         SampleUtils::scalePoseMatrix(kObjectScale, kObjectScale, kObjectScale,
103                                      &modelViewMatrix.data[0]);
104         SampleUtils::multiplyMatrix(&projectionMatrix.data[0],
105                                     &modelViewMatrix.data[0] ,
106                                     &modelViewProjection.data[0]);
107
108         glUseProgram(shaderProgramID);
109
110         glVertexAttribPointer(vertexHandle, 3, GL_FLOAT, GL_FALSE, 0,
111                               (const GLvoid*) &teapotVertices[0]);
112         glVertexAttribPointer(normalHandle, 3, GL_FLOAT, GL_FALSE, 0,
113                               (const GLvoid*) &teapotNormals[0]);
114         glVertexAttribPointer(textureCoordHandle, 2, GL_FLOAT, GL_FALSE, 0,
115                               (const GLvoid*) &teapotTexCoords[0]);
116
117         glEnableVertexAttribArray(vertexHandle);
118         glEnableVertexAttribArray(normalHandle);
119         glEnableVertexAttribArray(textureCoordHandle);
120
121         glActiveTexture(GL_TEXTURE0);
122         glBindTexture(GL_TEXTURE_2D, thisTexture->mTextureID);
123         glUniformMatrix4fv(mvpMatrixHandle, 1, GL_FALSE,
124                            (GLfloat*)&modelViewProjection.data[0] );
125         glDrawElements(GL_TRIANGLES, NUM_TEAPOT_OBJECT_INDEX, GL_UNSIGNED_SHORT,
126                        (const GLvoid*) &teapotIndices[0]);
127
128         SampleUtils::checkGlError("ImageTargets renderFrame");
129 #endif
130
131     }
132
133     glDisable(GL_DEPTH_TEST);
134
135 #ifdef USE_OPENGL_ES_1_1
136     glDisable(GL_TEXTURE_2D);
137     glDisableClientState(GL_VERTEX_ARRAY);
138     glDisableClientState(GL_NORMAL_ARRAY);
139     glDisableClientState(GL_TEXTURE_COORD_ARRAY);
140 #else
141     glDisableVertexAttribArray(vertexHandle);
142     glDisableVertexAttribArray(normalHandle);
143     glDisableVertexAttribArray(textureCoordHandle);
144 #endif
145
146     QCAR::Renderer::getInstance().end();
147         LOG("Render2");
148         char buf[128];
149         sprintf(buf,"(%f,%f)",coordX,coordY);
150         jstring j_aux = env->NewStringUTF(buf);
151         //LOG("----- Saliendo... -----");
152         //coordX = 0.0;
```

```
153             //coordY = 0.0;
154             return j_aux;
155     }
```

G

Código del servidor de la caracterización de clientes CAR

A continuación se expone el código fuente del servidor utilizado en la caracterización de los clientes, llamado *EchoServer*, porque se limita a responder al cliente con la misma información que recibe.

Fichero EchoServer.java

```
1  import java.io.BufferedReader;
2  import java.io.BufferedWriter;
3  import java.io.InputStreamReader;
4  import java.io.OutputStreamWriter;
5  import java.io.PrintWriter;
6  import java.net.ServerSocket;
7  import java.net.Socket;
8
9
10 public class EchoServer implements Runnable{
11
```

```
12      public static final String SERVERIP = "192.168.0.17";
13      public static final int SERVERPORT = 4444;
14
15      public void run() {
16          try {
17              System.out.println("S: Connecting...");
18              ServerSocket serverSocket = new ServerSocket(SERVERPORT);
19
20              while (true) {
21                  Socket client = serverSocket.accept();
22                  System.out.println("S: Receiving...");
23
24                  try {
25
26                      BufferedReader in = new BufferedReader(new
27   InputStreamReader(client.getInputStream()));
28                      String str = in.readLine();
29                      System.out.println("S: Received: '" + str + "'");
30
31                      PrintWriter out = new PrintWriter(new BufferedWriter(new
32   OutputStreamWriter(client.getOutputStream())), true);
33                      out.println("S: Sending: "+str);
34                      System.out.println("S: Sending back: '" + str + "'");
35
36                  } catch(Exception e) {
37                         System.out.println("S: Error");
38                         e.printStackTrace();
39                  } finally {
40                         client.close();
41                         System.out.println("S: Done.");
42                     }
43              }
44
45          } catch (Exception e) {
46              System.out.println("S: Error");
47              e.printStackTrace();
48          }
49      }
50
51      public static void main (String a[]) {
52
53          Thread desktopServerThread = new Thread(new TCPDesktopServer());
54          desktopServerThread.start();
55      }
56  }
```

H

Sobre la utilización del simulador CAR

La cantidad máxima de ficheros abiertos en un computador es un parámetro del sistema que se puede comprobar con la orden de Linux *Ulimit -n*. Como las pruebas utilizan miles de clientes y cada cliente tiene varios hilos y ficheros de estadísticas y demás, se ha determinado que como mínimo son necesarios unos 4000 ficheros abiertos. El comando anterior consulta el fichero */etc/security/limits.conf* para establecer los límites a los que puedes llegar, por lo que si necesitas más ficheros abiertos de los que te permite *Ulimit* tienes que modificar ese fichero, concretamente las líneas referentes a tu usuario con la opción *nofile*. A continuación se muestra un ejemplo poniendo el valor mínimo (*soft*) a 4000 y el máximo (*hard*) a 8000.

```
bauset   soft     nofile   4000
bauset   hard     nofile   8000
```

Se ha comentado que el servidor se ejecuta en un computador y los clientes repartidos en otros dos computadores, por lo que se tiene que ejecutar los procesos adecuados en tres máquinas diferentes antes de poder ejecutar la prueba. Para poder realizar las pruebas cómodamente se ha hecho un programa en Python (llamado “lanzador.py”) que se conecta remótamente a las máquinas elegidas y lanza en ellas la simulación del sistema con los parámetros elegidos, mostrando los datos de la simulación al finalizar la prueba. Con este programa se consigue lanzar una única prueba fácilmente pero, como se tiene un gran número de pruebas pendientes de ejecutar en las que se evaluará el comportamiento del sistema conforme se varían los parámetros del mismo, se ha realizado otro programa Python (llamado “multiple.py”) que lanza toda una serie de pruebas usando como base el programa anterior. A este programa se le pasa la cantidad de clientes inicial (múltiplo de 50), la cantidad de clientes final, el incremento de clientes en cada prueba, la frecuencia de actuación de los clientes y el tamaño del grupo de trabajo. Un ejemplo de ejecución sería:

```
./multiple.py 100 1000 100 698.34 5
```

Con esta orden se ejecutaría pruebas desde 100 clientes hasta 1000, de 100 en 100, con la frecuencia de actuación del Motorola Milestone (el más lento) y grupos de trabajo de 5 vecinos.

Resto de tablas de la evaluación de prestaciones

A continuación se muestran el resto de tablas que no aparecen en la evaluación de prestaciones.

Las Tablas I.1, I.2 y I.3 muestran la evaluación de prestaciones del servidor activo, para grupos de trabajo de 5, 20 y 25 clientes, respectivamente.

	One			Milestone		
WG 5	**RT**	**Dev**	**CPU**	**RT**	**Dev**	**CPU**
100	60,82	22,08	17	60,91	22,95	6,9
200	59,83	20,5	13	62,45	21,15	9
300	63,55	20,83	20	64,4	20,6	11
400	66,71	20,77	30	67,33	21,36	13
500	68,77	21,62	38	67,84	21,02	18,8
600	70,39	23,1	49,5	68,84	21,64	19
700	70,92	23,45	56	68,14	21,74	24
800	68,97	22,44	60,4	66,01	20,74	25
900	68,61	22,92	69	68,03	21,51	27,7
1000	70,84	24,51	72	67,91	21,6	31

Tabla I.1: Rendimiento del sistema para un grupo de trabajo de 5 vecinos en ambos teléfonos móviles para el servidor *activo*

	One			Milestone		
WG 20	**RT**	**Dev**	**CPU**	**RT**	**Dev**	**CPU**
100	77,37	23,26	23,8	78,65	20,81	12,8
200	77,13	19,95	38	77,69	17,25	16
300	79,59	17,17	56,6	84,26	17,78	35,4
400	83,07	25,81	74,2	96,71	24,51	38
500	195,92	63,28	82,9	81,19	21,24	40
600	246,46	86,01	82,8	103,62	39,43	46
700	297,3	101,41	82,8	97,92	26,56	47
800	350,88	122,83	84,2	82,63	26,52	62,4
900	394,66	142,36	86	93,31	30,57	56,5
1000	410,14	181,39	87	88,52	26,54	66,3

Tabla I.2: Rendimiento del sistema para un grupo de trabajo de 20 vecinos en ambos teléfonos móviles para el servidor *activo*

	One			Milestone		
WG 25	**RT**	**Dev**	**CPU**	**RT**	**Dev**	**CPU**
100	79,94	19,4	32	77,77	16,28	13,3
200	77,94	17	44,5	78,48	18,31	19,2
300	104,03	23,48	66	88,55	24,19	31
400	178,11	51,51	83	97,39	31,33	36,6
500	238,16	72,82	82,8	104,62	43,89	47
600	304,12	92,3	84,9	124,26	42,17	61,2
700	358,98	115,61	85,1	108,87	45,85	65,3
800	417,96	133,95	85,7	107,1	50,06	62,4
900	473,13	155,19	87,9	116,88	60,37	71
1000	524,35	191,4	88	109,12	55,79	79

Tabla I.3: Rendimiento del sistema para un grupo de trabajo de 25 vecinos en ambos teléfonos móviles para el servidor *activo*

Las Tablas I.4, I.5, I.6 y I.7 muestran las pruebas para el servidor *pasivo* utilizando el Motorola Milestone y corresponden a los grupos de trabajo de 5, 10, 20 y 25, respectivamente.

	Motorola Milestone pasivo				
WG 5	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	64,92	23,49	8	20,77	22,12
200	64,51	23,32	9,1	21,37	24,44
300	67,47	23,14	26,3	25,79	28
400	70,84	23,81	65	27,49	29,68
500	71,89	24,53	65	26,81	29,24
600	70,23	23,91	21,8	26,28	28,66
700	74,24	24,44	44	27,76	32,21
800	70,34	23,81	28	26,53	32
900	71,36	24,4	30,7	26,72	35,45
1000	70,77	24,16	34	26,26	29,76

Tabla I.4: Tiempo de respuesta en el servidor (*pasivo*) para un grupo de trabajo de 5 vecinos para el Motorola Milestone

	Motorola Milestone pasivo				
WG 10	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	77,03	27,63	9,9	19,55	21,14
200	78,16	26,07	26	19,77	23,54
300	83,6	28,61	25	28,2	31,21
400	82,54	23,76	23,8	27,24	29,35
500	88,27	26,76	31,3	28,98	37,14
600	87,66	27,45	33	27,21	33,17
700	88,45	28,58	33,7	27,64	35,46
800	94,43	34,05	62	29,97	39,38
900	84,8	25,7	69,3	26,28	34,03
1000	87,72	27,56	46,5	26,47	31,77

Tabla I.5: Tiempo de respuesta en el servidor (*pasivo*) para un grupo de trabajo de 10 vecinos para el Motorola Milestone

	Motorola Milestone pasivo				
WG 20	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	93,88	26,9	20,2	24,77	26,96
200	94,98	25,74	21,2	27,3	31,77
300	98,01	24,79	29,7	29,26	32,21
400	129,24	30,8	37,7	45,11	59,71
500	101,96	30,91	43	28,45	34,19
600	121,53	39,5	50,5	36,62	47,13
700	148,84	66,52	83,9	46,07	85,93
800	110,41	39,07	65,3	30,27	43,94
900	101,4	32,33	67,3	29,5	50,46
1000	106,66	37,8	74,7	30,93	39,93

Tabla I.6: Tiempo de respuesta en el servidor (*pasivo*) para un grupo de trabajo de 20 vecinos para el Motorola Milestone

	Motorola Milestone pasivo				
WG 25	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	96,19	25,12	25	25,09	26,33
200	96,3	24,83	47,1	26,87	28,06
300	120,52	26,36	43,6	43,42	51,63
400	125,16	39,86	53,3	39,28	51,3
500	144,87	64,99	56	44,94	56,22
600	135,65	52,91	57	38,51	58,86
700	112,33	36,48	60,7	30,53	37,38
800	184,17	107,43	77,8	52,39	103,97
900	126,64	61,98	78	37	51,26
1000	131,47	63,24	81	38,14	55,34

Tabla I.7: Tiempo de respuesta en el servidor (*pasivo*) para un grupo de trabajo de 25 vecinos para el Motorola Milestone

Las Tablas I.8, I.9, I.10 y I.11 muestran las pruebas para el servidor *pasivo* utilizando el Nexus One y corresponden a los grupos de trabajo de 5, 10, 20 y 25, respectivamente.

	Nexus One pasivo				
WG 5	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	62,37	22,68	9,9	19,36	20,9
200	63,77	22,21	15	20,12	22,43
300	66,71	22,66	22	25,15	28,28
400	68,68	22,5	32,7	27,14	29,56
500	71,04	23,56	45	27,14	30,9
600	71,5	24,18	48,6	26,58	31,95
700	72,37	25,01	59	27,17	35,42
800	72,85	26,01	68	27,87	34,54
900	75,01	28,98	79,2	28,61	35,89
1000	147,33	101,71	85	43,95	48,66

Tabla I.8: Tiempo de respuesta en el servidor (*pasivo*) para un grupo de trabajo de 5 vecinos para el Nexus One

	Nexus One pasivo				
WG 10	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	78,23	30,41	16	18,2	20,15
200	74,02	25,06	25,2	18,5	24,9
300	79,45	24,57	37	25,78	28,62
400	81,02	24,05	59,6	27,71	30,4
500	93,61	28,56	68,7	31,83	40,51
600	147,73	100,48	83,8	40,93	44,37
700	195,47	76,24	83,2	49,85	52,39
800	237,71	91,09	85,9	58,08	62,23
900	270,25	104,97	84,2	64,79	68,9
1000	300,96	132,04	84,2	68,77	75,48

Tabla I.9: Tiempo de respuesta en el servidor (*pasivo*) para un grupo de trabajo de 10 vecinos para el Nexus One

	Nexus One pasivo				
WG 20	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	88,22	32,16	27,6	19,17	22,83
200	94,57	29,06	45,5	25,56	32,77
300	128,16	28,88	73,3	45,23	50,87
400	195,92	63,42	84,2	43,15	47,4
500	273,58	89,08	84	59,82	63,94
600	326,79	106,94	86,2	70,21	76,9
700	400,45	142,76	87	76,44	86,92
800	447,02	155,46	87	90,43	96,68
900	473,99	223,98	86	89,56	94,92
1000	467,04	286,28	86,9	80,15	88,09

Tabla I.10: Tiempo de respuesta en el servidor (*pasivo*) para un grupo de trabajo de 20 vecinos para el Nexus One

	Nexus One pasivo				
WG 25	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	96,03	25,91	60,4	24,26	26,28
200	103,19	38,71	83	30,98	34,53
300	167,58	50,19	86,1	40,24	43,24
400	250,53	77,68	85	49,89	54,97
500	357,13	126,19	92	74,41	78,1
600	496,62	218,79	94,1	96,25	104,29
700	480,87	156,16	85,3	93,42	100,18
800	531,93	210,02	86	97,47	107,15
900	505,89	300,11	89	93,45	102,97
1000	524,96	372,62	87,1	82,39	96,23

Tabla I.11: Tiempo de respuesta en el servidor (*pasivo*) para un grupo de trabajo de 25 vecinos para el Nexus One

Las Tablas I.12, I.13, I.14 y I.15 muestran el tiempo de respuesta en el servidor *activo* sobre el Motorola Milestone para grupos de trabajo de 5, 10, 20 y 25 clientes, respectivamente.

	Motorola Milestone activo				
WG 5	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	60,91	22,95	6,9	24,28	28,18
200	62,45	21,15	9	23,75	30,91
300	64,4	20,6	11	25,24	28,58
400	67,33	21,36	13	27,2	31,12
500	67,84	21,02	18,8	28,08	35,99
600	68,84	21,64	19	27,42	32,67
700	68,14	21,74	24	27,08	30,36
800	66,01	20,74	25	26,42	30,37
900	68,03	21,51	27,7	26,99	32,67
1000	67,91	21,6	31	27	33,41

Tabla I.12: Tiempo de respuesta en el servidor (*activo*) para un grupo de trabajo de 5 vecinos para el Motorola Milestone

	Motorola Milestone activo				
WG 10	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	72,89	28,02	7,1	19,63	20,49
200	69,35	23,92	13	17,71	22,5
300	73,67	18,43	17,8	26,16	30,1
400	78,13	23,07	18,8	25,1	31,38
500	75,9	19,04	26	25,95	29,28
600	80,35	26,04	31,7	27,09	33,51
700	81,73	26,06	31	26,58	34,59
800	78,69	23,82	40,8	25,5	38,22
900	76,32	20,47	43,9	24,21	31,13
1000	78,29	21,99	45,1	25,83	36,23

Tabla I.13: Tiempo de respuesta en el servidor (*activo*) para un grupo de trabajo de 10 vecinos para el Motorola Milestone

	Motorola Milestone activo				
WG 20	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	78,65	20,81	12,8	16,84	17,99
200	77,69	17,25	16	22,86	30,96
300	84,26	17,78	35,4	30,79	36,26
400	96,71	24,51	38	32,77	37,39
500	81,19	21,24	40	28,81	33,09
600	103,62	39,43	46	31,9	41,46
700	97,92	26,56	47	32,83	45,56
800	82,63	26,52	62,4	26,84	33,44
900	93,31	30,57	56,5	30,24	41,98
1000	88,52	26,54	66,3	25,72	34,21

Tabla I.14: Tiempo de respuesta en el servidor (*activo*) para un grupo de trabajo de 20 vecinos para el Motorola Milestone

	Motorola Milestone activo				
WG 25	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	77,77	16,28	13,3	21,32	23,97
200	78,48	18,31	19,2	27,51	32,37
300	88,55	24,19	31	28,79	33,46
400	97,39	31,33	36,6	31,72	40,48
500	104,62	43,89	47	31,2	37,1
600	124,26	42,17	61,2	42,25	58,62
700	108,87	45,85	65,3	36,73	44,19
800	107,1	50,06	62,4	35,51	53,67
900	116,88	60,37	71	35,16	56,56
1000	109,12	55,79	79	34,75	53,66

Tabla I.15: Tiempo de respuesta en el servidor (*activo*) para un grupo de trabajo de 25 vecinos para el Motorola Milestone

Las Tablas I.16, I.17 y I.18 muestran el tiempo de respuesta en el servidor *activo* sobre el Nexus One para grupos de trabajo de 5, 20 y 25 clientes, respectivamente.

	Nexus One activo				
WG 5	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	60,82	22,08	17	21,27	24,48
200	59,83	20,5	13	19,83	25,35
300	63,55	20,83	20	24,38	28
400	66,71	20,77	30	27,42	32,44
500	68,77	21,62	38	27,26	32,04
600	70,39	23,1	49,5	28,47	39,78
700	70,92	23,45	56	28,4	37,18
800	68,97	22,44	60,4	28,26	34,84
900	68,61	22,92	69	28,48	37,62
1000	70,84	24,51	72	28,32	37,28

Tabla I.16: Tiempo de respuesta en el servidor (*activo*) para un grupo de trabajo de 5 vecinos para el Nexus One

	Nexus One activo				
WG 20	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	77,37	23,26	23,8	16,67	19,68
200	77,13	19,95	38	22,27	34
300	79,59	17,17	56,6	26,64	32,02
400	83,07	25,81	74,2	29,14	32,58
500	195,92	63,28	82,9	26,69	31,07
600	246,46	86,01	82,8	30,26	34,65
700	297,3	101,41	82,8	34,03	39,09
800	350,88	122,83	84,2	39,41	45,17
900	394,66	142,36	86	43,64	50,8
1000	410,14	181,39	87	50,86	57,13

Tabla I.17: Tiempo de respuesta en el servidor (*activo*) para un grupo de trabajo de 20 vecinos para el Nexus One

	Nexus One activo				
WG 25	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	79,94	19,4	32	18,96	21,39
200	77,94	17	44,5	28,09	33,47
300	104,03	23,48	66	35,11	42,08
400	178,11	51,51	83	25,88	28,02
500	238,16	72,82	82,8	28,84	33,4
600	304,12	92,3	84,9	32,34	37,55
700	358,98	115,61	85,1	35,98	42,81
800	417,96	133,95	85,7	40,76	45,84
900	473,13	155,19	87,9	45,84	52,94
1000	524,35	191,4	88	57,49	71,36

Tabla I.18: Tiempo de respuesta en el servidor (*activo*) para un grupo de trabajo de 25 vecinos para el Nexus One

Las Tablas I.19 y I.20 muestran la comparación entre la implementación inicial y la implementación “select” sobre el Nexus One, al variar sus grupos de trabajo a valores de 10 y 20, respectivamente.

ONE	TCP base					Select				
WG 10	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	78,23	30,41	16	18,2	20,15	77,93	28,16	8	17,71	19,24
200	74,02	25,06	25,2	18,5	24,9	76,77	21,69	21,5	23,39	27,76
300	79,45	24,57	37	25,78	28,62	76,42	21,74	32	24,16	28,61
400	81,02	24,05	59,6	27,71	30,4	86,01	25,26	46	29,7	35,67
500	93,61	28,56	68,7	31,83	40,51	91,77	30,8	52	29,2	38,75
600	147,73	100,48	83,8	40,93	44,37	97,62	34,65	67,7	33,87	48,1
700	195,47	76,24	83,2	49,85	52,39	111,58	36,6	82	44,71	53,14
800	237,71	91,09	85,9	58,08	62,23	177,23	50,66	89,9	54,34	59,09
900	270,25	104,97	84,2	64,79	68,9	214,09	64,17	87	64,23	68,56
1000	300,96	132,04	84,2	68,77	75,48	249,27	65,51	86	79,76	82,48

Tabla I.19: Rendimiento del sistema para el Nexus One con un grupo de trabajo de 10 vecinos. Implementación base y mejora select

ONE	TCP base					Select				
WG 20	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	88,22	32,16	27,6	19,17	22,83	91,7	27	21,6	18,85	19,81
200	94,57	29,06	45,5	25,56	32,77	84,03	19,92	39	27,93	31,36
300	128,16	28,88	73,3	45,23	50,87	102,75	28,86	61,4	42,85	51,05
400	195,92	63,42	84,2	43,15	47,4	121,89	36,04	87,1	50,24	60,11
500	273,58	89,08	84	59,82	63,94	214,52	43,43	88	86,52	90,08
600	326,79	106,94	86,2	70,21	76,9	261,65	49,07	89,9	108,72	113,13
700	400,45	142,76	87	76,44	86,92	306,15	60,33	87	122,88	136,73
800	447,02	155,46	87	90,43	96,68	337,05	79,58	89,1	99,43	107,46
900	473,99	223,98	86	89,56	94,92	384,47	81,41	88,2	112,63	115,05
1000	467,04	286,28	86,9	80,15	88,09	438,94	95,03	91	129,32	131,82

Tabla I.20: Rendimiento del sistema para el Nexus One con un grupo de trabajo de 20 vecinos. Implementación base y mejora select

Las Tablas I.21, I.22 y I.23 corresponden a la evaluación de prestaciones del sistema con la mejora *select*, sobre el Nexus One y para un WG de 5 al cambiar el valor de SPT a 2, 4 y 6, respectivamente.

ONE	2 SPT				
WG 5	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	65,24	21,79	7	20,25	22,48
200	67,45	22,77	12,9	21,8	23,81
300	66,93	22,58	16,2	24,41	27,38
400	68,51	22,88	22,8	26,03	29,08
500	69,95	23,68	30,7	26,54	30,23
600	70,2	25,75	44,5	23,69	28,62
700	69,06	23,77	49	23,24	29,04
800	69,94	24,23	54	24,22	30,75
900	74,31	27,67	64,3	25,05	34,15
1000	74,85	28,68	66,7	25,68	33,79

Tabla I.21: Tiempo de respuesta en la implementación *select* para un grupo de trabajo de 5 vecinos sobre el Nexus One con 2 SPT

ONE	4 SPT				
WG 5	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	65,75	21,85	10,2	20,55	22,44
200	67,23	22,8	14,2	22,02	23,89
300	66,3	22,69	17,8	22,09	27,62
400	68,05	22,75	22,2	25,1	28,98
500	69,34	22,72	34,3	26,01	28,85
600	71,94	24,41	38,4	26,96	32,69
700	69,18	23,57	51	24,32	29,54
800	70,93	24,61	55	25,03	30,89
900	70,41	25,27	65	23,01	29,77
1000	71,17	26,33	69	23,25	29,92

Tabla I.22: Tiempo de respuesta en la implementación *select* para un grupo de trabajo de 5 vecinos sobre el Nexus One con 4 SPT

ONE	6 SPT				
WG 5	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	66,05	21,86	10	20,45	22,28
200	66,84	22,79	17,2	21,92	24,3
300	66,99	22,69	22,8	23,59	27,62
400	68,07	22,78	23,2	24,97	27,56
500	68,82	23,23	34	24,22	29,34
600	72,19	24,78	39	26,75	30,98
700	70,28	24,11	53	23,82	29,46
800	70,7	24,11	56,5	24,05	28,85
900	72,22	26,2	65,6	24,38	30,66
1000	75,26	30,24	79	25,61	32,81

Tabla I.23: Tiempo de respuesta en la implementación *select* para un grupo de trabajo de 5 vecinos sobre el Nexus One con 6 SPT

Las Tablas I.24, I.25 y I.26 corresponden a la evaluación de prestaciones del sistema con la mejora *select*, sobre el Nexus One y para un WG de 10 al cambiar el valor de SPT a 2, 4 y 6, respectivamente.

ONE	2 SPT				
WG 10	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	74,59	26,86	10,8	15,97	17,25
200	76,68	22,54	20,2	23,04	26,31
300	79,21	22,31	31	26,41	30,64
400	79,98	22,88	45	25,6	34,61
500	87,95	28,37	54	31,67	46,8
600	88,54	29,88	69	28,92	48,29
700	115,3	34,39	85	45,29	52,64
800	186,6	48,99	84	61,05	68,86
900	215,6	59,38	87,9	66,79	69,69
1000	246,2	60,92	89	77,24	79,8

Tabla I.24: Tiempo de respuesta en la implementación *select* para un grupo de trabajo de 10 vecinos sobre el Nexus One con 2 SPT

ONE	4 SPT				
WG 10	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	81,41	29,1	13,1	18,12	19,6
200	76,85	22,75	21,2	22,98	26,93
300	77,74	22,69	36	24,2	30,09
400	81,16	24,04	43	25,01	36,52
500	92,22	31,67	53	31,6	41,15
600	86,4	28,75	68	28,81	39,73
700	144,4	41,78	88	57,89	62,14
800	184,1	45,86	87,9	60,55	70,13
900	217	58,62	89,9	67,31	69,96
1000	248,8	73,38	88,9	78,98	82,38

Tabla I.25: Tiempo de respuesta en la implementación *select* para un grupo de trabajo de 10 vecinos sobre el Nexus One con 4 SPT

ONE	6 SPT				
WG 10	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	82,35	29,11	14,2	18,51	20,01
200	77,14	22,62	38	24,05	27,76
300	84,88	45,76	75,2	27,17	30,96
400	88,81	28	74,8	30,27	42,64
500	93,13	32,35	66	34,34	46,03
600	118,9	36,97	78	50,29	62,95
700	165,4	44,39	87	61,96	64,97
800	195,1	47,56	89,9	66,94	74,9
900	222	57,18	90,9	68,54	71,33
1000	252	54,42	88	79,95	82,04

Tabla I.26: Tiempo de respuesta en la implementación *select* para un grupo de trabajo de 10 vecinos sobre el Nexus One con 6 SPT

Las Tablas I.27, I.28 y I.29 corresponden a la evaluación de prestaciones del sistema con la mejora *select*, sobre el Nexus One y para un WG de 20 al cambiar el valor de SPT a 2, 4 y 6, respectivamente.

ONE	2 SPT				
WG 20	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	88,99	26,8	23	19,14	21,79
200	85,88	20,01	41	29,97	33,95
300	110,5	28,65	67,3	47,22	56,94
400	162,7	35,71	89	65,35	67,82
500	220,3	36,47	88	91,29	93,44
600	271,6	46,4	87	115	119,14
700	320,7	46,9	88,9	137,7	143,63
800	355,1	79,89	90	112,6	126,48
900	392,2	79,85	89	116,6	119,26
1000	445	94,41	91,9	132,6	141,16

Tabla I.27: Tiempo de respuesta en la implementación *select* para un grupo de trabajo de 20 vecinos sobre el Nexus One con 2 SPT

ONE	4 SPT				
WG 20	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	87,15	25,58	21,2	19,99	22,16
200	89,88	22,19	43,9	32,28	38,27
300	115	29,52	69	50,02	58,81
400	173,1	30,88	88	68,95	70,92
500	223,3	36,09	90,1	92,57	95
600	275,4	40,25	87,9	118,3	121,66
700	326,7	52,81	88,9	140,6	144,37
800	360,9	75,08	89	119,6	138,78
900	395,8	82,54	92	120,1	131,55
1000	454,6	95,25	90,1	135,7	158,75

Tabla I.28: Tiempo de respuesta en la implementación *select* para un grupo de trabajo de 20 vecinos sobre el Nexus One con 4 SPT

ONE	6 SPT				
WG 20	**RT**	**Dev**	**CPU**	**SRT**	**MaxSRT**
100	88,05	24,85	24	20,42	21,39
200	89,89	21,35	42,5	32,27	38,06
300	114,7	30,82	71	47,57	58,49
400	171,1	28,86	87	68,09	69,63
500	225,1	33	86,7	94,92	97,11
600	276,9	39,26	88,2	119,7	123,27
700	324,4	50,79	87,8	137	145,77
800	369,7	76,09	89	125,6	140,49
900	403,2	83,19	90	121,8	150,55
1000	463,9	95,14	91	140,8	149,16

Tabla I.29: Tiempo de respuesta en la implementación *select* para un grupo de trabajo de 20 vecinos sobre el Nexus One con 6 SPT

Las Tablas I.30 y I.31 muestran el rendimiento de la implementación UDP para el Nexus One con los grupos de trabajo de 10 y 20 clientes, respectivamente.

WG 10	RT	Dev	CPU	SRT	MaxSRT			
100	1,93	0,44	22,70	0,84	0,92			
200	2,47	1,27	36,90	1,00	1,53			
300	3,79	2,89	50,00	1,35	2,57			
400	4,28	3,34	62,10	1,50	2,70			
500	6,52	6,17	77,00	2,20	4,62			
600	9,82	12,57	91,90	3,34	8,13			
700	42,54	46,60	93,00	74,41	81,81			
800	54,84	54,39	93,01	88,38	97,74			
900	64,85	72,84	93,20	98,58	122,63			
1000	81,26	103,61	94,00	104,04	117,35			
	% pérdidas servidor				**% pérdidas clientes**			
	POS	**FIN**	**ACK**	**TRC**	**POS**	**FIN**	**ACK**	**CTR**
100	0,00	0,00	0,00	0,01	0,00	0,00	0,00	0,00
200	0,00	0,00	0,01	0,04	0,00	0,00	0,01	0,00
300	0,00	0,00	0,01	0,09	0,00	0,00	0,01	0,00
400	0,00	0,00	0,06	0,14	0,00	0,00	0,06	0,00
500	0,00	0,00	0,10	0,19	0,00	0,00	0,10	0,00
600	0,00	0,00	0,18	0,38	0,00	0,00	0,18	0,00
700	0,00	0,00	7,03	91,60	0,00	0,00	7,03	0,00
800	0,00	0,00	14,76	96,15	0,00	0,00	14,76	0,00
900	0,00	0,00	24,71	96,68	0,00	0,00	24,71	0,00
1000	0,00	0,00	25,90	97,58	0,00	0,00	25,90	0,00

Tabla I.30: Rendimiento del sistema para el Nexus One con un grupo de trabajo de 10 clientes con la implementación UDP

WG 20	RT	Dev	CPU	SRT	MaxSRT			
100	2,94	1,30	32,30	1,11	1,21			
200	6,15	5,91	60,00	2,11	2,67			
300	13,48	12,08	84,80	5,57	7,60			
400	83,71	49,92	93,00	83,65	92,74			
500	78,96	55,84	93,00	87,04	96,82			
600	84,01	53,51	93,40	88,54	108,31			
700	84,15	54,56	93,42	89,70	98,39			
800	97,61	67,05	93,60	90,62	102,50			
900	170,26	170,65	94,00	99,07	109,81			
1000	260,26	206,57	94,01	102,81	119,73			
	% pérdidas servidor				**% pérdidas clientes**			
	POS	**FIN**	**ACK**	**TRC**	**POS**	**FIN**	**ACK**	**CTR**
	0,00	0,00	0,00	0,02	0,00	0,00	0,00	0,00
	0,00	0,00	0,03	0,08	0,00	0,00	0,03	0,00
	0,00	0,00	0,20	0,33	0,00	0,00	0,20	0,00
	0,00	0,00	10,55	95,04	0,00	0,00	10,55	0,00
	0,00	0,00	22,72	97,56	0,00	0,00	22,72	0,00
	0,00	0,00	35,23	98,14	0,00	0,00	35,23	0,00
	0,00	0,00	40,50	98,48	0,00	0,00	40,50	0,00
	0,00	0,00	44,83	98,84	0,00	0,00	44,83	0,00
	0,00	0,00	50,46	99,00	0,00	0,00	50,46	0,00
	0,00	0,00	51,72	99,18	0,50	6,06	52,04	0,54

Tabla I.31: Rendimiento del sistema para el Nexus One con un grupo de trabajo de 20 clientes con la implementación UDP

Las Tablas I.32, I.33, I.34 y I.35 muestran el rendimiento del Motorola Milestone para la implementación UDP con todos los grupos de trabajo considerados: 5, 10, 20 y 25, respectivamente.

WG 5	RT	Dev	CPU	SRT	MaxSRT			
100	1,64	1,93	5,00	0,77	0,84			
200	2,00	2,92	9,90	0,88	0,96			
300	2,30	4,56	15,20	1,00	1,12			
400	2,73	5,74	15,50	1,18	1,31			
500	3,27	7,57	17,90	1,26	1,53			
600	3,50	12,34	21,70	1,39	1,88			
700	3,58	12,28	25,80	1,50	1,96			
800	3,79	13,59	26,80	1,56	2,01			
900	4,25	14,75	29,50	1,71	2,04			
1000	4,26	15,29	29,50	1,79	2,07			
	% pérdidas servidor				**% pérdidas clientes**			
	POS	**FIN**	**ACK**	**TRC**	**POS**	**FIN**	**ACK**	**CTR**
100	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
200	0,00	0,00	0,00	0,01	0,00	0,00	0,00	0,00
300	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
400	0,00	0,00	0,00	0,02	0,00	0,00	0,00	0,00
500	0,00	0,00	0,01	0,05	0,00	0,00	0,01	0,00
600	0,00	0,00	0,00	0,03	0,00	0,00	0,00	0,00
700	0,00	0,00	0,01	0,02	0,00	0,00	0,01	0,00
800	0,00	0,00	0,00	0,02	0,00	0,00	0,00	0,00
900	0,00	0,00	0,01	0,04	0,00	0,00	0,01	0,00
1000	0,00	0,00	0,00	0,02	0,00	0,00	0,00	0,00

Tabla I.32: Rendimiento del sistema para el Motorola Milestone con un grupo de trabajo de 5 clientes con la implementación UDP

WG 10	RT	Dev	CPU	SRT	MaxSRT			
100	2,10	2,84	12,10	0,96	1,01			
200	3,01	5,25	13,30	1,24	1,33			
300	3,79	5,53	19,00	1,44	1,62			
400	4,34	9,32	28,10	1,76	1,91			
500	4,96	10,33	29,90	2,00	2,24			
600	6,39	14,96	59,40	2,32	2,61			
700	6,98	18,99	36,40	2,50	2,87			
800	7,21	17,54	38,50	2,84	3,27			
900	7,87	19,57	46,90	3,17	3,84			
1000	12,66	35,24	53,10	3,89	4,92			
	% pérdidas servidor				**% pérdidas clientes**			
	POS	**FIN**	**ACK**	**TRC**	**POS**	**FIN**	**ACK**	**CTR**
100	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
200	0,00	0,00	0,00	0,01	0,00	0,00	0,00	0,00
300	0,00	0,00	0,07	0,13	0,00	0,00	0,00	0,00
400	0,00	0,00	0,01	0,03	0,00	0,00	0,01	0,00
500	0,00	0,00	0,00	0,05	0,00	0,00	0,00	0,00
600	0,00	0,00	0,01	0,04	0,00	0,00	0,01	0,00
700	0,00	0,00	0,00	0,05	0,00	0,00	0,00	0,00
800	0,00	0,00	0,02	0,06	0,00	0,00	0,02	0,00
900	0,00	0,00	0,01	0,06	0,00	0,00	0,01	0,00
1000	0,00	0,00	0,02	0,06	0,00	0,00	0,02	0,00

Tabla I.33: Rendimiento del sistema para el Motorola Milestone con un grupo de trabajo de 10 clientes con la implementación UDP

WG 20	RT	Dev	CPU	SRT	MaxSRT			
100	3,11	4,17	14,30	1,37	1,46			
200	5,15	6,23	21,60	2,15	2,33			
300	7,73	7,40	32,70	2,76	3,30			
400	9,33	8,87	39,10	3,76	4,57			
500	12,48	10,10	47,90	4,20	4,95			
600	14,24	13,20	55,70	5,62	7,16			
700	18,72	13,75	65,00	6,28	8,98			
800	27,46	24,01	81,00	9,68	14,32			
900	27,72	27,56	81,50	9,05	11,89			
1000	43,02	39,69	83,30	11,02	14,84			
	% pérdidas servidor				**% pérdidas clientes**			
	POS	**FIN**	**ACK**	**TRC**	**POS**	**FIN**	**ACK**	**CTR**
100	0,00	0,00	0,00	0,01	0,00	0,00	0,00	0,00
200	0,00	0,00	0,00	0,04	0,00	0,00	0,00	0,00
300	0,00	0,00	0,05	0,09	0,00	0,00	0,05	0,00
400	0,00	0,00	0,02	0,05	0,00	0,00	0,02	0,00
500	0,00	0,00	0,07	0,12	0,00	0,00	0,07	0,00
600	0,00	0,00	0,02	0,06	0,00	0,00	0,02	0,00
700	0,00	0,00	0,05	0,11	0,00	0,00	0,05	0,00
800	0,00	0,00	0,04	0,25	0,00	0,00	0,04	0,00
900	0,00	0,00	0,10	0,22	0,00	0,00	0,10	0,00
1000	0,00	0,00	0,11	0,49	0,00	0,00	0,11	0,00

Tabla I.34: Rendimiento del sistema para el Motorola Milestone con un grupo de trabajo de 20 clientes con la implementación UDP

WG 25	RT	Dev	CPU	SRT	MaxSRT			
100	4,00	5,17	16,30	1,61	1,66			
200	5,96	6,86	25,80	2,48	2,58			
300	13,14	45,13	47,50	4,41	4,79			
400	12,69	9,50	49,50	4,43	5,94			
500	24,04	35,14	90,90	10,55	12,36			
600	27,05	46,88	92,00	10,74	13,30			
700	36,08	59,29	92,30	22,15	25,53			
800	48,63	67,22	92,40	13,94	17,70			
900	137,88	122,65	93,00	27,41	35,40			
1000	337,07	192,71	93,00	83,69	122,10			
	% pérdidas servidor				**% pérdidas clientes**			
	POS	**FIN**	**ACK**	**TRC**	**POS**	**FIN**	**ACK**	**CTR**
100	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
200	0,00	0,00	0,01	0,04	0,00	0,00	0,01	0,00
300	0,00	0,00	0,01	0,07	0,00	0,00	0,01	0,00
400	0,00	0,00	0,04	0,08	0,00	0,00	0,04	0,00
500	0,00	0,00	0,05	1,10	0,00	0,00	0,05	0,00
600	0,00	0,00	0,05	1,62	0,00	0,00	0,05	0,00
700	0,00	0,00	0,06	3,02	0,00	0,00	0,06	0,00
800	0,00	0,00	0,11	7,23	0,00	0,00	0,11	0,00
900	0,00	0,00	0,12	9,20	0,00	0,00	0,12	0,00
1000	0,00	0,00	0,18	43,55	0,00	0,00	0,18	0,00

Tabla I.35: Rendimiento del sistema para el Motorola Milestone con un grupo de trabajo de 25 clientes con la implementación UDP

BIBLIOGRAFÍA

ACE (2013). The adaptive communication environment. ACE: http://www.cs.wustl.edu/s̃chmidt/ACE.html.

Ahonen, T. (2010). *TomiAhonen Phone Book 2010*. TomiAhonen Consulting.

Apple Store (2014). Apple Store: https://itunes.apple.com/es/genre/ios/id36?mt=8.

AR (2013). Augmented Reality: http://en.wikipedia.org/wiki/Augmented reality.

AR-PDA (2009). Ar-pda. ARPDA: http://www.c-lab.de/en/rd_projects/completed_research_projects/2004/ar_pda.

Assad, M., Carmichael, D. J., Cutting, D., and Hudson, A. (2003). Ar phone: Accessible augmented reality in the intelligent environment. In *In OZCHI2003*, pages 26–28.

Azuma, R., Baillot, Y., Behringer, R., Feiner, S., Julier, S., and MacIntyre, B. (2001). Recent advances in augmented reality. *Computer Graphics and Applications, IEEE*, 21(6):34 –47.

Barakonyi, I., Fahmy, T., and Schmalstieg, D. (2004). Remote collaboration using augmented reality videoconferencing. In *Proceedings of Graphics interface 2004*, pages 89–96. Canadian Human-Computer Communications Society.

Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L. (2008). Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359.

Bichlmeier, C., Wimme, F., Heining, S. M., and Navab, N. (2007). Contextual anatomic mimesis hybrid in-situ visualization method for improving multi-sensory depth perception in medical augmented reality. In *Mixed and Aug-*

mented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on, pages 129–138.

Billinghurst, M. and Kato, H. (1999). Real world teleconferencing. In *Proc. of the conference on Human Factors in Computing Systems (CHI 99)*.

Billinghurst, M., Kato, H., and Poupyrev, I. (2001). The magicbook: a transitional ar interface. *Computers & Graphics*, 25(5):745–753.

Billinghurst, M., Poupyrev, I., Kato, H., and May, R. (2000). Mixing realities in shared space: an augmented reality interface for collaborative computing. In *IEEE International Conference on Multimedia and Expo (ICME 2000)*, volume 3, pages 1641–1644.

Billinghurst, M., Weghorst, S., and Furness III, T. (1998). Shared space: An augmented reality approach for computer supported collaborative work. *Virtual Reality*, 3(1):25–36.

Bimber, O., Raskar, R., and Inami, M. (2005). *Spatial augmented reality*. AK Peters Wellesley.

Bruns, E., Brombach, B., Zeidler, T., and Bimber, O. (2007). Enabling mobile phones to support large-scale museum guidance. *MultiMedia, IEEE*, 14(2):16–25.

BuildAR (2009). Buildar. BuildAR: http://www.hitlabnz.org/index.php/products/buildar.

Carmigniani, J. and Furht, B. (2011). *Handbook of Augmented Reality*. Springer New York.

CFS (2014). CFS Scheduler design: https://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt.

Chun, W. H. and Höllerer, T. (2013). Real-time hand interaction for augmented reality on mobile phones. In *Proceedings of the 2013 International Conference on Intelligent User Interfaces*, IUI '13, pages 307–314, New York, NY, USA. ACM.

Cool; Augmented Reality Advertisements (2013). Geekology: http://www.geekologie.com/2008/12/14-week.

Cooper, N., Keatley, A., Dahlquist, M., Mann, S., Slay, H., Zucco, J., Smith, R., and Thomas, B. H. (2004). Augmented reality chinese checkers. In *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 117–126. ACM.

Duato, J., Yalamanchili, S., and Ni, L. M. (1997). *Interconnection Networks: An Engineering Approach*. IEEE Computer Society Press.

Encyclopidia (2013). Encyclopidia: http://encyclopedia.jrank.org/articles/pages/6843/Multimodal-Interfaces.html.

Feiner, S., MacIntyre, B., Höllerer, T., and Webster, A. (1997). A touring machine: Prototyping 3d mobile augmented reality systems for exploring the urban environment. In *First IEEE Int. Symp. On Wearable Computer*, pages 74–81.

Feldman, A., Tapia, E. M., Sadi, S., Maes, P., and Schmandt, C. (2005). Reachmedia: On-the-move interaction with everyday objects. In *Proceedings of the Ninth IEEE International Symposium on Wearable Computers*, ISWC '05, pages 52–59, Washington, DC, USA. IEEE Computer Society.

Fiala, M. (2005). Artag, a fiducial marker system using digital techniques. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 590 – 596 vol. 2.

Fitts, P. M. (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of experimental psychology*, 47(6):381.

Fitzmaurice, G. W. (1993). Situated information spaces and spatially aware palmtop computers. *Commun. ACM*, 36(7):39–49.

Gausemeier, J., Fruend, J., Matysczok, C., Bruederlin, B., and Beier, D. (2003). Development of a real time image based object recognition method for mobile ar-devices. In *Proceedings of the 2nd international conference on Computer graphics, virtual Reality, visualisation and interaction in Africa*, AFRIGRAPH '03, pages 133–139, New York, NY, USA. ACM.

Google (2011). Android. Available at http://www.android.com/index.html.

Google Play (2014). Google Play: https://play.google.com/store.

Grimm, P., Haller, M., Paelke, V., Reinhold, S., Reimann, C., and Zauner, R. (2002). Amire - authoring mixed reality. In *Augmented Reality Toolkit, The First IEEE International Workshop*, pages 2 pp.–.

Hall, S. P. and Anderson, E. (2009). Operating systems for mobile computing. *J. Comput. Small Coll.*, 25:64–71.

Henderson, T. and Bhatti, S. (2003). Networked games: a qos-sensitive application for qos-insensitive users? In *Proceedings of the ACM SIGCOMM workshop on Revisiting IP QoS: What have we learned, why do we care?*, pages 141–147. ACM.

Henrysson, A., Billinghurst, M., and Ollila, M. (2005). Face to face collaborative ar on mobile phones. In *Mixed and Augmented Reality, 2005. Proceedings. Fourth IEEE and ACM International Symposium on*, pages 80 – 89.

Henrysson, A. and Ollila, M. (2004). Umar: Ubiquitous mobile augmented reality. In *Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, MUM '04, pages 41–45, New York, NY, USA. ACM.

Hinterstoisser, S., Lepetit, V., Benhimane, S., Fua, P., and Navab, N. (2011). Learning real-time perspective patch rectification. *International Journal of Computer Vision*, 91(1):107–130.

historyAR (2013). historyAR: https://www.icg.tugraz.at/ daniel/HistoryOfMobileAR/.

Hofmann, R., Seichter, H., and Reitmayr, G. (2012). A gpgpu accelerated descriptor for mobile devices. In *Mixed and Augmented Reality (ISMAR), 2012 IEEE International Symposium on*, pages 289–290. IEEE.

Höllerer, T., Feiner, S., Terauchi, T., Rashid, G., and Hallaway, D. (1999). Exploring mars: Developing indoor and outdoor user interfaces to a mobile augmented reality system. *Computers and Graphics*, 23:779–785.

Huang, Y., Liu, Y., and Wang, Y. (2009). Ar-view: An augmented reality device for digital reconstruction of yuangmingyuan. In *Mixed and Augmented Reality - Arts, Media and Humanities, 2009. ISMAR-AMH 2009. IEEE International Symposium on*, pages 3–7.

Huawei (2014). Huawei Ascend Mate: http://www.huaweidevice.es/smartphones/huawei-mate.

ITIA-CNR (2013). ITIA-CNR: http://www.itia.cnr.it/en/.

Johnson, L. F. and Smith, R. S. (2005). 2005 horizon report.

Jones, M. (2003). Bsd sockets programming from a multi-language perspective. charles river media. *Inc.: Hingham, MA, USA, ISBN*, pages 1–58450.

Julier, S., Feiner, S., and Rosenblum, L. (1999). Augmented reality as an example of a complex and demanding human-centered system. In *First EC/NSF Advanced Research Workshop*, pages 39–46.

Kato, D. H. (2011). Artoolkit. Available at http://www.hitl.washington.edu/artoolkit/.

Kato, H. and Billinghurst, M. (1999). Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *Augmented Reality, 1999. (IWAR '99) Proceedings. 2nd IEEE and ACM International Workshop on*, pages 85–94.

Kato, H., Billinghurst, M., Poupyrev, I., Imamoto, K., and Tachibana, K. (2000). Virtual object manipulation on a table-top ar environment. In *Augmented Reality, 2000.(ISAR 2000). Proceedings. IEEE and ACM International Symposium on*, pages 111–119. Ieee.

Kinect (2013). Kinect: http://es.wikipedia.org/wiki/Kinect.

Kretschmer, U., Coors, V., Spierling, U., Grasbon, D., Schneider, K., Rojas, I., and Malaka, R. (2001). Meeting the spirit of history. In *Proceedings of the 2001 conference on Virtual reality, archeology, and cultural heritage*, pages 141–152. ACM.

Lee, B. and Chun, J. (2010). Interactive manipulation of augmented objects in marker-less ar using vision-based hand interaction. In *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, pages 398–403.

Lee, J.-Y., Lee, S.-H., Park, H.-M., Lee, S.-K., Choi, J.-S., and Kwon, J.-S. (2010a). Design and implementation of a wearable ar annotation system using gaze interaction. In *Consumer Electronics (ICCE), 2010 Digest of Technical Papers International Conference on*, pages 185–186. IEEE.

Lee, S. E., Zhang, Y., Fang, Z., Srinivasan, S., Iyer, R., and Newell, D. (2009). Accelerating mobile augmented reality on a handheld platform. In *Computer Design, 2009. ICCD 2009. IEEE International Conference on*, pages 419–426. IEEE.

Lee, W., Park, Y., Lepetit, V., and Woo, W. (2010b). Point-and-shoot for ubiquitous tagging on mobile phones. In *Mixed and Augmented Reality (ISMAR), 2010 9th IEEE International Symposium on*, pages 57–64.

Loulier, B. (2011). Augmented reality on iphone using artoolkitplus. Available at http://www.benjaminloulier.com/.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110.

MacIntyre, B., Gandy, M., Dow, S., and Bolter, J. D. (2004). Dart: A toolkit for rapid design exploration of augmented reality experiences. In *In ACM Symp. on User Interface Software and Technology (UIST04*, pages 197–206.

Macromedia Director (2014). Macromedia Director: http://es.wikipedia.org/wiki/Adobe_Director.

Malaka, R., Schneider, K., and Kretschmer, U. (2004). Stage-based augmented edutainment. In *Smart Graphics*, pages 54–65. Springer.

Mashable, The Social Media Guide (2013). 10 amazing Augmented Reality iPhone Apps: http://mashable.com/2009/12/05/augmented-reality-iphone/.

Milgram, P. and Kishino, F. (1994). A taxonomy of mixed reality visual displays. *IEICE TRANSACTIONS on Information and Systems*, 77(12):1321–1329.

Mini (2013). Mini: http://www.mini.com.

Mistry, P., Kuroki, T., and Chang, C. (2008). Tapuma: tangible public map for information acquirement through the things we carry. In *Proceedings of the 1st international conference on Ambient media and systems*, page 12. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Mistry, P., Maes, P., and Chang, L. (2009). Wuw-wear ur world: a wearable gestural interface. In *CHI'09 extended abstracts on Human factors in computing systems*, pages 4111–4116. ACM.

Miyashita, T., Meier, P., Tachikawa, T., Orlic, S., Eble, T., Scholz, V., Gapel, A., Gerl, O., Arnaudov, S., and Lieberknecht, S. (2008). An augmented reality museum guide. In *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 103–106. IEEE Computer Society.

Mogilev, D., Kiyokawa, K., Billinghurst, M., and Pair, J. (2002). Ar pad: An interface for face-to-face ar collaboration. In *In CHI 2002 Conference Proceedings*, pages 654–655.

Mohring, M., Lessig, C., and Bimber, O. (2004). Video see-through ar on consumer cell-phones. In *Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 252–253. IEEE Computer Society.

Morillo, P., Orduna, J. M., Fernandez, M., and Duato, J. (2005). Improving the performance of distributed virtual environment systems. *Parallel and Distributed Systems, IEEE Transactions on*, 16(7):637–649.

Newman, J., Ingram, D., and Hopper, A. (2001). Augmented reality in a wide area sentient environment. In *Augmented Reality, 2001. Proceedings. IEEE and ACM International Symposium on*, pages 77–86.

Nilsson, J., Odblom, A. C. E., Fredriksson, J., Zafar, A., and Ahmed, F. (2010). Performance evaluation method for mobile computer vision systems using augmented reality. In *Virtual Reality Conference (VR), 2010 IEEE*, pages 19–22.

Norma 802.11ac (2014). Norma 802.11ac: http://www.ieee802.org/11/Reports/tgac_update.htm.

nyartoolkit (2011). Source code of nyartoolkit. Available at http://sourceforge.jp/projects/nyartoolkit-and/.

Ozuysal, M., Fua, P., and Lepetit, V. (2007). Fast keypoint recognition in ten lines of code. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8.

Papagiannakis, G., Singh, G., and Magnenat-Thalmann, N. (2008). A survey of mobile and wireless technologies for augmented reality systems. *Computer Animation and Virtual Worlds*, 19(1):3–22.

Paredes Figueras, Josep; Simonetti Ibañez, A. (2013). Vuforia v1.5 sdk: Analysis and evaluation of capabilities.

Piekarski, W. and Thomas, B. H. (2001). Tinmith-evo5 a software architecture for supporting research into outdoor augmented reality environments. In *Proceedings of the IEEE and ACM International Symposium on Augmented Reality*, pages 177–178.

Piekarski, W. and Thomas, B. H. (2002). Tinmith-hand: Unified user interface technology for mobile outdoor augmented reality and indoor virtual reality.

Planificación CPU (2014). Planificación CPU: https://access.redhat.com/site/documentation/es-ES/Red_Hat_Enterprise_Linux/6/html/Performance_Tuning_Guide/s-cpu-scheduler.html.

Quake (2005). Quake. Quake: http://www.quakelive.com.

Qualcomm (2012). Vuforia sdk 1.5. Available at http://www.qualcomm.com/solutions/augmented-reality.

Qualcomm Vuforia (2011). Qualcomm. Qualcomm: http://www.qualcomm.com/solutions/augmented-reality.

Ranking GPUs (2014). Ranking GPUs: http://s-smartphone.com/cell-phones/other/ranking-the-best-gpu-for-smartphonestablet/.

Regenbrecht, H. T. and Specht, R. (2000). A mobile passive augmented reality device - mpard. In *Augmented Reality, 2000. (ISAR 2000). Proceedings. IEEE and ACM International Symposium on*, pages 81–84.

Reitmayr, G. and Schmalstieg, D. (2001a). Mobile collaborative augmented reality. In *Augmented Reality, 2001. Proceedings. IEEE and ACM International Symposium on*, pages 114 –123.

Reitmayr, G. and Schmalstieg, D. (2001b). Opentracker-an open software architecture for reconfigurable tracking based on xml. *vr*, 1:285.

Reitmayr, G. and Schmalstieg, D. (2003). Location based applications for mobile augmented reality. In *Proceedings of the Fourth Australasian user interface conference on User interfaces 2003-Volume 18*, pages 65–73. Australian Computer Society, Inc.

Rekimoto, J. (1996). Transvision: A hand-held augmented reality system for collaborative design. In *Virtual Systems and Multi-Media (VSMM)'96*.

Sandor, C., Olwal, A., Bell, B., and Feiner, S. (2005). Immersive mixed-reality configuration of hybrid user interfaces. In *Mixed and Augmented Reality, 2005. Proceedings. Fourth IEEE and ACM International Symposium on*, pages 110–113. IEEE.

Schmalstieg, D., Fuhrmann, A., and Hesina, G. (2000). Bridging multiple user interface dimensions with augmented reality. In *In Proc. ISAR 2000*, pages 20–29.

Schmalstieg, D., Fuhrmann, A., Hesina, G., Szalavári, Z., Encarnaçao, L. M., Gervautz, M., and Purgathofer, W. (2002). The studierstube augmented reality project. *Presence: Teleoperators and Virtual Environments*, 11(1):33–54.

Schmalstieg, D. and Wagner, D. (2007). Experiences with handheld augmented reality. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 3 –18.

Schmeil, A. and Broll, W. (2007). An anthropomorphic ar-based personal information manager and guide. In *Proceedings of the 4th international conference on Universal access in human-computer interaction: ambient interaction*, UAHCI'07, pages 699–708, Berlin, Heidelberg. Springer-Verlag.

Sensorama (2013). Sensorama: http://es.wikipedia.org/wiki/Sensorama.

Sinem, G. and Feiner, S. (2003). Authoring 3d hypermedia for wearable augmented and virtual reality. In *Wearable Computers, 2003. Proceedings. Seventh IEEE International Symposium on*, pages 118–126.

SkyTex (2014). SkyTex SKYPAD SP722: http://www.skytex.com/es/skypad/sp722.

Sodhi, R. S., Jones, B. R., Forsyth, D., Bailey, B. P., and Maciocci, G. (2013). Bethere: 3d mobile collaboration with spatial input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 179–188. ACM.

Srinivasan, S., Fang, Z., Iyer, R., Zhang, S., Espig, M., Newell, D., Cermak, D., Wu, Y., Kozintsev, I., and Haussecker, H. (2009). Performance characterization and optimization of mobile augmented reality on handheld platforms. In

Workload Characterization. IISWC 2009. IEEE International Symposium on, pages 128 –137.

Strauss, P. S. and Carey, R. (1992). An object-oriented 3d graphics toolkit. In *ACM SIGGRAPH Computer Graphics*, volume 26, pages 341–349. ACM.

Szalavári, Z., Schmalstieg, D., Fuhrmann, A., and Gervautz, M. (1998). Studierstube: An environment for collaboration in augmented reality. *Virtual Reality*, 3:37–48.

Thomas, B., Close, B., Donoghue, J., Squires, J., De Bondi, P., Morris, M., and Piekarski, W. (2000). Arquake: An outdoor/indoor augmented reality first person application. In *In 4th Int'l Symposium on Wearable Computers*, pages 139–146.

Unity (2014). Unity3D: http://unity3d.com.

Wagner, D., Langlotz, T., and Schmalstieg, D. (2008a). Robust and unobtrusive marker tracking on mobile phones. In *Mixed and Augmented Reality, 2008. ISMAR 2008. 7th IEEE/ACM International Symposium on*, pages 121–124.

Wagner, D., Pintaric, T., Ledermann, F., and Schmalstieg, D. (2005). Towards massively multi-user augmented reality on handheld devices. In Gellersen, H.-W., Want, R., and Schmidt, A., editors, *Pervasive Computing*, volume 3468 of *Lecture Notes in Computer Science*, pages 208–219. Springer Berlin Heidelberg.

Wagner, D., Reitmayr, G., Mulloni, A., Drummond, T., and Schmalstieg, D. (2008b). Pose tracking from natural features on mobile phones. In *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, ISMAR '08, pages 125–134, Washington, DC, USA. IEEE Computer Society.

Wagner, D., Reitmayr, G., Mulloni, A., Drummond, T., and Schmalstieg, D. (2010). Real-time detection and tracking for augmented reality on mobile phones. *Visualization and Computer Graphics, IEEE Transactions on*, 16(3):355 – 368.

Wagner, D. and Schmalstieg, D. (2003). First steps towards handheld augmented reality. In *Proceedings of the 7th IEEE International Symposium on Wea-*

rable Computers, ISWC '03, pages 127–135, Washington, DC, USA. IEEE Computer Society.

Wagner, D. and Schmalstieg, D. (2006). Handheld augmented reality displays. In *Virtual Reality Conference, 2006*, pages 321–321. IEEE.

Wagner, D. and Schmalstieg, D. (2007). Artoolkitplus for pose tracking on mobile devices. In *Proceedings of 12th Computer Vision Winter Workshop (CVWW'07)*, pages 139–146.

Wagner, D., Schmalstieg, D., and Bischof, H. (2009). Multiple target detection and tracking with guaranteed framerates on mobile phones. In *Mixed and Augmented Reality. ISMAR 2009. 8th IEEE International Symposium on*, pages 57 –64.

Wang, Y., Langlotz, T., Billinghurst, M., and Bell, T. (2009). An authoring tool for mobile phone ar environments. In *Proceedings of New Zealand Computer Science Research Student Conference*, volume 9, pages 1–4.

Wavelet Haar (2013). wavelet: http://es.wikipedia.org/wiki/Wavelet_de_Haar.

White, S., Lister, L., and Feiner, S. (2007). Visual hints for tangible gestures in augmented reality. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 47–50. IEEE.

Wiggins, M. (2013). Report: Mobile augmented reality. smartphones, tablets and smart glasses 2013-2018. page 150.

Wikitude (2013). WikitudeDrive: http://www.wikitude.com/app/.

Yi-bo, L., Shao-peng, K., Zhi-hua, Q., and Qiong, Z. (2008). Development actuality and application of registration technology in augmented reality. In *Computational Intelligence and Design, 2008. ISCID '08. International Symposium on*, volume 2, pages 69–74.

Yohan, S. J., Julier, S., Baillot, Y., Lanzagorta, M., Brown, D., and Rosenblum, L. (2000). Bars: Battlefield augmented reality system. In *In NATO Symposium on Information Processing Techniques for Military Systems*, pages 9–11.

Zhou, F., Duh, H. B.-L., and Billinghurst, M. (2008). Trends in augmented reality tracking, interaction and display: A review of ten years of ismar. In *Mixed and*

Augmented Reality, 2008. ISMAR 2008. 7th IEEE/ACM International Symposium on, pages 193–202.